개발자 가이드

Amazon Lex V1

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Lex V1: 개발자 가이드

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon의 상표 및 트레이드 드레스는 Amazon 외 제품 또는 서비스와 함께, Amazon 브랜드 이미지를 떨어뜨리거나 고객에게 혼동을 일으킬 수 있는 방식으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계와 관계없이 해당 소유자의 자산입니다.

Table of Contents

..... viii
Amazon Lex란 무엇입니까? 1
Amazon Lex를 처음 사용하십니까? 2
작동 방식 3
지원되는 언어 5
지원되는 언어 및 로캘 5
Amazon Lex 기능에서 지원하는 언어 및 로캘 6
프로그래밍 모델 7
모델 구축 API 작업 7
런타임 API 작업 8
코드 후크로서의 Lambda 함수 9
메시지 관리 12
메시지 유형 13
메시지 구성을 위한 컨텍스트 14
지원되는 메시지 형식 18
메시지 그룹 19
응답 카드 20
대화 컨텍스트 관리 25
의도 컨텍스트 설정 26
기본 슬롯 값 사용 28
Setting Session Attributes 29
Setting Request Attributes 31
세션 시간 제한 설정 34
의도 간 정보 공유 34
복합 속성 설정 35
신뢰도 점수 사용 36
세션 관리 38
대화 로그 39
대화 로그에 대한 IAM 정책 40
대화 로그 구성 43
대화 로그 암호화 47
Amazon CloudWatch 로그에서 텍스트 로그 보기 48
Amazon S3에서 오디오 로그에 액세스 52
CloudWatch 지표를 통해 대화 로그 상태 모니터링 52

세션 관리 ... 53
의도 전환 ... 55
이전 의도 다시 시작 ... 55
새 세션 시작 ... 56
슬롯 값 유효성 검사 ... 56
배포 옵션 ... 57
기본 제공 의도 및 슬롯 유형 ... 57
기본 제공 의도 ... 57
기본 제공 슬롯 유형 ... 74
사용자 지정 슬롯 유형 ... 84
슬롯 난독화 ... 86
감정 분석 ... 87
리소스 태그 지정 ... 88
리소스에 태그 지정 ... 89
태그 제한 ... 89
리소스에 태그 지정(콘솔) ... 90
리소스에 태그 지정(AWS CLI) ... 92
시작하기 ... 94
1단계: 계정 설정 ... 94
에 가입 AWS ... 94
사용자 생성 ... 95
다음 단계 ... 96
2단계: 설정 AWS CLI ... 96
... 97
3단계: 시작하기(콘솔) ... 97
연습 1: 청사진을 사용하여 봇 생성 ... 97
연습 2: 사용자 지정 봇 생성 ... 134
연습 3: 버전 게시 및 별칭 만들기 ... 150
4단계: 시작하기(AWS CLI) ... 151
연습 1: 봇 생성 ... 152
연습 2: 새 표현 추가 ... 169
연습 3: Lambda 함수 추가 ... 175
연습 4: 버전 게시 ... 179
연습 5: 별칭 만들기 ... 185
연습 6: 정리 ... 186
버전 관리 및 별칭 ... 188

버전 관리 ... 188
$LATEST 버전 ... 188
Amazon Lex 리소스 버전 게시 ... 189
Amazon Lex 리소스 업데이트 ... 190
Amazon Lex 리소스 또는 버전 삭제 ... 190
에일리어스 ... 190
Lambda 함수 사용 ... 193
Lambda 함수 입력 이벤트 및 응답 형식 ... 193
입력 이벤트 형식 ... 193
응답 형식 ... 200
Amazon Lex와 AWS Lambda 청사진 ... 207
특정 로캘에 대한 청사진 업데이트 ... 208
봇 배포 ... 209
메시징 플랫폼에 Amazon Lex 봇 배포하기 ... 209
Facebook과 통합 ... 212
Kik와의 통합 ... 215
슬랙에 통합 ... 219
Twilio SMS와 통합 ... 225
모바일 애플리케이션에서 Amazon Lex 봇 배포하기 ... 228
가져오기 및 내보내기 ... 229
Amazon Lex 형식으로 내보내기 및 가져오기 ... 229
Amazon Lex 형식으로 내보내기 ... 230
Amazon Lex 형식으로 가져오기 ... 231
내보내기 및 가져오기를 위한 JSON 형식 ... 232
Alexa Skill로 내보내기 ... 236
봇 예제 ... 238
스케줄 예약 ... 238
봇 청사진(ScheduleAppointment) 개요 ... 240
Lambda 함수 청사진(lex-make-appointment-python) 개요 ... 241
1단계: Amazon Lex 봇 생성 ... 242
2단계: Lambda 함수 생성 ... 245
3단계: 의도 업데이트: 코드 후크 구성 ... 246
4단계: Facebook Messenger 플랫폼에 봇 배포 ... 247
정보 흐름의 세부 정보 ... 248
여행 예약 ... 265
1단계: 청사진 검토 ... 266

2단계: Amazon Lex 봇 생성 ... 269
3단계: Lambda 함수 생성 ... 272
4단계: Lambda 함수를 코드 후크로 추가 ... 273
정보 흐름의 세부 정보 ... 277
예: 응답 카드 사용 ... 297
표현 업데이트 ... 301
웹 사이트와의 통합 ... 303
콜센터 에이전트 어시스턴트 ... 303
1단계: Amazon Kendra 인덱스를 생성합니다. ... 305
2단계: Amazon Lex 봇 생성 ... 305
3단계: 사용자 지정 및 기본 제공 의도에 추가 ... 306
4단계: Amazon Cognito 설정 ... 307
5단계: 봇을 웹 애플리케이션으로 배포 ... 309
6단계: 봇 사용 ... 309
봇 마이그레이션 ... 312
봇 마이그레이션(콘솔) ... 312
Lambda 함수 마이그레이션 ... 313
마이그레이션 메시지 ... 314
기본 제공 의도 ... 314
기본 제공 슬롯 유형 ... 314
대화 로그 ... 314
메시지 그룹 ... 315
프롬프트 및 프레이즈 ... 315
기타 Amazon Lex V1 기능 ... 315
Lambda 함수 마이그레이션 ... 316
업데이트된 필드 목록 ... 317
보안 ... 326
데이터 보호 ... 326
유휴 데이터 암호화 ... 327
전송 중 데이터 암호화 ... 328
키 관리 ... 328
ID 및 액세스 관리 ... 328
대상 ... 329
ID를 통한 인증 ... 330
정책을 사용하여 액세스 관리 ... 333
Amazon Lex에서 IAM을 사용하는 방법 ... 335

자격 증명 기반 정책 예제 345
Amazon Lex에 대한 AWS 관리형 정책 351
서비스 연결 역할 사용 360
문제 해결 362
모니터링 364
CloudWatch를 사용한 Amazon Lex 모니터링 364
를 사용하여 Amazon Lex API 호출 로깅 AWS CloudTrail 375
규정 준수 검증 380
복원성 381
인프라 보안 381
가이드라인 및 할당량 382
지원되는 리전 382
일반 가이드라인 382
할당량 385
런타임 서비스 할당량 386
모델 구축 할당량 387
API 참조 392
작업 392
Amazon Lex 모델 구축 서비스 394
Amazon Lex 런타임 서비스 605
데이터 타입 648
Amazon Lex 모델 구축 서비스 649
Amazon Lex 런타임 서비스 706
문서 기록 725
AWS 용어집 732

지원 종료 공지: 2025년 9월 15 AWS 일에는 Amazon Lex V1에 대한 지원을 중단할 예정입니다. 2025년 9월 15일 이후에는 Amazon Lex V1 콘솔 또는 Amazon Lex V1 리소스에 더 이상 액세스할 수 없습니다. Amazon Lex V2를 사용하는 경우 대신 Amazon Lex V2 가이드를 참조하세요.

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.

Amazon Lex란 무엇입니까?

Amazon Lex는 음성 및 텍스트를 사용하는 애플리케이션에 대화형 인터페이스를 구축하기 위한 AWS 서비스입니다. Amazon Lex를 사용하면 이제 모든 개발자가 Amazon Alexa를 지원하는 동일한 대화 엔진을 사용할 수 있으므로 새로운 애플리케이션과 기존 애플리케이션에 정교한 자연어 챗봇을 구축할 수 있습니다. Amazon Lex는 자연어 이해(NLU) 및 자동 음성 인식(ASR)의 심층적인 기능과 유연성을 제공하므로 실제와 같은 대화형 상호 작용을 통해 매력적인 사용자 경험을 구축하고 새로운 제품 카테고리를 만들 수 있습니다.

Amazon Lex를 사용하면 어떤 개발자도 대화형 챗봇을 신속하게 구축할 수 있습니다. Amazon Lex를 사용하면 딥 러닝 전문 지식이 필요하지 않습니다. 봇을 생성하려면 Amazon Lex 콘솔에서 기본 대화 흐름을 지정하기만 하면 됩니다. Amazon Lex 는 대화를 관리하며 대화 중에 응답을 동적으로 조정합니다. 콘솔을 사용하여 텍스트 또는 음성 챗봇을 구축, 테스트 및 게시할 수 있습니다. 그런 다음 모바일 장치, 웹 애플리케이션 및 채팅 플랫폼(예: Facebook Messenger)에서 봇에 대화형 인터페이스를 추가할 수 있습니다.

Amazon Lex는 와의 사전 구축된 통합을 제공하며 Amazon Cognito AWS Lambda,, Amazon AWS Mobile Hub Amazon CloudWatch 및 Amazon DynamoDB를 포함한 AWS 플랫폼의 다른 많은 서비스와 쉽게 통합할 수 있습니다. Amazon Cognito Lambda와의 통합을 통해 봇은 사전 구축된 서버리스 엔터프라이즈 커넥터에 액세스하여 Salesforce, HubSpot 또는 Marketo와 같은 SaaS 애플리케이션의 데이터에 연결할 수 있습니다.

Amazon Lex를 사용하면 다음과 같은 이점이 있습니다.

- 단순성 - Amazon Lex는 사용자가 콘솔을 사용하여 몇 분 만에 챗봇을 만들 수 있도록 안내합니다. 예시 구절을 몇 개만 제공하면 Amazon Lex가 완성된 자연어 모델을 구성하고, 이를 통해 봇은 음성과 텍스트를 사용하여 질문하고 응답을 받고 정교한 작업을 완수합니다.

- 대중화된 딥 러닝 기술 – Alexa와 동일한 기술로 구동되는 Amazon Lex는 ASR 및 NLU 기술을 제공하여 음성 언어 이해(SLU) 시스템을 생성합니다. Amazon Lex는 SLU를 통해 자연어 음성 및 텍스트 입력을 받아들이고, 입력 이면의 의도를 이해하며, 적절한 비즈니스 함수를 간접 호출하여 사용자 의도를 이행합니다.

  음성 인식 및 자연어 이해는 컴퓨터 공학에서 해결해야 할 가장 까다로운 문제들 중 일부로서, 이 문제를 해결하려면 정교한 딥 러닝 알고리즘을 막대한 양의 데이터 및 인프라에서 훈련되어야 합니다.

Amazon Lex는 Alexa와 동일한 기술을 기반으로 모든 개발자가 이용할 수 있는 딥 러닝 기술을 제공합니다. Amazon Lex 챗봇은 사용자에게서 받은 음성을 텍스트로 변환하여 사용자 의도를 이해함으로써 지능형 응답을 생성합니다. 따라서 고객에게 차별화된 부가 가치를 제공하는 봇을 구축하는 데 집중할 수 있어 대화형 인터페이스를 통해 가능한 완전히 새로운 범주의 제품을 정의할 수 있습니다.

- 원활한 배포 및 규모 조정 – Amazon Lex를 사용하면 Amazon Lex 콘솔에서 직접 챗봇을 구축, 테스트 및 배포할 수 있습니다. Amazon Lex를 사용하면 모바일 장치, 웹 앱 및 채팅 서비스(예: Facebook Messenger)에서 사용할 수 있는 음성 또는 텍스트 챗봇을 쉽게 게시할 수 있습니다. Amazon Lex는 규모를 자동으로 조정할 수 있으므로 봇 체험에 동력을 제공하기 위해 하드웨어 프로비저닝 및 인프라 관리에 신경을 쓸 필요가 없습니다.

- AWS 플랫폼과의 기본 통합 - Amazon Lex는 Amazon Cognito, AWS Lambda, Amazon CloudWatch 및 같은 다른 AWS 서비스와 기본적으로 상호 운용 가능합니다 AWS Mobile Hub. 보안, 모니터링, 사용자 인증, 비즈니스 로직, 스토리지 및 모바일 앱 개발을 위해 AWS 플랫폼의 강력한 기능을 활용할 수 있습니다.

- 비용 효율성 - Amazon Lex를 사용하면 선결제 비용이나 최소 요금이 없습니다. 텍스트나 음성을 통해 요청한 것에 대해서만 청구됩니다. 사용량에 따른 요금 및 요청당 저비용 방식이므로 서비스를 대화형 인터페이스를 구축할 수 있는 비용 효율적인 방법으로 활용할 수 있습니다. Amazon Lex 프리티어를 사용하면 초기 투자 비용을 지불하지 않고 쉽게 Amazon Lex를 사용할 수 있습니다.

Amazon Lex를 처음 사용하십니까?

Amazon Lex를 처음 사용한다면, 다음 섹션을 순서대로 읽어보기를 권장합니다.

1. Amazon Lex 시작하기 – 이 섹션에서는 계정을 설정하고 Amazon Lex를 테스트합니다.
2. API 참조 - 이 섹션에서는 Amazon Lex에 대해 살펴볼 수 있는 추가 예시를 제공합니다.

Amazon Lex: 작동 방식

Amazon Lex 를 통해 Amazon Alexa 와 동일한 기술로 구동되는 음성 또는 텍스트 인터페이스를 사용하여 애플리케이션을 구축할 수 있습니다. 다음은 Amazon Lex 사용할 때 수행하는 일반적인 몇 가지 단계입니다.

1. 봇을 생성한 후 지원하고 싶은 하나 이상의 의도로 이를 구성합니다. 사용자의 목표(의도)를 이해하고 사용자와의 대화에 참여하여 정보를 유도하며 사용자의 의도를 이행하도록 봇을 구성합니다.
2. 봇 테스트. Amazon Lex 콘솔에서 제공하는 테스트 창 클라이언트를 사용할 수 있습니다.
3. 버전을 게시하고 별칭을 만듭니다.
4. 봇을 배포합니다. 모바일 애플리케이션 등의 플랫폼이나 Facebook Messenger 등의 메시징 플랫폼에 봇을 배포할 수 있습니다.

시작하기 전에 다음과 같은 Amazon Lex 의 핵심 개념 및 용어를 익힙니다.

- 봇 - 봇은 피자 주문, 호텔 예약, 꽃 주문 등과 같은 자동화 작업을 수행합니다. Amazon Lex 봇은 자동 음성 인식(ASR)및 자연어 이해(NLU) 기능으로 구동됩니다. 각 데이터 스트림은 계정 내에서 고유한 이름을 가져야 합니다.

  Amazon Lex 봇은 텍스트나 음성으로 제공된 사용자 입력을 이해하고 자연 언어로 대화할 수 있습니다. Lambda 함수 를 만들고 이를 사용자의 의도 구성에 코드 후크로 추가하여, 사용자의 데이터 검증 및 이행 작업을 수행할 수 있습니다.

- 의도 – 의도는 사용자가 수행하고자 하는 작업을 나타냅니다. 하나 이상의 관련 의도를 지원하도록 봇을 생성합니다. 예를 들어, 피자 및 음료를 주문하는 봇을 만들 수 있습니다. 각 의도에 대해 다음 필수 정보를 제공합니다.

  - 의도 이름– 의도를 설명하는 이름입니다. 예: **`OrderPizza`**. 의도 이름은 계정 내에서 고유해야 합니다.
  - 샘플 표현 – 사용자가 의도를 전달하는 방식입니다. 예를 들어, 사용자는 "피자 주문할 수 있나요" 또는 "피자 주문하고 싶어요"라고 말할 수 있습니다.

- 의도 이행 방법 - 사용자가 필수 정보를 제공한 후 의도를 이행하는 방법입니다(예: 동네 피자 가게에서 주문). Lambda 함수 를 생성하여 의도를 이행하는 것을 권장합니다.

  Amazon Lex 에서 간단히 클라이언트 애플리케이션에 정보를 반환하여 필요한 이행을 수행하도록 의도를 선택적으로 구성할 수 있습니다.

Amazon Lex 는 피자 주문과 같은 사용자 지정 의도 외에도 봇을 빠르게 설정할 수 있도록 내장 의도도 제공합니다. 자세한 내용은 기본 제공 의도 및 슬롯 유형을 참조하세요.

- 슬롯 – 의도에는 0개 이상의 슬롯 또는 파라미터가 필요할 수 있습니다. 의도 구성의 일부로 슬롯을 추가합니다. 런타임 시에는 Amazon Lex 는 사용자에게 특정 슬롯 값을 묻습니다. 사용자가 모든 필수 슬롯의 값을 제공해야 Amazon Lex 가 의도를 이행할 수 있습니다.

  예를 들어, `OrderPizza` 의도에는 피자 크기, 크러스트 유형 및 피자 개수와 같은 슬롯이 필요합니다. 의도 구성에 이러한 슬롯을 추가합니다. 각 슬롯에 대해 슬롯 유형 및 Amazon Lex 가 사용자로부터 데이터를 유도하도록 클라이언트에 보내는 프롬프트를 제공합니다. 사용자는 "라지로 부탁해요" 또는 "스몰 사이즈가 좋아요."와 같은 추가 단어가 포함된 슬롯 값으로 응답할 수 있습니다. Amazon Lex 는 여전히 의도한 슬롯 값을 이해할 수 있습니다.

- 슬롯 유형 – 각 슬롯에는 유형이 있습니다. 사용자 지정 슬롯 유형을 생성하거나 내장 슬롯 유형을 사용할 수 있습니다. 각 슬롯 유형은 계정 내에서 고유한 이름을 가져야 합니다. 예를 들어, `OrderPizza` 의도에 대한 다음과 같은 슬롯 유형을 만들고 사용할 수 있습니다.

  - 사이즈 – 열거 값은 `Small`, `Medium` 및 `Large`입니다.
  - 크러스트 – 열거 값은 `Thick` 및 `Thin`입니다.

Amazon Lex 는 내장 슬롯 유형도 제공합니다. 예를 들어 `AMAZON.NUMBER` 는 주문한 피자의 개수로 사용할 수 있는 내장 슬롯 유형입니다. 자세한 내용은 기본 제공 의도 및 슬롯 유형을 참조하세요.

현재 Amazon Lex 사용 가능한 모든 AWS 리전 목록은 Amazon Web Services 일반 참조의 AWS 리전 및 엔드포인트 를 참조하세요.

다음 주제에서는 추가 정보를 제공합니다. 이를 순서대로 검토한 다음 Amazon Lex 시작하기 연습을 수행하는 것이 좋습니다.

주제

- Amazon Lex에서 지원되는 언어
- 프로그래밍 모델
- 메시지 관리
- 대화 컨텍스트 관리
- 신뢰도 점수 사용
- 대화 로그
- Amazon Lex API로 세션 관리
- 봇 배포 옵션
- 기본 제공 의도 및 슬롯 유형
- 사용자 지정 슬롯 유형
- 슬롯 난독화
- 감정 분석
- Amazon Lex 리소스 태그 지정

Amazon Lex에서 지원되는 언어

Amazon Lex V1은 다양한 언어와 로캘을 지원합니다. 지원되는 언어와 이를 지원하는 기능은 다음 표에 나열되어 있습니다.

Amazon Lex V2는 추가 언어를 지원합니다. Amazon Lex V2에서 지원되는 언어를 참조하십시오.

지원되는 언어 및 로캘

Amazon Lex V1은 다음 언어 및 로캘을 지원합니다.

코드	언어 및 로캘
de-DE	독일어(독일)
en-AU	영어(호주)
en-GB	영어(영국)
en-IN	영어(인도)
en-US	영어(미국)
es-419	스페인어(라틴 아메리카)
es-ES	스페인어(스페인)
es-US	스페인어(미국)
fr-CA	프랑스어(캐나다)
fr-FR	프랑스어(프랑스)
it-IT	이탈리아어(이탈리아)
ja-JP	일본어(일본)
ko-KR	한국어(한국)

Amazon Lex 기능에서 지원하는 언어 및 로캘

모든 Amazon Lex 기능은 이 표에 나열된 경우를 제외하고 모든 언어 및 로캘에서 지원됩니다.

Feature	지원되는 언어 및 로캘
의도 컨텍스트 설정	영어(미국)(en-US)

프로그래밍 모델

봇은 Amazon Lex 의 기본 리소스 유형입니다. Amazon Lex 의 기타 리소스 유형은 의도, 슬롯 유형, 별칭 및 봇 채널 연결입니다.

Amazon Lex 콘솔 또는 모델 구축 API를 사용하여 봇을 생성합니다. 콘솔은 애플리케이션을 위해 프로덕션용으로 사용할 준비가 된 봇을 구축할 때 사용하는 그래픽 사용자 인터페이스를 제공합니다. 원하는 경우 AWS CLI 또는 자체 사용자 지정 프로그램을 통해 모델 구축 API를 사용하여 봇을 생성할 수 있습니다.

봇을 생성한 후 지원되는 플랫폼 중 하나에 이를 배포하거나 자체 애플리케이션에 통합할 수 있습니다. 사용자가 봇과 상호 작용하는 경우 클라이언트 애플리케이션은 Amazon Lex 런타임 API를 사용하여 봇에 요청을 보냅니다. 예를 들어, 사용자가 "피자 주문하고 싶어요"라고 말하면, 클라이언트는 런타임 API 작업 중 하나를 사용하여 Amazon Lex 에 이 입력을 보냅니다. 사용자는 입력을 음성이나 텍스트로 제공할 수 있습니다.

또한 Lambda 함수 를 생성하여 의도에 사용할 수 있습니다. 이러한 Lambda 함수 코드 후크를 사용하여 초기화, 사용자 입력 검증, 의도 이행 등과 같은 런타임 활동을 수행합니다. 다음 섹션에서 추가 정보를 제공합니다.

주제

- 모델 구축 API 작업
- 런타임 API 작업
- 코드 후크로서의 Lambda 함수

모델 구축 API 작업

봇, 의도 및 슬롯 유형을 프로그래밍 방식으로 생성하려면 모델 구축 API 작업을 사용합니다. 또한 모델 구축 API를 사용하여 봇에 대한 리소스를 관리, 업데이트 및 삭제할 수 있습니다. 모델 구축 API 작업은 다음과 같습니다.

- PutBot, PutBotAlias, PutIntent 및 PutSlotType - 각각 봇, 봇 별칭, 의도 및 슬롯 유형을 생성 및 업데이트합니다.
- CreateBotVersion, CreateIntentVersion 및 CreateSlotTypeVersion- 각각 봇, 의도 및 슬롯 유형의 버전을 생성 및 게시합니다.
- GetBot 및 GetBots - 각각 생성한 특정 봇 또는 봇 목록을 가져옵니다.
- GetIntent 및 GetIntents- 각각 생성한 특정 의도 또는 의도 목록을 가져옵니다.

- GetSlotType 및 GetSlotTypes- 각각 생성한 특정 슬롯 유형 또는 슬롯 유형 목록을 가져옵니다.
- GetBuiltinIntent, GetBuiltinIntents 및 GetBuiltinSlotTypes - 각각 봇에서 사용할 수 있는 Amazon Lex 내장 의도, Amazon Lex 내장 의도 목록 또는 내장 슬롯 유형 목록을 가져옵니다.
- GetBotChannelAssociation 및 GetBotChannelAssociations - 각각 봇과 메시징 플랫폼 간의 연결 또는 이러한 연결 목록을 가져옵니다.
- DeleteBot, DeleteBotAlias, DeleteBotChannelAssociation, DeleteIntent 및 DeleteSlotType- 계정에서 불필요한 리소스를 제거합니다.

모델 구축 API를 사용하면 Amazon Lex 리소스를 관리하기 위한 사용자 지정 도구를 생성할 수 있습니다. 예를 들어 봇, 의도 및 슬롯 유형에 대해 각각 100개의 버전 한도가 있습니다. 모델 구축 API를 사용하여 봇이 이 한도에 근접하면 오래된 버전을 자동으로 삭제하는 도구를 구축할 수 있습니다.

한 번에 하나의 작업만 리소스를 업데이트하도록 하기 위해 Amazon Lex는 체크섬을 사용합니다. `Put` API 작업 (PutBot, PutBotAlias, PutIntent 또는 PutSlotType)을 사용하여 리소스를 업데이트하는 경우 요청에서 리소스의 현재 체크섬을 전달해야 합니다. 도구 두 개가 동시에 리소스를 업데이트하려고 하면 둘 다 동일한 현재 체크섬을 제공합니다. Amazon Lex 에 도달한 첫 번째 요청이 리소스의 현재 체크섬과 일치합니다. 두 번째 요청이 도착할 때, 체크섬은 다릅니다. 두 번째 도구는 `PreconditionFailedException` 예외를 수신하고 업데이트가 종료됩니다.

`Get` 작업 (GetBot, GetIntent, 및 GetSlotType)은 결국에는 일관성을 유지하게 됩니다. `Put` 작업 중 하나를 사용하여 리소스를 생성 또는 수정한 직후에 `Get` 작업을 사용하면 변경 사항이 반환되지 않을 수 있습니다. `Get` 작업이 최신 업데이트를 반환하면 리소스가 다시 수정될 때까지 항상 업데이트된 리소스를 반환합니다. 체크섬을 확인해 업데이트된 리소스가 반환되었는지 확인할 수 있습니다.

런타임 API 작업

클라이언트 애플리케이션은 다음과 같은 런타임 API 작업을 사용하여 Amazon Lex와 통신합니다.

- PostContent – 음성 또는 텍스트 입력을 받아들여 의도 정보와 사용자에게 전달할 텍스트 또는 음성 메시지를 반환합니다. Amazon Lex 는 현재 다음 오디오 형식을 지원합니다.

  입력 오디오 형식 - LPCM 및 Opus

  출력 오디오 형식 - MPEG, OGG, PCM

`PostContent` 작업은 8kHz 및 16kHz에서 오디오 입력을 지원합니다. 최종 사용자가 전화를 통해 Amazon Lex 와 대화하는 애플리케이션(예: 자동화된 콜 센터)은 8kHz 오디오를 직접 전달할 수 있습니다.

- PostText – 텍스트 입력을 받아들여 의도 정보와 사용자에게 전달할 텍스트 메시지를 반환합니다.

클라이언트 애플리케이션은 런타임 API를 사용하여 특정 Amazon Lex 봇을 직접 호출하여 표현(사용자 텍스트 또는 음성 입력)을 처리합니다. 예를 들어, 사용자가 "피자가 필요해"라고 말한다고 가정하겠습니다. 클라이언트는 Amazon Lex 런타임 API 작업 중 하나를 사용하여 봇에 이 사용자 입력을 보냅니다. 사용자 입력으로부터, Amazon Lex 는 사용자 요청이 봇에서 정의된 `OrderPizza` 의도에 대한 것임을 인식합니다. Amazon Lex 는 사용자를 대화에 참여시켜 피자 크기, 토핑, 피자 수와 같은 필수 정보 또는 슬롯 데이터를 수집합니다. 사용자가 필수 슬롯 데이터를 모두 제공하면 Amazon Lex는 Lambda 함수 코드 후크를 간접 호출하여 의도를 이행하거나 의도 구성 방법에 따라 클라이언트에 의도 데이터를 반환합니다.

봇에서 음성 입력을 사용하는 경우 PostContent 작업을 사용합니다. 예를 들어, 자동화된 콜 센터 애플리케이션은 상담원 대신에 Amazon Lex 봇에 음성을 보내서 고객 문의를 처리할 수 있습니다. 8kHz 오디오 형식을 사용하여 전화에서 Amazon Lex 로 직접 오디오를 보낼 수 있습니다.

Amazon Lex 콘솔의 테스트 창에서는 PostContent API를 사용하여 텍스트 및 음성 요청을 Amazon Lex 에 보냅니다. Amazon Lex 시작하기 연습에서 이 테스트 창을 사용합니다.

코드 후크로서의 Lambda 함수

Lambda 함수를 코드 후크로 간접 호출하도록 Amazon Lex 봇을 구성할 수 있습니다. 코드 후크는 다음과 같이 여러 가지 용도로 사용할 수 있습니다.

- 사용자 상호 작용을 사용자 지정합니다. 예를 들어 Joe가 사용 가능한 피자 토핑을 요청하면 Joe의 선택에 대한 사전 지식을 활용하여 토핑의 하위 집합을 표시할 수 있습니다.
- 사용자 입력 검증 - Jen이 몇 시간 후에 꽃을 찾아가고 싶어 한다고 가정하겠습니다. Jen이 적절한 응답을 입력하고 전송하는 시간을 검증할 수 있습니다.
- 사용자의 의도 이행 - Joe가 피자 주문에 필요한 모든 정보를 제공하면, Amazon Lex는 동네 피자 전문점에서 주문할 수 있는 Lambda 함수를 간접 호출하도록 구성할 수 있습니다.

의도를 구성할 때 다음 위치에서 Lambda 함수 를 코드 후크로 지정합니다.

- 대화 코드 후크(초기화 및 검증) - 이 Lambda 함수 는 각 사용자 입력에 대해 간접 호출됩니다 (Amazon Lex가 사용자 의도를 올바르게 이해하고 있다고 가정).
- 이행 코드 후크 - 이 Lambda 함수 는 사용자가 의도를 이행하는 데 필요한 모든 슬롯 데이터를 제공한 후 간접 호출됩니다.

아래 예제 스크린샷에 표시된 것처럼 Amazon Lex 콘솔에서 의도를 선택하고 이러한 코드 후크를 설정할 수 있습니다.

OrderFlowers Latest

Sample utterances

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

I would like to pick up flowers

I would like to order some flowers

Order flowers

Lambda initialization and validation

Initialization and validation code hook

Lambda Function Name

Slots

Priority	Required	Name	Slot type		Prompt
		e.g. Location	e.g. A…		e.g. What city?
1.	✓	FlowerType	Flowe…	1	What type of flow
2.	✓	PickupDate	AMA…	Built-in	What day do you
3.	✓	PickupTime	AMA…	Built-in	At what time do y

Confirmation prompt

Confirmation prompt

Confirm

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

Cancel (if the user says "no")

Okay, I will not place your order.

Fulfillment

AWS Lambda function Return parameters to client

Lambda Function Name

Response

또한 PutIntent 작업의 `dialogCodeHook` 및 `fulfillmentActivity` 필드를 사용하여 코드 후크를 설정할 수도 있습니다.

하나의 Lambda 함수 가 초기화, 검증 및 이행을 수행할 수 있습니다. Lambda 함수 가 수신하는 이벤트 데이터에는 호출자를 대화 또는 이행 코드 후크로 식별하는 필드가 있습니다. 이러한 정보를 사용하여 적절한 코드 부분을 실행할 수 있습니다.

Lambda 함수 를 사용하여 복잡한 대화를 탐색할 수 있는 봇을 구축할 수 있습니다. Lambda 함수 응답에서 `dialogAction` 필드를 사용하여 Amazon Lex 에 특정 조치를 취하라고 지시할 수 있습니다. 예를 들어 `ElicitSlot` 대화 작업을 사용하여 Amazon Lex 에 사용자에게 필요 없는 슬롯 값을 묻도록 지시할 수 있습니다. 확인 프롬프트가 정의되어 있는 경우에는 사용자가 이전 의도를 마치면 `ElicitIntent` 대화 작업을 사용하여 새 의도를 유도할 수 있습니다.

자세한 내용은 Lambda 함수 사용을 참조하세요.

메시지 관리

주제

- 메시지 유형
- 메시지 구성을 위한 컨텍스트
- 지원되는 메시지 형식
- 메시지 그룹
- 응답 카드

봇을 생성할 때 봇에서 클라이언트로 보낼 설명 또는 정보 제공용 메시지를 구성할 수 있습니다. 다음 예제를 살펴보세요.

- 다음의 설명 프롬프트를 사용하여 봇을 구성할 수 있습니다.

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

  Amazon Lex 가 사용자의 의도를 이해하지 못한 경우 클라이언트에 이 메시지를 보냅니다.

- `OrderPizza`이라는 이름의 의도를 지원하는 봇을 생성한다고 가정하겠습니다. 피자 주문의 경우 사용자가 피자 크기, 토핑, 크러스트 유형과 같은 정보를 제공하도록 해야 합니다. 다음 프롬프트를 구성할 수 있습니다.

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

Amazon Lex 는 피자 주문이라는 사용자의 의도를 확인한 후, 클라이언트에 이러한 메시지를 보내 사용자로부터 정보를 얻습니다.

이 섹션에서는 봇 구성에서 사용자 상호 작용을 설계하는 작업에 관해 설명합니다.

메시지 유형

메시지는 프롬프트 또는 문장일 수 있습니다.

- 프롬프트는 일반적으로 질문으로 사용자 응답을 기대합니다.
- 문장은 정보를 제공합니다. 응답을 기대하지 않습니다.

메시지에는 슬롯, 세션 속성 및 요청 속성에 대한 참조가 포함될 수 있습니다. 런타임 실행 시 Amazon Lex 는 이러한 참조를 실제 값으로 대체합니다.

설정된 슬롯 값을 참조하려면 다음 구문을 사용합니다.

```
{SlotName}
```

세션 속성을 참조하려면 다음 구문을 사용합니다.

```
[SessionAttributeName]
```

요청 속성을 참조하려면 다음 구문을 사용합니다.

```
((RequestAttributeName))
```

메시지에는 슬롯 값, 세션 속성 및 요청 속성이 모두 포함될 수 있습니다.

예를 들어, 봇의 OrderPizza 의도에서 다음과 같은 메시지를 구성한다고 가정하겠습니다.

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

이 메시지는 슬롯(PizzaTopping)과 세션 속성(FirstName 및 DeliveryTime)을 모두 참조합니다. 런타임 실행 시 Amazon Lex 는 이러한 자리 표시자를 값으로 바꾸고, 클라이언트에 다음 메시지를 반환합니다.

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

메시지 안에 대괄호([]) 또는 중괄호({})를 포함하려면 백슬래시(\) 이스케이프 문자를 사용합니다. 예를 들면 다음 메시지에는 중괄호와 대괄호가 포함됩니다.

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

클라이언트 애플리케이션으로 반환된 텍스트는 다음과 비슷합니다.

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

세션 속성에 대한 자세한 내용은 런타임 API 작업(PostText 및 PostContent)을 참조하십시오. 예시는 여행 예약에서 확인하십시오.

또한 Lambda 함수 는 메시지를 생성하고 이를 Amazon Lex 에 반환하여 사용자에게 보낼 수 있습니다. 의도 구성 시 Lambda 함수 를 추가하는 경우 메시지를 동적으로 생성할 수 있습니다. 봇을 구성하는 동안 메시지를 제공하면 Lambda 함수 에서 프롬프트를 구성할 필요가 없습니다.

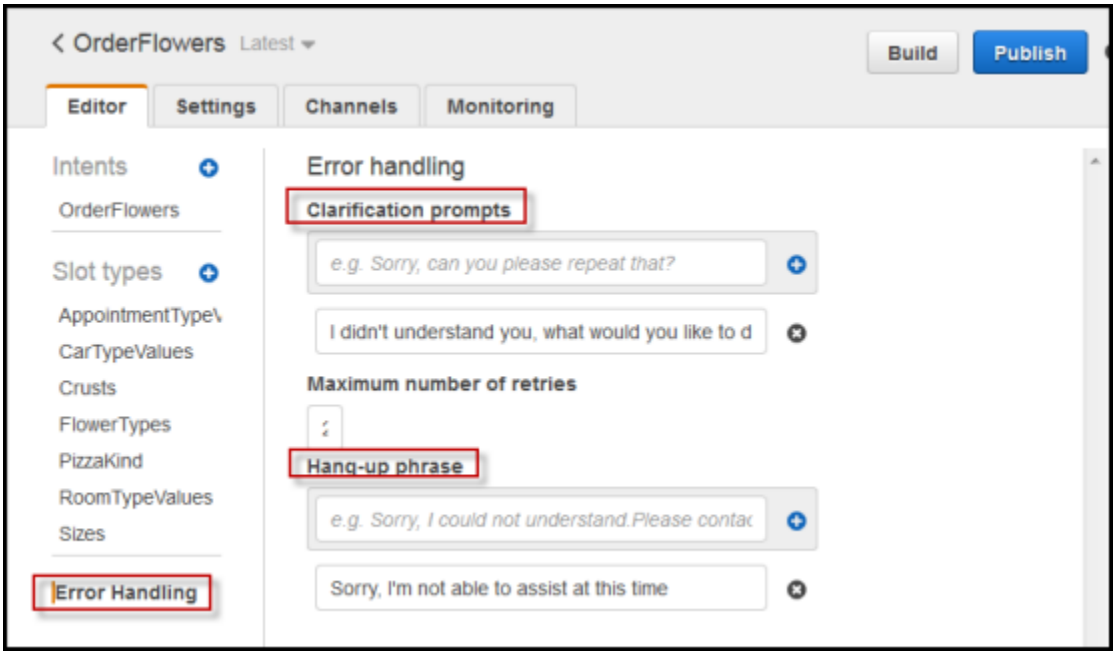
메시지 구성을 위한 컨텍스트

봇을 생성하려는 경우 봇의 설명 프롬프트, 슬롯 값 프롬프트 및 의도의 메시지 등 다양한 컨텍스트에서 메시지를 생성할 수 있습니다. Amazon Lex 는 각 컨텍스트에서 사용자에게 반환할 적절한 메시지를 선택합니다. 각 컨텍스트에 대한 메시지 그룹을 제공할 수 있습니다. 메시지 그룹을 제공할 경우 Amazon Lex 는 그룹에서 메시지 하나를 임의로 선택합니다. 또한 메시지의 형식을 지정하거나 메시지를 함께 그룹화할 수도 있습니다. 자세한 내용은 지원되는 메시지 형식을 참조하세요.

의도와 연결된 Lambda 함수 가 있는 경우 빌드 시에 구성한 메시지 중 하나를 재정의할 수 있습니다. 하지만 이러한 메시지를 사용하는 데는 Lambda 함수 가 필요하지 않습니다.

봇 메시지

설명 프롬프트와 세션 종료 메시지로 봇을 구성할 수 있습니다. 런타임 실행 시 Amazon Lex 가 사용자의 의도를 이해하지 못한 경우 설명 프롬프트를 사용합니다. Amazon Lex 가 세션 종료 메시지를 보내기 전에 설명을 요청하는 횟수를 구성할 수 있습니다. Amazon Lex 콘솔의 오류 처리 섹션에서 다음 이미지와 같이 봇 수준 메시지를 구성합니다.



API를 사용하면 PutBot 작업에서 `clarificationPrompt` 및 `abortStatement`필드를 설정하여 메시지를 구성할 수 있습니다.

의도와 함께 Lambda 함수 를 사용하는 경우 Lambda 함수 는 응답을 반환하여 Amazon Lex 에 사용자의 의도를 묻도록 지정할 수도 있습니다. Lambda 함수 에서 이러한 메시지를 제공하지 않는 경우 Amazon Lex 는 설명 프롬프트를 사용합니다.

슬롯 프롬프트

의도에는 각각의 필요한 슬롯에 대해 한 개 이상의 프롬프트 메시지를 지정해야 합니다. 런타임 실행 시 Amazon Lex 는 이러한 메시지 중 하나를 사용하여 사용자에게 이 슬롯의 값을 입력하라는 메시지를 표시합니다. 예를 들어 `cityName` 슬롯에 대해 유효한 프롬프트는 다음과 같습니다.

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

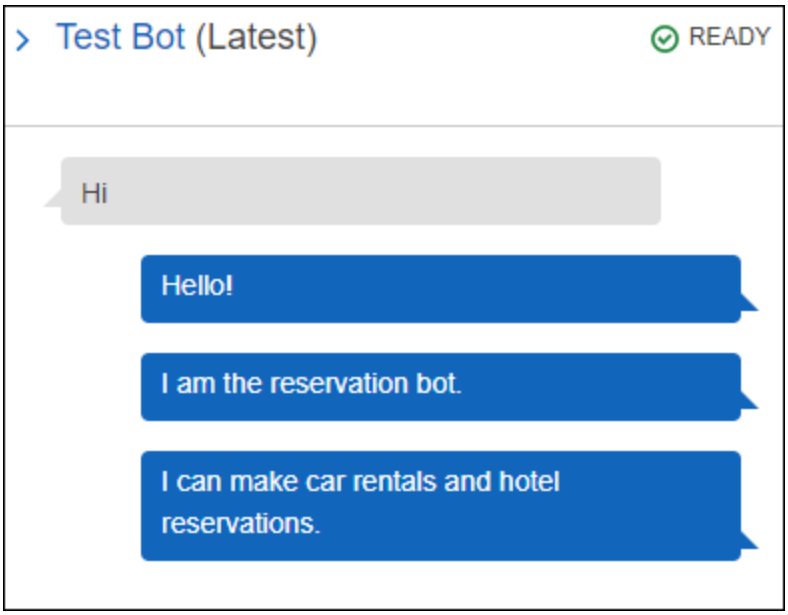
콘솔을 사용하여 각 슬롯에 대해 하나 이상의 프롬프트를 설정할 수 있습니다. 또한 PutIntent 작업을 사용하여 프롬프트 그룹을 생성할 수도 있습니다. 자세한 내용은 메시지 그룹을 참조하세요.

응답

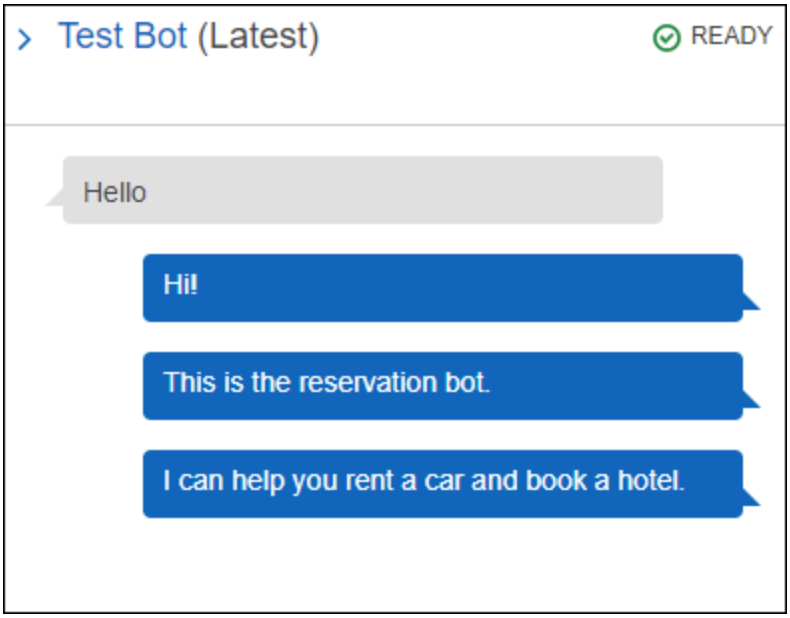
콘솔에서 응답 섹션을 사용하여 봇에 대한 동적 참여 대화를 구축합니다. 응답에 대해 하나 이상의 메시지 그룹을 생성할 수 있습니다. 런타임 시에 Amazon Lex 는 각 메시지 그룹에서 메시지 하나를 선택하여 응답을 빌드합니다. 메시지 그룹에 대한 자세한 내용은 메시지 그룹 섹션을 참조하십시오.

예를 들면 첫 번째 메시지 그룹에는 "안녕하세요", "안녕" 및 "반가워" 같은 다양한 인사말이 포함될 수 있습니다. 두 번째 메시지 그룹에는 "저는 예약 봇입니다" 및 "예약 봇입니다" 같은 다양한 형태의 소개말이 포함될 수 있습니다. 세 번째 메시지 그룹은 "저는 자동차 렌탈과 호텔 예약을 도울 수 있습니다", "자동차 렌탈과 호텔 예약을 할 수 있습니다", "자동차 렌탈과 호텔 예약을 하도록 도울 수 있습니다" 등 봇의 기능을 전달할 수 있습니다.

Lex는 각 메시지 그룹의 메시지를 사용하여 대화의 응답을 동적으로 빌드합니다. 예를 들면 다음과 같은 대화가 있을 수 있습니다.

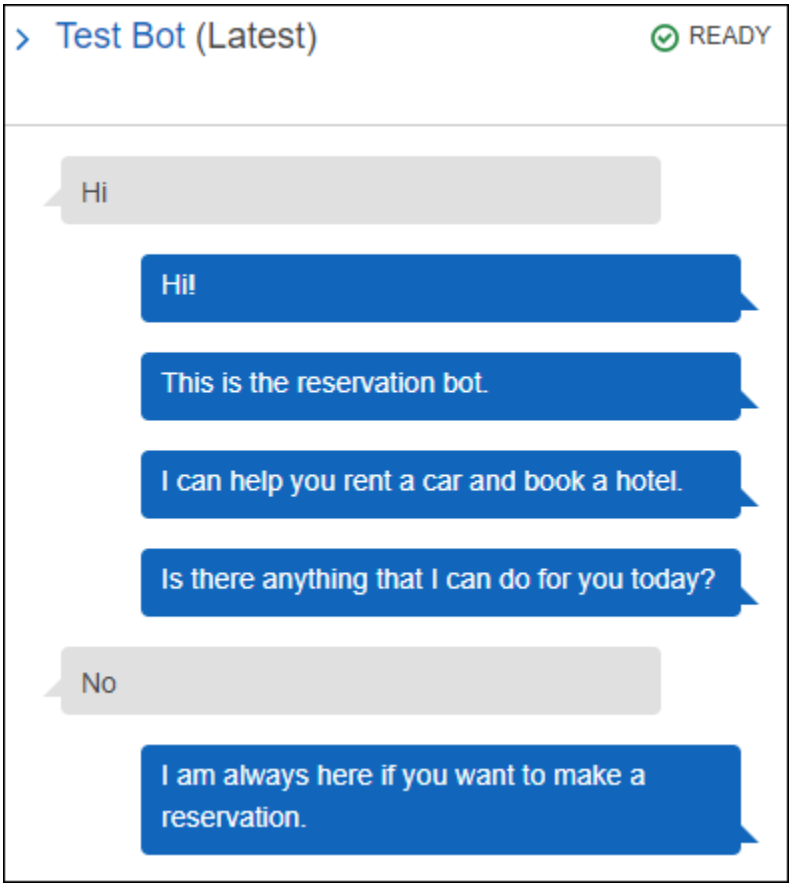


다른 하나는 다음과 같을 수 있습니다.



둘 중 어느 경우든, 사용자는 `BookCar` 또는 `BookHotel` 의도와 같이 새 의도로 응답할 수 있습니다.

응답에서 후속 질문을 하도록 봇을 설정할 수 있습니다. 예를 들면, 이전 대화의 경우 "자동차 렌탈과 호텔 예약을 도와드릴까요?", "예약을 하고 싶으신가요?" 및 "제가 도울 수 있을까요?" 같은 질문을 사용하여 네 번째 메시지 그룹을 만들 수 있습니다. "아니요"를 응답으로 포함하는 메시지의 경우 후속 프롬프트를 만들 수 있습니다. 다음은 예제를 제공합니다.



후속 프롬프트를 만들려면 사용자 응답 대기를 선택합니다. 그 다음 사용자가 "아니요"라고 말할 때 보낼 메시지를 입력합니다. 후속 프롬프트로 사용할 응답을 만드는 경우 설명문에 대한 응답이 "아니요"일 때에 적합한 설명문도 지정해야 합니다. 다음 이미지를 예시로 참조하세요.

API를 사용하여 의도에 응답을 추가하려면 `PutIntent` 작업을 사용합니다. 응답을 지정하려면 `PutIntent` 요청에서 `conclusionStatement` 필드를 설정합니다. 후속 프롬프트를 설정하려면 `followUpPrompt` 필드를 설정하고 사용자가 "아니요"라고 말할 때 보낼 문장을 포함시킵니다. 동일한 의도에 대해 `conclusionStatement` 필드와 `followUpPrompt` 필드를 둘 다 설정할 수는 없습니다.

지원되는 메시지 형식

PostText 작업을 사용하거나 `Accept` 헤더가 `text/plain;charset=utf8`로 설정된 PostContent 작업을 사용하는 경우 Amazon Lex 는 다음 형식의 메시지를 지원합니다.

- `PlainText`—메시지에 일반 UTF-8 텍스트가 포함됩니다.
- `SSML`—메시지에 음성 출력용으로 서식이 지정된 텍스트가 포함됩니다.
- `CustomPayload`—메시지에 클라이언트용으로 생성한 사용자 지정 형식이 포함됩니다. 애플리케이션의 요건을 이행하도록 페이로드를 정의할 수 있습니다.
- `Composite`—메시지는 각 메시지 그룹에서 하나씩 있는 메시지 모음입니다. 메시지 그룹에 대한 자세한 내용은 메시지 그룹 섹션을 참조하십시오.

기본적으로 Amazon Lex 는 특정 프롬프트에 대해 정의된 메시지 중 하나를 반환합니다. 예를 들어 슬롯 값을 유도하기 위한 다섯 개 메시지를 정의하는 경우 Amazon Lex 는 메시지 중 하나를 임의로 선택하여 클라이언트에게 반환합니다.

Amazon Lex 가 런타임 요청 시 클라이언트에 특정 메시지 유형을 반환하도록 하려면 `x-amzn-lex:accept-content-types` 요청 파라미터를 설정합니다. 응답이 요청한 유형으로 제한됩니

다. 지정한 유형의 메시지가 두 개 이상인 경우 Amazon Lex 는 임의로 하나를 반환합니다. `x-amz-lex:accept-content-types` 헤더에 대한 자세한 내용은 응답 유형 설정 섹션을 참조하십시오.

메시지 그룹

메시지 그룹은 특정 프롬프트에 대한 적절한 응답 집합입니다. 봇이 대화에서 응답을 동적으로 빌드하도록 하려면 메시지 그룹을 사용합니다. Amazon Lex 는 클라이언트 애플리케이션에 응답을 반환할 때 각 그룹에서 메시지 하나를 임의로 선택합니다. 각 응답마다 최대 다섯 개 메시지 그룹을 만들 수 있습니다. 각 그룹에는 최대 다섯 개 메시지가 포함될 수 있습니다. 콘솔에서 메시지 그룹을 생성하는 예는 응답 섹션을 참조하십시오.

메시지 그룹을 만들려는 경우 콘솔을 사용하거나 PutBot, PutIntent 또는 PutSlotType 작업을 사용하여 그룹 번호를 메시지에 할당할 수 있습니다. 메시지 그룹을 만들지 않는 경우 또는 메시지 그룹을 하나만 만드는 경우 Amazon Lex 는 `Message` 필드에 메시지를 보냅니다. 클라이언트 애플리케이션은 콘솔에서 두 개 이상의 메시지 그룹을 만든 경우 또는 PutIntent 작업을 사용하여 의도를 생성하거나 업데이트할 때 두 개 이상의 메시지 그룹을 생성하는 경우 응답 하나를 통해 여러 메시지를 가져옵니다.

Amazon Lex 가 그룹에서 메시지를 보낼 때에는, 응답의 `Message` 필드는 메시지를 포함하는 이스케이프된 JSON 객체를 포함합니다. 다음 예제는 여러 메시지를 포함하는 `Message` 필드의 내용을 보여줍니다.

> **Note**
>
> 이 예제는 쉽게 읽을 수 있도록 서식이 지정되었습니다. 응답에는 캐리지 리턴(CR)이 포함되지 않습니다.

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

메시지의 형식을 설정할 수 있습니다. 형식은 다음 중 하나일 수 있습니다.

- PlainText - 메시지가 일반 UTF-8 텍스트로 표시됩니다.
- SSML—음성 합성 마크업 언어 형식의 메시지입니다.
- CustomPayload - 사용자가 지정한 사용자 정의 형식의 메시지입니다.

`PostContent` 및 `PostText` 작업이 `Message` 필드에 반환하는 메시지 형식을 제어하려면 `x-amz-lex:accept-content-types` 요청 속성을 설정합니다. 예를 들면 헤더를 다음으로 설정하는 경우 응답으로 일반 텍스트와 SSML 메시지만 받게 됩니다.

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

특정 메시지 형식을 요청했지만 메시지 그룹에 해당 형식의 메시지가 포함되지 않은 경우, `NoUsableMessageException` 예외가 발생합니다. 메시지 그룹을 사용하여 유형별로 메시지를 그룹화하는 경우 `x-amz-lex:accept-content-types` 헤더를 사용하지 마십시오.

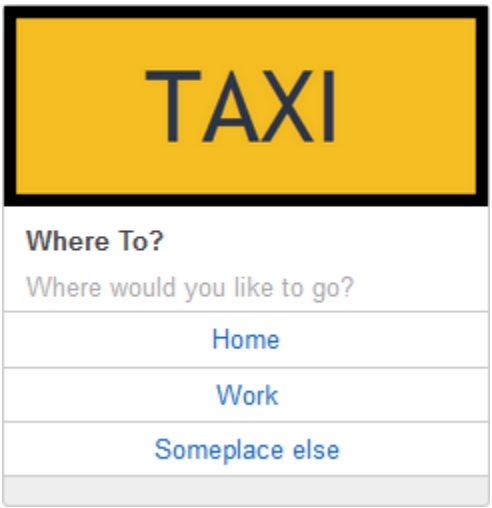
`x-amz-lex:accept-content-types` 헤더에 대한 자세한 내용은 응답 유형 설정 섹션을 참조하십시오.

응답 카드

> **Note**
>
> Amazon Connect 채팅에서는 응답 카드를 사용할 수 없습니다. 하지만 비슷한 기능에 대해서는 채팅에 대화형 메시지 추가를 참조하십시오.

응답 카드에는 프롬프트에 적절한 응답 세트가 들어 있습니다. 응답 카드를 사용하여 사용자의 상호 작용을 간소화하고 텍스트 상호 작용 시 오타를 줄여 봇의 정확도를 높입니다. Amazon Lex 가 클라이언트 애플리케이션에 보내는 각 프롬프트에 대한 응답 카드를 보낼 수 있습니다. Facebook Messenger, Slack, Twilio 및 자체 클라이언트 애플리케이션에서 응답 카드를 사용할 수 있습니다.

예를 들어, 택시 애플리케이션의 경우 "집"에 대한 응답 카드에서 옵션을 구성할 수 있는데 그 값을 사용자의 집 주소로 설정할 수 있습니다. 사용자가 이 옵션을 선택하면 Amazon Lex 는 전체 주소를 입력 텍스트로 받아들입니다. 다음 이미지를 참조하세요.



다음 프롬프트에 대한 응답 카드를 정의할 수 있습니다.

- 결론 문장
- 확인 프롬프트
- 후속 프롬프트
- 거부 문장
- 슬롯 유형 표현

각 프롬프트에 대해 응답 카드는 하나만 정의할 수 있습니다.

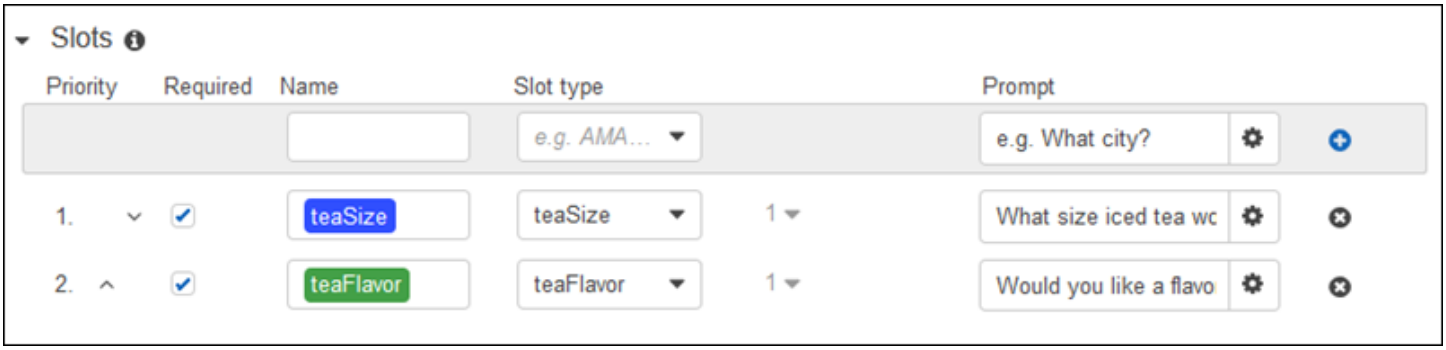
의도를 생성할 때 응답 카드를 구성합니다. 콘솔 또는 PutIntent 작업을 사용하여 빌드 시에 정적 응답 카드를 정의할 수 있습니다. 또는 런타임 시 Lambda 함수 에서 동적 응답 카드를 정의할 수 있습니다. 정적 및 동적 응답 카드를 모두 정의하면, 동적 응답 카드가 우선 적용됩니다.

Amazon Lex 는 클라이언트가 이해할 수 있는 형식으로 응답 카드를 보냅니다. Facebook Messenger, Slack, Twilio에 맞춰 응답 카드를 변환합니다. 다른 클라이언트의 경우 Amazon Lex 는 PostText 응답에 JSON 구조를 보냅니다. 예를 들어, 클라이언트가 Facebook Messenger인 경우 Amazon Lex 는 응답 카드를 일반 템플릿으로 변환합니다. Facebook Messenger 일반 템플릿에 대한 자세한 내용은 Facebook 웹 사이트의 일반 템플릿을 참조하십시오. JSON 구조에 대한 예는 동적 응답 카드 생성 섹션을 참조하십시오.

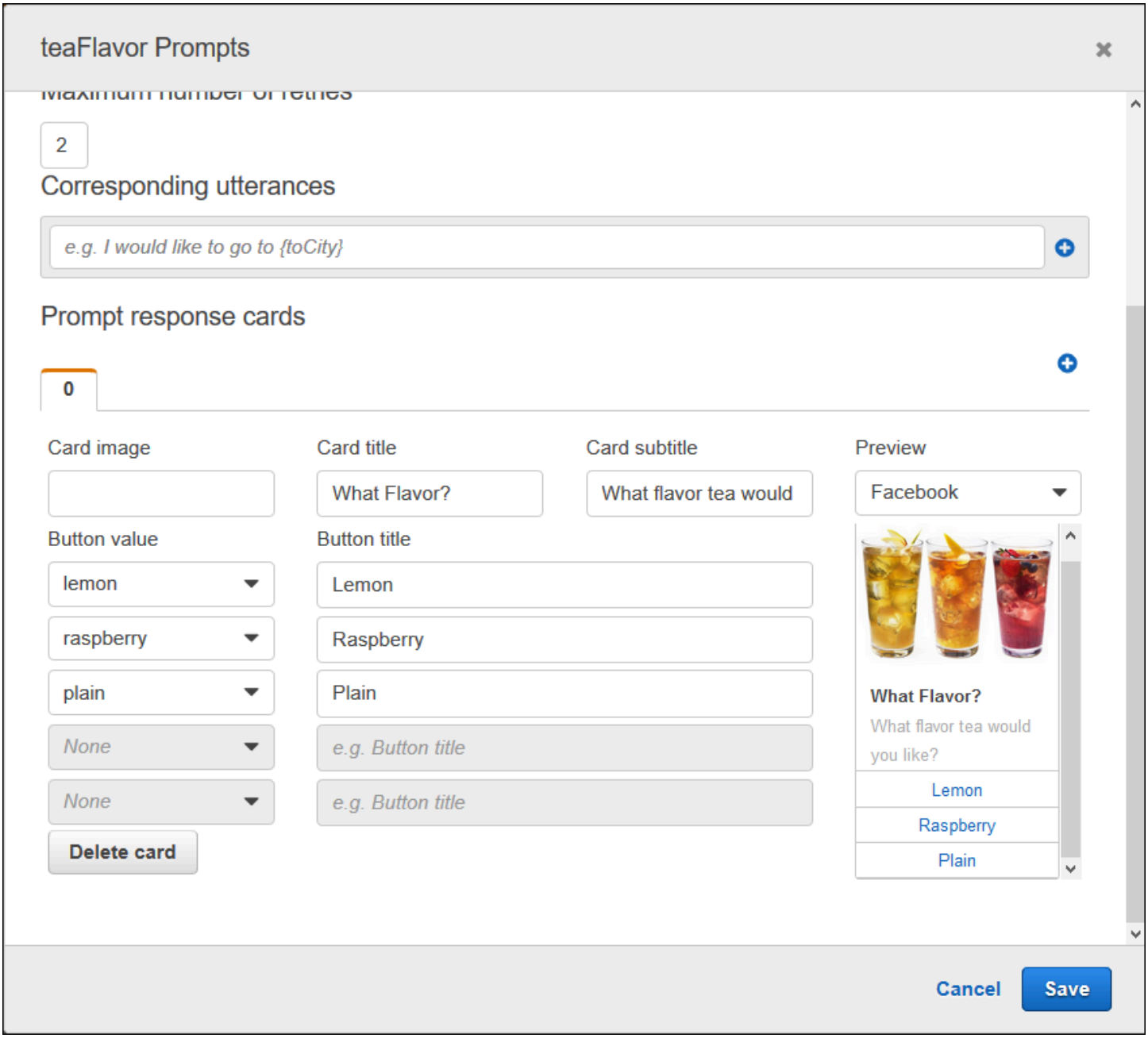
PostText 작업에서만 응답 카드를 사용할 수 있습니다. PostContent 작업에서는 응답 카드를 사용할 수 없습니다.

정적 응답 카드 정의

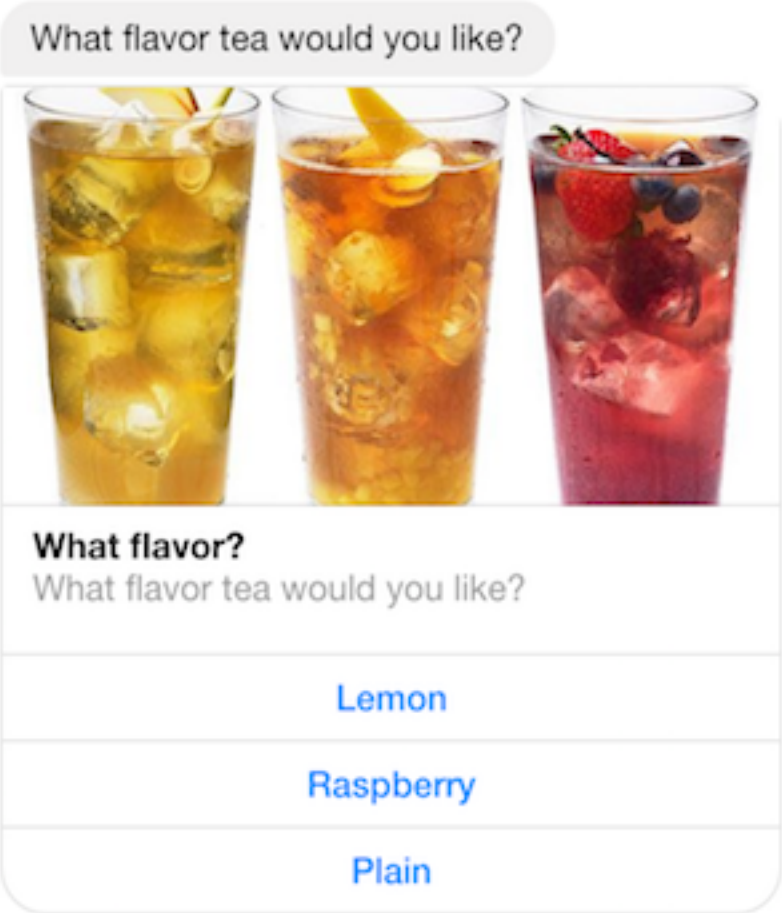
의도를 생성할 때 PutBot 작업 또는 Amazon Lex 콘솔을 사용하여 정적 응답 카드를 정의합니다. 정적 응답 카드는 의도와 동시에 정의됩니다. 응답이 고정된 경우 정적 응답 카드를 사용합니다. 맛에 대한 슬롯이 있는 의도를 사용해 봇을 생성한다고 가정하겠습니다. 다음 콘솔 스크린샷에 나와 있듯이 맛 슬롯을 정의할 때 프롬프트를 지정합니다.



다음 예제에 나와 있듯이 프롬프트를 정의할 때 선택적으로 응답 카드를 연결하고 PutBot 작업을 사용하거나 Amazon Lex 콘솔에서 세부 정보를 정의할 수도 있습니다.



이제 봇을 Facebook Messenger와 통합했다고 가정하겠습니다. 사용자는 버튼을 클릭하여 다음 그림과 같이 맛을 선택할 수 있습니다.



응답 카드의 내용을 사용자 지정하기 위해 세션 속성을 참조할 수 있습니다. 런타임 시에 Amazon Lex 는 이러한 참조를 세션 속성의 해당 값으로 대체합니다. 자세한 내용은 Setting Session Attributes을 참조하세요. 예시는 응답 카드 사용에서 확인하십시오.

동적 응답 카드 생성

런타임 시 응답 카드를 동적인 방식으로 생성하려면 의도에 초기화 및 검증 Lambda 함수 를 사용합니다. 동적 응답 카드는 런타임 시 Lambda 함수 에서 응답이 결정되는 경우 사용합니다. 사용자 입력에 대한 응답으로 Lambda 함수 는 응답 카드를 생성하고 응답의 `dialogAction` 섹션에서 해당 응답 카드를 반환합니다. 자세한 내용은 응답 형식을 참조하세요.

다음은 `responseCard` 요소를 보여 주는 Lambda 함수 의 부분 응답입니다. 이 응답은 이전 섹션에 표시된 것과 유사한 사용자 환경을 생성합니다.

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

예시는 스케줄 예약에서 확인하십시오.

대화 컨텍스트 관리

대화 컨텍스트는 사용자, 애플리케이션 또는 Lambda 함수가 의도를 이행하기 위해 Amazon Lex 봇에 제공하는 정보입니다. 대화 컨텍스트에는 사용자가 제공하는 슬롯 데이터, 클라이언트 애플리케이션에서 설정한 요청 속성, 클라이언트 애플리케이션과 Lambda 함수가 생성하는 세션 속성이 포함됩니다.

주제

- 의도 컨텍스트 설정
- 기본 슬롯 값 사용
- Setting Session Attributes
- Setting Request Attributes
- 세션 시간 제한 설정

- 의도 간 정보 공유
- 복합 속성 설정

의도 컨텍스트 설정

Amazon Lex가 컨텍스트 기반으로 의도를 트리거하도록 할 수 있습니다. 컨텍스트는 봇을 정의할 때 인텐트와 연결할 수 있는 상태 변수입니다.

콘솔이나 PutIntent 작업을 사용하여 의도를 만들 때 의도의 컨텍스트를 구성합니다. 영어 (미국) (en-US) 로케일의 컨텍스트만 사용할 수 있으며,PutBot 작업을 통해 봇을 만들`true`때 `enableModelImprovements` 파라미터를 로 설정한 경우에만 사용할 수 있습니다.

컨텍스트에는 출력 컨텍스트와 입력 컨텍스트라는 두 가지 유형의 관계가 있습니다. 관련 의도가 수행되면 출력 컨텍스트가 활성화됩니다. 출력 컨텍스트는 PostText 또는 PostContent 작업의 응답으로 애플리케이션에 반환되며, 이 컨텍스트는 현재 세션에 맞게 설정됩니다. 컨텍스트가 활성화된 후에는 컨텍스트가 정의될 때 구성된 턴 수 또는 시간 제한 동안 활성 상태를 유지합니다.

입력 컨텍스트는 의도를 인식할 수 있는 조건을 지정합니다. 의도는 모든 입력 컨텍스트가 활성화된 경우에만 대화 중에 인식될 수 있습니다. 입력 컨텍스트가 없는 의도는 항상 인식될 수 있습니다.

Amazon Lex는 출력 컨텍스트로 의도를 수행하여 활성화되는 컨텍스트의 수명 주기를 자동으로 관리합니다. `PostContent` 또는 `PostText` 작업에 대한 직접 호출에서 활성 컨텍스트를 설정할 수도 있습니다.

또한 의도에 대한 Lambda 함수를 사용하여 대화의 컨텍스트를 설정할 수 있습니다. Amazon Lex의 출력 컨텍스트는 Lambda 함수 입력 이벤트로 전송됩니다. Lambda 함수는 응답으로 컨텍스트를 전송할 수 있습니다. 자세한 내용은 Lambda 함수 입력 이벤트 및 응답 형식을 참조하세요.

예를 들어, "book_car_fulfilled"라는 출력 컨텍스트를 반환하도록 구성된 렌터카를 예약하려는 의도가 있다고 가정해 보겠습니다. 의도가 이행되면 Amazon Lex는 출력 컨텍스트 변수 "book_car_fulfilled"를 설정합니다. "book_car_fulfilled"는 활성 컨텍스트이므로 "book_car_fulfilled" 컨텍스트가 입력 컨텍스트로 설정된 의도는 이제 인식 대상으로 간주됩니다. 단, 사용자 표현이 해당 의도를 이끌어내려는 시도로 인식되어야 합니다. 영수증을 이메일로 보내거나 예약을 수정하는 등 차량 예약 이후에만 의미가 있는 의도에 이 방법을 사용할 수 있습니다.

출력 컨텍스트

Amazon Lex는 의도가 이행되면 의도의 출력 컨텍스트를 활성화합니다. 출력 컨텍스트를 사용하여 현재 의도의 후속 조치로 적합한 의도를 제어할 수 있습니다.

각 컨텍스트에는 세션에서 유지 관리되는 파라미터 목록이 있습니다. 파라미터는 수행된 의도의 슬롯 값입니다. 이 매개변수를 사용하여 다른 의도의 슬롯 값을 미리 채울 수 있습니다. 자세한 내용은 기본 슬롯 값 사용을 참조하세요.

콘솔이나 PutIntent 작업을 사용하여 의도를 생성할 때 출력 컨텍스트를 구성합니다. 둘 이상의 출력 컨텍스트로 의도를 구성할 수 있습니다. 의도가 이행되면 모든 출력 컨텍스트가 활성화되고 PostText 또는 PostContent 응답에 반환됩니다.

다음은 콘솔을 사용하여 의도에 출력 컨텍스트를 할당하는 방법입니다.

출력 컨텍스트를 정의할 때는 TTL(Time to Live) 즉, 컨텍스트가 Amazon Lex의 응답에 포함되는 시간 길이 또는 턴 수 또한 정의합니다. 턴은 애플리케이션에서 Amazon Lex로 보내는 한 번의 요청입니다. 턴 수 또는 시간이 만료되면 컨텍스트는 더 이상 활성화되지 않습니다.

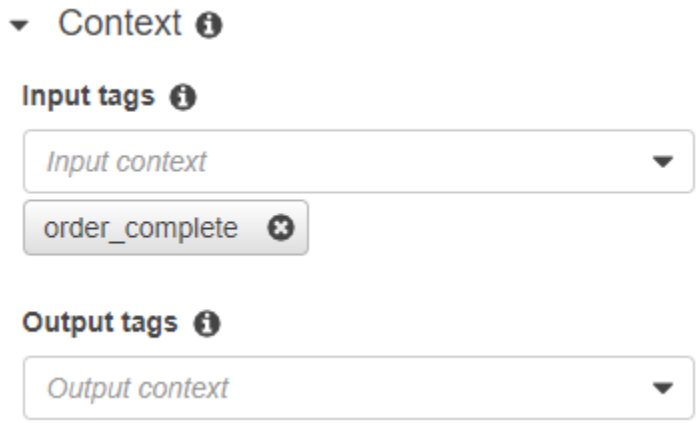
애플리케이션은 필요에 따라 출력 컨텍스트를 사용할 수 있습니다. 예를 들어, 애플리케이션은 출력 컨텍스트를 사용하여 다음을 수행할 수 있습니다.

- 컨텍스트를 기반으로 응용 프로그램의 동작을 변경합니다. 예를 들어 여행 애플리케이션은 "book_car_fulfilled" 컨텍스트에서 "rental_hotel_fulfilled"와는 다른 작업을 수행할 수 있습니다.
- 출력 컨텍스트를 Amazon Lex에 다음 표현의 입력 컨텍스트로 반환합니다. Amazon Lex가 표현을 의도를 이끌어내려는 시도로 인식하면 반환될 수 있는 의도를 특정 컨텍스트를 가진 의도로 제한합니다.

컨텍스트 입력

대화에서 의도가 인식되는 지점을 제한하도록 입력 컨텍스트를 설정합니다. 입력 컨텍스트가 없는 의도는 항상 인식될 수 있습니다.

콘솔이나 `PutIntent` 작업을 사용하여 의도가 응답하는 입력 컨텍스트를 설정합니다. 의도에는 입력 컨텍스트가 둘 이상 있을 수 있습니다. 다음은 콘솔을 사용하여 의도에 입력 컨텍스트를 할당하는 방법을 보여줍니다.



입력 컨텍스트가 두 개 이상인 의도의 경우 의도를 트리거하려면 모든 컨텍스트가 활성 상태여야 합니다. PostText, PostContent 또는 PutSession 작업을 호출할 때 입력 컨텍스트를 설정할 수 있습니다.

현재 활성 컨텍스트에서 기본값을 가져오도록 의도에서 슬롯을 구성할 수 있습니다. Amazon Lex가 새 의도를 인식하지만 슬롯 값을 받지 못할 때 기본값이 사용됩니다. 슬롯을 정의할 때 컨텍스트 이름과 슬롯 이름을 `#context-name.parameter-name` 형식으로 지정합니다. 자세한 내용은 기본 슬롯 값 사용 단원을 참조하십시오.

기본 슬롯 값 사용

기본값을 사용하는 경우 사용자 입력으로 슬롯이 제공되지 않을 때 새 의도에 맞게 채워질 슬롯 값의 소스를 지정합니다. 이 소스는 이전 대화상자, 요청 또는 세션 속성이거나 빌드 시 설정한 고정 값일 수 있습니다.

다음을 기본값의 소스로 사용할 수 있습니다.

- 이전 대화 상자(컨텍스트) – #context -name.parameter-name
- 세션 속성 – [attribute-name]
- 요청 속성 – <attribute-name>
- 고정 값 – 이전 값과 일치하지 않는 모든 값

PutIntent 작업을 사용하여 의도에 슬롯을 추가할 때 기본값 목록을 추가할 수 있습니다. 기본값은 나열된 순서대로 사용됩니다. 예를 들어 다음과 같은 정의의 슬롯이 있는 의도를 가정합니다.

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

의도가 인식되면 이름이 "reservation-start-date"라는 슬롯의 값은 다음 중 하나로 설정됩니다.

1. "book-car-fulfilled" 컨텍스트가 활성화된 경우 "startDate" 매개 변수의 값이 기본값으로 사용됩니다.
2. "book-car-fulfilled" 컨텍스트가 비활성화된 경우 또는 "startDate" 매개 변수가 설정되지 않은 경우, "reservationStartDate"의 세션 속성의 값이 기본값으로 사용됩니다.
3. 처음 두 기본값을 모두 사용하지 않으면 슬롯에 기본값이 없으며 Amazon Lex는 평소와 같이 값을 가져옵니다.

슬롯에 기본값을 사용하면 필요한 경우에도 슬롯이 추출되지 않습니다.

Setting Session Attributes

세션 속성에는 세션 중에 봇과 클라이언트 애플리케이션 간에 전달되는 애플리케이션별 정보가 포함됩니다. Amazon Lex는 봇에 대해 구성된 모든 Lambda 함수에 세션 속성을 전달합니다. Lambda 함수가 세션 속성을 추가 또는 업데이트하는 경우 Amazon Lex는 새로운 정보를 클라이언트 애플리케이션에 다시 전달합니다. 예시:

- 연습 1: 청사진을 사용하여 Amazon Lex 봇 생성(콘솔)에서 예제 봇은 `price` 세션 속성을 사용하여 꽃 가격을 유지합니다. Lambda 함수는 주문한 꽃 유형에 따라 이 속성을 설정합니다. 자세한 내용은 5단계(선택 사항): 정보 흐름의 세부 정보 검토(콘솔) 단원을 참조하십시오.

- 여행 예약에서 예제 봇은 대화 중에 `currentReservation` 세션 속성을 사용하여 슬롯 유형 데이터의 사본을 유지 관리하여 호텔을 예약하거나 렌터카를 예약합니다. 자세한 내용은 정보 흐름의 세부 정보 단원을 참조하십시오.

Lambda 함수의 세션 속성을 사용하여 봇을 초기화하고 프롬프트 및 응답 카드를 사용자 지정합니다. 예시:

- 초기화 — 피자 주문 봇에서 클라이언트 애플리케이션은 PostContent 또는 PostText 작업에 대한 첫 번째 호출 시 사용자의 위치를 세션 속성으로 전달합니다. 예: `"Location": "111 Maple Street"`. Lambda 함수는 이 정보를 사용하여 주문할 수 있는 가장 가까운 피자 가게를 찾습니다.
- 프롬프트 개인화 - 세션 속성을 참조하도록 프롬프트와 응답 카드를 구성합니다. 예를 들어, "[FirstName]님, 어떤 토핑을 드시겠어요?" 사용자의 이름을 세션 속성(`{"FirstName": "Jo"}`)으로 전달하면 Amazon Lex가 자리 표시자를 이름으로 대체합니다. 그런 다음 사용자에게 "Jo님, 어떤 토핑을 원하세요?"라는 맞춤형 프롬프트를 보냅니다.

세션 속성은 세션 기간 동안 암호화된 저장소에서 지속됩니다. Amazon Lex는 세션이 종료될 때까지 이를 암호화된 데이터 스토어에 저장합니다. 클라이언트는 `sessionAttributes` 필드가 값으로 설정된 상태에서 PostContent 또는 PostText 작업을 호출하여 요청에 세션 속성을 생성할 수 있습니다. Lambda 함수는 응답에 세션 속성을 생성할 수 있습니다. 클라이언트 또는 Lambda 함수가 세션 속성을 생성한 후, 저장된 속성 값은 클라이언트 애플리케이션이 Amazon Lex에 대한 요청에 `sessionAttribute` 필드를 포함하지 않을 때마다 사용됩니다.

예를 들어, 다음과 같은 두 개의 세션 속성 `{"x": "1", "y": "2"}`이 있다고 가정하겠습니다. 클라이언트가 `sessionAttributes` 필드를 지정하지 않고 `PostContent` or `PostText` 작업을 직접적으로 호출하면 Amazon Lex는 저장된 세션 속성 (`{"x": 1, "y": 2}`)와 함께 Lambda 함수를 직접적으로 호출합니다. Lambda 함수가 세션 속성을 반환하지 않는 경우 Amazon Lex는 저장된 세션 속성을 클라이언트 애플리케이션에 반환합니다.

클라이언트 애플리케이션 또는 Lambda 함수가 세션 속성을 전달한 경우, Amazon Lex는 저장된 세션 속성 업데이트합니다. `{"x": 2}`과 같은 기존 값을 전달하면 저장된 값이 업데이트됩니다. 새 세션 속성 세트(예를 들어, `{"z": 3}`) 를 전달하면 기존 값은 제거되고 새 값만 유지됩니다. 빈 맵 `{}`가 전달되면 저장된 값이 지워집니다.

세션 속성을 Amazon Lex로 전송하려면 속성의 string-to-string 맵을 생성합니다. 다음은 세션 속성을 매핑하는 방법을 보여줍니다.

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

```
    "attributeName": "attributeValue"
}
```

PostText 작업의 경우 다음과 같이 sessionAttributes 필드를 사용하여 요청 본문에 맵을 삽입합니다.

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

PostContent 작업의 경우, 맵을 base64로 인코딩한 다음 x-amz-lex-session-attributes 헤더로 전송합니다.

세션 속성으로 이진수 또는 구조화된 데이터를 보내는 경우 먼저 데이터를 단순 문자열로 변환해야 합니다. 자세한 내용은 복합 속성 설정 단원을 참조하십시오.

Setting Request Attributes

요청 속성은 요청 관련 정보를 포함하고 있고 현재 요청에만 적용됩니다. 클라이언트 애플리케이션은 이 정보를 Amazon Lex로 전송합니다. 전체 세션에서 유지할 필요가 없는 정보를 전달하려면 요청 속성을 사용합니다. 요청 속성을 직접 생성할 수도 있고 사전 정의된 속성을 사용할 수도 있습니다. 요청 속성을 보내려면 the section called “PostText” 요청에 포함된 the section called “PostContent” 또는 requestAttributes 필드 안의 x-amz-lex-request-attributes 헤더를 사용합니다. 요청 속성은 세션 속성처럼 요청 전체에 지속되지 않기 때문에 요청 속성은 PostContent 또는 PostText 응답에서 반환되지 않습니다.

> **Note**
>
> 요청 간에 유지되는 정보를 보내려면 세션 속성을 사용하십시오.

네임스페이스 x-amz-lex:는 사전 정의된 요청 속성용으로 보류되어 있습니다. x-amz-lex: 접두사를 사용하여 요청 속성을 생성하지 마세요.

사전 정의된 요청 속성 설정

Amazon Lex는 봇으로 전송된 정보를 처리하는 방식을 관리하기 위해 사전 정의된 요청 속성을 제공합니다. 속성은 전체 세션 동안 지속되지 않으므로 각 요청에서 사전 정의된 속성을 전송해야 합니다. 사전 정의된 모든 속성은 x-amz-lex: 네임스페이스에 있습니다.

Amazon Lex는 다음과 같은 사전 정의된 속성 외에도 메시징 플랫폼을 위한 사전 정의된 속성을 제공합니다. 이러한 속성의 목록은 메시징 플랫폼에 Amazon Lex 봇 배포하기을 참조하십시오.

응답 유형 설정

기능이 다른 두 개의 클라이언트 애플리케이션을 사용하는 경우 응답의 메시지 형식을 제한해야 할 수 있습니다. 예를 들어 웹 클라이언트에 보내는 메시지는 일반 텍스트로 제한하고 모바일 클라이언트는 일반 텍스트와 음성 합성 마크업 언(SSML)를 모두 사용할 수 있도록 할 수 있습니다. PostContent 및 PostText 작업이 필드에 반환하는 메시지 형식을 제어하려면 `x-amz-lex:accept-content-types"` 요청 속성을 사용합니다.

다음 메시지 유형의 모든 조합으로 속성을 설정할 수 있습니다.

- `PlainText`—메시지에 일반 UTF-8 텍스트가 포함됩니다.
- `SSML`—메시지에 음성 출력용으로 서식이 지정된 텍스트가 포함됩니다.
- `CustomPayload`—메시지에 클라이언트용으로 생성한 사용자 지정 형식이 포함됩니다. 애플리케이션의 요건을 이행하도록 페이로드를 정의할 수 있습니다.

Amazon Lex는 응답 `Message` 필드에 지정된 유형의 메시지만 반환합니다. 값을 쉼표로 구분하여 둘 이상의 값을 설정할 수 있습니다. 메시지 그룹을 사용하는 경우 모든 메시지 그룹에는 지정된 유형의 메시지가 하나 이상 포함되어야 합니다. 그러지 않을 경우 `NoUsableMessageException` 오류가 발생합니다. 자세한 내용은 메시지 그룹 단원을 참조하십시오.

> **Note**
>
> `x-amz-lex:accept-content-types` 요청 속성은 HTML 본문의 내용에 영향을 주지 않습니다. `PostText` 작업 응답의 내용은 항상 일반 UTF-8 텍스트입니다. `PostContent` 작업 응답 본문에는 요청 안에 있는 `Accept` 헤더에서 설정된 형식의 데이터가 포함됩니다.

선호 시간대 설정

날짜를 확인하는 데 사용되는 시간대를 사용자의 시간대를 기준으로 설정하려면 `x-amz-lex:time-zone` 요청 속성을 사용하십시오. `x-amz-lex:time-zone` 속성에 시간대를 지정하지 않는 경우 기본값은 봇에 사용하는 지역에 따라 달라집니다.

지역	기본 설정 시간대
미국 동부(버지니아 북부)	`America/New_York`
미국 서부(오레곤)	`America/Los_Angeles`
아시아 태평양(싱가포르)	`Asia/Singapore`
아시아 태평양(시드니)	`Australia/Sydney`
아시아 태평양(도쿄)	`Asia/Tokyo`
유럽(프랑크푸르트)	`Europe/Berlin`
유럽(아일랜드)	`Europe/Dublin`
유럽(런던)	`Europe/London`

예를 들어, 사용자가 "패키지를 어느 날짜에 배송하시겠습니까?" 라는 프롬프트에 `tomorrow` 응답하는 경우. 패키지가 배송되는 실제 날짜는 사용자의 시간대에 따라 다릅니다. 예를 들어 뉴욕의 경우 9월 16일 01:00이고 로스앤젤레스의 경우 9월 15일 22:00입니다. 서비스가 미국 동부 (버지니아 북부) 지역에서 운영되고 있고 로스앤젤레스에 있는 사람이 기본 시간대를 사용하여 "내일"에 배송되도록 패키지를 주문하는 경우 패키지는 16일이 아닌 17일에 배송됩니다. 하지만 `x-amz-lex:time-zone` 요청 속성을 `America/Los_Angeles`로 설정하면 패키지는 16일에 배달됩니다.

속성을 원하는 인터넷 지정 번호 기관(IANA) 시간대 이름으로 설정할 수 있습니다. 시간대 이름 목록은 Wikipedia의 tz 데이터베이스 시간대 목록을 참조하십시오.

사용자-정의 요청 속성 설정

사용자-정의 요청 속성은 각 요청에서 봇에 보내는 데이터입니다. `PostContent` 요청의 `amz-lex-request-attributes` 헤더나 `PostText` 요청의 `requestAttributes` 필드에 정보를 전송합니다.

세션 속성을 Amazon Lex로 전송하려면 속성의 string-to-string 맵을 생성합니다. 다음은 세션 속성을 매핑하는 방법을 보여줍니다.

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

```
    "attributeName": "attributeValue"
}
```

PostText 작업의 경우 다음과 같이 requestAttributes 필드를 사용하여 요청 본문에 맵을 삽입합니다.

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

PostContent 작업의 경우, 맵을 base64로 인코딩한 다음 x-amz-lex-request-attributes 헤더로 전송합니다.

세션 속성으로 이진수 또는 구조화된 데이터를 보내는 경우 먼저 데이터를 단순 문자열로 변환해야 합니다. 자세한 내용은 복합 속성 설정 단원을 참조하세요.

세션 시간 제한 설정

Amazon Lex는 대화 세션이 종료될 때까지 슬롯 데이터 및 세션 속성과 같은 컨텍스트 정보를 유지합니다. 봇의 세션 지속 시간을 제어하려면 세션 제한 시간을 설정하세요. 기본적으로 세션 기간은 5분이지만 0~1,440분(24시간) 사이의 기간을 지정할 수 있습니다.

예를 들어, OrderShoes 및 GetOrderStatus 와 같은 의도를 지원하는 ShoeOrdering 봇을 생성한다고 가정해 보겠습니다. Amazon Lex는 사용자의 의도가 신발을 주문하는 것임을 감지하면 슬롯 데이터를 요청합니다. 예를 들어 신발 사이즈, 색상, 브랜드 등을 묻습니다. 사용자가 일부 슬롯 데이터를 제공했지만 신발 구매를 완료하지 않은 경우 Amazon Lex는 전체 세션의 슬롯 데이터와 세션 속성을 모두 기억합니다. 사용자가 세션이 만료되기 전에 세션으로 돌아오면 나머지 슬롯 데이터를 제공하고 구매를 완료할 수 있습니다.

Amazon Lex 콘솔에서는 봇을 생성할 때 세션 제한 시간을 설정합니다. AWS Command Line Interface(AWS CLI) 또는 API에서는 idleSessionTTLInSeconds 필드를 설정함으로써 PutBot 작업을 갖는 봇을 생성 또는 업데이트할 때에 제한 시간을 설정합니다.

의도 간 정보 공유

Amazon Lex는 의도 간 정보 공유를 지원합니다. 의도 간에 공유하려면 세션 속성을 사용하십시오.

예를 들어 ShoeOrdering 봇 사용자는 신발을 주문하는 것으로 시작합니다. 봇은 사용자와 대화하면서 신발 사이즈, 색상, 브랜드와 같은 슬롯 데이터를 수집합니다. 사용자가 주문을 하면 주문을

이행하는 Lambda 함수가 주문 번호가 포함된 `orderNumber` 세션 속성을 설정합니다. 사용자는 `GetOrderStatus` 의도를 사용하여 주문 상태를 확인합니다. 봇은 사용자에게 주문 번호 및 주문 날짜와 같은 슬롯 데이터를 요청할 수 있습니다. 봇이 필수 정보를 수집하면 주문 상태를 반환합니다.

같은 세션에서 사용자가 의도를 바꿀 수 있다고 생각되면 최신 주문 상태를 반환하도록 봇을 설계할 수 있습니다. 사용자에게 주문 정보를 다시 요청하는 대신 `orderNumber` 세션 속성을 사용하여 의도 간에 정보를 공유하고 `GetOrderStatus` 의도를 이행할 수 있습니다. 봇은 사용자가 마지막으로 주문한 상태를 반환하여 이 작업을 수행합니다.

의도 간의 정보 공유의 예제는 여행 예약을 참조하십시오.

복합 속성 설정

세션 및 요청 속성은 속성 및 값의 문자열 간 매핑입니다. 대부분의 경우 문자열 맵을 사용하여 클라이언트 애플리케이션과 봇 간에 속성 값을 전송할 수 있습니다. 하지만 문자열 맵으로 쉽게 변환할 수 없는 이진 데이터나 복잡한 구조를 전송해야 하는 경우도 있습니다. 예를 들어 다음 JSON 객체는 미국에서 인구가 가장 많은 세 도시의 배열을 나타냅니다.

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

```
}
```

이 데이터 배열은 문자열 대 문자열 맵으로 잘 변환되지 않습니다. 이 경우 객체를 간단한 문자열로 변환하여 PostContent 및 PostText 연산과 함께 봇에 보낼 수 있습니다.

예를 들어 JavaScript를 사용하는 경우 객체를 JSON으로 변환하는 `JSON.stringify` 작업과 JSON 텍스트를 JavaScript 객체로 변환하는 `JSON.parse` 작업을 사용할 수 있습니다.

```
// 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);
```

`PostContent` 작업과 함께 세션 속성을 보내려면 다음 JavaScript 코드와 같이, 속성을 요청 헤더에 추가하기 전에 속성을 base64로 인코딩해야 합니다.

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

먼저 데이터를 base64로 인코딩된 문자열로 변환한 다음 이 문자열을 세션 속성의 값으로 전송하여 `PostContent` 및 `PostText` 연산에 이진 데이터를 보낼 수 있습니다.

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

신뢰도 점수 사용

사용자가 말을 하면 Amazon Lex는 자연어 이해(NLU) 를 사용하여 사용자의 요청을 이해하고 적절한 의도를 제시합니다. 기본적으로 Amazon Lex는 봇이 정의한 가장 가능성이 높은 의도를 반환합니다.

경우에 따라 Amazon Lex에서 가장 가능성이 높은 의도를 파악하기 어려울 수 있습니다. 예를 들어, 사용자가 모호한 말을 하거나 비슷한 의도가 두 개 있을 수 있습니다. 적절한 의도를 판단하는 데 도움이 되도록 도메인 지식을 대체 의도 목록의 신뢰도 점수와 결합할 수 있습니다. 신뢰도 점수는 Amazon Lex가 제공하는 등급으로, 의도가 올바른 의도라고 확신하는 정도를 나타냅니다.

두 대체 의도 간의 차이를 확인하기 위해 두 대체 의도의 신뢰도 점수를 비교할 수 있습니다. 예를 들어 한 의도의 신뢰도 점수가 0.95이고 다른 의도의 신뢰도 점수가 0.65이면 첫 번째 의도가 올바른 것일

수 있습니다. 하지만 한 의도의 점수가 0.75점이고 다른 의도의 점수가 0.72인 경우 애플리케이션에서 도메인 지식을 사용하여 구별할 수 있는 모호함이 두 가지 의도 사이에 있습니다.

신뢰도 점수를 사용하여 의도의 표현 변경이 봇의 동작에 영향을 미치는지 확인하는 테스트 애플리케이션을 만들 수도 있습니다. 예를 들어 표현 세트를 사용하여 봇의 의도에 대한 신뢰도 점수를 얻은 다음 새 표현으로 의도를 업데이트할 수 있습니다. 그런 다음 신뢰도 점수를 확인하여 개선이 있었는지 확인할 수 있습니다.

Amazon Lex가 반환하는 신뢰도 점수는 비교 값입니다. 절대 점수로서 의존해서는 안됩니다. 점수는 Amazon Lex의 개선 사항에 따라 변경될 수 있습니다.

신뢰도 점수를 사용하면 Amazon Lex는 각 응답에서 가장 가능성이 높은 의도와 최대 4개의 대체 의도를 관련 점수와 함께 반환합니다. 모든 신뢰도 점수가 임계값 미만인 경우 Amazon Lex는 `AMAZON.FallbackIntent`, `AMAZON.KendraSearchIntent`, 또는 둘 다를 포함합니다(미리 구성한 경우). 기본 임계값을 사용하거나, 직접 자신의 임계값을 설정할 수도 있습니다.

다음 JSON 코드는 PostText 작업에 대한 응답의 `alternativeIntents` 필드를 보여줍니다.

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

봇을 생성 또는 업데이트할 때 임계값을 설정합니다. API 또는 Amazon Lex 콘솔을 사용할 수 있습니다. 아래 나열된 지역의 경우 정확성 향상 및 신뢰도 점수를 활성화하려면 옵트인해야 합니다. 콘솔의 고급 옵션 섹션에서 신뢰도 점수를 선택합니다. API를 사용하여 PutBot 작업을 호출할 때 `enableModelImprovements` 파라미터를 설정하십시오.

- 미국 동부(버지니아 북부)(us-east-1)
- 미국 서부(오레곤)(us-west-2)
- 아시아 태평양(시드니)(ap-southeast-2)
- 유럽(아일랜드)(eu-west-1)

다른 모든 지역에서는 정확도 개선 및 신뢰도 점수 지원이 기본적으로 제공됩니다.

신뢰도 임계값을 변경하려면 콘솔에서 설정하거나 PutBot 작업을 사용하여 신뢰도 임계값을 설정하십시오. 임계값은 1.00에서 0.00 사이의 숫자여야 합니다.

콘솔을 사용하려면 봇을 만들거나 업데이트할 때 신뢰 임계값을 설정하세요.

봇 생성 시 신뢰 임계값 설정하기(콘솔)

- 봇 만들기에서 신뢰도 점수 임계값 필드에 값을 입력합니다.

신뢰도 임계값을 업데이트하려면(콘솔)

1. 봇 목록에서 업데이트할 봇을 선택합니다.
2. 설정 탭을 선택합니다.
3. 왼쪽 탐색 창에서 일반을 선택합니다.
4. 신뢰도 점수 임계값 필드의 값을 업데이트하십시오.

신뢰도 임계값을 설정 또는 업데이트하려면(SDK)

- PutBot 작업 `nluIntentConfidenceThreshold` 파라미터를 설정합니다. 다음 JSON 코드는 설정 중인 파라미터를 보여줍니다.

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

세션 관리

Amazon Lex가 사용자와의 대화에서 사용하는 의도를 변경하려면 대화 상자 코드 후크 Lambda 함수의 응답을 사용하거나 사용자 지정 애플리케이션에서 세션 관리 API를 사용할 수 있습니다.

Lambda 함수 사용

Lambda 함수를 사용하는 경우 Amazon Lex는 함수에 대한 입력이 포함된 JSON 구조를 사용하여 함수를 호출합니다. JSON 구조는 Amazon Lex가 사용자 표현에 대한 가장 가능성이 높은 의도로 식별한 의도가 포함된, `currentIntent`로 불리는 필드를 포함합니다. JSON 구조에는 사용자의 의도를 충족시킬 수 있는 최대 4개의 추가 의도를 포함하는 `alternativeIntents` 필드도 포함됩니다. 각 의도에는 Amazon Lex가 의도에 할당한 신뢰도 점수가 포함된, `nluIntentConfidenceScore`라고 불리는 필드가 포함되어 있습니다.

대체 의도를 사용하려면 Lambda 함수의 `ConfirmIntent` 또는 `ElicitSlot` 대화 작업에서 해당 의도를 지정합니다.

자세한 내용은 Lambda 함수 사용을 참조하세요.

세션 관리 API 사용

현재 의도와 다른 의도를 사용하려면 PutSession 작업을 사용하세요. 예를 들어 Amazon Lex가 선택한 의도보다 첫 번째 대안이 더 낫다고 판단되면 `PutSession` 작업을 사용하여 의도를 변경하여 사용자가 상호 작용하는 다음 의도가 선택한 의도가 되도록 할 수 있습니다.

자세한 내용은 Amazon Lex API로 세션 관리 단원을 참조하십시오.

대화 로그

대화 로그를 활성화하여 봇 상호 작용을 저장합니다. 이러한 로그를 사용하여 봇의 성능을 검토하고 대화 관련 문제를 해결할 수 있습니다. PostText 작업에 대한 텍스트를 로그할 수 있습니다. PostContent 작업에 대한 텍스트와 오디오를 모두 로그할 수 있습니다. 대화 로그를 활성화하면 사용자가 봇과 나누는 대화를 자세히 볼 수 있습니다.

예를 들어 봇이 있는 세션에는 세션 ID가 있습니다. 이 ID를 사용하여 사용자 표현 및 해당 봇 응답을 포함한 대화의 기록을 가져올 수 있습니다. 또한 표현에 대한 의도 이름 및 슬롯 값과 같은 메타 데이터를 얻을 수 있습니다.

> **Note**
>
> 어린이 온라인 사생활 보호법(COPPA)이 적용되는 봇과의 대화 로그는 사용할 수 없습니다.

대화 로그는 별칭에 대해 구성됩니다. 각 별칭은 텍스트 및 오디오 로그에 대해 서로 다른 설정을 가질 수 있습니다. 각 별칭에 대해 텍스트 로그, 오디오 로그 또는 둘 다를 사용하도록 설정할 수 있습니다. 텍스트 로그는 텍스트 입력, 오디오 입력 기록 및 관련 메타 데이터를 CloudWatch Logs에 저장합니다. 오디오 로그는 오디오 입력을 Amazon S3에 저장합니다. AWS KMS 고객 관리형 CMKs.

로깅을 구성하려면 콘솔 또는 PutBotAlias작업을 사용합니다. 봇의 `$LATEST` 별칭이나 Amazon Lex 콘솔에서 사용할 수 있는 테스트 봇에 대한 대화를 기록할 수 없습니다. 별칭에 대한 대화 로그를 활성화하면 해당 별칭에 대한 PostContent 또는 PostText 작업이 구성된 CloudWatch Logs 로그 그룹 또는 S3 버킷에 텍스트 또는 오디오 표현을 기록합니다.

주제

- 대화 로그에 대한 IAM 정책
- 대화 로그 구성
- 대화 로그 암호화
- Amazon CloudWatch 로그에서 텍스트 로그 보기
- Amazon S3에서 오디오 로그에 액세스
- CloudWatch 지표를 통해 대화 로그 상태 모니터링

대화 로그에 대한 IAM 정책

선택하는 로깅 유형에 따라, Amazon Lex는 Amazon CloudWatch Logs 및 S3(Amazon Simple Storage Service) 버킷을 사용해 로그를 보관할 권한이 필요합니다. Amazon Lex가 이러한 리소스에 액세스할 수 있도록 AWS Identity and Access Management 역할과 권한을 생성해야 합니다.

대화 로그에 대한 IAM 역할 및 정책 생성

대화 로그를 활성화하려면, CloudWatch Logs 및 Amazon S3 에 대한 쓰기 권한을 부여해야 합니다. S3 객체에 대해 객체 암호화를 활성화하는 경우 객체를 암호화하는 데 사용되는 AWS KMS 키에 액세스 권한을 부여해야 합니다.

IAM AWS Management Console, IAM API 또는를 사용하여 역할 및 정책을 AWS Command Line Interface 생성할 수 있습니다. 이 지침은 AWS CLI 를 사용하여 역할 및 정책을 생성합니다. 콘솔을 사용한 정책 생성에 대한 자세한 내용은 AWS Identity and Access Management 사용 설명서의 JSON 탭에 정책 생성을 참조하세요.

> **Note**
>
> 다음 코드는 Linux 및 MacOS 용으로 형식이 지정됩니다. Windows의 경우 Linux 줄 연속 문자(\)를 캐럿(^)으로 바꿉니다.

대화 로그에 대한 IAM 역할 을 만들려면

1. **`LexConversationLogsAssumeRolePolicyDocument.json`**이라는 현재 디렉터리에 문서를 만들고 다음 코드를 추가한 다음 저장합니다. 이 정책 문서는 Amazon Lex를 역할에 대한 신뢰할 수 있는 엔터티로써 추가합니다. 이를 통해 Lex는 대화 로그를 위해 구성된 리소스로 로그를 전달하기 위한 역할을 맡을 수 있습니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lex.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. 에서 다음 명령을 AWS CLI실행하여 대화 로그에 대한 IAM 역할을 생성합니다.

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

그런 다음 Amazon Lex가 CloudWatch Logs 에 쓸 수 있도록 하는 정책을 만들어 역할에 연결합니다.

대화 텍스트를 CloudWatch Logs 에 로깅하기 위한 IAM 정책 을 만들려면

1. **LexConversationLogsCloudWatchLogsPolicy.json**이라는 현재 디렉터리에 문서를 만들고 다음 IAM 정책 을 추가한 다음 저장합니다.

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

2. 에서 CloudWatch Logs 로그 그룹에 쓰기 권한을 부여하는 IAM 정책을 AWS CLI생성합니다.

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

3. 대화 로그에 대해 생성한 IAM 역할 에 정책을 연결합니다.

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

S3 버킷에 오디오를 로깅하는 경우 Amazon Lex가 버킷에 쓸 수 있도록 허용하는 정책을 생성합니다.

S3 버킷에 오디오를 로깅하기 위한 IAM 정책을 만들려면

1. **`LexConversationLogsS3Policy.json`**이라는 현재 디렉터리에 문서를 만들고 다음의 정책을 추가한 후 저장합니다.

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

2. 에서 S3 버킷에 쓰기 권한을 부여하는 IAM 정책을 AWS CLI생성합니다.

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

3. 대화 로그에 대해 생성한 역할에 정책을 연결합니다.

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

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

IAM 역할 전달 권한 부여

콘솔 AWS Command Line Interface, 또는 AWS SDK를 사용하여 대화 로그에 사용할 IAM 역할을 지정하는 경우 IAM 역할을 지정하는 사용자는 Amazon Lex에 역할을 전달할 권한이 있어야 합니다. 사용자가 역할을 Amazon Lex에 전달하도록 하려면 사용자, 역할 또는 그룹에 `PassRole` 권한을 부여해야 합니다.

다음 정책은 사용자, 역할 또는 그룹에게 부여할 권한을 정의합니다. `iam:AssociatedResourceArn` 및 `iam:PassedToService` 조건 키를 사용해 권한 범위를 제한할 수 있습니다. 자세한 내용은 사용 AWS Identity and Access Management 설명서의 AWS 서비스에 역할을 전달할 수 있는 사용자 권한 부여 및 IAM 및 AWS STS 조건 컨텍스트 키를 참조하세요.

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

대화 로그 구성

콘솔 또는 `PutBotAlias` 작업의 `conversationLogs` 필드를 사용하여 대화 로그를 활성화 및 비활성화합니다. 오디오 로그, 텍스트 로그 또는 둘 다 설정하거나 해제할 수 있습니다. 새 봇 세션에서 로깅이 시작됩니다. 로그 설정에 대한 변경 사항은 활성 세션에 반영되지 않습니다.

텍스트 로그를 저장하려면 AWS 계정에 Amazon CloudWatch Logs 로그 그룹을 사용합니다. 유효한 로그 그룹 어느 것이든 사용할 수 있습니다. 로그 그룹은 Amazon Lex 봇과 동일한 리전에 있어야 합니다. CloudWatch Logs 로그 그룹 생성에 대한 자세한 내용을 알아보려면 Amazon CloudWatch Logs 사용 설명서의 로그 그룹 및 로그 스트림 작업을 참조하세요.

오디오 로그를 저장하려면 AWS 계정에 Amazon S3 버킷을 사용합니다. 유효한 S3 버킷 어느 것이든 사용할 수 있습니다. 버킷은 Amazon Lex 봇과 동일한 리전에 있어야 합니다. Amazon S3 버킷에 대한 자세한 내용은 Amazon Simple Storage Service 시작 가이드의 버킷 생성을 참조하세요.

Amazon Lex가 구성된 로그 그룹 또는 버킷에 쓸 수 있도록 하는 정책을 IAM 역할 에 제공해야 합니다. 자세한 내용은 대화 로그에 대한 IAM 역할 및 정책 생성 단원을 참조하십시오.

를 사용하여 서비스 연결 역할을 생성하는 경우 다음과 같이 `custom-suffix` 옵션을 사용하여 역할에 사용자 지정 접미사를 추가 AWS Command Line Interface해야 합니다.

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

대화 로그를 활성화하는 데 사용하는 IAM 역할에 `iam:PassRole` 권한이 있어야 합니다. 다음 정책이 역할에 연결되어야 합니다.

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

대화 로그 활성화

콘솔을 사용하여 로그를 활성화하려면

1. https://console.aws.amazon.com/lex에서 Amazon Lex 콘솔을 엽니다.
2. 목록에서 봇을 선택합니다.
3. 설정 탭을 선택한 다음 왼쪽 메뉴에서 대화 로그를 선택합니다.

4. 별칭 목록에서 대화 로그를 구성할 별칭의 설정 아이콘을 선택합니다.
5. 텍스트, 오디오 또는 둘 다 기록할지 선택합니다.
6. 텍스트 로깅의 경우 Amazon CloudWatch Logs 로그 그룹 이름을 입력합니다.
7. 오디오 로깅의 경우 S3 버킷 정보를 입력합니다.
8. 선택 사항. 오디오 로그를 암호화하려면 암호화에 사용할 AWS KMS 키를 선택합니다.
9. 필요한 권한이 있는 IAM 역할을 선택합니다.
10. 저장을 선택하여 대화 로깅을 시작합니다.

API를 사용한 텍스트 로그를 활성화하려면

1. `conversationLogs` 필드의 `logSettings` 멤버에 있는 항목으로 PutBotAlias 작업을 호출합니다.
   - `destination` 멤버를 `CLOUDWATCH_LOGS`로 설정
   - `logType` 멤버를 `TEXT`로 설정
   - `resourceArn` 멤버를 로그의 대상인 CloudWatch Logs 로그 그룹의 Amazon 리소스 이름(ARN)으로 설정
2. `conversationLogs` 필드의 `iamRoleArn` 멤버를 IAM 역할(특정한 리소스의 대화 로그를 활성화할 수 있는 필수 권한을 가진)의 Amazon 리소스 이름(ARN)으로 설정합니다.

API를 사용하여 오디오 로그를 설정하려면

1. `conversationLogs` 필드의 `logSettings` 멤버에 있는 항목으로 PutBotAlias 작업을 호출합니다.
   - `destination` 멤버를 `S3`로 설정
   - `logType` 멤버를 `AUDIO`로 설정
   - `resourceArn` 멤버를 오디오 로그가 저장된 Amazon S3 버킷의 ARN으로 설정
   - 선택 사항. 특정 AWS KMS 키로 오디오 로그를 암호화하려면 암호화에 사용되는 키의 ARN `kmsKeyArn` 멤버를 설정합니다.
2. `conversationLogs` 필드의 `iamRoleArn` 멤버를 IAM 역할(특정한 리소스의 대화 로그를 활성화할 수 있는 필수 권한을 가진)의 Amazon 리소스 이름(ARN)으로 설정합니다.

대화 로그 비활성화

콘솔을 사용하여 로그를 해제하려면

1. https://console.aws.amazon.com/lex에서 Amazon Lex 콘솔을 엽니다.
2. 목록에서 봇을 선택합니다.
3. 설정 탭을 선택한 다음 왼쪽 메뉴에서 대화 로그를 선택합니다.
4. 별칭 목록에서 대화 로그를 구성할 별칭의 설정 아이콘을 선택합니다.
5. 로깅을 해제하려면 텍스트, 오디오 또는 둘 다의 선택을 취소합니다.
6. 대화 로깅을 중지하려면 저장을 선택합니다.

API를 사용하여 로그를 해제하려면

- `conversationLogs` 필드 없이 `PutBotAlias` 작업을 호출합니다.

API를 사용하여 텍스트 로그를 해제하려면

- - 오디오를 로깅하는 경우
    - `AUDIO`에 대한 `logSettings` 항목만 사용하여 PutBotAlias 작업을 호출합니다.
    - `PutBotAlias` 작업 호출에는 `TEXT`에 대한 `logSettings` 항목이 없어야 합니다.
  - 오디오를 로깅하지 않는 경우
    - `conversationLogs` 필드 없이 PutBotAlias 작업을 호출합니다.

API를 사용하여 오디오 로그를 해제하려면

- - 텍스트를 로깅하는 경우
    - `TEXT`에 대한 `logSettings`항목만 사용하여 PutBotAlias 작업을 호출합니다.
    - `PutBotAlias` 작업 호출에는 `AUDIO`에 대한 `logSettings` 항목이 없어야 합니다.
  - 텍스트를 로깅하지 않는 경우
    - `conversationLogs` 필드 없이 PutBotAlias 작업을 호출합니다.

대화 로그 암호화

암호화를 사용하여 대화 로그의 내용을 보호할 수 있습니다. 텍스트 및 오디오 로그의 경우 AWS KMS 고객 관리형 CMKs 사용하여 CloudWatch Logs 로그 그룹 및 S3 버킷의 데이터를 암호화할 수 있습니다.

> **Note**
>
> Amazon Lex는 대칭 CMK만 지원합니다. 비대칭 CMK를 사용하여 데이터를 암호화하지 마십시오.

Amazon Lex가 텍스트 로그에 사용하는 CloudWatch Logs 로그 그룹에서 AWS KMS 키를 사용하여 암호화를 활성화합니다. 로그 그룹의 AWS KMS 암호화를 활성화하기 위해 로그 설정에 AWS KMS 키를 제공할 수 없습니다. 자세한 내용은 Amazon CloudWatch Logs 사용자 가이드의 Encrypt Log Data in CloudWatch Logs Using AWS KMS를 참조하세요.

오디오 로그의 경우 S3 버킷에서 기본 암호화를 사용하거나 AWS KMS 키를 지정하여 오디오 객체를 암호화합니다. S3 버킷이 기본 암호화를 사용하더라도 다른 AWS KMS 키를 지정하여 오디오 객체를 암호화할 수 있습니다. 자세한 내용은 Amazon Simple Storage Service 개발자 가이드의 S3 버킷에 대한 Amazon S3 기본 암호화를 참조하세요.

오디오 로그를 암호화하도록 선택한 경우 Amazon Lex에 AWS KMS 권한이 필요합니다. 대화 로그에 사용되는 IAM 역할에 추가 정책을 연결해야 합니다. S3 버킷에서 기본 암호화를 사용하는 경우 정책은 해당 버킷에 대해 구성된 AWS KMS 키에 대한 액세스 권한을 부여해야 합니다. 오디오 로그 설정에서 AWS KMS 키를 지정하는 경우는 해당 키에 대한 액세스 권한을 부여해야 합니다.

대화 로그의 역할을 만들지 않은 경우 대화 로그에 대한 IAM 정책 섹션을 참조하십시오.

오디오 로그 암호화에 AWS KMS 키를 사용하기 위한 IAM 정책을 생성하려면

1. **`LexConversationLogsKMSPolicy.json`**이라는 현재 디렉터리에 문서를 만들고 다음의 정책을 추가한 후 저장합니다.

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

```
            "Action": [
                "kms:GenerateDataKey"
            ],
            "Resource": "kms-key-arn"
        }
    ]
}
```

2. 에서 오디오 로그 암호화에 AWS KMS 키를 사용할 수 있는 권한을 부여하는 IAM 정책을 AWS CLI생성합니다.

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

3. 대화 로그에 대해 생성한 역할에 정책을 연결합니다.

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

Amazon CloudWatch 로그에서 텍스트 로그 보기

Amazon Lex는 Amazon CloudWatch Logs 의 사용자 대화에 대한 텍스트 로그를 저장합니다. 로그를 보려면 CloudWatch Logs 콘솔이나 API를 사용하십시오. 자세한 내용은 필터 패턴을 사용한 로그 데이터 검색 및 Amazon CloudWatch Logs 사용자 가이드의 CloudWatch Logs Insights 쿼리 구문을 참조하세요.

Amazon Lex 콘솔을 사용하여 로그를 보려면

1. https://console.aws.amazon.com/lex에서 Amazon Lex 콘솔을 엽니다.
2. 목록에서 봇을 선택합니다.
3. 설정 탭을 선택한 다음 왼쪽 메뉴에서 대화 로그를 선택합니다.
4. 텍스트 로그 아래의 링크를 선택하여 CloudWatch 콘솔에 있는 별칭에 대한 로그를 봅니다.

CloudWatch 콘솔이나 API를 사용하여 로그 항목을 볼 수도 있습니다. 로그 항목을 찾으려면 별칭에 대해 구성한 로그 그룹으로 이동합니다. Amazon Lex 콘솔 또는 GetBotAlias 작업을 사용하여 로그의 로그 스트림 접두사를 찾습니다.

사용자 표현에 대한 로그 항목은 여러 로그 스트림 에 있습니다. 대화의 표현에는 지정된 접두사가 있는 로그 스트림 중 하나의 항목이 있습니다. 로그 스트림 의 항목에는 다음 정보가 있습니다.

```
{
   "messageVersion": "1.0",
   "botName": "bot name",
   "botAlias": "bot alias",
   "botVersion": "bot version",
   "inputTranscript": "text used to process the request",
   "botResponse": "response from the bot",
   "intent": "matched intent",
   "nluIntentConfidence": "number",
   "slots": {
       "slot name": "slot value",
       "slot name": null,
       "slot name": "slot value"
       ...
   },
   "alternativeIntents": [
       {
           "name": "intent name",
           "nluIntentConfidence": "number",
           "slots": {
               "slot name": slot value,
               "slot name": null,
               "slot name": slot value
               ...
           }
       },
       {
           "name": "intent name",
           "nluIntentConfidence": number,
           "slots": {}
       }
   ],
   "developerOverride": "true" | "false",
   "missedUtterance": true | false,
   "inputDialogMode": "Text" | "Speech",
   "requestId": "request ID",
   "s3PathForAudio": "S3 path to audio file",
   "userId": "user ID",
   "sessionId": "session ID",
   "sentimentResponse": {
```

```
      "sentimentScore": "{Positive: number, Negative: number, Neutral: number,
Mixed: number}",
      "sentimentLabel": "Positive" | "Negative" | "Neutral" | "Mixed"
  },
  "slotToElicit": "slot name",
  "dialogState": "ElicitIntent" | "ConfirmIntent" | "ElicitSlot" | "Fulfilled" |
"ReadyForFulfillment" | "Failed",
  "responseCard": {
      "genericAttachments": [
          ...
      ],
      "contentType": "application/vnd.amazonaws.card.generic",
      "version": 1
   },
  "locale": "locale",
  "timestamp": "ISO 8601 UTC timestamp",
  "kendraResponse": {
     "totalNumberOfResults": number,
     "resultItems": [
         {
             "id": "query ID",
             "type": "DOCUMENT" | "QUESTION_ANSWER" | "ANSWER",
             "additionalAttributes": [
                 {
                    ...
                 }
             ],
             "documentId": "document ID",
             "documentTitle": {
                 "text": "title",
                 "highlights": null
             },
             "documentExcerpt": {
                 "text": "text",
                 "highlights": [
                     {
                         "beginOffset": number,
                         "endOffset": number,
                         "topAnswer": true | false
                     }
                 ]
             },
             "documentURI": "URI",
             "documentAttributes": []
```

```
            }
        ],
        "facetResults": [],
        "sdkResponseMetadata": {
            "requestId": "request ID"
        },
        "sdkHttpMetadata": {
            "httpHeaders": {
                "Content-Length": "number",
                "Content-Type": "application/x-amz-json-1.1",
                "Date": "date and time",
                "x-amzn-RequestId": "request ID"
            },
            "httpStatusCode": 200
        },
        "queryId": "query ID"
    },
    "sessionAttributes": {
        "attribute name": "attribute value"
        ...
     },
    "requestAttributes": {
        "attribute name": "attribute value"
        ...
     }
}
```

로그 항목의 내용은 거래 결과와 봇 및 요청 구성에 따라 다릅니다.

- `intent`, `slots`, `slotToElicit` 필드는 `missedUtterance` 필드가 `true`인 경우 항목에 나타나지 않습니다.
- 오디오 로그가 비활성화되어 있거나 `inputDialogMode` 필드가 `Text`인 경우 `s3PathForAudio` 필드가 나타나지 않습니다.
- `responseCard` 필드는 봇에 대한 응답 카드를 지정한 경우에만 나타납니다.
- `requestAttributes` 맵은 요청에 지정된 속성이 있는 경우에만 나타납니다.
- 이 `kendraResponse`필드는 `AMAZON.KendraSearchIntent`가 Amazon Kendra 인덱스 검색을 요청할 때만 표시됩니다.
- 봇의 Lambda 함수에 대체 의도가 지정된 경우 이 `developerOverride` 필드는 참입니다.
- `sessionAttributes` 맵은 요청에 지정된 세션 속성이 있는 경우에만 나타납니다.
- `sentimentResponse` 맵은 사용자가 봇을 구성해 감정 값을 반환할 경우에만 나타납니다.

> **Note**
>
> `messageVersion`에서 해당 내용을 변경하지 않아도 입력 형식을 변경할 수 있습니다. 새 필드가 있는 경우 코드에서 오류가 발생하면 안 됩니다.

Amazon Lex가 CloudWatch 로그에 쓸 수 있도록 설정된 역할과 정책이 있어야 합니다. 자세한 내용은 대화 로그에 대한 IAM 정책을 참조하세요.

Amazon S3에서 오디오 로그에 액세스

Amazon Lex는 S3 버킷에 대화에 대한 오디오 로그를 보관합니다.

콘솔을 사용하여 오디오 로그에 액세스하려면

1. https://console.aws.amazon.com/lex에서 Amazon Lex 콘솔을 엽니다.
2. 목록에서 봇을 선택합니다.
3. 설정 탭을 선택한 다음 왼쪽 메뉴에서 대화 로그를 선택합니다.
4. 오디오 로그 아래의 링크를 선택하여 Amazon S3 콘솔에 있는 별칭에 대한 로그에 액세스합니다.

Amazon S3 콘솔 또는 API를 사용하여 오디오 로그에 액세스할 수도 있습니다. Amazon Lex 콘솔 또는 `GetBotAlias` 작업 응답의 `resourcePrefix` 필드에서 오디오 파일의 S3 객체 키 접두사를 볼 수 있습니다.

CloudWatch 지표를 통해 대화 로그 상태 모니터링

Amazon CloudWatch 를 사용하여 대화 로그의 전달 지표를 모니터링합니다. 로깅 문제가 발생할 경우 이를 인식할 수 있도록 지표에 경보를 설정할 수 있습니다.

Amazon Lex에서는 대화 로그에 대한 `AWS/Lex` 네임스페이스에 다음과 같은 4개의 지표를 제공합니다.

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

자세한 내용은 대화 로그에 대한 CloudWatch 지표을 참조하세요.

성공 지표는 Amazon Lex가 대상에 오디오 또는 텍스트 로그를 성공적으로 작성했음을 보여줍니다.

실패 지표는 Amazon Lex가 지정된 대상에 오디오 또는 텍스트 로그를 전달할 수 없음을 보여줍니다. 일반적으로 이는 구성 오류입니다. 실패 지표가 0보다 높으면 다음을 확인하세요.

- Amazon Lex가 IAM 역할에 대한 신뢰할 수 있는 엔터티인지 확인합니다.
- 텍스트 로깅의 경우 CloudWatch Logs 로그 그룹이 있는지 확인합니다. 오디오 로깅의 경우 S3 버킷이 있는지 확인합니다.
- Amazon Lex가 CloudWatch 로그 그룹 또는 S3 버킷에 액세스하는 데 사용하는 IAM 역할에 로그 그룹 또는 버킷에 대한 쓰기 권한이 있는지 확인합니다.
- S3 버킷이 Amazon Lex 봇과 동일한 리전에 존재하며, 사용자 계정에 속하는지 확인합니다.
- S3 암호화에 AWS KMS 키를 사용하는 경우 Amazon Lex가 키를 사용하지 못하게 하는 정책이 없는지 확인하고 제공한 IAM 역할에 필요한 AWS KMS 권한이 있는지 확인합니다. 자세한 내용은 대화 로그에 대한 IAM 정책 단원을 참조하십시오.

Amazon Lex API로 세션 관리

사용자가 봇과 대화를 시작하면 Amazon Lex는 세션을 생성합니다. 애플리케이션과 Amazon Lex 간에 교환된 정보는 대화의 세션 상태를 구성합니다. 요청하면 지정한 봇 이름과 사용자 식별자 조합으로 세션이 식별됩니다. 사용자 식별자에 대한 자세한 내용은 PostContent 또는 PostText 작업의 `userId` 필드를 참조하십시오.

세션 작업의 응답에는 사용자로 특정 세션을 식별하는 고유한 세션 식별자가 포함됩니다. 테스트 중에 또는 봇의 문제 해결을 위해 이 식별자를 사용할 수 있습니다.

애플리케이션과 봇 간에 전송된 세션 상태를 수정할 수 있습니다. 예를 들어 세션에 대한 사용자 정의 정보가 포함된 세션 속성을 생성 및 수정할 수 있으며, 다음 표현을 해석하도록 대화 컨텍스트를 설정하여 대화의 흐름을 변경할 수 있습니다.

세션 상태를 업데이트할 수 있는 방법에는 두 가지가 있습니다. 첫 번째 방법은 대화가 바뀔 때마다 호출되는 `PostContent` 또는 `PostText` 작업과 함께 Lambda 함수를 사용하는 것입니다. 자세한 내용은 Lambda 함수 사용을 참조하세요. 다른 방법은 애플리케이션에서 Amazon Lex 런타임 API를 사용하여 세션 상태를 변경하는 것입니다.

Amazon Lex 런타임 API는 봇과의 대화에 대한 세션 정보를 관리할 수 있도록 하는 작업을 제공합니다. 이 작업은 PutSession 작업, GetSession 작업, DeleteSession 작업입니다. 이러한 작업을 사용하여 봇과의 사용자 세션 상태에 대한 정보를 얻고 상태를 세부적으로 제어할 수 있습니다.

현재 세션 상태를 확인하려는 경우 `GetSession` 작업을 사용합니다. 이 작업은 사용자와의 대화 상태, 설정된 모든 세션 속성, 사용자가 상호 작용한 마지막 세 의도에 대한 슬롯 값을 포함하여 현재 세션 상태를 반환합니다.

`PutSession` 작업을 사용하면 현재 세션 상태를 직접 조작할 수 있습니다. 봇이 다음으로 수행할 대화 작업의 유형을 설정할 수 있습니다. 이를 통해 봇과의 대화 흐름을 제어할 수 있습니다. Amazon Lex가 봇의 다음 작업을 결정하도록 하려면 대화 작업 `type` 필드를 `Delegate`로 설정합니다.

`PutSession` 작업을 사용하여 봇과의 새로운 세션을 만들고 봇이 시작해야 하는 의도를 설정할 수 있습니다. `PutSession` 작업을 사용하여 한 의도에서 다른 의도로 변경할 수도 있습니다. 세션을 만들거나 의도를 변경할 때 슬롯 값 및 세션 속성 등의 세션 상태를 설정할 수도 있습니다. 새 의도를 마치면 이전 의도를 다시 시작할 수 있습니다. `GetSession` 작업을 사용하여 Amazon Lex에서 이전 의도의 대화 상태를 가져오고 이 정보를 사용하여 의도의 대화 상태를 설정할 수 있습니다.

`PutSession` 작업의 응답에는 `PostContent` 작업과 동일한 정보가 포함되어 있습니다. `PostContent` 작업의 응답과 마찬가지로 이 정보를 사용하여 사용자에게 다음 정보를 입력하라는 메시지를 표시할 수 있습니다.

`DeleteSession` 작업을 사용하여 기존 세션을 제거하고 새 세션을 시작합니다. 예를 들어 봇을 테스트하려는 경우 `DeleteSession` 작업을 사용하여 봇에서 테스트 세션을 제거할 수 있습니다.

이 세션 작업은 이행 Lambda 함수와 함께 작동합니다. 예를 들어 Lambda 함수가 이행 상태로 `Failed`를 반환하는 경우, `PutSession` 작업을 사용하여 대화 작업 유형을 `close`로 설정하고 `fulfillmentState`를 `ReadyForFulfillment`로 설정하여 이행 단계를 재시도할 수 있습니다.

다음은 세션 작업으로 수행할 수 있는 작업입니다.

- 봇이 사용자를 기다리지 않고 대화를 시작하도록 합니다.
- 대화 중에 의도를 전환합니다.
- 이전 의도로 돌아갑니다.
- 상호 작용 중에 대화를 시작하거나 다시 시작합니다.
- 슬롯 값의 유효성을 검사하고 봇이 유효하지 않은 값을 다시 묻도록 합니다.

이러한 각 내용은 아래에 자세히 설명되어 있습니다.

의도 전환

PutSession 작업을 사용하여 한 의도에서 다른 의도로 전환할 수 있습니다. 이 작업을 사용하여 이전 의도로 다시 전환할 수도 있습니다. PutSession 작업을 사용하여 새 의도의 슬롯 값 또는 세션 속성을 설정할 수 있습니다.

- PutSession 작업을 직접적으로 호출합니다. 의도 이름을 새 의도의 이름으로 설정하고 대화 작업을 Delegate로 설정합니다. 새 의도에 필요한 슬롯 값 또는 세션 속성을 설정할 수도 있습니다.
- Amazon Lex는 새 의도를 사용하여 사용자와의 대화를 시작합니다.

이전 의도 다시 시작

이전 의도를 다시 시작하려면 GetSession 작업을 사용하여 의도의 요약을 가져온 후, PutSession 작업을 사용하여 의도를 이전 대화 상태로 설정합니다.

- GetSession 작업을 직접적으로 호출합니다. 작업의 응답에는 사용자가 상호 작용한 마지막 세 의도의 대화 상태에 대한 요약이 포함되어 있습니다.
- 의도 요약의 정보를 사용하여 PutSession 작업을 호출합니다. 그러면 사용자가 대화의 같은 위치에 있는 이전 의도로 돌아갑니다.

경우에 따라 봇과 사용자의 대화를 다시 시작해야 할 수 있습니다. 예를 들어 고객 서비스 봇을 만들었다고 가정하겠습니다. 애플리케이션은 사용자가 고객 서비스 담당자와 대화해야 한다고 결정합니다. 사용자와 대화한 후 이 담당자는 수집한 정보와 함께 대화를 다시 봇에 전달할 수 있습니다.

세션을 다시 시작하려면 다음과 비슷한 단계를 사용합니다.

- 애플리케이션은 사용자가 고객 서비스 담당자와 대화해야 한다고 결정합니다.
- GetSession 작업을 사용하여 의도의 현재 대화 상태를 가져옵니다.
- 고객 서비스 담당자는 사용자와 대화하고 문제를 해결합니다.
- PutSession 작업을 사용하여 의도의 대화 상태를 설정합니다. 여기에는 슬롯 값 설정, 세션 속성 설정 또는 의도 변경이 포함될 수 있습니다.
- 봇이 사용자와의 대화를 다시 시작합니다.

나중에 찾을 수 있도록 PutSession 작업 checkpointLabel 파라미터를 사용하여 의도에 레이블을 지정할 수 있습니다. 예를 들어, 고객에게 정보를 요청하는 봇은 고객이 정보를 수집하는 동안 Waiting 의도로 들어갈 수 있습니다. 이 봇은 현재 의도에 대한 체크포인트 레이블을 만든 다음

Waiting 의도를 시작합니다. 고객이 돌아오면 봇은 체크포인트 레이블을 사용하여 이전 의도를 찾아 다시 전환할 수 있습니다.

의도는 GetSession 작업에 의해 반환된 recentIntentSummaryView 구조에 있어야 합니다. GetSession 작업 요청에 체크포인트 레이블을 지정하는 경우 해당 체크포인트 레이블이 있는 의도를 최대 3개까지 반환합니다.

- GetSession 작업을 사용하여 현재 세션 상태를 가져옵니다.
- PutSession 작업을 사용하여 마지막 의도에 체크포인트 레이블을 추가합니다. 필요한 경우 이 PutSession 호출을 사용하여 다른 의도로 전환할 수 있습니다.
- 레이블이 지정된 의도로 다시 전환해야 하는 경우 GetSession 작업을 호출하여 최근 의도 목록을 반환합니다. Amazon Lex에서 지정된 체크포인트 레이블이 있는 의도만 반환하도록 checkpointLabelFilter 파라미터를 사용할 수 있습니다.

새 세션 시작

봇이 사용자와 대화를 시작하도록 하려면 PutSession 작업을 사용하면 됩니다.

- 슬롯이 없는 환영 의도와 사용자에게 의도를 명시하라는 결론 메시지를 만듭니다. 예를 들어 "무엇을 주문하시겠어요? '음료 주문' 또는 '피자 주문.'이라고 말할 수 있습니다.
- PutSession 작업을 직접적으로 호출합니다. 의도 이름을 환영 의도의 이름으로 설정하고 대화 작업을 Delegate로 설정합니다.
- Amazon Lex는 환영 의도의 프롬프트에 응답하여 사용자와의 대화를 시작합니다.

슬롯 값 유효성 검사

클라이언트 애플리케이션을 사용하여 봇에 대한 응답의 유효성을 검사할 수 있습니다. 응답이 유효하지 않은 경우 PutSession 작업을 사용하여 사용자로부터 새 응답을 얻을 수 있습니다. 예를 들어 꽃 주문 봇이 튤립, 장미, 백합만 팔 수 있다고 가정하겠습니다. 사용자가 카네이션을 주문하면 애플리케이션이 다음을 수행할 수 있습니다.

- PostText 또는 PostContent 응답에서 반환된 슬롯 값을 검사합니다.
- 슬롯 값이 유효하지 않으면 PutSession 작업을 직접적으로 호출합니다. 애플리케이션이 슬롯 값을 지우고, slotToElicit 필드를 설정하고, dialogAction.type 값을 elicitSlot으로 설정합니다. 선택적으로, Amazon Lex가 슬롯 값을 유도하기 위해 사용하는 메시지를 변경하려는 경우 message 및 messageFormat 필드를 설정할 수 있습니다.

봇 배포 옵션

현재 Amazon Lex 는 다음의 봇 배포 옵션을 제공합니다.

- AWS Mobile SDK - AWS Mobile SDK 를 사용하여 Amazon Lex와 통신하는 모바일 애플리케이션을 구축할 수 있습니다.
- Facebook Messenger – Facebook Messenger 페이지를 Amazon Lex 봇과 통합하여 Facebook에서 최종 사용자가 봇과 통신하도록 할 수 있습니다. 현재 구현에서 이 통합은 텍스트 입력 메시지만 지원합니다.
- Slack – Slack 메시징 애플리케이션과 Amazon Lex 봇을 통합할 수 있습니다.
- Twilio – Twilio의 SMS(Simple Messaging Service)와 Amazon Lex 봇을 통합할 수 있습니다.

예를 보려면 Amazon Lex 봇 배포을 참조하세요.

기본 제공 의도 및 슬롯 유형

봇을 더 쉽게 생성할 수 있도록 Amazon Lex에서는 표준 기본 제공 의도 및 슬롯 유형을 사용할 수 있습니다.

주제

- 기본 제공 의도
- 기본 제공 슬롯 유형

기본 제공 의도

일반적인 작업에 표준 기본 제공 의도 라이브러리를 사용할 수 있습니다. 기본 제공 의도에서 의도를 생성하려면, 콘솔에서 기본 제공 의도를 선택한 후 해당 기본 제공 의도에 새 이름을 지정합니다. 새 의도에는 샘플 표현 등 기본 의도의 구성이 포함되어 있습니다.

현재 구현에서는 다음 작업을 수행할 수 없습니다.

- 기본 의도에서 샘플 표현 추가 또는 제거
- 기본 제공 의도의 슬롯 구성

봇에 기본 제공 의도를 추가하려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lex/ Amazon Lex 콘솔을 엽니다.
2. 기본 제공 의도를 추가할 봇을 선택합니다.
3. 탐색 창에서 의도 옆에 있는 더하기(+) 기호를 선택합니다.
4. 의도 추가에서 기존 의도 검색을 선택합니다.
5. 검색 의도 상자에 봇에 추가할 기본 제공 의도의 이름을 입력합니다.
6. 기본 제공 의도 복사에서 의도 이름을 입력한 다음 추가를 선택합니다.
7. 봇에 필요한 대로 의도를 구성하세요.

주제

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

> **Note**
>
> 영어(미국)(en-US) 로캘의 경우 Amazon Lex는 Alexa 표준 기본 제공 의도로부터 의도를 지원합니다. 기본 제공 의도 목록은 Alexa Skills Kit의 표준 기본 제공 의도를 참조하십시오. Amazon Lex는 다음과 같은 의도는 지원하지 않습니다.
>
> - `AMAZON.YesIntent`
> - `AMAZON.NoIntent`
> - Alexa Skills Kit의 기본 제공 의도 라이브러리에 있는 의도

AMAZON.CancelIntent

사용자가 현재 상호 작용을 취소하기를 원한다는 것을 나타내는 단어와 문구에 응답합니다. 애플리케이션은 이 의도를 사용하여 사용자와의 상호작용을 종료하기 전에 슬롯 유형 값 및 기타 속성을 제거할 수 있습니다.

공통 표현:

- 취소
- 신경 쓰지 마
- 잊어버려

AMAZON.FallbackIntent

사용자의 의도 입력이 봇의 예상과 다를 경우 Amazon Lex가 폴백 의도를 호출하도록 구성할 수 있습니다. 예를 들어 사용자 입력 "캔디를 주문하고 싶어"가 봇의 `OrderFlowers` 의도와 맞지 않는 경우 Amazon Lex는 응답 처리를 위해 폴백 의도를 호출합니다.

기본 제공 `AMAZON.FallbackIntent` 의도 유형을 봇에 추가하여 폴백 의도를 추가할 수 있습니다. PutBot 작업을 사용하거나 콘솔의 기본 제공 의도 목록에서 의도를 선택하여 의도를 지정할 수 있습니다.

폴백 의도 간접 호출은 두 단계로 진행됩니다. 첫 번째 단계에서 폴백 의도는 사용자의 입력을 기반으로 매칭됩니다. 폴백 의도가 일치할 경우 봇이 작동하는 방식은 프롬프트에 설정된 재시도 수에 따라 다릅니다. 예를 들어 의도를 판단하는 최대 시도 횟수가 2라면 봇은 폴백 의도를 호출하기 전에 봇의 확인 프롬프트를 두 번 반환합니다.

Amazon Lex가 폴백 의도와 일치하는 경우는 다음과 같습니다.

- 의도에 대한 사용자 입력이 봇이 예상한 입력과 같지 않습니다.
- 음성 입력에 노이즈가 있거나, 텍스트 입력이 단어로 인식되지 않습니다.
- 사용자 입력이 모호하여 Amazon Lex에서 호출할 의도를 판단할 수 없습니다.

폴백 의도가 간접적으로 호출되는 시점은 다음과 같습니다.

- 대화가 시작될 때 명확히 하기 위해 설정된 시도 횟수를 넘긴 후에도 봇이 사용자 입력을 의도로 인식하지 않는 경우
- 설정된 시도 횟수를 넘긴 후에도 의도가 사용자 입력을 슬롯 값으로 인식하지 않는 경우

- 설정된 시도 횟수를 넘긴 후에도 의도가 사용자 입력을 확인 프롬프트에 대한 응답으로 인식하지 않는 경우

폴백 의도에 사용할 수 있는 항목은 다음과 같습니다.

- Lambda 함수 이행
- 결론문
- 후속 프롬프트

다음은 폴백 의도에 추가할 수 없습니다.

- 표현
- 슬롯
- 초기화 및 검증 Lambda 함수
- 확인 프롬프트

봇의 취소문과 폴백 의도를 모두 구성한 경우 Amazon Lex가 폴백 의도를 사용합니다. 봇에 취소문이 있어야 한다면 폴백 의도에 대한 이행 함수를 사용하여 취소문과 동일한 동작을 제공할 수 있습니다. 자세한 내용은 PutBot 작업의 `abortStatement` 파라미터를 참조하십시오.

확인 프롬프트 사용

봇에 확인 프롬프트를 제공하면 사용자의 유효한 의도를 얻을 수 있도록 해당 프롬프트가 사용됩니다. 확인 프롬프트는 설정해 놓은 횟수를 반복합니다. 그 이후에 폴백 의도가 호출됩니다.

봇을 만들 때 확인 프롬프트를 설정하지 않았고, 사용자가 올바른 의도로 대화를 시작하지 않는다면 Amazon Lex는 폴백 의도를 즉시 호출합니다.

확인 프롬프트를 사용하지 않고 폴백 의도를 사용한다면 Amazon Lex는 다음 상황에서 폴백을 호출하지 않습니다.

- 사용자가 후속 프롬프트에 응답하지만, 의도를 제공하지 않는 경우 예를 들어, "오늘 다른 것을 원하시나요?"라는 후속 프롬프트에 대한 응답에 대해, 사용자는 "네"라고 대답합니다. Amazon Lex에 사용자의 의도를 파악하기 위한 확인 프롬프트가 없기 때문에 400 잘못된 요청 예외가 반환됩니다.
- AWS Lambda 함수를 사용할 때 `ElicitIntent` 대화 유형을 반환합니다. Amazon Lex에 사용자의 의도를 파악하기 위한 확인 프롬프트가 없기 때문에 400 잘못된 요청 예외가 반환됩니다.

- `PutSession` 작업을 사용 중인 경우 `ElicitIntent` 대화 유형을 전송합니다. Amazon Lex에 사용자의 의도를 파악하기 위한 확인 프롬프트가 없기 때문에 400 잘못된 요청 예외가 반환됩니다.

폴백 의도에 Lambda 함수 사용

폴백 의도가 호출되면 응답은 PutIntent 작업에 대한 `fulfillmentActivity` 파라미터 설정에 따라 달라집니다. 봇은 다음 중 하나를 수행합니다.

- 의도 정보를 클라이언트 애플리케이션에 반환합니다.
- 이행 Lambda 함수를 호출합니다. 세션에 설정된 세션 변수로 함수를 직접적으로 호출합니다.

폴백 의도가 호출될 때 응답을 설정하는 작업에 대한 자세한 내용은 PutIntent 작업의 `fulfillmentActivity` 파라미터를 참조하십시오.

폴백 의도에서 이행 Lambda 함수를 사용하는 경우에는 이 함수로 다른 의도를 호출할 수 있습니다. 회신 번호를 수집하거나 고객 서비스 담당자와의 세션을 개설하는 사용자와 일종의 커뮤니케이션을 수행할 수도 있습니다.

이행 함수에서 다른 의도에 대해 수행할 수 있는 모든 작업을 폴백 의도 Lambda 함수에서 수행할 수 있습니다. 를 사용하여 이행 함수를 생성하는 방법에 대한 자세한 내용은 섹션을 AWS Lambda참조하세요Lambda 함수 사용.

세션이 동일하면 폴백 의도는 여러 번 간접적으로 호출할 수 있습니다. 예를 들어 Lambda 함수가 `ElicitIntent` 대화 작업을 사용하여 사용자에게 다른 의도를 묻는 프롬프트를 표시한다고 해 보겠습니다. Amazon Lex가 설정된 시도 횟수 이후에 사용자의 의도를 추론할 수 없다면 폴백 의도를 다시 호출합니다. 또한 사용자가 구성된 시도 횟수 후 올바른 슬롯 값으로 응답하지 않을 때도 폴백 의도를 간접적으로 호출합니다.

세션 변수를 사용하여 폴백 의도를 호출하는 횟수를 추적하도록 Lambda 함수를 구성할 수 있습니다. Lambda 함수는 Lambda 함수에 설정한 임계값보다 더 많이 호출될 경우 다른 작업을 수행할 수 있습니다. 세션 변수에 대한 자세한 내용은 Setting Session Attributes 단원을 참조하세요.

AMAZON.HelpIntent

사용자가 봇과 상호작용하는 동안 도움이 필요함을 나타내는 단어나 문구에 응답합니다. 이 의도가 간접적으로 호출되면 Lambda 함수 또는 애플리케이션을 구성하여 봇 기능에 대한 정보를 제공하고, 도움이 필요한 영역에 대한 후속 질문을 하거나, 상담원에게 상호 작용을 넘겨줄 수 있습니다.

공통 표현:

- 도움
- 도와줘
- 나 좀 도와줄래?

AMAZON.KendraSearchIntent

Amazon Kendra로 인덱싱된 문서를 검색하려면 `AMAZON.KendraSearchIntent` 의도를 사용합니다. Amazon Lex가 사용자와의 대화에서 다음 작업을 결정할 수 없으면 검색 의도가 트리거됩니다.

`AMAZON.KendraSearchIntent`는 영어(미국)(en-US) 및 미국 동부(버지니아 북부), 미국 서부(오레곤) 및 유럽(아일랜드) 리전만 사용 가능합니다.

Amazon Kendra는 PDF 문서 또는 Microsoft Word 파일과 같은 자연어 문서를 인덱싱하는 기계 학습 기반 검색 서비스입니다. 인덱싱된 문서를 검색하고 질문에 대해 다음 유형의 응답을 반환할 수 있습니다.

- 대답
- 질문에 대한 답이 될 수 있는 FAQ 항목
- 질문과 관련된 문서

`AMAZON.KendraSearchIntent` 사용 예는 예제: Amazon Kendra 인덱스에 대한 FAQ 봇 생성 섹션을 참조하십시오.

봇에 `AMAZON.KendraSearchIntent` 의도를 구성하면 Amazon Lex가 슬롯 또는 의도에 대한 사용자 표현을 파악할 수 없을 때마다 이 의도를 호출합니다. 예를 들어 봇이 "피자 토핑"이라는 슬롯 유형에 대한 응답을 유도할 때 사용자가 "피자가 뭐야?"라고 말할 경우 Amazon Lex는 이 질문을 처리하기 위해 `AMAZON.KendraSearchIntent`를 호출합니다. Amazon Kendra에서 응답이 없으면 봇에 구성된 대로 대화가 계속됩니다.

동일한 봇에서 `AMAZON.KendraSearchIntent`와 `AMAZON.FallbackIntent`가 모두 사용되는 경우 Amazon Lex는 이러한 의도를 다음과 같이 사용합니다.

1. Amazon Lex가 `AMAZON.KendraSearchIntent`을 호출합니다. 이 의도는 Amazon Kendra `Query` 작업을 호출합니다.
2. Amazon Kendra에서 응답을 반환하면 Amazon Lex가 사용자에게 결과를 표시합니다.
3. Amazon Kendra에서 응답이 없으면 Amazon Lex가 사용자에게 다시 메시지를 표시합니다. 다음 작업은 사용자의 응답에 따라 달라집니다.

- 사용자의 응답에 슬롯 값을 채우거나 의도를 확인하는 것과 같이 Amazon Lex에서 인식하는 표현이 포함된 경우 봇에 구성된 대로 사용자와의 대화가 진행됩니다.
- 사용자의 응답에 Amazon Lex에서 인식하는 표현이 포함되어 있지 않으면 Amazon Lex가 새로운 `Query` 작업을 호출합니다.

4. 구성된 재시도 횟수 이후에 응답이 없으면 Amazon Lex가 `AMAZON.FallbackIntent`를 호출하여 사용자와의 대화를 종료합니다.

`AMAZON.KendraSearchIntent`를 사용하여 Amazon Kendra에 요청을 하는 방법에는 다음 3가지가 있습니다.

- 의도 검색이 대신 요청하도록 하세요. Amazon Lex는 사용자의 말을 검색 문자열로 사용하여 Amazon Kendra를 호출합니다. 의도를 생성할 때 Amazon Kendra에서 반환되는 응답 수를 제한하는 쿼리 필터 문자열을 정의할 수 있습니다. Amazon Lex는 쿼리 요청에서 필터를 사용합니다.
- 대화 Lambda 함수를 사용하여 요청에 추가 쿼리 파라미터 추가하세요. `delegate` 대화 작업에 Amazon Kendra 쿼리 파라미터가 포함된 `kendraQueryFilterString` 필드를 추가합니다. Lambda 함수를 사용하여 요청에 쿼리 파라미터를 추가하면 해당 파라미터가 의도를 생성할 때 정의한 쿼리 필터보다 우선 적용됩니다.
- 대화 Lambda 함수를 사용하여 새 쿼리 생성. Amazon Lex가 보내는 전체 Amazon Kendra 쿼리 요청을 생성할 수 있습니다. `delegate` 대화 작업의 `kendraQueryRequestPayload` 필드에 쿼리를 지정합니다. `kendraQueryRequestPayload` 필드가 `kendraQueryFilterString` 필드보다 우선 적용됩니다.

봇을 생성할 때 `queryFilterString` 파라미터를 지정하거나 대화 Lambda 함수에서 `delegate` 작업을 직접적으로 호출할 때 `kendraQueryFilterString` 필드를 지정하려면 Amazon Kendra 쿼리에 대한 속성 필터로 사용되는 문자열을 지정합니다. 문자열이 유효한 속성 필터가 아닌 경우 런타임에 `InvalidBotConfigException` 예외가 발생합니다. 속성 필터에 대한 자세한 내용은 Amazon Kendra 개발자 안내서의 문서 속성을 사용하여 쿼리 필터링을 참조하십시오.

Amazon Lex가 Amazon Kendra에 보내는 쿼리를 제어하려면 대화 Lambda 함수의 `kendraQueryRequestPayload` 필드에 쿼리를 지정합니다. 쿼리가 유효하지 않으면 Amazon Lex에서 `InvalidLambdaResponseException` 예외를 반환합니다. 자세한 내용은 Amazon Kendra 개발자 안내서의 쿼리 작업을 참조하세요.

`AMAZON.KendraSearchIntent` 사용 방법의 예는 예제: Amazon Kendra 인덱스에 대한 FAQ 봇 생성 단원을 참조하세요.

Amazon Kendra 검색에 사용되는 IAM 정책

`AMAZON.KendraSearchIntent` 의도를 사용하려면 Amazon Lex가 Amazon Kendra `Query` 의도를 호출할 권한이 있는 런타임 역할을 수임할 수 있도록 하는 AWS Identity and Access Management (IAM) 정책을 제공하는 역할을 사용해야 합니다. 사용하는 IAM 설정은 Amazon Lex 콘솔을 `AMAZON.KendraSearchIntent` 사용하여를 생성하는지 아니면 AWS SDK 또는 AWS Command Line Interface ()를 사용하여를 생성하는지에 따라 달라집니다AWS CLI. 콘솔을 사용하는 경우 Amazon Lex 서비스 연결 역할에 Amazon Kendra 호출 권한을 추가하거나, Amazon Kendra `Query` 작업 호출을 위한 전용 역할을 사용할 수 있습니다. AWS CLI 또는 SDK를 사용하여 의도를 생성할 때는 `Query` 작업을 호출하는 데 특별히 역할을 사용해야 합니다.

권한 연결

Amazon Kendra 콘솔을 사용하여 `Query` 작업에 액세스할 수 있는 권한을 기본 Amazon Lex 서비스 연결 역할에 연결할 수 있습니다. 서비스 연결 역할에 권한을 연결하면 Amazon Kendra 인덱스에 연결하기 위한 전용 런타임 역할을 생성하고 관리할 필요가 없습니다.

Amazon Lex 콘솔에 액세스하는 데 사용하는 사용자, 역할 또는 그룹에는 역할 정책을 관리할 수 있는 권한이 있어야 합니다. 콘솔 액세스 역할에 다음 IAM 정책을 연결합니다. 이러한 권한을 부여하면 해당 역할이 기존 서비스 연결 역할 정책을 변경할 수 있는 권한을 갖게 됩니다.

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

역할 지정

콘솔, AWS CLI또는 API를 사용하여 Amazon Kendra `Query` 작업을 호출할 때 사용할 런타임 역할을 지정할 수 있습니다.

런타임 역할을 지정하는 데 사용하는 사용자, 역할 또는 그룹에는 `iam:PassRole` 권한이 있어야 합니다. 다음 정책은 권한을 정의합니다. `iam:AssociatedResourceArn` 및 `iam:PassedToService` 조건 컨텍스트 키를 사용해 권한 범위를 추가로 제한할 수 있습니다. 자세한 내용은 AWS Identity and Access Management 사용 설명서의 IAM 및 AWS STS 조건 컨텍스트 키를 참조하세요.

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

Amazon Lex가 Amazon Kendra를 호출하는 데 사용하는 런타임 역할에는 `kendra:Query` 권한이 있어야 합니다. Amazon Kendra `Query` 작업을 호출할 수 있는 권한을 위해 기존 IAM 역할을 사용하는 경우 역할에 다음 정책이 연결되어 있어야 합니다.

IAM 콘솔, IAM API 또는 AWS CLI 를 사용하여 정책을 생성하고 역할에 연결할 수 있습니다. 여기에 나온 지침에서는 AWS CLI를 사용하여 역할과 정책을 생성합니다.

> **Note**
>
> 다음 코드는 Linux 및 MacOS 용으로 형식이 지정됩니다. Windows의 경우 Linux 줄 연속 문자(\)를 캐럿(^)으로 바꿉니다.

역할에 쿼리 작업 권한을 추가하려면

1. 현재 디렉터리에 **`KendraQueryPolicy.json`**이라는 문서를 만들고 다음 코드를 추가한 다음 저장합니다.

   ```
   {
   ```

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

2. 에서 다음 명령을 AWS CLI실행하여 Amazon Kendra `Query` 작업을 실행하기 위한 IAM 정책을 생성합니다.

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

3. `Query` 작업을 호출하는 데 사용하는 IAM 역할에 정책을 연결합니다.

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

Amazon Lex 서비스 연결 역할을 업데이트하거나 봇에 `AMAZON.KendraSearchIntent`를 만들 때 생성한 역할을 사용하도록 선택할 수 있습니다. 다음 절차에서는 사용할 IAM 역할 을 선택하는 방법을 보여 줍니다.

AMAZON.KendraSearchItent에 대한 런타임 역할을 지정하려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lex/ Amazon Lex 콘솔을 엽니다.
2. `AMAZON.KendraSearchIntent`를 추가할 봇을 선택합니다.
3. 의도 옆에 있는 더하기(+) 기호를 선택합니다.
4. 의도 추가에서 기존 의도 검색을 선택합니다.
5. 검색 의도에 **`AMAZON.KendraSearchIntent`**를 입력한 다음 추가를 선택합니다.

6. 기본 제공 의도 복사에서 의도의 이름(예: **`KendraSearchIntent`**)을 입력한 다음 추가를 선택합니다.
7. Amazon Kendra 쿼리 섹션을 엽니다.
8. IAM 역할에서 다음 옵션 중 하나를 선택합니다.
   - 봇이 Amazon Kendra 인덱스를 쿼리할 수 있도록 Amazon Lex 서비스 연결 역할을 업데이트하려면 Amazon Kendra 권한 추가를 선택합니다.
   - Amazon Kendra `Query` 작업을 호출할 수 있는 권한이 있는 역할을 사용하려면 기존 역할 사용을 선택합니다.

요청 및 세션 속성을 필터로 사용

Amazon Kendra의 응답을 현재 대화와 관련된 항목으로 필터링하려면 봇을 생성할 때 `queryFilterString` 파라미터를 추가하여 세션 및 요청 속성을 필터로 사용합니다. 의도를 생성할 때 속성의 자리 표시자를 지정하면 Amazon Lex V2가 Amazon Kendra를 직접적으로 호출하기 전에 해당 값을 대체합니다. 요청 속성에 대한 자세한 내용은 Setting Request Attributes 단원을 참조하세요. 세션 속성에 대한 자세한 내용은 Setting Session Attributes 단원을 참조하세요.

다음은 Amazon Kendra라는 요청 속성을 사용하여 쿼리를 필터링하는 `queryFilterString` 파라미터의 예입니다.

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

다음은 `"SourceURI"`이라는 세션 속성을 사용하여 Amazon Kendra 쿼리를 필터링하는 `queryFilterString` 파라미터의 예입니다.

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

다음은 `"DepartmentName"`이라는 요청 속성을 사용하여 Amazon Kendra 쿼리를 필터링하는 `queryFilterString` 파라미터의 예입니다.

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

`AMAZON.KendraSearchInteng` 필터는 Amazon Kendra 검색 필터와 동일한 형식을 사용합니다. 자세한 내용은 Amazon Kendra 개발자 안내서의 문서 속성을 사용하여 검색 결과 필터링을 참조하세요.

AMAZON.KendraSearchIntent과 함께 사용되는 쿼리 필터의 문자열은 각 필터의 첫 글자에 소문자를 사용해야 합니다. 예를 들어, 다음은 AMAZON.KendraSearchIntent에 대한 유효한 쿼리 필터입니다.

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

검색 응답 사용

Amazon Kendra는 의도의 conclusion 문에서 검색에 대한 응답을 반환합니다. 이행 Lambda 함수가 결론 메시지를 생성하지 않는 한, 의도에 conclusion 문이 있어야 합니다.

Amazon Kendra에는 네 가지 유형의 응답이 있습니다.

- x-amz-lex:kendra-search-response-question_answer-question-<N> – 검색과 일치하는 FAQ 질문.
- x-amz-lex:kendra-search-response-question_answer-answer-<N> – 검색과 일치하는 FAQ 답변.
- x-amz-lex:kendra-search-response-document-<N> – 표현 텍스트와 관련된 인덱스에 있는 문서의 발췌문.
- x-amz-lex:kendra-search-response-document-link-<N> – 표현 텍스트와 관련된 인덱스에 있는 문서의 URL.

- `x-amz-lex:kendra-search-response-answer-<N>` – 질문에 대한 답이 되는 인덱스에 있는 문서의 발췌문.

응답은 `request` 속성에서 반환됩니다. 각 속성에 1부터 5까지 번호가 매겨진 최대 5개의 응답이 있을 수 있습니다. 서비스 이름 변경에 대한 자세한 내용을 알아보려면 Amazon Kendra 개발자 가이드의 응답 유형을 참조하세요.

`conclusion` 문에는 하나 이상의 메시지 그룹이 있어야 합니다. 각 메시지 그룹에는 하나 이상의 메시지가 포함됩니다. 각 메시지에는 Amazon Kendra의 응답에서 요청 속성으로 대체되는 하나 이상의 자리 표시자 변수가 포함될 수 있습니다. 메시지 그룹에는 해당 메시지의 모든 변수가 런타임 응답에서 요청 속성 값으로 대체되는 메시지가 하나 이상 있어야 합니다. 그렇지 않은 경우 자리 표시자 변수가 없는 메시지 하나가 그룹에 있어야 합니다. 요청 속성은 이중 괄호("((" "))")로 묶입니다. 다음 메시지 그룹 메시지는 Amazon Kendra의 모든 응답과 일치합니다.

- "질문에 대한 FAQ를 찾았습니다: ((x-amz-lex:kendra-search-response-question_answer-question-1)), and the answer is ((x-amz-lex:kendra-search-response-question_answer-answer-1))"
- "I found an excerpt from a helpful document: ((x-amz-lex:kendra-search-response-document-1))"
- "질문에 대한 답변은 다음과 같습니다 ((x-amz-lex:kendra-search-response-answer-1))"

Lambda 함수를 사용하여 요청 및 응답 관리

`AMAZON.KendraSearchIntent` 의도는 대화 코드 후크 및 이행 코드 후크를 사용하여 Amazon Kendra에 대한 요청과 응답을 관리할 수 있습니다. Amazon Kendra에 보내는 쿼리를 수정하려면 대화 코드 후크 Lambda 함수를 사용하고, 응답을 수정하려면 이행 코드 후크 Lambda 함수를 사용합니다.

대화 코드 후크를 사용하여 쿼리 생성

대화 코드 후크를 사용하여 Amazon Kendra에 보낼 쿼리를 생성할 수 있습니다. 대화 코드 후크 사용은 선택 사항입니다. 대화 코드 후크를 지정하지 않으면 Amazon Lex가 사용자 표현으로부터 쿼리를 구성하고 의도 구성 시 제공된(있는 경우) `queryFilterString`를 사용합니다.

Amazon Kendra에 대한 요청을 수정하기 위해 대화 코드 후크 응답에서 다음 두 필드를 사용할 수 있습니다.

- `kendraQueryFilterString` – Amazon Kendra 요청에 대한 속성 필터를 지정하려면 이 문자열을 사용합니다. 인덱스에 정의된 인덱스 필드 중 하나를 사용하여 쿼리를 필터링할 수 있습니다. 필터 문자열의 구조는 Amazon Kendra 개발자 안내서의 문서 속성을 사용하여 쿼리 필터링을 참조하세요. 지정된 필터 문자열이 유효하지 않으면 `InvalidLambdaResponseException` 예외가 발생

합니다. `kendraQueryFilterString` 문자열은 해당 의도에 구성된 `queryFilterString`에 지정되어 있는 모든 쿼리 문자열을 재정의합니다.

- `kendraQueryRequestPayload` – Amazon Kendra 쿼리를 지정하려면 이 문자열을 사용합니다. 쿼리에서 Amazon Kendra의 모든 기능을 사용할 수 있습니다. 유효한 쿼리를 지정하지 않으면 `InvalidLambdaResponseException` 예외가 발생합니다. 자세한 정보는 Amazon Kendra 개발자 안내서의 쿼리를 참조하세요.

필터 또는 쿼리 문자열을 생성한 후 응답 `dialogAction` 필드를 `delegate`로 설정하여 Amazon Lex에 응답을 보냅니다. Amazon Lex는 Amazon Kendra 에 쿼리를 보낸 다음 이행 코드 후크에 쿼리 응답을 반환합니다.

응답에 이행 코드 후크 사용

Amazon Lex가 Amazon Kendra에 쿼리를 보내면 쿼리 응답이 `AMAZON.KendraSearchIntent` 이행 Lambda 함수로 반환됩니다. 코드 후크에 대한 입력 이벤트에는 Amazon Kendra의 전체 응답이 포함되어 있습니다. 쿼리 데이터는 Amazon Kendra `Query` 작업에서 반환된 것과 동일한 구조입니다. 자세한 정보는 Amazon Kendra 개발자 안내서의 쿼리 응답 구문을 참조하세요.

이행 코드 후크는 선택 사항입니다. 이행 코드 후크가 존재하지 않거나 이행 코드 후크가 응답에 메시지를 반환하지 않는 경우 Amazon Lex는 응답에 `conclusion` 문을 사용합니다.

예제: Amazon Kendra 인덱스에 대한 FAQ 봇 생성

이 예제에서는 Amazon Kendra 인덱스를 사용하여 사용자의 질문에 대한 답변을 제공하는 Amazon Lex 봇을 생성합니다. FAQ 봇은 사용자와의 대화를 관리합니다. `AMAZON.KendraSearchIntent` 의도를 사용하여 인덱스에 쿼리하고 사용자에게 응답을 제공합니다. 봇을 생성하려면

1. 고객이 봇으로부터 답변을 얻기 위해 상호 작용할 봇을 생성합니다.
2. 사용자 지정 의도를 생성합니다. 봇에는 하나 이상의 표현과 함께 하나 이상의 의도가 필요합니다. 이 의도는 봇을 빌드하는 데 필요하지만 다른 방식으로는 사용되지 않습니다.
3. 봇에 `KendraSearchIntent` 의도를 추가하고 Amazon Kendra 인덱스와 함께 작동하도록 구성합니다.
4. Amazon Kendra 인덱스에 저장된 문서로 답변할 수 있는 질문을 하여 봇을 테스트합니다.

이 예제를 사용하기 전에 Amazon Kendra 인덱스를 생성해야 합니다. 자세한 내용은 Amazon Kendra 개발자 안내서의 S3 버킷(콘솔)으로 시작하기를 참조하세요.

FAQ 봇을 생성하려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lex/ Amazon Lex 콘솔을 엽니다.
2. 탐색 창에서 봇을 선택합니다.
3. 생성(Create)을 선택합니다.
4. 사용자 지정 봇을 선택합니다. 다음과 같이 봇을 구성합니다.
   - 봇 이름 – 봇에 용도를 나타내는 이름(예: **KendraTestBot**)을 지정합니다.
   - 음성 출력 – 없음을 선택합니다.
   - 세션 제한 시간 – **5**를 입력합니다.
   - 감정 분석 – 아니요를 선택합니다.
   - COPPA – 아니요를 선택합니다.
   - 사용자 표현 저장 – 저장 안 함을 선택합니다.
5. 생성(Create)을 선택합니다.

봇을 성공적으로 빌드하려면 하나 이상의 샘플 표현을 사용하여 하나 이상의 의도를 생성해야 합니다. 이 의도는 Amazon Lex 봇을 빌드하는 데 필요하지만 FAQ 응답에는 사용되지 않습니다. 이 의도의 표현은 고객이 묻는 질문에 해당하지 않아야 합니다.

필요한 의도를 생성하려면

1. 봇 시작하기 페이지에서 의도 생성을 선택합니다.
2. 의도 추가에서 의도 생성을 선택합니다.
3. 의도 생성 대화 상자에서 의도의 이름(예: **RequiredIntent**)을 지정합니다.
4. 샘플 표현에 표현(예: **Required utterance**)를 입력합니다.
5. 의도 저장을 선택합니다.

이제 Amazon Kendra 인덱스를 검색하기 위한 의도와 이를 통해 반환되어야 하는 응답 메시지를 생성합니다.

AMAZON.KendraSearchIntent 및 응답 메시지를 생성하려면

1. 탐색 창에서 의도 옆에 있는 더하기(+) 기호를 선택합니다.
2. 의도 추가에서 기존 의도 검색을 선택합니다.

3. 의도 검색 상자에 **AMAZON.KendraSearchIntent**를 입력한 다음 목록에서 이를 선택합니다.
4. 기본 제공 의도 복사에서 의도 이름(예: **KendraSearchIntent**)을 입력한 다음 추가를 선택합니다.
5. 의도 편집기에서 Amazon Kendra 쿼리를 선택하여 쿼리 옵션을 엽니다.
6. Amazon Kendra 인덱스 메뉴에서 검색하려는 인덱스를 선택합니다.
7. 응답 섹션에서 다음 3가지 메시지를 추가합니다.

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

8. 의도 저장을 선택한 다음 빌드를 선택하여 봇을 빌드합니다.

마지막으로 콘솔 테스트 창을 사용하여 봇의 응답을 테스트합니다. 질문이 인덱스가 지원하는 도메인에 있어야 합니다.

FAQ 봇을 테스트하려면

1. 콘솔 테스트 창에서 인덱스에 대한 질문을 입력합니다.
2. 테스트 창의 응답 섹션에서 답을 확인합니다.
3. 다른 질문에 대한 테스트 창을 재설정하려면 채팅 기록 지우기를 선택합니다.

AMAZON.PauseIntent

사용자가 봇과의 상호 작용을 일시 중지하여 나중에 다시 돌아올 수 있도록 하는 단어 및 구문에 응답합니다. Lambda 함수 또는 애플리케이션이 의도 데이터를 세션 변수에 저장하거나, 현재 의도를 재개할 때 GetSession 작업을 사용하여 의도 데이터를 검색해야 합니다.

공통 표현:

- 일시 중지
- 일시 중지

AMAZON.RepeatIntent

사용자가 이전 메시지를 반복하게 할 수 있는 단어와 구문에 응답합니다. 애플리케이션은 Lambda 함수를 사용하여 이전 의도 정보를 세션 변수에 저장하거나 GetSession 작업을 사용하여 이전 의도 정보를 가져와야 합니다.

공통 표현:

- 반복
- 다시 말해줘
- 반복해줘

AMAZON.ResumeIntent

사용자가 이전에 일시 중지된 의도를 재개할 수 있도록 단어와 구문에 응답합니다. Lambda 함수 또는 애플리케이션은 이전 의도를 재개하는 데 필요한 정보를 관리해야 합니다.

공통 표현:

- 재개
- 계속
- 지속

AMAZON.StartOverIntent

사용자가 현재 의도 처리를 중단하고 처음부터 다시 시작할 수 있도록 하는 단어와 구문에 응답합니다. Lambda 함수 또는 `PutSession` 작업을 사용하여 첫 번째 슬롯 값을 다시 이끌어낼 수 있습니다.

공통 표현:

- 다시 시작
- 재시작
- 다시 시작해

AMAZON.StopIntent

사용자가 현재 의도 처리를 중단하고 봇과의 상호작용을 종료하기를 원한다는 것을 나타내는 단어와 문구에 응답합니다. Lambda 함수 또는 애플리케이션은 기존 속성 및 슬롯 유형 값을 모두 지운 다음 상호 작용을 종료해야 합니다.

공통 표현:

- 멈춰
- 꺼
- 조용히 해

기본 제공 슬롯 유형

Amazon Lex는 슬롯의 데이터를 인식하고 처리하는 방법을 정의하는 기본 제공 슬롯 유형을 지원합니다. 의도에 이러한 유형의 슬롯을 만들 수 있습니다. 따라서 날짜, 시간 및 위치와 같이 흔히 사용되는 슬롯 데이터의 열거 값을 만들 필요가 없습니다. 기본 제공 슬롯 유형에는 버전이 없습니다.

슬롯 유형	간략한 설명	지원되는 로캘
AMAZON.Airport	공항을 나타내는 단어를 인식합니다.	모든 로캘
AMAZON.AlphaNumeric	문자와 숫자로 구성된 단어를 인식합니다.	한국어(Ko-KR)를 제외한 모든 로캘
AMAZON.City	공항을 나타내는 단어를 인식합니다.	모든 로캘
AMAZON.Country	공항을 나타내는 단어를 인식합니다.	모든 로캘
AMAZON.DATE	날짜를 나타내는 단어를 인식하여 표준 형식으로 변환합니다.	모든 로캘

슬롯 유형	간략한 설명	지원되는 로캘
AMAZON.DURATION	지속 시간을 나타내는 단어를 인식하여 표준 형식으로 변환합니다.	모든 로캘
AMAZON.EmailAddress	이메일 주소를 나타내는 단어를 표준 이메일 주소로 변환합니다.	모든 로캘
AMAZON.FirstName	이름을 나타내는 단어를 인식합니다.	모든 로캘
AMAZON.LastName	성을 나타내는 단어를 인식합니다.	모든 로캘
AMAZON.NUMBER	숫자 단어를 인식하여 숫자로 변환합니다.	모든 로캘
AMAZON.Percentage	백분율을 나타내는 단어를 숫자와 퍼센트 기호(%)로 변환합니다.	모든 로캘
AMAZON.PhoneNumber	전화번호를 나타내는 단어를 숫자 문자열로 변환합니다.	모든 로캘
AMAZON.SpeedUnit	속도 단위를 나타내는 단어를 표준 약어로 변환합니다.	영어(미국)(en-US)
AMAZON.State	상태를 나타내는 단어를 인식합니다.	모든 로캘
AMAZON.StreetName	거리 이름을 나타내는 단어를 인식합니다.	영어(미국)(en-US)을 제외한 모든 로캘

슬롯 유형	간략한 설명	지원되는 로캘
AMAZON.TIME	시간을 나타내는 단어를 시간 형식으로 변환합니다.	모든 로캘
AMAZON.WeightUnit	속도 단위를 나타내는 단어를 표준 약어로 변환합니다.	영어(미국)(en-US)

> **Note**
>
> 영어(미국)(en-US) 로캘의 경우 Amazon Lex는 Alexa 스킬 키트의 슬롯 유형을 지원합니다. 사용 가능한 기본 제공 슬롯 유형의 목록은 Alexa Skills Kit 설명서의 슬롯 유형 참조를 참조하십시오.
>
> - Amazon Lex는 `AMAZON.LITERAL` 또는 `AMAZON.SearchQuery` 기본 제공 슬롯 유형을 지원하지 않습니다.

AMAZON.Airport

공항 목록을 제공합니다. 그러한 예는 다음과 같습니다.

- 존 F. 케네디 국제공항
- 멜버른 공항

AMAZON.AlphaNumeric

`APQ123`과 같은 문자와 숫자로 구성된 문자열을 인식합니다.

이 슬롯 유형은 한국어(Ko-KR)로캘에서 사용할 수 없습니다.

다음을 포함하는 문자열에 `AMAZON.AlphaNumeric` 슬롯 유형을 사용할 수 있습니다.

- 알파벳 문자(예: **`ABC`**)
- 숫자(예: **`123`**)
- 영숫자 조합(예: **`ABC123`**)

`AMAZON.AlphaNumeric` 슬롯 유형에 정규식을 추가하여 슬롯에 입력된 값의 유효성을 검사할 수 있습니다. 예를 들어 정규식을 사용하여 다음 항목의 유효성을 검사할 수 있습니다.

- 영국 또는 캐나다 우편 번호
- 운전 면허증 번호
- 차량 식별 번호

표준 정규 표현식을 사용합니다. Amazon Lex는 정규 표현식에서 다음 문자를 지원합니다.

- A~Z, a~z
- 0~9

Amazon Lex는 정규식에서 유니코드 문자도 지원합니다. 형식은 `\u`*`Unicode`*입니다. 유니코드 문자를 나타내려면 4자리 숫자를 사용합니다. 예를 들어 `[\u0041-\u005A]`는 [A~Z]와 같습니다.

다음 정규식 연산자는 지원되지 않습니다.

- 무한 반복자: 상한이 없는 *, + 또는 {x,}
- 와일드 카드(.)

정규식의 최대 길이는 300자입니다. 정규식을 사용하는 AMAZON.AlphaNumeric 슬롯 유형에 저장되는 문자열의 최대 길이는 30자입니다.

다음은 정규식의 몇 가지 예입니다.

- **`APQ123`** 또는 **`APQ1`** 같은 영숫자 문자열: `[A-Z]{3}[0-9]{1,3}` 또는 보다 제약된 `[A-DP-T]{3}[1-5]{1,3}`
- **`CP123456789US`**같은 USPS(US Postal Service) 국제 우선 취급 우편 형식: `CP[0-9]{9}US`
- **`123456789`** 같은 은행 송금 번호: `[0-9]{9}`

슬롯 유형에 정규식을 설정하려면 콘솔 또는 PutSlotType 작업을 사용합니다. 슬롯 유형을 저장하면 정규식의 유효성이 검사됩니다. 정규식이 유효하지 않으면 Amazon Lex에서 오류 메시지가 반환됩니다.

슬롯 유형에 정규식을 사용하는 경우 Amazon Lex는 정규식과 대조하여 해당 유형의 슬롯에 대한 입력을 확인합니다. 입력이 정규식과 일치하면 해당 값이 슬롯에 허용되고, 입력이 정규식과 일치하지 않으면 Amazon Lex에서 다시 입력하라는 메시지가 표시됩니다.

AMAZON.City

지역 및 세계 도시 목록을 제공합니다. 슬롯 유형은 도시 이름의 일반적인 변형을 인식합니다. Amazon Lex는 변형을 공식 명칭으로 변환하지 않습니다.

예시:

- 뉴욕
- 레이캬비크
- 도쿄
- 베르사유

AMAZON.Country

전 세계 국가의 이름입니다. 예시:

- 호주
- 독일
- 일본
- 미국
- 우루과이

AMAZON.DATE

날짜를 나타내는 단어를 날짜 형식으로 변환합니다.

날짜는 의도에 ISO-8601 날짜 형식으로 제공됩니다. 의도가 슬롯에서 수신되는 날짜는 사용자가 말한 특정 문구에 따라 달라질 수 있습니다.

- "오늘", "지금" 또는 "11월 25일"과 같이 특정 날짜에 매핑되는 표현은 전체 날짜로 변환됩니다: `2020-11-25`. 기본값은 현재 날짜 또는 그 이후의 날짜입니다.
- "이번 주" 또는 "다음 주"와 같이 특정 주에 매핑되는 표현은 해당 주의 첫째 날의 날짜로 변환됩니다. ISO-8601 형식에서는 한 주가 월요일에 시작하여 일요일에 끝납니다. 예를 들어 오늘이 2020-11-25인 경우 "다음 주"는 `2020-11-30`로 변환됩니다.
- "다음 달"과 같이 특정 날짜가 아닌 달에 매핑되는 표현은 해당 월의 마지막 날로 변환됩니다. 예를 들어 오늘이 2020-11-25인 경우 "다음 달"은 `2020-12-31`로 변환됩니다.

- 연도에 매핑되지만 특정 월이나 요일은 아닌 표현(예: "다음 연도")는 다음 해의 마지막 날로 변환됩니다. 예를 들어 오늘이 2020-11-25인 경우 "다음 연도"는 `2021-12-31`로 변환됩니다.

AMAZON.DURATION

지속 시간을 나타내는 단어를 지속 시간으로 변환합니다.

지속 시간은 ISO-8601 지속 시간 형식, `PnYnMnWnDTnHnMnS`를 기반으로 하는 형식으로 확인됩니다. `P`은 해당 사항이 기간이며, `n`가 숫자 값이고, `n` 뒤에 오는 대문자가 특정 날짜 또는 시간 요소임을 나타냅니다. 예를 들어, `P3D`은 3일을 의미합니다. `T`는 나머지 값이 날짜 요소가 아닌 시간 요소를 나타낸다는 것을 나타내는 데 사용됩니다.

예시:

- "10분": `PT10M`
- "다섯 시간": `PT5H`
- "3일": `P3D`
- "45초": `PT45S`
- "8주": `P8W`
- "7년": `P7Y`
- "5시간 10분": `PT5H10M`
- "2년 3시간 10분": `P2YT3H10M`

AMAZON.EmailAddress

username@domain으로 제공된 이메일 주소를 나타내는 단어를 인식합니다. 주소에는 사용자 이름 부분에 밑줄(_), 하이픈(-), 마침표(.) 및 더하기 기호(+)와 같은 특수 문자를 포함할 수 있습니다.

AMAZON.FirstName

일반적으로 사용되는 이름입니다. 이 슬롯 유형은 공식 이름과 비공식 닉네임을 모두 인식합니다. 의도에 전송된 이름은 사용자가 보낸 값입니다. Amazon Lex는 별명을 정식 이름으로 변환하지 않습니다.

비슷하지만 철자가 다른 이름의 경우 Amazon Lex는 의도에 단일 공통 양식을 전송합니다.

영어(미국)(en-US) 로캘에서는 AMAZON.US_First_Name이라는 슬롯 이름을 사용하십시오.

예시:

- 에밀리
- 존
- 소피

AMAZON.LastName

일반적으로 사용되는 성입니다. 철자가 다르지만 비슷하게 들리는 이름의 경우 Amazon Lex는 의도에 단일 공통 양식을 전송합니다.

영어(미국)(en-US) 로캘에서는 AMAZON.US_Last_Name이라는 슬롯 이름을 사용합니다.

예시:

- 브로스키
- 대셔
- 에버스
- 파레스
- 웰트

AMAZON.NUMBER

숫자를 표현하는 단어나 숫자를 십진수를 포함한 숫자로 변환합니다. 다음 표에는 `AMAZON.NUMBER` 슬롯 유형이 숫자 단어를 캡처하는 방식이 나와 있습니다.

입력	응답
백이십삼점사오	123.45
백이십삼점사오	123.45
영점사이	0.42
영점사이	0.42
232.998	232.998

입력	응답
50	50

AMAZON.Percentage

백분율을 나타내는 단어와 기호를 숫자와 퍼센트 기호(%)로 변환합니다.

사용자가 퍼센트 기호 또는 "퍼센트"라는 단어 없이 숫자를 입력하면 슬롯 값은 숫자로 설정됩니다. 다음 표에는 `AMAZON.Percentage` 슬롯 유형이 백분율을 캡처하는 방식이 나와 있습니다.

입력	응답
50 퍼센트	50%
0.4 퍼센트	0.4%
23.5%	23.5%
이십오 퍼센트	25%

AMAZON.PhoneNumber

전화번호를 나타내는 숫자 또는 단어를 다음과 같이 구두점이 없는 문자열 형식으로 변환합니다.

유형	설명	입력	결과

앞에 더하기(+) 기호가 표시된 국제 전화번호	앞에 더하기 기호가 표시된 11자리 숫자	+61 7 4445 1061
		
		+1 (509) 555-1212
			`+61744431061`
			
			`+15095551212`

앞에 더하기(+) 기호가 표시되지 않은 국제 전화번호	앞에 더하기 기호가 표시되지 않은 11자리 숫자	1 (509) 555-1212
		
		61 7 4445 1061
			`15095551212`
			
			`61744451061`

국내 전화번호	국가 번호가 표시되지 않은 10자리 숫자	(03) 5115 4444 (509) 555-1212	`0351154444`

유형	설명	입력	결과
			5095551212
시내 전화번호	국가 번호 또는 지역 번호가 표시되지 않은 7자리 전화번호	555-1212	5551212

AMAZON.SpeedUnit

속도 단위를 나타내는 단어를 해당 약어로 변환합니다. 예를 들어 "시간당 마일"은 mph로 변환됩니다.

이 슬롯 유형은 영어(미국)(en-US) 로캘에서만 사용할 수 있습니다.

다음 예제는 `AMAZON.SpeedUnit` 슬롯 유형이 속도 단위를 캡처하는 방식을 보여 줍니다.

속도 단위	약어
시간당 마일, mph, MPH, m/h	mph
시간당 킬로미터, 시간당 km, kmph, KMPH, km/h	kmph
초당 미터, mps, MPS, m/s	mps
시간당 해리, knots, knot	knot

AMAZON.State

국가 내 지리적 및 정치적 지역의 이름.

예시:

- 바이에른
- 후쿠시마 현
- 퍼시픽 노스웨스트
- 퀸즐랜드

- 웨일스

AMAZON.StreetName

일반적인 도로명 주소 내의 거리 이름. 여기에는 집 번호가 아닌 도로명만 포함됩니다.

이 슬롯 유형은 영어(미국)(en-US) 로캘에서는 사용할 수 없습니다.

예시:

- 캔버라 애비뉴
- 프론트 스트리트
- 마켓 로드

AMAZON.TIME

시간을 나타내는 단어를 시간 값으로 변환합니다. 모호한 시기에 대한 해결책이 포함되어 있습니다. 사용자가 모호한 시간을 입력하면 Amazon Lex는 Lambda 이벤트의 `slotDetails` 속성을 사용하여 모호한 시간에 대한 확인을 Lambda 함수로 전달합니다. 예를 들어 봇이 사용자에게 배송 시간을 입력하라는 메시지를 표시하면 사용자는 "10시 정각"이라고 응답할 수 있습니다. 사용자가 응답한 시간은 모호합니다. 이 시간은 10 AM 또는 10 PM을 의미할 수 있습니다. 이 경우 `slots` 맵의 값은 `null`이고 `slotDetails` 엔터티에는 이 시간에 대해 가능한 두 가지 확인이 포함되어 있습니다. Amazon Lex는 다음 이벤트를 사용하여 Lambda 함수를 호출합니다.

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

사용자가 모호한 시간으로 응답하면 Amazon Lex는 해당 시간을 Lambda 이벤트의 `slots` 속성의 Lambda 함수로 보내고 `slotDetails` 속성은 비어 있습니다. 예를 들어 사용자가 배송 시간을 묻는 메시지에 "10 PM"이라고 응답하면 Amazon Lex는 Lambda 함수에 다음과 같이 입력합니다.

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

Amazon Lex에서 Lambda 함수로 보낸 데이터에 대한 자세한 내용은 입력 이벤트 형식 섹션을 참조하십시오.

AMAZON.WeightUnit

중량 단위를 나타내는 단어를 해당 약어로 변환합니다. 예를 들어 "킬로그램"은 kg로 변환됩니다.

이 슬롯 유형은 영어(미국)(en-US) 로캘에서만 사용할 수 있습니다.

다음 예제는 `AMAZON.WeightUnit` 슬롯 유형이 중량 단위를 캡처하는 방식을 보여 줍니다.

중량 단위	약어
킬로그램, 킬로, kgs, KGS	kg
그램, gms, gm, GMS, g	g
밀리그램, mg, mgs	mg
파운드, lbs, LBS	lbs
온스, oz, OZ	oz
톤, ton, t	t
킬로톤, kt	kt

사용자 지정 슬롯 유형

각 의도에 대해 사용자의 요청을 이행하기 위해 의도에 필요한 정보를 나타내는 파라미터를 지정할 수 있습니다. 이러한 파라미터 또는 슬롯에는 일종의 유형이 있습니다. 슬롯 유형은 Amazon Lex가

슬롯에 대한 값을 인식하기 위해 기계 학습 모델을 학습하는 데 사용하는 값 목록입니다. 예를 들어 "`Genres.`"라는 슬롯 유형을 정의할 수 있는데, 슬롯 유형의 각 값은 "코미디," "어드벤처," "다큐멘터리" 등의 장르 이름입니다. 슬롯 유형 값에 대한 동의어를 정의할 수 있습니다. 예를 들어 "코미디" 값에 대한 동의어로 "재밌는" 및 "유머러스한"를 정의할 수 있습니다.

슬롯 값에 대한 확인을 제한하도록 슬롯 유형을 구성할 수 있습니다. 슬롯 값은 열거로 사용되며 사용자가 입력한 값은 슬롯 값 또는 동의어 중 하나와 동일한 경우에만 슬롯 값으로 확인됩니다. 동의어는 해당 슬롯 값으로 확인됩니다. 예를 들어 사용자가 "재밌는"를 입력하면 슬롯 값 "코미디"로 확인됩니다.

또는 값을 확장하도록 슬롯 유형을 구성할 수 있습니다. 슬롯 값은 학습 데이터로 사용되며 사용자가 입력한 값이 슬롯 값 및 동의어와 유사한 경우 슬롯이 해당 값으로 확인됩니다. 이는 기본 설정 동작입니다.

Amazon Lex 는 슬롯에 가능한 확인 목록을 유지 관리합니다. 목록의 각 항목은 Amazon Lex 가 슬롯의 추가 가능함으로 인식한 확인 값을 제공합니다. 확인 값은 슬롯 값을 일치시키기 위한 최고의 방법입니다. 확인 목록에는 최대 다섯 개의 값이 포함됩니다.

사용자가 입력한 값이 동의어인 경우 확인 값 목록의 첫 번째 항목이 슬롯 유형 값입니다. 예를 들어 사용자가 "재밌는"를 입력한 경우 `slots` 필드에 "재밌는"이 포함되어 있고 `slotDetails` 필드의 첫 번째 항목은 "코미디"입니다. PutSlotType 작업을 사용하여 슬롯 유형을 생성 또는 업데이트할 때 슬롯 값이 확인 목록의 첫 번째 값으로 채워지도록 `valueSelectionStrategy`를 구성할 수 있습니다.

Lambda 함수 를 사용하는 경우 이 함수의 입력 이벤트에는 `slotDetails`라는 확인 목록이 포함되어 있습니다. 다음 예제는 Lambda 함수에 대한 입력의 슬롯 및 슬롯 세부 정보 단원을 보여 줍니다.

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

각 슬롯 유형에 대해 값과 동의어는 최대 10,000개까지 정의할 수 있습니다. 각 봇은 슬롯 유형 값 및 동의어를 합해 최대 50,000개까지 포함할 수 있습니다. 예를 들면, 각각 5,000개 값과

5,000개 동의어를 가진 5개의 슬롯 유형을 가지거나 아니면 각각 2,500개 값과 2,500개 동의어를 가진 10개의 슬롯 유형을 가질 수 있습니다. 이 제한을 초과한 경우, PutBot 작업을 호출할 때 `LimitExceededException`를 가져올 수 있습니다.

슬롯 난독화

Amazon Lex를 사용하면 슬롯의 내용이 표시되지 않도록 해당 내용을 난독화하거나 숨길 수 있습니다. 슬롯 값으로 캡처된 중요한 데이터를 보호하려면 슬롯 난독화를 활성화하여 대화 로그에서 해당 값을 마스킹할 수 있습니다.

슬롯 값을 난독화하도록 선택한 경우 Amazon Lex는 슬롯 값을 대화 로그의 슬롯 이름으로 바꿉니다. `full_name`이라는 슬롯의 경우 슬롯 값은 다음과 같이 난독화됩니다.

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

표현에 괄호 문자({})가 있는 경우 Amazon Lex는 괄호 문자를 역슬래시 두 개(\\)로 이스케이프합니다. 예를 들어 텍스트 `{John Stiles}`는 다음과 같이 난독화됩니다.

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

슬롯 값은 대화 로그에서 난독화됩니다. 슬롯 값은 `PostContent` 및 `PostText` 작업의 응답에서 계속 사용할 수 있으며, Lambda 함수의 유효성 검사 및 이행에 사용할 수 있습니다. 프롬프트 또는 응답에 슬롯 값을 사용하는 경우 이러한 슬롯 값은 대화 로그에서 난독화되지 않습니다.

대화의 첫 번째 차례에서 Amazon Lex가 표현의 슬롯 및 슬롯 값을 인식하면 슬롯 값을 난독화합니다. 슬롯 값이 인식되지 않으면 Amazon Lex는 표현을 난독화하지 않습니다.

두 번째 및 이후 차례에서 Amazon Lex는 유도할 슬롯과 슬롯 값을 난독화해야 하는지 여부를 알고 있습니다. Amazon Lex가 슬롯 값을 인식하면 값이 난독화됩니다. Amazon Lex가 값을 인식하지 못하면 전체 표현이 난독화됩니다. 누락된 표현의 슬롯 값은 난독화되지 않습니다.

또한 Amazon Lex는 요청 또는 세션 속성에 저장하는 슬롯 값을 난독화하지 않습니다. 속성으로 난독화해야 하는 슬롯 값을 저장하는 경우 값을 암호화하거나 난독화해야 합니다.

Amazon Lex는 오디오의 슬롯 값을 난독화하지 않습니다. 오디오 트랜스크립션의 슬롯 값을 난독화합니다.

봇의 모든 슬롯을 난독화할 필요는 없습니다. 콘솔을 사용하거나 Amazon Lex API를 사용하여 난독화할 슬롯을 선택할 수 있습니다. 콘솔의 슬롯에 대한 설정에서 슬롯 난독화를 선택합니다. API를 사용하는 경우 PutIntent 작업을 호출할 때 슬롯의 `obfuscationSetting` 필드를 `DEFAULT_OBFUSCATION`으로 설정합니다.

감정 분석

감정 분석을 사용하여 사용자 표현으로 표현된 감정을 결정할 수 있습니다. 감정 정보를 사용하여 대화 흐름을 관리하거나 직접 호출 후 분석을 수행할 수 있습니다. 예를 들어 사용자 감정이 부정적이면 대화를 인간 담당자에게 넘기는 흐름을 만들 수 있습니다.

Amazon Lex 는 Amazon Comprehend와 통합되어 사용자 감정을 감지합니다. Amazon Comprehend의 응답은 텍스트의 전체 감정이 긍정, 중립, 부정 또는 혼합인지 여부를 나타냅니다. 응답에는 사용자 표현에 대한 가장 가능성이 높은 감정 및 각 감정 범주에 대한 점수가 포함됩니다. 점수는 감성이 올바르게 감지되었을 가능성을 나타냅니다.

콘솔을 사용하거나 Amazon Lex API를 사용하여 봇에 대한 감정 분석을 활성화합니다. Amazon Lex 콘솔에서 봇의 설정 탭을 선택한 다음 감정 분석 옵션을 예로 설정합니다. API를 사용하는 경우 `detectSentiment` 필드가 `true`로 설정된 PutBot 작업을 호출합니다.

감정 분석이 활성화되면 PostContent 및 PostText 작업의 응답은 다른 메타데이터와 함께 봇 응답에서 `sentimentResponse`라는 필드를 반환합니다. `sentimentResponse` 필드에는 감정 분석 결과를 포함하는 두 개의 필드인 `SentimentLabel` 및 `SentimentScore`가 있습니다. Lambda 함수 를 사용하는 경우 해당 `sentimentResponse` 필드는 함수로 전송된 이벤트 데이터에 포함됩니다.

다음은 `PostText` 또는 `PostContent`응답의 일부로 반환된 `sentimentResponse` 필드의 예입니다. `SentimentScore` 필드는 응답에 대한 점수를 포함하는 문자열입니다.

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

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

Amazon Lex 는 사용자를 대신해 Amazon Comprehend 를 호출하여 봇이 처리하는 모든 표현에서 감정을 결정합니다. 감정 분석을 활성화하면 Amazon Comprehend 에 대한 서비스 약관 및 계약에 동의하게 됩니다. Amazon Comprehend 요금에 대한 내용은 Amazon Comprehend 요금을 참조하세요.

Amazon Comprehend 감정 분석의 작동 방식에 대한 자세한 내용은 Amazon Comprehend 개발자 가이드의 감정 파악을 참조하십시오.

Amazon Lex 리소스 태그 지정

Amazon Lex 봇, 봇 별칭 및 봇 채널을 관리하는 데 도움이 되도록 각 리소스에 태그로 메타데이터를 할당할 수 있습니다. 태그는 AWS 리소스에 할당하는 레이블입니다. 각 태그는 키와 값으로 구성됩니다.

태그를 사용하면 용도, 소유자 또는 애플리케이션을 기준으로 하는 등 AWS 리소스를 다양한 방식으로 분류할 수 있습니다. 태그를 통해 다음 작업을 수행할 수 있습니다.

- AWS 리소스를 식별하고 구성합니다. 많은 AWS 리소스가 태그 지정을 지원하므로 서로 다른 서비스의 리소스에 동일한 태그를 할당하여 리소스가 관련이 있음을 나타낼 수 있습니다. 예를 들어, 봇과 봇이 사용하는 Lambda 함수에 동일한 태그를 지정할 수 있습니다.
- 비용을 할당합니다. AWS 결제 및 비용 관리 대시보드에서 태그를 활성화합니다.는 태그를 AWS 사용하여 비용을 분류하고 월별 비용 할당 보고서를 제공합니다. Amazon Lex의 경우 `$LATEST` 별칭을 제외하고 별칭과 관련된 태그를 사용하여 각 별칭에 대한 비용을 할당할 수 있습니다. `$LATEST` 별칭에 대한 비용을 할당하는 데는 Amazon Lex 봇과 관련된 태그를 사용합니다. 자세한 내용은 AWS 결제 및 비용 관리 사용 설명서의 비용 할당 태그 사용을 참조하세요.
- 리소스에 대한 액세스를 제어합니다. Amazon Lex에 태그를 사용하여 Amazon Lex 리소스에 대한 액세스를 제어하는 정책을 생성할 수 있습니다. 이러한 정책을 IAM 역할 또는 사용자에 연결하여 태그 기반 액세스 제어를 활성화할 수 있습니다. 자세한 내용은 Amazon Lex의 ABAC을 참조하세요. 리소스의 태그를 기반으로 리소스에 대한 액세스를 제한하는 자격 증명 기반 정책의 예시는 태그를 사용하여 리소스 액세스에서 확인할 수 있습니다.

AWS Management Console AWS Command Line Interface, 또는 Amazon Lex API를 사용하여 태그로 작업할 수 있습니다.

리소스에 태그 지정

Amazon Lex 콘솔을 사용하는 경우 리소스를 생성할 때 리소스에 태그를 지정하거나 나중에 태그를 추가할 수 있습니다. 이 콘솔을 사용하여 기존 태그를 업데이트하거나 제거할 수도 있습니다.

AWS CLI 또는 Amazon Lex API를 사용하는 경우 다음 작업을 사용하여 리소스에 대한 태그를 관리합니다.

- ListTagsForResource – 리소스와 연결된 태그를 봅니다.
- PutBot 및 PutBotAlias – 봇 또는 봇 별칭을 생성할 때 태그를 적용합니다.
- TagResource – 기존 리소스에서 태그를 추가 및 수정합니다.
- UntagResource – 리소스에서 태그를 제거합니다.

Amazon Lex의 다음 리소스는 태깅을 지원합니다.

- 봇 - 다음과 같은 Amazon 리소스 이름(ARN)을 사용합니다.
  - `arn:${partition}:lex:${region}:${account}:bot:${bot-name}`
- 봇 별칭 - 다음과 같은 ARN을 사용합니다.
  - `arn:${partition}:lex:${region}:${account}:bot:${bot-name}:${bot-alias}`
- 봇 채널 - 다음과 같은 ARN을 사용합니다.
  - `arn:${partition}:lex:${region}:${account}:bot-channel:${bot-name}:${bot-alias}:${channel-name}`

태그 제한

Amazon Lex 리소스의 태그에 다음과 같은 기본 제한이 적용됩니다.

- 최대 태그 수 - 50
- 최대 키 길이 - 128자
- 최대 값 길이 - 256자
- 키 및 값에 사용할 수 있는 문자 - a-z, A-Z, 0-9, 스페이스 및 _ . : / = + - @ 문자
- 키와 값은 대/소문자를 구분합니다.
- 키 접두사로 `aws:`를 사용하지 마십시오. AWS 전용입니다.

리소스에 태그 지정(콘솔)

콘솔을 사용하여 봇, 봇 별칭 또는 봇 채널 리소스의 태그를 관리할 수 있습니다. 리소스를 생성할 때 태그를 추가하거나 기존 리소스에서 태그를 추가, 수정 또는 제거할 수 있습니다.

봇을 생성할 때 태그를 추가하려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lex/ Amazon Lex 콘솔을 엽니다.
2. 생성을 선택하여 새 봇을 생성합니다.
3. 봇 생성 페이지 하단에서 태그를 선택합니다.
4. 태그 추가를 선택하고 하나 이상의 태그를 봇에 추가합니다. 최대 50개의 태그를 추가할 수 있습니다.

봇 별칭을 생성할 때 태그를 추가하려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lex/ Amazon Lex 콘솔을 엽니다.
2. 봇 별칭을 추가할 봇을 선택합니다.
3. 설정을 선택합니다.
4. 별칭 이름을 추가하고 봇 버전을 선택한 다음 태그 추가를 선택합니다.
5. 태그 추가를 선택하고 하나 이상의 태그를 봇 별칭에 추가합니다. 최대 50개의 태그를 추가할 수 있습니다.

봇 채널을 생성할 때 태그를 추가하려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lex/ Amazon Lex 콘솔을 엽니다.
2. 봇 채널을 추가할 봇을 선택합니다.
3. 채널을 선택한 다음 추가할 채널을 선택합니다.
4. 봇 채널에 대한 세부 정보를 추가한 다음 태그를 선택합니다.
5. 태그 추가를 선택하고 하나 이상의 태그를 봇 채널에 추가합니다. 최대 50개의 태그를 추가할 수 있습니다.

봇을 가져올 때 태그를 추가하려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lex/ Amazon Lex 콘솔을 엽니다.
2. 작업을 선택한 다음 가져오기를 선택합니다.
3. 봇을 가져올 zip 파일을 선택합니다.
4. 태그를 선택한 다음 태그 추가를 선택하고 하나 이상의 태그를 봇에 추가합니다. 최대 50개의 태그를 추가할 수 있습니다.

기존 봇에 대한 태그를 추가, 제거 또는 수정하려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lex/ Amazon Lex 콘솔을 엽니다.
2. 왼쪽 메뉴에서 봇을 선택한 다음 수정할 봇을 선택합니다.
3. 설정을 선택한 다음 왼쪽 메뉴에서 일반을 선택합니다.
4. 태그를 선택한 다음 봇에 대한 태그를 추가, 수정 또는 제거합니다.

봇 별칭에 대한 태그를 추가, 제거 또는 수정하려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lex/ Amazon Lex 콘솔을 엽니다.
2. 왼쪽 메뉴에서 봇을 선택한 다음 수정할 봇을 선택합니다.
3. 설정을 선택한 다음 왼쪽 메뉴에서 별칭을 선택합니다.
4. 수정할 별칭에 대한 태그 관리를 선택한 다음 봇 별칭에 대한 태그를 추가, 수정 또는 제거합니다.

기존 봇 채널에 대한 태그를 추가, 제거 또는 수정하려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lex/ Amazon Lex 콘솔을 엽니다.
2. 왼쪽 메뉴에서 봇을 선택한 다음 수정할 봇을 선택합니다.
3. 채널을 선택합니다.
4. 태그를 선택한 다음 봇 채널에 대한 태그를 추가, 수정 또는 제거합니다.

리소스에 태그 지정(AWS CLI)

AWS CLI 를 사용하여 봇, 봇 별칭 또는 봇 채널 리소스의 태그를 관리할 수 있습니다. 봇 또는 봇 별칭을 생성할 때 태그를 추가하거나 봇, 봇 별칭 또는 봇 채널의 태그를 추가, 수정 또는 제거할 수 있습니다.

모든 예제는 Linux와 macOS용입니다. Windows에서 명령을 사용하려면 Linux 줄 연속 문자(\)를 캐럿(^)으로 바꿉니다.

봇을 생성할 때 태그를 추가하려면

- 다음 단축 `put-bot` AWS CLI 명령은 봇을 생성할 때 태그를 추가하는 데 사용해야 하는 파라미터를 보여줍니다. 실제로 봇을 생성하려면 다른 파라미터를 사용해야 합니다. 자세한 내용은 4단계: 시작하기(AWS CLI) 단원을 참조하십시오.

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

봇 별칭을 생성할 때 태그를 추가하려면

- 다음 단축 `put-bot-alias` AWS CLI 명령은 봇 별칭을 생성할 때 태그를 추가하는 데 사용해야 하는 파라미터를 보여줍니다. 실제로 봇 별칭을 생성하려면 다른 파라미터를 사용해야 합니다. 자세한 내용은 연습 5: 별칭 만들기(AWS CLI) 단원을 참조하십시오.

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

리소스의 태그를 나열하려면

- `list-tags-for-resource` AWS CLI 명령을 사용하여 봇, 봇 별칭, 봇 채널과 연결된 리소스를 표시합니다.

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

리소스에서 태그를 추가하거나 수정하려면

- `tag-resource` AWS CLI 명령을 사용하여 봇, 봇 별칭 또는 봇 채널을 추가하거나 수정합니다.

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

리소스에서 태그 제거

- `untag-resource` AWS CLI 명령을 사용하여 봇, 봇 별칭 또는 봇 채널에서 태그를 제거합니다.

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

Amazon Lex 시작하기

Amazon Lex는 기존 애플리케이션과 쉽게 통합할 수 있는 간단한 API 작업을 제공합니다. 지원되는 작업 목록은 API 참조 항목을 참조하십시오. 다음 옵션 중 하나를 사용할 수 있습니다.

- AWS SDK — SDK를 사용할 때 Amazon Lex에 대한 요청은 사용자가 제공한 보안 인증을 사용하여 자동으로 서명되고 인증됩니다. 이것은 애플리케이션을 구축할 때 권장되는 선택 사항입니다.
- AWS CLI - 코드를 작성하지 않고도 AWS CLI 를 사용하여 모든 Amazon Lex 기능에 액세스할 수 있습니다.
- AWS 콘솔 — 콘솔은 Amazon Lex 테스트 및 사용을 가장 쉽게 시작하는 방법입니다.

Amazon Lex가 처음이라면 먼저 Amazon Lex: 작동 방식을 읽어 보는 것이 좋습니다.

주제

- 1단계: AWS 계정 설정 및 관리자 사용자 생성
- 2단계: 설정 AWS Command Line Interface
- 3단계: 시작하기(콘솔)
- 4단계: 시작하기(AWS CLI)

1단계: AWS 계정 설정 및 관리자 사용자 생성

Amazon Lex를 처음 사용한다면 먼저 다음 작업을 완료합니다.

1. 에 가입 AWS
2. 사용자 생성

에 가입 AWS

이미 AWS 계정이 있는 경우이 작업을 건너뜁니다.

Amazon Web Services(AWS)에 가입하면 Amazon Lex를 AWS포함한의 모든 서비스에 AWS 계정이 자동으로 등록됩니다. 사용자에게는 사용한 서비스에 대해서만 요금이 청구됩니다.

Amazon Lex 에서는 사용한 리소스에 대해서만 비용을 지불합니다. AWS 를 처음 사용하는 고객인 경우 Amazon Lex를 무료로 시작할 수 있습니다. 자세한 내용은 AWS 프리 티어를 참조하세요.

AWS 계정이 이미 있는 경우 다음 작업으로 건너뜁니다. AWS 계정이 없는 경우 다음 절차에 따라 계정을 생성합니다.

AWS 계정을 생성하려면

1. https://portal.aws.amazon.com/billing/signup을 엽니다.
2. 온라인 지시 사항을 따릅니다.

   가입 절차의 일부로는 전화 또는 문자 메시지를 수신하고 전화 키패드에 확인 코드를 입력하는 것이 포함됩니다.

   에 가입하면 AWS 계정AWS 계정 루트 사용자이 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

AWS 계정 ID는 다음 작업에 필요하므로 적어 둡니다.

사용자 생성

Amazon Lex AWS와 같은의 서비스는 서비스에 액세스할 때 자격 증명을 제공해야 해당 서비스가 소유한 리소스에 액세스할 수 있는 권한이 있는지 여부를 확인할 수 있습니다. 콘솔은 암호를 요구합니다. 그러나 AWS 계정의 자격 증명을 AWS 사용하여에 액세스하는 것은 권장하지 않습니다. 대신 다음과 같이 하는 것이 좋습니다.

- AWS Identity and Access Management (IAM)를 사용하여 사용자 생성
- 관리 권한이 있는 IAM 그룹에 사용자 추가합니다.
- 생성한 사용자에게 관리 권한을 부여합니다.

그런 다음 특수 URL과 사용자의 자격 증명을 AWS 사용하여에 액세스할 수 있습니다.

이 가이드의 시작하기 연습에서는 관리자 권한이 있는 사용자(`adminuser`)가 있다고 가정합니다. 절차에 따라 계정에서 `adminuser`를 만듭니다.

관리자 사용자를 만들고 콘솔에 로그인하려면

1. 사용자의 AWS 계정에서 `adminuser`라는 관리자 사용자를 만듭니다. 관련 지침은 IAM 사용자 가이드의 첫 번째 IAM 사용자 및 관리자 그룹 생성을 참조하십시오.

2. 사용자는 특수 URL을 AWS Management Console 사용하여에 로그인할 수 있습니다. 자세한 설명은 IAM 사용자 가이드의 사용자 계정에 로그인하는 방법을 참조하세요.

IAM에 대한 자세한 설명은 다음을 참조하세요.

- AWS Identity and Access Management (IAM)
- 시작하기
- IAM 사용자 가이드

다음 단계

2단계: 설정 AWS Command Line Interface

2단계: 설정 AWS Command Line Interface

Amazon Lex를 AWS Command Line Interface (AWS CLI)와 함께 사용하려면 다운로드하여 구성합니다.

> **Important**
> 시작하기 연습의 단계를 수행하는 AWS CLI 데가 필요하지 않습니다. 이 가이드의 일부 연습에서는 AWS CLI를 사용합니다. 콘솔을 사용하여 시작하려면 이 단계를 건너뛰고 3단계: 시작하기(콘솔)으로 이동하십시오. 나중에가 필요할 때 여기로 AWS CLI돌아와 설정합니다.

를 설정하려면 AWS CLI

1. AWS CLI를 다운로드하고 구성합니다. 관련 지침은 AWS Command Line Interface 사용 설명서에서 다음 주제를 참조하세요.
   - 를 사용하여 설정 AWS Command Line Interface
   - AWS Command Line Interface구성
2. 관리자 사용자의 명명된 프로필을 AWS CLI 구성 파일의 끝에 추가합니다. AWS CLI 명령을 실행할 때이 프로파일을 사용합니다. 프로파일 명명에 대한 자세한 설명은AWS Command Line Interface 사용자 가이드의 프로파일 명명을 참조하십시오.

   ```
   [profile adminuser]
   ```

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

사용 가능한 AWS 리전 목록은의 리전 및 엔드포인트를 참조하세요Amazon Web Services 일반 참조.

3. 명령 프롬프트에서 다음 Help 명령을 입력하여 설정을 확인합니다.

```
aws help
```

3단계: 시작하기(콘솔)

3단계: 시작하기(콘솔)

Amazon Lex 를 사용하는 방법을 배우는 가장 쉬운 방법은 콘솔을 사용하는 것입니다. 시작하는 데 도움이 되도록 다음 연습을 만들었으며, 모두 콘솔을 사용합니다.

- 연습 1 — 필요한 봇 구성을 모두 제공하는 사전 정의된 봇인 청사진을 사용하여 Amazon Lex 봇을 생성합니다. 엔드 투 엔드 설정을 테스트하기 위한 최소한의 작업만 수행합니다.

  또한에서 제공하는 Lambda 함수 블루프린트 AWS Lambda를 사용하여 Lambda 함수를 생성합니다. 함수는 봇과 호환되는 사전 정의된 코드를 사용하는 코드 후크입니다.
- 연습 2 — 봇을 수동으로 생성 및 구성하여 사용자 지정 봇을 생성합니다. Lambda 함수를 코드 후크로 생성할 수도 있습니다. 샘플 코드가 제공됩니다.
- 연습 3 - 봇을 게시하고 새로운 버전을 생성합니다. 이 실습의 일부로서 봇 버전을 가리키는 별칭을 생성합니다.

주제

- 연습 1: 청사진을 사용하여 Amazon Lex 봇 생성(콘솔)
- 연습 2: 사용자 지정 Amazon Lex 봇 생성
- 연습 3: 버전 게시 및 별칭 만들기

연습 1: 청사진을 사용하여 Amazon Lex 봇 생성(콘솔)

이 연습에서는 다음 작업을 수행합니다.

- 첫 번째 Amazon Lex 봇을 생성하여 Amazon Lex 콘솔에서 테스트합니다.

  이 연습에서는 OrderFlowers 청사진을 사용합니다. 청사진에 대한 자세한 내용은 Amazon Lex와 AWS Lambda 청사진을 참조하십시오.

- AWS Lambda 함수를 생성하고 Lambda 콘솔에서 테스트합니다. 요청 처리 중 봇이 이 Lambda 함수를 호출합니다. 이 연습에서는 AWS Lambda 콘솔에 제공된 Lambda 블루프린트(lex-order-flowers-python)를 사용하여 Lambda 함수를 생성합니다. 청사진 코드를 통해 동일한 Lambda 함수를 사용하여 초기화 및 검증을 수행하고 `OrderFlowers` 의도를 이행하는 방법에 대해 설명합니다.

- 봇을 업데이트하여 Lambda 함수를 코드 후크로 추가함으로써 의도를 이행합니다. 종합적 경험을 테스트합니다.

다음 섹션에서는 청사진에서 수행하는 작업에 대해 설명합니다.

Amazon Lex Bot: 청사진 개요

OrderFlowers 청사진을 사용하여 Amazon Lex 봇을 생성합니다. 봇의 구조에 대한 자세한 내용은 Amazon Lex: 작동 방식 섹션을 참조하십시오. 봇은 다음과 같이 미리 구성되어 있습니다.

- 의도 – OrderFlowers
- 슬롯 유형 – `FlowerTypes`이란 이름의 1개의 사용자 지정 슬롯 유형과 다음과 같은 열거 값: `roses`, `lilies` 및 `tulips`
- 슬롯 – 의도에 다음 정보(즉, 슬롯)가 충족되어야 봇이 의도를 이행할 수 있습니다.
  - `PickupTime` (AMAZON.TIME 기본 제공 유형)
  - `FlowerType` (FlowerTypes 사용자 지정 유형)
  - `PickupDate` (AMAZON.DATE 기본 제공 유형)
- 표현 – 다음 샘플 표현는 사용자의 의도를 나타냅니다.
  - "꽃을 픽업하고 싶습니다."
  - "꽃을 주문하고 싶습니다"
- 프롬프트 – 봇이 의도를 식별한 후에는 다음 프롬프트를 사용하여 슬롯을 채웁니다.
  - `FlowerType` 슬롯에 대한 프롬프트 – "어떤 꽃을 주문하고 싶으세요?"
  - `PickupDate` 슬롯에 대한 프롬프트 – "언제 {FlowerType}를 픽업하고 싶으세요?"

- `PickupTime` 슬롯에 대한 프롬프트 – "{FlowerType}의 픽업 시간은 언제인가요?"
- 확인 설명문 – "네, {FlowerType}을 {PickupDate} 날 {PickupTime}에 픽업할 수 있도록 준비할게요. 괜찮으신가요?"

AWS Lambda 함수: 블루프린트 요약

이 연습의 Lambda 함수는 초기화 및 검증과 이행 작업을 모두 수행합니다. 따라서 Lambda 함수를 생성한 후 동일한 Lambda 함수를 코드 후크로 지정하여 의도 구성을 업데이트함으로써 초기화 및 검증과 이행 작업을 모두 처리합니다.

- Lambda 함수는 초기화 및 검증 코드 후크로서 기본 검증을 수행합니다. 예를 들어, 사용자가 상품 수령 시간을 정상 영업 시간 외로 지정하는 경우, Lambda 함수는 사용자에게 시간을 되묻도록 Amazon Lex에 지시합니다.
- 이행 코드 후크의 일부로, Lambda 함수는 꽃 주문이 접수되었음(즉, 의도가 이행되었음)을 나타내는 요약 메시지를 회신합니다.

다음 단계

1단계: Amazon Lex 봇 생성(콘솔)

1단계: Amazon Lex 봇 생성(콘솔)

이 연습에서는 꽃을 주문하는 OrderFlowersBot이라는 봇을 생성합니다.

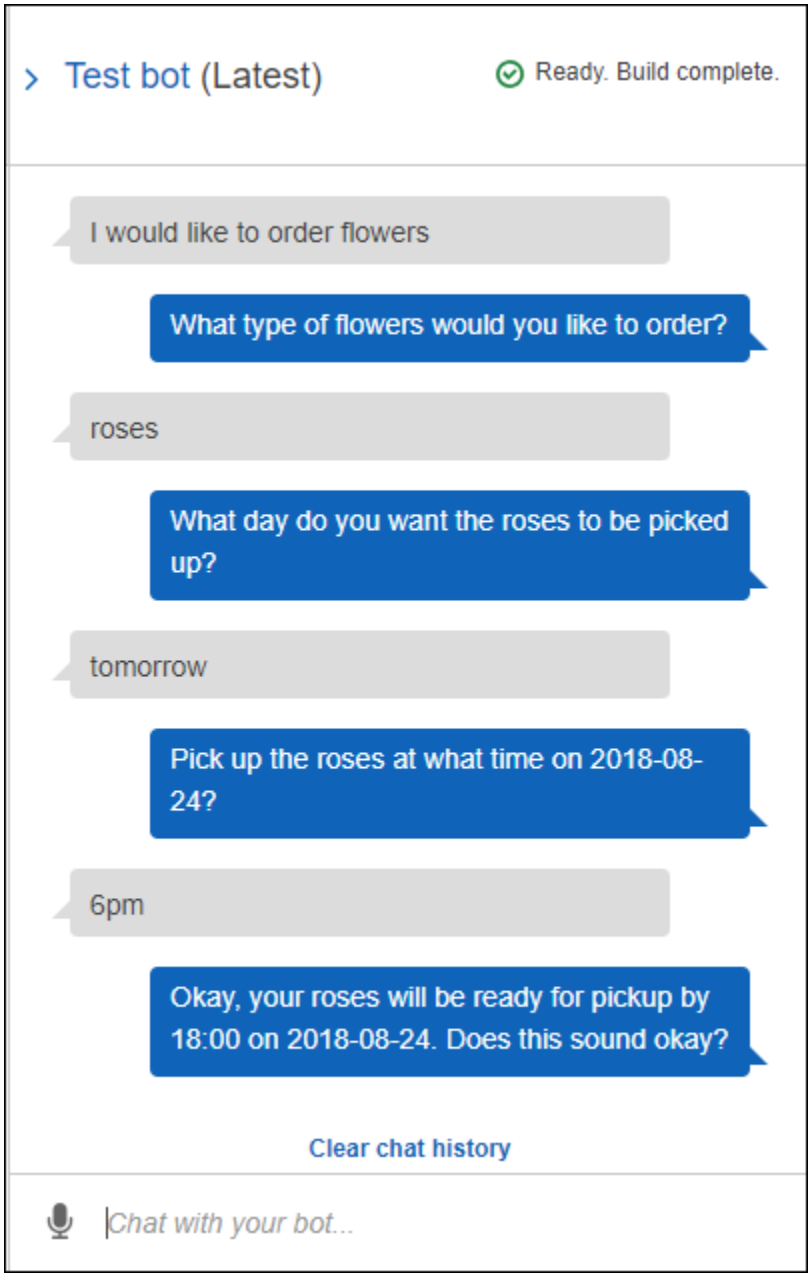
Amazon Lex 봇 생성(콘솔)

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lex/ Amazon Lex 콘솔을 엽니다.
2. 봇을 처음 사용하는 경우 시작하기를 선택합니다. 그렇지 않으면 봇 페이지에서생성을 선택합니다.
3. Lex 봇 생성 페이지에서 다음 정보를 입력한 후 생성를 선택합니다.
   - OrderFlowers 청사진을 선택합니다.
   - 봇 이름(OrderFlowers)의 기본값을 그대로 둡니다.
   - COPPA에서 **No**를 선택합니다.
   - 사용자 표현 스토리지의 경우 적절한 응답을 선택하십시오.

4. 생성(Create)을 선택합니다. Amazon Lex 콘솔이 구성을 저장하는 데 필요한 요청을 합니다. 그러면 콘솔에 봇 편집기 창이 표시됩니다.
5. 귀하의 봇이 구축되었다는 확인을 기다려 주십시오.
6. 봇 테스트.

> **Note**
>
> 테스트 창에 텍스트를 입력하거나, 호환되는 브라우저의 경우 테스트 창에서 마이크 버튼을 선택하여 말해 봇을 테스트할 수 있습니다.

다음 예제 텍스트에서는 봇과 대화하여 꽃을 주문합니다.



이 입력에서 봇은 `OrderFlowers` 의도를 유추하고 슬롯 데이터를 묻는 메시지를 표시합니다. 필요한 모든 슬롯 데이터를 제공하면, 봇은 모든 정보를 클라이언트 애플리케이션(이 경우에는 콘솔)에 반환하여 의도(`OrderFlowers`)를 이행합니다. 콘솔은 테스트 창에서 정보를 보여 줍니다.

구체적으로 설명하면 다음과 같습니다.

- `pickupDate` 슬롯에 대한 프롬프트가 대체물인 `{FlowerType}`을 사용하여 구성되기 때문에, "언제 장미를 픽업하고 싶으세요?" 라는 문장에서는 "장미"라는 단어가 표시됩니다. 콘솔에서 이러한 점을 확인합니다.

- "네, 장미를..." 문장장은 사용자가 구성한 확인 프롬프트입니다.
- 마지막 문장("`FlowerType:roses...`")은 단순히 클라이언트에게 반환되는 슬롯 데이터이며, 이 경우 테스트 창에 표시됩니다. 다음 연습에서는 Lambda 함수를 사용하여 의도를 이행해 보겠습니다. 이 경우 사용자는 주문이 이행되었다는 메시지를 받게 됩니다.

다음 단계

2단계(선택 사항): 정보 흐름의 세부 정보 검토(콘솔)

2단계(선택 사항): 정보 흐름의 세부 정보 검토(콘솔)

이 섹션에서는 샘플 대화의 각 사용자 입력에 대한 클라이언트와 Amazon Lex 간의 정보 흐름에 대해 설명합니다.

이 예제에서는 콘솔 테스트 창을 사용하여 봇과의 대화를 진행합니다.

Amazon Lex 테스트 창을 열려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lex/ Amazon Lex 콘솔을 엽니다.
2. 테스트할 봇을 선택합니다.
3. 콘솔 오른쪽에서 챗봇 테스트을 선택합니다.

말한 내용 또는 입력한 내용에 대한 정보 흐름을 확인하려면 적절한 주제를 선택합니다.

주제

- 2a단계(선택 사항): 음성 정보 흐름의 세부 정보 검토(콘솔)
- 2b단계(선택 사항): 입력한 정보 흐름의 세부 정보 검토(콘솔)

2a단계(선택 사항): 음성 정보 흐름의 세부 정보 검토(콘솔)

이 섹션에서는 클라이언트가 음성으로 요청을 보낼 때 클라이언트와 간의 정보 흐름에 대해 설명합니다. 자세한 내용은 PostContent을 참조하세요.

1. 사용자: 꽃을 주문하고 싶습니다
    a. 클라이언트(콘솔)는 Amazon Lex에 다음 PostContent 요청을 보냅니다.

```
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
```

요청 URI 및 본문 모두 Amazon Lex에 다음 정보를 제공합니다.

- 요청 URI – 봇 이름(`OrderFlowers`), 봇 별칭(`$LATEST`), 사용자 이름(사용자를 식별하는 임의 문자열)을 제공합니다. `content`는 이것이 `PostContent` API 요청(`PostText` 요청 아님)임을 나타냅니다.
- 요청 헤더
  - `x-amz-lex-session-attributes` – base64 인코딩 값은 "{}"를 나타냅니다. 클라이언트가 첫 번째 요청을 한 경우 세션 속성이 없습니다.
  - `Content-Type` – 오디오 형식을 반영합니다.
- 요청 본문 – 사용자 입력 오디오 스트림("꽃을 주문하고 싶습니다").

> **Note**
>
> 사용자가 말하기 대신에 `PostContent` API에 텍스트("꽃을 주문하고 싶습니다") 보내기를 선택하는 경우, 요청 본문이 사용자 입력입니다. `Content-Type` 헤더는 그에 따라 설정됩니다.
>
> ```
> POST /bot/OrderFlowers/alias/$LATEST/
> user/4o9wwdhx6nlheferh6a73fujd3118f5w/content HTTP/1.1
> x-amz-lex-session-attributes: "e30="
> Content-Type: "text/plain; charset=utf-8"
> Accept: accept
>
> Request body
> input stream
> ```

b. 입력 스트림에서 Amazon Lex는 의도(`OrderFlowers`)를 감지합니다. 그런 다음 의도의 슬롯 중 하나(이 경우 `FlowerType`)와 해당 값 유도 프롬프트 중 하나를 선택한 다음, 다음 헤더가 포함된 응답을 보냅니다.

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

헤더 값은 다음 정보를 제공합니다.

- `x-amz-lex-input-transcript` – 요청에 있는 오디오(사용자 입력) 전사(transcript) 기록 제공
- `x-amz-lex-message` – 응답에서 반환된 오디오 Amazon Lex의 기록 제공
- `x-amz-lex-slots` – 슬롯 및 값의 base64 인코딩된 버전:

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

- `x-amz-lex-session-attributes` – 세션 속성({})의 base64 인코딩된 버전

클라이언트는 응답 본문에서 오디오를 재생합니다.

2. 사용자의 말: 장미

a. 클라이언트(콘솔)는 Amazon Lex에 다음 PostContent 요청을 보냅니다.

```
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")
```

요청 본문은 사용자 입력 오디오 스트림(장미)입니다. `sessionAttributes`는 비어있습니다.

b. Amazon Lex는 현재 의도의 컨텍스트에서 입력 스트림을 해석합니다(이 사용자에게 `FlowerType` 슬롯에 관한 정보를 요청했음을 기억). Amazon Lex는 먼저 현재 의도의 슬롯 값을 업데이트합니다. 그런 다음 또 다른 슬롯(`PickupDate`)과 해당 프롬프트 메시지 중 하나(언제 장미를 픽업하고 싶으세요?)를 선택하고 다음 헤더가 포함된 응답을 반환합니다.

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

헤더 값은 다음 정보를 제공합니다.

- `x-amz-lex-slots` – 슬롯 및 값의 base64 인코딩된 버전:

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

- `x-amz-lex-session-attributes` – 세션 속성({})의 base64 인코딩된 버전

클라이언트는 응답 본문에서 오디오를 재생합니다.

3. 사용자의 말: 내일

a. 클라이언트(콘솔)는 Amazon Lex에 다음 PostContent 요청을 보냅니다.

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


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

요청 본문은 사용자 입력 오디오 스트림("내일")입니다. `sessionAttributes`는 비어있습니다.

b. Amazon Lex는 현재 의도의 컨텍스트에서 입력 스트림을 해석합니다(이 사용자에게 `PickupDate` 슬롯에 관한 정보를 요청했음을 기억). Amazon Lex는 현재 의도의 슬롯(`PickupDate`) 값을 업데이트합니다. 그런 다음 (`PickupTime`)의 값을 유도할 또 다른 슬롯과 값 유도 프롬프트 중 하나(2017-03-18의 언제 장미를 픽업하고 싶으세요?)를 선택하고, 다음 헤더가 포함된 응답을 반환합니다.

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

헤더 값은 다음 정보를 제공합니다.

- `x-amz-lex-slots` – 슬롯 및 값의 base64 인코딩된 버전:

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

- `x-amz-lex-session-attributes` – 세션 속성({})의 base64 인코딩된 버전

클라이언트는 응답 본문에서 오디오를 재생합니다.

4. 사용자의 말: 6 pm

a. 클라이언트(콘솔)는 Amazon Lex에 다음 PostContent 요청을 보냅니다.

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


Request body
```

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

요청 본문은 사용자 입력 오디오 스트림("6 pm")입니다. `sessionAttributes`는 비어있습니다.

b. Amazon Lex는 현재 의도의 컨텍스트에서 입력 스트림을 해석합니다(이 사용자에게 `PickupTime` 슬롯에 관한 정보를 요청했음을 기억). 먼저 현재 의도의 슬롯 값을 업데이트합니다.

이제 Amazon Lex는 모든 슬롯에 대한 정보를 갖고 있음을 감지합니다. 그러나 `OrderFlowers` 의도는 확인 메시지로 구성됩니다. 따라서 Amazon Lex는 의도를 이행하기 전에 사용자의 명시적 확인이 필요합니다. 따라서 꽃을 주문하기 전에 확인을 요청하는 다음 헤더가 포함된 응답을 보냅니다.

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

헤더 값은 다음 정보를 제공합니다.

- `x-amz-lex-slots` – 슬롯 및 값의 base64 인코딩된 버전:

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

- `x-amz-lex-session-attributes` – 세션 속성({})의 base64 인코딩된 버전

클라이언트는 응답 본문에서 오디오를 재생합니다.

5. 사용자가 "예"라고 한 경우

a. 클라이언트(콘솔)는 Amazon Lex에 다음 PostContent 요청을 보냅니다.

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/
content HTTP/1.1
x-amz-lex-session-attributes: "e30="
```

```
Content-Type: "audio/x-l16; sample-rate=16000; channel-count=1"
Accept: "audio/mpeg"


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

요청 본문은 사용자 입력 오디오 스트림("예")입니다. `sessionAttributes`는 비어있습니다.

b. Amazon Lex는 입력 스트림을 해석하여 사용자가 주문을 진행하기를 원한다고 이해합니다. `OrderFlowers` 의도는 이행 활동인 `ReturnIntent`로 구성됩니다. 이것은 의도 데이터를 모두 클라이언트에 다시 반환하라고 Amazon Lex에 지시합니다. Amazon Lex는 다음이 포함된 응답을 반환합니다.

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

`x-amz-lex-dialog-state` 응답 헤더는 `ReadyForFulfillment`로 설정되어 있습니다. 그러면 클라이언트는 의도를 이행할 수 있습니다.

6. 이제 봇을 다시 테스트합니다. 새 (사용자) 컨텍스트를 설정하려면 콘솔에서 지우기 링크를 선택합니다. `OrderFlowers` 의도에 대한 데이터를 제공하고, 일부 유효하지 않은 데이터를 포함시킵니다. 예시:

   - 재스민을 꽃 유형으로 입력(재스민은 지원되는 꽃 유형이 아님)
   - 어제를 꽃을 찾아갈 날로 입력

   사용자 데이터를 초기화 및 검증할 코드가 없으므로 봇은 이러한 값을 수락합니다. 다음 섹션에서는 Lambda 함수를 추가하여 이를 수행합니다. 다음은 Lambda 함수에 대한 참고 사항입니다.

   - 각 사용자의 입력 후 슬롯 데이터를 검증합니다. 마지막에 의도를 이행합니다. 즉, 단순히 클라이언트에게 슬롯 데이터를 반환하는 것이 아니라 봇이 꽃 주문을 처리하고 사용자에게 메시지를 반환하는 것입니다. 자세한 내용은 Lambda 함수 사용을 참조하세요.
   - 세션 속성도 설정합니다. 세션 속성에 대한 자세한 내용은 PostText 섹션을 참조하십시오.

시작하기 섹션을 완료한 후 추가 연습(추가 예제: Amazon Lex 봇 생성)을 할 수 있습니다. 여행 예약에서는 세션 속성을 사용하여 의도 전반에 걸쳐 정보를 공유함으로써 사용자와의 동적 대화에 참여합니다.

다음 단계

3단계: Lambda 함수 만들기(콘솔)

2b단계(선택 사항): 입력한 정보 흐름의 세부 정보 검토(콘솔)

이 섹션에서는 클라이언트가 `PostText` API를 사용하여 요청을 보낼 때의 클라이언트와 Amazon Lex 간의 정보 흐름에 대해 설명합니다. 자세한 내용은 PostText을 참조하세요.

1. 사용자의 입력: 꽃을 주문하고 싶습니다

   a. 클라이언트(콘솔)는 Amazon Lex에 다음 PostText 요청을 보냅니다.

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

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

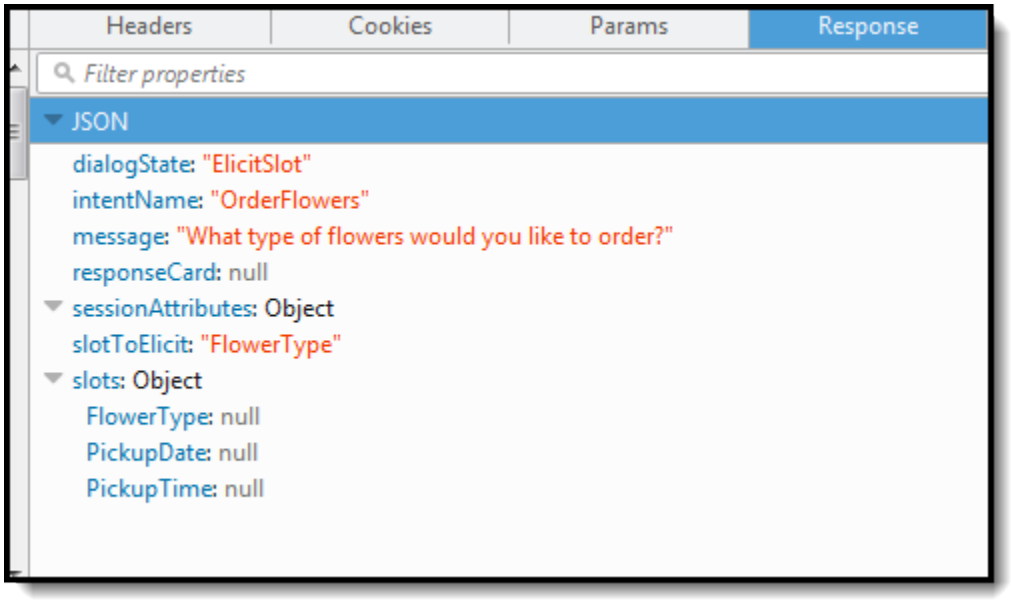
   요청 URI 및 본문 모두 Amazon Lex에 다음 정보를 제공합니다.

   - 요청 URI – 봇 이름(`OrderFlowers`), 봇 별칭(`$LATEST`), 사용자 이름(사용자를 식별하는 임의 문자열)을 제공합니다. 후행 `text`는 이것이 `PostText` API 요청(`PostContent`가 아님)임을 나타냅니다.

   - 요청 본문 – 사용자 입력(`inputText`)과 빈 `sessionAttributes`를 포함합니다. 클라이언트가 첫 번째 요청을 한 경우 세션 속성이 없습니다. Lambda 함수는 이를 나중에 초기화합니다.

   b. `inputText`에서 Amazon Lex는 의도(`OrderFlowers`)를 감지합니다. 이 의도에는 사용자 입력 또는 이행의 초기화/검증을 위한 코드 후크(즉, Lambda 함수)가 없습니다.

Amazon Lex는 값을 유도할 의도 슬롯 중 하나(`FlowerType`)를 선택합니다. 또한 슬롯의 값 유도 프롬프트 중 하나(모두 의도 구성에 포함)를 선택한 다음, 클라이언트에 다음 응답을 다시 보냅니다. 콘솔은 사용자에 대한 응답에 다음과 같은 메시지를 표시합니다.



클라이언트는 응답에 메시지를 표시합니다.

2. 사용자의 입력: 장미

   a. 클라이언트(콘솔)는 Amazon Lex에 다음 PostText 요청을 보냅니다.

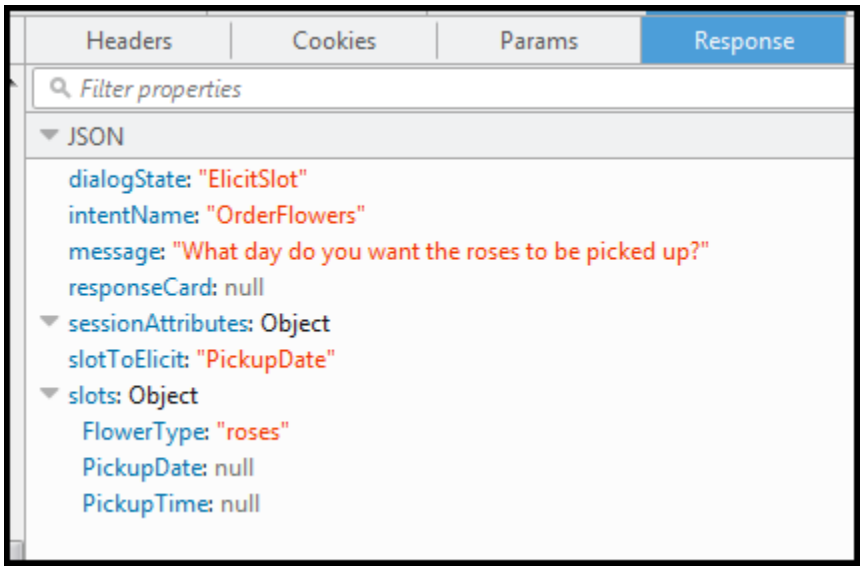
```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

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

   요청 본문에서 `inputText`는 사용자 입력입니다. `sessionAttributes`는 비어있습니다.

   b. Amazon Lex는 먼저 현재 의도의 컨텍스트에서 `inputText`를 해석합니다(이 서비스는 특정 사용자에게 `FlowerType` 슬롯에 대한 정보를 요청했음을 기억). Amazon Lex는 먼저 현재 의도의 슬롯 값을 업데이트하고, 또 다른 슬롯(`PickupDate`)을 해당 슬롯을 위한 로서 프롬프트 메시지 중 하나(언제 장미를 픽업하고 싶으세요?)를 선택합니다.

그런 다음 Amazon Lex는 다음 응답을 반환합니다.



클라이언트는 응답에 메시지를 표시합니다.

3. 사용자의 입력: 내일

   a. 클라이언트(콘솔)는 Amazon Lex에 다음 PostText 요청을 보냅니다.

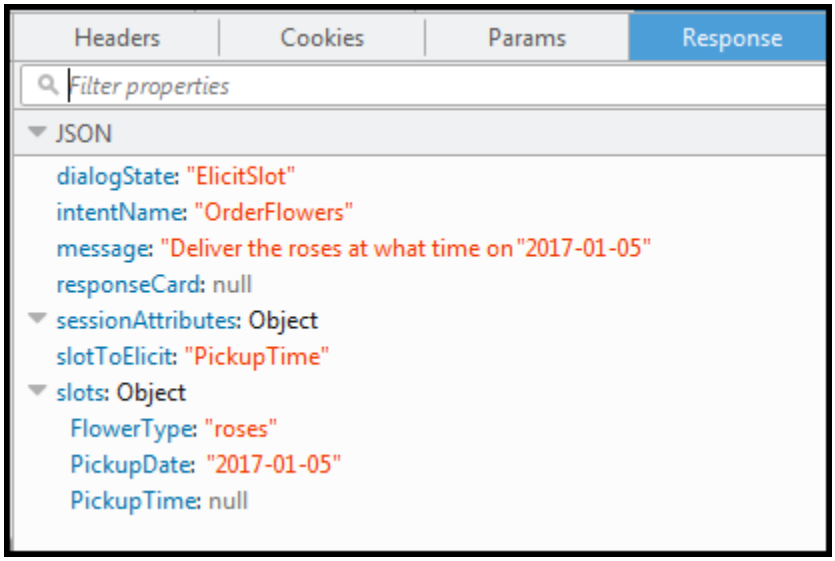
```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

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

   요청 본문에서 `inputText`는 사용자 입력입니다. `sessionAttributes`는 비어있습니다.

   b. Amazon Lex는 먼저 현재 의도의 컨텍스트에서 `inputText`를 해석합니다(이 서비스는 특정 사용자에게 `PickupDate` 슬롯에 대한 정보를 요청했음을 기억). Amazon Lex는 현재 의도의 슬롯(`PickupDate`) 값을 업데이트합니다. (`PickupTime`)의 값을 유도할 또 다른 슬롯을 선택합니다. 클라이언트에 값 유도 프롬프트 중 하나(2017-01-05 몇 시에 장미를 배송하시겠습니까?)를 반환합니다.

   그런 다음 Amazon Lex는 다음 응답을 반환합니다.



클라이언트는 응답에 메시지를 표시합니다.

4. 사용자의 입력: 6 pm

   a. 클라이언트(콘솔)는 Amazon Lex에 다음 PostText 요청을 보냅니다.

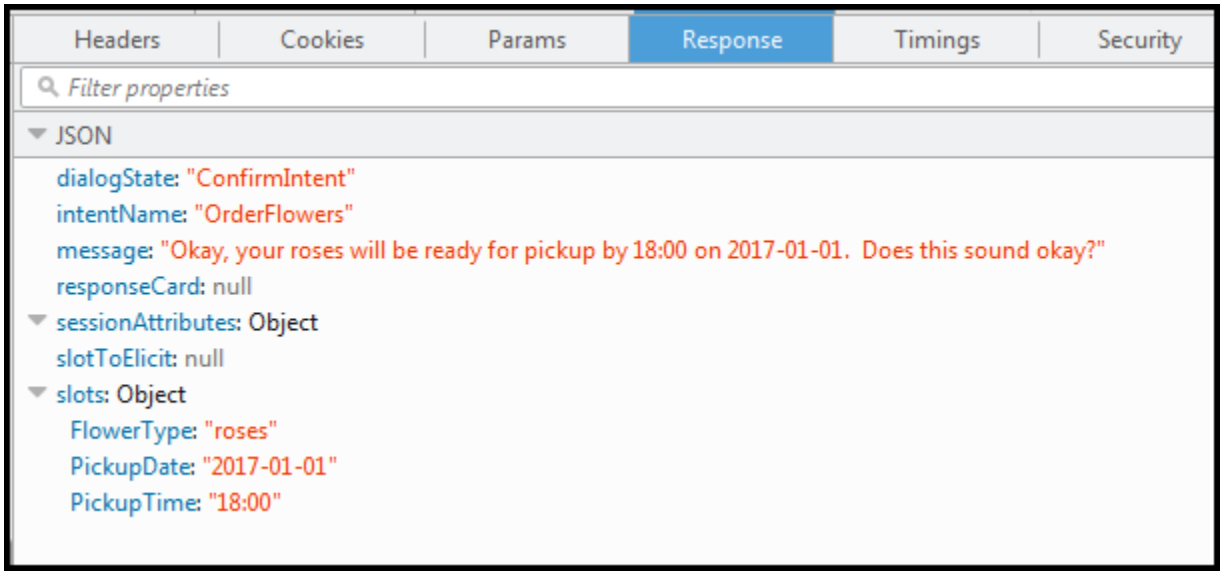
```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

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

   요청 본문에서 `inputText`는 사용자 입력입니다. `sessionAttributes`는 비어있습니다.

   b. Amazon Lex는 먼저 현재 의도의 컨텍스트에서 `inputText`를 해석합니다(이 서비스는 특정 사용자에게 `PickupTime` 슬롯에 대한 정보를 요청했음을 기억). Amazon Lex는 먼저 현재 의도의 슬롯 값을 업데이트합니다. 이제 Amazon Lex는 모든 슬롯에 대한 정보를 갖고 있음을 감지합니다.

   `OrderFlowers` 의도는 확인 메시지로 구성됩니다. 따라서 Amazon Lex는 의도를 이행하기 전에 사용자의 명시적 확인이 필요합니다. Amazon Lex는 꽃을 주문하기 전에 확인을 요청하는 다음 메시지를 클라이언트에게 보냅니다.



클라이언트는 응답에 메시지를 표시합니다.

5. 사용자의 입력: 예

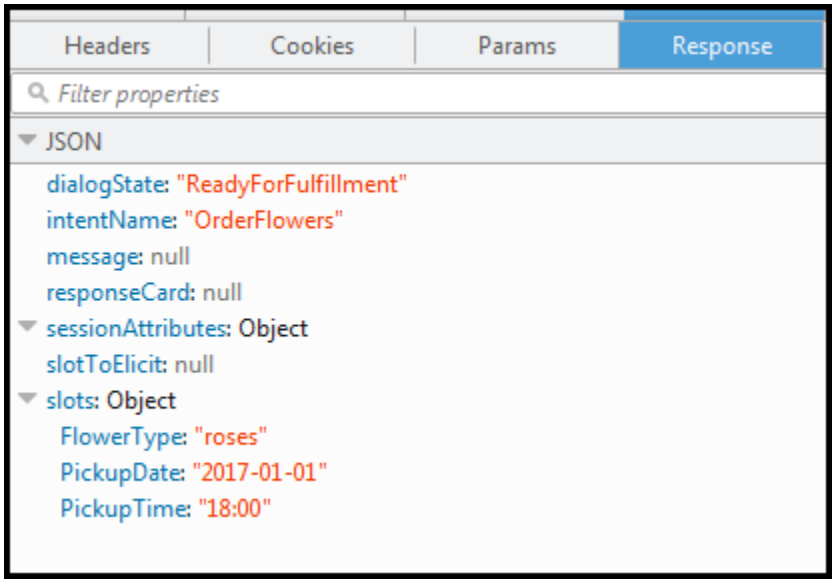
   a. 클라이언트(콘솔)는 Amazon Lex에 다음 PostText 요청을 보냅니다.

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

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

   요청 본문에서 `inputText`는 사용자 입력입니다. `sessionAttributes`는 비어있습니다.

   b. Amazon Lex는 현재 의도를 확인하는 컨텍스트에서 `inputText`를 해석합니다. 사용자가 주문을 진행하기를 원한다고 이해합니다. `OrderFlowers` 의도는 이행 활동인 `ReturnIntent`로 구성됩니다(의도를 이행할 Lambda 함수가 없음). 따라서 Amazon Lex는 클라이언트에 슬롯 데이터를 반환합니다.



Amazon Lex는 다음과 같이 `dialogState`을 `ReadyForFulfillment`로 설정했습니다. 그러면 클라이언트는 의도를 이행할 수 있습니다.

6. 이제 봇을 다시 테스트합니다. 이를 위해서는 콘솔에서 지우기 링크를 선택하여 새 (사용자) 컨텍스트를 설정해야 합니다. 이제 꽃 주문 의도에 대한 데이터를 제공할 때 잘못된 데이터를 제공해 봅니다. 예시:

   - 재스민을 꽃 유형으로 입력(재스민은 지원되는 꽃 유형이 아님)
   - 어제를 꽃을 찾아갈 날로 입력

   사용자 데이터를 초기화 및 검증할 코드가 없으므로 봇은 이러한 값을 수락합니다. 다음 섹션에서는 Lambda 함수를 추가하여 이를 수행합니다. 다음은 Lambda 함수에 대한 참고 사항입니다.

   - Lambda 함수는 각 사용자의 입력 후 슬롯 데이터를 검증합니다. 마지막에 의도를 이행합니다. 즉, 봇은 꽃 주문을 처리한 후 클라이언트에 슬롯 데이터를 반환하는 대신에 사용자에게 메시지를 반환합니다. 자세한 내용은 Lambda 함수 사용을 참조하세요.
   - Lambda 함수는 세션 속성도 설정합니다. 세션 속성에 대한 자세한 내용은 PostText 섹션을 참조하십시오.

     시작하기 섹션을 완료한 후 추가 연습(추가 예제: Amazon Lex 봇 생성)을 할 수 있습니다. 여행 예약에서는 세션 속성을 사용하여 의도 전반에 걸쳐 정보를 공유함으로써 사용자와의 동적 대화에 참여합니다.

다음 단계

3단계: Lambda 함수 만들기(콘솔)

3단계: Lambda 함수 만들기(콘솔)

Lambda 함수를 만들고(lex-order-flowers-python 청사진 사용), AWS Lambda 콘솔의 샘플 이벤트 데이터를 사용하여 테스트 간접 호출을 수행합니다.

Amazon Lex 콘솔로 돌아가서 Lambda 함수를 코드 후크로 추가하여 사용자가 이전 섹션에서 생성한 `OrderFlowersBot`에서 `OrderFlowers` 의도를 이행해 보겠습니다.

Lambda 함수를 만들려면(콘솔)

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lambda/ AWS Lambda 콘솔을 엽니다.
2. 함수 생성(Create function)을 선택합니다.
3. 함수 생성 페이지에서 청사진 사용을 선택합니다. 필터 텍스트 상자에 **`lex-`**를 입력하고 `Enter`를 눌러 청사진을 찾은 후 `lex-order-flowers-python` 청사진을 선택합니다.

   Lambda 함수 청사진은 Node.js 및 Python 모두에서 제공됩니다. 이 연습에서는 Python 기반 청사진을 사용합니다.
4. 기본 정보 페이지에서 다음을 수행합니다.
   - Lambda 함수 이름(`OrderFlowersCodeHook`)을 입력합니다.
   - Execution role(실행 역할)에서 기본 Lambda 권한을 가진 새 역할 생성을 선택합니다.
   - 나머지는 기본값을 그대로 사용합니다.
5. Create function(함수 생성)을 선택합니다.
6. 영어(미국)(en-US) 이외의 지역을 사용하는 경우 특정 로캘에 대한 청사진 업데이트에 설명된 대로 의도 이름을 업데이트하십시오.
7. Lambda 함수 테스트.
   a. 테스트 이벤트 선택, 테스트 이벤트 구성을 선택합니다.
   b. 이벤트 템플릿 목록에서 Amazon Lex 꽃 주문를 선택합니다. 이 샘플 이벤트는 요청 및 응답 모델과 일치합니다(Lambda 함수 사용 참조). 테스트 이벤트에 이름(`LexOrderFlowersTest`)을 부여합니다.
   c. 생성(Create)을 선택합니다.
   d. 테스트를 선택하여 코드 후크를 테스트합니다.

e. Lambda 함수가 성공적으로 실행되었는지 확인합니다. 이 경우 응답은 Amazon Lex 응답 모델과 일치합니다.

다음 단계

4단계: Lambda 함수를 코드 후크로 추가(콘솔)

4단계: Lambda 함수를 코드 후크로 추가(콘솔)

이 섹션에서는 OrderFlowers 의도의 구성을 업데이트하여 Lambda 함수를 다음과 같이 사용합니다.

- 먼저 Lambda 함수를 코드 후크로 사용하여 `OrderFlowers` 의도를 이행합니다. 봇을 테스트하고 Lambda 함수로부터 이행 메시지를 받았는지 확인합니다. Amazon Lex는 사용자가 꽃을 주문하는 데 필요한 모든 슬롯에 대한 데이터를 제공한 후에만 Lambda 함수를 간접 호출합니다.
- 동일한 Lambda 함수를 코드 후크로 구성하여 초기화 및 검증을 수행합니다. 테스트하고 Lambda 함수가 검증을 수행하는지 확인합니다(슬롯 데이터를 제공할 때).

Lambda 함수를 코드 후크로 추가하려면(콘솔)

1. Amazon Lex 콘솔에서 OrderFlowers 봇을 선택합니다. 콘솔에 OrderFlowers 의도가 표시됩니다. 이 버전이 수정 가능한 유일한 버전이므로 반드시 의도 버전이 `$LATEST`로 설정되어 있는지 확인합니다.
2. Lambda 함수를 이행 코드 후크로 추가하고 테스트합니다.

   a. 편집기에서 AWS Lambda 함수를 이행으로 선택하고, 사용자가 이전 단계(`OrderFlowersCodeHook`)에서 생성한 Lambda 함수를 선택합니다. 확인을 선택하여 Amazon Lex에 Lambda 함수를 간접 호출할 권한을 부여합니다.

      의도를 이행하기 위해 이 Lambda 함수를 코드 후크로 구성하고 있습니다. Amazon Lex는 사용자로부터 의도를 이행하는 데 필요한 모든 슬롯 데이터를 수신한 후에만 이 함수를 간접 호출합니다.

   b. 종료 메시지를 지정합니다.

   c. 구축을 선택합니다.

   d. 이전 대화를 사용하여 봇을 테스트합니다.

마지막 문장 "감사합니다, 장미에 대한 주문은....."은 사용자가 코드 후크로 구성한 Lambda 함수가 보내는 응답입니다. 이전 섹션에는 Lambda 함수가 없었습니다. 지금은 `OrderFlowers` 의도를 실제로 이행하기 위해 Lambda 함수를 사용하고 있습니다.

3. Lambda 함수를 초기화 및 검증 코드 후크로 추가하고 테스트합니다.

   현재 사용 중인 샘플 Lambda 함수 코드를 통해 사용자 입력에 대한 검증과 이행을 모두 수행할 수 있습니다. Lambda 함수가 수신하는 입력 이벤트에는, 해당 코드가 어떠한 부분의 코드를 실행할지 결정하는 데 사용되는 필드(`invocationSource`)가 있습니다. 자세한 내용은 Lambda 함수 입력 이벤트 및 응답 형식을 참조하세요.

   a. `OrderFlowers` 의도의 $LATEST 버전을 선택합니다. 업데이트할 수 있는 유일한 버전입니다.

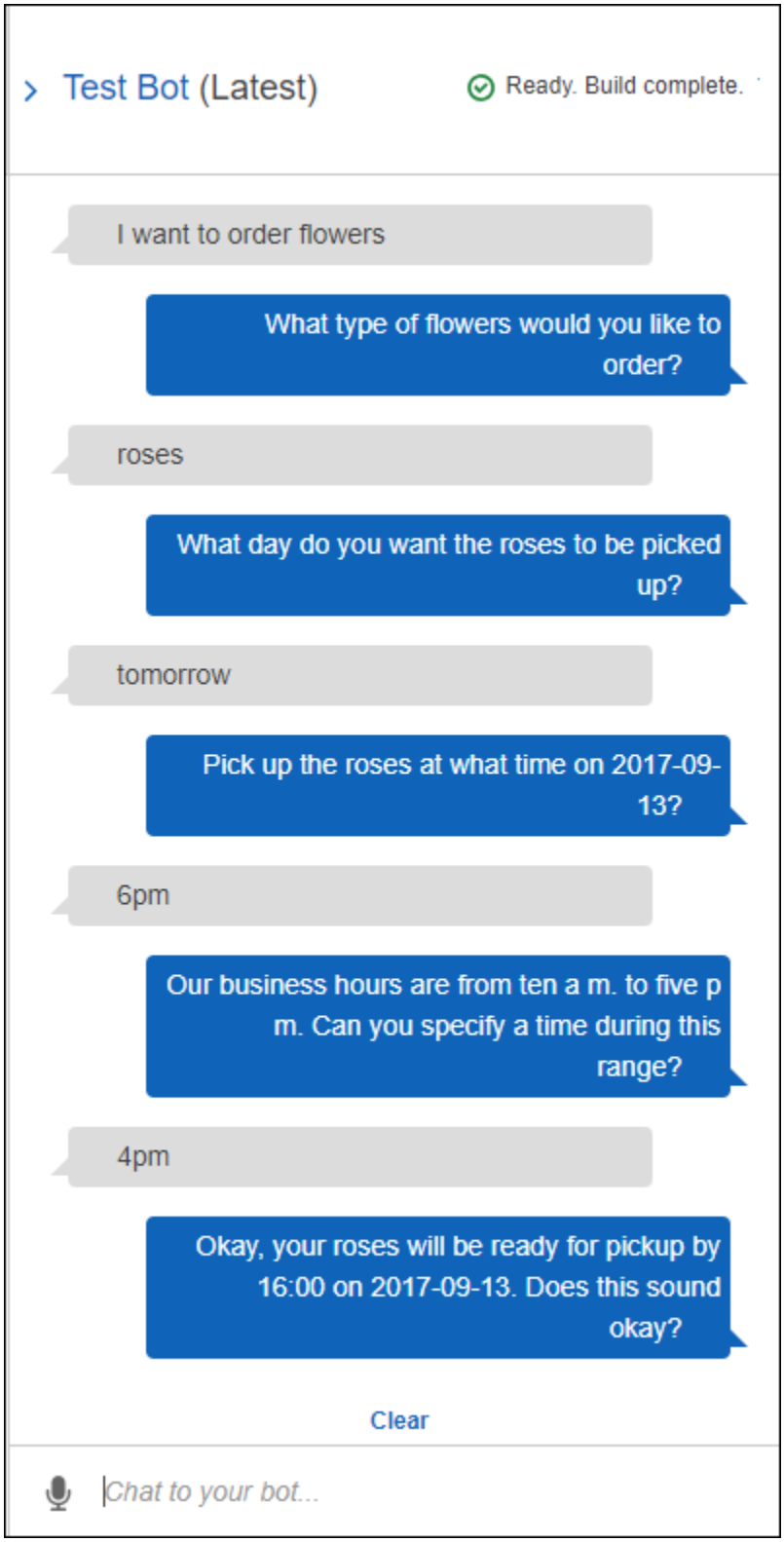
   b. 옵션에서 초기화 및 유효성 검사 코드 후크를 선택합니다.

   c. 다시 한번 동일한 Lambda 함수를 선택합니다.

   d. 구축을 선택합니다.

   e. 봇 테스트.

      이제 다음 이미지와 같이 Amazon Lex와 대화할 준비가 갖추어졌습니다. 검증 부분을 테스트하기 위해 시간을 오후 6시로 선택하면, 사용자의 Lambda 함수가 응답("영업시간은 10 AM부터 5 PM까지입니다")을 반환하고 사용자에게 한 번 더 프롬프트를 보냅니다. 유효한 슬롯 데이터가 모두 제공된 후에 Lambda 함수가 주문을 이행합니다.


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

다음 단계

5단계(선택 사항): 정보 흐름의 세부 정보 검토(콘솔)

5단계(선택 사항): 정보 흐름의 세부 정보 검토(콘솔)

이 섹션에서는 Lambda 함수의 통합을 포함하여 각 사용자 입력에 대한 클라이언트와 Amazon Lex 간의 정보 흐름에 대해 설명합니다.

> **Note**
>
> 이 섹션에서는 클라이언트가 `PostText` 런타임 API를 사용하여 Amazon Lex에 요청을 보내고 그에 따라 요청 및 응답의 세부 정보를 보여 준다고 가정합니다. 클라이언트가 `PostContent` API를 사용할 때의 클라이언트와 Amazon Lex 간의 정보 흐름의 예는 2a단계(선택 사항): 음성 정보 흐름의 세부 정보 검토(콘솔) 을 참조하십시오.

`PostText` 런타임 API 및 다음 단계에 나오는 요청 및 응답의 추가 세부 정보에 대한 자세한 내용은 PostText 섹션을 참조하십시오.

1. 사용자: 꽃을 주문하고 싶습니다.

   a. 클라이언트(콘솔)는 Amazon Lex에 다음 PostText 요청을 보냅니다.

```
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": {}
}
```

   요청 URI 및 본문 모두 Amazon Lex에 다음 정보를 제공합니다.

   - 요청 URI – 봇 이름(`OrderFlowers`), 봇 별칭(`$LATEST`), 사용자 이름(사용자를 식별하는 임의 문자열)을 제공합니다. 후행 `text`는 이것이 `PostText` API 요청(`PostContent`가 아님)임을 나타냅니다.
   - 요청 본문 – 사용자 입력(`inputText`)과 빈 `sessionAttributes`를 포함합니다. 클라이언트가 첫 번째 요청을 한 경우 세션 속성이 없습니다. Lambda 함수는 이를 나중에 초기화합니다.

b. `inputText`에서 Amazon Lex는 의도(`OrderFlowers`)를 감지합니다. 이 의도는 사용자 데이터 초기화 및 검증을 위한 코드 후크인 Lambda 함수로 구성됩니다. 따라서 Amazon Lex는 다음 정보를 이벤트 데이터로 전달하여 해당 Lambda 함수를 간접 호출합니다.

```
{
    "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"
    }
}
```

자세한 내용은 입력 이벤트 형식을 참조하세요.

클라이언트가 보낸 정보 외에도 Amazon Lex에는 다음과 같은 데이터가 추가로 포함되어 있습니다.

- `messageVersion` - Amazon Lex는 현재 1.0 버전만 지원합니다.
- `invocationSource` – Lambda 함수 간접 호출의 목적을 나타냅니다. 이 경우에 목적은 사용자 데이터 초기화 및 검증을 수행하는 것입니다. 이때 Amazon Lex는 사용자가 의도를 이행하기 위한 모든 슬롯 데이터를 제공하지 않았음을 압니다.
- 모든 슬롯 값이 null로 설정된 `currentIntent` 정보

c. 이때 모든 슬롯 값은 null입니다. Lambda 함수가 검증할 것이 없습니다. Lambda 함수는 Amazon Lex에 다음 응답을 반환합니다.

```
{
```

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

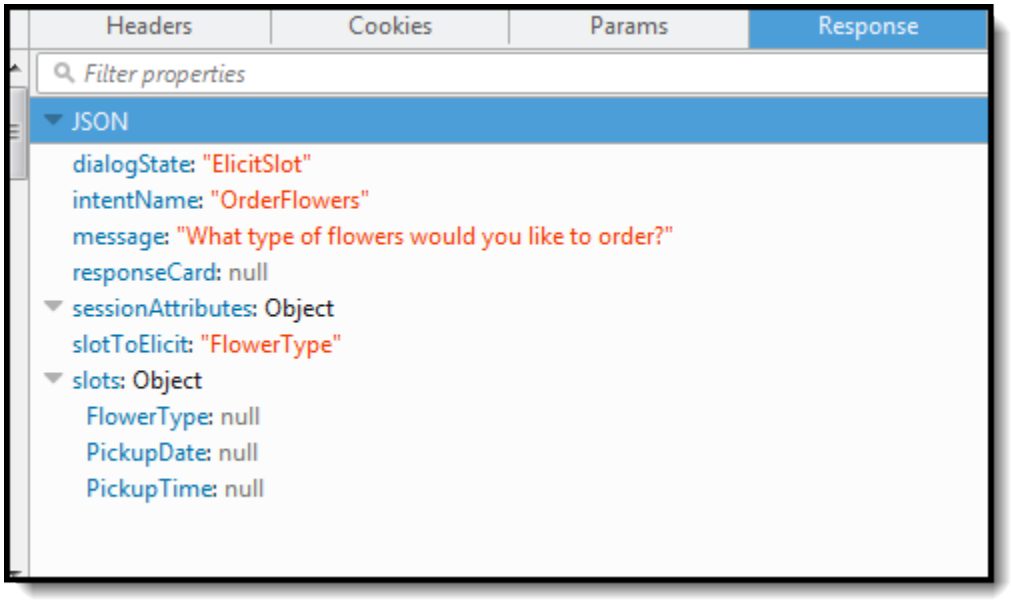
응답 형식에 대한 자세한 내용은 응답 형식 섹션을 참조하십시오.

다음을 참고합니다.

- `dialogAction.type` – Lambda 함수는 이 값을 `Delegate`로 설정하여 Amazon Lex에 일련의 다음 조치를 결정할 책임을 위임합니다.

> **Note**
>
> Lambda 함수가 사용자 데이터 검증에서 무언가를 감지하면, 다음 단계에 나와 있듯이 Amazon Lex에 다음에 해야 할 조치를 지시합니다.

d. `dialogAction.type`에 따라 Amazon Lex는 일련의 다음 조치를 결정합니다. 슬롯이 하나도 채워지지 않았기 때문에 `FlowerType` 슬롯의 값을 유도하기로 결정합니다. 이 슬롯의 값 유도 프롬프트 중 하나("어떤 꽃을 주문하고 싶으세요?")를 선택하고, 클라이언트에 다음 응답을 다시 보냅니다.



클라이언트는 응답에 메시지를 표시합니다.

2. 사용자: 장미

   a. 클라이언트는 Amazon Lex에 다음 PostText요청을 보냅니다.

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

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

   요청 본문에서 `inputText`는 사용자 입력입니다. `sessionAttributes`는 비어있습니다.

   b. Amazon Lex는 먼저 현재 의도의 컨텍스트에서 `inputText`를 해석합니다. 이 서비스는 특정 사용자에게 `FlowerType` 슬롯에 대한 정보를 요청했음을 기억합니다. 현재 의도의 슬롯 값을 업데이트하고, 다음 이벤트 데이터로 Lambda 함수를 간접 호출합니다.

```
{
    "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"
    }
}
```

다음 사항에 유의하세요.

- `invocationSource` – 계속 `DialogCodeHook`입니다(단순히 사용자 데이터를 검증하는 중).
- `currentIntent.slots` – Amazon Lex가 `FlowerType` 슬롯을 장미로 업데이트했습니다.

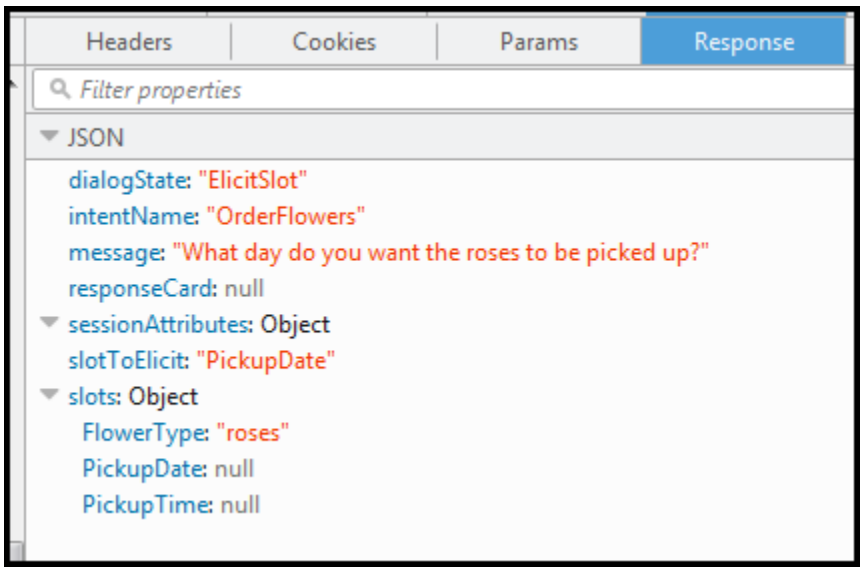
c. `DialogCodeHook`의 `invocationSource` 값에 따라 Lambda 함수는 사용자 데이터 검증을 수행합니다. `roses`를 유효한 슬롯 값으로 인식하고(그리고 `Price`를 세션 속성으로 설정), Amazon Lex에 다음 응답을 반환합니다.

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

다음 사항에 유의하세요.

- `sessionAttributes` – Lambda 함수가 (장미의) `Price`를 세션 속성으로 추가했습니다.
- `dialogAction.type` – `Delegate`로 설정됩니다. 사용자 데이터가 유효했으므로 Lambda 함수는 Amazon Lex에 일련의 다음 조치를 선택하라고 지시합니다.

d. `dialogAction.type`에 따라 Amazon Lex는 일련의 다음 조치를 선택합니다. Amazon Lex는 더 많은 슬롯 데이터가 필요함을 알고 의도 구성에 따라 우선 순위가 가장 높은 채워지지 않은 다음 슬롯(`PickupDate`)을 선택합니다. 의도 구성에 따라 Amazon Lex 이 슬롯에 대한 프롬프트 메시지 중 하나("언제 체크인 예정인가요?")를 선택하고, 클라이언트에 다음 응답을 다시 보냅니다.



클라이언트는 응답에 단순히 "언제 장미를 픽업하고 싶으세요?" 메시지를 표시합니다.

3. 사용자: 내일

a. 클라이언트는 Amazon Lex에 다음 PostText 요청을 보냅니다.

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

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

```
}
```

요청 본문에서 inputText는 사용자 입력이며, 클라이언트는 서비스에 다시 세션 속성을 전달합니다.

b. Amazon Lex는 컨텍스트를 기억합니다. 이를 통해 PickupDate 슬롯에 대한 데이터를 유도했습니다. 이 컨텍스트에서 inputText 값이 PickupDate 슬롯에 대한 것임을 압니다. 그런 다음 Amazon Lex는 다음 이벤트를 보내 Lambda 함수를 간접 호출합니다.

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

Amazon Lex는 currentIntent.slots 값을 설정하여 PickupDate를 업데이트했습니다. 또한 이 서비스는 sessionAttributes를 그대로 Lambda 함수에 전달합니다.

c. DialogCodeHook의 invocationSource값에 따라 Lambda 함수는 사용자 데이터 검증을 수행합니다. PickupDate 슬롯 값이 유효함을 인식하고 Amazon Lex에 다음 응답을 반환합니다.

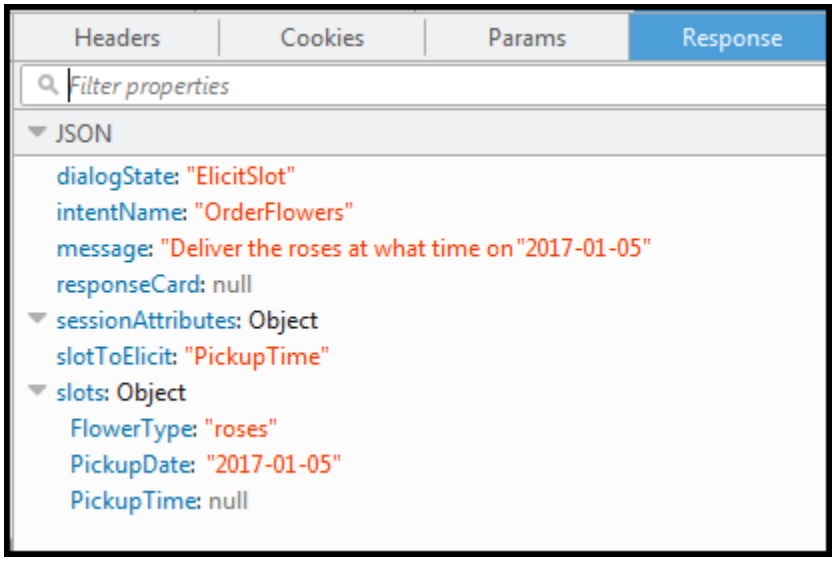
```
{
    "sessionAttributes": {
```

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

다음 사항에 유의하세요.

- `sessionAttributes` – 변경 사항이 없습니다.
- `dialogAction.type` – `Delegate`로 설정됩니다. 사용자 데이터가 유효했으므로 Lambda 함수는 Amazon Lex에 일련의 다음 조치를 선택하라고 지시합니다.

d. `dialogAction.type`에 따라 Amazon Lex는 일련의 다음 조치를 선택합니다. Amazon Lex는 더 많은 슬롯 데이터가 필요함을 알고 의도 구성에 따라 우선 순위가 가장 높은 채워지지 않은 다음 슬롯(`PickupTime`)을 선택합니다. Amazon Lex는 프롬프트 메시지 중 하나를 선택합니다 ("2017-01-05 몇 시에 장미를 배송하시겠습니까?") 이 슬롯의 경우 인텐트 구성에 따라 다음 응답을 클라이언트에 다시 보냅니다.



클라이언트는 응답에 "2017-01-05 몇 시에 장미를 배송하시겠습니까?" 메시지를 표시합니다.

4. 사용자: 4 pm

a. 클라이언트는 Amazon Lex에 다음 PostText 요청을 보냅니다.

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

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

요청 본문에서 `inputText`는 사용자 입력입니다. 클라이언트는 요청에서 `sessionAttributes`를 전달합니다.

b. Amazon Lex는 컨텍트스를 이해합니다. `PickupTime` 슬롯에 대한 데이터를 유도하고 있었음을 이해합니다. 이 컨텍스트에서 `inputText` 값이 `PickupTime`슬롯에 대한 것임을 압니다. 그런 다음 Amazon Lex는 다음 이벤트를 보내 Lambda 함수를 간접 호출합니다.

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

Amazon Lex는 `PickupTime` 값을 설정하여 `currentIntent.slots`를 업데이트했습니다.

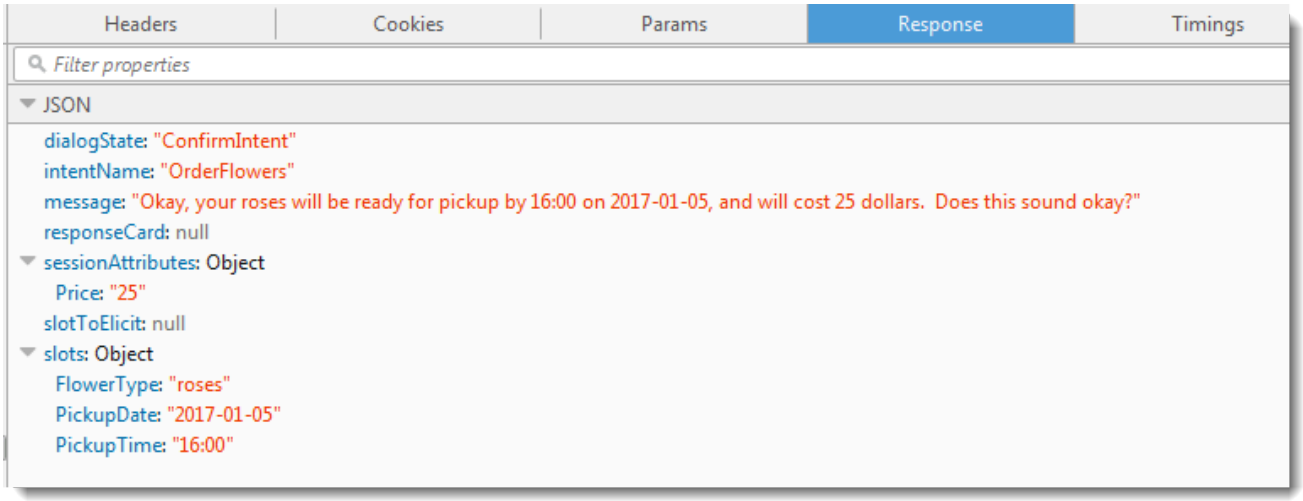
c. `DialogCodeHook`의 `invocationSource` 값에 따라 Lambda 함수는 사용자 데이터 검증을 수행합니다. `PickupDate` 슬롯 값이 유효함을 인식하고 Amazon Lex에 다음 응답을 반환합니다.

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

다음 사항에 유의하세요.

- `sessionAttributes` – 세션 속성의 변경 사항이 없습니다.
- `dialogAction.type` – `Delegate`로 설정됩니다. 사용자 데이터가 유효했으므로 Lambda 함수는 Amazon Lex에 일련의 다음 조치를 선택하라고 지시합니다.

d. 이때 Amazon Lex는 슬롯 데이터가 모두 있음을 압니다. 이 의도는 확인 프롬프트로 구성됩니다. 따라서 Amazon Lex는 의도를 이행하기 전에 확인을 요청하는 다음 응답을 사용자에게 보냅니다.



클라이언트는 단순히 응답에 메시지를 표시하고 사용자 응답을 기다립니다.

5. 사용자: 예

   a. 클라이언트는 Amazon Lex에 다음 PostText 요청을 보냅니다.

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

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

   b. Amazon Lex는 현재 의도를 확인하는 컨텍스트에서 `inputText`를 해석합니다. Amazon Lex는 사용자가 주문을 진행하기를 원한다고 이해합니다. 이때 Amazon Lex는 다음 이벤트를 보내 의도를 이행하도록 Lambda 함수를 간접 호출합니다. 함수에 보내는 이벤트에서 `invocationSource`를 `FulfillmentCodeHook`로 설정합니다. Amazon Lex는 또한 다음과 같이 `confirmationStatus`을 `Confirmed`로 설정합니다.

```
{
    "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"
    }
}
```

다음 사항에 유의하세요.

- `invocationSource` – 이때 Amazon Lex는 이 값을 `FulfillmentCodeHook`로 설정하여 의도를 이행하도록 Lambda 함수에 지시합니다.
- `confirmationStatus` – `Confirmed`로 설정됩니다.

c. 이때 Lambda 함수는 `OrderFlowers` 의도를 이행하고, 다음 응답을 반환합니다.

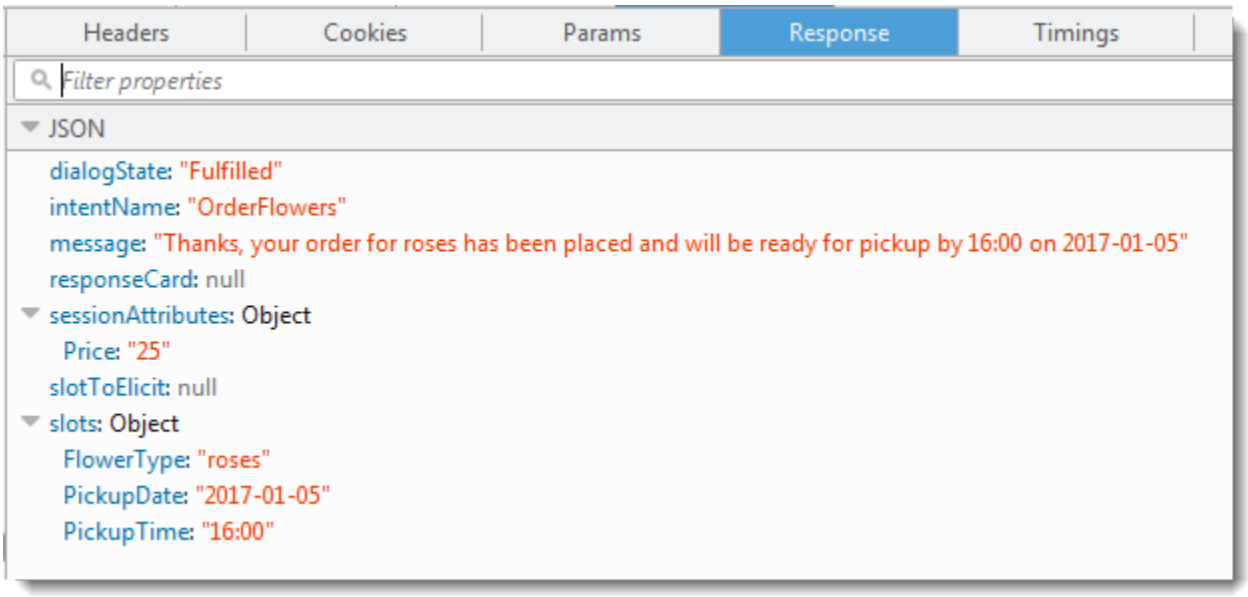
```
{
    "sessionAttributes": {
        "Price": "25"
    },
    "dialogAction": {
        "type": "Close",
        "fulfillmentState": "Fulfilled",
        "message": {
            "contentType": "PlainText",
            "content": "Thanks, your order for roses has been placed and will
 be ready for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

다음 사항에 유의하세요.

- `dialogAction.type` 설정 – Lambda 함수는 이 값을 `Close`로 설정하여 사용자 응답을 기대하지 않도록 Amazon Lex에 지시합니다.
- `dialogAction.fulfillmentState` – Fulfilled로 설정되고, 사용자에게 전달할 적절한 `message`를 포함합니다.

d. Amazon Lex는 `fulfillmentState`를 검토하고 클라이언트에 다음 응답을 다시 보냅니다.

그런 다음 Amazon Lex는 클라이언트에 다음을 반환합니다.



참고:

- `dialogState` – Amazon Lex는 이 값을 `fulfilled`로 설정합니다.
- `message` – Lambda 함수가 제공한 동일한 메시지입니다.

클라이언트가 이 메시지를 표시합니다.

6. 이제 봇을 다시 테스트합니다. 새 (사용자) 컨텍스트를 설정하려면 테스트 창에서 지우기 링크를 선택합니다. 이제 `OrderFlowers` 의도에 대한 유효하지 않은 슬롯 데이터를 제공합니다. 이때 Lambda 함수는 데이터 검증을 수행하고, 잘못된 슬롯 데이터 값을 null로 재설정한 다음, 유효한 데이터를 묻는 메시지를 사용자에게 표시하라고 Amazon Lex에 요청합니다. 예를 들어, 다음을 시도합니다.

- 재스민을 꽃 유형으로 입력(재스민은 지원되는 꽃 유형이 아님)
- 어제를 꽃을 찾아갈 날로 입력

- 주문 후 "예"를 회신하는 대신에 다른 꽃 유형을 입력하여 주문을 확인. 이에 대한 응답으로 Lambda 함수는 세션 속성의 `Price`를 업데이트하여 꽃 주문의 누계를 유지합니다.

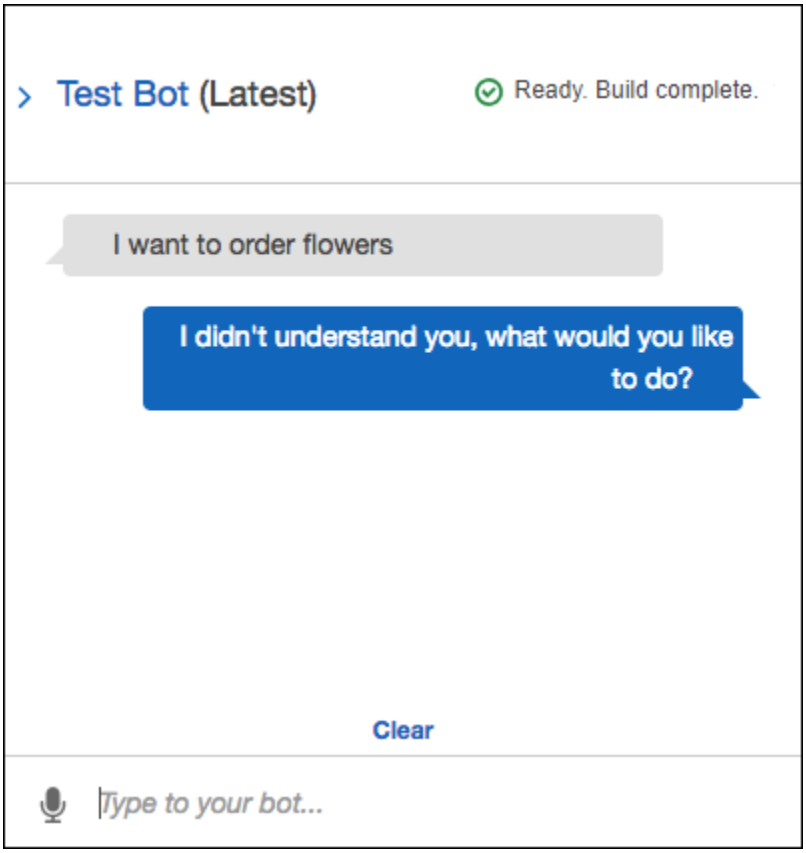
Lambda 함수는 이행 활동도 수행합니다.

다음 단계

[6단계: 표현을 추가하도록 의도 구성 업데이트(콘솔)]

6단계: 표현을 추가하도록 의도 구성 업데이트(콘솔)

`OrderFlowers` 봇은 표현 두 개로만 구성되어 있습니다. 따라서 이 봇은 에서 사용자의 의도를 인식해 응답하는 기계 학습 모델을 구축하는 데 제한된 정보를 제공합니다. 다음 테스트 창에서처럼 “꽃을 주문하고 싶어요”라고 입력해 보십시오. Amazon Lex는 텍스트를 인식하지 못하고 “이해하지 못했습니다. 어떻게 하시겠습니까?”라고 응답합니다. 표현을 추가해 기계 학습 모델을 개선할 수 있습니다.

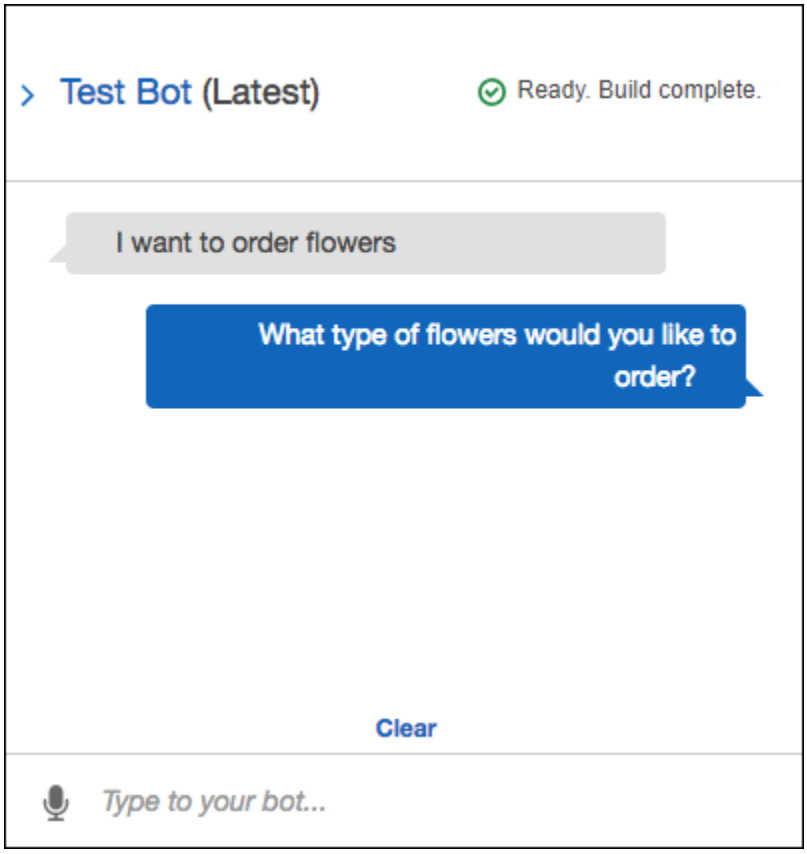


추가한 각 표현은 Amazon Lex에 사용자에게 어떻게 응답할지에 대한 추가 정보를 제공합니다. Amazon Lex는 정확하게 일치하는 입력과 유사한 입력을 둘 다 인식하기 위해 사용자가 제공한 샘플로부터 일반화하기 때문에 정확한 표현을 추가할 필요가 없습니다.

표현을 추가하려면(콘솔)

1. 의도 편집기의 샘플 표현 섹션에 “꽃을 원해요"라고 입력한 다음 새 표현 옆에 있는 더하기 아이콘을 클릭하여 의도에 표현을 추가합니다.

2. 변경 사항을 적용하도록 봇을 구축합니다. 빌드를 선택한 후 빌드를 다시 선택합니다.
3. 봇을 테스트해 새 표현을 인식하는지 확인합니다. 다음 테스트 창에서처럼 “꽃을 주문하고 싶어요”라고 입력해 보십시오. Amazon Lex는 이 문구를 인식하고 “어떤 종류의 꽃을 주문하시겠습니까?” 로 응답합니다.



다음 단계

7단계(선택 사항): 정리(콘솔)

7단계(선택 사항): 정리(콘솔)

이제 생성한 리소스를 삭제하고 계정을 정리합니다.

사용하지 않는 리소스만 삭제할 수 있습니다. 일반적으로 다음 순서로 리소스를 삭제해야 합니다.

- 봇을 삭제하여 의도 리소스를 확보합니다.
- 의도를 삭제하여 슬롯 유형 리소스를 확보합니다.
- 마지막으로 슬롯 유형을 삭제합니다.

계정을 정리하려면(콘솔)

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lex/ Amazon Lex 콘솔을 엽니다.
2. 봇 목록에서 OrderFlowers 옆에 있는 체크박스를 선택합니다.
3. 봇을 삭제하려면 삭제를 선택한 후, 확인 대화 상자에서 계속를 선택합니다.
4. 왼쪽 창에서 의도를 선택합니다.
5. 의도 목록에서 OrderFlowersIntent를 선택합니다.
6. 의도를 삭제하려면 삭제를 선택한 후, 확인 대화 상자에서 계속을 선택합니다.
7. 왼쪽 창에서 슬롯 유형을 선택합니다.
8. 슬롯 유형 목록에서 Flowers를 선택합니다.
9. 슬롯 유형을 삭제하려면 삭제를 선택한 후, 확인 대화 상자에서 계속를 선택합니다.

생성한 모든 Amazon Lex 리소스를 제거하고 계정을 정리했습니다. 원한다면, 콘솔을 사용해 이 연습에 사용되는 Lambda 함수를 삭제할 수 있습니다.

연습 2: 사용자 지정 Amazon Lex 봇 생성

이 연습에서는 Amazon Lex 콘솔을 사용하여 피자를 주문하는 사용자 지정 봇(`OrderPizzaBot`)을 생성합니다. 사용자 지정 의도(`OrderPizza`)를 추가하거나, 사용자 지정 슬롯 유형을 정의하거나, 피자 주문을 이행하는 데 필요한 슬롯(피자 크러스트, 크기 등)을 정의하여 봇을 구성합니다. 슬롯 유형 및 슬롯에 대한 자세한 내용은 Amazon Lex: 작동 방식을 참조하십시오 .

주제

- 1단계: Lambda 함수 생성
- 2단계: 봇 생성
- 3단계: 봇 구축 및 테스트
- 4단계(선택 사항): 정리

1단계: Lambda 함수 생성

먼저 피자 주문을 이행하는 Lambda 함수를 생성합니다. 다음 섹션에서 생성할 함수를 Amazon Lex 봇에서 지정합니다.

Lambda 함수 생성하려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lambda/ AWS Lambda 콘솔을 엽니다.
2. 함수 생성(Create function)을 선택합니다.
3. 함수 생성 페이지에서 처음부터 새로 작성을 선택합니다.

   이 연습에서 제공된 사용자 지정 코드를 사용해 Lambda 함수를 생성할 것이므로 처음부터 함수 작성자를 선택합니다.

   다음 내용을 따르세요.

   a. 이름(`PizzaOrderProcessor`)을 입력합니다.
   b. 런타임에서 최신 버전의 Node.js를 선택합니다.
   c. 역할의 경우, 템플릿에서 새 역할 생성을 선택합니다.
   d. 새 역할 이름(`PizzaOrderProcessorRole`)을 입력합니다.
   e. 함수 생성(Create function)을 선택합니다.
4. 함수 페이지에서 다음을 수행합니다.

   함수 코드 섹션에서 코드 인라인 편집을 선택한 후 다음의 Node.js 함수 코드를 복사하여 창에 붙여넣습니다.

```
'use strict';

// Close dialog with the customer, reporting fulfillmentState of Failed or
 Fulfilled ("Thanks, your pizza will arrive in 20 minutes")
```

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

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

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

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

}

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

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

5. 저장을 선택합니다.

샘플 이벤트 데이터를 사용하여 Lambda 함수 테스트

콘솔에서 샘플 이벤트 데이터를 사용하여 Lambda 함수를 수동으로 호출해 해당 함수를 테스트합니다.

Lambda 함수를 테스트하려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lambda/ AWS Lambda 콘솔을 엽니다.
2. Lambda 함수 페이지에서 Lambda 함수(`PizzaOrderProcessor`).를 선택합니다.
3. 함수 페이지의 테스트 이벤트 목록에서 테스트 이벤트 구성을 선택합니다.
4. 테스트 이벤트 구성 페이지에서 다음 작업을 수행하십시오.
   a. 새로운 테스트 이벤트 생성을 선택하세요.
   b. 이벤트 이름 필드에 이벤트 이름(`PizzaOrderProcessorTest`)을 입력합니다.
   c. 다음 Amazon Lex 이벤트를 창에 복사합니다.

```
{
  "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. 생성(Create)을 선택합니다.

AWS Lambda 가 테스트를 생성하고 함수 페이지로 돌아갑니다. 테스트를 선택하면 Lambda가 Lambda 함수를 실행합니다.

결과 상자에서 세부 정보를 선택합니다. 콘솔이 실행 결과 창에 다음 출력을 표시합니다.

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

다음 단계

2단계: 봇 생성

2단계: 봇 생성

이 단계에서는 피자 주문을 처리하는 봇을 생성합니다.

주제

- 봇 생성
- 의도 생성
- 슬롯 유형 생성
- 의도 구성
- 봇 구성하기

봇 생성

필요한 최소한의 정보를 사용하여 `PizzaOrderingBot` 봇을 생성합니다. 의도를 추가합니다. 의도는 봇에 대해 나중에 사용자가 수행하고자 하는 작업입니다.

봇을 생성하려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lex/ Amazon Lex 콘솔을 엽니다.

2. 봇을 생성합니다.

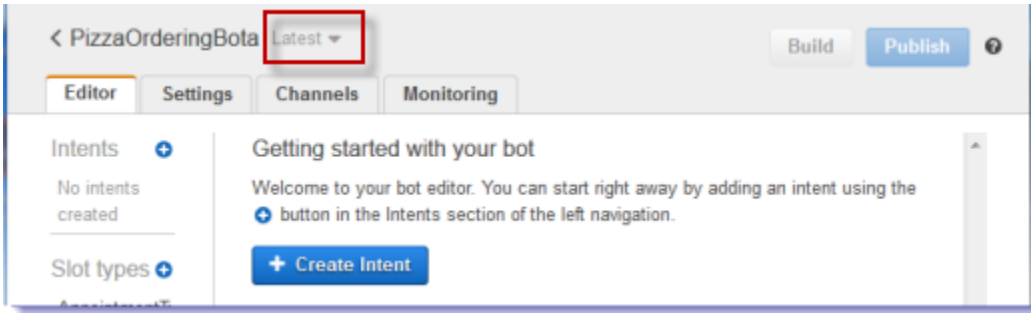
   a. 첫 번째 봇을 생성하는 경우 시작하기를 선택합니다. 그렇지 않은 경우 봇, 생성을 차례로 선택합니다.

   b. Lex 봇 생성 페이지에서 사용자 지정 봇을 선택하고 다음 정보를 입력합니다.

      - 봇 이름: PizzaOrderingBot
      - 언어: 봇의 언어와 로캘을 선택합니다.
      - 출력 음성: Salli
      - 세션 제한 시간: 5분.
      - COPPA: 적절한 응답을 선택하세요.
      - 사용자 표현 스토리지: 적절한 응답을 선택하십시오.

   c. 생성(Create)을 선택합니다.

      콘솔은 Amazon Lex에 새 봇을 생성하라는 요청을 보냅니다. Amazon Lex는 봇 버전을 $LATEST로 설정합니다. 봇을 생성한 후 Amazon Lex는 다음 이미지와 같이 봇 편집기 탭을 표시합니다.



      - 봇 버전 최신은 콘솔에서 봇 이름 옆에 표시됩니다. 새로운 Amazon Lex 리소스의 버전은 $LATEST입니다. 자세한 내용은 버전 관리 및 별칭을 참조하세요.
      - 의도 또는 슬롯 유형을 생성하지 않았으므로 아무 것도 표시되지 않습니다.
      - 빌드 및 게시는는 봇 수준 작업입니다. 전체 봇을 구성한 후에 이러한 작업에 대해 자세히 알아보겠습니다.

다음 단계

의도 생성

의도 생성

이제 사용자가 수행하려는 작업인 `OrderPizza` 의도를 필요한 최소한의 정보를 사용해 생성합니다. 의도에 대한 슬롯 유형을 추가한 후 나중에 의도를 구성합니다.

의도를 생성하려면

1. Amazon Lex 콘솔에서 의도 옆의 더하기 기호(+)를 선택한 후 새 의도 생성을 선택합니다.
2. 의도 생성 대화 상자에 의도 이름(`OrderPizza`)을 입력한 후, 추가를 선택합니다.

그러면 콘솔이 `OrderPizza` 의도를 생성하도록 Amazon Lex에 요청을 보냅니다. 이 예제에서 슬롯 유형을 생성한 다음 의도에 맞는 슬롯을 생성합니다.

다음 단계

슬롯 유형 생성

슬롯 유형 생성

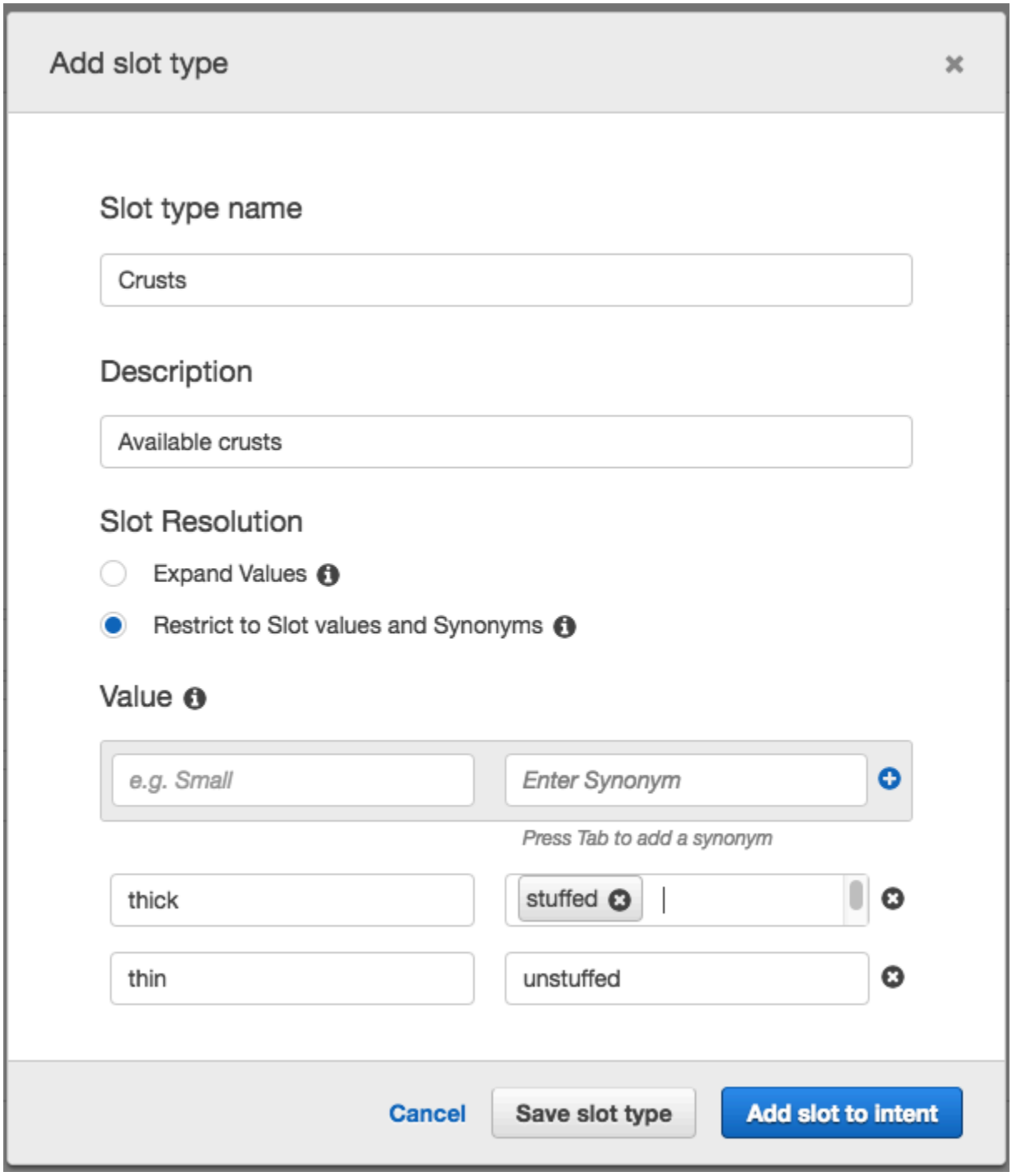
슬롯 유형을 생성하려면

`OrderPizza` 의도에서 사용하는 슬롯 유형 또는 파라미터 값을 생성합니다.

슬롯 유형을 생성하려면

1. 왼쪽 메뉴에서 슬롯 유형 옆의 더하기 기호(+)를 선택합니다.
2. 슬롯 유형 추가 대화 상자에서 다음을 추가합니다.
   - 슬롯 유형 이름 – 크러스트
   - 설명 – 선택 가능한 크러스트
   - 슬롯 값 및 동의어로 제한 선택
   - 값 – **thick**를 입력합니다. 탭을 누르고 동의어 필드에 **stuffed**를 입력합니다. 더하기 기호를 선택합니다 (+). **thin**을 입력한 후 더하기 기호(+)를 다시 선택합니다.

   대화 상자는 다음과 같아야 합니다.



3. 의도에 슬롯 추가를 선택합니다.
4. 의도 페이지에서 필수를 선택합니다. **`slotOne`**에서 **`crust`**로 슬롯 이름을 변경합니다. 프롬프트를 **`What kind of crust would you like?`**로 변경합니다.
5. 다음 표의 값을 사용하여 Step 1~Step 4를 반복합니다.

명칭	설명	값	슬롯 이름	프롬프트
크기	선택 가능한 크기	스몰, 미디엄, 라지	크기	피자 크기는?
PizzaKind	선택 가능한 피자	야채, 치즈	pizzaKind	야채 혹은 치즈 피자를 원하나요?

다음 단계

[의도 구성](#)

의도 구성

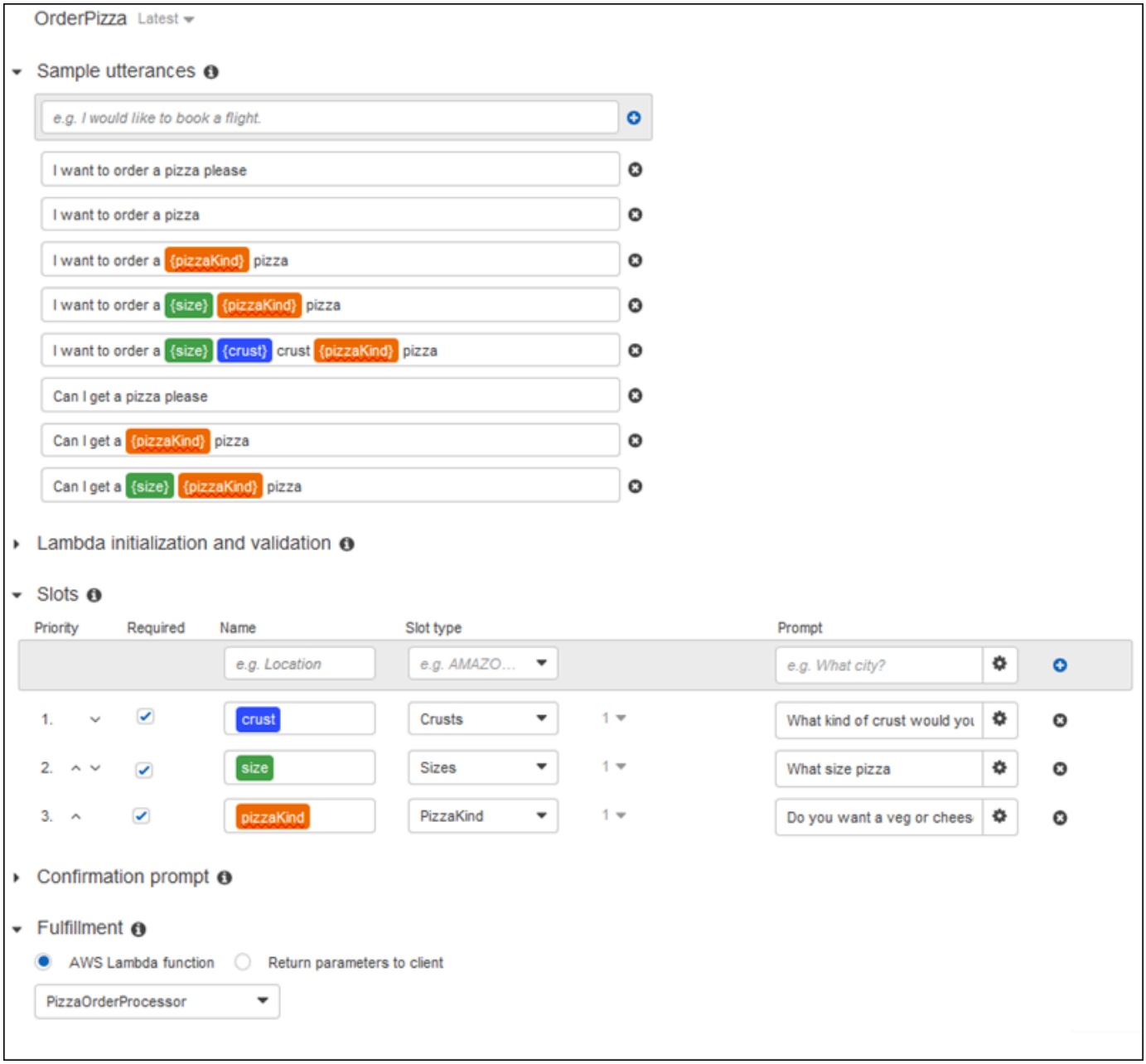
사용자의 피자 주문 요청을 이행하도록 `OrderPizza` 의도를 구성합니다.

의도를 구성하려면

- OrderPizza 구성 페이지에서 다음과 같이 의도를 구성합니다.
  - 샘플 표현 – 다음 문자열을 입력합니다. 중괄호 {}로 슬롯 이름을 묶습니다.
    - 피자 주문을 할 수 있을까요
    - 피자 주문하고 싶어요
    - {pizzaKind} 피자를 주문하고 싶어요
    - {size} 크기의 {pizzaKind} 피자를 주문하고 싶어요
    - {size} 크기의 {crust} 크러스트 {pizzaKind} 피자 주문할게요
    - 피자를 먹을 수 있을까요
    - {pizzaKind} 피자를 먹을 수 있을까요
    - {size} 크기의 {pizzaKind} 피자를 먹을 수 있을까요
  - Lambda 초기화 및 검증 – 기본 설정을 그대로 둡니다.
  - 확인 프롬프트 – 기본 설정을 그대로 둡니다.
  - 이행 – 아래의 작업을 수행합니다.
    - AWS Lambda 함수를 선택합니다.
    - **`PizzaOrderProcessor`**를 선택합니다.

- Lambda 함수에 권한 추가 대화 상자가 표시되면, 확인을 선택하여 `OrderPizza` 의도에 `PizzaOrderProcessor` Lambda 함수를 호출할 권한을 부여합니다.
- 없음이 선택된 상태로 둡니다.

이 의도는 다음과 같아야 합니다.



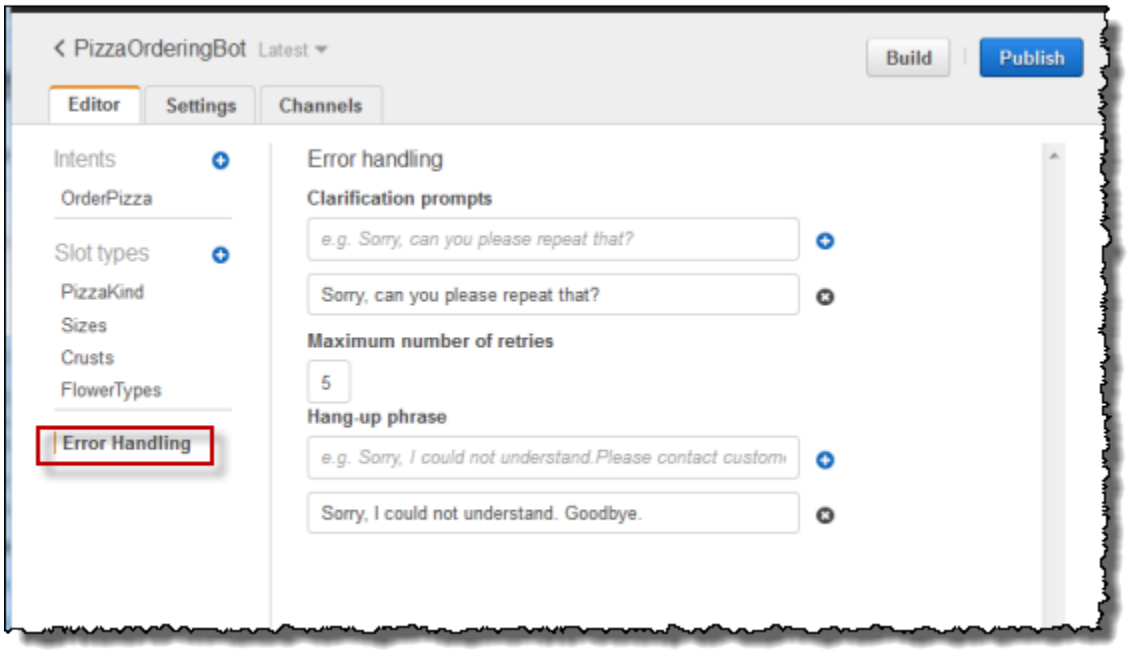
다음 단계

[봇 구성하기]

봇 구성하기

`PizzaOrderingBot` 봇에 대한 오류 처리를 구성합니다.

1. `PizzaOrderingBot` 봇으로 이동합니다. 편집기를 선택한 다음, 다음 이미지와 같이 오류 처리를 선택합니다.



2. 편집기 탭을 사용하여 봇 오류 처리를 구성합니다.

   - 명료화 프롬프트에 입력한 정보가 봇의 clarificationPrompt 구성으로 매핑됩니다.

     Amazon Lex에서 사용자 의도를 확인할 수 없는 경우, 이 서비스는 이 메시지가 포함된 응답을 반환합니다.

   - 중단 문구에 입력한 정보가 봇의 abortStatement 구성으로 매핑됩니다.

     이 서비스에서 정해진 연속 요청 이후 사용자의 의도를 확인할 수 없는 경우, Amazon Lex는 이 메시지가 포함된 응답을 반환합니다.

   기본값을 그대로 둡니다.

다음 단계

3단계: 봇 구축 및 테스트

3단계: 봇 구축 및 테스트

봇을 구축한 다음 테스트해 작동하는지 확인합니다.

봇을 구축 및 테스트하려면

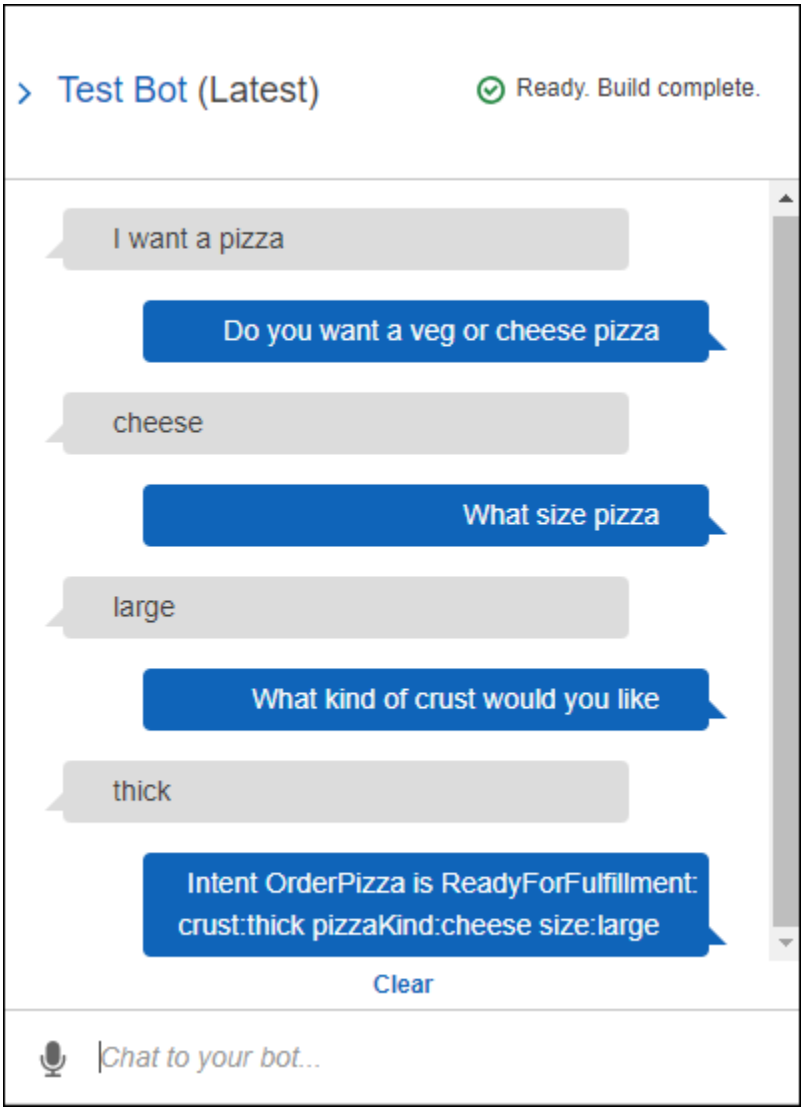
1. `PizzaOrderingBot` 봇을 구축하려면 빌드를 선택합니다.

   Amazon Lex가 봇을 위한 기계 학습 모델을 구축합니다. 콘솔에서 봇을 테스트할 경우, 콘솔은 런타임 API를 사용하여 사용자 입력을 Amazon Lex에 되돌려 보냅니다. 그 뒤에 Amazon Lex 는 기계 학습 모델을 사용하여 사용자 입력을 해석합니다.

   구축을 완료하는 데 다소 시간이 걸릴 수 있습니다.

2. Amazon Lex 봇을 테스트하려면 봇 테스트 창에서 봇과의 통신을 시작합니다.

   - 예를 들어 다음과 같이 말하거나 입력할 수 있습니다.



- `OrderPizza` 의도에서 구성한 샘플 표현을 사용해 봇을 테스트합니다. 예를 들어, 다음은 사용자가 `PizzaOrder` 의도에 대해 구성한 샘플 표현 중 하나입니다.

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

  테스트하려면 다음과 같이 입력합니다.

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

"피자 주문하고 싶어요"라고 입력하면 Amazon Lex가 의도(`OrderPizza`)를 감지합니다. 그러면 Amazon Lex는 슬롯 정보를 묻습니다.

사용자가 모든 슬롯 정보를 제공하면, Amazon Lex는 사용자가 의도에 대해 구성한 Lambda 함수를 호출합니다.

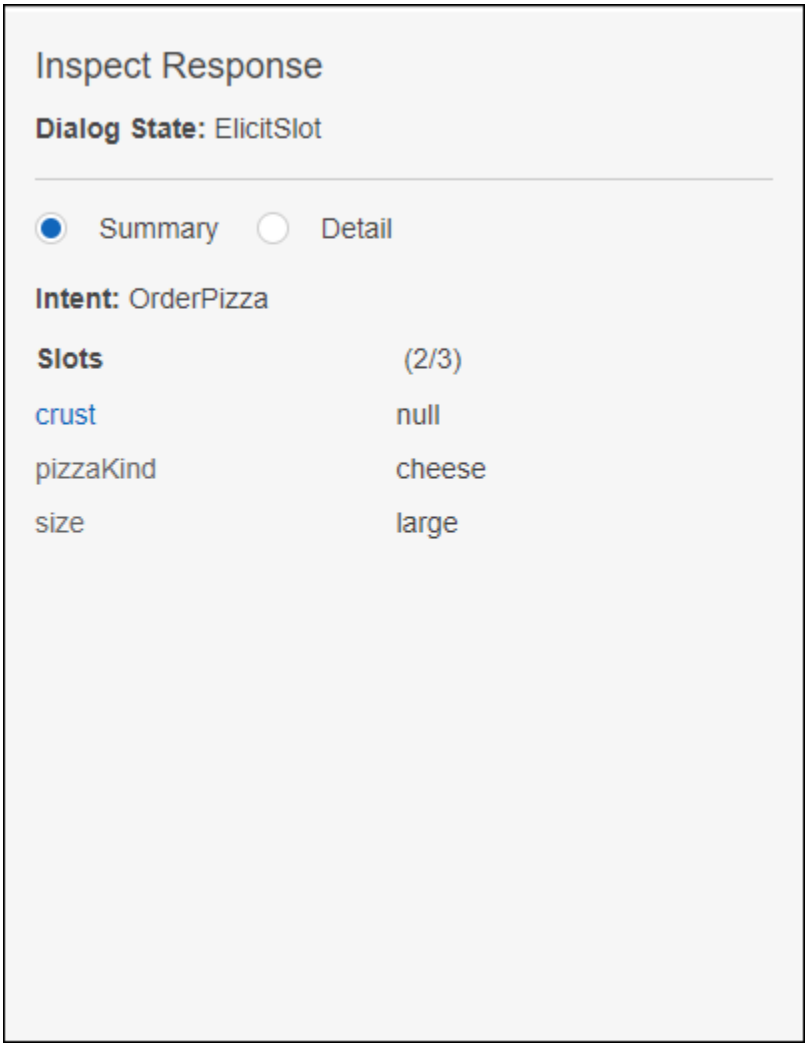
Lambda 함수가 에 메시지("네, 당신의 주문을...")를 반환하고, Amazon Lex는 해당 메시지를 사용자에게 반환합니다.

응답 검사

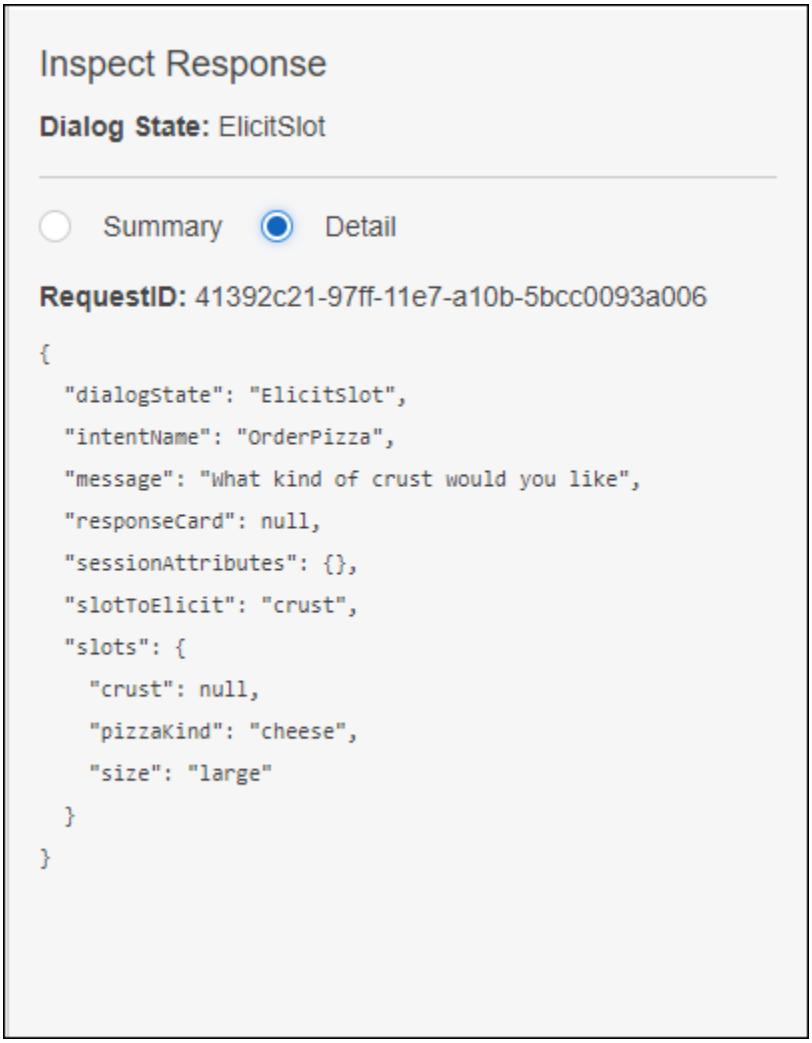
채팅 창 아래에는 Amazon Lex의 응답을 검사할 수 있는 창이 있습니다. 이 창은 봇과의 상호 작용에 따라 변하는 봇의 상태에 대한 포괄적인 정보를 제공합니다. 이 창의 내용은 작업의 현재 상태를 보여 줍니다.

- 대화 상자 상태 – 사용자와의 대화에 대한 현재 상태입니다. `ElicitIntent`, `ElicitSlot`, `ConfirmIntent` 또는 `Fulfilled`일 수 있습니다.

- 요약 – 정보 흐름을 계속 추적할 수 있도록 이행 중인 의도에 대한 슬롯 값을 보여 주는 대화의 단순화된 보기가 표시됩니다. 여기에는 의도 이름, 슬롯 수, 이행된 슬롯 수, 모든 슬롯 및 연결된 값 목록이 표시됩니다. 다음 이미지를 참조하세요.



- 세부 정보 – 챗봇을 테스트 및 디버그함에 따른 대화의 현재 상태와 봇 상호 작용을 더욱 자세히 볼 수 있도록 챗봇의 원시 JSON 응답을 보여 줍니다. 채팅 창에 입력하면 검사 창에 PostText 작업의 JSON 응답이 표시됩니다. 채팅 창에 말하면 검사 창에 PostContent 작업의 응답 헤더가 표시됩니다. 다음 이미지를 참조하세요.



다음 단계

4단계(선택 사항): 정리

4단계(선택 사항): 정리

생성한 리소스에 대해 추가 비용이 발생하지 않도록 생성한 리소스를 삭제하고 계정을 정리합니다.

사용하지 않는 리소스만 삭제할 수 있습니다. 예를 들어 의도에서 참조하는 슬롯 유형은 삭제할 수 없습니다. 봇에서 참조하는 의도는 삭제할 수 없습니다.

다음 순서에 따라 리소스를 삭제합니다.

- 봇을 삭제하여 의도 리소스를 확보합니다.
- 의도를 삭제하여 슬롯 유형 리소스를 확보합니다.

- 마지막으로 슬롯 유형을 삭제합니다.

계정을 정리하려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lex/ Amazon Lex 콘솔을 엽니다.
2. 봇 목록에서 PizzaOrderingBot을 선택합니다.
3. 봇을 삭제하려면 삭제를 선택한 후 계속을 선택합니다.
4. 왼쪽 창에서 의도를 선택합니다.
5. 의도 목록에서 OrderPizza를 선택합니다.
6. 의도를 삭제하려면 삭제를 선택한 후 계속을 선택합니다.
7. 왼쪽 메뉴에서 슬롯 유형를 선택합니다.
8. 슬롯 유형 목록에서 크러스트를 선택합니다.
9. 슬롯 유형을 삭제하려면 삭제를 선택한 후 계속을 선택합니다.
10. `Sizes` 및 `PizzaKind` 슬롯 유형에 대해 Step 8 및 Step 9를 반복합니다.

생성한 모든 리소스를 제거하고 계정을 정리했습니다.

다음 단계

- 버전 게시 및 별칭 생성
- 를 사용하여 Amazon Lex 봇 생성 AWS Command Line Interface

연습 3: 버전 게시 및 별칭 만들기

시작하기 연습 1 및 2에서는 봇을 생성하여 테스트했습니다. 이 연습에서는 다음 작업을 수행합니다.

- 새 버전의 봇을 게시합니다. Amazon Lex가 `$LATEST` 버전의 스냅샷 복사본을 만들어 새 버전을 게시합니다.
- 새 버전을 가리키는 별칭을 생성합니다.

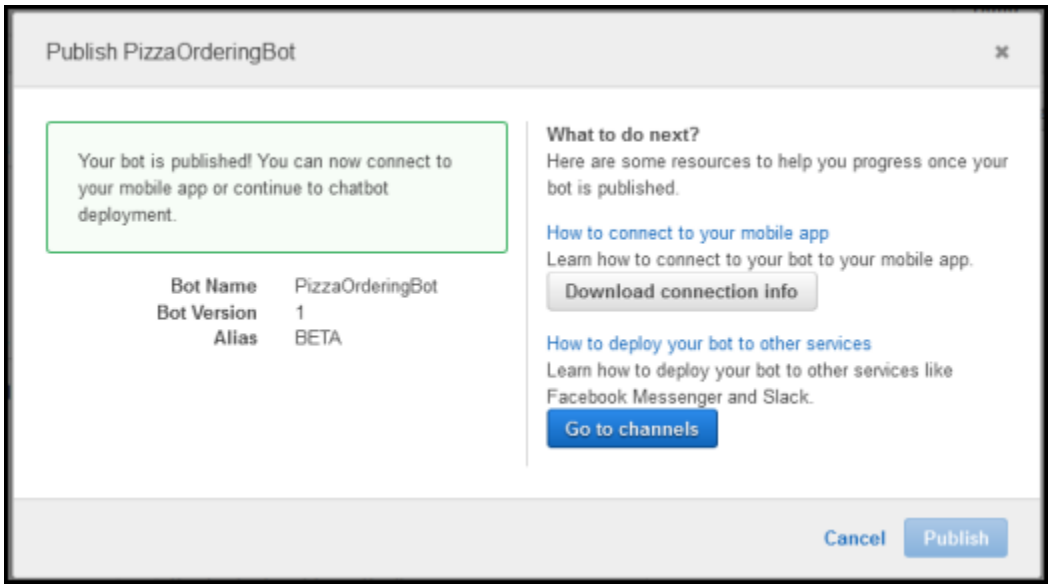
버전 관리 및 별칭에 대한 자세한 내용은 버전 관리 및 별칭을 참조하십시오.

다음을 수행하여 이 실습에서 생성한 봇 버전을 게시합니다.

1. Amazon Lex 콘솔에서, 사용자가 생성한 봇 중 하나를 선택합니다.

   콘솔에서 $LATEST를 봇 이름 옆에 있는 봇 버전으로 표시하는지 확인합니다.

2. 게시를 선택합니다.
3. 게시 **# ##** 마법사에서 별칭 **BETA**를 지정한 다음 게시를 선택합니다.
4. Amazon Lex 콘솔에서 봇 이름 옆에 새로운 버전이 표시되는지 확인합니다.



이제 게시된 버전 및 별칭을 지닌 봇이 사용 가능하므로 이 봇을 배포(모바일 애플리케이션에서 배포하거나 Facebook Messenger에 통합)할 수 있습니다. 관련 예제는 Amazon Lex 봇을 Facebook Messenger와 통합하기을 참조하세요.

4단계: 시작하기(AWS CLI)

이 단계에서는 AWS CLI 를 사용하여 Amazon Lex 봇을 생성, 테스트 및 수정합니다. 다음 연습을 완료하기 위해서는 CLI를 능숙하게 사용할 수 있고 텍스트 편집기가 있어야 합니다. 자세한 내용은 2단계: 설정 AWS Command Line Interface을 참조하세요.

- 연습 1 — Amazon Lex 봇을 만들고 테스트하십시오. 이 연습에서는 사용자 지정 슬롯 유형, 의도 및 봇을 생성하는 데 필요한 JSON 객체를 모두 제공합니다. 자세한 내용은 Amazon Lex: 작동 방식을 참조하세요.
- 연습 2 — 연습 1에서 만든 봇을 업데이트하여 추가 샘플 표현을 추가합니다. Amazon Lex는 샘플 표현을 사용하여 봇을 위한 기계 학습 모델을 구축합니다.

- 연습 3 — 연습 1에서 생성한 봇을 업데이트하여 Lambda 함수를 추가하여 사용자 입력을 검증하고 의도를 이행합니다.
- 연습 4 — 연습 1에서 생성한 슬롯 유형, 의도, 봇 리소스의 버전을 게시하십시오. 버전은 변경할 수 없는 리소스의 스냅샷입니다.
- 연습 5 — 연습 1에서 만든 봇의 별칭을 만드십시오.
- 연습 6 — 연습 1에서 만든 슬롯 유형, 의도, 봇, 연습 5에서 만든 별칭을 삭제하여 계정을 정리합니다.

주제

- 연습 1 — Amazon Lex 봇(AWS CLI)을 만들고 테스트하십시오.
- 연습 2: 새 표현 추가(AWS CLI)
- 연습 3: Lambda 함수 추가(AWS CLI)
- 4단계: 버전 게시(AWS CLI)
- 연습 5: 별칭 만들기(AWS CLI)
- 연습 6: 정리(AWS CLI)

연습 1 — Amazon Lex 봇(AWS CLI)을 만들고 테스트하십시오.

일반적으로 봇을 생성할 때 다음을 수행합니다.

1. 봇이 처리할 정보를 정의하는 슬롯 유형을 생성합니다.
2. 봇이 지원하는 사용자 작업을 정의하는 의도를 생성합니다. 앞서 생성한 사용자 지정 슬롯 유형을 사용하여 의도에 필요한 슬롯 또는 파라미터를 정의합니다.
3. 정의한 의도를 사용하는 봇을 생성합니다.

이 연습에서는 CLI를 사용하여 새 Amazon Lex 봇을 생성하고 테스트합니다. Amazon에서 제공하는 JSON 구조를 사용하여 봇을 생성합니다. 이 연습에서 명령을 실행하려면 명령이 실행될 리전을 알아야 합니다. 리전 목록은 모델 구축 할당량 을 참조하십시오.

주제

- 1단계: 서비스 연결 역할 생성(AWS CLI)
- 2단계: 사용자 지정 슬롯 유형 생성(AWS CLI)
- 3단계: 의도 생성(AWS CLI)

- 4단계: 봇 생성(AWS CLI)
- 5단계: 봇 테스트(AWS CLI)

1단계: 서비스 연결 역할 생성(AWS CLI)

Amazon Lex는 AWS Identity and Access Management 서비스 연결 역할을 수임하여 봇을 대신하여 AWS 서비스를 호출합니다. 사용자의 계정에 속한 역할은 Amazon Lex 사용 사례에 연결되어 사전 정의된 권한을 갖습니다. 자세한 내용은 Amazon Lex에 서비스 연결 역할 사용을 참조하세요.

콘솔을 사용하여 Amazon Lex 봇을 이미 생성한 경우 서비스 연결 역할이 자동으로 생성되었습니다. 2단계: 사용자 지정 슬롯 유형 생성(AWS CLI)로 이동하세요.

서비스 연결 역할을 만들려면(AWS CLI)

1. 에 다음 명령을 AWS CLI입력합니다.

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

2. 다음 명령을 사용하여 정책을 확인합니다.

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

다음과 같이 응답합니다.

```
{
    "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"
}
```

다음 단계

2단계: 사용자 지정 슬롯 유형 생성(AWS CLI)

2단계: 사용자 지정 슬롯 유형 생성(AWS CLI)

주문 가능한 꽃에 대한 열거 값을 사용해 사용자 지정 슬롯 유형을 생성합니다. `OrderFlowers` 의도를 생성하는 경우 다음 단계에서 이 유형을 사용합니다. 슬롯 유형은 의도의 슬롯 또는 파라미터에 가능한 값을 정의합니다.

이 연습에서 명령을 실행하려면 명령이 실행될 리전을 알아야 합니다. 리전 목록은 모델 구축 할당량을 참조하십시오.

사용자 지정 슬롯 유형을 생성하려면(AWS CLI)

1. **`FlowerTypes.json`**라는 이름의 텍스트 파일을 만듭니다. FlowerTypes.json의 JSON 코드를 텍스트 파일에 복사합니다.
2. 를 사용하여 PutSlotType 작업을 호출 AWS CLI 하여 슬롯 유형을 생성합니다. 다음은 Unix, Linux, macOS용 형식으로 지정된 예제입니다. Windows의 경우 각 줄의 끝에 있는 백슬래시(\) Unix 연속 문자를 캐럿(^)으로 바꿉니다.

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

   서버에서 다음과 같이 응답합니다.

   ```
   {
       "enumerationValues": [
           {
               "value": "tulips"
           },
           {
               "value": "lilies"
           },
   ```

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

다음 단계

[3단계: 의도 생성(AWS CLI)]

FlowerTypes.json

다음 코드는 `FlowerTypes` 사용자 지정 슬롯 유형을 생성하는 데 필요한 JSON 데이터입니다.

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

3단계: 의도 생성(AWS CLI)

`OrderFlowersBot` 봇에 대한 의도를 생성하고 세 개의 슬롯 또는 파라미터를 제공합니다. 다음 슬롯은 봇이 의도를 이행하도록 합니다.

- `FlowerType`은 주문 가능한 꽃의 종류를 지정하는 사용자 지정 슬롯 유형입니다.
- `AMAZON.DATE` 및 `AMAZON.TIME`은 사용자로부터 꽃 배송 날짜 및 시간을 얻는 데 사용되는 기본 제공 슬롯 유형입니다.

이 연습에서 명령을 실행하려면 명령이 실행될 리전을 알아야 합니다. 리전 목록은 모델 구축 할당량을 참조하십시오.

`OrderFlowers` 의도를 생성하려면(AWS CLI)

1. **`OrderFlowers.json`**라는 이름의 텍스트 파일을 만듭니다. OrderFlowers.json의 JSON 코드를 텍스트 파일에 복사합니다.
2. 에서 PutIntent 작업을 AWS CLI호출하여 의도를 생성합니다. 다음은 Unix, Linux, macOS용 형식으로 지정된 예제입니다. Windows의 경우 각 줄의 끝에 있는 백슬래시(\) Unix 연속 문자를 캐럿(^)으로 바꿉니다.

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

서버가 다음과 같이 응답합니다.

```
{
    "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"
}
```

다음 단계

4단계: 봇 생성(AWS CLI)

OrderFlowers.json

다음 코드는 `OrderFlowers` 의도를 생성하는 데 필요한 JSON 데이터입니다.

```
{
    "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"
}
```

4단계: 봇 생성(AWS CLI)

`OrderFlowersBot` 봇에는 이전 단계에서 생성한 `OrderFlowers` 의도 하나만 있습니다. 이 연습에서 명령을 실행하려면 명령이 실행될 리전을 알아야 합니다. 리전 목록은 모델 구축 할당량 을 참조하십시오.

> **Note**
>
> 다음 AWS CLI 예제는 Unix, Linux 및 macOS용 형식입니다. Windows의 경우 `"\$LATEST"`를 `$LATEST`로 변경합니다.

`OrderFlowersBot` 봇을 생성하려면(AWS CLI)

1. **`OrderFlowersBot.json`**라는 이름의 텍스트 파일을 만듭니다. OrderFlowersBot.json의 JSON 코드를 텍스트 파일에 복사합니다.
2. 에서 PutBot 작업을 AWS CLI호출하여 봇을 생성합니다. 다음은 Unix, Linux, macOS용 형식으로 지정된 예제입니다. Windows의 경우 각 줄의 끝에 있는 백슬래시(\) Unix 연속 문자를 캐럿(^)으로 바꿉니다.

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

서버에서 다음과 같이 응답합니다. 봇을 생성하거나 업데이트하는 경우 `status` 필드가 `BUILDING`으로 설정됩니다. 이는 봇이 사용할 준비가 되지 않았음을 나타냅니다. 봇이 사용할 준비가 되었는지 확인하려면 다음 단계에서 GetBot 작업을 사용합니다.

```
{
    "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. 새 봇이 사용할 준비가 되었는지 확인하려면 다음 명령을 실행합니다. `status` 필드가 `READY`를 반환할 때까지 이 명령을 반복 실행합니다. 다음은 Unix, Linux, macOS용 형식으로 지정된 예제입니다. Windows의 경우 각 줄의 끝에 있는 백슬래시(\) Unix 연속 문자를 캐럿(^)으로 바꿉니다.

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

응답에서 `status` 필드를 찾습니다.

```
{
    "status": "READY",

    ...

}
```

다음 단계

[5단계: 봇 테스트(AWS CLI)]

OrderFlowersBot.json

다음 코드는 `OrderFlowers` Amazon Lex 봇을 구축하는 데 필요한 JSON 데이터를 제공합니다.

```
{
    "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"
}
```

5단계: 봇 테스트(AWS CLI)

봇을 테스트하기 위해 텍스트 기반 또는 음성 기반 테스트를 사용할 수 있습니다.

주제

- 텍스트 입력을 사용하여 봇 테스트(AWS CLI)
- 음성 입력을 사용하여 봇 테스트(AWS CLI)

텍스트 입력을 사용하여 봇 테스트(AWS CLI)

봇이 텍스트 입력에 올바르게 작동하는지 확인하려면 PostText 작업을 사용합니다. 이 연습에서 명령을 실행하려면 명령이 실행될 리전을 알아야 합니다. 리전 목록은 런타임 서비스 할당량을 참조하십시오.

> **Note**
>
> 다음 AWS CLI 예제는 Unix, Linux 및 macOS용 형식입니다. Windows의 경우 `"\$LATEST"`를 `$LATEST`로 바꾸고, 각 줄의 끝에 있는 백슬래시(\) 연속 문자를 캐럿(^)으로 변경합니다.

텍스트를 사용하여 봇을 테스트하려면(AWS CLI)

1. 에서 `OrderFlowersBot` 봇과의 대화를 AWS CLI시작합니다. 다음은 Unix, Linux, macOS용 형식으로 지정된 예제입니다. Windows의 경우 각 줄의 끝에 있는 백슬래시(\) Unix 연속 문자를 캐럿(^)으로 바꿉니다.

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

   Amazon Lex는 사용자의 의도를 인식하고 다음 응답을 반환하여 대화를 시작합니다.

   ```
   {
       "slotToElicit": "FlowerType",
       "slots": {
           "PickupDate": null,
           "PickupTime": null,
           "FlowerType": null
       },
       "dialogState": "ElicitSlot",
   ```

```
    "message": "What type of flowers would you like to order?",
    "intentName": "OrderFlowers"
}
```

2. 다음 명령을 실행하여 봇과의 대화를 종료합니다.

```
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"
```

주문을 확인하면 Amazon Lex는 이행 응답을 보내 대화를 완료합니다.

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

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

다음 단계

음성 입력을 사용하여 봇 테스트(AWS CLI)

음성 입력을 사용하여 봇 테스트(AWS CLI)

오디오 파일을 사용하여 봇을 테스트하려면 PostContent 작업을 사용합니다. Amazon Polly 텍스트 음성 변환 작업을 사용하여 오디오 파일을 생성합니다.

이 연습에서 명령을 실행하려면 Amazon Lex 및 Amazon Polly 명령이 실행될 리전을 알아야 합니다. Amazon Lex의 지역 목록은 런타임 서비스 할당량을 참조하십시오. Amazon Polly 리전 목록은 Amazon Web Services 일반 참조에서 AWS 리전 및 엔드포인트를 참조하세요.

> **Note**
>
> 다음 AWS CLI 예제는 Unix, Linux 및 macOS용 형식입니다. Windows의 경우 `"\$LATEST"`를 `$LATEST`로 바꾸고, 각 줄의 끝에 있는 백슬래시(\) 연속 문자를 캐럿(^)으로 변경합니다.

음성 입력을 사용하여 봇을 테스트하려면(AWS CLI)

1. 에서 Amazon Polly를 사용하여 오디오 파일을 AWS CLI생성합니다. 다음은 Unix, Linux, macOS용 형식으로 지정된 예제입니다. Windows의 경우 각 줄의 끝에 있는 백슬래시(\) Unix 연속 문자를 캐럿(^)으로 바꿉니다.

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

2. Amazon Lex 에서 오디오 파일을 보내려면 다음 명령을 실행합니다. Amazon Lex는 오디오 응답을 지정된 출력 파일에 저장합니다.

```
aws lex-runtime post-content \
    --region region \
```

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

Amazon Lex는 첫 번째 슬롯에 대한 요청으로 응답합니다. 오디오 응답을 지정된 출력 파일에 저장합니다.

```
{
    "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. 장미를 주문하려면 다음 오디오 파일을 생성하여 Amazon Lex에 보냅니다.

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

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

4. 배달 날짜를 설정하려면 다음 오디오 파일을 만들어 Amazon Lex로 보냅니다.

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

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

5. 배달 날짜를 설정하려면 다음 오디오 파일을 만들어 Amazon Lex 로 보냅니다.

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

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

6. 배달 여부를 확인하려면 다음 오디오 파일을 만들어 Amazon Lex로 보냅니다.

```
aws polly synthesize-speech \
    --region region \
    --output-format pcm \
```

```
    --text "yes" \
    --voice-id "Salli" \
    ConfirmSpeech.mpg
```

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

배달 확인 후에는 Amazon Lex가 의도 이행을 확인하는 응답을 보냅니다.

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

다음 단계

연습 2: 새 표현 추가(AWS CLI)

연습 2: 새 표현 추가(AWS CLI)

Amazon Lex가 사용자의 요청을 인식하는 데 사용하는 기계 학습 모델을 개선하려면 봇에 다른 샘플 표현을 추가합니다.

새 표현 추가는 4단계 프로세스입니다.

1. GetIntent 작업을 사용하여 Amazon Lex 에서 의도를 가져옵니다.
2. 의도를 업데이트합니다.

3. PutIntent 작업을 사용하여 업데이트된 의도를 다시 Amazon Lex로 보냅니다.
4. GetBot 및 PutBot 작업을 사용하여 해당 의도를 사용하는 봇을 다시 구축합니다.

이 연습에서 명령을 실행하려면 명령이 실행될 리전을 알아야 합니다. 리전 목록은 모델 구축 할당량을 참조하십시오.

`GetIntent` 작업의 응답에는 의도의 특정 개정을 식별하는 `checksum`이라는 필드가 포함되어 있습니다. PutIntent 작업을 사용하여 의도를 업데이트하는 경우 체크섬 값을 제공해야 합니다. 체크섬 값을 제공하지 않으면 다음 오류 메시지가 표시됩니다.

```
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**
>
> 다음 AWS CLI 예제는 Unix, Linux 및 macOS용 형식입니다. Windows의 경우 `"\$LATEST"`를 `$LATEST`로 바꾸고, 각 줄의 끝에 있는 백슬래시(\) 연속 문자를 캐럿(^)으로 변경합니다.

`OrderFlowers` 의도를 업데이트하려면(AWS CLI)

1. 에서 Amazon Lex에서 의도를 AWS CLI가져옵니다. Amazon Lex는 출력을 **`OrderFlowers-V2.json`.**라는 파일로 보냅니다.

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

2. 텍스트 편집기에서 **`OrderFlowers-V2.json`**을 엽니다.

   1. `createdDate`, `lastUpdatedDate` 및 `version` 필드를 찾아 삭제합니다.
   2. `sampleUtterances` 필드에 다음을 추가합니다.

      ```
      I want to order flowers
      ```

3. 파일을 저장합니다.

3. 다음 명령을 사용하여 Amazon Lex에 업데이트된 의도를 보냅니다.

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

Amazon Lex가 다음 응답을 보냅니다.

```
{
    "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"
}
```

이제 의도를 업데이트했으므로 해당 의도를 사용하는 봇을 다시 구축합니다.

`OrderFlowersBot` 봇을 다시 구축하려면(AWS CLI)

1. 에서 `OrderFlowersBot` 봇의 정의를 AWS CLI가져와 다음 명령을 사용하여 파일에 저장합니다.

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

2. 텍스트 편집기에서 **`OrderFlowersBot-V2.json`** 파일을 엽니다. `createdDate`, `lastUpdatedDate`, `status` 및 `version` 필드를 제거합니다.
3. 텍스트 편집기에서 봇 정의에 다음 줄을 추가합니다.

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

4. 에서에 대한 다음 명령을 실행하여 봇의 새 개정을 AWS CLI빌드합니다.

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

서버에서 다음과 같이 응답합니다.

```
{
    "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"
}
```

다음 단계

[연습 3: Lambda 함수 추가(AWS CLI)]

연습 3: Lambda 함수 추가(AWS CLI)

사용자 입력을 검증하고 사용자의 의도를 이행하는 Lambda 함수를 봇에 추가합니다.

Lambda 표현식 추가는 5단계 프로세스입니다.

1. AddPermission 함수를 사용하여 `OrderFlowers` 의도가 Lambda 호출 작업을 호출하도록 합니다.
2. GetIntent 작업을 사용하여 Amazon Lex 에서 의도를 가져옵니다.
3. Lambda 함수를 추가하도록 의도를 업데이트합니다.
4. PutIntent 작업을 사용하여 업데이트된 의도를 다시 Amazon Lex로 보냅니다.
5. GetBot 및 PutBot 작업을 사용하여 해당 의도를 사용하는 봇을 다시 구축합니다.

이 연습에서 명령을 실행하려면 명령이 실행될 리전을 알아야 합니다. 리전 목록은 모델 구축 할당량을 참조하십시오.

`InvokeFunction` 권한을 추가하기 전에 의도에 Lambda 함수를 추가하면 다음 오류 메시지가 표시됩니다.

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

`GetIntent` 작업의 응답에는 의도의 특정 개정을 식별하는 `checksum`이라는 필드가 포함되어 있습니다. PutIntent 작업을 사용하여 의도를 업데이트하는 경우 체크섬 값을 제공해야 합니다. 체크섬 값을 제공하지 않으면 다음 오류 메시지가 표시됩니다.

```
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.
```

이 연습에서는 연습 1: 청사진을 사용하여 Amazon Lex 봇 생성(콘솔)의 Lambda 함수를 사용합니다. Lambda 함수를 생성하는 방법에 대한 지침은 3단계: Lambda 함수 만들기(콘솔)을 참조하십시오.

> **Note**
>
> 다음 AWS CLI 예제는 Unix, Linux 및 macOS용 형식입니다. Windows의 경우 `"\$LATEST"`를 `$LATEST`로 변경합니다.

의도에 Lambda 함수를 추가하려면

1. 에서 `OrderFlowers` 의도에 대한 `InvokeFunction` 권한을 AWS CLI추가합니다.

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

   Lambda가 다음의 응답을 보냅니다.

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

2. Amazon Lex 에서 의도를 얻으십시오. Amazon Lex는 출력을 **`OrderFlowers-V3.json`**라는 파일로 보냅니다.

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

3. 텍스트 편집기에서 **`OrderFlowers-V3.json`**을 엽니다.

   1. `createdDate`, `lastUpdatedDate` 및 `version` 필드를 찾아 삭제합니다.
   2. `fulfillmentActivity` 필드를 업데이트합니다.

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

   3. 파일을 저장합니다.

4. 에서 업데이트된 의도를 Amazon Lex로 AWS CLI전송합니다.

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

이제 의도를 업데이트했으므로 봇을 다시 구축합니다.

`OrderFlowersBot` 봇을 다시 구축하려면

1. 에서 `OrderFlowersBot` 봇의 정의를 AWS CLI가져와 파일에 저장합니다.

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

2. 텍스트 편집기에서 **`OrderFlowersBot-V3.json`** 파일을 엽니다. `createdDate`, `lastUpdatedDate`, `status` 및 `version` 필드를 제거합니다.
3. 텍스트 편집기에서 봇 정의에 다음 줄을 추가합니다.

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

4. 에서 봇의 새 개정을 AWS CLI빌드합니다.

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

서버에서 다음과 같이 응답합니다.

```
{
    "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"
```

```
}
```

다음 단계

4단계: 버전 게시(AWS CLI)

4단계: 버전 게시(AWS CLI)

이제 연습 1에서 생성한 봇의 버전을 생성합니다. 버전은 봇의 스냅샷입니다. 버전을 생성한 후에는 변경할 수 없습니다. 업데이트할 수 있는 유일한 봇 버전은 `$LATEST` 버전입니다. 버전에 대한 자세한 내용은 버전 관리 및 별칭을 참조하십시오.

봇의 버전을 게시하려면 먼저 봇에서 사용하는 의도를 게시해야 합니다. 마찬가지로 해당 의도에서 참조하는 슬롯 유형을 게시해야 합니다. 일반적으로 버전을 게시하려면 다음을 수행합니다.

1. CreateSlotTypeVersion 작업을 사용하여 슬롯 유형의 버전을 게시합니다.
2. CreateIntentVersion 작업을 사용하여 의도의 버전을 게시합니다.
3. CreateBotVersion 작업을 사용하여 봇의 버전을 게시합니다.

이 연습에서 명령을 실행하려면 명령이 실행될 리전을 알아야 합니다. 리전 목록은 모델 구축 할당량을 참조하십시오.

주제

- 1단계: 슬롯 유형 게시(AWS CLI)
- 2단계: 의도 게시(AWS CLI)
- 3단계: 봇 게시(AWS CLI)

1단계: 슬롯 유형 게시(AWS CLI)

슬롯 유형을 사용하는 의도의 버전을 게시하려면 먼저 해당 슬롯 유형의 버전을 게시해야 합니다. 이 경우 `FlowerTypes` 슬롯 유형을 게시합니다.

> **Note**
>
> 다음 AWS CLI 예제는 Unix, Linux 및 macOS용 형식입니다. Windows의 경우 `"\$LATEST"`를 `$LATEST`로 바꾸고, 각 줄의 끝에 있는 백슬래시(\) 연속 문자를 캐럿(^)으로 변경합니다.

슬롯 유형을 게시하려면(AWS CLI)

1. 에서 슬롯 유형의 최신 버전을 AWS CLI가져옵니다.

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

Amazon Lex에서 다음과 같이 응답합니다. $LATEST 버전의 현재 개정에 대한 체크섬을 기록합니다.

```
{
    "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. 슬롯 유형의 버전을 게시합니다. 이전 단계에서 기록한 체크섬을 사용합니다.

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

Amazon Lex에서 다음과 같이 응답합니다. 다음 단계를 위해 버전 번호를 기록합니다.

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

다음 단계

2단계: 의도 게시(AWS CLI)

2단계: 의도 게시(AWS CLI)

의도를 게시하려면 먼저 의도에서 참조하는 모든 슬롯 유형을 게시해야 합니다. 슬롯 유형은 `$LATEST` 버전이 아니라 번호가 지정된 버전이어야 합니다.

먼저 이전 단계에서 게시한 `FlowerTypes` 슬롯 유형의 버전을 사용하도록 `OrderFlowers` 의도를 업데이트합니다. 그런 다음 `OrderFlowers` 의도의 새 버전을 게시합니다.

> **Note**
>
> 다음 AWS CLI 예제는 Unix, Linux 및 macOS용 형식입니다. Windows의 경우 `"\$LATEST"`를 `$LATEST`로 바꾸고, 각 줄의 끝에 있는 백슬래시(\) 연속 문자를 캐럿(^)으로 변경합니다.

의도의 버전을 게시하려면(AWS CLI)

1. 에서 `OrderFlowers` 의도의 `$LATEST` 버전을 AWS CLI가져와 파일에 저장합니다.

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

2. 텍스트 편집기에서 **OrderFlowers_V4.json** 파일을 엽니다. `createdDate`, `lastUpdatedDate` 및 `version` 필드를 삭제합니다. `FlowerTypes` 슬롯 유형을 찾아 이전 단계에서 기록한 버전 번호로 버전을 변경합니다. **OrderFlowers_V4.json** 파일의 다음 조각은 변경 위치를 보여 줍니다.

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

3. 에서 의도의 개정을 AWS CLI저장합니다.

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

4. 의도의 최신 개정에 대한 체크섬을 가져옵니다.

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

응답의 다음 조각은 의도의 체크섬을 보여 줍니다. 다음 단계를 위해 이 체크섬을 기록합니다.

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

5. 의도의 새 버전을 게시합니다.

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

응답의 다음 조각은 의도의 새 버전을 보여 줍니다. 다음 단계를 위해 버전 번호를 기록합니다.

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

다음 단계

3단계: 봇 게시(AWS CLI)

3단계: 봇 게시(AWS CLI)

봇에서 사용하는 모든 슬롯 유형 및 의도를 게시한 후 봇을 게시할 수 있습니다.

이전 단계에서 업데이트한 `OrderFlowers` 의도를 사용하도록 `OrderFlowersBot` 봇을 업데이트합니다. 그런 다음 새 버전의 `OrderFlowersBot` 봇을 게시합니다.

> **Note**
> 다음 AWS CLI 예제는 Unix, Linux 및 macOS용 형식입니다. Windows의 경우 `"\$LATEST"`를 `$LATEST`로 바꾸고, 각 줄의 끝에 있는 백슬래시(\) 연속 문자를 캐럿(^)으로 변경합니다.

봇의 버전을 게시하려면(AWS CLI)

1. 에서 `OrderFlowersBot` 봇의 `$LATEST` 버전을 AWS CLI가져와 파일에 저장합니다.

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

2. 텍스트 편집기에서 **OrderFlowersBot_V4.json** 파일을 엽니다. `createdDate`, `lastUpdatedDate`, `status` 및 `version` 필드를 삭제합니다. `OrderFlowers` 의도를 찾아 이전 단계에서 기록한 버전 번호로 버전을 변경합니다. **OrderFlowersBot_V4.json**의 다음 조각은 변경 위치를 보여 줍니다.

```
    "intents": [
        {
            "intentVersion": "version",
            "intentName": "OrderFlowers"
        }
```

3. 에서 봇의 새 개정을 AWS CLI저장합니다. `put-bot`을 호출하면 반환되는 버전 번호를 기록해 둡니다.

```
aws lex-models put-bot \
    --name OrderFlowersBot \
    --cli-input-json file://OrderFlowersBot_V4.json
```

4. 봇의 최신 개정에 대한 체크섬을 가져옵니다. 3단계에서 반환된 버전 번호를 사용합니다.

```
aws lex-models get-bot \
    --region region \
    --version-or-alias version \
    --name OrderFlowersBot > OrderFlowersBot_V4a.json
```

응답의 다음 조각은 봇의 체크섬을 보여 줍니다. 다음 단계를 위해 이 체크섬을 기록합니다.

```
    "name": "OrderFlowersBot",
    "locale": "en-US",
    "checksum": "checksum",
```

5. 새 버전의 봇을 게시합니다.

```
aws lex-models create-bot-version \
    --region region \
```

```
    --name OrderFlowersBot \
    --checksum "checksum"
```

응답의 다음 조각은 봇의 새 버전을 보여 줍니다.

```
    "checksum": "checksum",
    "abortStatement": {
        ...
    },
    "version": "1",
    "lastUpdatedDate": timestamp,
```

다음 단계

연습 5: 별칭 만들기(AWS CLI)

연습 5: 별칭 만들기(AWS CLI)

별칭은 특정 봇 버전에 대한 포인터입니다. 별칭을 사용하여 클라이언트 애플리케이션에서 사용 중인 버전을 손쉽게 업데이트할 수 있습니다. 자세한 내용은 버전 관리 및 별칭을 참조하십시오. 이 연습에서 명령을 실행하려면 명령이 실행될 리전을 알아야 합니다. 리전 목록은 모델 구축 할당량 을 참조하십시오.

별칭을 만들려면(AWS CLI)

1. 에서 `OrderFlowersBot`에서 생성한 봇의 버전을 AWS CLI가져옵니다4단계: 버전 게시(AWS CLI).

```
aws lex-models get-bot \
    --region region \
    --name OrderFlowersBot \
    --version-or-alias version > OrderFlowersBot_V5.json
```

2. 텍스트 편집기에서 **`OrderFlowersBot_v5.json`** 파일을 엽니다. 버전 번호를 찾아 기록합니다.
3. 에서 봇 별칭을 AWS CLI생성합니다.

```
aws lex-models put-bot-alias  \
    --region region \
    --name PROD \
    --bot-name OrderFlowersBot \
```

```
    --bot-version version
```

서버에서 다음과 같이 응답합니다.

```
{
    "name": "PROD",
    "createdDate": timestamp,
    "checksum": "checksum",
    "lastUpdatedDate": timestamp,
    "botName": "OrderFlowersBot",
    "botVersion": "1"
}}
```

다음 단계

연습 6: 정리(AWS CLI)

연습 6: 정리(AWS CLI)

생성한 리소스를 삭제하고 계정을 정리합니다.

사용하지 않는 리소스만 삭제할 수 있습니다. 일반적으로 다음 순서로 리소스를 삭제해야 합니다.

1. 별칭을 삭제하여 봇 리소스를 확보합니다.
2. 봇을 삭제하여 의도 리소스를 확보합니다.
3. 의도를 삭제하여 슬롯 유형 리소스를 확보합니다.
4. 슬롯 유형을 삭제합니다.

이 연습에서 명령을 실행하려면 명령이 실행될 리전을 알아야 합니다. 리전 목록은 모델 구축 할당량을 참조하십시오.

계정을 정리하려면(AWS CLI)

1. AWS CLI 명령줄에서 별칭을 삭제합니다.

```
aws lex-models delete-bot-alias \
    --region region \
    --name PROD \
```

```
    --bot-name OrderFlowersBot
```

2. AWS CLI 명령줄에서 봇을 삭제합니다.

```
aws lex-models delete-bot \
    --region region \
    --name OrderFlowersBot
```

3. AWS CLI 명령줄에서 의도를 삭제합니다.

```
aws lex-models delete-intent \
    --region region \
    --name OrderFlowers
```

4. AWS CLI 명령줄에서 슬롯 유형을 삭제합니다.

```
aws lex-models delete-slot-type \
    --region region \
    --name FlowerTypes
```

생성한 모든 리소스를 제거하고 계정을 정리했습니다.

버전 관리 및 별칭

Amazon Lex는 클라이언트 애플리케이션에서 사용하는 구현을 제어할 수 있도록 봇, 의도 및 슬롯 유형의 게시 버전을 지원합니다.. 버전은 개발, 베타 배포, 프로덕션 등의 여러 워크플로우 부분에서 사용하도록 게시할 수 있는 작업의 번호가 지정된 스냅샷입니다.

Amazon Lex 봇은 별칭도 지원합니다. 별칭은 특정 봇 버전에 대한 포인터입니다. 별칭을 사용하여 클라이언트 애플리케이션에서 사용 중인 버전을 손쉽게 업데이트할 수 있습니다. 예를 들어, 별칭이 봇의 버전 1을 포인팅할 수 있습니다. 봇을 업데이트할 준비가 되면 버전 2를 게시하고 별칭이 새 버전을 가리키도록 변경합니다. 애플리케이션은 특정 버전 대신 별칭을 사용하므로 모든 클라이언트는 업데이트할 필요 없이 새 기능을 사용할 수 있습니다.

주제

- 버전 관리
- 에일리어스

버전 관리

Amazon Lex 리소스 버전을 지정하면 해당 버전이 만들어졌을 당시의 리소스를 그대로 사용할 수 있도록 리소스의 스냅샷이 생성됩니다. 버전을 생성하면 애플리케이션에 대해 계속 작업하는 동안에는 버전이 동일하게 유지됩니다.

$LATEST 버전

Amazon Lex 봇, 의도 또는 슬롯 유형을 생성할 때 버전은 $LATEST 버전 하나만 있습니다.

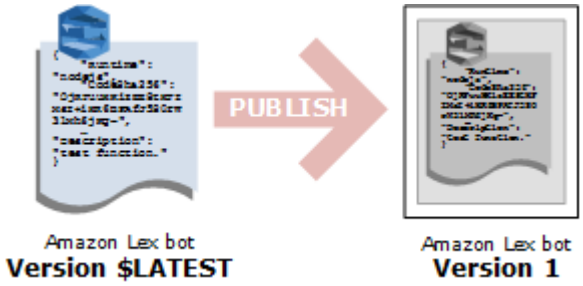
$LATEST는 리소스의 작업 복사본입니다. $LATEST 버전만 업데이트할 수 있으며, 첫 번째 버전을 게시할 때까지 $LATEST가 보유한 리소스의 유일한 버전입니다.

리소스의 $LATEST 버전만 다른 리소스의 $LATEST 버전을 사용할 수 있습니다. 예를 들어 봇의 $LATEST 버전은 의도의 $LATEST 버전을 사용할 수 있고, 의도의 $LATEST 버전은 슬롯 유형의 $LATEST 버전을 사용할 수 있습니다.

봇 $LATEST 버전은 수동 테스트에만 사용해야 합니다. Amazon Lex는 봇 $LATEST 버전에 대해 실행할 수 있는 런타임 요청 수를 제한합니다.

Amazon Lex 리소스 버전 게시

리소스를 게시하면 Amazon Lex는 $LATEST 버전의 복사본을 만들어 번호가 매겨진 버전으로 저장합니다. 게시된 버전은 변경할 수 없습니다.



Amazon Lex 콘솔이나 CreateBotVersion 작업을 사용하여 버전을 생성하고 게시합니다. 예시는 연습 3: 버전 게시 및 별칭 만들기에서 확인하십시오.

$LATEST 버전의 리소스를 수정할 때 클라이언트 애플리케이션에서 변경 사항을 사용할 수 있도록 새 버전을 게시할 수 있습니다. 버전을 게시할 때마다 Amazon Lex는 $LATEST 버전을 복사하여 새 버전을 생성하고 버전 번호를 1씩 증가시킵니다. 버전 번호는 절대 재사용되지 않습니다. 예를 들어, 버전 10이 지정된 리소스를 제거하고 다시 생성하면 Amazon Lex가 할당하는 다음 버전 번호는 버전 11이 됩니다.

봇을 게시하려면 먼저 봇을 봇에서 사용하는 의도의 버전으로 지정해야 합니다. 의도의 $LATEST 버전을 사용하는 새로운 봇 버전을 게시하려고 하면, Amazon Lex는 HTTP 400 잘못된 요청 예외를 반환합니다. 숫자가 지정된 의도 버전을 게시하려면 먼저 의도를 의도에서 사용하는 슬롯 유형의 버전으로 지정해야 합니다. 그렇지 않으면 HTTP 400 잘못된 요청 예외가 발생합니다.

> **Note**
>
> Amazon Lex는 마지막으로 게시된 버전이 $LATEST 버전과 다른 경우에만 새 버전을 게시합니다. $LATEST 버전을 수정하지 않고 게시하려고 하면 Amazon Lex는 새 버전을 생성하거나 게시하지 않습니다.

Amazon Lex 리소스 업데이트

Amazon Lex 봇, 의도 또는 슬롯 유형의 $LATEST 버전만 업데이트할 수 있습니다. 게시된 버전은 변경할 수 없습니다. 콘솔에서 또는 CreateBotVersion, CreateIntentVersion 또는 CreateSlotTypeVersion 작업을 사용하여 리소스를 업데이트한 후 언제든 새 버전을 게시할 수 있습니다.

Amazon Lex 리소스 또는 버전 삭제

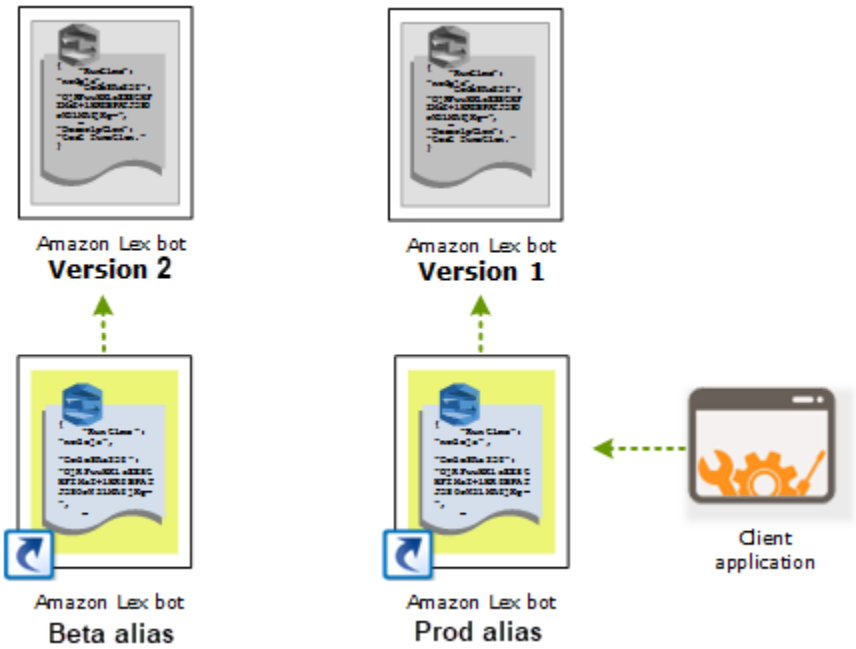
Amazon Lex는 콘솔 또는 API 작업 중 하나를 사용하여 리소스 또는 버전을 삭제하는 작업을 지원합니다.

- DeleteBot
- DeleteBotVersion
- DeleteBotAlias
- DeleteBotChannelAssociation
- DeleteIntent
- DeleteIntentVersion
- DeleteSlotType
- DeleteSlotTypeVersion

에일리어스

별칭은 특정 버전의 봇에 대한 포인터입니다. 별칭을 사용하면 클라이언트 애플리케이션에서 어떤 버전인지 추적할 필요 없이 봇의 특정 버전을 사용할 수 있습니다.

다음 예에서는 Amazon Lex 봇의 두 가지 버전인 버전 1과 버전 2를 보여줍니다. 이러한 봇 버전에는 각각 BETA와 PROD라는 연결된 별칭이 있습니다. 클라이언트 애플리케이션은 PROD 별칭을 사용하여 봇에 액세스합니다.



봇의 두 번째 버전을 생성할 때 콘솔 또는 PutBot 작업을 사용하여 봇의 새 버전을 가리키도록 별칭을 업데이트할 수 있습니다. 별칭을 변경하면 모든 클라이언트 애플리케이션에서 새 버전을 사용합니다. 새 버전에 문제가 있는 경우, 이전 버전을 가리키도록 별칭을 변경하면 해당 버전으로 롤백할 수 있습니다.

> **Note**
>
> 콘솔에서 봇의 `$LATEST` 버전을 테스트할 수 있으나, 봇을 클라이언트 애플리케이션에 통합할 때 먼저 버전을 게시하고 해당 버전을 가리키는 별칭을 만드는 것을 권장합니다. 이 섹션에 클라이언트 애플리케이션에서 별칭을 사용하는 이유가 설명되어 있습니다. 별칭을 업데이트하면 Amazon Lex는 새 버전을 사용하기 전에 모든 현재 세션의 세션 제한 시간이 만료될 때까지 기다립니다. 세션 제한 시간에 대한 자세한 내용은 the section called "세션 시간 제한 설정"을 참조하십시오.

Lambda 함수 사용

Amazon Lex 봇의 코드 후크로 사용할 AWS Lambda 함수를 생성할 수 있습니다. 사용자 의도 구성에서 초기화와 확인, 의도 이행 또는 두 가지 모두를 수행할 Lambda 함수를 식별할 수 있습니다.

Lambda 함수를 봇을 위한 코드 후크로 사용하는 것이 좋습니다. Lambda 함수가 없는 경우, 이행하기 위해 봇이 의도 정보를 클라이언트 애플리케이션에 반환합니다.

주제

- Lambda 함수 입력 이벤트 및 응답 형식
- Amazon Lex와 AWS Lambda 청사진

Lambda 함수 입력 이벤트 및 응답 형식

이 단원에서는 Amazon Lex가 Lambda 함수에 제공하는 이벤트 데이터의 구조에 대해 설명합니다. 이 정보를 사용하여 코드에 입력된 내용을 분석합니다. 또한 Lambda 함수가 반환할 것으로 Amazon Lex가 예상하는 응답 형식에 대해 설명합니다.

주제

- 입력 이벤트 형식
- 응답 형식

입력 이벤트 형식

다음은 Lambda 함수로 전달되는 Amazon Lex 이벤트의 일반적인 형식을 나타냅니다. 이 정보는 Lambda 함수를 작성할 때 사용합니다.

> **Note**
>
> `messageVersion`에서 해당 내용을 변경하지 않아도 입력 형식을 변경할 수 있습니다. 새 필드가 있는 경우 코드에서 오류가 발생하면 안 됩니다.

```
{
  "currentIntent": {
    "name": "intent-name",
```

```
    "nluIntentConfidenceScore": score,
    "slots": {
      "slot name": "value",
      "slot name": "value"
    },
    "slotDetails": {
      "slot name": {
        "resolutions" : [
          { "value": "resolved value" },
          { "value": "resolved value" }
        ],
        "originalValue": "original text"
      },
      "slot name": {
        "resolutions" : [
          { "value": "resolved value" },
          { "value": "resolved value" }
        ],
        "originalValue": "original text"
      }
    },
    "confirmationStatus": "None, Confirmed, or Denied (intent confirmation, if
configured)"
  },
  "alternativeIntents": [
    {
      "name": "intent-name",
      "nluIntentConfidenceScore": score,
      "slots": {
        "slot name": "value",
        "slot name": "value"
      },
      "slotDetails": {
        "slot name": {
          "resolutions" : [
            { "value": "resolved value" },
            { "value": "resolved value" }
          ],
          "originalValue": "original text"
        },
        "slot name": {
          "resolutions" : [
            { "value": "resolved value" },
            { "value": "resolved value" }
```

```
          ],
          "originalValue": "original text"
        }
      },
      "confirmationStatus": "None, Confirmed, or Denied (intent confirmation, if
configured)"
    }
  ],
  "bot": {
    "name": "bot name",
    "alias": "bot alias",
    "version": "bot version"
  },
  "userId": "User ID specified in the POST request to Amazon Lex.",
  "inputTranscript": "Text used to process the request",
  "invocationSource": "FulfillmentCodeHook or DialogCodeHook",
  "outputDialogMode": "Text or Voice, based on ContentType request header in runtime
API request",
  "messageVersion": "1.0",
  "sessionAttributes": {
     "key": "value",
     "key": "value"
  },
  "requestAttributes": {
     "key": "value",
     "key": "value"
  },
  "recentIntentSummaryView": [
    {
        "intentName": "Name",
        "checkpointLabel": Label,
        "slots": {
          "slot name": "value",
          "slot name": "value"
        },
        "confirmationStatus": "None, Confirmed, or Denied (intent confirmation, if
configured)",
        "dialogActionType": "ElicitIntent, ElicitSlot, ConfirmIntent, Delegate, or
Close",
        "fulfillmentState": "Fulfilled or Failed",
        "slotToElicit": "Next slot to elicit"
    }
  ],
   "sentimentResponse": {
```

```
        "sentimentLabel": "sentiment",
        "sentimentScore": "score"
    },
    "kendraResponse": {
        Complete query response from Amazon Kendra
    },
    "activeContexts": [
         {
             "timeToLive": {
                 "timeToLiveInSeconds": seconds,
                 "turnsToLive": turns
             },
             "name": "name",
             "parameters": {
                 "key name": "value"
             }
         }
     ]
}
```

다음은 이벤트 필드에 대한 추가 정보입니다.

- currentIntent - 의도 `name`, `slots`, `slotDetails` 및 `confirmationStatus` 필드를 제공합니다.

  `nluIntentConfidenceScore`은 Amazon Lex가 현재 의도가 사용자의 현재 의도와 가장 잘 일치하는 의도라는 확신입니다.

  `slots`는 의도에 대해 구성된 슬롯 이름의 맵으로, 슬롯 이름은 사용자 대화에서 Amazon Lex가 인식한 슬롯 값으로 매핑됩니다. 슬롯 값은 사용자가 값을 제공할 때까지 null을 유지합니다.

  입력 이벤트의 슬롯 값이 슬롯에 대해 구성된 값 중 하나와 일치하지 않을 수 있습니다. 예를 들어, 사용자가 "어떤 색상의 자동차를 원하시나요?"라는 프롬프트에 "피자"로 응답하면, Amazon Lex는 "피자"를 슬롯 값으로 반환합니다. 함수는 값이 컨텍스트 내에서 적절한지 확인해야 합니다.

slotDetails는 슬롯 값에 대한 추가 정보를 제공합니다. resolutions 어레이는 슬롯에 대해 인식된 추가 값 목록을 포함합니다. 각 슬롯은 값을 최대 5개까지 보유할 수 있습니다.

originalValue 필드에는 사용자가 슬롯에 대해 입력한 값이 들어 있습니다. 슬롯 유형이 최상위 확인 값을 슬롯 값으로 반환하도록 구성된 경우, originalValue가 slots 필드의 값과 다를 수 있습니다.

confirmationStatus는 확인 프롬프트에 대한 사용자 응답을 제공합니다(있는 경우). 예를 들어, Amazon Lex가 "라지 사이즈 피자를 주문을 원하시나요?"라고 묻는 경우 사용자 응답에 따라 이 필드의 값은 Confirmed 또는 Denied일 수 있습니다. 그렇지 않은 경우 이 필드 값은 None입니다.

사용자가 의도를 확인하면 Amazon Lex는 이 필드를 Confirmed로 설정합니다. 사용자가 의도를 거부하면 Amazon Lex는 이 값을 Denied로 설정합니다.

확인 응답에서 사용자 표현은 슬롯 업데이트를 제공할 수 있습니다. 예를 들어, 사용자가 "네, 미디엄으로 사이즈 변경해주세요."라고 말할 수 있습니다. 이 경우 후속 Lambda 이벤트는 PizzaSize를 medium으로 설정하여 슬롯 값을 업데이트했습니다. Amazon Lex는 confirmationStatus를 None으로 설정하는데, 이는 사용자가 일부 슬롯 데이터를 수정하여 Lambda 함수가 사용자 데이터 검증을 수행해야 하기 때문입니다.

- alternativeIntents – 신뢰도 점수를 활성화하면 Amazon Lex는 최대 4개의 대체 의도를 반환합니다. 각 의도에는 사용자의 말을 바탕으로 Amazon Lex가 해당 의도가 올바른 의도라고 확신하는 수준을 나타내는 점수가 포함됩니다.

  대체 의도의 콘텐츠는 currentIntent 필드의 콘텐츠와 동일합니다. 자세한 내용은 신뢰도 점수 사용을 참조하세요.

- 봇 – 요청을 처리한 봇에 대한 정보.
  - `name` – 요청을 처리한 봇의 이름.
  - `alias` – 요청을 처리한 봇 버전의 별칭.
  - `version` – 요청을 처리한 봇의 버전.

- userId – 이 값은 클라이언트 애플리케이션에서 제공합니다. Amazon Lex는 Lambda 함수에 이 값을 전달합니다.

- inputTranscript – 요청을 처리하는 데 사용되는 텍스트.

  입력이 텍스트인 경우 `inputTranscript` 필드에는 사용자가 입력한 텍스트가 표시됩니다.

  입력이 오디오 스트림인 경우 `inputTranscript` 필드에는 오디오 스트림에서 추출한 텍스트가 포함됩니다. 이 텍스트는 의도 및 슬롯 값을 인식하기 위해 실제로 처리되는 텍스트입니다.

- invocationSource - Amazon Lex가 Lambda 함수를 호출하는 이유를 나타내기 위해 이를 다음 값 중 하나로 설정합니다.
  - `DialogCodeHook` – 는 이 값을 설정하여 함수를 초기화하고 사용자의 데이터 입력을 확인하도록 Lambda 함수에 지시합니다.

    의도가 Lambda 함수를 초기화 및 확인 코드 후크로 호출하도록 구성된 경우 Amazon Lex가 의도를 이해한 후 Amazon Lex는 각 사용자 입력(표현)에 대해 지정된 Lambda 함수를 호출합니다.

    > **Note**
    > 의도가 명확하지 않은 경우, Amazon Lex는 Lambda 함수를 호출할 수 없습니다.

  - `FulfillmentCodeHook` - Amazon Lex는 이 값을 설정하여 의도를 이행하도록 Lambda 함수에 지시합니다.

의도가 Lambda 함수를 이행 코드 후크로 호출하도록 구성된 경우 Amazon Lex는 의도를 이행하기 위한 모든 슬롯 데이터가 확보된 후에만 `invocationSource`를 이 값으로 설정합니다.

의도 구성에 사용자 데이터를 초기화 및 확인하고 의도를 이행하기 위한 별개의 Lambda 함수 두 개가 있을 수 있습니다. Lambda 함수 하나를 사용하여 두 작업을 모두 수행할 수도 있습니다. 이 경우 Lambda 함수가 올바른 코드 경로를 따르도록 `invocationSource` 값을 사용할 수 있습니다.

- outputDialogMode - 각 사용자 입력에 대해 클라이언트는 런타임 API 작업인 PostContent 또는 PostText 중 하나를 사용하여 Amazon Lex에 요청을 보냅니다. Amazon Lex는 요청 파라미터에서 클라이언트에 대한 응답이 텍스트인지 음성인지 판단하고, 그에 따라 이 필드를 설정합니다.

  Lambda 함수는 이 정보를 사용하여 적절한 메시지를 생성할 수 있습니다. 예를 들어 클라이언트가 음성 응답을 기대하는 경우, Lambda 함수는 텍스트 대신에 음성 합성 마크업 언어(SSML)를 반환할 수 있습니다.

- messageVersion - Lambda 함수로 들어가는 이벤트 데이터의 형식과 Lambda 함수의 예상 응답 형식을 식별하는 메시지 버전입니다.

  > **Note**
  >
  > 의도를 정의할 때 이 값을 구성합니다. 현재 구현에서는 메시지 버전 1.0만 지원됩니다. 따라서 콘솔은 기본값을 1.0으로 가정하며 이는 메시지 버전에 표시되지 않습니다.

- sessionAttributes – 클라이언트가 요청에서 전송하는 애플리케이션별 세션 속성입니다. Amazon Lex가 클라이언트에 대한 응답에 이 세션 속성을 포함시키도록 하려는 경우, Lambda 함수는 응답에서 이 세션 속성을 다시 Amazon Lex에 전송해야 합니다. 자세한 내용은 Setting Session Attributes을 참조하세요.

- requestAttributes – 클라이언트가 요청에서 전송하는 요청별 속성입니다. 전체 세션에서 유지할 필요가 없는 정보를 전달하려면 요청 속성을 사용합니다. 요청 속성이 존재하지 않는 경우, 값은 null이 됩니다. 자세한 내용은 Setting Request Attributes을 참조하세요.

- recentIntentSummaryView – 의도 상태에 대한 정보입니다. 사용된 의도 중 마지막 3개에 대한 정보를 볼 수 있습니다. 이 정보를 사용하여 의도의 값을 설정하거나 이전 의도로 돌아갈 수 있습니다. 자세한 내용은 Amazon Lex API로 세션 관리을 참조하세요.

- sentimentResponse – 마지막 표현에 대한 Amazon Comprehend 감정 분석의 결과입니다. 이 정보를 사용하여 사용자가 표현한 감정에 따라 봇의 대화 흐름을 관리할 수 있습니다. 자세한 내용은 감정 분석을 참조하세요.

- kendraResponse – Amazon Kendra 인덱스에 대한 쿼리 결과. 해당 의도가 `AMAZON.KendraSearchIntent` 기본 제공 의도를 확장하는 경우에 한해 이행 코드 후크에 대한 입력에만 표시됩니다. 이 필드에는 Amazon Kendra 검색의 전체 응답이 포함됩니다. 자세한 내용은 AMAZON.KendraSearchIntent을 참조하세요.

- activeContexts – 사용자와의 대화가 진행되는 동안 활성화되는 하나 이상의 컨텍스트.
  - timeToLive — 컨텍스트가 활성 상태로 유지될 때까지 사용자와 대화하는 시간 또는 턴 수.
  - name – 컨텍스트의 이름.
  - 파라미터는 컨텍스트를 활성화한 의도의 슬롯 이름과 값을 포함하는 키/값 쌍의 목록입니다.

  자세한 내용은 의도 컨텍스트 설정을 참조하세요.

응답 형식

Amazon Lex는 Lambda 함수에서 다음 형식의 응답을 기대합니다.

```
{
  "sessionAttributes": {
    "key1": "value1",
    "key2": "value2"
    ...
  },
```

```
  "recentIntentSummaryView": [
    {
       "intentName": "Name",
       "checkpointLabel": "Label",
       "slots": {
         "slot name": "value",
         "slot name": "value"
        },
       "confirmationStatus": "None, Confirmed, or Denied (intent confirmation, if
 configured)",
        "dialogActionType": "ElicitIntent, ElicitSlot, ConfirmIntent, Delegate, or
 Close",
        "fulfillmentState": "Fulfilled or Failed",
        "slotToElicit": "Next slot to elicit"
    }
  ],
  "activeContexts": [
     {
       "timeToLive": {
          "timeToLiveInSeconds": seconds,
          "turnsToLive": turns
      },
      "name": "name",
      "parameters": {
        "key name": "value"
      }
    }
  ],
  "dialogAction": {
    "type": "ElicitIntent, ElicitSlot, ConfirmIntent, Delegate, or Close",
    Full structure based on the type field. See below for details.
  }
}
```

응답은 두 개의 필드로 구성되어 있습니다. `sessionAttributes`, `recentIntentSummaryView` 및 `activeContexts`필드는 선택 사항이고, `dialogAction` 필드는 필수입니다. `dialogAction` 필드의 내용은 `type` 필드의 값에 따라 달라집니다. 자세한 내용은 dialogAction을 참조하세요.

sessionAttributes

선택 사항 `sessionAttributes` 필드를 포함시킬 경우 비워둘 수 있습니다. Lambda 함수가 세션 속성을 반환하지 않으면 API 또는 Lambda 함수를 통해 전달된 마지막으로 알려진 `sessionAttributes`이 유지됩니다. 자세한 내용은 PostContent 및 PostText작업을 참조하십시오.

```
  "sessionAttributes": {
     "key1": "value1",
     "key2": "value2"
  }
```

recentIntentSummaryView

선택 사항. 포함된 경우 하나 이상의 최근 의도에 대한 값을 설정합니다. 최대 3개의 의도에 대한 정보를 포함할 수 있습니다. 예를 들어, 현재 의도에서 수집한 정보를 기반으로 이전 의도의 값을 설정할 수 있습니다. 요약에 있는 정보는 의도에 대해 유효해야 합니다. 예를 들어 의도 이름은 봇의 의도여야 합니다. 요약 보기에 슬롯 값을 포함하는 경우 슬롯이 의도에 있어야 합니다. 응답에 `recentIntentSummaryView`를 포함하지 않는 경우 최근 의도에 대한 모든 값이 변경되지 않고 그대로 유지됩니다. 자세한 내용은 PutSession 작업 또는 IntentSummary 데이터 형식을 참조하십시오.

```
"recentIntentSummaryView": [
    {
       "intentName": "Name",
       "checkpointLabel": "Label",
       "slots": {
         "slot name": "value",
         "slot name": "value"
        },
       "confirmationStatus": "None, Confirmed, or Denied (intent confirmation, if
 configured)",
        "dialogActionType": "ElicitIntent, ElicitSlot, ConfirmIntent, Delegate, or
 Close",
        "fulfillmentState": "Fulfilled or Failed",
        "slotToElicit": "Next slot to elicit"
    }
  ]
```

activeContexts

선택 사항. 포함된 경우, 하나 이상의 컨텍스트에 대한 값을 설정합니다. 예를 들어, 컨텍스트를 포함하여 해당 컨텍스트를 입력으로 사용하는 하나 이상의 의도를 대화의 다음 단계에서 인식할 수 있도록 만들 수 있습니다.

응답에 포함되지 않은 모든 활성 컨텍스트의 time-to-live 값은 감소하므로 다음 요청에서도 여전히 활성 상태일 수 있습니다.

입력 이벤트에 포함된 컨텍스트의 time-to-live를 0으로 지정하면 다음 요청 시 해당 컨텍스트가 비활성화됩니다.

자세한 내용은 의도 컨텍스트 설정을 참조하세요.

dialogAction

필수 사항. `dialogAction` 필드는 Amazon Lex에 일련의 다음 조치를 지시하고, Amazon Lex가 클라이언트에 응답을 반환한 후 사용자로부터 무엇을 기대하는지를 설명합니다.

`type` 필드는 일련의 다음 조치를 나타냅니다. 또한 이 필드는 Lambda 함수가 `dialogAction` 값의 일부로 제공해야 하는 다른 필드를 결정합니다.

- `Close` — 사용자의 응답을 기대하지 않도록 Amazon Lex에 알립니다. 예를 들어 "피자 주문이 완료되었습니다"에는 응답이 필요 없습니다.

  `fulfillmentState` 필드는 필수 사항입니다. Amazon Lex는 이 값을 사용하여 클라이언트 애플리케이션에 대한 PostContent 또는 PostText 응답의 `dialogState` 필드를 설정합니다. `message` 및 `responseCard` 필드는 선택 사항입니다. 사용자가 메시지를 지정하지 않으면 Amazon Lex는 의도에 대해 구성된 종료 메시지 또는 후속 메시지를 사용합니다.

```
"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` — 사용자가 현재 의도를 확정하거나 거부하기 위해 예 또는 아니요 대답을 제공할 것으로 예상된다고 Amazon Lex에 알립니다.

  `intentName` 및 `slots` 필드를 포함시켜야 합니다. `slots` 필드에는 지정된 의도에 대해 입력된 각 슬롯에 대한 항목이 포함되어야 합니다. 입력되지 않은 슬롯의 `slots` 필드에는 항목을 포함시킬 필요가 없습니다. 해당 의도의 `confirmationPrompt` 필드가 null인 경우 `message` 필드를 포함해야 합니다. Lambda 함수가 반환한 `message` 필드의 내용은 의도에 지정된 `confirmationPrompt`보다 우선합니다. `responseCard` 필드는 선택 사항입니다.

```
"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` — 봇 구성에 따라 일련의 다음 조치를 선택하도록 Amazon Lex에 지시합니다. 응답에 세션 속성이 없을 경우 Amazon Lex는 기존 속성을 유지합니다. 슬롯 값이 nul)이 되게 하려면 요청에 슬롯 필드를 포함할 필요가 없습니다. 이행 함수가 슬롯을 제거하지 않고 `Delegate` 대화 작업을 반환하면 `DependencyFailedException` 예외가 발생합니다.

  `kendraQueryRequestPayload` 및 `kendraQueryFilterString` 필드는 선택 사항이며 해당 의도가 `AMAZON.KendraSearchIntent` 기본 제공 의도에서 파생된 경우에만 사용됩니다. 자세한 내용은 AMAZON.KendraSearchIntent를 참조하십시오.

```
"dialogAction": {
 "type": "Delegate",
 "slots": {
    "slot-name": "value",
    "slot-name": "value",
    "slot-name": "value"
 },
 "kendraQueryRequestPayload": "Amazon Kendra query",
 "kendraQueryFilterString": "Amazon Kendra attribute filters"
}
```

- `ElicitIntent` — 사용자가 의도를 포함하는 표현으로 응답할 것으로 예상된다고 Amazon Lex에 알립니다. 예를 들어 `OrderPizzaIntent`를 나타내는 "피자가 먹고 싶어요"입니다. 반면에 표현 "라지"는 Amazon Lex가 사용자의 의도를 유추하기에 부족합니다.

message 및 responseCard 필드는 선택 사항입니다. 사용자가 메시지를 제공하지 않으면 Amazon Lex는 봇의 분류 프롬프트 중 하나를 사용합니다. 확인 프롬프트가 정의되어 있지 않으면 Amazon Lex는 400 잘못된 요청 예외를 반환합니다.

```
{
  "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 — 사용자가 응답에 슬롯 값을 제공할 것으로 예상된다고 Amazon Lex에 알립니다.

intentName, slotToElicit 및 slots필드는 필수 사항입니다. message 및 responseCard 필드는 선택 사항입니다. 사용자가 메시지를 지정하지 않으면 Amazon Lex는 슬롯에 대해 구성된 슬롯 유도 프롬프트 중 하나를 사용합니다.

```
 "dialogAction": {
   "type": "ElicitSlot",
   "message": {
     "contentType": "PlainText or SSML or CustomPayload",
     "content": "Message to convey to the user. For example, What size pizza would
you like?"
   },
  "intentName": "intent-name",
  "slots": {
     "slot-name": "value",
     "slot-name": "value",
     "slot-name": "value"
  },
  "slotToElicit" : "slot-name",
  "responseCard": {
     "version": integer-value,
     "contentType": "application/vnd.amazonaws.card.generic",
     "genericAttachments": [
         {
            "title":"card-title",
            "subTitle":"card-sub-title",
            "imageUrl":"URL of the image to be shown",
            "attachmentLinkUrl":"URL of the attachment to be associated with the
card",
            "buttons":[
                {
                   "text":"button-text",
                   "value":"Value sent to server on button click"
                }
             ]
          }
      ]
    }
 }
```

Amazon Lex와 AWS Lambda 청사진

Amazon Lex 콘솔은 콘솔에서 봇을 신속하게 생성하고 테스트할 수 있도록 미리 구성된 예제 봇(봇 청사진이라고 함)을 제공합니다. 이러한 각 봇 청사진에 대해 Lambda 함수 청사진도 제공됩니다. 이러한 청사진은 해당 봇에 사용할 수 있는 샘플 코드를 제공합니다. 이러한 청사진을 사용하여 코드 후크인

Lambda 함수로 구성된 봇을 신속하게 생성하고, 코드를 작성하지 않고도 엔드 투 엔드 설정을 테스트할 수 있습니다.

다음 Amazon Lex 봇 블루프린트와 해당 AWS Lambda 함수 블루프린트를 봇의 코드 후크로 사용할 수 있습니다.

- Amazon Lex 청사진 — `OrderFlowers`
  - AWS Lambda 블루프린트 - `lex-order-flowers-python`
- Amazon Lex 청사진 — `ScheduleAppointment`
  - AWS Lambda 블루프린트 - `lex-make-appointment-python`
- Amazon Lex 청사진 — `BookTrip`
  - AWS Lambda 블루프린트 - `lex-book-trip-python`

청사진을 사용하여 봇을 생성하고 Lambda 함수를 코드 후크로 사용하도록 봇을 구성하려면 연습 1: 청사진을 사용하여 Amazon Lex 봇 생성(콘솔)을 참조하십시오. 기타 청사진의 사용 예제는 추가 예제: Amazon Lex 봇 생성을 참조하십시오.

특정 로캘에 대한 청사진 업데이트

영어(미국)(en-US) 이외의 로캘에서 청사진을 사용하는 경우, 해당 로캘을 포함하도록 의도 이름을 업데이트해야 합니다. 예를 들어 `OrderFlowers` 청사진을 사용하는 경우 다음 작업을 수행해야 합니다.

- Lambda 함수 코드의 끝 부분에서 `dispatch` 함수를 찾으세요.
- `dispatch` 함수에서 사용 중인 로캘을 포함하도록 의도 이름을 업데이트하십시오. 예를 들어 영어(호주)(en-AU) 로캘을 사용하는 경우 다음 줄을 변경하십시오.

  ```
  if intent_name == 'OrderFlowers':
  ```

  아래로 변경합니다.

  ```
  if intent_name == 'OrderFlowers_enAU':
  ```

다른 청사진은 다른 의도 이름을 사용하므로 사용하기 전에 위와 같이 업데이트해야 합니다.

Amazon Lex 봇 배포

이 섹션에서는 다양한 메시징 플랫폼 및 모바일 애플리케이션에 Amazon Lex 봇을 배포하는 예를 제공합니다.

주제

- 메시징 플랫폼에 Amazon Lex 봇 배포하기
- 모바일 애플리케이션에서 Amazon Lex 봇 배포하기

메시징 플랫폼에 Amazon Lex 봇 배포하기

이 섹션에서는 페이스북, 슬랙, 트윌리오 메시징 플랫폼에 Amazon Lex 봇을 배포하는 방법을 설명합니다.

> **Note**
>
> Facebook, Slack 또는 Twilio 구성을 저장할 때 Amazon Lex는 AWS Key Management Service 고객 관리 키를 사용하여 정보를 암호화합니다. 이러한 메시징 플랫폼 중 하나에 채널을 처음 생성할 때 Amazon Lex는 기본 고객 관리 키 (`aws/lex`) 를 생성합니다. 또는를 사용하여 자체 고객 관리형 키를 생성할 수 있습니다 AWS KMS. 자체 CMK를 사용하여 키 생성, 교체 및 비활성화 기능을 비롯한 다양한 작업을 수행할 수 있습니다. 또한 액세스 제어를 정의하고 데이터를 보호하는 데 사용하는 암호화 키를 감사할 수 있습니다. 자세한 내용은 AWS Key Management Service 개발자 안내서를 참조하세요.

메시징 플랫폼이 Amazon Lex에 요청을 보내면 플랫폼별 정보가 Lambda 함수의 요청 속성으로 포함됩니다. 이러한 속성을 사용하여 봇의 동작 방식을 사용자 지정할 수 있습니다. 자세한 내용은 Setting Request Attributes을 참조하세요.

모든 속성에는 네임스페이스, `x-amz-lex:`를 접두사로 사용합니다. 예를 들어, `user-id` 속성이 `x-amz-lex:user-id`를 호출합니다. 특정 플랫폼에만 적용되는 속성 외에도 모든 메시징 플랫폼에서 보내는 공통 속성이 있습니다. 다음 표에는 메시징 플랫폼이 봇의 Lambda 함수로 보내는 요청 속성이 나와 있습니다.

공통 요청 속성

속성	설명
`channel-id`	Amazon Lex의 채널 엔드포인트 식별자입니다.
`channel-name`	Amazon Lex에서 가져온 채널 이름입니다.
`channel-type`	다음 값 중 하나입니다. • `Facebook` • `Kik` • `Slack` • `Twilio-SMS`
`webhook-endpoint-url`	채널의 Amazon Lex 엔드포인트입니다.

Facebook Request 속성

속성	설명
`user-id`	발신자의 페이스북 식별자. https://developers.facebook.com/docs/messenger-platform/webhook-reference/message-received를 참조하십시오.
`facebook-page-id`	수신자의 페이스북 페이지 식별자. https://developers.facebook.com/docs/messenger-platform/webhook-reference/message-received를 참조하십시오.

Kik 요청 속성

속성	설명
`kik-chat-id`	봇이 관여한 대화의 식별자. 자세한 내용은 https://dev.kik.com/#/docs/messaging#message-formats를 참조하십시오.

속성	설명
`kik-chat-type`	메시지가 시작된 대화의 유형. 자세한 내용은 https://dev.kik.com/#/docs/messaging#message-formats를 참조하십시오.
`kik-message-id`	UUID는 메시지를 식별합니다. 자세한 내용은 https://dev.kik.com/#/docs/messaging#message-formats를 참조하십시오.
`kik-message-type`	메시지 유형. 자세한 내용은 https://dev.kik.com/#/docs/messaging#message-types를 참조하십시오.

Twilio 요청 속성

속성	설명
`user-id`	발신자의 전화번호 (“보낸 사람”). https://www.twilio.com/docs/api/rest/message를 참조하십시오.
`twilio-target-phone-number`	수신자 ("To")의 전화번호입니다. https://www.twilio.com/docs/api/rest/message를 참조하십시오.

슬랙 요청 속성

속성	설명
`user-id`	슬랙 사용자 식별자입니다. https://api.slack.com/types/user를 참조하십시오.
`slack-team-id`	메시지를 보낸 팀의 식별자. https://api.slack.com/methods/team.info를 참조하십시오.
`slack-bot-token`	봇에게 Slack API에 대한 액세스 권한을 부여하는 개발자 토큰입니다. https://api.slack.com/docs/token-types를 참조하십시오.

Amazon Lex 봇을 Facebook Messenger와 통합하기

이 연습에서는 Facebook Messenger를 Amazon Lex 봇에 통합하는 방법을 보여줍니다. 다음 절차를 수행합니다.

1. Amazon Lex 봇 생성
2. Facebook 애플리케이션 생성
3. Facebook Messenger를 Amazon Lex 봇에 통합합니다.
4. 통합을 검증하세요.

주제

- 1단계: Amazon Lex 봇 생성
- 2단계: Facebook 애플리케이션 생성
- 3단계: Facebook Messenger를 Amazon Lex봇에 통합하기
- 4단계: 통합 테스트

1단계: Amazon Lex 봇 생성

아직 Amazon Lex 봇이 없는 경우 하나를 생성하고 배포합니다. 이 주제에서는 사용자가 시작하기 연습 1에서 생성한 봇을 사용하는 것으로 가정합니다. 그러나 이 설명서에서 제공된 모든 예제 봇을 사용할 수 있습니다. 시작하기 연습 1은 연습 1: 청사진을 사용하여 Amazon Lex 봇 생성(콘솔)을 참조하십시오.

1. Amazon Lex 봇 생성. 지침은 연습 1: 청사진을 사용하여 Amazon Lex 봇 생성(콘솔)을 참조하세요.
2. 봇을 배포하고 별칭을 생성합니다. 지침은 연습 3: 버전 게시 및 별칭 만들기을 참조하세요.

2단계: Facebook 애플리케이션 생성

Facebook 개발자 포털에서 Facebook 애플리케이션 및 Facebook 페이지를 만듭니다. 이에 대한 지침은 Facebook Messenger 플랫폼 설명서의 빠른 시작을 참조하십시오. 다음 사항을 적어둡니다.

- 페이스북 앱을 위한 앱 시크릿
- 페이스북 페이지를 위한 페이지 액세스 토큰

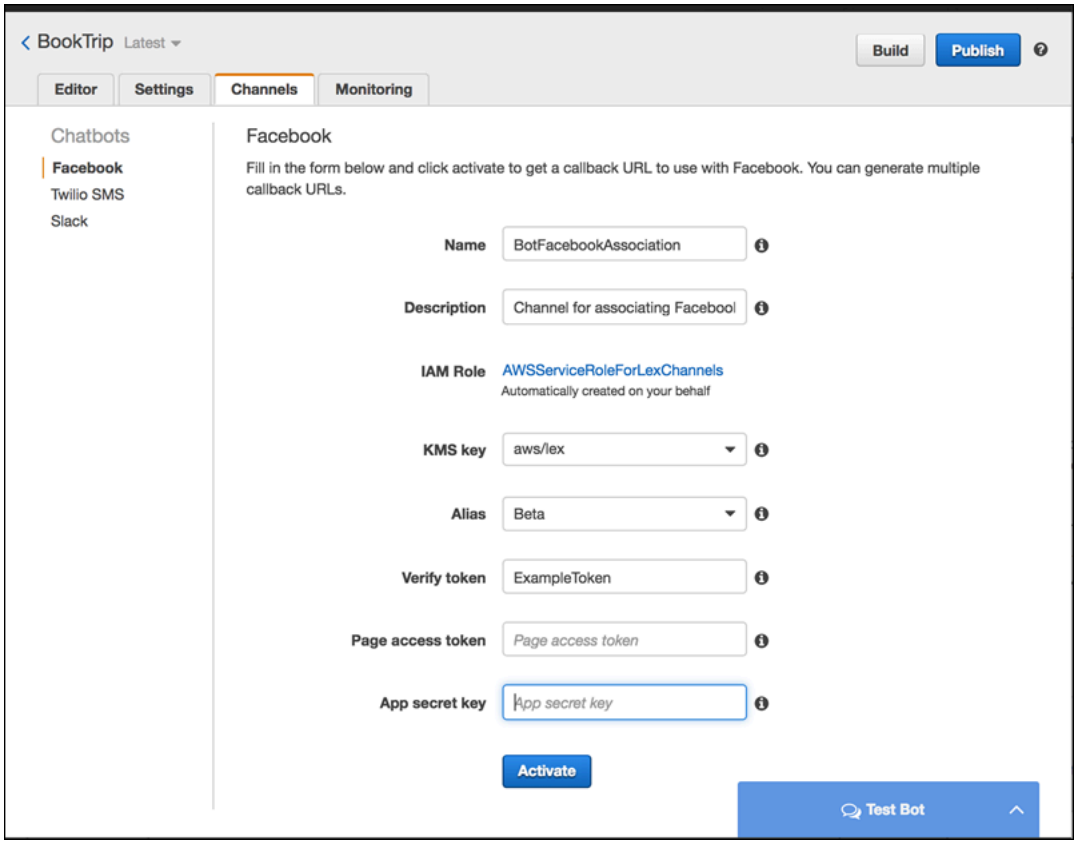
3단계: Facebook Messenger를 Amazon Lex봇에 통합하기

이 섹션에서는 Facebook Messenger를 Amazon Lex 봇에 통합합니다.

이 단계를 완료하면 콘솔에서 콜백 URL을 제공합니다. 이 URL을 적어둡니다.

Facebook Messenger를 봇에 통합하려면

1. a. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lex/ Amazon Lex 콘솔을 엽니다.
   b. Amazon Lex 봇을 선택하세요.
   c. 채널을 선택합니다.
   d. 챗봇에서 페이스북을 선택합니다. 콘솔에 Facebook 통합 페이지가 표시됩니다.
   e. Facebook 통합 페이지에서 다음을 수행합니다.
      - 다음 이름을 입력합니다: `BotFacebookAssociation`.
      - KMS 키에서 aws/lex를 선택합니다.
      - 별칭에서 봇 별칭을 선택합니다.
      - 토큰 검증의 경우 토큰을 입력합니다. 선택한 임의의 문자열이 될 수 있습니다(예: `ExampleToken`). 나중에 웹훅을 설정할 때 Facebook 개발자 포털에서 이 토큰을 사용합니다.
      - 페이지 액세스 토큰의 경우 2단계에서 Facebook에서 받은 토큰을 입력합니다.
      - 앱 비밀 키의 경우 2단계에서 Facebook에서 받은 키를 입력합니다.



f. 활성화를 선택합니다.

콘솔은 봇 채널 연결을 생성하고 콜백 URL을 반환합니다. 이 URL을 적어둡니다.

2. Facebook 개발자 포털에서 앱을 선택합니다.
3. 메신저 제품을 선택하고 페이지의 웹후크 섹션에서 웹후크 설정을 선택합니다.

   이에 대한 지침은 Facebook Messenger 플랫폼 설명서의 빠른 시작을 참조하십시오.
4. 웹훅 페이지 구독 마법사에서 다음 작업을 수행합니다.
   - 콜백 URL의 경우 절차 앞부분에서 Amazon Lex 콘솔에서 제공한 콜백 URL을 입력합니다.
   - 토큰 확인에는 Amazon Lex에서 사용한 것과 동일한 토큰을 입력합니다.
   - 구독 필드(메시지, message_postbacks 및 message_options)를 선택합니다.
   - 확인 및 저장을 선택합니다. 이렇게 하면 페이스북과 Amazon Lex 간의 핸드셰이크가 시작됩니다.
5. 웹훅 통합을 활성화합니다. 생성한 페이지를 선택한 다음, 구독을 선택합니다.

> **Note**
> 웹훅을 업데이트하거나 다시 생성하는 경우 페이지 구독을 취소한 다음 다시 구독하세요.

4단계: 통합 테스트

이제 Messenger Facebook에서 Amazon Lex 봇과 대화를 시작할 수 있습니다.

1. Facebook 페이지를 열고 메시지를 선택합니다.
2. 메신저 창에서는 1단계: Amazon Lex 봇 생성(콘솔)에서 제공하는 것과 동일한 테스트 표현을 사용합니다.

Amazon Lex 봇과 Kik의 통합

이번 연습에서는 Amazon Lex 봇을 Kik 메시징 애플리케이션과 통합하기 위한 지침을 제공합니다. 다음 절차를 수행합니다.

1. Amazon Lex 봇 생성.
2. Kik 앱과 웹 사이트를 사용하여 Kik 봇을 만드세요.
3. Amazon Lex 콘솔을 사용하여 Amazon Lex 봇을 Kik 봇과 통합하십시오.
4. Kik을 사용하여 Amazon Lex 봇과 대화하여 Amazon Lex 봇과 Kik 간의 연관성을 테스트해 보십시오.

주제

- 1단계: Amazon Lex 봇 생성
- 2단계: Kik 봇 생성
- 3단계: Kik 봇을 Amazon Lex 봇과 통합
- 4단계: 통합 테스트

1단계: Amazon Lex 봇 생성

아직 Amazon Lex 봇이 없는 경우 하나를 생성하고 배포합니다. 이 주제에서는 사용자가 시작하기 연습 1에서 생성한 봇을 사용하는 것으로 가정합니다. 그러나 이 설명서에서 제공된 모든 예제 봇을 사용

할 수 있습니다. 시작하기 연습 1은 연습 1: 청사진을 사용하여 Amazon Lex 봇 생성(콘솔)을 참조하십시오.

1. Amazon Lex 봇 생성. 지침은 연습 1: 청사진을 사용하여 Amazon Lex 봇 생성(콘솔)을 참조하세요.
2. 봇을 배포하고 별칭을 생성합니다. 지침은 연습 3: 버전 게시 및 별칭 만들기을 참조하세요.

다음 단계

2단계: Kik 봇 생성

2단계: Kik 봇 생성

이 단계에서는 Kik 사용자 인터페이스를 사용하여 Kik 봇을 생성합니다. 봇을 생성하는 동안 생성된 정보를 사용하여 봇을 Amazon Lex 봇에 연결합니다.

1. 아직 설치하지 않았다면 Kik 앱을 다운로드하여 설치하고 Kik 계정을 등록하십시오. 계정이 있는 경우, 로그인하세요.
2. https://dev.kik.com/ 에서 Kik 웹 사이트를 엽니다. 브라우저 창을 열어 둡니다.
3. Kik 앱에서 톱니바퀴 아이콘을 선택하여 설정을 연 다음 Your Kik Code 선택합니다.
4. Kik 웹사이트에서 Kik 코드를 스캔하여 보츠워스 챗봇을 엽니다. Yes를 선택하여 봇 대시보드를 엽니다.
5. Kik 앱에서 봇 만들기를 선택합니다. 지시에 따라 Kik 봇을 생성하세요.
6. 봇이 생성되면 브라우저에서 구성을 선택합니다. 새 봇이 선택되었는지 확인하세요.
7. 다음 섹션의 봇 이름과 API 키를 기록해 둡니다.

다음 단계

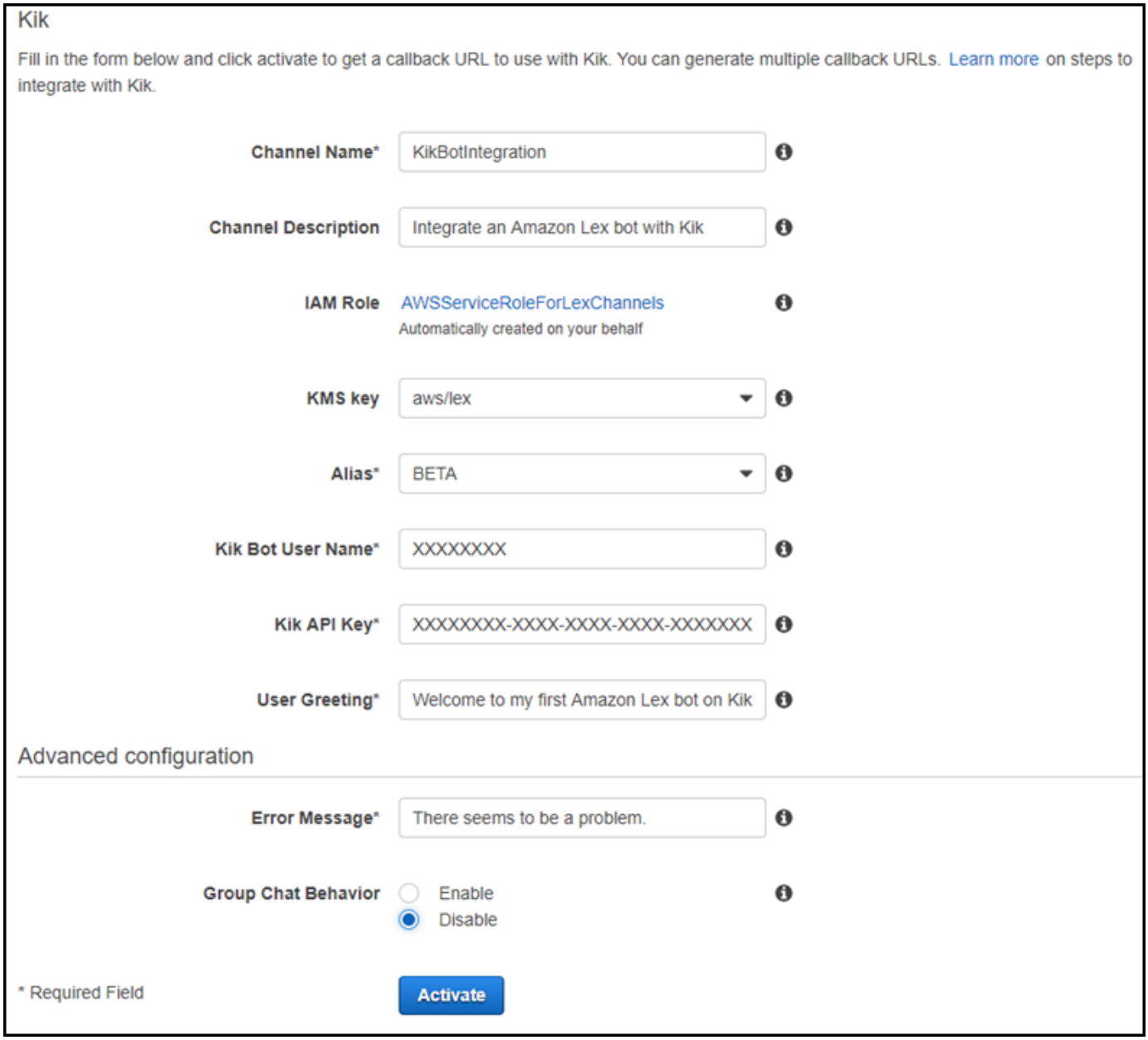
3단계: Kik 봇을 Amazon Lex 봇과 통합

3단계: Kik 봇을 Amazon Lex 봇과 통합

Amazon Lex 봇과 Kik 봇을 만들었으므로 이제 Amazon Lex에서 둘 사이에 채널 연결을 생성할 준비가 되었습니다. 연결이 활성화되면 Amazon Lex는 Kik을 사용하여 콜백 URL을 자동으로 설정합니다.

1. AWS Management Console 에 로그인하여 https://console.aws.amazon.com/lex/에서 Amazon Lex 콘솔을 엽니다.

2. 1단계에서 생성한 Amazon Lex 봇을 선택합니다.
3. 채널 탭을 선택합니다.
4. 채널 섹션에서 Kik을 선택합니다.
5. Kik 페이지에서 다음 사항을 제공합니다.
   - 이름 입력. 예: `BotKikIntegration`.
   - 설명을 입력합니다.
   - KMS 키 드롭다운에서 "aws/lex"를 선택합니다.
   - 별칭의 경우, 드롭다운에서 별칭을 선택합니다.
   - Kik 봇 사용자 이름의 경우 Kik에서 봇에게 부여한 이름을 입력합니다.
   - Kik API 키의 경우 Kik의 봇에 할당된 API 키를 입력합니다.
   - 사용자 인사말에는 사용자가 봇과 처음 채팅할 때 봇이 보내길 원하는 인사말을 입력합니다.
   - 오류 메시지에는 대화의 일부를 이해할 수 없을 때 사용자에게 표시되는 오류 메시지를 입력합니다.
   - 그룹 채팅 동작의 경우 다음 옵션 중 하나를 선택합니다.
     - 활성화 – 전체 채팅 그룹이 단일 대화로 봇과 상호 작용할 수 있습니다.
     - 비활성화 – 채팅 그룹 내 한 명의 사용자만 대화를 할 수 있도록 제한합니다.
   - 활성화를 선택하여 연결을 만들고 Kik 봇에 연결합니다.



다음 단계

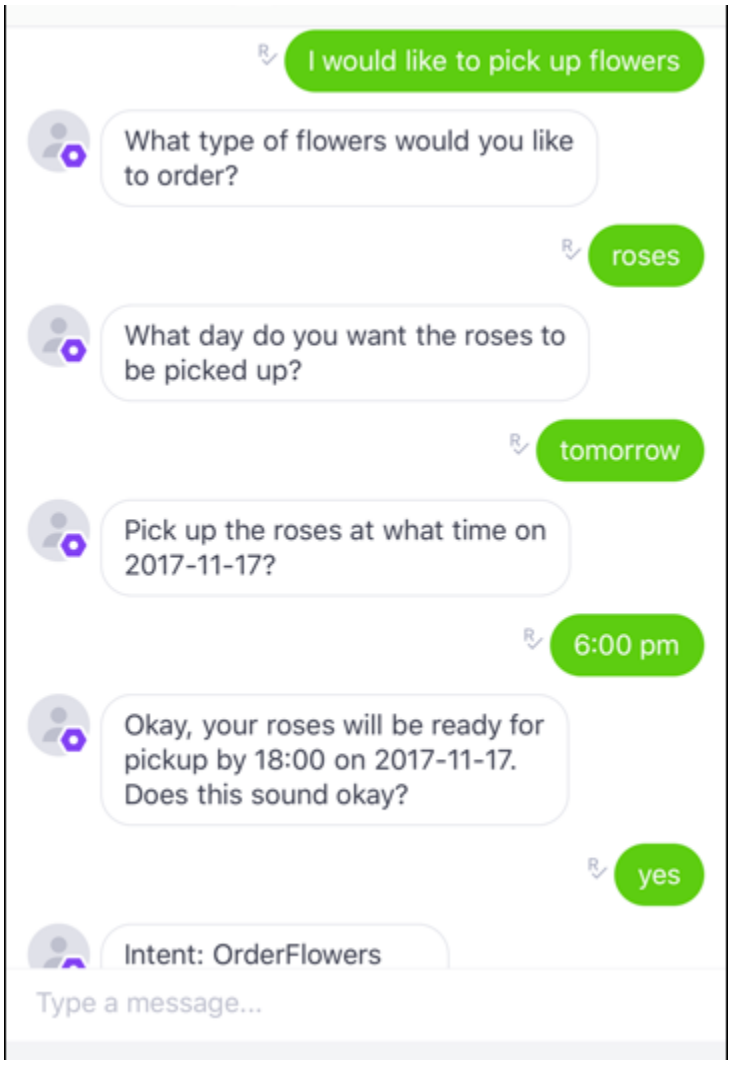
[4단계: 통합 테스트](#)

4단계: 통합 테스트

Amazon Lex 봇과 Kik 간에 연결을 생성했으므로 Kik 앱을 사용하여 연결을 테스트할 수 있습니다.

1. Kik 앱을 시작하고 로그인합니다. 생성한 봇을 선택합니다.

2. 다음과 같이 봇을 테스트할 수 있습니다.



각 구문을 입력하면 Amazon Lex 봇이 Kik을 통해 각 슬롯에 대해 생성한 프롬프트와 함께 응답합니다.

Amazon Lex 봇 슬랙에 통합하기

이번 연습에서는 Amazon Lex 봇을 슬랙 메시징 애플리케이션과 통합하기 위한 지침을 제공합니다. 다음 절차를 수행합니다.

1. Amazon Lex 봇 생성
2. 슬랙 메시징 애플리케이션을 생성합니다.
3. 슬랙 애플리케이션을 Amazon Lex 봇과 통합합니다.

4. Amazon Lex 봇과의 대화에 참여하여 통합을 테스트합니다. 슬랙 애플리케이션을 통해 메시지를 보내고 브라우저 창에서 테스트합니다.

주제

- 1단계: Amazon Lex 봇 생성
- 2단계: 슬랙에 가입하여 슬랙 팀 만들기
- 3단계: 슬랙 애플리케이션 생성
- 4단계: 슬랙 애플리케이션을 Amazon Lex 봇과 통합
- 5단계: 슬랙 통합 완료
- 6단계: 통합 테스트

1단계: Amazon Lex 봇 생성

아직 Amazon Lex 봇이 없는 경우 하나를 생성하고 배포합니다. 이 주제에서는 사용자가 시작하기 연습 1에서 생성한 봇을 사용하는 것으로 가정합니다. 그러나 이 설명서에서 제공된 모든 예제 봇을 사용할 수 있습니다. 시작하기 연습 1은 연습 1: 청사진을 사용하여 Amazon Lex 봇 생성(콘솔)을 참조하십시오.

1. Amazon Lex 봇 생성. 지침은 연습 1: 청사진을 사용하여 Amazon Lex 봇 생성(콘솔)을 참조하세요.
2. 봇을 배포하고 별칭을 생성합니다. 지침은 연습 3: 버전 게시 및 별칭 만들기을 참조하세요.

다음 단계

2단계: 슬랙에 가입하여 슬랙 팀 만들기

2단계: 슬랙에 가입하여 슬랙 팀 만들기

슬랙 계정에 가입하여 슬랙 팀을 만듭니다. 지침은 슬랙 사용을 참조하십시오. 다음 섹션에서는 모든 슬랙 팀이 설치할 수 있는 슬랙 애플리케이션을 만듭니다.

다음 단계

3단계: 슬랙 애플리케이션 생성

3단계: 슬랙 애플리케이션 생성

이 섹션에서는 다음 작업을 수행합니다.

1. 슬랙 API 콘솔에 슬랙 애플리케이션을 생성합니다.
2. 봇에 대화형 메시징을 추가하도록 애플리케이션을 구성합니다.

이 섹션의 마지막 부분에서 애플리케이션 보안 인증(클라이언트 ID, 클라이언트 암호, 확인 토큰)을 가져옵니다. 다음 섹션에서는 이 정보를 사용하여 Amazon Lex 콘솔에서 봇 채널 연결을 구성합니다.

1. http://api.slack.com 에서 슬랙 API 콘솔에 로그인합니다.
2. 애플리케이션을 생성합니다.

   애플리케이션을 만들면 슬랙은 해당 애플리케이션의 기본 정보 페이지를 표시합니다.
3. 다음과 같이 애플리케이션 기능을 구성합니다.
   - 왼쪽 메뉴에서 상호 작용 및 바로가기를 선택합니다.
     - 대화형 구성 요소를 실행하도록 토글을 선택합니다.
     - 요청 URL 상자에서 유효한 URL을 지정합니다. 예를 들어 **`https://slack.com`**를 사용할 수 있습니다.

       > **Note**
       > 이제 다음 단계에서 필요한 확인 토큰을 가져올 수 있도록 유효한 URL을 입력합니다. Amazon Lex 콘솔에서 봇 채널 연결을 추가한 후 이 URL을 업데이트합니다.
     - 변경 사항 저장을 선택합니다.
4. 왼쪽 메뉴의 설정에서 기본 정보를 선택합니다. 다음과 같은 애플리케이션 보안 인증을 기록합니다.
   - 클라이언트 ID
   - 클라이언트 암호
   - 확인 토큰

다음 단계

4단계: 슬랙 애플리케이션을 Amazon Lex 봇과 통합

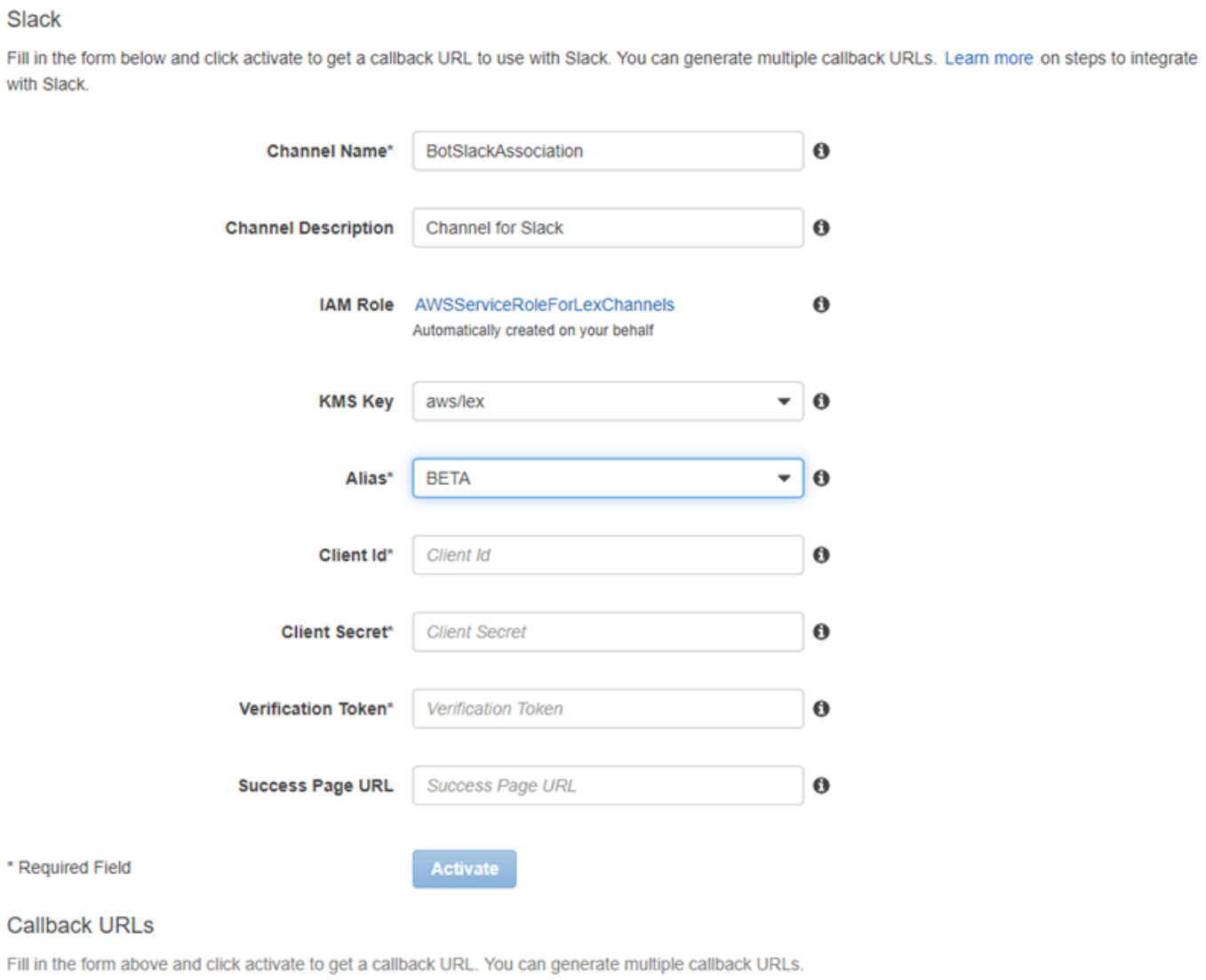
4단계: 슬랙 애플리케이션을 Amazon Lex 봇과 통합

이제 슬랙 애플리케이션 보안 인증이 있으므로 애플리케이션을 Amazon Lex 봇과 통합할 수 있습니다. 슬랙 애플리케이션을 Amazon Lex 봇과 연결하려면 에서 봇 채널 연결을 추가합니다.

Amazon Lex 콘솔에서 봇 채널 연결을 활성화하여 봇을 슬랙 애플리케이션과 연결합니다. 봇 채널 연결이 활성화되면 Amazon Lex 는 URL 두 개(Postback URL 및 OAuth URL)를 반환합니다. 나중에 필요하므로 이 URL을 기록해 둡니다.

슬랙 애플리케이션을 Amazon Lex 봇과 통합합니다.

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/lex/에서 Amazon Lex 콘솔을 엽니다.
2. 1단계에서 생성한 Amazon Lex 봇을 선택합니다.
3. 채널 탭을 선택합니다.
4. 왼쪽 메뉴에서 슬랙을 선택합니다.
5. 슬랙 페이지에서 다음 정보를 제공합니다.
   - 이름 입력. 예: `BotSlackIntegration`.
   - KMS 키 드롭다운에서 "aws/lex"를 선택합니다.
   - 별칭에서 봇 별칭을 선택합니다.
   - 이전 단계에서 기록해 둔 Client Id, Client secret 및 검증 토큰을 입력합니다. 다음은 슬랙 애플리케이션의 보안 인증입니다.



6. 활성화를 선택합니다.

   콘솔은 봇 채널 연결을 생성하고 URL 두 개(Postback URL 및 OAuth URL)를 반환합니다. 이를 기록해 둡니다. 다음 섹션에서는 이러한 엔드포인트를 사용하도록 슬랙 애플리케이션 구성을 다음과 같이 업데이트합니다.

   - Postback URL은 슬랙 이벤트를 수신하는 Amazon Lex 봇의 엔드포인트입니다. 이 URL은 다음과 같이 사용할 수 있습니다.
     - 슬랙 애플리케이션의 이벤트 구독 기능에 있는 요청 URL로 사용합니다
     - 슬랙 애플리케이션의 상호 작용 메시지 기능에 있는 요청 URL에 대한 자리 표시자 값을 바꾸기 위해 사용합니다.

- OAuth URL은 슬랙과의 OAuth 핸드셰이크를 위한 Amazon Lex 봇의 엔드포인트입니다.

다음 단계

5단계: 슬랙 통합 완료

5단계: 슬랙 통합 완료

이 섹션에서는 슬랙 API 콘솔을 사용하여 슬랙 애플리케이션의 구성을 완료합니다.

1. http://api.slack.com에서 슬랙 API 콘솔에 로그인합니다. 3단계: 슬랙 애플리케이션 생성에서 생성한 앱을 선택합니다.
2. 다음과 같이 OAuth 및 권한기능을 업데이트합니다.
   a. 왼쪽 메뉴에서 OAuth 및 권한을 선택합니다.
   b. 리디렉션 URL에서 이전 단계에서 Amazon Lex 가 제공한 OAuth URL을 추가합니다. 리디렉션 URL 추가를 선택한 후 URL 저장을 선택합니다.
   c. 봇 토큰 범위에서 OAuth 범위 추가 버튼을 사용하여 두 개의 권한을 추가합니다. 다음 텍스트를 사용하여 목록을 필터링합니다.
      - **`chat:write`**
      - **`team:read`**
3. 요청 URL 값을 Amazon Lex 가 이전 단계에서 제공한 Postback URL로 업데이트하여 상호작용 및 바로 가기 기능을 업데이트합니다. 4단계에서 저장한 postback URL을 입력한 다음 변경 사항 저장을 선택합니다.
4. 다음과 같이 이벤트 구독 기능을 구독합니다.
   - 켜기 옵션을 선택하여 이벤트를 활성화합니다.
   - 요청 URL 값을 Amazon Lex가 이전 단계에서 제공한 postback URL로 설정합니다.
   - 봇 이벤트 구독에서 `message.im` 봇 이벤트를 구독하여 최종 사용자와 슬랙 봇 간 직접 메시징을 활성화합니다.
   - 변경 사항을 저장합니다.
5. 다음과 같이 메시지 탭에서 메시지 전송을 활성화합니다.
   - 왼쪽 메뉴에서 앱 홈을 선택합니다.
   - 탭 표시 섹션에서 메시지 탭에서 사용자가 슬래시 명령 및 메시지를 보내도록 허용을 선택합니다.

다음 단계

6단계: 통합 테스트

6단계: 통합 테스트

이제 브라우저 창을 사용하여 Amazon Lex 봇과 슬랙의 통합을 테스트합니다.

1. 설정에서 배포 관리를 선택합니다. 슬랙에 추가를 선택하여 애플리케이션을 설치합니다. 메시지에 응답하도록 봇에 권한을 부여합니다.
2. 해당 슬랙 팀으로 리디렉션됩니다. 왼쪽 메뉴의 다이렉트 메시지 섹션에서 봇을 선택합니다. 봇이 보이지 않으면 다이렉트 메시지 옆의 더하기 아이콘(+)을 선택하여 봇을 검색합니다.
3. Amazon Lex 봇에 연결되어 있는 슬랙 애플리케이션과의 채팅에 참여합니다. 이제 봇이 메시지에 응답합니다.

   시작하기 연습 1을 참조하여 봇을 생성했다면 본 연습에 제공된 예제 대화를 사용할 수 있습니다. 자세한 내용은 4단계: Lambda 함수를 코드 후크로 추가(콘솔)을 참조하세요.

Amazon Lex 봇을 Twilio의 프로그래밍 가능한 SMS와 통합

이 연습에서는 Amazon Lex 봇을 Twilio의 SMS(Simple Messaging Service)와 통합하는 지침을 제공합니다. 다음 절차를 수행합니다.

1. Amazon Lex 봇 생성
2. Twilio의 프로그래밍 가능한 SMS를 봇 Amazon Lex와 통합합니다.
3. 휴대폰에서 SMS 서비스를 사용하여 설정을 테스트함으로써 Amazon Lex 봇과의 상호 작용에 참여합니다.
4. 통합 테스트

주제

- 1단계: Amazon Lex 봇 생성
- 2단계: Twilio SMS 계정 생성
- 3단계: Twilio 메시징 서비스 엔드포인트를 Amazon Lex 봇과 통합
- 4단계: 통합 테스트

1단계: Amazon Lex 봇 생성

아직 Amazon Lex 봇이 없는 경우 하나를 생성하고 배포합니다. 이 주제에서는 사용자가 시작하기 연습 1에서 생성한 봇을 사용하는 것으로 가정합니다. 그러나 이 설명서에서 제공된 모든 예제 봇을 사용할 수 있습니다. 시작하기 연습 1은 연습 1: 청사진을 사용하여 Amazon Lex 봇 생성(콘솔)을 참조하십시오.

1. Amazon Lex 봇 생성. 지침은 연습 1: 청사진을 사용하여 Amazon Lex 봇 생성(콘솔)을 참조하세요.
2. 봇을 배포하고 별칭을 생성합니다. 지침은 연습 3: 버전 게시 및 별칭 만들기 단원을 참조하십시오.

2단계: Twilio SMS 계정 생성

Twilio 계정에 가입하고 다음과 같은 계정 정보를 기록합니다.

- 계정 SID
- 인증 토큰

가입 지침은 https://www.twilio.com/console을 참조하세요.

3단계: Twilio 메시징 서비스 엔드포인트를 Amazon Lex 봇과 통합

Twilio를 Amazon Lex 봇과 통합하려면

1. Amazon Lex 봇을 Twilio의 프로그래밍 가능한 SMS 엔드포인트와 연결하려면 Amazon Lex 콘솔의 봇 채널 연결을 활성화합니다. 봇 채널 연결이 활성화되면 Amazon Lex는 콜백 URL을 반환합니다. 나중에 이 콜백 URL이 필요하므로 적어 둡니다.

   a. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lex/ Amazon Lex 콘솔을 엽니다.

   b. 1단계에서 생성한 Amazon Lex 봇을 선택합니다.

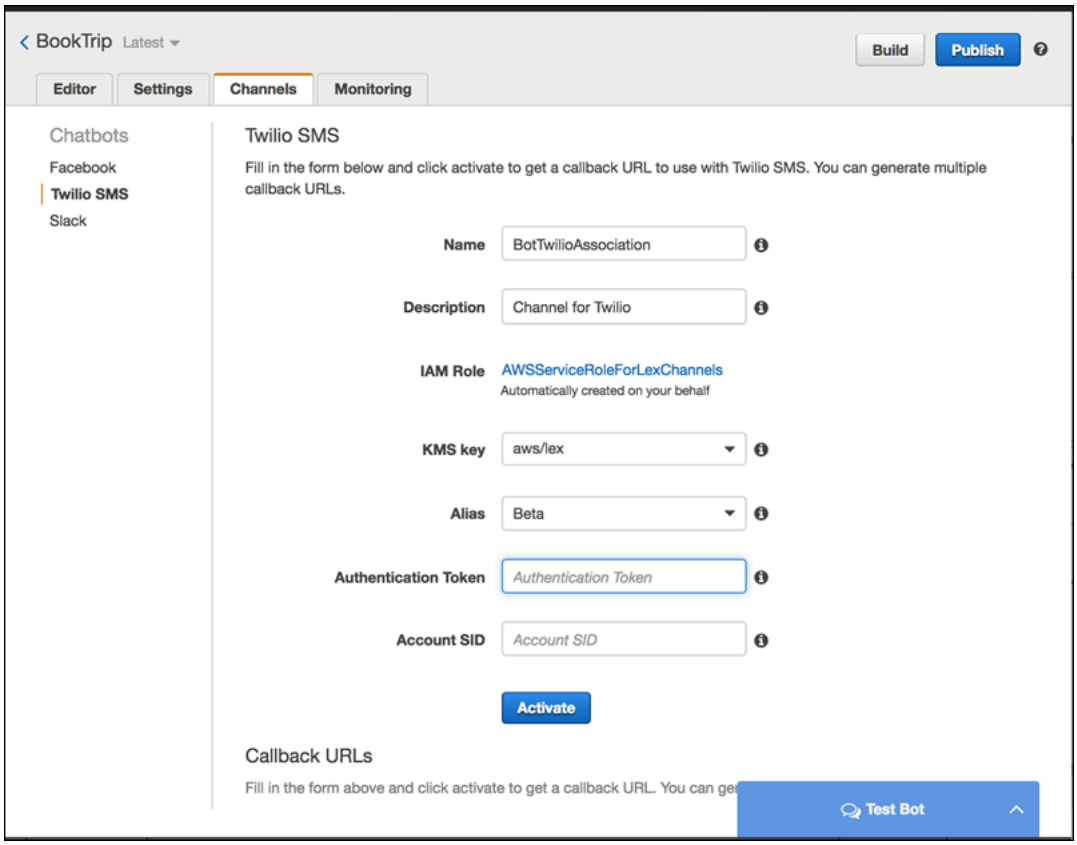
   c. 채널 탭을 선택합니다.

   d. 챗봇 섹션에서 Twilio SMS를 선택합니다.

   e. Twilio SMS 페이지에서 다음 정보를 제공합니다.

      - 이름 입력. 예: `BotTwilioAssociation`.
      - KMS 키에서 "aws/lex"를 선택합니다.

- Alias에서 봇 별칭을 선택합니다.
- 인증 토큰에 Twilio 계정의 인증 토큰을 입력합니다.
- 계정 SID에 Twilio 계정의 계정 SID를 입력합니다.



f. 활성화를 선택합니다.

콘솔은 봇 채널 연결을 생성하고 콜백 URL을 반환합니다. 이 URL을 기록해 두세요.

2. Twilio 콘솔에서 Twilio SMS 엔드포인트를 Amazon Lex 봇에 연결합니다.

   a. https://www.twilio.com/console에서 Twilio 콘솔에 로그인합니다.

   b. Twilio SMS 엔드포인트가 없는 경우 생성합니다.

   c. 요청 URL 값을 Amazon Lex가 이전 단계에서 제공한 콜백 URL로 설정하여 메시징 서비스의 수신 설정 구성을 업데이트합니다.

4단계: 통합 테스트

휴대폰을 사용하여 Twilio SMS와 봇 간의 상호 작용을 테스트합니다.

통합을 테스트하려면

1. https://www.twilio.com/console에서 Twilio 콘솔에 로그인한 후 다음을 수행합니다.

   a. 번호 관리 아래에서 메시징 서비스와 연결된 Twilio 번호가 있는지 확인합니다.

      휴대폰에서 이 번호로 메시지를 보내고, Amazon Lex 봇과 SMS 상호 작용에 참여합니다.

   b. 휴대폰이 확인된 발신자 ID에 리스트 되어 있는지 확인하세요.

      그렇지 않은 경우 Twilio 콘솔의 지침에 따라 테스트용으로 사용할 휴대폰을 사용 가능하도록 합니다.

      이제 휴대폰을 사용하여 Amazon Lex 봇에 매핑되는 Twilio SMS 엔드포인트에 메시지를 보낼 수 있습니다.

2. 휴대폰을 사용하여 Twilio 번호에 메시지를 보냅니다.

   Amazon Lex 봇이 응답합니다. 시작하기 연습 1을 참조하여 봇을 생성했다면 본 연습에 제공된 예제 대화를 사용할 수 있습니다. 자세한 내용은 4단계: Lambda 함수를 코드 후크로 추가(콘솔)을 참조하세요.

모바일 애플리케이션에서 Amazon Lex 봇 배포하기

AWS Amplify를 사용하면 Amazon Lex 봇을 모바일 또는 웹 애플리케이션과 통합할 수 있습니다. 자세한 내용은 AWS Amplify 문서의 상호작용 – 시작하기 를 참조하세요.

Amazon Lex 봇, 의도 및 슬롯 유형을 가져오고 내보내기

Amazon Lex 봇, 의도 또는 슬롯 유형을 가져오거나 내보낼 수 있습니다. 예를 들어 다른 AWS 계정의 동료와 봇을 공유하려면 봇을 내보낸 다음 동료에게 보낼 수 있습니다. 봇에 여러 개의 utterance를 추가하려면 봇을 내보내고 utterance를 추가한 다음 해당 봇을 계정으로 다시 가져올 수 있습니다.

Amazon Lex 봇, 인텐트 및 슬롯 유형을 Amazon Lex(공유 또는 수정) 또는 Alexa 스킬 형식으로 내보내기 할 수 있습니다. 가져오기는 Amazon Lex 형식으로만 가능합니다.

리소스를 내보낼 때는 내보내고 있는 대상 서비스, Amazon Lex 또는 Alexa Skills Kit와 호환되는 형식으로 내보내야 합니다. 봇을 Amazon Lex 형식으로 내보내는 경우 해당 봇을 내 계정으로 다시 가져올 수 있으며, 다른 계정의 Amazon Lex 사용자가 자신의 계정으로 가져올 수도 있습니다. 또한 Alexa Skill과 호환되는 형식으로 봇을 내보낼 수 있습니다. 그런 다음 Alexa Skills Kit를 사용하여 봇을 가져와서 Alexa와 함께 사용 가능하도록 설정할 수 있습니다. 자세한 내용은 Alexa Skill로 내보내기을 참조하세요.

봇, 의도 또는 슬롯 유형을 내보낼 때 해당 리소스는 JSON 파일에 기록됩니다. 봇, 의도 또는 슬롯 유형을 내보내기 위해 Amazon Lex 콘솔 또는 GetExport 작업을 사용할 수 있습니다. StartImport 섹션을 참조하여 봇, 의도 또는 슬롯 유형을 가져옵니다.

주제

- Amazon Lex 형식으로 내보내기 및 가져오기
- Alexa Skill로 내보내기

Amazon Lex 형식으로 내보내기 및 가져오기

Amazon Lex로 다시 가져올 목적으로 봇, 의도 및 슬롯 유형을 Amazon Lex로부터 내보내려면 Amazon Lex 형식의 JSON 파일을 사용합니다. 이 파일에서 리소스를 편집하고 Amazon Lex로 다시 가져올 수 있습니다. 예를 들어 의도에 utterance를 추가한 다음 변경된 의도를 계정으로 다시 가져올 수 있습니다. JSON 형식을 사용하여 리소스를 공유할 수도 있습니다. 예를 들어 한 AWS 리전 에서 봇을 내보낸 다음 다른 리전으로 가져올 수 있습니다. 또는 JSON 파일을 동료에게 보내 봇을 공유할 수 있습니다.

주제

- Amazon Lex 형식으로 내보내기

- Amazon Lex 형식으로 가져오기
- 내보내기 및 가져오기를 위한 JSON 형식

Amazon Lex 형식으로 내보내기

Amazon Lex 봇, 의도 및 슬롯 유형을 AWS 계정으로 가져올 수 있는 형식으로 내보냅니다. 다음과 같은 리소스를 내보낼 수 있습니다.

- 봇 - 봇에서 사용하는 모든 의도 및 사용자 지정 슬롯 유형 포함
- 의도 - 의도에서 사용하는 모든 사용자 지정 슬롯 유형 포함
- 사용자 지정 슬롯 유형 - 슬롯 유형의 모든 값 포함

번호가 지정된 버전의 리소스만 내보낼 수 있습니다. 리소스의 `$LATEST` 버전은 내보낼 수 없습니다.

내보내기는 비동기식 프로세스입니다. 내보내기가 완료되면 미리 서명된 Amazon S3 URL을 얻을 수 있습니다. URL은 내보낸 리소스가 포함된 .zip 아카이브의 위치를 JSON 형식으로 제공합니다.

콘솔 또는 GetExport 작업을 사용하여 봇, 의도 및 사용자 지정 슬롯 유형을 내보냅니다.

봇, 의도 또는 슬롯 유형 내보내기 절차는 동일합니다. 다음 절차에서는 봇에 대한 의도 또는 슬롯 유형을 대체합니다.

봇 내보내기

봇을 내보내려면

1. AWS Management Console 에 로그인하여 https://console.aws.amazon.com/lex/에서 Amazon Lex 콘솔을 엽니다.
2. 봇을 선택한 다음 내보낼 봇을 선택합니다.
3. 작업 메뉴에서 내보내기를 선택합니다.
4. 봇 내보내기 대화 상자에서 내보낼 봇의 버전을 선택합니다. 플랫폼으로 Amazon Lex 를 선택합니다.
5. 내보내기를 선택합니다.
6. .zip 아카이브를 다운로드하고 저장합니다.

Amazon Lex에서 .zip 아카이브에 포함된 JSON 파일로 봇을 내보냅니다. 봇을 업데이트하려면 JSON 텍스트를 수정한 다음 Amazon Lex로 다시 가져옵니다.

다음 단계

Amazon Lex 형식으로 가져오기

Amazon Lex 형식으로 가져오기

리소스를 Amazon Lex 형식의 JSON 파일로 내보낸 후 리소스가 포함된 JSON 파일을 하나 이상의 AWS 계정으로 가져올 수 있습니다. 예를 들어 한 봇을 내보낸 다음 다른 AWS 리전 으로 가져올 수 있습니다. 또는 동료가 봇을 자신의 계정으로 가져올 수 있도록 해당 봇을 동료에게 보낼 수 있습니다.

봇, 의도 또는 슬롯 유형을 가져올 때는 가져오기 도중에 의도 또는 슬롯 유형과 같은 리소스의 `$LATEST` 버전을 덮어쓸지 여부를 결정하거나, 계정에 있는 리소스를 보존하기 위해 가져오기가 실패하기를 원하는지 결정해야 합니다. 예를 들어 리소스의 편집된 버전을 계정에 업로드하는 경우 `$LATEST` 버전을 덮어쓰도록 선택할 수 있습니다. 동료가 보낸 리소스를 업로드하는 경우에는 리소스 충돌이 있으면 내 고유 리소스가 교체되지 않게 하기 위해 가져오기가 실패하도록 선택할 수 있습니다.

리소스를 가져올 때는 가져오기를 요청하는 사용자에게 할당되어 있는 권한이 적용됩니다. 사용자는 가져오기가 영향을 미치는 계정의 모든 리소스에 대한 권한이 있어야 합니다. 또한 사용자는 GetBot, PutBot, GetIntent, PutIntent, GetSlotType, PutSlotType 작업에 대한 권한이 있어야 합니다. 권한에 대한 자세한 내용은 Amazon Lex에서 IAM을 사용하는 방법을 참조하세요.

가져오기는 처리 중에 발생하는 오류를 보고합니다. 일부 오류는 가져오기가 시작되기 전에 보고되고 다른 오류는 가져오기 프로세스 중에 보고됩니다. 예를 들어 의도를 가져오는 계정에 해당 의도가 사용하는 Lambda 함수에 대한 호출 권한이 없는 경우, 슬롯 유형 또는 의도에 변경이 가해지기 전에 가져오기가 실패합니다. 가져오기 프로세스 중에 가져오기가 실패하는 경우 프로세스 실패 이전에 가져온 모든 의도 또는 슬롯 유형의 `$LATEST` 버전이 수정됩니다. `$LATEST` 버전에 만들어진 변경 사항을 롤백할 수 없습니다.

리소스를 가져오는 경우 모든 종속 리소스는 리소스의 `$LATEST` 버전으로 가져오기된 다음 번호가 지정된 버전을 부여받습니다. 예를 들어, 봇이 의도를 사용하는 경우 해당 의도에 번호가 지정된 버전이 부여되며, 의도가 사용자 지정 슬롯 유형을 사용하는 경우 해당 슬롯 유형에 번호가 지정된 버전이 부여됩니다.

리소스는 한 번만 가져오기됩니다. 예를 들어 봇에 `OrderPizza` 의도와 `OrderDrink` 의도가 포함되어 있고 두 의도 모두 사용자 지정 슬롯 유형 `Size`에 의존하는 경우, `Size` 슬롯 유형은 한 번 가져오기되며 두 의도 모두에 사용됩니다.

> **Note**
>
> `enableModelImprovements` 파라미터가 `false`로 설정된 상태로 봇을 내보낸 경우, 봇 정의가 포함된 .zip 파일을 열고 `enableModelImprovements` 파라미터를 다음 지역에서 `true`로 변경해야 합니다.
>
> - 아시아 태평양(싱가포르)(ap-southeast-1)
> - 아시아 태평양(도쿄)(ap-northeast-1)
> - EU(프랑크푸르트)(eu-central-1)
> - EU(런던)(eu-west-2)

봇, 의도 또는 사용자 지정 슬롯 유형 가져오기 절차는 동일합니다. 다음 절차에서는 상황에 맞게 의도 또는 슬롯 유형을 대체합니다.

봇 가져오기

봇을 가져오려면

1. AWS Management Console 에 로그인하여 https://console.aws.amazon.com/lex/에서 Amazon Lex 콘솔을 엽니다.
2. 봇을 선택한 다음 가져올 봇을 선택합니다. 새 봇을 가져오려면 이 단계를 건너뜁니다.
3. 작업에서 가져오기를 선택합니다.
4. 봇 가져오기에서 가져올 봇이 포함된 JSON 파일을 포함하는 .zip 아카이브를 선택합니다. 병합 전에 병합 충돌을 확인하려면 병합 충돌 알림을 선택합니다. 충돌 확인 기능을 끄면 봇에서 사용되는 모든 리소스의 `$LATEST` 버전이 덮어쓰기됩니다.
5. 가져오기를 선택합니다. 병합 충돌에 대한 알림을 받도록 선택한 경우 충돌이 있으면 해당 충돌이 리스트된 대화 상자가 나타납니다. 충돌하는 모든 리소스의 `$LATEST` 버전을 덮어쓰려면 덮어쓰고 계속을 선택합니다. 가져오기를 중지하려면 취소를 선택합니다.

이제 계정에서 봇을 테스트할 수 있습니다.

내보내기 및 가져오기를 위한 JSON 형식

다음 예에서는 Amazon Lex 형식의 봇, 의도 및 슬롯 유형을 내보내고 가져오기 위한 JSON 구조를 보여 줍니다.

슬롯 유형 구조

다음은 사용자 지정 슬롯 유형에 대한 JSON 구조입니다. 슬롯 유형을 가져오거나 내보낼 때 그리고 사용자 정의 슬롯 유형에 의존하는 의도를 내보낼 때 이 구조를 사용하세요.

```
{
  "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"
  }
}
```

의도 구조

다음은 의도에 대한 JSON 구조입니다. 의도 및 의도에 의존하는 봇을 가져오거나 내보낼 때 이 구조를 사용하십세요.

```
{
  "metadata": {
    "schemaVersion": "1.0",
    "importType": "LEX",
    "importFormat": "JSON"
  },
  "resource": {
    "description": "intent description",
    "rejectionStatement": {
      "messages": [
```

```
        {
          "contentType": "PlainText or SSML or CustomPayload",
          "content": "string"
        }
      ]
    },
    "name": "intent name",
    "version": "version number",
    "fulfillmentActivity": {
      "type": "ReturnIntent or CodeHook"
    },
    "sampleUtterances": [
      "string",
      "string"
    ],
    "slots": [
      {
        "name": "slot name",
        "description": "slot description",
        "slotConstraint": "Required or Optional",
        "slotType": "slot type",
        "valueElicitationPrompt": {
          "messages": [
            {
              "contentType": "PlainText or SSML or CustomPayload",
              "content": "string"
            }
          ],
          "maxAttempts": value
        },
        "priority": value,
        "sampleUtterances": []
      }
    ],
    "confirmationPrompt": {
      "messages": [
        {
          "contentType": "PlainText or SSML or CustomPayload",
          "content": "string"
        },
        {
          "contentType": "PlainText or SSML or CustomPayload",
          "content": "string"
        }
```

```
      ],
      "maxAttempts": value
    },
    "slotTypes": [
        List of slot type JSON structures.
        For more information, see ## ## ##.
    ]
  }
}
```

봇 구조

다음은 봇에 대한 JSON 구조입니다. 봇을 가져오거나 내보낼 때 이 구조를 사용하세요.

```
{
  "metadata": {
    "schemaVersion": "1.0",
    "importType": "LEX",
    "importFormat": "JSON"
  },
  "resource": {
    "name": "bot name",
    "version": "version number",,
    "nluIntentConfidenceThreshold": 0.00-1.00,
    "enableModelImprovements": true | false,
    "intents": [
        List of intent JSON structures.
        For more information, see ## ##.
    ],
    "slotTypes": [
        List of slot type JSON structures.
        For more information, see ## ## ##.
    ],
    "voiceId": "output voice ID",
    "childDirected": boolean,
    "locale": "en-US",
    "idleSessionTTLInSeconds": timeout,
    "description": "bot description",
    "clarificationPrompt": {
      "messages": [
        {
          "contentType": "PlainText or SSML or CustomPayload",
          "content": "string"
```

```
        }
      ],
      "maxAttempts": value
    },
    "abortStatement": {
      "messages": [
        {
          "contentType": "PlainText or SSML or CustomPayload",
          "content": "string"
        }
      ]
    }
  }
}
```

Alexa Skill로 내보내기

Alexa Skill과 호환 가능한 형식으로 봇 스키마를 내보낼 수 있습니다. 봇을 JSON 파일로 내보낸 후 스킬 빌더를 사용하여 Alexa로 봇을 업로드합니다.

봇과 해당 스키마를 내보내려면(상호 작용 모델)

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lex/ Amazon Lex 콘솔을 엽니다.
2. 내보내려는 봇을 선택합니다.
3. 작업에서 내보내기를 선택합니다.
4. 내보내려는 봇의 버전을 선택합니다. 형식으로 Alexa Skills Kit를 선택한 다음 내보내기를 선택합니다.
5. 다운로드 대화 상자가 나타나면 파일을 저장할 위치를 선택한 후 저장을 선택합니다.

다운로드한 파일은 내보낸 봇의 이름을 사용하는 파일 하나가 포함된 .zip 아카이브입니다. 여기에는 봇을 Alexa skill로 가져오는 데 필요한 정보가 들어 있습니다.

> **Note**
>
> Amazon Lex와 Alexa Skills Kit는 다음과 같은 부분에서 서로 다릅니다.
>
> - 대괄호([])로 표시되는 세션 속성은 Alexa Skills Kit에서 지원하지 않습니다. 세션 속성을 사용하는 프롬프트를 업데이트해야 합니다.

- 구두점은 Alexa Skills Kit에서 지원되지 않습니다. 구두점을 사용하는 utterance를 업데이트해야 합니다.

봇을 Alexa Skill에 업로드하려면

1. 개발자 포털 https://developer.amazon.com/에 로그인합니다.
2. Alexa Skills 페이지에서 스킬 생성을 선택합니다.
3. 새 스킬 생성 페이지에서 스킬 이름과 스킬의 기본 언어를 입력합니다. 스킬 모델로 사용자 지정이 선택되었는지 확인한 후 스킬 생성을 선택합니다.
4. 새로 시작을 선택했는지 확인한 후 선택을 선택합니다.
5. 왼쪽 메뉴에서 JSON 편집기를 선택합니다. 내보내려는 JSON 파일을 Amazon Lex에서 JSON 편집기로 끌어 놓습니다.
6. 모델 저장을 선택하여 상호 작용 모델을 저장합니다.

스키마를 Alexa Skill로 업로드한 후에는 Alexa를 사용해 스킬을 실행하는 데 필요한 사항을 변경하세요. Alexa 스킬 생성에 대한 자세한 내용은 Alexa Skill Kit의 Skill 빌더 사용(베타)을 참조하십시오.

추가 예제: Amazon Lex 봇 생성

다음 섹션에서는 추가 Amazon Lex 연습과 단계별 지침을 제공합니다.

주제

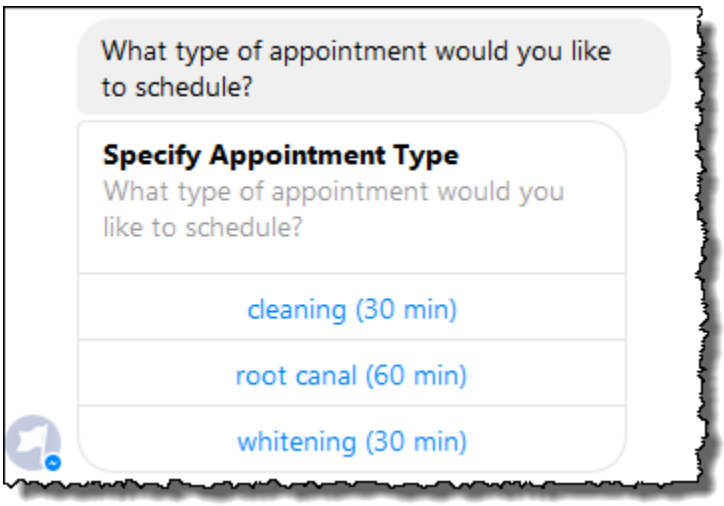
- 스케줄 예약
- 여행 예약
- 응답 카드 사용
- 표현 업데이트
- 웹 사이트와의 통합
- 콜센터 에이전트 어시스턴트

스케줄 예약

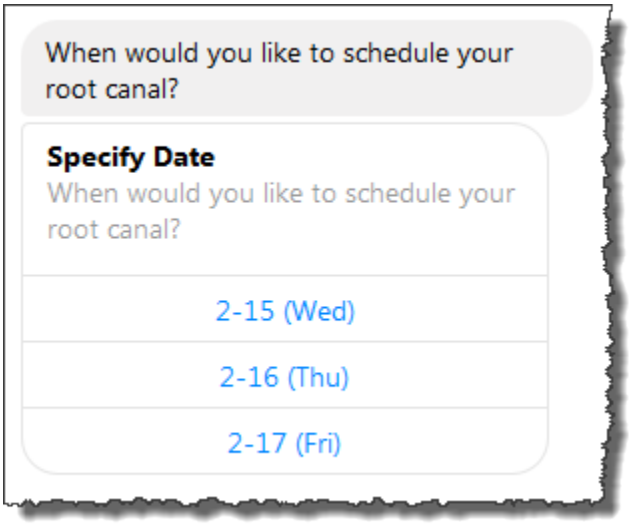
이번 봇 예제에서는 치과에서 사용하는 스케줄 예약을 연습합니다. 또한 이 예제에서는 응답 카드를 사용하여 버튼으로 사용자 입력을 가져오는 방법을 보여줍니다. 특히 예제에서는 런타임에서 동적으로 응답 카드를 생성하는 방법을 보여 줍니다.

빌드 시 응답 카드(정적 응답 카드라고도 함)를 구성하거나 AWS Lambda 함수에서 동적으로 생성할 수 있습니다. 이 예제에서 봇은 다음의 응답 카드를 사용합니다.

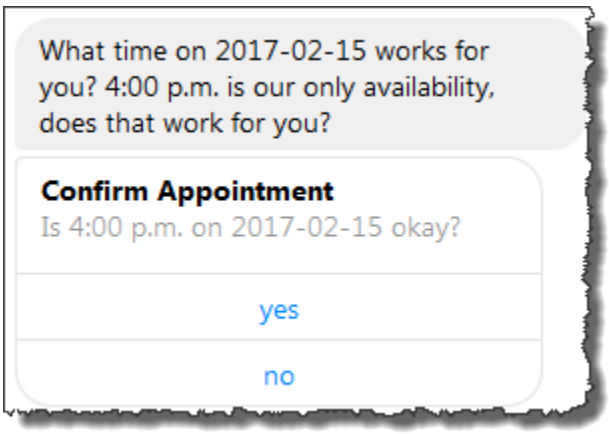
- 예약 유형에 대한 버튼을 목록화하는 응답 카드. 다음 이미지를 예시로 참조하세요.



- 예약 날짜에 대한 버튼을 목록화하는 응답 카드. 다음 이미지를 예시로 참조하세요.



- 재안된 예약 시간을 확인하기 위한 버튼을 목록화하는 응답 카드. 다음 이미지를 예시로 참조하세요.



예약 가능한 약속 날짜와 시간이 변하므로 런타임 시에 응답 카드를 생성해야 합니다. AWS Lambda 함수를 사용하여 이러한 응답 카드를 동적으로 생성합니다. Lambda 함수는 Amazon Lex에 대한 응답으로 응답 카드를 반환합니다. Amazon Lex에는 클라이언트에 대한 응답 카드가 포함되어 있습니다.

클라이언트(예: Facebook Messenger)가 응답 카드를 지원하는 경우, 사용자는 버튼 목록에서 선택하거나 응답을 입력할 수 있습니다. 그 외의 경우에는, 단순히 응답을 입력합니다.

앞의 예제에 표시된 버튼 외에도 응답 카드에 표시할 이미지, 첨부 파일, 기타 유용한 정보를 포함시킬 수 있습니다. 응답 카드에 대한 자세한 내용은 응답 카드을 참조하십시오.

이번 연습에서는, 다음 작업을 수행합니다.

- 봇 생성 및 테스트(ScheduleAppointment 청사진 사용). 이 연습에서는 봇 청사진을 사용하여 봇을 빠르게 설정하고 테스트합니다. 사용 가능한 청사진 목록은 Amazon Lex와 AWS Lambda 청사진을 참조하십시오. 이 봇은 하나의 의도(`MakeAppointment`)로 미리 구성되어 있습니다.

- Lambda 함수 생성 및 테스트(Lambda에서 제공한 lex-make-appointment-python 청사진 사용). 이 Lambda 함수를 코드 후크로 사용하도록 `MakeAppointment` 의도를 구성하여 초기화, 검증 및 이행 작업을 수행합니다.

> **Note**
>
> 제시된 Lambda 함수 예제에서는 가상의 치과 진료 예약 가용성을 기반으로 한 동적 대화를 보여 줍니다. 실제 application에서는 실제 달력을 사용하여 예약할 수 있습니다.

- Lambda 함수를 코드 후크로 사용하도록 `MakeAppointment` 의도 구성을 업데이트합니다. 그런 다음 종합 경험을 테스트합니다.
- 실행 중인 응답 카드를 볼 수 있도록 Facebook Messenger에 일정 약속 봇을 게시합니다(Amazon Lex 콘솔의 클라이언트는 현재 응답 카드를 지원하지 않음).

다음 섹션에서는 이 연습에서 사용하는 블루프린에 대한 요약 정보를 제공합니다.

주제

- 봇 청사진(ScheduleAppointment) 개요
- Lambda 함수 청사진(lex-make-appointment-python) 개요
- 1단계: Amazon Lex 봇 생성
- 2단계: Lambda 함수 생성
- 3단계: 의도 업데이트: 코드 후크 구성
- 4단계: Facebook Messenger 플랫폼에 봇 배포
- 정보 흐름의 세부 정보

봇 청사진(ScheduleAppointment) 개요

이번 연습에서 봇을 생성하는 데 사용하는 ScheduleAppointment 청사진은 다음과 같이 미리 구성되어 있습니다.

- 슬롯 유형 – `AppointmentTypeValue`란 이름의 1개의 사용자 지정 슬롯 유형과 다음과 같은 열거 값: `root canal`, `cleaning`및 `whitening`
- 의도 – 다음과 같이 미리 구성된 의도 한 개(`MakeAppointment`)
  - 슬롯 – 의도는 다음 슬롯으로 구성되어 있습니다.
    - `AppointmentTypes` 사용자 지정 유형의 슬롯 `AppointmentType`.
    - `AMAZON.DATE` 기본 제공 유형의 슬롯 `Date`.
    - `AMAZON.TIME` 기본 제공 유형의 슬롯 `Time`.
  - 표현 – 의도는 다음 표현으로 미리 구성되어 있습니다.
    - "예약을 진행하고 싶습니다"
    - "예약 진행"
    - "{AppointmentType} 예약"

    사용자가 이러한 표현을 하면 Amazon Lex는 `MakeAppointment`가 의도라고 판단하고 프롬프트를 사용하여 슬롯 데이터를 유도합니다.
  - 프롬프트 – 의도는 다음 프롬프트로 미리 구성되어 있습니다.
    - `AppointmentType` 슬롯에 대한 프롬프트 – "어떤 유형의 예약을 진행하고 싶으세요?"
    - `Date` 슬롯에 대한 프롬프트 – "{AppointmentType} 예약을 언제 진행할까요?”
    - `Time` 슬롯에 대한 프롬프트 – "{AppointmentType} 예약을 몇시로 할까요?" and

      "{Date}의 몇시인가요?"
    - 확인 프롬프트 – "{Time}에 가능합니다. 예약을 진행할까요?"
    - 취소 메시지 – "네, 더 이상 진행하지 않겠습니다."

Lambda 함수 청사진(lex-make-appointment-python) 개요

Lambda 함수 청사진(lex-make-appointment-python)는 ScheduleAppointment 봇 청사진을 사용하여 생성한 봇의 코드 후크입니다.

이 Lambda 함수 청사진 코드는 초기화/검증 및 이행 작업을 모두 수행할 수 있습니다.

- 이 Lambda 함수 코드에서는 치과 진료 예약의 예를 토대로 한 동적 대화를 보여 줍니다(실제 애플리케이션에서는 달력을 사용할 수 있음). 사용자가 지정하는 요일 또는 날짜에 대해 이 코드는 다음과 같이 구성됩니다.

- 예약 가능한 요일 또는 날짜가 없는 경우, Lambda 함수는 Amazon Lex 로 다이렉팅하여 사용자에게 다른 요일 또는 날짜를 선택하도록 하는 응답을 반환합니다(`dialogAction` 유형을 `ElicitSlot`)로 설정하여). 자세한 내용은 응답 형식을 참조하세요.
- 지정된 요일 또는 날짜에 예약 가능한 시간이 하나 뿐인 경우, Lambda 함수는 응답으로서 예약 가능한 시간을 제안하고 Amazon Lex를 다이렉팅하여 사용자의 확인을 얻습니다(`ConfirmIntent`에 대한 응답으로 `dialogAction`을 설정하여). 이는 예약 가능한 약속 시간을 사전에 제시하여 사용자 경험을 향상시킬 수 있는 방법을 보여 줍니다.
- 예약 가능한 시간이 여러 개인 경우, Lambda 함수는 Amazon Lex에 대한 응답으로 예약 가능한 시간 목록을 반환합니다. Amazon Lex는 Lambda 함수부터 받은 메시지가 포함된 응답을 반환합니다.

- 이행 코드 후크로서 Lambda 함수는 약속이 예약되었음(즉, 의도가 이행됨)을 나타내는 요약 메시지를 반환합니다.

> **Note**
>
> 이 예제에서는 응답 카드를 사용하는 방법을 보여 줍니다. Lambda 함수는 Amazon Lex에 대한 응답 카드를 구성하고 반환합니다. 응답 카드에는 예약 가능한 날짜와 시간이 선택할 수 있는 버튼으로 나열되어 있습니다. Amazon Lex 콘솔에서 제공한 클라이언트를 사용하여 봇을 테스트하는 경우 응답 카드가 표시되지 않습니다. 응답 카드를 보려면 봇을 Facebook Messenger와 같은 메시징 플랫폼에 통합해야 합니다. 지침은 Amazon Lex 봇을 Facebook Messenger와 통합하기을 참조하세요. 응답 카드에 대한 자세한 내용은 메시지 관리 을 참조하십시오.

Amazon Lex가 Lambda 함수를 간접 호출할 때 이벤트 데이터를 입력으로 전달합니다. 이벤트 필드 중 하나는 `invocationSource`이며, Lambda 함수는 이 필드를 사용하여 입력 검증과 이행 활동 중 하나를 선택합니다. 자세한 내용은 입력 이벤트 형식을 참조하세요.

다음 단계

1단계: Amazon Lex 봇 생성

1단계: Amazon Lex 봇 생성

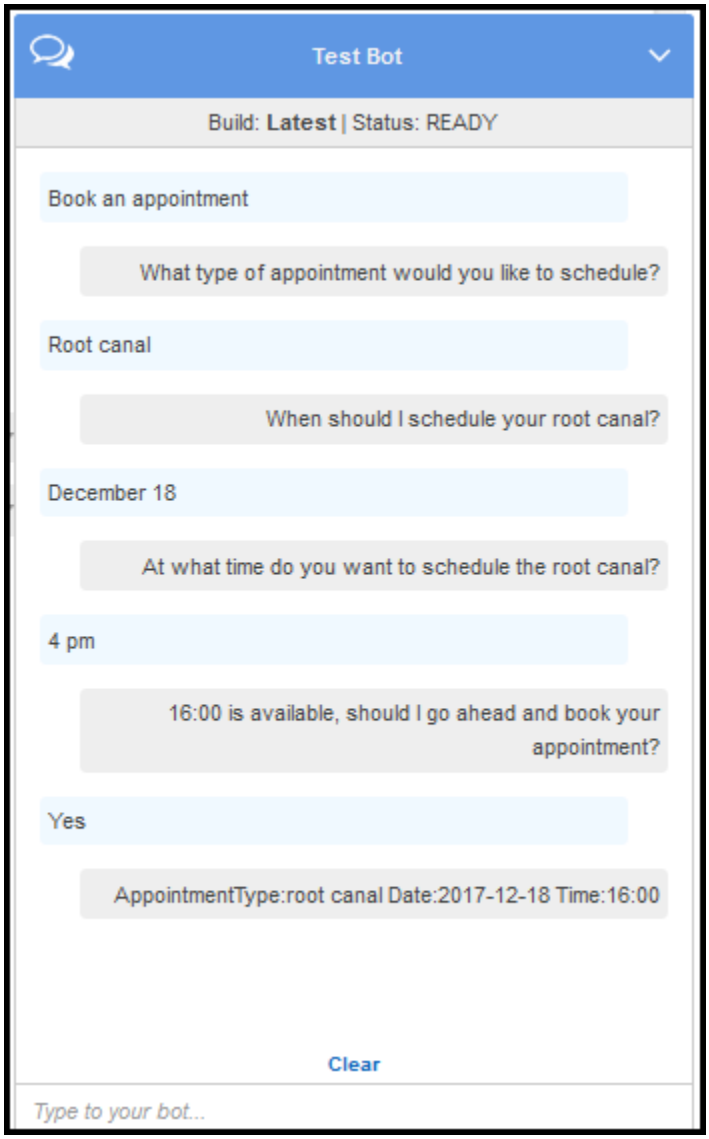
이 섹션에서는 Amazon Lex 콘솔에 제공된 ScheduleAppointment 청사진을 사용하여 Amazon Lex 봇을 생성합니다.

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lex/ Amazon Lex 콘솔을 엽니다.
2. 봇 페이지에서 생성을 선택합니다.
3. Lex bot 생성 페이지에서 다음 작업을 수행합니다.
   - ScheduleAppointment 청사진을 선택합니다.
   - 봇 이름의 기본값(ScheduleAppointment)을 그대로 둡니다.
4. 생성(Create)을 선택합니다.

   이 단계에서는 봇을 저장하고 구축합니다. 콘솔은 빌드 프로세스 중에 Amazon Lex에 다음 요청을 보냅니다.

   - 슬롯 유형의 새 버전을($LATEST 버전으로부터) 생성합니다. 이 봇 청사진에서 정의된 슬롯 유형에 대한 자세한 내용은 봇 청사진(ScheduleAppointment) 개요을 참조하십시오.
   - `MakeAppointment` 의도의 버전을($LATEST 버전으로부터) 생성합니다. 경우에 따라 콘솔은 새 버전을 생성하기 전에 `update` API 작업에 대한 요청을 보냅니다.
   - $LATEST 버전의 봇을 업데이트합니다.

     이때, Amazon Lex가 봇을 위한 기계 학습 모델을 구축합니다. 콘솔에서 봇을 테스트할 경우, 콘솔은 런타임 API를 사용하여 사용자 입력을 Amazon Lex에 되돌려 보냅니다. 그 뒤에 Amazon Lex는 기계 학습 모델을 사용하여 사용자 입력을 해석합니다.
5. 콘솔에 ScheduleAppointment 봇이 표시됩니다. 편집기 탭에서, 미리 구성된 의도(`MakeAppointment`)의 세부 정보를 검토합니다.
6. 테스트 창에서 봇을 테스트합니다. 다음 스크린샷을 사용하여 사용자의 봇을 상대로 테스트 대화에 참여합니다.



참고사항

- 초기 사용자 입력("예약 진행")에서 봇은 의도(`MakeAppointment`)를 유추합니다.
- 그러면 봇은 구성된 프롬프트를 사용하여 사용자로부터 슬롯 데이터를 얻습니다.
- 봇 청사진에는 다음 확인 프롬프트로 구성된 `MakeAppointment` 의도가 있습니다.

```
{Time} is available, should I go ahead and book your appointment?
```

사용자가 모든 슬롯 데이터를 제공하면, Amazon Lex는 확인 프롬프트가 포함된 응답을 클라이언트에 메시지로 반환합니다. 클라이언트는 사용자에게 다음 메시지를 표시합니다.

```
16:00 is available, should I go ahead and book your appointment?
```

사용자 데이터를 초기화하거나 검증할 코드가 없으므로 봇은 모든 약속 날짜 및 시간 값을 수락합니다. 다음 섹션에서는 Lambda 함수를 추가하여 이를 수행합니다.

다음 단계

2단계: Lambda 함수 생성

2단계: Lambda 함수 생성

이 섹션에서는 콘솔에서 제공된 청사진(lex-make-appointment-python)을 사용하여 Lambda 함수를 생성합니다. 또한 콘솔에서 제공하는 샘플 Amazon Lex 이벤트 데이터를 사용하여 함수를 호출함으로써 이 Lambda 함수를 테스트합니다.

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lambda/ AWS Lambda 콘솔을 엽니다.
2. Lambda 함수 생성을 선택합니다.
3. 청사진 선택에 **`lex`**를 입력하여 청사진을 찾은 다음, lex-make-appointment-python 청사진을 선택합니다.
4. Lambda 함수를 다음과 같이 구성합니다.
   - Lambda 함수(`MakeAppointmentCodeHook`)를 입력합니다.
   - 역할의 경우 템플릿에서 새 역할 생성(Create a new role from template(s))을 선택한 다음 역할 이름을 입력합니다.
   - 다른 기본값을 그대로 둡니다.
5. Create Function(함수 생성)을 선택합니다.
6. 영어(미국)(en-US) 이외의 지역을 사용하는 경우 특정 로캘에 대한 청사진 업데이트에 설명된 대로 의도 이름을 업데이트하십시오.
7. Lambda 함수 테스트.
   a. 액션(Actions)과 테스트 이벤트 구성(Configure test event)을 차례로 선택합니다.
   b. 샘플 이벤트 템플릿 목록에서 Lex-Make Appointment(미리보기)를 선택합니다. 이 샘플 이벤트는 Amazon Lex 봇의 요청과 일치하도록 값이 설정된 Amazon Lex 요청 및 응답 모델을 사

용합니다. Amazon Lex 요청 및 응답 모델에 대한 자세한 내용은 Lambda 함수 사용을 참조하십시오.

c. 저장 및 테스트를 선택합니다.

d. Lambda 함수가 성공적으로 실행되었는지 확인합니다. 이 경우 응답은 Amazon Lex 응답 모델과 일치합니다.

다음 단계

3단계: 의도 업데이트: 코드 후크 구성

3단계: 의도 업데이트: 코드 후크 구성

이 섹션에서는 Lambda 함수를 검증 및 이행 활동을 위한 코드 후크로 사용하도록 `MakeAppointment` 의도의 구성을 업데이트합니다.

1. Amazon Lex 콘솔에서 ScheduleAppointment 봇을 선택합니다. 콘솔은 MakeAppointment 의도를 보여줍니다. 의도 구성을 다음과 같이 수정합니다.

> **Note**
>
> 의도를 포함하여 모든 Amazon Lex 리소스 중 하나의 $LATEST 버전만 업데이트할 수 있습니다. 의도 버전이 $LATEST로 설정되어 있어야 합니다. 아직 봇의 버전을 게시하지 않았으므로 콘솔에서 여전히 $LATEST 버전일 것입니다.

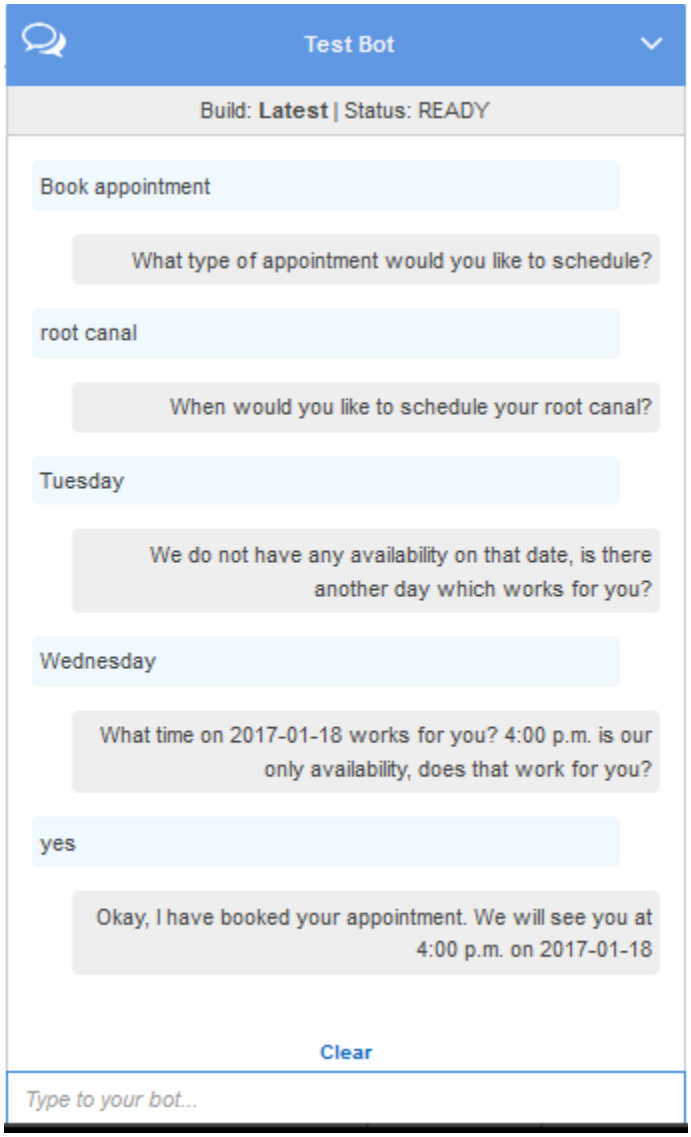
   a. 옵션 섹션에서 초기화 및 검증 코드 후크를 선택한 다음, 목록에서 Lambda 함수를 선택합니다.

   b. 이행 섹션에서 AWS Lambda 함수를 선택한 다음, 목록에서 Lambda 함수를 선택합니다.

   c. 종료 메시지를 선택하고 메시지를 입력합니다.

2. 저장을 선택한 후 빌드를 선택합니다.

3. 다음 이미지와 같이 봇을 테스트하십시오.



다음 단계

4단계: Facebook Messenger 플랫폼에 봇 배포

4단계: Facebook Messenger 플랫폼에 봇 배포

이전 섹션에서는 Amazon Lex 콘솔의 클라이언트를 사용하여 ScheduleAppointment 봇을 테스트했습니다. 현재 Amazon Lex 콘솔은 응답 카드를 지원하지 않습니다. 봇이 지원하는 동적으로 생성된 응답 카드를 테스트하려면, Facebook Messenger 플랫폼에 봇을 배포하여 테스트합니다.

지침은 Amazon Lex 봇을 Facebook Messenger와 통합하기을 참조하세요.

다음 단계

정보 흐름의 세부 정보

정보 흐름의 세부 정보

ScheduleAppointment 봇 청사진은 주로 동적으로 생성된 응답 카드의 사용을 보여 줍니다. 이 실습의 Lambda 함수 에는 Amazon Lex에 대한 응답 카드가 포함되어 있습니다. Amazon Lex에는 클라이언트에 대한 회신 응답 카드가 포함되어 있습니다. 이 섹션에서는 다음 두 가지에 대해 설명합니다.

- 클라이언트와 Amazon Lex 간의 데이터 흐름.

  이 섹션에서는 클라이언트가 PostText 런타임 API를 사용하여 Amazon Lex에 요청을 전송하고 그에 따라 요청 및 응답의 세부 정보를 보여 준다고 가정합니다. PostText 런타임 API에 대한 자세한 내용은 PostText를 참조하십시오.

  > **Note**
  > 클라이언트가 PostContent API를 사용할 때의 클라이언트와 Amazon Lex 간의 정보 흐름의 예는 2a단계(선택 사항): 음성 정보 흐름의 세부 정보 검토(콘솔) 를 참조하십시오.

- Amazon Lex와 Lambda 함수 간의 데이터 흐름. 자세한 내용은 Lambda 함수 입력 이벤트 및 응답 형식을 참조하세요.

> **Note**
> 이 예제에서는 요청에서 세션 속성을 Amazon Lex에 전달하지 않는 Facebook Messenger 클라이언트를 사용한다고 가정합니다. 따라서, 이 섹션에 나와 있는 예제 요청은 빈 sessionAttributes을 표시합니다. Amazon Lex 콘솔에 제공된 클라이언트를 사용하여 봇을 테스트할 경우, 클라이언트에 세션 속성이 포함됩니다.

이 섹션에서는 각 사용자가 입력한 후에 일어나는 상황을 설명합니다.

1. 사용자: 유형 **Book an appointment**.

   a. 클라이언트(콘솔)는 Amazon Lex에 다음 PostContent 요청을 보냅니다.

```
POST /bot/ScheduleAppointment/alias/$LATEST/
user/bijt6rovckwecnzesbthrr1d7lv3ja3n/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText":"book appointment",
   "sessionAttributes":{}
}
```

요청 URI 및 본문 모두 Amazon Lex에 다음 정보를 제공합니다.

- URI 요청 – 봇 이름(ScheduleAppointment), 봇 별칭($LATEST), 사용자 이름 ID를 제공합니다. 후행 `text`은 이것이 `PostText`(`PostContent` 아님) API 요청임을 나타냅니다.
- 요청 본문 – 사용자 입력(`inputText`)과 빈 `sessionAttributes`를 포함합니다.

b. `inputText`에서 Amazon Lex는 의도(`MakeAppointment`)를 감지합니다. 이 서비스는 코드 후크로 구성된 Lambda 함수를 간접 호출하여 다음 이벤트를 전달함으로써 초기화 및 검증을 수행합니다. 자세한 내용은 입력 이벤트 형식을 참조하세요.

```
{
    "currentIntent": {
        "slots": {
            "AppointmentType": null,
            "Date": null,
            "Time": null
        },
        "name": "MakeAppointment",
        "confirmationStatus": "None"
    },
    "bot": {
        "alias": null,
        "version": "$LATEST",
        "name": "ScheduleAppointment"
    },
    "userId": "bijt6rovckwecnzesbthrr1d7lv3ja3n",
    "invocationSource": "DialogCodeHook",
    "outputDialogMode": "Text",
    "messageVersion": "1.0",
    "sessionAttributes": {}
}
```

클라이언트가 보낸 정보 외에도 Amazon Lex에는 다음 데이터가 포함되어 있습니다.

- `currentIntent` – 현재 의도 정보를 제공합니다.
- `invocationSource` - Lambda 함수 간접 호출의 목적을 나타냅니다. 이 경우 목적은 사용자 데이터 초기화 및 검증을 수행하는 것입니다. (Amazon Lex는 사용자가 아직 의도를 이행하기 위한 모든 슬롯 데이터를 제공하지 않았음을 알고 있습니다.)
- `messageVersion` - Amazon Lex는 현재 1.0 버전만 지원합니다.

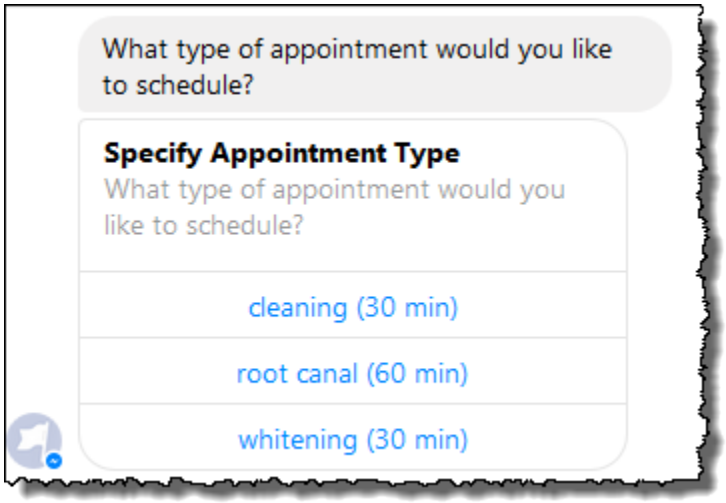
c. 이때 모든 슬롯 값은 null(검증할 값이 없음)입니다. Lambda 함수는 Amazon Lex에 다음 응답을 반환하여 `AppointmentType` 슬롯에 대한 정보를 유도하도록 서비스에 지시합니다. 응답 형식에 대한 자세한 내용은 응답 형식을 참조하십시오.

```
{
    "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"
        },
```

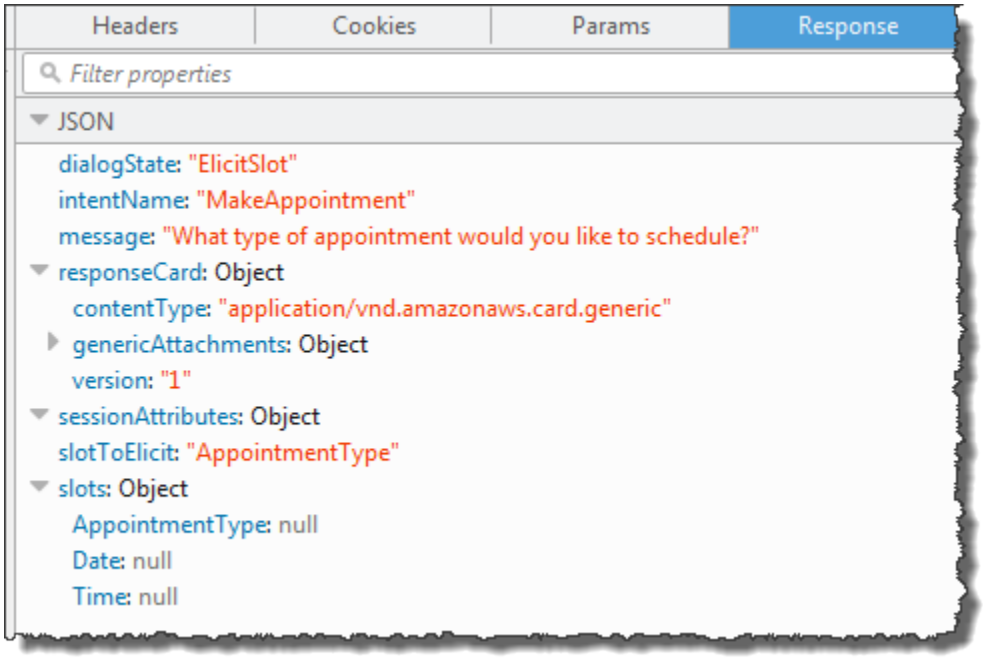
```
        "slots": {
            "AppointmentType": null,
            "Date": null,
            "Time": null
        },
        "type": "ElicitSlot",
        "message": {
            "content": "What type of appointment would you like to schedule?",
            "contentType": "PlainText"
        }
    },
    "sessionAttributes": {}
}
```

응답에는 `dialogAction` 및 `sessionAttributes` 필드가 포함되어 있습니다. 이 중 `dialogAction` 필드는 다음 필드를 반환합니다.

- `type` – Lambda 함수는 이 필드를 `ElicitSlot`으로 설정하여 `slotToElicit` 필드에 지정된 슬롯의 값을 유도하도록 Amazon Lex에 지시합니다. 또한 Lambda 함수는 사용자에게 전달할 `message`를 제공합니다.
- `responseCard` – `AppointmentType` 슬롯의 가능한 값 목록을 식별합니다. 응답 카드를 지원하는 클라이언트(예: Facebook Messenger)는 사용자가 예약 유형을 선택할 수 있도록 다음 이미지와 같은 응답 카드를 표시합니다.



d. Lambda 함수 응답 내의 `dialogAction.type`에서 지정된 바에 따라, Amazon Lex는 클라이언트에 다음 응답을 다시 보냅니다.



클라이언트는 응답을 읽은 다음 "어떤 유형의 예약을 진행하고 싶으세요?"라는 메시지를 표시한 뒤에. 응답 카드(클라이언트가 응답 카드를 지원하는 경우)를 표시합니다.

2. 사용자: 클라이언트에 따라 사용자에게는 다음과 같은 두 가지 옵션이 있습니다.

   - 응답 카드가 표시되는 경우 신경치료(60 분)을 선택하거나 **`root canal`**을 입력합니다.
   - 클라이언트가 응답 카드를 지원하지 않는 경우 **`root canal`**을 입력합니다.

   a. 클라이언트는 다음 `PostText` 요청을 Amazon Lex에 전송합니다(가독성을 위해 줄 바꿈이 추가됨).

   ```
   POST /bot/BookTrip/alias/$LATEST/user/bijt6rovckwecnzesbthrr1d7lv3ja3n/text
   "Content-Type":"application/json"
   "Content-Encoding":"amz-1.0"

   {
      "inputText": "root canal",
       "sessionAttributes": {}
   }
   ```

   b. Amazon Lex는 다음 이벤트를 파라미터로 전송하여 사용자 데이터 검증을 위한 Lambda 함수를 간접 호출합니다.

   ```
   {
   ```

```
    "currentIntent": {
        "slots": {
            "AppointmentType": "root canal",
            "Date": null,
            "Time": null
        },
        "name": "MakeAppointment",
        "confirmationStatus": "None"
    },
    "bot": {
        "alias": null,
        "version": "$LATEST",
        "name": "ScheduleAppointment"
    },
    "userId": "bijt6rovckwecnzesbthrr1d7lv3ja3n",
    "invocationSource": "DialogCodeHook",
    "outputDialogMode": "Text",
    "messageVersion": "1.0",
    "sessionAttributes": {}
}
```

이벤트 데이터에서 다음 사항에 유의하십시오.

- `invocationSource`는 `DialogCodeHook`로 계속됩니다. 이 단계에서는 사용자 데이터만 검증합니다.
- Amazon Lex는 `currentIntent.slots` 슬롯의 `AppointmentType` 필드를 `root canal`로 설정합니다.
- Amazon Lex는 단순히 클라이언트와 Lambda 함수 사이에 `sessionAttributes` 필드를 전달합니다.

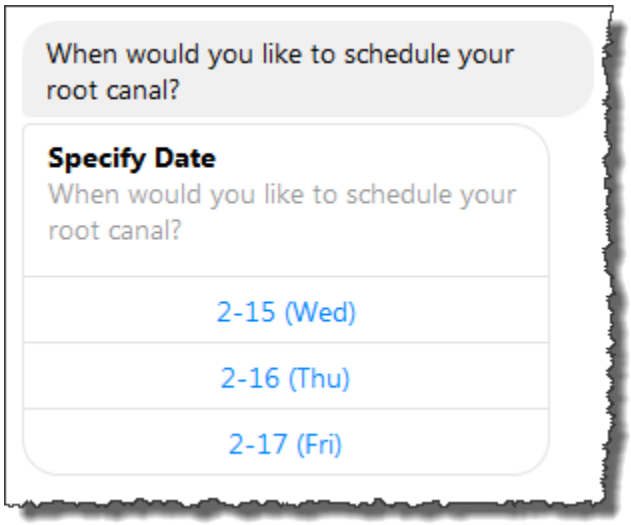
c. Lambda 함수는 사용자 입력을 검증하고 Amazon Lex에 다음 응답을 반환하여 약속 날짜의 값을 유도하도록 서비스에 지시합니다.

```
{
    "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": {}
}
```

이번에도 응답에는 `dialogAction` 및 `sessionAttributes`필드가 포함되어 있습니다. 이 중 `dialogAction` 필드는 다음 필드를 반환합니다.

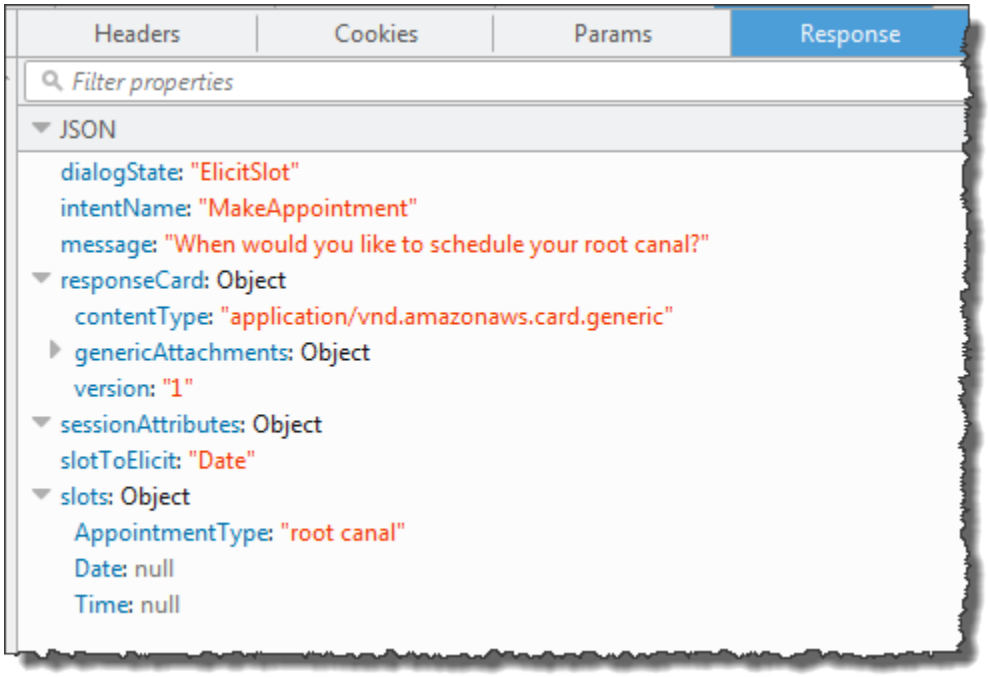
- `type` – Lambda 함수는 이 필드를 `ElicitSlot`으로 설정하여 `slotToElicit` 필드에 지정된 슬롯의 값을 유도하도록 Amazon Lex에 지시합니다. 또한 Lambda 함수는 사용자에게 전달할 `message`를 제공합니다.
- `responseCard` – `Date` 슬롯의 가능한 값 목록을 식별합니다. 응답 카드를 지원하는 클라이언트(예: Facebook Messenger)는 사용자가 약속 날짜를 선택할 수 있도록 다음과 같은 응답 카드를 표시합니다.



Lambda 함수는 날짜 다섯 개를 반환했지만 클라이언트(Facebook Messenger)에는 응답 카드의 버튼이 세 개로 제한되어 있습니다. 따라서 스크린샷에서 첫 세 개의 값만 볼 수 있습니다.

이러한 날짜는 Lambda 함수 에 하드 코딩 됩니다. 프로덕션 애플리케이션에서는 달력을 사용하여 실시간으로 예약 가능한 날짜를 가져올 수 있습니다. 날짜가 동적이므로 Lambda 함수 에서 응답 카드를 동적으로 생성해야 합니다.

d. Amazon Lex는 `dialogAction.type`을 공지하고 Lambda 함수 응답에서 얻은 정보가 포함된 응답을 클라이언트에게 반환합니다.



클라이언트는 언제 신경치료를 예약하시겠어요? 메시지와 응답 카드(클라이언트가 응답 카드를 지원하는 경우)를 표시합니다.

3. 사용자: 유형 **Thursday**.

   a. 클라이언트는 다음 `PostText` 요청을 Amazon Lex에 전송합니다(가독성을 위해 줄 바꿈이 추가됨).

```
POST /bot/BookTrip/alias/$LATEST/user/bijt6rovckwecnzesbthrr1d7lv3ja3n/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

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

   b. Amazon Lex는 다음 이벤트를 파라미터로 전송하여 사용자 데이터 검증을 위한 Lambda 함수를 간접 호출합니다.

```
{
    "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": {}
}
```

이벤트 데이터에서 다음 사항에 유의하십시오.

- `invocationSource`는 `DialogCodeHook`로 계속됩니다. 이 단계에서는 사용자 데이터를 검증만 합니다.
- Amazon Lex는 `currentIntent.slots` 슬롯의 `Date` 필드를 `2017-02-16`로 설정합니다.
- Amazon Lex는 단순히 클라이언트와 Lambda 함수 사이에 `sessionAttributes`를 전달합니다.

c. Lambda 함수는 사용자 입력을 검증합니다. 이때 Lambda 함수는 지정된 날짜에 예약 가능한 시간이 없음을 확인합니다. Amazon Lex에 다음 응답을 반환하여 예약 날짜의 값을 다시 유도하도록 이 서비스에 지시합니다.

```
{
    "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\": []}"
   }
}
```

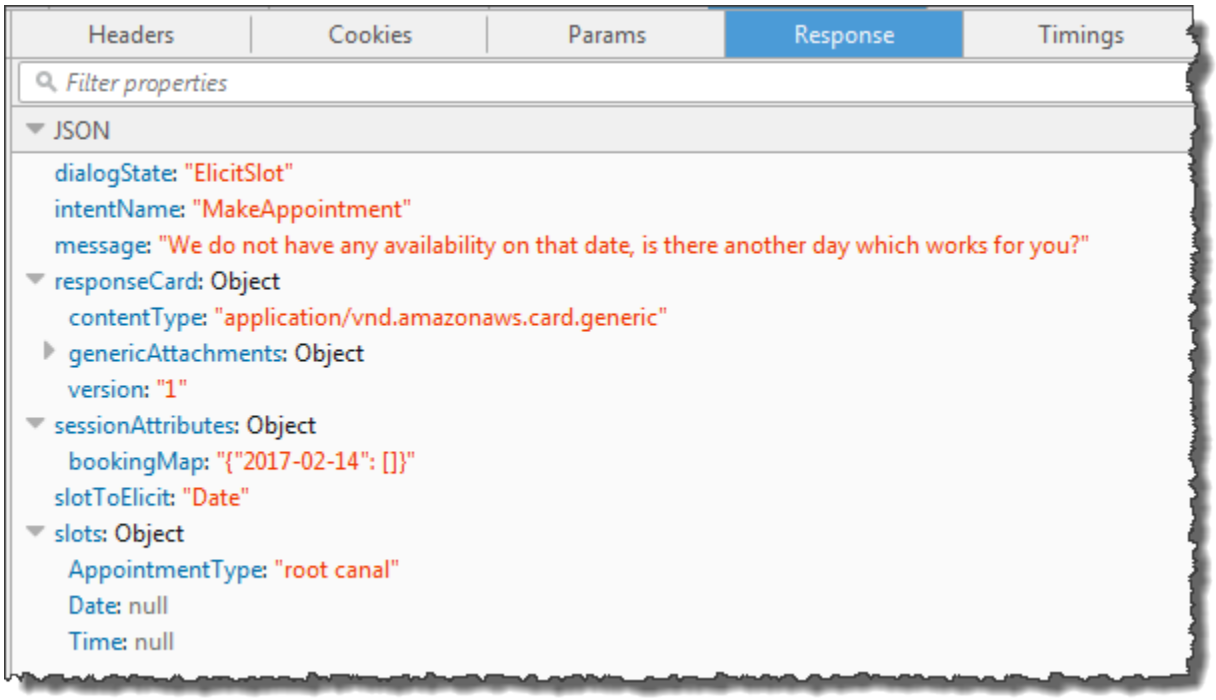
이번에도 응답에는 `dialogAction` 및 `sessionAttributes` 필드가 포함되어 있습니다. 이 중 `dialogAction`은 다음 필드를 반환합니다.

- `dialogAction` 필드

- `type` – Lambda 함수는 이 값을 `ElicitSlot`으로 설정하고, `slotToElicit` 필드를 `Date`로 재설정합니다. 또한 Lambda 함수는 사용자에게 전달할 적절한 `message`를 제공합니다.
- `responseCard` – `Date` 슬롯의 값 목록을 반환합니다.

- `sessionAttributes` - 이때 Lambda 함수 에는 `bookingMap` 세션 속성이 포함되어 있습니다. 이 값은 요청된 약속 날짜와 예약 가능한 약속입니다(빈 객체는 예약이 불가함을 나타냄).

d. Amazon Lex는 `dialogAction.type`을 공지하고 Lambda 함수 응답에서 얻은 정보가 포함된 응답을 클라이언트에게 반환합니다.



클라이언트는 해당 날짜에는 예약할 수 없으니, 다른 가능한 날이 있을까요? 메시지와 응답 카드(클라이언트가 응답 카드를 지원하는 경우)를 표시합니다.

4. 사용자: 클라이언트에 따라 사용자에게는 다음과 같은 두 가지 옵션이 있습니다.

- 응답 카드가 표시되는 경우 2-15(수) 또는 유형 **Wednesday**를 선택합니다.
- 클라이언트가 응답 카드를 지원하지 않는 경우 **Wednesday**을 입력합니다.

a. 클라이언트는 Amazon Lex에 다음 `PostText` 요청을 보냅니다.

```
POST /bot/BookTrip/alias/$LATEST/user/bijt6rovckwecnzesbthrr1d7lv3ja3n/text
"Content-Type":"application/json"
```

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

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

> **Note**
>
> Facebook Messenger 클라이언트는 세션 속성을 설정하지 않습니다. 요청 간의 세션 상태를 유지하려면 Lambda 함수 에서 그렇게 해야 합니다. 실제 어플리케이션에서는 백엔드 데이터베이스에서 이러한 세션 속성을 유지해야 할 수 있습니다.

b. Amazon Lex는 다음 이벤트를 파라미터로 전송하여 사용자 데이터 검증을 위한 Lambda 함수를 간접 호출합니다.

```
{
    "currentIntent": {
        "slots": {
            "AppointmentType": "root canal",
            "Date": "2017-02-15",
            "Time": null
        },
        "name": "MakeAppointment",
        "confirmationStatus": "None"
    },
    "bot": {
        "alias": null,
        "version": "$LATEST",
        "name": "ScheduleAppointment"
    },
    "userId": "u3fpr9gghj02zts7y5tpq5mm4din2xqy",
    "invocationSource": "DialogCodeHook",
    "outputDialogMode": "Text",
    "messageVersion": "1.0",
    "sessionAttributes": {
    }
}
```

Amazon Lex는 `Date`슬롯을 `2017-02-15`로 설정하여 `currentIntent.slots`를 업데이트했습니다.

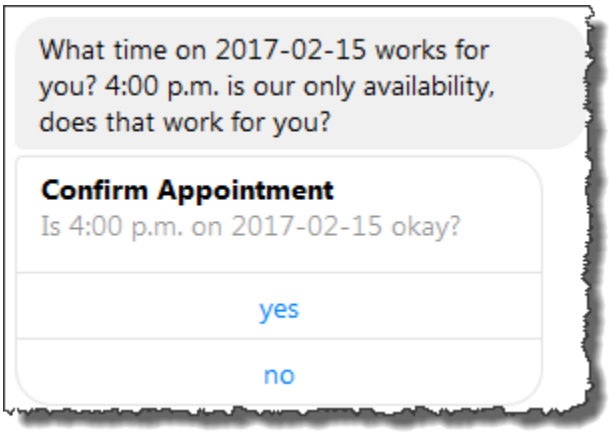
c. Lambda 함수는 사용자 입력을 검증하고 Amazon Lex에 다음 응답을 반환하여 약속 시간의 값을 유도하도록 이 서비스에 지시합니다.

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

```
    }
}
```

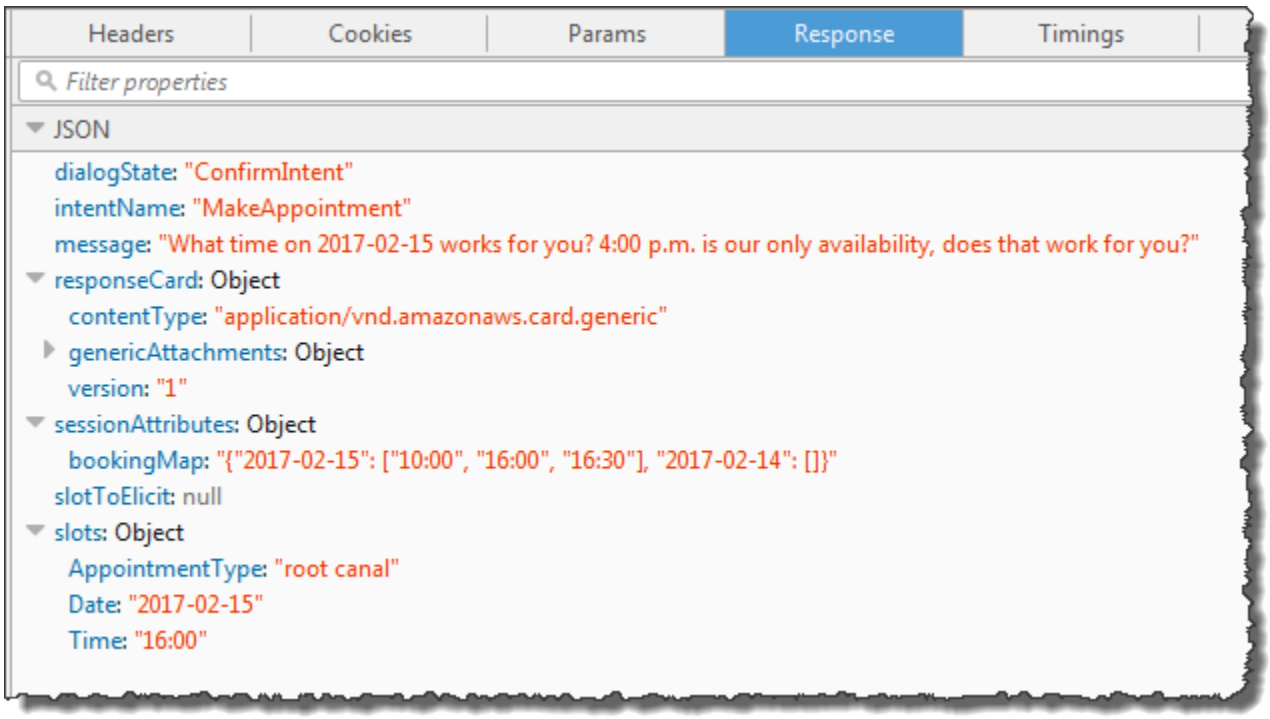
이번에도 응답에는 `dialogAction` 및 `sessionAttributes` 필드가 포함되어 있습니다. 이 중 `dialogAction`은 다음 필드를 반환합니다.

- `dialogAction` 필드:
  - `type` – Lambda 함수는 이 값을 `ConfirmIntent`로 설정하여 `message`에 제시된 약속 시간에 대한 사용자 확인을 받도록 Amazon Lex에 지시합니다.
  - `responseCard` – 사용자가 선택할 수 있는 예 혹은 아니오 값 목록을 반환합니다. 클라이언트가 응답 카드를 지원하는 경우, 다음 예와 같이 응답 카드를 표시합니다.



  - `sessionAttributes` - Lambda 함수는 해당 예약 날짜와 그 예약 날짜에 가능한 예약 시간으로 값을 지정하여 `bookingMap` 세션 속성을 설정합니다. 이 예에서는 30분 진료 예약입니다. 진료 시간이 1시간 걸리는 신경 치료의 경우에는, 4:00 p.m만 예약할 수 있습니다.

d. Lambda 함수 의 응답 내의 `dialogAction.type`에 지정된 설정에 따라 Amazon Lex는 클라이언트에 다음 응답을 반환합니다.



클라이언트는 다음과 같은 메시지를 표시합니다. 2017-02-15 중 언제가 괜찮나요? 이용 가능 시간은 4:00 p.m 뿐입니다. 가능한가요?

5. 사용자: **yes**를 선택합니다.

Amazon Lex는 다음 이벤트 데이터로 Lambda 함수를 간접 호출합니다. 사용자가 **yes**를 회신했기 때문에 Amazon Lex는 `confirmationStatus`를 `Confirmed`으로 설정하고, `currentIntent.slots` 필드 내의 `Time`를 `4 p.m`으로 설정합니다.

```
{
    "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": {
   }
}
```

confirmationStatus가 확인되었으므로 Lambda 함수는 의도(치과 진료 예약)를 처리하고 Amazon Lex에 다음 응답을 반환합니다.

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

참고사항

- Lambda 함수는 sessionAttributes를 업데이트했습니다.
- dialogAction.type은 Close로 설정됩니다. 이는 사용자 응답을 기대하지 않도록 Amazon Lex에 지시합니다.
- dialogAction.fulfillmentState는 Fulfilled로 설정됩니다. 이는 의도가 이행되었음을 나타냅니다.

클라이언트는 네, 예약을 처리했습니다. 메시지를 표시합니다. 2017-02-15 4:00 p.m에 뵙겠습니다. 메시지를 표시합니다.

여행 예약

이 예제에서는 여러 의도를 지원하도록 구성된 봇을 생성하는 것을 보여 줍니다. 또한 이 예제에서는 의도 전반에 걸쳐 정보를 공유하기 위해 세션 속성을 사용하는 방법을 보여 줍니다. 봇을 생성한 후 콘솔에서 테스트 클라이언트를 사용하여 봇(BookTrip)을 테스트합니다. 클라이언트는 PostText 런타임 API 작업을 사용하여 각 사용자 입력에 대한 요청을 Amazon Lex에 전송합니다.

이 예제에서 BookTrip 봇은 두 가지 의도(BookHotel 및 BookCar)로 구성되어 있습니다. 예를 들어, 사용자가 먼저 호텔을 예약한다고 가정하겠습니다. 상호 작용을 하는 동안 사용자는 체크인 날짜, 위치, 숙박 일수와 같은 정보를 제공합니다. 의도가 이행된 후 클라이언트는 세션 속성을 사용하여 이 정보를 유지할 수 있습니다. 세션 속성에 대한 자세한 내용은 PostText을 참조하십시오.

이제 사용자가 계속해서 차량을 예약한다고 가정하겠습니다. 사용자가 BookCar 의도를 초기화 및 검증하도록 구성한 코드 후크(Lambda 함수)는 사용자가 이전의 BookHotel 의도(도착 도시, 체크인 및 체크아웃 날짜)에서 제공한 정보를 사용하여 BookCar 의도에 대한 슬롯 데이터(도착지, 수령 도시, 수령 날짜 및 반환 날짜)를 초기화합니다. 이는 의도 전반에 걸친 정보 공유를 통해 사용자와의 동적 대화에 참여할 수 있는 봇을 구축하는 방법을 보여 줍니다.

이 예제에서는 다음 세션 속성을 사용합니다. 클라이언트와 Lambda 함수만 세션 속성을 설정하고 업데이트할 수 있습니다. Amazon Lex는 오직 클라이언트와 Lambda 함수 사이에 이를 전달합니다. Amazon Lex는 어떠한 세션 속성도 유지하거나 수정하지 않습니다.

- `currentReservation` – 진행 중인 예약 및 기타 관련 정보에 대한 슬롯 데이터를 포함합니다. 예를 들어, 다음은 클라이언트가 Amazon Lex에 보내는 샘플 요청으로, 요청 본문에 `currentReservation` 세션 속성을 보여 줍니다.

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText":"Chicago",
   "sessionAttributes":{
       "currentReservation":"{\"ReservationType\":\"Hotel\",
                              \"Location\":\"Moscow\",
                              \"RoomType\":null,
                              \"CheckInDate\":null,
                              \"Nights\":null}"
```

```
    }
}
```

- `lastConfirmedReservation` – 이전 의도와 유사한 정보를 포함합니다(있는 경우). 예를 들어, 사용자가 호텔을 예약한 다음 차량 예약을 진행 중이라면 이 세션 속성은 이전 BookHotel 의도에 대한 슬롯 데이터를 저장합니다.

- `confirmationContext` – 이전 예약의 슬롯 데이터(있는 경우)를 기반으로 일부 슬롯 데이터를 미리 채우는 경우, Lambda 함수는 이를 `AutoPopulate`로 설정합니다. 이를 통해 의도 전반에 걸쳐 정보를 공유할 수 있습니다. 예를 들어, 이전에 호텔을 예약한 사용자가 차량을 예약하려는 경우 Amazon Lex는 사용자에게 호텔 예약과 동일한 도시 및 날짜로 차량이 예약됨을 확인(또는 거부)하라는 메시지를 표시할 수 있습니다.

이 연습에서는 청사진을 사용하여 Amazon Lex 봇 및 Lambda 함수를 생성합니다. 청사진에 대한 자세한 내용은 Amazon Lex와 AWS Lambda 청사진을 참조하십시오.

다음 단계

1단계: 이 연습에서 사용되는 청사진 검토

1단계: 이 연습에서 사용되는 청사진 검토

주제

- 봇 청사진(BookTrip) 개요
- Lambda 함수 청사진(lex-book-trip-python) 개요

봇 청사진(BookTrip) 개요

봇을 생성할 때 사용하는 청사진(BookTrip)는 다음과 같은 사전 구성을 제공합니다.

- 슬롯 유형 – 사용자 지정 슬롯 유형 두 가지:
  - `RoomTypes`(열거 값: `king`, `queen`, `deluxe`)로, `BookHotel`의도에서 사용

- `CarTypes`(열거 값: `economy`, `standard`, `midsize`, `full size`, `luxury`, `minivan`)로, `BookCar`의도에서 사용

- 의도 1(BookHotel) – 다음과 같이 미리 구성되어 있습니다.
  - 미리 구성된 슬롯
    - 사용자 지정 슬롯 유형 `RoomTypes`의 `RoomType`
    - 내장 슬롯 유형 `AMAZON.US_CITY`의 `Location`
    - 내장 슬롯 유형 `AMAZON.DATE`의 `CheckInDate`
    - 내장 슬롯 유형 `AMAZON.NUMBER`의 `Nights`
  - 미리 구성된 표현
    - "호텔을 예약해줘"
    - "호텔을 예약하고 싶어"
    - "{지역}의 {야간}으로 예약해줘"

    사용자가 이러한 말을 하면 Amazon Lex는 `BookHotel`이 의도라고 판단하고 사용자에게 슬롯 데이터 메시지를 표시합니다.
  - 미리 구성된 프롬프트
    - `Location` 슬롯에 대한 프롬프트 – "어떤 도시에서 묵고 싶으세요?"
    - `CheckInDate` 슬롯에 대한 프롬프트 – "체크인 날짜는 언제인가요?"
    - `Nights` 슬롯에 대한 프롬프트 – "몇 박을 묵을 예정이신가요?"
    - `RoomType` 슬롯에 대한 프롬프트 – "퀸, 킹, 디럭스 중에 어떤 방을 예약할까요?"
    - 확인문 – "네, {CheckInDate}에 {Location}에서 {Nights}에 묵는 것으로 확인되었습니다. 예약을 진행할까요?"
    - 거부 – "네, 예약 진행을 취소하였습니다.”

- 의도 2(BookCar) – 다음과 같이 미리 구성되어 있습니다.
  - 미리 구성된 슬롯
    - 내장 유형 `AMAZON.US_CITY`의 `PickUpCity`
    - 내장 유형 `AMAZON.DATE`의 `PickUpDate`
    - 내장 유형 `AMAZON.DATE`의 `ReturnDate`
    - 내장 유형 `AMAZON.NUMBER`의 `DriverAge`

  - 사용자 지정 유형 `CarTypes`의 `CarType`
- 미리 구성된 표현
  - "차량을 예약해줘"
  - "차량 예약해"
  - "차량 예약을 진행해줘"

  사용자가 이러한 말을 하면 Amazon Lex는 BookCar가 의도라고 판단하고 사용자에게 슬롯 데이터 메시지를 표시합니다.
- 미리 구성된 프롬프트
  - `PickUpCity` 슬롯에 대한 프롬프트 – "어느 도시에서 차를 빌리실 예정인가요?"
  - `PickUpDate` 슬롯에 대한 프롬프트 – "렌탈 시작 날짜는 언제인가요?"
  - `ReturnDate` 슬롯에 대한 프롬프트 – "언제 차를 반납하실 건가요?"
  - `DriverAge` 슬롯에 대한 프롬프트 – "운전자의 나이는 몇살인가요?"
  - `CarType` 슬롯에 대한 프롬프트 – "어떤 종류의 차를 원하세요? 가장 많이 선택하는 옵션은 이코노미, 미드사이즈, 럭셔리입니다."
  - 확인문 – "네, {CarType}의 차량으로 {PickUpCity}에서 차량을 렌탈하며 렌탈 기간은 {PickUpDate}부터 {ReturnDate}까지 입니다. 예약을 진행할까요?"
  - 거부 – "네, 예약 진행을 취소하였습니다.”

Lambda 함수 청사진(lex-book-trip-python) 개요

봇 블루프린트 외에도는 봇 블루프린트와 함께 코드 후크로 사용할 수 있는 블루프린트(lex-book-trip-python)를 AWS Lambda 제공합니다. 봇 청사진 및 해당 Lambda 함수 청사진 목록은 Amazon Lex와 AWS Lambda 청사진를 참조하십시오.

BookTrip 청사진을 사용하여 봇을 생성할 때 사용자 데이터 입력을 초기화/검증하고 의도를 이행하기 위해, 이 함수를 코드 후크로 추가하여 두 가지 의도(BookCar 및 BookHotel)의 구성을 업데이트합니다.

제공되는 이 Lambda 함수 코드는 이전에 알려진 사용자 정보(세션 속성에 유지됨)를 사용하여 의도에 대한 슬롯 값을 초기화하는 동적 대화를 보여 줍니다. 자세한 내용은 대화 컨텍스트 관리을 참조하세요.

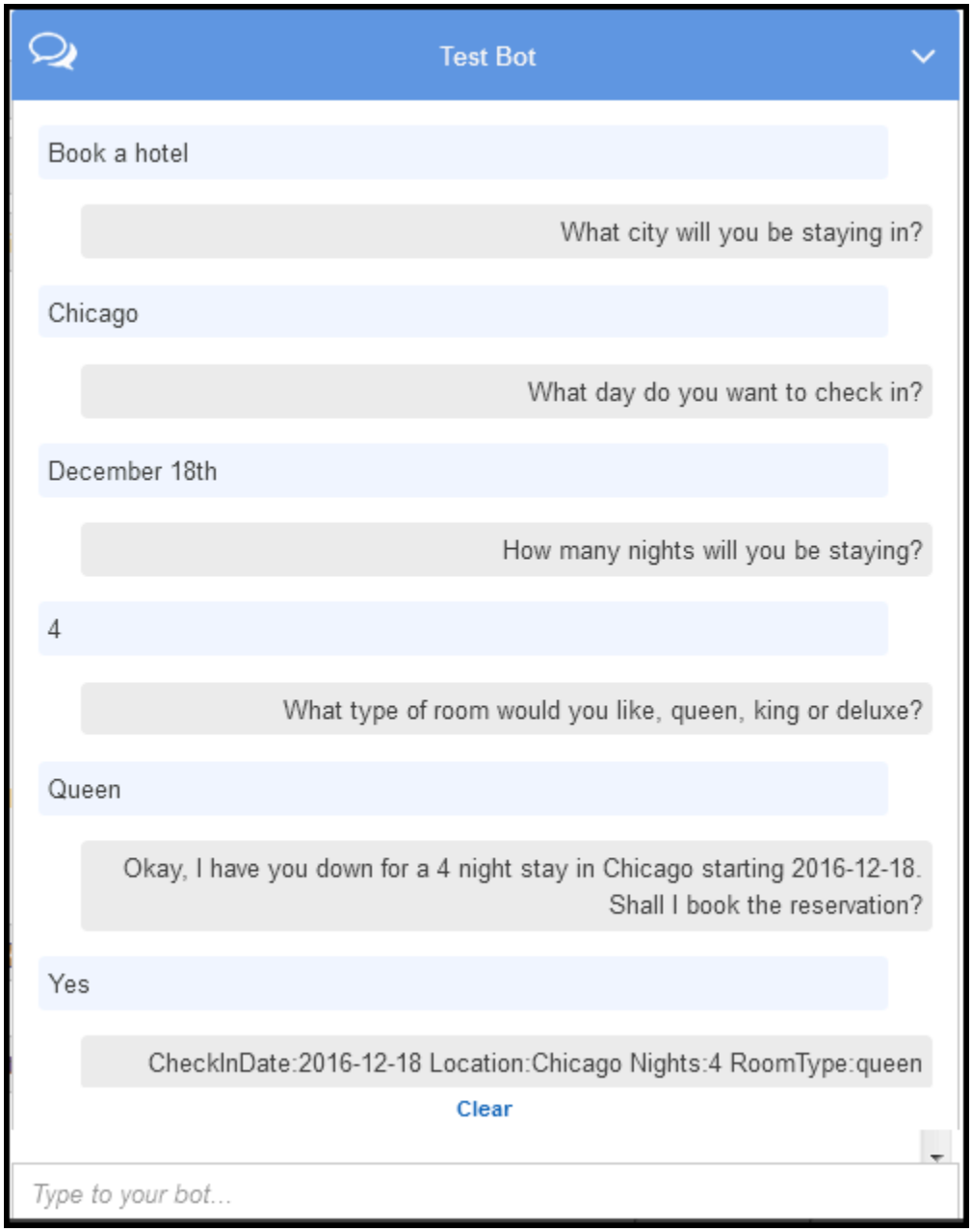
다음 단계

2단계: Amazon Lex 봇 생성

2단계: Amazon Lex 봇 생성

이 섹션에서는 봇(BookTrip)을 생성합니다.

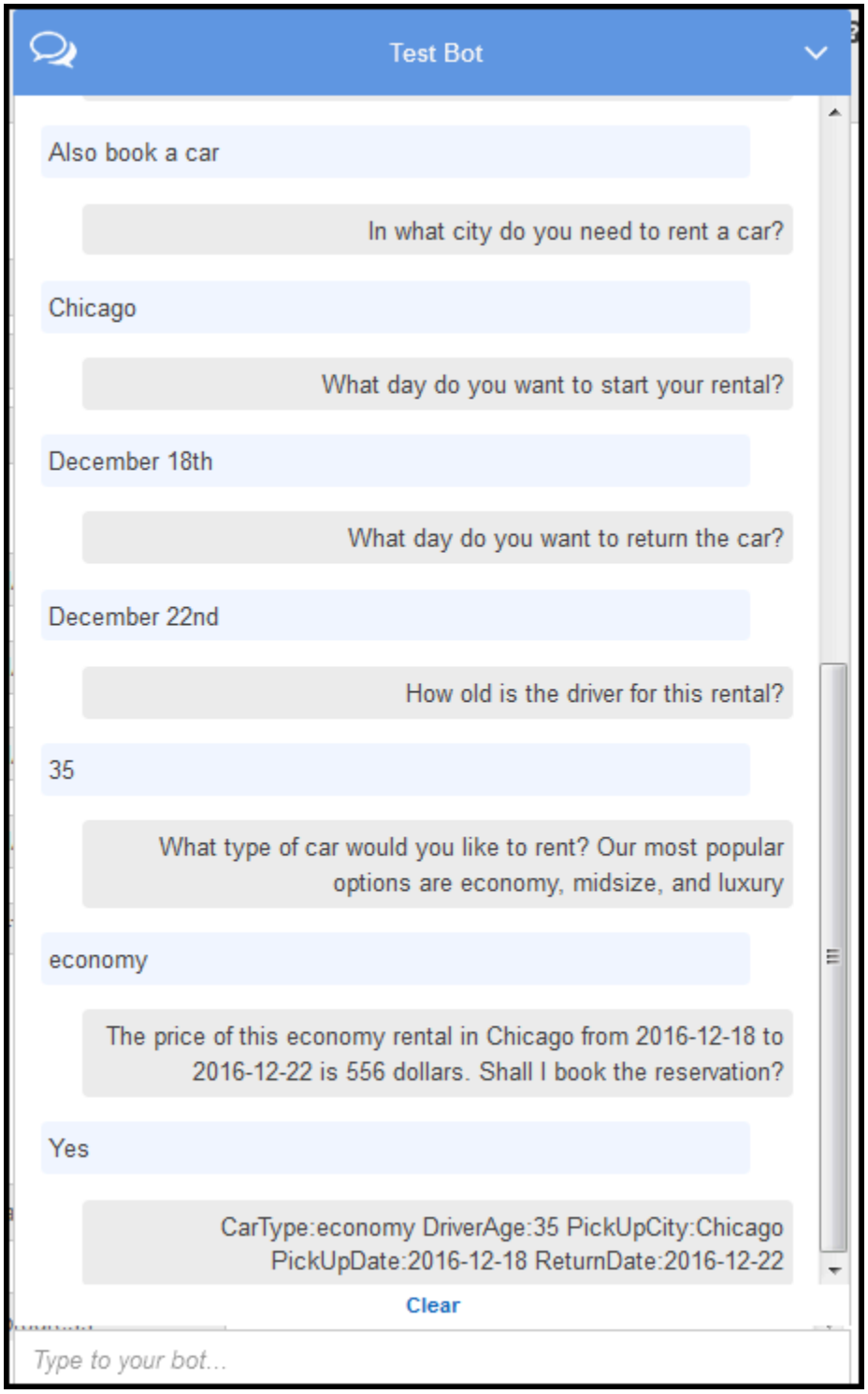
1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lex/ Amazon Lex 콘솔을 엽니다.
2. 봇 페이지에서 생성을 선택합니다.
3. Lex 봇 생성 페이지에서
   - BookTrip 청사진을 선택합니다.
   - 봇 이름의 기본값(BookTrip)을 그대로 둡니다.
4. 생성(Create)을 선택합니다. 그러면 콘솔이 봇을 생성하도록 Amazon Lex에 일련의 요청을 보냅니다. 다음 사항에 유의하세요.
5. 콘솔에 BookTrip 봇이 표시됩니다. 편집기 탭에서 미리 구성된 의도(BookCar 및 BookHotel)의 세부 정보를 검토합니다.
6. 테스트 창에서 봇을 테스트합니다. 다음을 사용하여 사용자의 봇을 상대로 테스트 대화에 참여합니다.



Amazon Lex는 초기 사용자 입력("호텔 예약해줘")에서 의도(BookHotel)를 유추합니다. 그런 다음 이 의도에서 미리 구성된 프롬프트를 사용하여 사용자로부터 슬롯 데이터를 유도합니다. 사용자가 모든 슬롯 데이터를 제공하면 Amazon Lex는 모든 사용자 입력이 메시지로 포함된 응답을 클라이언트에 다시 반환합니다. 클라이언트는 응답에 다음과 같은 메시지를 표시합니다.

```
CheckInDate:2016-12-18 Location:Chicago Nights:5 RoomType:queen
```

이제 대화를 계속하여 차량을 예약합니다.



참고.

- 현재 사용자 데이터 검증이 없습니다. 예를 들어, 호텔을 예약할 도시를 제공할 수 있습니다.

- 차량을 예약하기 위해 몇 가지 동일한 정보(도착지, 수령 도시, 수령 날짜, 반환 날짜)를 다시 제공합니다. 동적 대화에서 사용자의 봇은 호텔 예약을 위해 사용자가 이전에 제공한 입력을 기반으로 이러한 정보를 초기화합니다.

다음 섹션에서는 Lambda 함수를 만들어 세션 속성을 통해 의도 전반에 걸쳐 정보를 공유함으로써 사용자 데이터 검증 및 초기화의 일부를 수행합니다. 그런 다음 Lambda 함수를 코드 후크로 추가하여 의도 구성을 업데이트함으로써 사용자 입력을 초기화/검증하고 의도를 이행합니다.

다음 단계

3단계: Lambda 함수 생성

3단계: Lambda 함수 생성

이 섹션에서는 AWS Lambda 콘솔에 제공된 블루프린트(lex-book-trip-python)를 사용하여 Lambda 함수를 생성합니다. 또한 콘솔에서 제공하는 샘플 이벤트 데이터를 사용하여 Lambda 함수를 간접 호출함으로써 이를 테스트합니다.

이 Lambda 함수는 Python으로 작성됩니다.

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lambda/ AWS Lambda 콘솔을 엽니다.
2. 함수 생성(Create function)을 선택합니다.
3. 청사진 사용을 선택합니다. **`lex`**를 입력하여 청사진을 검색한 후 `lex-book-trip-python` 청사진을 선택합니다.
4. Lambda 함수의 구성을 다음과 같이 선택합니다.
   - Lambda 함수 이름(`BookTripCodeHook`)을 입력합니다.
   - 역할의 경우 템플릿에서 새로운 역할 생성을 선택한 다음 역할 이름을 입력합니다.
   - 나머지는 기본값을 그대로 사용합니다.
5. Create function(함수 생성)을 선택합니다.
6. 영어(미국)(en-US) 이외의 지역을 사용하는 경우 특정 로캘에 대한 청사진 업데이트에 설명된 대로 의도 이름을 업데이트하십시오.
7. Lambda 함수 테스트. 차량 예약과 호텔 예약 모두에 샘플 데이터를 사용하여 Lambda 함수를 두 번 호출합니다.

a. 테스트 이벤트 선택 드랍 다운에서 테스트 이벤트 구성을 선택합니다.
b. 샘플 이벤트 템플릿 목록에서 Amazon Lex 호텔 예약을 선택합니다.

   이 샘플 이벤트는 요청 및 응답 모델과 일치합니다. 자세한 내용은 Lambda 함수 사용을 참조하세요.
c. 저장 및 테스트를 선택합니다.
d. Lambda 함수가 성공적으로 실행되었는지 확인합니다. 이 경우 응답은 Amazon Lex 응답 모델과 일치합니다.
e. 이 단계를 반복합니다. 샘플 이벤트 템플릿 목록에서 Amazon Lex 차량 예약을 선택합니다. Lambda 함수가 차량 예약을 처리합니다.

다음 단계

4단계: Lambda 함수를 코드 후크로 추가

4단계: Lambda 함수를 코드 후크로 추가

이 섹션에서는 초기화/검증 및 이행 활동을 위해 Lambda 함수를 코드 후크로 추가하여 BookCar 및 BookHotel 의도의 구성을 업데이트합니다. Amazon Lex 리소스의 $LATEST 버전만 업데이트할 수 있으므로 의도의 $LATEST 버전을 선택해야 합니다.

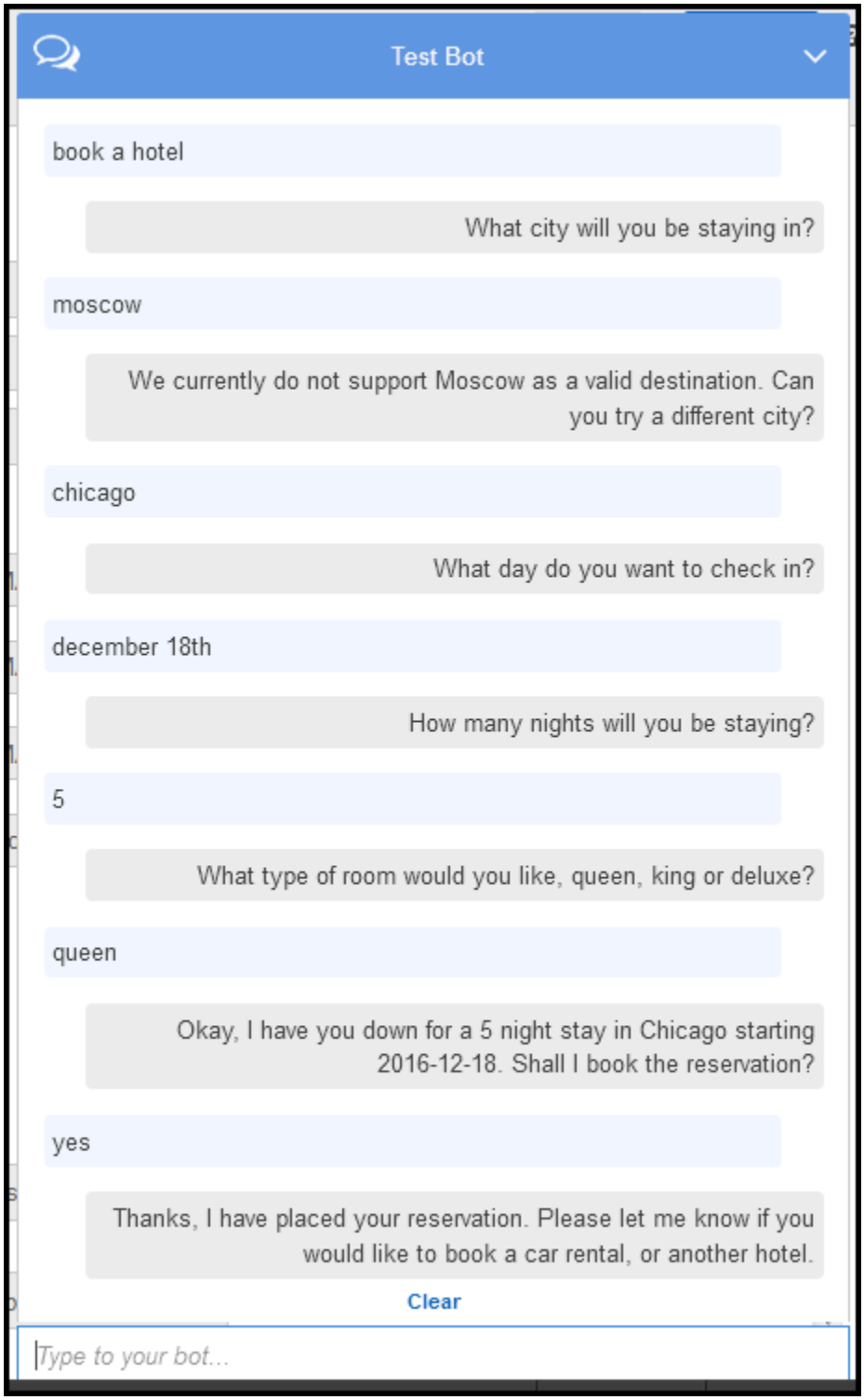
1. Amazon Lex 콘솔에서 BookTrip 봇을 선택합니다.
2. 편집기 탭에서 BookHotel 의도를 선택합니다. 다음과 같이 의도 구성을 업데이트합니다.
   a. 의도 버전(의도 이름 옆)이 $LATEST인지 확인합니다.
   b. 다음과 같이 Lambda 함수를 초기화 및 검증 코드 후크로 추가합니다.
      - 옵션에서 초기화 및 유효성 검사 코드 후크를 선택합니다.
      - 목록에서 Lambda 함수를 선택합니다.
   c. 다음과 같이 Lambda 함수를 이행 코드 후크로 추가합니다.
      - 이행에서 AWS Lambda 함수를 선택합니다.
      - 목록에서 Lambda 함수를 선택합니다.
      - Goodbye message를 선택하고 메시지를 입력합니다.

d. 저장을 선택합니다.

3. 편집기 탭에서 BookCar 의도를 선택합니다. 이전 단계에 따라 Lambda 함수를 검증 및 이행 코드 후크로 추가합니다.

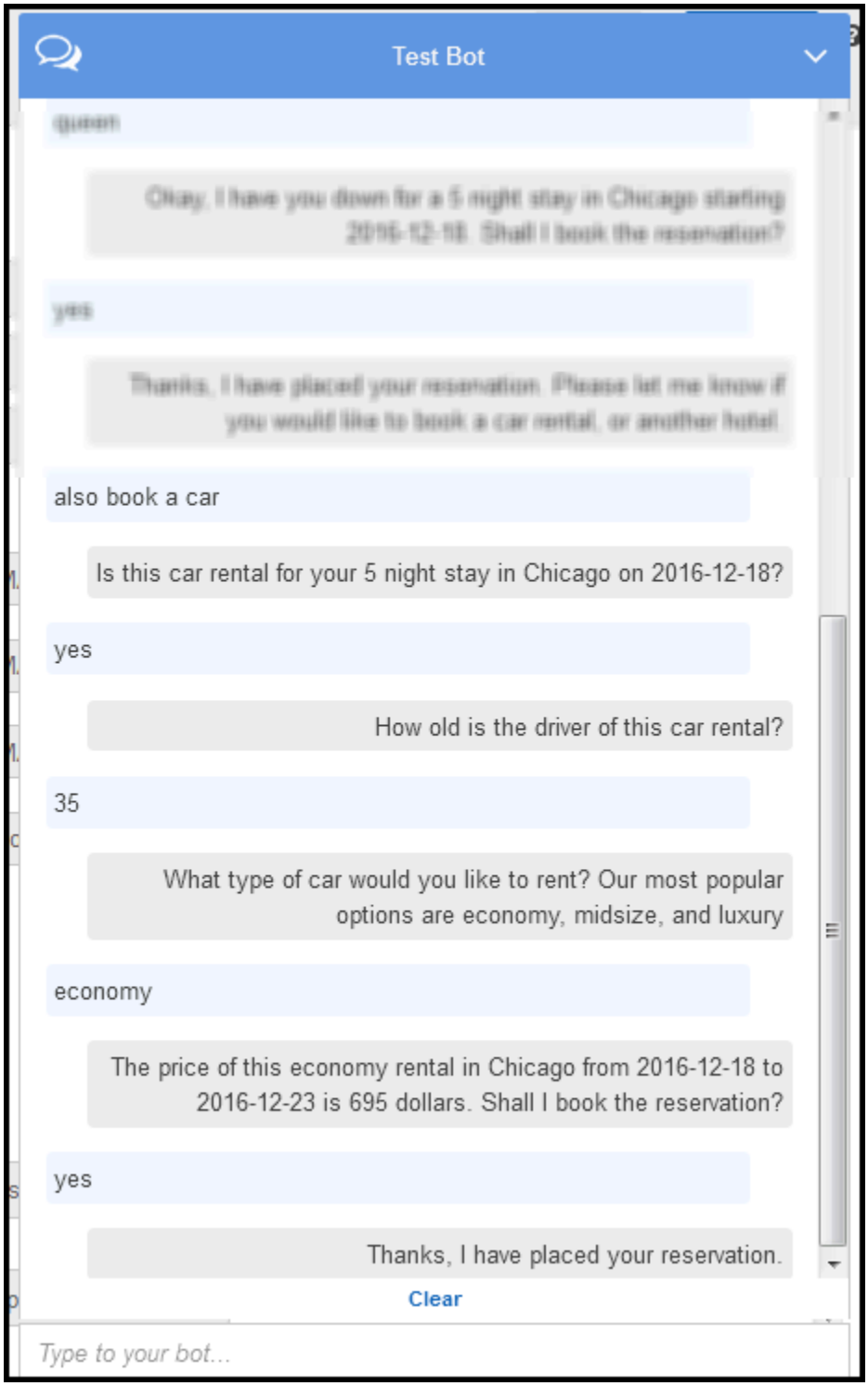
4. 구축을 선택합니다. 그러면 콘솔이 구성을 저장하도록 Amazon Lex에 일련의 요청을 보냅니다.

5. 봇 테스트. 이제 초기화, 사용자 데이터 검증 및 이행을 수행하는 Lambda 함수가 있으므로 사용자 상호 작용의 차이를 볼 수 있습니다.



클라이언트(콘솔)에서 Amazon Lex로, Amazon Lex에서 Lambda 함수로의 데이터 흐름에 대한 자세한 내용은 데이터 흐름: 호텔 예약 의도을 참조하십시오.

6. 다음과 같이 대화를 계속하여 차량을 예약합니다.



사용자가 차량을 예약할 때 클라이언트(콘솔)는 세션 속성(이전 대화인 BookHotel에서 가져온)이 포함된 요청을 Amazon Lex에 보냅니다. Amazon Lex가 Lambda 함수에 이 정보를 전달하면, 일부 BookCar 슬롯 데이터(PickUpDate, ReturnDate, PickUpCity)가 초기화됩니다(즉, 미리 채워짐).

> **Note**
>
> 이는 세션 속성을 사용하여 의도 전반에 걸쳐 컨텍스트를 유지하는 방법을 보여 줍니다. 콘솔 클라이언트는 테스트 창에 사용자가 이전의 세션 속성을 지우기 위해 사용할 수 있는 지우기 링크를 제공합니다.

클라이언트(콘솔)에서 Amazon Lex로, Amazon Lex에서 Lambda 함수로의 데이터 흐름에 대한 자세한 내용은 데이터 흐름: 차량 예약 의도을 참조하십시오.

정보 흐름의 세부 정보

이 연습에서는 Amazon Lex 콘솔에서 제공되는 테스트 창 클라이언트를 사용하여 Amazon Lex BookTrip 봇과의 대화에 참여했습니다. 이 섹션에서는 다음에 대해 설명합니다.

- 클라이언트와 Amazon Lex 간의 데이터 흐름.

  이 섹션에서는 클라이언트가 `PostText` 런타임 API를 사용하여 Amazon Lex에 요청을 보내고 그에 따라 요청 및 응답의 세부 정보를 보여 준다고 가정합니다. `PostText` 런타임 API에 대한 자세한 내용은 PostText을 참조하십시오.

  > **Note**
  >
  > 클라이언트가 `PostContent` API를 사용할 때의 클라이언트와 Amazon Lex 간의 정보 흐름의 예는 2a단계(선택 사항): 음성 정보 흐름의 세부 정보 검토(콘솔) 을 참조하십시오.

- Amazon Lex와 Lambda 함수 간의 데이터 흐름. 자세한 내용은 Lambda 함수 입력 이벤트 및 응답 형식을 참조하세요.

주제

- 데이터 흐름: 호텔 예약 의도
- 데이터 흐름: 차량 예약 의도

데이터 흐름: 호텔 예약 의도

이 섹션에서는 각 사용자가 입력한 후에 일어나는 일을 설명합니다.

1. 사용자: "호텔 예약해줘"

   a. 클라이언트(콘솔)는 Amazon Lex에 다음 PostText 요청을 보냅니다.

   ```
   POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
   "Content-Type":"application/json"
   "Content-Encoding":"amz-1.0"

   {
      "inputText":"book a hotel",
      "sessionAttributes":{}
   }
   ```

   요청 URI 및 본문 모두 Amazon Lex에 다음 정보를 제공합니다.

   - 요청 URI – 봇 이름(BookTrip), 봇 별칭($LATEST) 및 사용자 이름을 제공합니다. 후행 `text`는 이것이 `PostText` API 요청(`PostContent`가 아님)임을 나타냅니다.
   - 요청 본문 – 사용자 입력(`inputText`)과 빈 `sessionAttributes`를 포함합니다. 처음에는 빈 객체이며, Lambda 함수가 먼저 세션 속성을 설정합니다.

   b. `inputText`에서 Amazon Lex는 의도(BookHotel)를 감지합니다. 이 의도는 사용자 데이터 초기화/검증을 위한 코드 후크인 Lambda 함수로 구성됩니다. 따라서 Amazon Lex는 다음 정보를 이벤트 파라미터로 전달하여 Lambda 함수를 호출합니다(입력 이벤트 형식 참조).

   ```
   {
      "messageVersion":"1.0",
      "invocationSource":"DialogCodeHook",
      "userId":"wch89kjqcpkds8seny7dly5x3otq68j3",
      "sessionAttributes":{
      },
      "bot":{
         "name":"BookTrip",
         "alias":null,
         "version":"$LATEST"
      },
      "outputDialogMode":"Text",
   ```

```
   "currentIntent":{
      "name":"BookHotel",
      "slots":{
         "RoomType":null,
         "CheckInDate":null,
         "Nights":null,
         "Location":null
      },
      "confirmationStatus":"None"
   }
}
```

클라이언트가 보낸 정보 외에도 Amazon Lex에는 다음과 같은 데이터가 추가로 포함되어 있습니다.

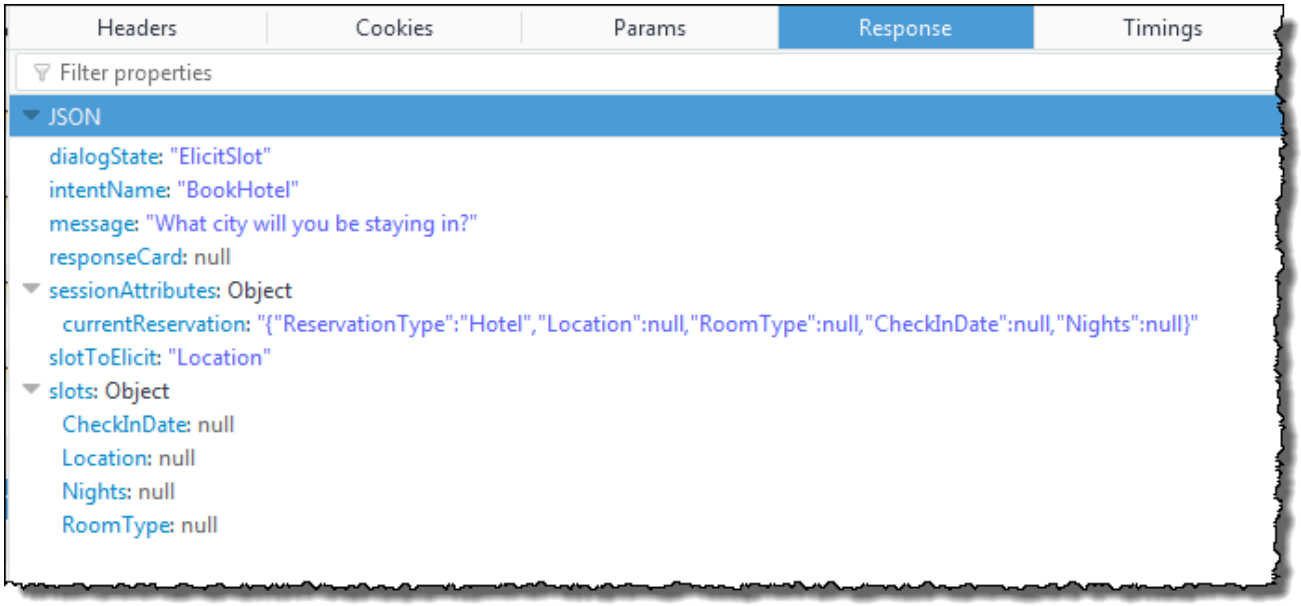
- `messageVersion` - Amazon Lex는 현재 1.0 버전만 지원합니다.
- `invocationSource` - Lambda 함수 호출의 목적을 나타냅니다. 이 경우에 목적은 사용자 데이터 초기화 및 검증을 수행하는 것입니다(이때 Amazon Lex는 사용자가 의도를 이행하기 위한 모든 슬롯 데이터를 제공하지 않았음을 인식함).
- `currentIntent` – 모든 슬롯 값은 null로 설정됩니다.

c. 이때 모든 슬롯 값은 null입니다. Lambda 함수가 검증할 것이 없습니다. Lambda 함수는 Amazon Lex에 다음 응답을 반환합니다. 응답 형식에 대한 자세한 내용은 응답 형식섹션을 참조하십시오.

```
{
   "sessionAttributes":{
      "currentReservation":"{\"ReservationType\":\"Hotel\",\"Location\":null,
\"RoomType\":null,\"CheckInDate\":null,\"Nights\":null}"
   },
   "dialogAction":{
      "type":"Delegate",
      "slots":{
         "RoomType":null,
         "CheckInDate":null,
         "Nights":null,
         "Location":null
      }
   }
}
```

> **Note**
>
> - `currentReservation` – Lambda 함수에는 이 세션 속성이 포함되어 있습니다. 해당 값은 현재 슬롯 정보 및 예약 유형의 사본입니다.
>
> Lambda 함수 및 클라이언트만 이러한 세션 속성을 업데이트할 수 있습니다. Amazon Lex는 단순히 이 값을 전달합니다.
>
> - `dialogAction.type` – Lambda 함수는 이 값을 `Delegate`로 설정하여 일련의 다음 조치에 대한 책임을 Amazon Lex에 위임합니다.
>
> Lambda 함수가 사용자 데이터 검증에서 무언가 감지한 경우 다음에 해야 할 조치를 Amazon Lex에게 지시합니다.

d. Amazon Lex는 `dialogAction.type`에 따라 일련의 다음 조치를 결정하고, `Location` 슬롯에 대한 사용자의 데이터를 유도합니다. 의도 구성에 따라 이 슬롯에 대한 프롬프트 메시지 중 하나("어떤 도시에서 묵고 싶으세요?")를 선택한 다음, 사용자에게 다음과 같은 응답을 보냅니다.



세션 속성이 클라이언트에 전달됩니다.

클라이언트는 응답을 읽은 다음 "어떤 도시에서 묵고 싶으세요?"이라는 메시지를 표시합니다.

2. 사용자: "모스크바"

a. 클라이언트는 Amazon Lex에 다음 `PostText` 요청을 보냅니다(가독성을 위해 줄 바꿈이 추가됨).

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText":"Moscow",
   "sessionAttributes":{
       "currentReservation":"{\"ReservationType\":\"Hotel\",
                              \"Location\":null,
                              \"RoomType\":null,
                              \"CheckInDate\":null,
                              \"Nights\":null}"
   }
}
```

클라이언트에는 `inputText` 외에도 클라이언트가 받은 동일한 `currentReservation` 세션 속성이 포함됩니다.

b. Amazon Lex는 먼저 현재 의도의 맥락에서 `inputText`를 해석합니다(이 서비스는 특정 사용자에게 `Location` 슬롯에 대한 정보를 요청했음을 기억). 현재 의도에 대한 슬롯 값을 업데이트하고, 다음 이벤트를 사용하여 Lambda 함수를 호출합니다.

```
{
    "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`는 `DialogCodeHook`로 계속됩니다. 이 단계에서는 사용자 데이터만 검증합니다.
> - Amazon Lex는 단지 Lambda 함수에 세션 속성을 전달합니다.
> - `currentIntent.slots`의 경우Amazon Lex는 `Location` 슬롯을 `Moscow`로 업데이트했습니다.

c. Lambda 함수는 사용자 데이터 검증을 수행하고 `Moscow`가 잘못된 위치라고 판단합니다.

> **Note**
>
> 이 연습의 Lambda 함수에는 유효한 도시가 나열된 간단한 목록이 있으며, `Moscow`는 이 목록에 없습니다. 프로덕션 애플리케이션에서는 백엔드 데이터베이스를 사용하여 이 정보를 확인할 수 있습니다.

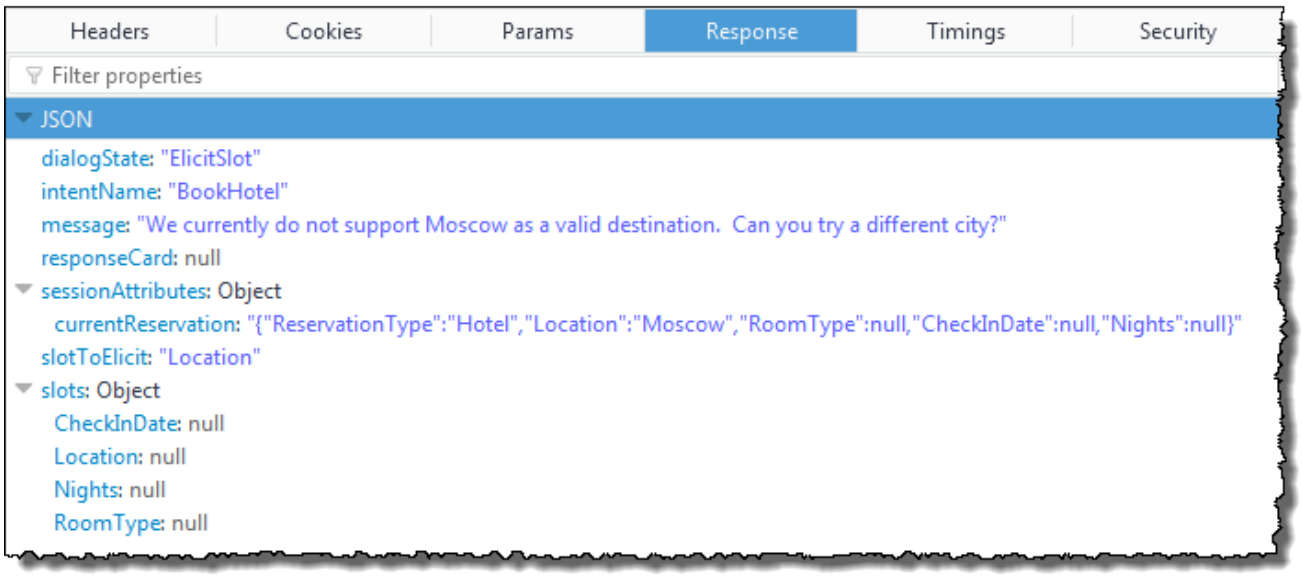
슬롯 값을 null로 다시 재설정하고, 다음 응답을 보내 사용자에게 다른 값을 입력하라는 메시지를 표시하도록 Amazon Lex에 지시합니다.

```
{
    "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`은 null로 재설정됩니다.
> - `dialogAction.type`은 `ElicitSlot`으로 설정됩니다. 이는 다음을 제공하여 사용자에게 메시지를 다시 표시하도록 Amazon Lex에 지시합니다.
> - `dialogAction.slotToElicit` – 사용자의 데이터를 유도하는 슬롯입니다.
> - `dialogAction.message` – 사용자에게 전달할 `message`입니다.

d. Amazon Lex는 `dialogAction.type`을 확인하고 다음 응답에서 클라이언트에 정보를 전달합니다.



클라이언트는 단순히 "현재 유효한 목적지로 모스크바는 지원되지 않습니다. 다른 도시를 선택하시겠어요?"라는 메시지를 표시합니다.

3. 사용자: "시카고"

   a. 클라이언트는 Amazon Lex에 다음 `PostText` 요청을 보냅니다.

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText":"Chicago",
   "sessionAttributes":{
       "currentReservation":"{\"ReservationType\":\"Hotel\",
                              \"Location\":\"Moscow\",
                              \"RoomType\":null,
                              \"CheckInDate\":null,
                              \"Nights\":null}"
   }
}
```

   b. Amazon Lex는 컨텍스트를 파악하고 이를 통해 `Location` 슬롯에 대한 데이터를 유도했습니다. 이 컨텍스트에서 `inputText` 값이 `Location` 슬롯에 대한 것임을 압니다. 그런 다음, 다음 이벤트를 보내 Lambda 함수를 호출합니다.

```
{
    "messageVersion": "1.0",
    "invocationSource": "DialogCodeHook",
    "userId": "wch89kjqcpkds8seny7dly5x3otq68j3",
    "sessionAttributes": {
        "currentReservation": "{\"ReservationType\":\"Hotel\",\"Location
\":Moscow,\"RoomType\":null,\"CheckInDate\":null,\"Nights\":null}"
    },
    "bot": {
        "name": "BookTrip",
        "alias": null,
        "version": "$LATEST"
    },
    "outputDialogMode": "Text",
    "currentIntent": {
        "name": "BookHotel",
```

```
        "slots": {
            "RoomType": null,
            "CheckInDate": null,
            "Nights": null,
            "Location": "Chicago"
        },
        "confirmationStatus": "None"
    }
}
```

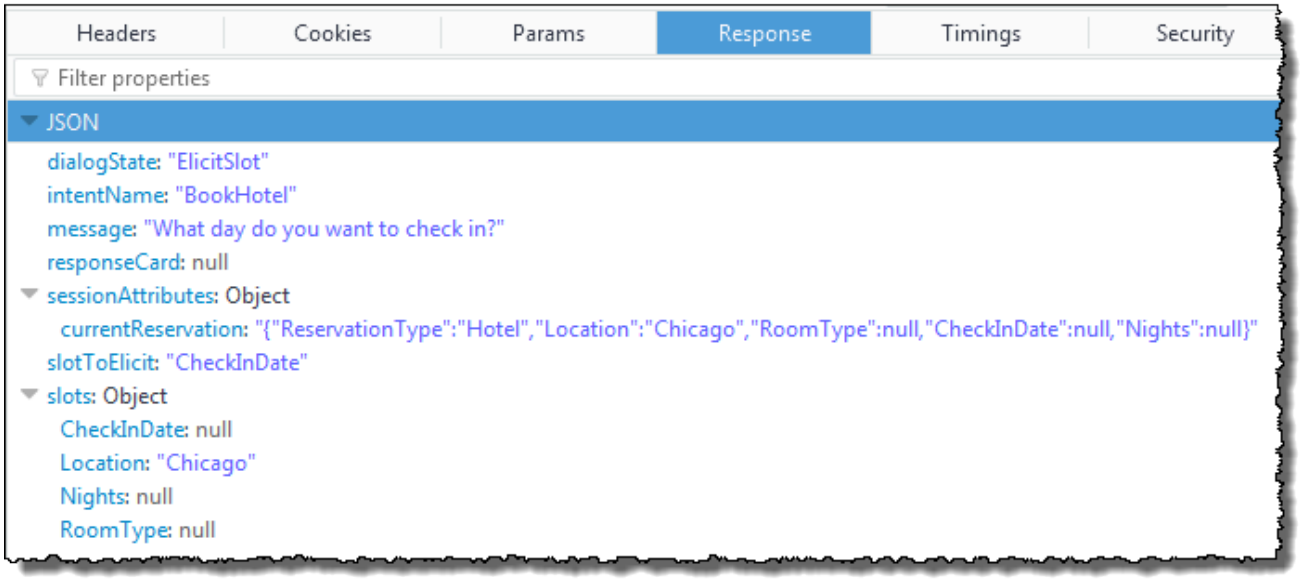
Amazon Lex는 `currentIntent.slots`슬롯을 `Location`로 설정하여 `Chicago`를 업데이트했습니다.

c. `DialogCodeHook`의 `invocationSource`값에 따라 Lambda 함수는 사용자 데이터 검증을 수행합니다. `Chicago`를 유효한 슬롯 값으로 인식하고, 세션 속성을 이에 따라 업데이트한 다음, Amazon Lex에 다음 응답을 반환합니다.

```
{
    "sessionAttributes": {
        "currentReservation": "{\"ReservationType\":\"Hotel\",\"Location\":
\"Chicago\",\"RoomType\":null,\"CheckInDate\":null,\"Nights\":null}"
    },
    "dialogAction": {
        "type": "Delegate",
        "slots": {
            "RoomType": null,
            "CheckInDate": null,
            "Nights": null,
            "Location": "Chicago"
        }
    }
}
```

> **Note**
>
> - `currentReservation` – Lambda 함수는 `Location`을 `Chicago`로 설정하여 이 세션 속성을 업데이트합니다.
> - `dialogAction.type` – `Delegate`로 설정됩니다. 사용자 데이터가 유효했으므로 Lambda 함수는 일련의 다음 조치를 선택하도록 Amazon Lex에 지시합니다.

d. `dialogAction.type`에 따라 Amazon Lex는 일련의 다음 조치를 선택합니다. Amazon Lex는 더 많은 슬롯 데이터가 필요함을 알고, 의도 구성에 따라 우선 순위가 가장 높은 채워지지 않은 다음 슬롯(`CheckInDate`)을 선택합니다. 의도 구성에 따라 이 슬롯에 대한 프롬프트 메시지 중 하나("어떤 도시에서 묵고 싶으세요?")를 선택하고, 클라이언트에 다음 응답을 다시 보냅니다.

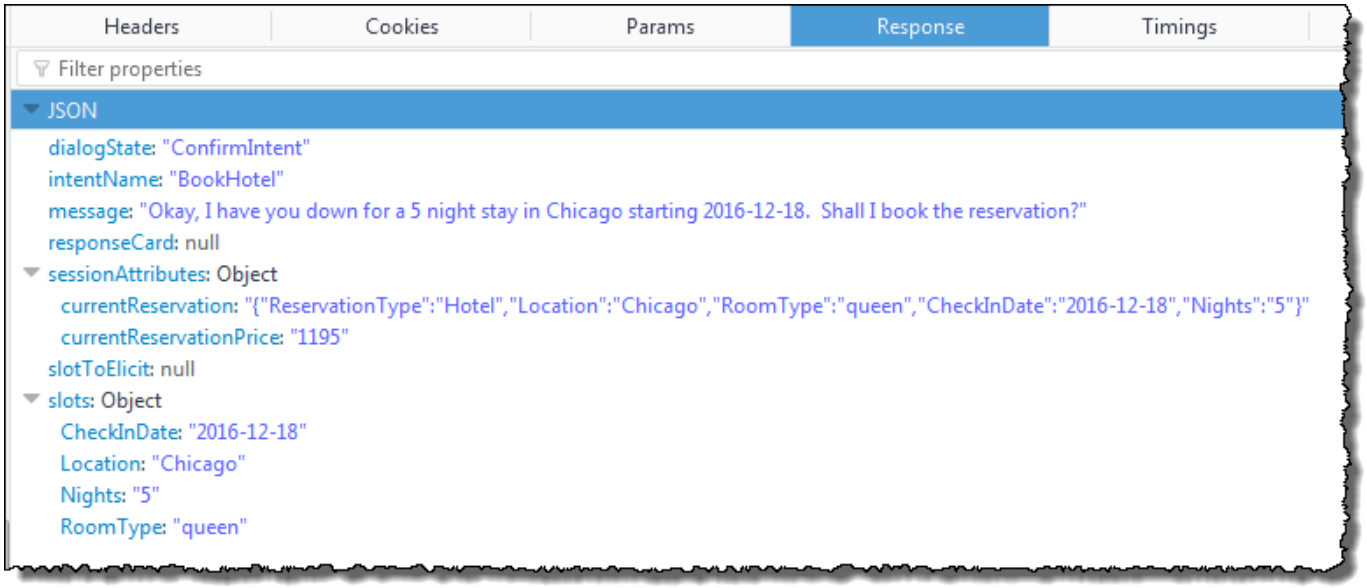


클라이언트는 "어떤 도시에서 묵고 싶으세요?"라는 메시지를 표시합니다.

4. 사용자의 상호 작용은 사용자가 데이터를 제공하며 계속됩니다. Lambda 함수는 데이터를 검증한 다음, 일련의 다음 조치를 Amazon Lex에 위임합니다. 결과적으로 사용자가 모든 슬롯 데이터를 제공하고 Lambda 함수가 모든 사용자 입력을 검증하면 Amazon Lex는 모든 슬롯 데이터를 보유했음을 인식합니다.

> **Note**
>
> 이 연습에서는 사용자가 모든 슬롯 데이터를 제공하면 Lambda 함수가 호텔 예약 가격을 계산하고 다른 세션 속성(`currentReservationPrice`)으로 이를 반환합니다.

이 시점에서 의도를 이행할 준비가 되었으나, Amazon Lex가 의도를 이행하기 전에 BookHotel 의도가 사용자 확인을 요구하는 확인 프롬프트로 구성됩니다. 따라서 Amazon Lex는 호텔을 예약하기 전에 확인을 요청하는 다음 메시지를 클라이언트에 보냅니다.



클라이언트는 "네, 시작 날짜 2016-12-18, 시카고에서 5박 확인했습니다. 예약을 진행할까요?"

5. 사용자: "예"

   a. 클라이언트는 Amazon Lex에 다음 `PostText` 요청을 보냅니다.

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText":"Yes",
   "sessionAttributes":{
       "currentReservation":"{\"ReservationType\":\"Hotel\",
                              \"Location\":\"Chicago\",
                              \"RoomType\":\"queen\",
                              \"CheckInDate\":\"2016-12-18\",
                              \"Nights\":\"5\"}",
      "currentReservationPrice":"1195"
   }
}
```

b. Amazon Lex는 현재 의도를 확인하는 컨텍스트에서 `inputText`를 해석합니다. Amazon Lex는 사용자가 예약을 진행하기를 원한다고 이해합니다. 이때 Amazon Lex는 다음 이벤트를 보내 의도를 이행하도록 Lambda 함수를 호출합니다. 이벤트에서 `invocationSource`를 `FulfillmentCodeHook`로 설정하여 Lambda 함수에 보냅니다. Amazon Lex는 또한 다음과 같이 `confirmationStatus`을 `Confirmed`로 설정합니다.

```
{
    "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` – 이때 Amazon Lex는 이 값을 `FulfillmentCodeHook`로 설정하여, 의도를 이행하도록 Lambda 함수에 지시합니다.
> - `confirmationStatus` – `Confirmed`로 설정됩니다.

c. 이때 Lambda 함수는 BookHotel 의도를 이행하며, Amazon Lex는 예약을 완료한 후 다음 응답을 반환합니다.

```
{
    "sessionAttributes": {
        "lastConfirmedReservation": "{\"ReservationType\":\"Hotel\",\"Location
\":\"Chicago\",\"RoomType\":\"queen\",\"CheckInDate\":\"2016-12-18\",\"Nights
\":\"5\"}"
    },
    "dialogAction": {
        "type": "Close",
        "fulfillmentState": "Fulfilled",
        "message": {
            "contentType": "PlainText",
            "content": "Thanks, I have placed your reservation.   Please let me
 know if you would like to book a car rental, or another hotel."
        }
    }
}
```

> **Note**
>
> - `lastConfirmedReservation` – Lambda 함수가 추가한 새로운 세션 속성 (`currentReservation`, `currentReservationPrice`대신)입니다.
> - `dialogAction.type` – Lambda 함수는 이 값을 `Close`로 설정하여 Amazon Lex가 사용자 응답을 기대하지 않도록 합니다.
> - `dialogAction.fulfillmentState` – `Fulfilled`로 설정되고, 사용자에게 전달할 적절한 `message`를 포함합니다.

d. Amazon Lex는 `fulfillmentState`를 검토하고 클라이언트에 다음 응답을 보냅니다.

```
Headers | Cookies | Params | Response | Timings
Filter properties
JSON
  dialogState: "Fulfilled"
  intentName: "BookHotel"
  message: "Thanks, I have placed your reservation.  Please let me know if you would like to book a car rental, or another hotel."
  responseCard: null
  sessionAttributes: Object
    lastConfirmedReservation: "{"ReservationType":"Hotel","Location":"Chicago","RoomType":"queen","CheckInDate":"2016-12-18","Nights":"5"}"
  slotToElicit: null
  slots: Object
    CheckInDate: "2016-12-18"
    Location: "Chicago"
    Nights: "5"
    RoomType: "queen"
```


> **Note**
>
> - `dialogState` – Amazon Lex는 이 값을 `Fulfilled`로 설정합니다.
> - `message` – Lambda 함수가 제공한 동일한 메시지입니다.

클라이언트가 이 메시지를 표시합니다.

데이터 흐름: 차량 예약 의도

이 연습에서 BookTrip 봇은 두 가지 의도(BookHotel 및 BookCar)를 지원합니다. 사용자는 호텔을 예약한 후 대화를 계속하여 차량을 예약할 수 있습니다. 세션이 시간 초과되지 않는 한 각 후속 요청에서 클라이언트는 계속해서 세션 속성(여기서는 `lastConfirmedReservation`)을 보냅니다. Lambda 함수는 이 정보를 사용하여 BookCar 의도에 대한 슬롯 데이터를 초기화할 수 있습니다. 이는 의도 전반에 걸친 데이터 공유에서 세션 속성을 사용하는 방법을 보여 줍니다.

특히, 사용자가 BookCar 의도를 선택하면 Lambda 함수는 세션 속성의 관련 정보를 사용하여 BookCar 의도에 대한 슬롯(PickUpDate, ReturnDate, PickUpCity)을 미리 채웁니다.

> **Note**
>
> Amazon Lex 콘솔은 지우기 링크를 제공하며, 사용자는 이 링크를 사용하여 이전 세션 속성을 지울 수 있습니다.

대화를 계속하려면 이 절차의 단계를 따릅니다.

1. 사용자: "차량도 예약해줘"

   a. 클라이언트는 Amazon Lex에 다음 `PostText` 요청을 보냅니다.

```
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\"}"
   }
}
```

   클라이언트에는 `lastConfirmedReservation` 세션 속성이 포함되어 있습니다.

   b. Amazon Lex는 `inputText`에서 의도(BookCar)를 감지합니다. 이 의도는 사용자 데이터의 초기화 및 검증을 수행하기 위해 Lambda 함수를 호출하도록 구성되어 있습니다. Amazon Lex는 다음 이벤트를 사용하여 Lambda 함수를 호출합니다.

```
{
    "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` – Amazon Lex는 현재 1.0 버전만 지원합니다.
> - `invocationSource` – 호출의 목적이 초기화 및 사용자 데이터 검증을 수행하는 것임을 나타냅니다.
> - `currentIntent` – 의도 이름과 슬롯을 포함합니다. 이때 모든 슬롯 값은 null입니다.

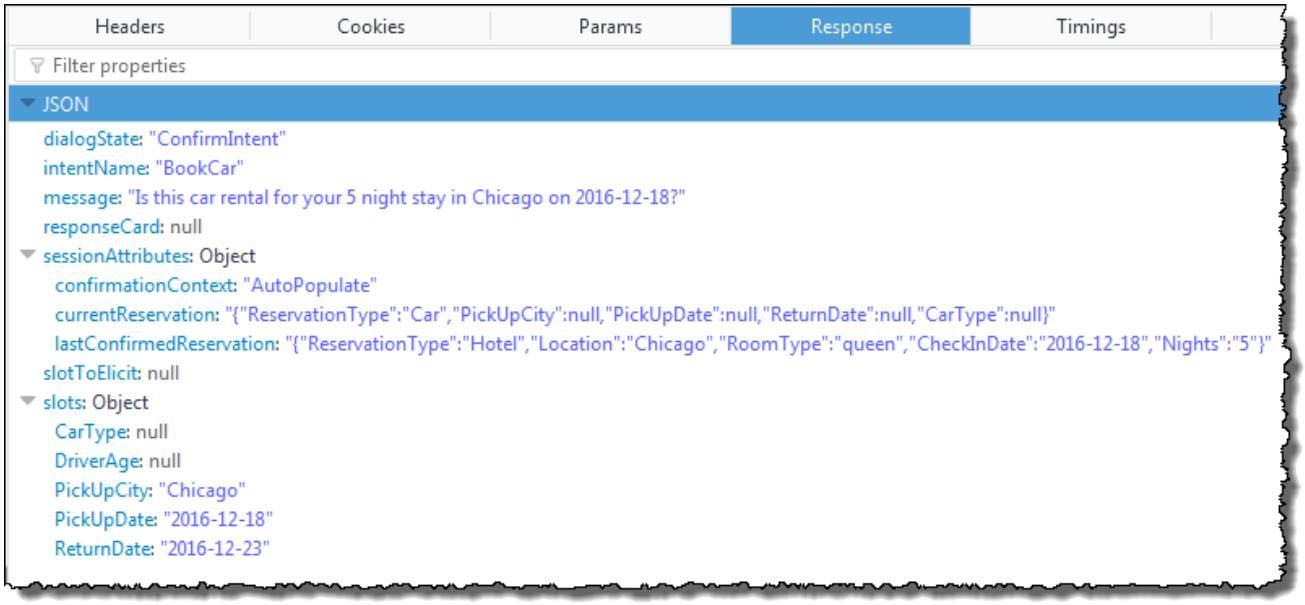
c. Lambda 함수는 검증할 필요가 없는 모든 null 슬롯 값을 확인합니다. 그러나 세션 속성을 사용하여 일부 슬롯 값(`PickUpDate`, `ReturnDate`, `PickUpCity`)을 초기화한 후, 다음 응답을 반환합니다.

```
{
    "sessionAttributes": {
        "lastConfirmedReservation": "{\"ReservationType\":\"Hotel\",\"Location
\":\"Chicago\",\"RoomType\":\"queen\",\"CheckInDate\":\"2016-12-18\",\"Nights
\":\"5\"}",
        "currentReservation": "{\"ReservationType\":\"Car\",\"PickUpCity
\":null,\"PickUpDate\":null,\"ReturnDate\":null,\"CarType\":null}",
        "confirmationContext": "AutoPopulate"
    },
    "dialogAction": {
        "type": "ConfirmIntent",
        "intentName": "BookCar",
        "slots": {
            "PickUpCity": "Chicago",
            "PickUpDate": "2016-12-18",
```

```
            "ReturnDate": "2016-12-22",
            "CarType": null,
            "DriverAge": null
        },
        "message": {
            "contentType": "PlainText",
            "content": "Is this car rental for your 5 night stay in Chicago on
 2016-12-18?"
        }
    }
}
```

> **Note**
>
> - `lastConfirmedReservation` 외에도, Lambda 함수는 더 많은 세션 속성(`currentReservation` 및 `confirmationContext`)을 포함합니다.
> - `dialogAction.type`은 `ConfirmIntent`로 설정되어, 사용자의 예/아니요 응답이 필요함을 Amazon Lex에 알립니다(confirmationContext는 AutoPopulate로 설정됨). 이때 Lambda 함수는 이것이 Lambda 함수가 수행한 초기화(슬롯 데이터 자동 입력)에 대해 사용자가 예/아니요로 확인하는 응답임을 알고 있습니다.
>
> 또한 Lambda 함수는 `dialogAction.message`의 정보 메시지를 응답에 포함시켜 Amazon Lex가 클라이언트에 반환하도록 합니다.
>
> > **Note**
> >
> > 용어 `ConfirmIntent`(`dialogAction.type`의 값)는 봇 의도와 관련이 없습니다. 이 예에서 Lambda 함수는 이 용어를 사용하여 사용자로부터 예 또는 아니요 응답을 얻도록 Amazon Lex에 지시합니다.

d. `dialogAction.type`에 따라 Amazon Lex는 클라이언트에 다음 응답을 반환합니다.



클라이언트는 "2016-12-18에 시작하는 시카고에서의 5박 동안 사용할 차량의 렌트인가요?"라는 메시지를 표시합니다.

2. 사용자: "예"

   a. 클라이언트는 Amazon Lex에 다음 `PostText` 요청을 보냅니다.

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText":"yes",
   "sessionAttributes":{
      "confirmationContext":"AutoPopulate",
      "currentReservation":"{\"ReservationType\":\"Car\",
                            \"PickUpCity\":null,
                            \"PickUpDate\":null,
                            \"ReturnDate\":null,
                            \"CarType\":null}",
      "lastConfirmedReservation":"{\"ReservationType\":\"Hotel\",
                                   \"Location\":\"Chicago\",
                                   \"RoomType\":\"queen\",
                                   \"CheckInDate\":\"2016-12-18\",
                                   \"Nights\":\"5\"}"
   }
}
```

b. Amazon Lex는 `inputText`를 읽고 컨텍스트를 알고 있습니다(사용자에게 자동 채우기를 확인하도록 요청). Amazon Lex는 다음 이벤트를 보내 Lambda 함수를 호출합니다.

```
{
    "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"
    }
}
```

사용자가 예라고 답했으므로 Amazon Lex는 `confirmationStatus`를 `Confirmed`로 설정합니다.

c. Lambda 함수는 `confirmationStatus`로부터 미리 채워진 값이 옳다는 것을 알고 있습니다. Lambda 함수는 다음 작업을 수행합니다.

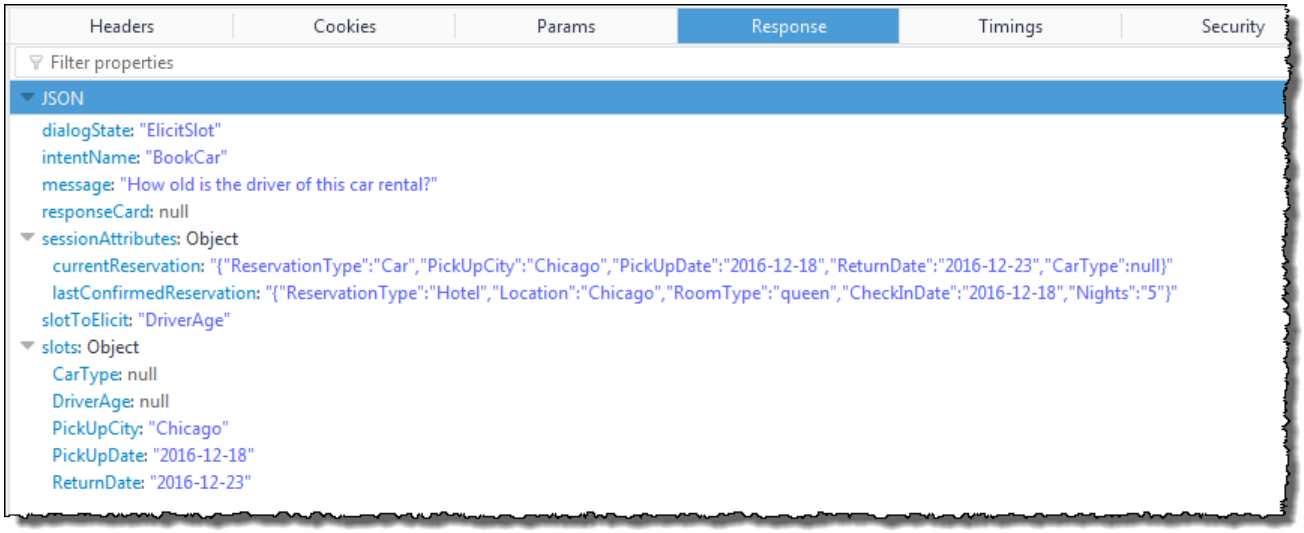
- `currentReservation` 세션 속성을 미리 채워진 슬롯 값으로 업데이트

- `dialogAction.type`을 `ElicitSlot`으로 설정
- `slotToElicit` 값을 `DriverAge`로 설정

다음 응답이 전송됩니다.

```
{
    "sessionAttributes": {
        "currentReservation": "{\"ReservationType\":\"Car\",\"PickUpCity
\":\"Chicago\",\"PickUpDate\":\"2016-12-18\",\"ReturnDate\":\"2016-12-22\",
\"CarType\":null}",
        "lastConfirmedReservation": "{\"ReservationType\":\"Hotel\",\"Location
\":\"Chicago\",\"RoomType\":\"queen\",\"CheckInDate\":\"2016-12-18\",\"Nights
\":\"5\"}"
    },
    "dialogAction": {
        "type": "ElicitSlot",
        "intentName": "BookCar",
        "slots": {
            "PickUpDate": "2016-12-18",
            "ReturnDate": "2016-12-22",
            "DriverAge": null,
            "CarType": null,
            "PickUpCity": "Chicago"
        },
        "slotToElicit": "DriverAge",
        "message": {
            "contentType": "PlainText",
            "content": "How old is the driver of this car rental?"
        }
    }
}
```

d. Amazon Lex는 다음 응답을 반환합니다.



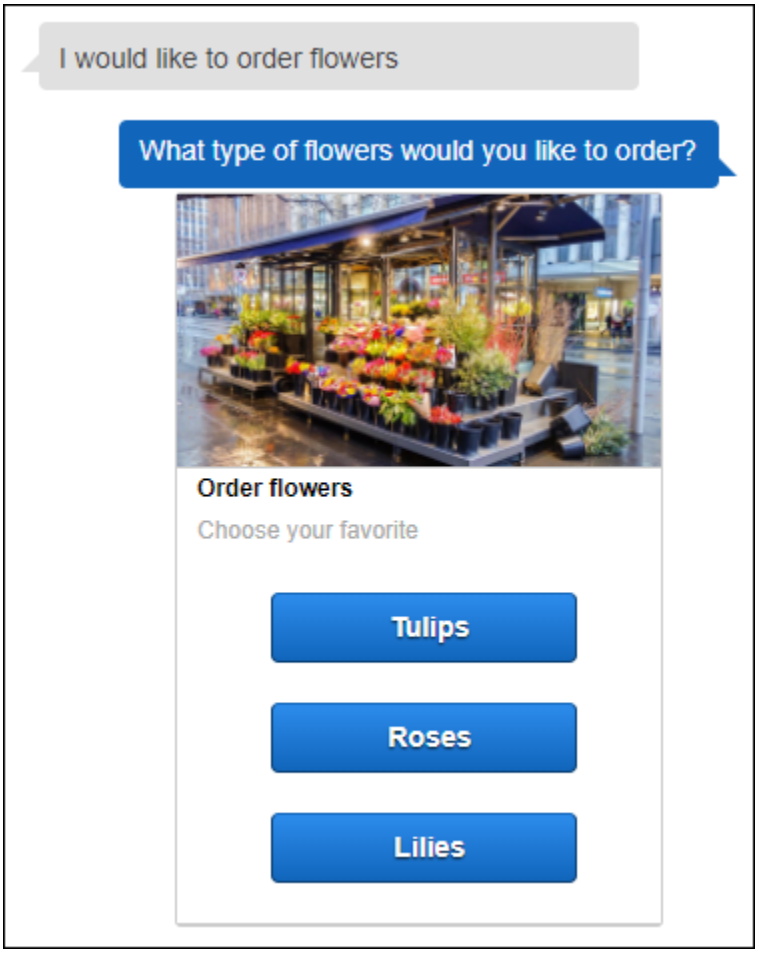
클라이언트가 "운전자의 나이는 몇살인가요?"라는 메시지를 표시합니다. 대화를 계속합니다.

응답 카드 사용

이 연습에서는 응답 카드를 추가하여 시작하기 연습 1을 확장합니다. OrderFlowers 의도를 지원하는 봇을 생성한 다음, `FlowerType` 슬롯에 대한 응답 카드를 추가하여 의도를 업데이트합니다. `FlowerType` 슬롯에 대한 다음 프롬프트 외에도 사용자는 응답 카드에서 꽃 유형을 선택할 수 있습니다.

```
What type of flowers would you like to order?
```

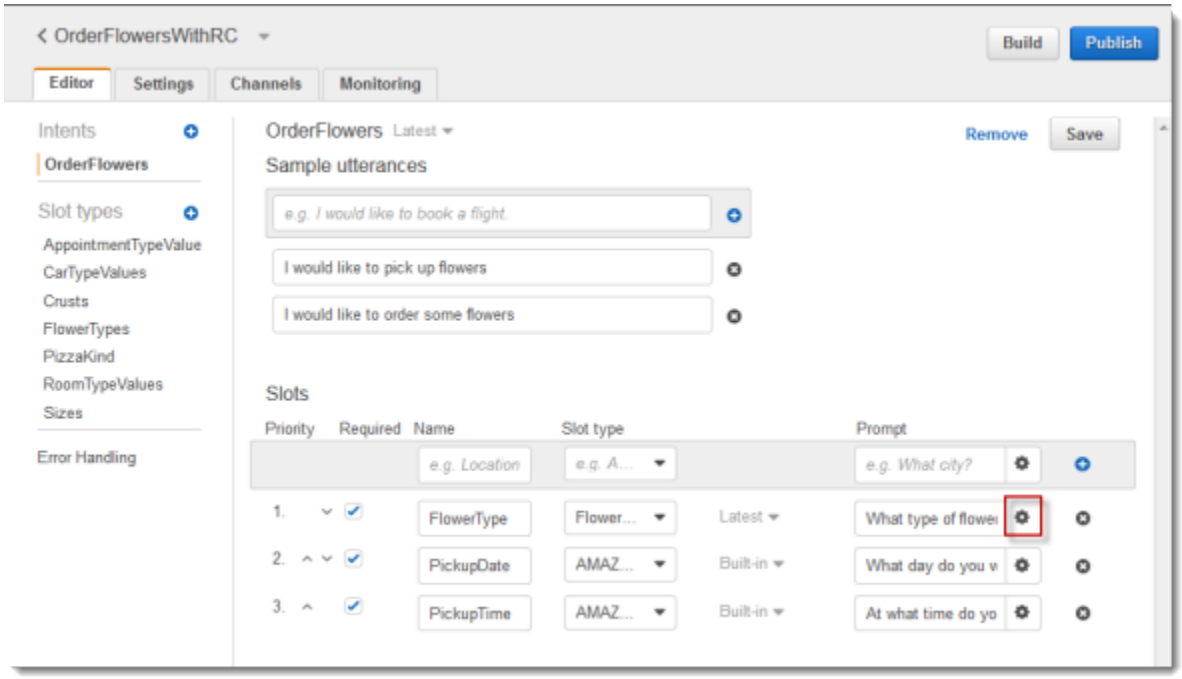
다음은 응답 카드입니다.



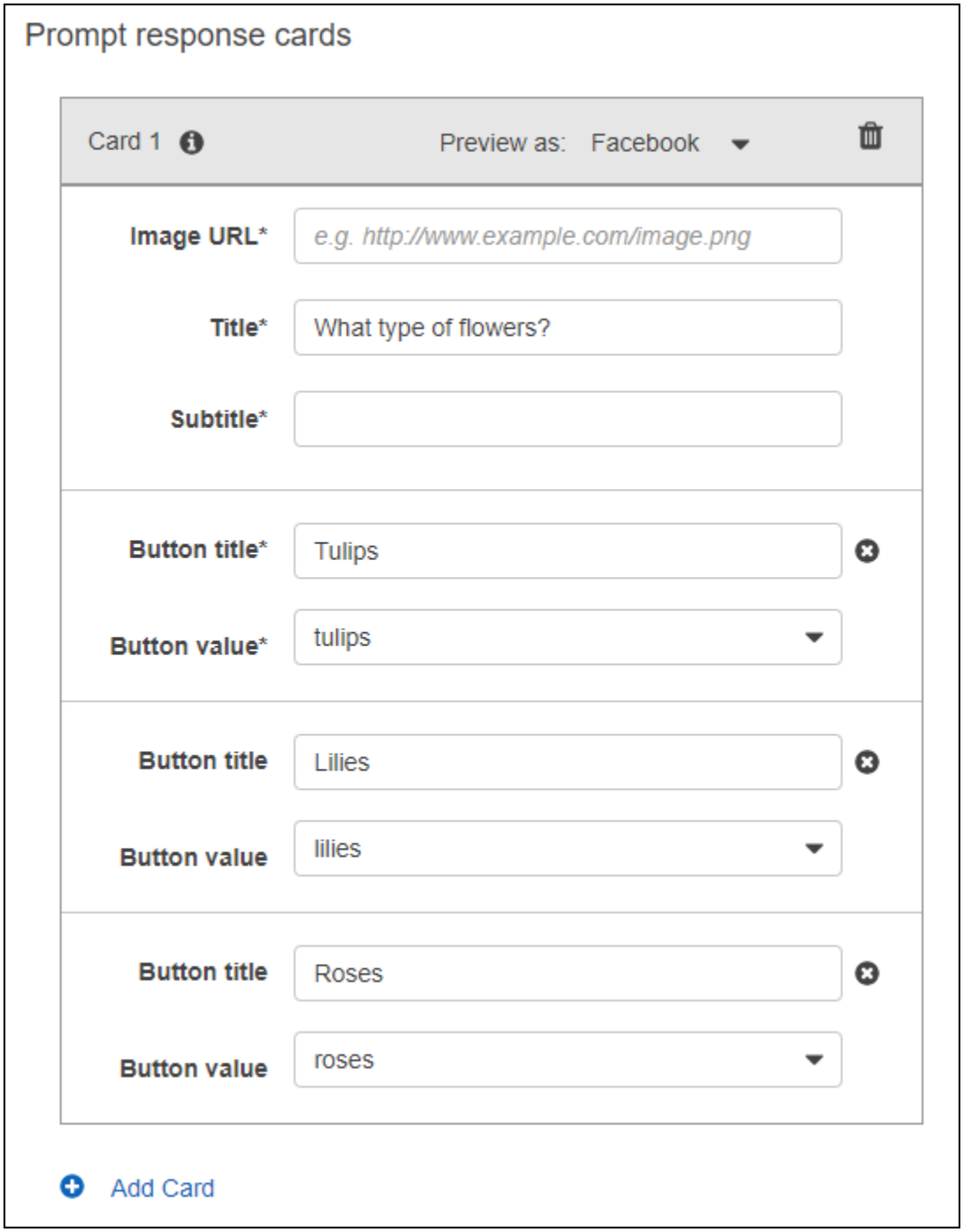
봇 사용자는 텍스트를 입력하거나 꽃 유형 목록에서 선택할 수 있습니다. 이 응답 카드는 이미지로 구성되어 있으며, 클라이언트에 다음과 같이 표시됩니다. 응답 카드에 대한 자세한 내용은 응답 카드을 참조하십시오.

응답 카드가 있는 봇을 생성하고 테스트하려면

1. 시작하기 연습 1에 따라 OrderFlowers 봇을 생성하고 테스트합니다. 1, 2, 3단계를 완료해야 합니다. 응답 카드를 테스트하기 위해 Lambda 함수를 추가할 필요는 없습니다. 지침은 연습 1: 청사진을 사용하여 Amazon Lex 봇 생성(콘솔)을 참조하세요.
2. 응답 카드를 추가하여 봇을 업데이트한 다음 버전을 게시합니다. 버전을 게시할 때 이를 가리키는 별칭(BETA)을 지정합니다.
   a. Amazon Lex 콘솔에서 봇을 선택합니다.
   b. `OrderFlowers` 의도를 선택합니다.
   c. "어떤 종류의 꽃" 프롬프트 옆의 기어 모양 설정 아이콘을 선택하여 다음 그림과 같이 `FlowerType`에 대한 응답 카드를 구성합니다.



d. 다음 스크린샷과 같이 카드에 제목을 부여하고 세 개의 버튼을 구성합니다. 이미지 URL이 있는 경우 응답 카드에 이미지를 선택적으로 추가할 수도 있습니다. Twilio SMS를 사용하여 봇을 배포하는 경우 이미지 URL을 제공해야 합니다.



e. 저장을 선택하여 응답 카드를 저장합니다.

f. 의도 저장을 선택하여 의도 구성을 저장합니다.

g. 봇을 구축하려면, 빌드를 선택합니다.

h. 봇 버전을 게시하려면 게시를 선택합니다. BETA를 봇 버전을 가리키는 별칭으로 지정합니다. 버전 관리에 대한 자세한 내용은 버전 관리 및 별칭를 참조하십시오.

3. 메시징 플랫폼에 봇 배포하기:

- Facebook Messenger 플랫폼에 봇을 배포하고 통합을 테스트합니다. 지침은 Amazon Lex 봇을 Facebook Messenger와 통합하기 단원을 참조하십시오. 꽃을 주문할 때 메시지 창에 꽃 유형을 선택할 수 있는 응답 카드가 표시됩니다.
- Slack 플랫폼에 봇을 배포하고 통합을 테스트합니다. 지침은 Amazon Lex 봇 슬랙에 통합하기 단원을 참조하십시오. 꽃을 주문할 때 메시지 창에 꽃 유형을 선택할 수 있는 응답 카드가 표시됩니다.
- Twilio SMS 플랫폼에 봇을 배포하세요. 지침은 Amazon Lex 봇을 Twilio의 프로그래밍 가능한 SMS와 통합 단원을 참조하십시오. 꽃을 주문하면 Twilio의 메시지에 응답 카드의 이미지가 표시됩니다. Twilio SMS는 응답 버튼을 지원하지 않습니다.

표현 업데이트

이 연습에서는 시작하기 연습 1에서 만든 표현에 표현을 추가합니다. Amazon Lex 콘솔의 모니터링 탭을 사용하여 봇이 인식하지 못한 표현을 볼 수 있습니다. 사용자 경험을 개선하려면 이러한 표현을 봇에 추가합니다.

다음 조건에서는 표현 통계가 생성되지 않습니다.

- 봇이 생성될 때 `childDirected` 필드는 참으로 설정되었습니다.
- 하나 이상의 슬롯에서 슬롯 난독화 기능을 사용하고 있습니다.
- Amazon Lex 개선에 참여하지 않기로 선택했습니다.

> **Note**
>
> 표현 통계는 매일 한 번 생성됩니다. 인식되지 않은 표현, 들은 횟수, 마지막으로 표현을 들은 날짜와 시간을 확인할 수 있습니다. 그 후 해당 데이터가 콘솔에 표시되는 데 최대 24시간이 걸릴 수 있습니다.

봇의 여러 버전의 표현을 볼 수 있습니다. 표현이 표시되는 봇 버전을 변경하려면 봇 이름 옆의 드롭다운에서 다른 버전을 선택하세요.

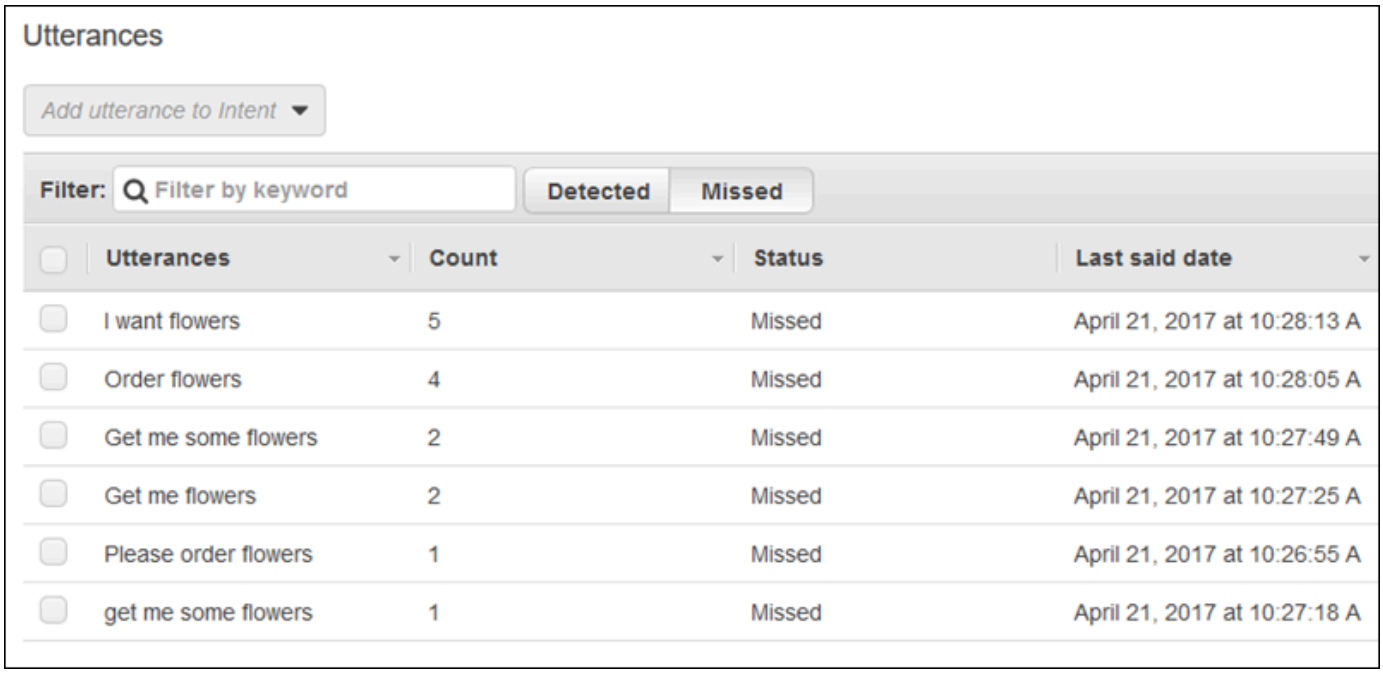
누락된 표현을 보고 봇에 추가하려면:

1. 시작하기 연습 1에 따라 `OrderFlowers` 봇을 생성하고 테스트합니다. 지침은 연습 1: 청사진을 사용하여 Amazon Lex 봇 생성(콘솔) 단원을 참조하십시오.

2. 테스트 봇 창에 다음 표현을 입력하여 봇을 테스트하십시오. 각 표현을 여러 번 입력합니다. 예시 봇은 다음 표현을 인식하지 못합니다.

   - 꽃 주문
   - 꽃 가져다 줘
   - 꽃을 주문하세요
   - 꽃 좀 가져다 주세요

3. Amazon Lex가 누락한 표현에 대한 사용 데이터를 수집할 때까지 기다리십시오. 표현 데이터는 하루에 한 번, 일반적으로 밤사이에 생성됩니다.
4. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lex/ Amazon Lex 콘솔을 엽니다.
5. `OrderFlowers` 봇을 선택하세요.
6. 모니터링 탭을 선택한 다음 왼쪽 메뉴에서 표현을 선택한 다음 누락 버튼을 선택합니다. 다음 창에는 누락된 표현이 최대 100개까지 표시됩니다.



7. 봇에 추가하고자 하는 누락된 표현을 선택하려면 옆의 확인란을 선택합니다. 의도 `$LATEST` 버전에 표현을 추가하려면 의도에 표현 추가 드롭다운 옆의 아래쪽 화살표를 선택한 다음 의도를 선택합니다.
8. 봇을 다시 빌드하려면, 빌드를 선택한 다음 빌드를 다시 선택하여 봇을 다시 빌드합니다.
9. 봇이 새 표현을 인식하는지 확인하려면 봇 테스트 창을 사용하세요.

웹 사이트와의 통합

이 예제에서는 텍스트와 음성을 사용하여 봇을 웹 사이트에 통합합니다. JavaScript 및 AWS 서비스를 사용하여 웹 사이트 방문자를 위한 대화형 환경을 구축합니다. AWS AI 블로그에 문서화된 다음 예제 중에서 선택할 수 있습니다.

- Chatbot용 웹 UI 배포—Amazon Lex 챗봇용 웹 클라이언트를 제공하는 모든 기능을 갖춘 웹 UI를 보여줍니다. 이를 웹 클라이언트에 대해 배우는 데 사용하거나 자신의 애플리케이션을 위한 빌딩 블록으로 사용할 수 있습니다.

- "안녕하세요, 방문자 여러분!"—Amazon Lex를 통한 웹 사용자 참여—Amazon Lex, AWS SDK for JavaScript in the Browser, Amazon Cognito를 사용하여 웹 사이트에서 대화형 환경을 구축하는 방법을 보여줍니다.

- 브라우저에서 음성 입력을 캡처하여 Amazon Lex로 보내기—브라우저에서 JavaScript용 SDK를 사용하여 웹 사이트에 음성 기반 챗봇을 내장하는 방법을 보여줍니다. 애플리케이션은 오디오를 녹음하고 Amazon Lex로 오디오를 전송한 다음 응답을 재생합니다.

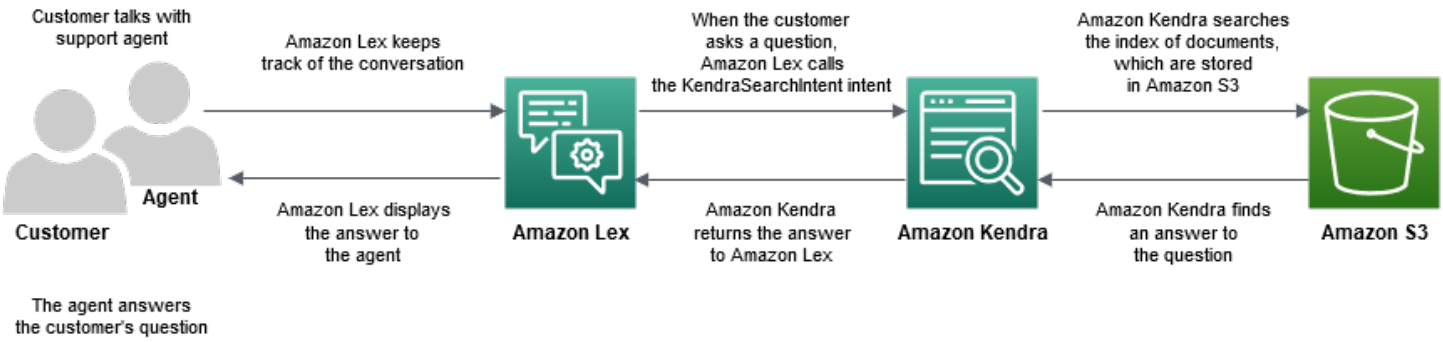
콜센터 에이전트 어시스턴트

이 튜토리얼에서는 Amazon Lex와 Amazon Kendra를 사용하여 고객 지원 에이전트를 지원하는 에이전트 지원 봇을 구축하고 이를 웹 애플리케이션으로 게시합니다. Amazon Kendra는 기계 학습을 사용하여 문서를 검색하여 답을 찾는 엔터프라이즈 검색 서비스입니다. Amazon Kendra에 대한 자세한 내용은 Amazon Kendra 개발자 가이드를 참조하세요.

Amazon Lex 봇은 콜 센터에서 고객의 첫 번째 연락 창구로 널리 사용됩니다. 봇은 종종 고객 질문을 해결할 수 있습니다. 봇이 질문에 답할 수 없는 경우 대화를 고객 지원 담당자에게 넘깁니다.

이 튜토리얼에서는 에이전트가 고객 쿼리에 실시간으로 답변하는 데 사용하는 Amazon Lex 봇을 만듭니다. 봇이 제공하는 답변을 읽으므로 에이전트는 일일이 답변을 찾아볼 필요가 없습니다.

이 튜토리얼에서 만드는 봇 및 웹 애플리케이션은 에이이전트가 적절한 리소스를 신속하게 제공하여 고객에게 효율적이고 정확하게 응답하는 데 도움이 됩니다. 다음 다이어그램은 작동 방식을 보여 줍니다.



다이어그램에서 볼 수 있듯이 문서의 Amazon Kendra 인덱스는 Amazon Simple Storage Service(S3) 버킷에 저장됩니다. S3 버킷이 아직 없는 경우 Amazon Kendra 인덱스를 생성할 때 설정할 수 있습니다. 이 튜토리얼에서는 Amazon S3 외에도 Amazon Cognito 를 사용하게 됩니다. Amazon Cognito 는 봇을 웹 애플리케이션으로 배포하기 위한 권한을 관리합니다.

이 튜토리얼에서는 고객 질문에 대한 답변을 제공하는 Amazon Kendra 색인을 생성하고, 봇을 생성하고, 고객과의 대화를 기반으로 답변을 제안할 수 있는 의도를 추가하고, 액세스 권한을 관리하도록 Amazon Cognito 를 설정하고, 봇을 웹 애플리케이션으로 배포합니다.

예상 소요 시간: 75분

예상 비용: Amazon Kendra 인덱스의 경우 시간당 2.50 USD, Amazon Lex 요청 1000건의 경우 0.75 USD. Amazon Kendra 인덱스는 이 연습을 마친 후에도 계속 실행됩니다. 불필요한 비용이 발생하지 않도록 반드시 삭제하십시오.

참고: 이 튜토리얼에서 사용하는 모든 서비스에 대해 동일한 AWS 리전을 선택해야 합니다.

주제

- 1단계: Amazon Kendra 인덱스를 생성합니다.
- 2단계: Amazon Lex 봇 생성
- 3단계: 사용자 지정 및 기본 제공 의도에 추가
- 4단계: Amazon Cognito 설정
- 5단계: 봇을 웹 애플리케이션으로 배포
- 6단계: 봇 사용

1단계: Amazon Kendra 인덱스를 생성합니다.

먼저 고객 질문에 답하는 Amazon Kendra 문서 색인을 생성하십시오. 인덱스는 클라이언트 쿼리를 위한 검색 API를 제공합니다. 소스 문서에서 색인을 만듭니다. Amazon Kendra는 인덱싱된 문서에서 찾은 답변을 봇에 반환하고 봇은 이를 에이전트에게 표시합니다.

Amazon Kendra에서 제안하는 응답의 품질과 정확성은 색인을 생성하는 문서에 따라 달라집니다. 문서에는 에이전트가 자주 액세스하고 S3 버킷에 저장해야 하는 파일이 포함되어야 합니다. .html, Microsoft Office(.doc, .ppt), PDF 및 텍스트 형식으로 비정형 및 반정형 데이터를 인덱싱할 수 있습니다.

Amazon Kendra 인덱스를 생성하려면 Amazon Kendra 개발자 가이드의 S3 버킷 시작하기(콘솔)을 참조하십시오.

고객 쿼리에 답변하는 데 도움이 되는 질문 및 답변(FAQ)을 추가하려면 Amazon Kendra 개발자 가이드에 질문 및 답변 추가를 참조하십시오. 이 튜토리얼에서는 GitHub의 ML_FAQ.csv 파일을 사용하세요.

다음 단계

2단계: Amazon Lex 봇 생성

2단계: Amazon Lex 봇 생성

Amazon Lex는 콜 센터 에이전트와 Amazon Kendra 인덱스 간의 인터페이스를 제공합니다. 에이전트와 고객 간의 대화를 추적하고 고객이 묻는 질문에 따라 `AMAZON.KendraSearchIntent` 의도를 직접 호출합니다. 의도는 사용자가 수행하고자 하는 작업입니다.

Amazon Kendra는 인덱싱된 문서를 검색하고 봇에 표시되는 답변을 Amazon Lex에 반환합니다. 이 답변은 에이전트에게만 표시됩니다.

에이전트 어시스턴트 봇을 만들려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lex/ Amazon Lex 콘솔을 엽니다.
2. 탐색 창에서 봇을 선택합니다.
3. 생성(Create)을 선택합니다.
4. 사용자 지정 봇을 선택하고 봇을 구성합니다.

   a. 봇 이름 – 봇의 용도를 나타내는 이름(예: **`AgentAssistBot`**)을 지정합니다.

b. 음성 출력 – 없음을 선택합니다.
c. 세션 제한 시간 – **5**를 입력합니다.
d. COPPA – 아니요를 선택합니다.

5. 생성(Create)을 선택합니다. 봇을 생성한 후 Amazon Lex는 봇 편집기 탭을 표시합니다.

다음 단계

3단계: 사용자 지정 및 기본 제공 의도에 추가

3단계: 사용자 지정 및 기본 제공 의도에 추가

의도는 콜 센터 에이전트가 봇이 수행하기를 원하는 작업을 나타냅니다. 이 경우 에이전트은 에이전트가 고객과 나눈 대화를 바탕으로 봇이 답변과 유용한 리소스를 제안하기를 원합니다.

Amazon Lex에는 사용자 지정 의도와 내장 의도라는 두 가지 유형의 의도가 있습니다. `AMAZON.KendraSearchIntent`은 기본 제공 의도입니다. 봇은 `AMAZON.KendraSearchIntent` 의도를 사용하여 색인을 쿼리하고 Amazon Kendra에서 제안한 응답을 표시합니다.

이 예제의 봇에는 사용자 지정 의도가 필요하지 않습니다. 그러나 봇을 성공적으로 빌드하려면 하나 이상의 샘플 표현을 사용하여 하나 이상의 사용자 지정 의도를 생성해야 합니다. 이 의도는 에이전트 어시스턴트 봇을 빌드하는 데만 필요합니다. 다른 기능은 수행하지 않습니다. 이 의도의 표현은 고객이 묻는 질문에 해당하지 않아야 합니다. 이렇게 하면 `AMAZON.KendraSearchIntent`을 호출하여 고객 문의에 응답할 수 있습니다. 자세한 내용은 AMAZON.KendraSearchIntent을 참조하세요.

필요한 의도를 생성하려면

1. 봇 시작하기 페이지에서 의도 생성을 선택합니다.
2. 의도 추가에서 의도 생성을 선택합니다.
3. 의도 생성 대화 상자에서 의도의 이름(예: **`RequiredIntent`**)을 지정합니다.
4. 샘플 표현의 경우 설명적인 표현(예: **`Required utterance`**)를 입력합니다.
5. 의도 저장을 선택합니다.

AMAZON.KendraSearchIntent 의도 및 응답 메시지를 추가하려면

1. 탐색 창에서 의도 옆에 있는 더하기(+) 기호를 선택합니다.
2. 기존 의도 검색을 선택합니다.
3. 의도 검색 상자에 **`AMAZON.KendraSearchIntent`**를 입력한 다음 목록에서 이를 선택합니다.

4. 의도에 설명이 포함된 이름(예: **AgentAssistSearchIntent**)을 지정한 다음 추가를 선택합니다.
5. 의도 편집기에서 Amazon Kendra 쿼리를 선택하여 쿼리 옵션을 엽니다.
6. 의도에서 검색하려는 색인을 선택하고
7. 응답 섹션에서 다음 3가지 메시지를 추가합니다.

```
I found an answer for the customer query: ((x-amz-lex:kendra-search-response-
question_answer-question-1)) and the answer is ((x-amz-lex:kendra-search-response-
question_answer-answer-1)).
I found an excerpt from a helpful document: ((x-amz-lex:kendra-search-response-
document-1)).
I think this answer will help the customer: ((x-amz-lex:kendra-search-response-
answer-1)).
```

8. 의도 저장을 선택합니다.
9. 봇을 구축하려면 빌드를 선택합니다.

다음 단계

4단계: Amazon Cognito 설정

4단계: Amazon Cognito 설정

웹 애플리케이션의 권한 및 사용자를 관리하려면 Amazon Cognito 를 설정해야 합니다. Amazon Cognito 는 웹 애플리케이션의 보안과 액세스 제어를 보장합니다. Amazon Cognito는 자격 증명 풀을 사용하여 사용자에게 다른 AWS 서비스에 대한 액세스 권한을 부여하는 AWS 자격 증명을 제공합니다. 이 튜토리얼에서는 Amazon Lex에 대한 액세스를 제공합니다.

자격 증명 풀을 생성할 때 Amazon Cognito는 인증 및 미인증 사용자를 위한 AWS Identity and Access Management (IAM) 역할을 제공합니다. Amazon Lex에 대한 액세스 권한을 부여하는 정책을 추가하여 IAM 역할을 수정합니다.

Amazon Cognito 를 설정하려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/cognito/ Amazon Cognito 콘솔을 엽니다.
2. 자격 증명 풀 관리를 선택합니다.
3. 새 자격 증명 풀 생성을 선택합니다.

4. 자격 증명 풀을 구성합니다.
   a. 자격 증명 풀 이름 – 봇의 용도를 나타내는 이름(예: **BotPool**)을 지정합니다.
   b. 인증되지 않은 자격 증명에서 인증되지 않은 자격 증명에 대한 액세스 활성화를 선택합니다.
5. 풀 생성을 선택합니다.
6. 새 자격 증명 풀과 사용할 IAM 역할 식별 페이지에서 세부 정보 보기를 선택합니다.
7. IAM 역할 이름을 기록해 둡니다. 나중에 수정할 것입니다.
8. 허용을 선택합니다.
9. Amazon Cognito 시작하기 페이지에서, 플랫폼에 대해 JavaScript를 선택합니다.
10. 자격 AWS 증명 가져오기 섹션에서 자격 증명 풀 ID를 찾아 기록합니다.
11. Amazon Lex에 대한 액세스를 허용하려면 인증된 IAM 역할과 인증되지 않은 IAM 역할을 수정하십시오.
    a. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/iam/ IAM 콘솔을 엽니다.
    b. 탐색 창의 액세스 관리에서 역할을 선택합니다.
    c. 검색 상자에 인증된 IAM 역할의 이름을 입력하고 옆에 있는 체크박스를 선택합니다.
       i. 정책 연결을 선택합니다.
       ii. 검색 상자에 **AmazonLexRunBotsOnly**을 입력하고 옆에 있는 체크박스를 선택합니다.
       iii. 정책 연결을 선택합니다.
    d. 검색 상자에 미인증 IAM 역할의 이름을 입력하고 옆에 있는 체크박스를 선택합니다.
       i. 정책 연결을 선택합니다.
       ii. 검색 상자에 **AmazonLexRunBotsOnly**을 입력하고 옆에 있는 체크박스를 선택합니다.
       iii. 정책 연결을 선택합니다.

다음 단계

5단계: 봇을 웹 애플리케이션으로 배포

5단계: 봇을 웹 애플리케이션으로 배포

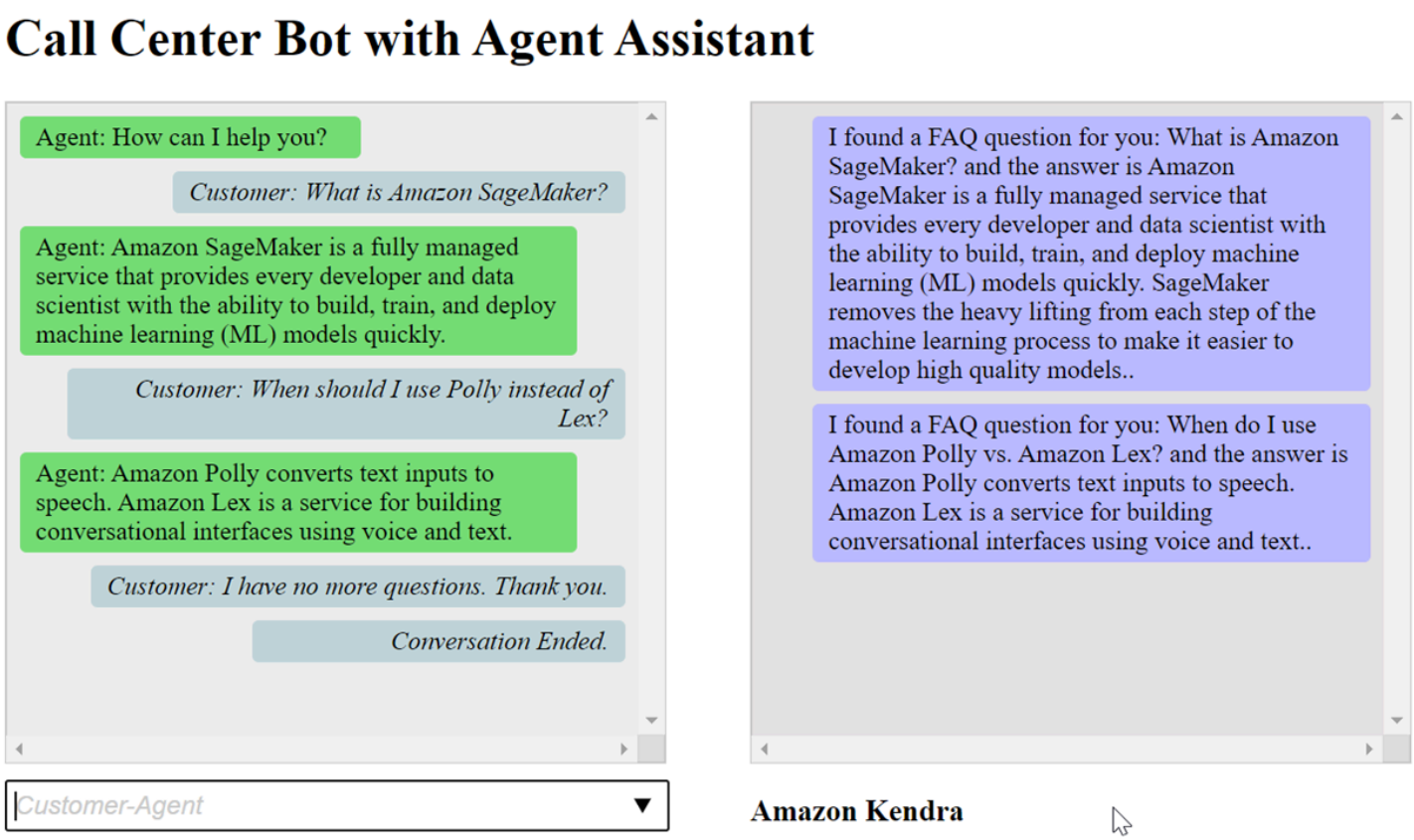
봇을 웹 애플리케이션으로 배포하려면

1. https://github.com/awsdocs/amazon-lex-developer-guide/blob/master/example_apps/agent_assistance_bot/의 리포지토리(repository)를 컴퓨터에 다운로드하세요.
2. 다운로드한 저장소로 이동하여 편집기에서 index.html 파일을 여세요.
3. 다음과 같이 변경합니다.
   a. `AWS.config.credentials` 섹션에 지역 이름과 자격 증명 풀 ID를 입력합니다.
   b. `Amazon Lex runtime parameters` 섹션에 봇 이름을 입력합니다.
   c. 파일을 저장합니다.

6단계: 봇 사용

데모용으로 봇에 고객 및 에이전트로서 입력을 제공합니다. 이 둘을 구분하기 위해 고객이 묻는 질문은 “고객:”으로 시작하고 에이전트가 제공하는 답변은 “에이전트:”로 시작합니다. 제안된 입력 메뉴 중에서 선택할 수 있습니다.

웹 애플리케이션을 열어 `index.html`을 실행하여 봇과 다음 이미지와 비슷한 대화에 참여하세요.



index.html 파일의 `pushChat()` 함수에 대한 설명은 다음과 같습니다.

```
            var endConversationStatement = "Customer: I have no more questions. Thank
you."
            // If the agent has to send a message, start the message with 'Agent'
            var inputText = document.getElementById('input');
            if (inputText && inputText.value && inputText.value.trim().length > 0 &&
inputText.value[0]=='Agent') {
                showMessage(inputText.value, 'agentRequest','conversation');
                inputText.value = "";
            }
            // If the customer has to send a message, start the message with 'Customer'
            if(inputText && inputText.value && inputText.value.trim().length > 0 &&
inputText.value[0]=='Customer') {
                // disable input to show we're sending it
                var input = inputText.value.trim();
                inputText.value = '...';
                inputText.locked = true;
                customerInput = input.substring(2);
```

```
                // Send it to the Lex runtime
                var params = {
                    botAlias: '$LATEST',
                    botName: 'KendraTestBot',
                    inputText: customerInput,
                    userId: lexUserId,
                    sessionAttributes: sessionAttributes
                };

                showMessage(input, 'customerRequest', 'conversation');
                if(input== endConversationStatement){
                    showMessage('Conversation
 Ended.','conversationEndRequest','conversation');
                }
                lexruntime.postText(params, function(err, data) {
                    if (err) {
                        console.log(err, err.stack);
                        showMessage('Error:  ' + err.message + ' (see console for
 details)', 'lexError', 'conversation1')
                    }

                    if (data &&input!=endConversationStatement) {
                        // capture the sessionAttributes for the next cycle
                        sessionAttributes = data.sessionAttributes;

                            showMessage(data, 'lexResponse', 'conversation1');
                    }
                    // re-enable input
                    inputText.value = '';
                    inputText.locked = false;
                });
            }
            // we always cancel form submission
            return false;
```

고객으로서 입력을 제공하면 Amazon Lex 런타임 API가 해당 입력을 Amazon Lex 로 전송합니다.

이 `showMessage(daText, senderRequest, displayWindow)` 함수는 채팅 창에 에이전트와 고객 간의 대화를 표시합니다. Amazon Kendra에서 제안한 응답은 인접한 창에 표시됩니다. 고객이 **“I have no more questions. Thank you.”**과 같이 말하면 대화가 종료됩니다.

참고: 사용하지 않을 때는 Amazon Kendra 색인을 삭제하십시오.

봇 마이그레이션

Amazon Lex V2 API는 업데이트된 정보 아키텍처를 사용하여 리소스 버전 관리를 간소화하고 봇에서 여러 언어를 지원할 수 있습니다. 자세한 내용은 Amazon Lex V2 개발자 가이드의 마이그레이션 가이드를 참조하십시오.

이러한 새 기능을 사용하려면 봇을 마이그레이션해야 합니다. 봇을 마이그레이션할 때 Amazon Lex는 다음을 제공합니다.

- 마이그레이션은 사용자 지정 의도와 슬롯 유형을 Amazon Lex V2 봇으로 복사합니다.
- 동일한 Amazon Lex V2 봇에 여러 언어를 추가할 수 있습니다. Amazon Lex V1에서는 각 언어에 대해 별도의 봇을 생성합니다. 각각 다른 언어를 사용하는 여러 Amazon Lex V1 봇을 하나의 Amazon Lex V2 봇으로 마이그레이션할 수 있습니다.
- Amazon Lex는 Amazon Lex V1의 내장 슬롯 유형과 의도를 Amazon Lex V2 내장 슬롯 유형 및 의도에 매핑합니다. 기본 제공을 마이그레이션할 수 없는 경우 Amazon Lex는 다음에 수행할 작업을 알려주는 메시지를 반환합니다.

마이그레이션 프로세스에서는 다음 항목을 마이그레이션하지 않습니다.

- 에일리어스
- Amazon Kendra 인덱스
- AWS Lambda 함수
- 대화 로그 설정
- Slack과 같은 메시징 채널
- Tags

봇을 마이그레이션하려면 사용자 또는 역할에 Amazon Lex 및 Amazon Lex V2 API 작업 모두에 대한 IAM 권한이 있어야 합니다. 필요한 권한에 대해서는 사용자가 봇을 Amazon Lex V2 API로 마이그레이션하도록 허용을 참조하세요.

봇 마이그레이션(콘솔)

Amazon Lex V1 콘솔을 사용하여 봇의 구조를 Amazon Lex V2 봇으로 마이그레이션할 수 있습니다.

콘솔을 사용하여 봇을 Amazon Lex V2 API로 마이그레이션하려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lex/ Amazon Lex 콘솔을 엽니다.
2. 왼쪽 메뉴에서 마이그레이션 도구를 선택합니다.
3. 봇 목록에서 마이그레이션하려는 봇을 선택한 다음 마이그레이션를 선택합니다.
4. 마이그레이션하려는 봇의 버전을 선택한 다음 마이그레이션할 봇의 이름을 입력합니다. 기존 Amazon Lex V2 봇의 이름을 입력하면 Amazon Lex V1 봇이 세부 정보에 표시된 언어로 마이그레이션되고 해당 언어의 초안 버전을 덮어씁니다.
5. 다음을 선택합니다.
6. Amazon Lex에서 Amazon Lex V2 API 버전의 봇을 실행하는 데 사용하는 IAM 역할을 선택합니다. 봇을 실행하는 데 필요한 최소 권한으로 새 역할을 생성할 수도 있고, 기존 IAM 역할 을 선택할 수도 있습니다.
7. 다음을 선택합니다.
8. 마이그레이션 설정을 검토합니다. 문제가 없다면 마이그레이션 시작을 선택합니다.

마이그레이션 프로세스를 시작하면 마이그레이션 도구 시작 페이지로 돌아갑니다. 히스토리 테이블에서 마이그레이션 진행률을 모니터링할 수 있습니다. 마이그레이션 상태 열에 완료라고 표시되면 마이그레이션이 완료된 것입니다.

Amazon Lex는 Amazon Lex V2 API의 `StartImport` 작업을 사용하여 마이그레이션된 봇을 가져옵니다. Amazon Lex V2 콘솔 가져오기 기록 표에 각 마이그레이션에 대한 항목이 표시됩니다.

Amazon Lex는 마이그레이션 중에 봇에서 마이그레이션할 수 없는 리소스를 발견할 수 있습니다. 마이그레이션할 수 없는 각 리소스에 대해 오류 또는 경고 메시지가 표시됩니다. 각 메시지에는 문제 해결 방법을 설명하는 설명서 링크가 포함되어 있습니다.

Lambda 함수 마이그레이션

Amazon Lex V2는 봇에 대해 Lambda 함수를 정의하는 방식을 변경합니다. 봇의 각 언어에 대한 별칭에 Lambda 함수를 하나만 사용할 수 있습니다. Lambda 함수 마이그레이션에 관한 더 자세한 내용은 Amazon Lex V1에서 Amazon Lex V2로 Lambda 함수 마이그레이션을 참조하세요.

마이그레이션 메시지

Amazon Lex는 마이그레이션 중에 내장 슬롯 유형과 같이 동등한 Amazon Lex V2 리소스로 마이그레이션할 수 없는 리소스를 발견할 수 있습니다. 이 경우 Amazon Lex는 발생한 상황을 설명하는 마이그레이션 메시지를 반환하고 마이그레이션 문제를 해결하는 방법을 알려주는 설명서 링크를 제공합니다. 다음 섹션에서는 봇을 마이그레이션할 때 발생할 수 있는 문제와 문제 해결 방법을 설명합니다.

주제

- 기본 제공 의도
- 기본 제공 슬롯 유형
- 대화 로그
- 메시지 그룹
- 프롬프트 및 프레이즈
- 기타 Amazon Lex V1 기능

기본 제공 의도

Amazon Lex V2에서 지원되지 않는 빌트인 의도를 사용하는 경우, 해당 의도는 Amazon Lex V2 봇의 사용자 지정 의도에 매핑됩니다. 사용자 지정 의도에는 표현이 포함되지 않습니다. 의도를 계속 사용하려면 샘플 표현을 추가하세요.

기본 제공 슬롯 유형

Amazon Lex V2에서 지원되지 않는 슬롯 유형을 사용하는 마이그레이션된 슬롯에는 슬롯 유형 값이 제공되지 않습니다. 이 슬롯을 사용하려면:

- 사용자 지정 슬롯 유형을 생성
- 슬롯 유형에 필요한 슬롯 유형 값을 추가
- 새 사용자 지정 슬롯 유형을 사용하도록 슬롯을 업데이트

대화 로그

마이그레이션은 Amazon Lex V2 봇의 대화 로그 설정을 업데이트하지 않습니다.

대화 로그 구성하기

1. https://console.aws.amazon.com/lexv2에서 Amazon Lex V2 콘솔을 엽니다.
2. 봇 목록에서 대화 로그를 구성하고자 하는 봇을 선택합니다.
3. 왼쪽 메뉴에서 별칭을 선택한 다음 목록에서 별칭을 선택합니다.
4. 대화 로그 섹션에서 대화 로그 관리를 선택하여 봇 별칭에 대한 대화 로그를 구성합니다.

메시지 그룹

Amazon Lex V2는 메시지 그룹당 메시지 하나와 대체 메시지 두 개만 지원합니다. Amazon Lex V1 봇의 메시지 그룹당 메시지가 3개 이상인 경우 처음 세 개의 메시지만 마이그레이션됩니다. 메시지 그룹에서 더 많은 메시지를 사용하려면 Lambda 함수를 사용하여 다양한 메시지를 출력하십시오.

프롬프트 및 프레이즈

Amazon Lex V2는 후속 조치, 설명 및 전화 끊기 프롬프트에 다른 메커니즘을 사용합니다.

후속 조치 프롬프트의 경우, 이행 후 컨텍스트 캐리오버를 사용하여 다른 의도로 전환하십시오.

예를 들어, `book_car_fulfilled`이라는 명칭을 갖는, “book_car_fulfilled”라는 출력 컨텍스트를 반환하도록 구성된 렌터카를 예약하려는 의도가 있다고 가정해 보겠습니다. 의도가 충족되면 Amazon Lex는 출력 컨텍스트 변수를 `book_car_fulfilled`로 설정합니다. `book_car_fulfilled`는 활성 컨텍스트이므로 입력 컨텍스트가 포함된 `book_car_fulfilled` 의도는 인식 대상으로 간주됩니다. 단, 사용자의 발언이 해당 의도를 이끌어내려는 시도로 인식되어야 합니다. 영수증을 이메일로 보내거나 예약을 수정하는 등 차량 예약 이후에만 의미가 있는 의도에 이 방법을 사용할 수 있습니다.

Amazon Lex V2는 설명 프롬프트 및 전화 끊기 문구(중단 문구)를 지원하지 않습니다. Amazon Lex V2 봇에는 일치하는 의도가 없는 경우 호출되는 기본 폴백 의도가 포함되어 있습니다. 재시도가 포함된 설명 프롬프트를 보내려면 Lambda 함수를 구성하고 폴백 의도에서 대화 코드 후크를 활성화하십시오. Lambda 함수는 설명 프롬프트를 응답으로 출력하고 세션 속성에 재시도 값을 출력할 수 있습니다. 재시도 값이 최대 재시도 횟수를 초과하는 경우 끊기 문구를 출력하고 대화를 종료할 수 있습니다.

기타 Amazon Lex V1 기능

마이그레이션 도구는 Amazon Lex V1 봇과 기본 의도, 슬롯 유형 및 슬롯의 마이그레이션만 지원합니다. 다른 기능에 대한 설명은 Amazon Lex V2 설명서에서 다음 주제를 참조하세요.

- 봇 별칭: 별칭

- 봇 채널: 메시징 플랫폼에 Amazon Lex V2 봇 배포하기
- 대화 로그 설정: 대화 로그를 사용한 모니터링
- 아마존 켄드라 인덱스: AMAZON.KendraSearchIntent
- Lambda 함수: AWS Lambda 함수 사용
- 태그: 태깅 리소스

Amazon Lex V1에서 Amazon Lex V2로 Lambda 함수 마이그레이션

Amazon Lex V2는 봇의 각 언어에 대해 하나의 Lambda 함수 만 허용합니다. Lambda 함수 및 해당 설정은 런타임 시 사용하는 봇 별칭에 맞게 구성됩니다.

Lambda 함수는 해당 의도에 대해 대화상자 및 이행 코드 후크가 활성화된 경우 해당 언어의 모든 의도에 대해 호출됩니다.

Amazon Lex V2 Lambda 함수는 Amazon Lex V1과 다른 입력 및 출력 메시지 형식을 사용합니다. Lambda 함수 입력 형식의 차이는 다음과 같습니다.

- Amazon Lex V2는 `currentIntent` 및 `alternativeIntents` 구조를 `interpretations` 구조로 대체합니다. 각 해석에는 의도, 해당 의도에 대한 NLU 신뢰도 점수 및 선택적 센티멘트 분석이 포함됩니다.
- Amazon Lex V2는 아마존 Lex V1의 `activeContexts`, `sessionAttributes`의 기능을 통합 `sessionState` 구조로 이전합니다. 이 구조는 시작 요청 ID를 포함하여 대화의 현재 상태에 대한 정보를 제공합니다.
- Amazon Lex V2는 `recentIntentSummaryView`을 반환하지 않습니다. 대신 `sessionState` 구조체의 정보를 사용합니다.
- Amazon Lex V2 입력은 `bot` 속성 내의 `botId` 및 `localeId`를 제공합니다.
- 입력 구조에는 입력 유형(텍스트, 음성 또는 DTMF)에 대한 정보를 제공하는 `inputMode` 속성이 포함되어 있습니다.

Lambda 함수 입력 형식의 차이는 다음과 같습니다.

- Amazon Lex V1의 `activeContexts` 및 `sessionAttributes` 구조는 아마존 Lex V2의 `sessionState` 구조로 대체되었습니다.
- `recentIntentSummaryView`은 출력에 포함되지 않습니다.

- Amazon Lex V1 `dialogAction` 구조는 두 개의 구조로 나뉘는데, `dialogAction` 구조는 `sessionState` 구조의 일부이며, `messages`는 `dialogAction.type`가 `ElicitIntent`일 때 필요합니다. Amazon Lex는 이 구조에서 사용자에게 표시할 메시지를 선택합니다.

Amazon Lex V2 API로 봇을 구축하는 경우, 각 의도에 대한 Lambda 함수 대신 언어별로 봇 별칭당 하나의 Lambda 함수 만 있습니다. 별도의 함수를 계속 사용하려는 경우 각 의도에 대해 별도의 함수를 활성화하는 라우터 함수를 생성할 수 있습니다. 다음은 애플리케이션에 사용하거나 수정할 수 있는 라우터 함수입니다.

```
import os
import json
import boto3

# reuse client connection as global
client = boto3.client('lambda')

def router(event):
    intent_name = event['sessionState']['intent']['name']
    fn_name = os.environ.get(intent_name)
    print(f"Intent: {intent_name} -> Lambda: {fn_name}")
    if (fn_name):
        # invoke lambda and return result
        invoke_response = client.invoke(FunctionName=fn_name, Payload =
 json.dumps(event))
        print(invoke_response)
        payload = json.load(invoke_response['Payload'])
        return payload
    raise Exception('No environment variable for intent: ' + intent_name)

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

업데이트된 필드 목록

다음 표에는 Amazon Lex V2 Lambda 요청 및 응답의 업데이트된 필드에 대한 자세한 정보가 나와 있습니다. 이 테이블을 사용하여 버전 간에 필드를 매핑할 수 있습니다.

요청

Lambda 함수 요청 형식으로 다음 필드가 업데이트되었습니다.

활성 컨텍스트

이제 `activeContexts` 구조가 `sessionState` 구조의 일부입니다.

V1 구조	V2 구조
activeContexts	sessionState.activeContexts
activeContexts[*].timeToLive	sessionState.activeContexts[*].timeToLive
activeContexts[*].timeToLive.timeToLiveInSeconds	sessionState.activeContexts[*].timeToLive.timeToLiveInSeconds
activeContexts[*].timeToLive.turnsToLive	sessionState.activeContexts[*].timeToLive.turnsToLive
activeContexts[*].name	sessionState.activeContexts[*].name
activeContexts[*].parameters	sessionState.activeContexts[*].contextAttributes

대체 의도

인덱스 1부터 N까지의 해석 목록에는 Amazon Lex V2에서 예측한 대체 의도 목록과 해당 신뢰도 점수가 포함되어 있습니다. Amazon Lex V2의 요청 구조에서 `recentIntentSummaryView`가 제거되었습니다. `recentIntentSummaryView`에서 세부 정보를 보려면 GetSession 작업을 사용하십시오.

V1 구조	V2 구조
alternativeIntents	interpretations[1:*]
recentIntentSummaryView	N/A

봇

Amazon Lex V2에서는 봇과 별칭에 식별자가 있습니다. 봇 ID는 코드후크 입력의 일부입니다. 앨리어스 ID는 포함되지만 앨리어스 이름은 포함되지 않습니다. Amazon Lex V2는 동일한 봇에 대해 여러 로케일을 지원하므로 로케일 ID가 포함됩니다.

V1 구조	V2 구조
bot	bot
bot.name	bot.name
N/A	bot.id
bot.alias	N/A
N/A	bot.aliasId
bot.version	bot.version
N/A	bot.localeId

현재 의도

`sessionState.intent` 구조에는 활성 의도의 세부 정보가 포함됩니다. 또한 Amazon Lex V2는 대체 의도를 포함하여 `interpretations` 구조에 있는 모든 의도의 목록을 반환합니다. 해석 목록의 첫 번째 요소는 항상 `sessionState.intent`와 동일합니다.

V1 구조	V2 구조
currentIntent	sessionState.intent OR interpretations[0].intent
currentIntent.name	sessionState.intent.name OR interpretations[0].intent.name
currentIntent.nluConfidenceScore	interpretations[0].nluConfidence.score

다이얼로그 액션

`confirmationStatus` 필드는 이제 `sessionState` 구조의 일부가 되었습니다.

V1 구조	V2 구조
currentIntent.confirmationStatus	sessionState.intent.confirmationState OR interpretations[0].intent.confirmationState
N/A	sessionState.intent.state OR interpretations[*].intent.state

Amazon Kendra

`kendraResponse` 필드는 이제 `sessionState` 및 `interpretations` 구조의 일부가 되었습니다.

V1 구조	V2 구조
kendraResponse	sessionState.intent.kendraResponse OR interpretations[0].intent.kendraResponse

Sentiment

`sentimentResponse` 구조가 새로운 `interpretations` 구조로 이동되었습니다.

V1 구조	V2 구조
sentimentResponse	interpretations[0].sentimentResponse
sentimentResponse.sentimentLabel	interpretations[0].sentimentResponse.sentiment
sentimentResponse.sentimentScore	interpretations[0].sentimentResponse.sentimentScore

슬롯

Amazon Lex V2는 해석된 값, 해석된 값 및 사용자가 말한 원래 값을 포함하는 `sessionState.intent` 구조 내에 단일 `slots` 객체를 제공합니다. Amazon Lex V2는

`slotShape`를 `List`로 설정하고 `values` 목록을 설정하여 다중 값 슬롯도 지원합니다. `value` 필드는 단일 값 슬롯을 지원하며 그 모양은 `Scalar`와 같이 가정합니다.

V1 구조	V2 구조
currentIntent.slots	sessionState.intent.slots OR interpretations[0].intent.slots
currentIntent.slots[*].value	sessionState.intent.slots[*].value.interpretedValue OR interpretations[0].intent.slots[*].value.interpretedValue
N/A	sessionState.intent.slots[*].value.shape OR interpretations[0].intent.slots[*].shape
N/A	sessionState.intent.slots[*].values OR interpretations[0].intent.slots[*].values
currentIntent.slotDetails	sessionState.intent.slots OR interpretations[0].intent.slots
currentIntent.slotDetails[*].resolutions	sessionState.intent.slots[*].resolvedValues OR interpretations[0].intent.slots[*].resolvedValues
currentIntent.slotDetails[*].originalValue	sessionState.intent.slots[*].originalValue OR interpretations[0].intent.slots[*].originalValue

기타

아마존 Lex V2 `sessionId` 필드는 아마존 Lex V1의 `userId` 필드와 동일합니다. 또한 Amazon Lex V2는 발신자의 `inputMode`에게 문자, DTMF 또는 음성과 같은 메시지를 보냅니다.

V1 구조	V2 구조
userId	sessionId
inputTranscript	inputTranscript
invocationSource	invocationSource

V1 구조	V2 구조
outputDialogMode	responseContentType
messageVersion	messageVersion
sessionAttributes	sessionState.sessionAttributes
requestAttributes	requestAttributes
N/A	inputMode
N/A	originatingRequestId

응답

Lambda 함수 응답 메시지 형식에서 다음 필드가 변경되었습니다.

활성 컨텍스트

`activeContexts` 구조가 `sessionState` 구조로 이동되었습니다.

V1 구조	V2 구조
activeContexts	sessionState.activeContexts
activeContexts[*].timeToLive	sessionState.activeContexts[*].timeToLive
activeContexts[*].timeToLive.timeToLiveInSeconds	sessionState.activeContexts[*].timeToLive.timeToLiveInSeconds
activeContexts[*].timeToLive.turnsToLive	sessionState.activeContexts[*].timeToLive.turnsToLive
activeContexts[*].name	sessionState.activeContexts[*].name
activeContexts[*].parameters	sessionState.activeContexts[*].contextAttributes

다이얼로그 액션

`dialogAction` 구조가 `sessionState` 구조로 이동되었습니다. 이제 대화 액션에서 여러 메시지를 지정할 수 있으며, 이제 `genericAttachments` 구조가 `imageResponseCard` 구조입니다.

V1 구조	V2 구조
dialogAction	sessionState.dialogAction
dialogAction.type	sessionState.dialogAction.type
dialogAction.slotToElicit	sessionState.intent.dialogAction.slotToElicit
dialogAction.type.fulfillmentState	sessionState.intent.state
dialogAction.message	messages
dialogAction.message.contentType	messages[*].contentType
dialogAction.message.content	messages[*].content
dialogAction.responseCard	messages[*].imageResponseCard
dialogAction.responseCard.version	N/A
dialogAction.responseCard.contentType	messages[*].contentType
dialogAction.responseCard.genericAttachments	N/A
dialogAction.responseCard.genericAttachments[*].title	messages[*].imageResponseCard.title
dialogAction.responseCard.genericAttachments[*].subTitle	messages[*].imageResponseCard.subtitle
dialogAction.responseCard.genericAttachments[*].imageUrl	messages[*].imageResponseCard.imageUrl
dialogAction.responseCard.genericAttachments[*].buttons	messages[*].imageResponseCard.buttons

V1 구조	V2 구조
dialogAction.responseCard.genericAttachments[*].buttons[*].value	messages[*].imageResponseCard.buttons[*].value
dialogAction.responseCard.genericAttachments[*].buttons[*].text	messages[*].imageResponseCard.buttons[*].text
dialogAction.kendraQueryRequestPayload	dialogAction.kendraQueryRequestPayload
dialogAction.kendraQueryFilterString	dialogAction.kendraQueryFilterString

의도 및 슬롯

`dialogAction` 구조의 일부였던 의도 및 슬롯 필드는 이제 `sessionState` 구조의 일부가 되었습니다.

V1 구조	V2 구조
dialogAction.intentName	sessionState.intent.name
dialogAction.slots	sessionState.intent.slots
dialogAction.slots[*].key	sessionState.intent.slots[*].key
dialogAction.slots[*].value	sessionState.intent.slots[*].value.interpretedValue
N/A	sessionState.intent.slots[*].value.shape
N/A	sessionState.intent.slots[*].values

기타

이제 `sessionAttributes` 구조가 `sessionState` 구조의 일부입니다. `recentIntentSummaryReview` 구조가 제거되었습니다.

V1 구조	V2 구조
sessionAttributes	sessionState.sessionAttributes
recentIntentSummaryView	N/A

Amazon Lex의 보안

의 클라우드 보안 AWS 이 최우선 순위입니다. AWS 고객은 보안에 가장 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 이점을 누릴 수 있습니다.

보안은 AWS 와 사용자 간의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

- 클라우드 보안 - AWS 는 AWS 클라우드에서 서비스를 실행하는 AWS 인프라를 보호할 책임이 있습니다. AWS 또한는 안전하게 사용할 수 있는 서비스를 제공합니다. 서드 파티 감사자는 정기적으로 AWS 규정 준수 프로그램의 일환으로 보안 효과를 테스트하고 검증합니다. Amazon Lex에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 AWS 규정 준수 프로그램 범위 내 서비스를 참조하세요.
- 클라우드의 보안 - 사용자의 책임은 사용하는 AWS 서비스에 따라 결정됩니다. 또한 데이터의 민감도, 조직의 요건 및 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는 Amazon Lex 사용 시 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 주제에서는 보안 및 규정 준수 목적에 맞게 Amazon Lex를 구성하는 방법을 보여줍니다. 또한 Amazon Lex 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스를 사용하는 방법을 배우게 됩니다.

주제

- Amazon Lex의 데이터 보호
- Amazon Lex의 Identity and Access Management
- Amazon Lex에 사용되는 모니터링
- Amazon Lex의 규정 준수 검증
- Amazon Lex의 복원력
- Amazon Lex의 인프라 보안

Amazon Lex의 데이터 보호

Amazon Lex는 문제 해결을 위해 고객 콘텐츠를 수집하고 서비스 개선에 기여합니다. 고객 콘텐츠는 기본적으로 보호됩니다. Amazon Lex API를 사용하여 개별 고객의 콘텐츠를 삭제할 수 있습니다.

Amazon Lex는 4개 유형의 콘텐츠를 저장합니다.

- 봇을 구축하고 훈련시키는 데 사용되는 샘플 표현
- 봇과 상호 작용하는 사용자로부터 얻은 고객 표현
- 사용자가 봇과 상호 작용하는 동안 애플리케이션별 정보를 제공하는 세션 속성
- 봇에 대한 단일 요청에 적용되는 정보를 포함하는 요청 속성

어린이가 사용하도록 설계된 모든 Amazon Lex 봇은 아동 온라인 개인정보 보호법(COPPA)의 적용을 받습니다. 콘솔 또는 Amazon Lex API를 사용하여 `childDirected` 필드를 `true`로 설정함으로써 봇이 COPPA의 적용 대상임을 Amazon Lex에게 알립니다. `childDirected` 필드가 `true`로 설정되면, 사용자 표현은 저장되지 않습니다.

주제

- 유휴 데이터 암호화
- 전송 중 데이터 암호화
- 키 관리

유휴 데이터 암호화

Amazon Lex는 저장하는 사용자 표현을 암호화합니다.

주제

- 샘플 표현
- 고객 표현
- 세션 속성
- 요청 속성

샘플 표현

봇을 개발할 때 각 의도와 슬롯에 대한 샘플 표현을 제공할 수 있습니다. 슬롯에 대한 사용자 정의 값 및 동의어도 제공할 수 있습니다. 이 정보는 저장 시 암호화되며, 봇을 구축하고 사용자 경험을 만드는 데 사용됩니다.

고객 표현

Amazon Lex는 `childDirected` 필드가 `true`로 설정되어 있지 않으면 사용자가 봇에 보내는 표현을 암호화합니다.

`childDirected` 필드가 `true`로 설정되면, 사용자 표현은 저장되지 않습니다.

`childDirected` 필드가 `false`(기본값)로 설정되면 사용자 표현이 암호화되고 GetUtterancesView 작업에 사용하기 위해 15일 동안 저장됩니다. 특정 사용자의 저장된 표현을 삭제하려면 DeleteUtterances 작업을 사용하십시오.

봇이 음성 입력을 수락하면 입력은 무기한 저장됩니다. Amazon Lex는 이를 사용하여 사용자 입력에 응답하기 위한 봇의 기능을 향상시킵니다.

특정 사용자의 저장된 표현을 삭제하려면 DeleteUtterances 작업을 사용하십시오.

세션 속성

세션 속성에는 Amazon Lex와 클라이언트 애플리케이션 간에 전달되는 애플리케이션별 정보가 포함됩니다. Amazon Lex는 봇에 대해 구성된 모든 AWS Lambda 함수에 세션 속성을 전달합니다. Lambda 함수가 세션 속성을 추가 또는 업데이트하는 경우 Amazon Lex는 새로운 정보를 클라이언트 애플리케이션에 다시 전달합니다.

세션 속성은 세션 기간 동안 암호화된 저장소에서 지속됩니다. 마지막 사용자 표현 후 최소 1분, 최대 24시간 동안 활성 상태로 유지되도록 세션을 구성할 수 있습니다. 기본 세션 지속 시간은 5분입니다.

요청 속성

요청 속성은 요청 관련 정보를 포함하고 있고 현재 요청에만 적용됩니다. 클라이언트 애플리케이션은 요청 속성을 사용하여 런타임 시 Amazon Lex에 정보를 보냅니다.

전체 세션에서 유지할 필요가 없는 정보를 전달하려면 요청 속성을 사용합니다. 요청 속성은 요청 사이에 지속되지 않으므로 저장되지 않습니다.

전송 중 데이터 암호화

Amazon Lex는 HTTPS 프로토콜을 사용하여 클라이언트 애플리케이션과 통신합니다. HTTPS 및 AWS 서명을 사용하여 Amazon Polly와 같은 다른 서비스와 애플리케이션을 대신하여 통신 AWS Lambda 합니다.

키 관리

Amazon Lex는 내부 키로 콘텐츠의 무단 사용을 방지합니다.

Amazon Lex의 Identity and Access Management

AWS Identity and Access Management (IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어할 수 AWS 서비스 있도록 도와주는 입니다. IAM 관리자는 어떤 사용자가 Amazon Lex 리소스를 사용할 수 있도록 인증(로그인)되고 권한 부여(권한 있음)될 수 있는지 제어합니다. IAM은 추가 비용 없이 사용할 수 AWS 서비스 있는 입니다.

주제

- 대상
- ID를 통한 인증
- 정책을 사용하여 액세스 관리
- Amazon Lex에서 IAM을 사용하는 방법
- Amazon Lex의 자격 증명 기반 정책 예제
- AWS Amazon Lex에 대한 관리형 정책
- Amazon Lex에 서비스 연결 역할 사용
- Amazon Lex 자격 증명 및 액세스 문제 해결

대상

사용 방법 AWS Identity and Access Management (IAM)은 Amazon Lex에서 수행하는 작업에 따라 다릅니다.

서비스 사용자 – Amazon Lex 서비스를 사용하여 작업을 수행하는 경우, 필요한 보안 인증 및 권한을 관리자가 제공합니다. 더 많은 Amazon Lex 기능을 사용하여 작업을 수행한다면 추가 권한이 필요할 수 있습니다. 액세스 권한 관리 방법을 이해하면 관리자에게 올바른 권한을 요청하는 데 도움이 됩니다. Amazon Lex의 기능에 액세스할 수 없다면 Amazon Lex 자격 증명 및 액세스 문제 해결을 참조하세요.

서비스 관리자 – 회사에서 Amazon Lex 리소스를 책임지고 있다면 Amazon Lex에 대한 모든 액세스 권한이 있을 것입니다. 서비스 관리자는 서비스 사용자가 액세스해야 하는 Amazon Lex 기능과 리소스를 결정합니다. 그런 다음 IAM 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경해야 합니다. 이 페이지의 정보를 검토하여 IAM의 기본 개념을 이해하세요. 회사가 Amazon Lex에서 IAM을 사용하는 방법에 대해 자세히 알아보려면 Amazon Lex에서 IAM을 사용하는 방법을 참조하세요.

IAM 관리자 – IAM 관리자라면 Amazon Lex에 대한 액세스 관리 정책 작성 방법을 자세히 알고 싶을 수도 있습니다. IAM에서 사용할 수 있는 Amazon Lex 자격 증명 기반 정책 예제를 확인하려면 Amazon Lex의 자격 증명 기반 정책 예제을 참조하세요.

ID를 통한 인증

인증은 AWS 자격 증명으로에 로그인하는 방법입니다. IAM 사용자 또는 AWS 계정 루트 사용자 IAM 역할을 수임하여 로 인증(로그인 AWS)되어야 합니다.

자격 증명 소스를 통해 제공된 자격 증명을 사용하여 페더레이션 자격 증명 AWS 으로에 로그인할 수 있습니다. AWS IAM Identity Center (IAM Identity Center) 사용자, 회사의 Single Sign-On 인증 및 Google 또는 Facebook 자격 증명은 페더레이션 자격 증명의 예입니다. 페더레이션형 ID로 로그인할 때 관리자가 이전에 IAM 역할을 사용하여 ID 페더레이션을 설정했습니다. 페더레이션을 사용하여 AWS 에 액세스하면 간접적으로 역할을 수임하게 됩니다.

사용자 유형에 따라 AWS Management Console 또는 AWS 액세스 포털에 로그인할 수 있습니다. 에 로그인하는 방법에 대한 자세한 내용은 AWS 로그인 사용 설명서의에 로그인하는 방법을 AWS참조하세요. AWS 계정

AWS 프로그래밍 방식으로에 액세스하는 경우는 자격 증명을 사용하여 요청에 암호화 방식으로 서명할 수 있는 소프트웨어 개발 키트(SDK)와 명령줄 인터페이스(CLI)를 AWS 제공합니다. AWS 도구를 사용하지 않는 경우 직접 요청에 서명해야 합니다. 권장 방법을 사용하여 요청에 직접 서명하는 자세한 방법은 IAM 사용 설명서에서 API 요청용AWS Signature Version 4를 참조하세요.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어는 다중 인증(MFA)을 사용하여 계정의 보안을 강화할 것을 AWS 권장합니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서에서 다중 인증 및 IAM 사용 설명서에서 IAM의AWS 다중 인증을 참조하세요.

AWS 계정 루트 사용자

를 생성할 때 계정의 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있는 하나의 로그인 자격 증명으로 AWS 계정시작합니다. 이 자격 증명을 AWS 계정 테루트 사용자라고 하며 계정을 생성하는 데 사용한 이메일 주소와 암호로 로그인하여 액세스합니다. 일상적인 작업에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 자격 증명을 보호하고 루트 사용자만 수행할 수 있는 작업을 수행하는 데 사용합니다. 루트 사용자로 로그인해야 하는 전체 작업 목록은 IAM 사용 설명서의 루트 사용자 보안 인증이 필요한 작업을 참조하세요.

페더레이션 자격 증명

가장 좋은 방법은 관리자 액세스가 필요한 사용자를 포함한 인간 사용자가 자격 증명 공급자와의 페더레이션을 사용하여 임시 자격 증명을 사용하여 AWS 서비스 에 액세스하도록 요구하는 것입니다.

페더레이션 자격 증명은 엔터프라이즈 사용자 디렉터리, 웹 자격 증명 공급자, AWS Directory Service, Identity Center 디렉터리 또는 자격 증명 소스를 통해 제공된 자격 증명을 사용하여 AWS 서비스 에 액

세스하는 모든 사용자의 사용자입니다. 페더레이션 자격 증명에 액세스할 때 역할을 AWS 계정수임하고 역할은 임시 자격 증명을 제공합니다.

중앙 집중식 액세스 관리를 위해 AWS IAM Identity Center을(를) 사용하는 것이 좋습니다. IAM Identity Center에서 사용자 및 그룹을 생성하거나 모든 및 애플리케이션에서 사용할 수 있도록 자체 자격 증명 소스의 사용자 AWS 계정 및 그룹 집합에 연결하고 동기화할 수 있습니다. IAM Identity Center에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서에서 IAM Identity Center란 무엇인가요?를 참조하세요.

IAM 사용자 및 그룹

IAM 사용자는 단일 사용자 또는 애플리케이션에 대한 특정 권한이 AWS 계정 있는 내 자격 증명입니다. 가능하면 암호 및 액세스 키와 같은 장기 자격 증명이 있는 IAM 사용자를 생성하는 대신 임시 자격 증명을 사용하는 것이 좋습니다. 하지만 IAM 사용자의 장기 자격 증명이 필요한 특정 사용 사례가 있는 경우, 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 장기 보안 인증이 필요한 사용 사례의 경우, 정기적으로 액세스 키 교체를 참조하세요.

IAM 그룹은 IAM 사용자 컬렉션을 지정하는 자격 증명입니다. 사용자는 그룹으로 로그인할 수 없습니다. 그룹을 사용하여 여러 사용자의 권한을 한 번에 지정할 수 있습니다. 그룹을 사용하면 대규모 사용자 집합의 권한을 더 쉽게 관리할 수 있습니다. 예를 들어, IAMAdmins라는 그룹이 있고 이 그룹에 IAM 리소스를 관리할 권한을 부여할 수 있습니다.

사용자는 역할과 다릅니다. 사용자는 한 사람 또는 애플리케이션과 고유하게 연결되지만, 역할은 해당 역할이 필요한 사람이라면 누구나 수임할 수 있습니다. 사용자는 영구적인 장기 자격 증명을 가지고 있지만, 역할은 임시 보안 인증만 제공합니다. 자세한 내용은 IAM 사용 설명서에서 IAM 사용자 사용 사례를 참조하세요.

IAM 역할

IAM 역할은 특정 권한이 AWS 계정 있는 내 자격 증명입니다. IAM 사용자와 유사하지만, 특정 개인과 연결되지 않습니다. 에서 IAM 역할을 일시적으로 수임하려면 사용자에서 IAM 역할(콘솔)로 전환할 AWS Management Console수 있습니다. https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-console.html 또는 AWS API 작업을 호출하거나 사용자 지정 URL을 AWS CLI 사용하여 역할을 수임할 수 있습니다. 역할 사용 방법에 대한 자세한 내용은 IAM 사용 설명서의 역할 수임 방법을 참조하세요.

임시 보안 인증이 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- 페더레이션 사용자 액세스 - 페더레이션 ID에 권한을 부여하려면 역할을 생성하고 해당 역할의 권한을 정의합니다. 페더레이션 ID가 인증되면 역할이 연결되고 역할에 정의된 권한이 부여됩니다. 페

더레이션 관련 역할에 대한 자세한 내용은 IAM 사용 설명서의 Create a role for a third-party identity provider (federation)를 참조하세요. IAM Identity Center를 사용하는 경우, 권한 집합을 구성합니다. 인증 후 ID가 액세스할 수 있는 항목을 제어하기 위해 IAM Identity Center는 권한 집합을 IAM의 역할과 연관짓습니다. 권한 집합에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서의 권한 집합을 참조하세요.

- 임시 IAM 사용자 권한 - IAM 사용자 또는 역할은 IAM 역할을 수임하여 특정 작업에 대한 다양한 권한을 임시로 받을 수 있습니다.
- 교차 계정 액세스 - IAM 역할을 사용하여 다른 계정의 사용자(신뢰할 수 있는 보안 주체)가 내 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 계정 간 액세스를 부여하는 기본적인 방법입니다. 그러나 일부 에서는 (역할을 프록시로 사용하는 대신) 정책을 리소스에 직접 연결할 AWS 서비스 수 있습니다. 교차 계정 액세스에 대한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 교차 계정 리소스 액세스를 참조하세요.
- 교차 서비스 액세스 - 일부는 다른에서 기능을 AWS 서비스 사용합니다 AWS 서비스. 예를 들어, 서비스에서 호출하면 일반적으로 해당 서비스는 Amazon EC2에서 애플리케이션을 실행하거나 Amazon S3에 객체를 저장합니다. 서비스는 직접적으로 호출하는 위탁자의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 작업을 수행할 수 있습니다.
  - 전달 액세스 세션(FAS) - IAM 사용자 또는 역할을 사용하여에서 작업을 수행하는 경우 AWS보안 주체로 간주됩니다. 일부 서비스를 사용하는 경우, 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는를 호출하는 보안 주체의 권한을 다운스트림 서비스에 AWS 서비스 대한 요청과 AWS 서비스함께 사용합니다. FAS 요청은 서비스가 다른 AWS 서비스 또는 리소스와의 상호 작용을 완료해야 하는 요청을 수신할 때만 수행됩니다. 이 경우, 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.
  - 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하기 위해 맡는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 Create a role to delegate permissions to an AWS 서비스를 참조하세요.
  - 서비스 연결 역할 - 서비스 연결 역할은에 연결된 서비스 역할의 한 유형입니다 AWS 서비스. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은에 나타나 AWS 계정 며 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.
- Amazon EC2에서 실행되는 애플리케이션 - IAM 역할을 사용하여 EC2 인스턴스에서 실행되고 AWS CLI 또는 AWS API 요청을 수행하는 애플리케이션의 임시 자격 증명을 관리할 수 있습니다. 이는 EC2 인스턴스 내에 액세스 키를 저장할 때 권장되는 방법입니다. EC2 인스턴스에 AWS 역할을 할당하고 모든 애플리케이션에서 사용할 수 있도록 하려면 인스턴스에 연결된 인스턴스 프로파일을 생성합니다. 인스턴스 프로필에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행되는 프로그

램이 임시 보안 인증을 얻을 수 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여를 참조하세요.

정책을 사용하여 액세스 관리

정책을 AWS 생성하고 자격 증명 또는 리소스에 연결하여 AWS 에서 액세스를 제어합니다. 정책은 자격 증명 또는 리소스와 연결된 AWS 경우 권한을 정의하는의 객체입니다.는 보안 주체(사용자, 루트 사용자 또는 역할 세션)가 요청할 때 이러한 정책을 AWS 평가합니다. 정책에서 권한은 요청이 허용되거나 거부되는 지를 결정합니다. 대부분의 정책은에 JSON 문서 AWS 로 저장됩니다. JSON 정책 문서의 구조와 콘텐츠에 대한 자세한 정보는 IAM 사용 설명서의 JSON 정책 개요를 참조하세요.

관리자는 AWS JSON 정책을 사용하여 누가 무엇을 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

기본적으로, 사용자 및 역할에는 어떠한 권한도 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 수임할 수 있습니다.

IAM 정책은 작업을 수행하기 위해 사용하는 방법과 상관없이 작업에 대한 권한을 정의합니다. 예를 들어, `iam:GetRole` 작업을 허용하는 정책이 있다고 가정합니다. 해당 정책이 있는 사용자는 AWS Management Console AWS CLI, 또는 API에서 역할 정보를 가져올 수 있습니다 AWS .

ID 기반 정책

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자 및 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

ID 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 사용자, 그룹 또는 역할에 직접 포함됩니다. 관리형 정책은의 여러 사용자, 그룹 및 역할에 연결할 수 있는 독립 실행형 정책입니다 AWS 계정. 관리형 정책에는 AWS 관리형 정책 및 고객 관리형 정책이 포함됩니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책 및 인라인 정책 중에서 선택을 참조하세요.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예제는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자

는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 위탁자가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 위탁자를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는가 포함될 수 있습니다 AWS 서비스.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 리소스 기반 정책에서는 IAM의 AWS 관리형 정책을 사용할 수 없습니다.

액세스 제어 목록(ACL)

액세스 제어 목록(ACL)은 어떤 보안 주체(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

Amazon S3 AWS WAF및 Amazon VPC는 ACLs. ACL에 관한 자세한 내용은 Amazon Simple Storage Service 개발자 가이드의 액세스 제어 목록(ACL) 개요를 참조하세요.

기타 정책 타입

AWS 는 덜 일반적인 추가 정책 유형을 지원합니다. 이러한 정책 타입은 더 일반적인 정책 유형에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 – 권한 경계는 ID 기반 정책에 따라 IAM 엔터티(IAM 사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 기능입니다. 개체에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 객체의 ID 기반 정책과 그 권한 경계의 교집합입니다. `Principal` 필드에서 사용자나 역할을 지정하는 리소스 기반 정책은 권한 경계를 통해 제한되지 않습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 권한 경계에 대한 자세한 정보는 IAM 사용 설명서의 IAM 엔티티에 대한 권한 경계를 참조하세요.
- 서비스 제어 정책(SCPs) - SCPs는의 조직 또는 조직 단위(OU)에 대한 최대 권한을 지정하는 JSON 정책입니다 AWS Organizations. AWS Organizations 는 비즈니스가 소유 AWS 계정 한 여러를 그룹화하고 중앙에서 관리하기 위한 서비스입니다. 조직에서 모든 기능을 활성화할 경우, 서비스 제어 정책(SCP)을 임의의 또는 모든 계정에 적용할 수 있습니다. SCP는 각각을 포함하여 멤버 계정의 엔터티에 대한 권한을 제한합니다 AWS 계정 루트 사용자. 조직 및 SCP에 대한 자세한 내용은 AWS Organizations 사용 설명서에서 Service control policies을 참조하세요.
- 리소스 제어 정책(RCP) - RCP는 소유한 각 리소스에 연결된 IAM 정책을 업데이트하지 않고 계정의 리소스에 대해 사용 가능한 최대 권한을 설정하는 데 사용할 수 있는 JSON 정책입니다. RCP는 멤버 계정의 리소스에 대한 권한을 제한하며 조직에 속하는지 여부에 AWS 계정 루트 사용자관계없이를 포함한 자격 증명에 대한 유효 권한에 영향을 미칠 수 있습니다. RCP를 AWS 서비스 지원하는 목록

을 포함하여 조직 및 RCPs에 대한 자세한 내용은 AWS Organizations 사용 설명서의 리소스 제어 정책(RCPs)을 참조하세요.

- 세션 정책 – 세션 정책은 역할 또는 페더레이션 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 결과적으로 얻는 세션의 권한은 사용자 또는 역할의 ID 기반 정책의 교차와 세션 정책입니다. 또한 권한을 리소스 기반 정책에서 가져올 수도 있습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 자세한 정보는 IAM 사용 설명서의 세션 정책을 참조하세요.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 가 여러 정책 유형이 관련될 때 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하세요.

Amazon Lex에서 IAM을 사용하는 방법

IAM을 사용하여 Amazon Lex에 대한 액세스를 관리하기 전에 Amazon Lex에서 사용할 수 있는 IAM 기능을 알아봅니다.

Amazon Lex에서 사용할 수 있는 IAM 기능

IAM 기능	Amazon Lex 지원
ID 기반 정책	예
리소스 기반 정책	아니요
정책 작업	예
정책 리소스	예
정책 조건 키(서비스별)	예
ACLs	아니요
ABAC(정책 내 태그)	부분
임시 자격 증명	예

IAM 기능	Amazon Lex 지원
보안 주체 권한	예
서비스 역할	예
서비스 연결 역할	예

Amazon Lex 및 기타 AWS 서비스가 대부분의 IAM 기능과 작동하는 방식을 전체적으로 알아보려면 IAM 사용 설명서의 AWS IAM으로 작업하는 서비스를 참조하세요.

Amazon Lex의 자격 증명 기반 정책

ID 기반 정책 지원: 예

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자 및 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

IAM ID 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스뿐 아니라 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. ID 기반 정책에서는 위탁자가 연결된 사용자 또는 역할에 적용되므로 위탁자를 지정할 수 없습니다. JSON 정책에서 사용하는 모든 요소에 대해 알아보려면 IAM 사용자 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

Amazon Lex의 자격 증명 기반 정책 예제

Amazon Lex 자격 증명 기반 정책 예제를 보려면 Amazon Lex의 자격 증명 기반 정책 예제을 참조하세요.

Amazon Lex 내 리소스 기반 정책

리소스 기반 정책 지원: 아니요

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예제는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 위탁자가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니

다. 리소스 기반 정책에서 위탁자를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는가 포함될 수 있습니다 AWS 서비스.

교차 계정 액세스를 활성화하려는 경우, 전체 계정이나 다른 계정의 IAM 개체를 리소스 기반 정책의 위탁자로 지정할 수 있습니다. 리소스 기반 정책에 크로스 계정 보안 주체를 추가하는 것은 트러스트 관계 설정의 절반밖에 되지 않는다는 것을 유념하세요. 보안 주체와 리소스가 다른 경우 신뢰할 수 AWS 계정있는 계정의 IAM 관리자는 보안 주체 엔터티(사용자 또는 역할)에게 리소스에 액세스할 수 있는 권한도 부여해야 합니다. 엔터티에 ID 기반 정책을 연결하여 권한을 부여합니다. 하지만 리소스 기반 정책이 동일 계정의 위탁자에 액세스를 부여하는 경우, 추가 자격 증명 기반 정책이 필요하지 않습니다. 자세한 내용은 IAM 사용 설명서의 교차 계정 리소스 액세스를 참조하세요.

Amazon Lex에 대한 정책 작업

정책 작업 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇을 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 위탁자가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

JSON 정책의 `Action` 요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 작업을 설명합니다. 정책 작업은 일반적으로 연결된 AWS API 작업과 이름이 동일합니다. 일치하는 API 작업이 없는 권한 전용 작업 같은 몇 가지 예외도 있습니다. 정책에서 여러 작업이 필요한 몇 가지 작업도 있습니다. 이러한 추가 작업을 일컬어 종속 작업이라고 합니다.

연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함하세요.

Amazon Lex 작업 목록을 보려면 서비스 승인 참조의 Amazon Lex에서 정의한 작업을 참조하세요.

Amazon Lex에 대한 정책 작업은 작업 앞에 다음 접두사를 사용합니다.

```
lex
```

단일 문에서 여러 작업을 지정하려면 다음과 같이 쉼표로 구분합니다.

```
"Action": [
      "lex:action1",
      "lex:action2"
         ]
```

와일드카드(*)를 사용하여 여러 작업을 지정할 수 있습니다. 예를 들어, `Describe` 라는 단어로 시작하는 모든 작업을 지정하려면 다음 작업을 포함합니다.

```
"Action": "lex:Describe*"
```

Amazon Lex에 대한 정책 리소스

정책 리소스 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇을 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

`Resource` JSON 정책 요소는 작업이 적용되는 하나 이상의 객체를 지정합니다. 문에는 `Resource`또는 `NotResource`요소가 반드시 추가되어야 합니다. 모범 사례에 따라 Amazon 리소스 이름(ARN)을 사용하여 리소스를 지정합니다. 리소스 수준 권한이라고 하는 특정 리소스 유형을 지원하는 작업에 대해 이를 수행할 수 있습니다.

작업 나열과 같이 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"
```

Amazon Lex 봇 리소스 ARN의 형식은 다음과 같습니다.

```
arn:aws:lex:${Region}:${Account}:bot:${Bot-Name}
```

ARN 형식에 대한 자세한 내용은 Amazon 리소스 이름(ARNs) 및 AWS 서비스 네임스페이스를 참조하세요.

예를 들어, 설명문에 `OrderFlowers` 봇을 지정하려면 다음 ARN을 사용합니다.

```
"Resource": "arn:aws:lex:us-east-2:123456789012:bot:OrderFlowers"
```

특정 계정에 속하는 모든 봇을 지정하려면 와일드카드(*)를 사용합니다.

```
"Resource": "arn:aws:lex:us-east-2:123456789012:bot:*"
```

리소스 생성 작업과 같은 일부 Amazon Lex 작업은 특정 리소스에서 수행할 수 없습니다. 이러한 경우 와일드카드(*)를 사용해야 합니다.

```
"Resource": "*"
```

Amazon Lex 리소스 유형 및 해당 ARNs 목록을 보려면 서비스 승인 참조에서 Amazon Lex에서 정의한 리소스를 참조하세요. 각 리소스의 ARN을 지정할 수 있는 작업을 알아보려면 Amazon Lex에서 정의한 작업을 참조하세요.

Amazon Lex에 사용되는 정책 조건 키

서비스별 정책 조건 키 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇을 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

`Condition` 요소(또는 `Condition` 블록)를 사용하면 정책이 발효되는 조건을 지정할 수 있습니다. `Condition` 요소는 옵션입니다. 같거나 작음과 같은 조건 연산자를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 생성할 수 있습니다.

한 문에서 여러 `Condition` 요소를 지정하거나 단일 `Condition` 요소에서 여러 키를 지정하는 경우, AWS 는 논리적 `AND` 작업을 사용하여 평가합니다. 단일 조건 키에 여러 값을 지정하는 경우는 논리적 `OR` 작업을 사용하여 조건을 AWS 평가합니다. 문의 권한을 부여하기 전에 모든 조건을 충족해야 합니다.

조건을 지정할 때 자리 표시자 변수를 사용할 수도 있습니다. 예를 들어, IAM 사용자에게 IAM 사용자 이름으로 태그가 지정된 경우에만 리소스에 액세스할 수 있는 권한을 부여할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 IAM 정책 요소: 변수 및 태그를 참조하세요.

AWS 는 전역 조건 키와 서비스별 조건 키를 지원합니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 AWS 전역 조건 컨텍스트 키를 참조하세요.

Amazon Lex 조건 키 목록을 보려면 서비스 승인 참조의 Amazon Lex에 사용되는 조건 키를 참조하세요. 조건 키를 사용할 수 있는 작업과 리소스를 알아보려면 Amazon Lex에서 정의한 작업을 참조하세요.

다음 표는 Amazon Lex 리소스에 적용되는 Amazon Lex 조건 키를 나열합니다. 이러한 키를 IAM 권한 정책의 `Condition` 요소에 포함할 수 있습니다.

Amazon Lex의 ACL

ACL 지원: 아니요

액세스 제어 목록(ACL)은 어떤 위탁자(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

Amazon Lex의 ABAC

ABAC 지원(정책의 태그): 부분적

속성 기반 액세스 제어(ABAC)는 속성에 근거하여 권한을 정의하는 권한 부여 전략입니다. 에서는 AWS이러한 속성을 태그라고 합니다. IAM 엔터티(사용자 또는 역할)와 많은 AWS 리소스에 태그를 연결할 수 있습니다. ABAC의 첫 번째 단계로 개체 및 리소스에 태그를 지정합니다. 그런 다음 위탁자의 태그가 액세스하려는 리소스의 태그와 일치할 때 작업을 허용하도록 ABAC 정책을 설계합니다.

ABAC는 빠르게 성장하는 환경에서 유용하며 정책 관리가 번거로운 상황에 도움이 됩니다.

태그에 근거하여 액세스를 제어하려면 `aws:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`* 또는 `aws:TagKeys` 조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다.

서비스가 모든 리소스 유형에 대해 세 가지 조건 키를 모두 지원하는 경우, 값은 서비스에 대해 예입니다. 서비스가 일부 리소스 유형에 대해서만 세 가지 조건 키를 모두 지원하는 경우, 값은 부분적입니다.

ABAC에 대한 자세한 내용은 IAM 사용 설명서의 ABAC 권한 부여를 통한 권한 정의를 참조하세요. ABAC 설정 단계가 포함된 자습서를 보려면 IAM 사용 설명서의 속성 기반 액세스 제어(ABAC) 사용을 참조하세요.

권한 부여를 위해 특정 유형의 Amazon Lex 리소스에 태그를 연결할 수 있습니다. 태그를 기반으로 액세스를 제어하려면 `lex:ResourceTag/${TagKey}`, `aws:RequestTag/${TagKey}` 또는 `aws:TagKeys` 조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다.

Amazon Lex 리소스 태그 지정에 대한 자세한 내용은 Amazon Lex 리소스 태그 지정을 참조하세요.

리소스의 태그를 기반으로 리소스에 대한 액세스를 제한하는 자격 증명 기반 정책의 예시는 태그를 사용하여 리소스 액세스에서 확인할 수 있습니다.

다음 표에는 태그 기반 액세스 제어에 필요한 작업과 해당 리소스 유형이 나열되어 있습니다. 각 작업은 해당 리소스 유형과 연결된 태그를 기반으로 권한이 부여됩니다.

작업	리소스 유형	조건 키	참고 사항
CreateBotVersion	bot	`lex:ResourceTag`	
DeleteBot	bot	`lex:ResourceTag`	
DeleteBotAlias	별칭	`lex:ResourceTag`	
DeleteBotChannelAssociation	채널	`lex:ResourceTag`	
DeleteBotVersion	bot	`lex:ResourceTag`	
DeleteSession	봇 또는 별칭	`lex:ResourceTag`	별칭이 `$LATEST`로 설정된 경우 봇과 연결된 태그를 사용합니다. 다른 별칭과 함께 사용되는 경우 지정된 별칭과 연결된 태그를 사용합니다.
DeleteUtterances	bot	`lex:ResourceTag`	
GetBot	봇 또는 별칭	`lex:ResourceTag`	`versionOrAlias` 가 `$LATEST` 또는 숫자 버전으로 설정된 경우 봇과 연결된 태그를 사용합니다. 별칭과 함께 사용되는 경우 지정된 별칭과 연결된 태그를 사용합니다.
GetBotAlias	별칭	`lex:ResourceTag`	
GetBotChannelAssociation	채널	`lex:ResourceTag`	
GetBotChannelAssociations	채널	`lex:ResourceTag`	별칭이 "-"로 설정된 경우 봇과 연결된 태그를

작업	리소스 유형	조건 키	참고 사항
			사용합니다. 봇 별칭이 지정된 경우 지정된 별칭과 연결된 태그를 사용합니다.
GetBotVersions	bot	`lex:ResourceTag`	
GetExport	bot	`lex:ResourceTag`	
GetSession	봇 또는 별칭	`lex:ResourceTag`	별칭이 `$LATEST`로 설정된 경우 봇과 연결된 태그를 사용합니다. 다른 별칭과 함께 사용되는 경우 지정된 별칭과 연결된 태그를 사용합니다.
GetUtterancesView	bot	`lex:ResourceTag`	
ListTagsForResource	봇, 별칭 또는 채널	`lex:ResourceTag`	
PostContent	봇 또는 별칭	`lex:ResourceTag`	별칭이 `$LATEST`로 설정된 경우 봇과 연결된 태그를 사용합니다. 다른 별칭과 함께 사용되는 경우 지정된 별칭과 연결된 태그를 사용합니다.
PostText	봇 또는 별칭	`lex:ResourceTag`	별칭이 `$LATEST`로 설정된 경우 봇과 연결된 태그를 사용합니다. 다른 별칭과 함께 사용되는 경우 지정된 별칭과 연결된 태그를 사용합니다.

작업	리소스 유형	조건 키	참고 사항
PutBot	bot	`lex:ResourceTag`, `aws:RequestTag`, `aws:TagKeys`	
PutBotAlias	별칭	`lex:ResourceTag`, `aws:RequestTag`, `aws:TagKeys`	
PutSession	봇 또는 별칭	`lex:ResourceTag`	별칭이 `$LATEST`로 설정된 경우 봇과 연결된 태그를 사용합니다. 다른 별칭과 함께 사용되는 경우 지정된 별칭과 연결된 태그를 사용합니다.
StartImport	bot	`lex:ResourceTag`	`PutBot` 작업에 대한 액세스 정책에 의존합니다. `StartImport` 작업과 관련된 태그 및 권한은 무시됩니다.
TagResource	봇, 별칭 또는 채널	`lex:ResourceTag`, `aws:RequestTag`, `aws:TagKeys`	
UntagResource	봇, 별칭 또는 채널	`lex:ResourceTag`, `aws:RequestTag`, `aws:TagKeys`	

Amazon Lex에서 임시 보안 인증 사용

임시 자격 증명 지원: 예

임시 자격 증명을 사용하여 로그인할 때 일부가 작동하지 AWS 서비스 않습니다. 임시 자격 증명으로 AWS 서비스 작업하는을 비롯한 자세한 내용은 AWS 서비스 IAM 사용 설명서의 IAM으로 작업하는를 참조하세요.

사용자 이름과 암호를 제외한 방법을 AWS Management Console 사용하여에 로그인하는 경우 임시 자격 증명을 사용합니다. 예를 들어 회사의 SSO(Single Sign-On) 링크를 AWS 사용하여에 액세스하면 해당 프로세스가 임시 자격 증명을 자동으로 생성합니다. 또한 콘솔에 사용자로 로그인한 다음 역할을 전환할 때 임시 자격 증명을 자동으로 생성합니다. 역할 전환에 대한 자세한 내용은 IAM 사용 설명서의 사용자에서 IAM 역할로 전환(콘솔)을 참조하세요.

AWS CLI 또는 AWS API를 사용하여 임시 자격 증명을 수동으로 생성할 수 있습니다. 그런 다음 이러한 임시 자격 증명을 사용하여 장기 액세스 키를 사용하는 대신 동적으로 임시 자격 증명을 생성하는 access AWS. AWS recommends에 액세스할 수 있습니다. 자세한 정보는 IAM의 임시 보안 자격 증명을 참조하세요.

임시 보안 인증을 사용하여 연동을 통해 로그인하거나, IAM 역할을 맡거나, 교차 계정 역할을 맡을 수 있습니다. AssumeRole 또는 GetFederationToken과 같은 AWS STS API 작업을 호출하여 임시 보안 자격 증명을 얻습니다.

Amazon Lex에 대한 교차 서비스 보안 주체 권한

전달 액세스 세션(FAS) 지원: 예

IAM 사용자 또는 역할을 사용하여에서 작업을 수행하는 경우 AWS보안 주체로 간주됩니다. 일부 서비스를 사용하는 경우, 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는를 호출하는 보안 주체의 권한을 다운스트림 서비스에 AWS 서비스 대한 요청과 AWS 서비스함께 사용합니다. FAS 요청은 서비스가 다른 AWS 서비스 또는 리소스와의 상호 작용을 완료해야 하는 요청을 수신할 때만 수행됩니다. 이 경우, 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.

Amazon Lex의 서비스 역할

서비스 역할 지원: 예

서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하는 것으로 가정하는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 Create a role to delegate permissions to an AWS 서비스를 참조하세요.

> **Warning**
> 서비스 역할에 대한 권한을 변경하면 Amazon Lex 기능이 중단될 수 있습니다. Amazon Lex에서 관련 지침을 제공하는 경우에만 서비스 역할을 편집합니다.

Amazon Lex에서 IAM 역할 선택

Amazon Lex는 서비스 연결 역할을 사용하여 Amazon Comprehend와 Amazon Polly를 호출합니다. AWS Lambda 함수에 대한 리소스 수준 권한을 사용하여 함수를 호출합니다.

태그 지정을 활성화하려면 IAM 역할을 제공해야 합니다. 자세한 내용은 대화 로그에 대한 IAM 역할 및 정책 생성을 참조하세요.

Amazon Lex의 서비스 연결 역할

서비스 링크 역할 지원: 예

서비스 연결 역할은에 연결된 서비스 역할의 한 유형입니다 AWS 서비스. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은에 표시 AWS 계정 되며 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.

Amazon Lex 서비스 연결 역할 생성 또는 관리에 대한 자세한 정보는 Amazon Lex에 서비스 연결 역할 사용을 참조하세요.

Amazon Lex의 자격 증명 기반 정책 예제

기본적으로 사용자 및 역할은 Amazon Lex 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console, AWS Command Line Interface (AWS CLI) 또는 AWS API를 사용하여 작업을 수행할 수 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 맡을 수 있습니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 IAM 정책 생성(콘솔)을 참조하세요.

각 리소스 유형에 대한 ARN 형식을 포함하여 Amazon Lex에서 정의한 작업 및 리소스 유형에 대한 자세한 내용은 서비스 권한 부여 참조에서 Amazon Lex에 대한 작업, 리소스 및 조건 키를 참조하세요.

주제

- 정책 모범 사례
- Amazon Lex 콘솔 사용
- 사용자가 자신의 고유한 권한을 볼 수 있도록 허용
- 모든 아마존 Lex 봇 삭제
- 사용자가 봇을 Amazon Lex V2 API로 마이그레이션하도록 허용
- 태그를 사용하여 리소스 액세스

정책 모범 사례

ID 기반 정책에 따라 계정에서 사용자가 Amazon Lex 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부가 결정됩니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따릅니다.

- AWS 관리형 정책을 시작하고 최소 권한으로 전환 - 사용자 및 워크로드에 권한 부여를 시작하려면 많은 일반적인 사용 사례에 대한 권한을 부여하는 AWS 관리형 정책을 사용합니다. 에서 사용할 수 있습니다 AWS 계정. 사용 사례에 맞는 AWS 고객 관리형 정책을 정의하여 권한을 추가로 줄이는 것이 좋습니다. 자세한 정보는 IAM 사용 설명서의 AWS 관리형 정책 또는 AWS 직무에 대한 관리형 정책을 참조하세요.
- 최소 권한 적용 – IAM 정책을 사용하여 권한을 설정하는 경우, 작업을 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. IAM을 사용하여 권한을 적용하는 방법에 대한 자세한 정보는 IAM 사용 설명서에 있는 IAM의 정책 및 권한을 참조하세요.
- IAM 정책의 조건을 사용하여 액세스 추가 제한 – 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어, SSL을 사용하여 모든 요청을 전송해야 한다고 지정하는 정책 조건을 작성할 수 있습니다. 조건을 사용하여 AWS 서비스와 같은 특정를 통해 사용되는 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다 AWS CloudFormation. 자세한 정보는 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.
- IAM Access Analyzer를 통해 IAM 정책을 확인하여 안전하고 기능적인 권한 보장 - IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 새로운 및 기존 정책을 확인합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 추천을 제공하여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 내용은 IAM 사용 설명서의 IAM Access Analyzer에서 정책 검증을 참조하세요.
- 다중 인증(MFA) 필요 -에서 IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 위해 MFA를 AWS 계정켭니다. API 작업을 직접 호출할 때 MFA가 필요하면 정책에 MFA 조건을 추가합니다. 자세한 내용은 IAM 사용 설명서의 MFA를 통한 보안 API 액세스를 참조하세요.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례를 참조하세요.

Amazon Lex 콘솔 사용

Amazon Lex 콘솔에 액세스하려면 최소한의 권한 집합이 있어야 합니다. 이러한 권한을 통해의 Amazon Lex 리소스에 대한 세부 정보를 나열하고 볼 수 있어야 합니다 AWS 계정. 최소 필수 권한보다 더 제한적인 ID 기반 정책을 생성하는 경우, 콘솔이 해당 정책에 연결된 엔티티(사용자 또는 역할)에 대해 의도대로 작동하지 않습니다.

AWS CLI 또는 AWS API에만 호출하는 사용자에 대해 최소 콘솔 권한을 허용할 필요는 없습니다. 대신 수행하려는 API 작업과 일치하는 작업에만 액세스할 수 있도록 합니다.

AWS 는에서 생성하고 관리하는 독립 실행형 IAM 정책을 제공하여 많은 일반적인 사용 사례를 처리합니다 AWS. 이러한 정책을 AWS 관리형 정책이라고 합니다. 직접 정책을 작성하는 경우보다는 AWS 관리형 정책을 사용하면 사용자, 그룹 및 역할에 적절한 권한을 할당하는 것이 더욱 쉽습니다. 자세한 내용은 IAM 사용자 설명서의 AWS 관리형 정책을 참조하십시오.

계정의 그룹 및 역할에 연결할 수 있는 다음 AWS 관리형 정책은 Amazon Lex에 고유합니다.

- AmazonLexReadOnly — Amazon Lex 리소스에 대한 읽기 전용 액세스 권한을 부여합니다.
- AmazonLexRunBotsOnly — Amazon Lex 대화형 봇을 실행할 수 있는 액세스 권한을 부여합니다.
- AmazonLexFullAccess - Amazon Lex 리소스 모두를 생성하고, 읽고, 업데이트하고, 삭제하고, 태그를 지정하고 실행할 모든 액세스 권한을 부여합니다. Amazon Lex 의도를 사용하여 이름이 `AmazonLex`로 시작하는 Lambda 함수를 연결할 수 있는 권한도 부여합니다.

> **Note**
>
> IAM 콘솔에 로그인하고 이 콘솔에서 특정 정책을 검색하여 이러한 권한 정책을 검토할 수 있습니다.

AmazonLexFullAccess 정책은 사용자에게 `KendraSearchIntent` 의도를 사용하여 Amazon Kendra 인덱스를 쿼리할 권한을 부여하지 않습니다. 인덱스를 쿼리하려면 정책에 권한을 추가해야 합니다. 필요한 권한에 대해서는 Amazon Kendra 검색에 사용되는 IAM 정책을 참조하세요.

고유의 사용자 지정 IAM 정책을 생성하여 Amazon Lex API 작업에 대한 권한을 허용할 수도 있습니다. 해당 권한이 필요한 IAM 역할 또는 그룹에 이러한 사용자 지정 정책을 연결할 수 있습니다.

Amazon Lex에 대한 AWS 관리형 정책에 대한 자세한 내용은 AWS Amazon Lex에 대한 관리형 정책을 참조하세요.

사용자가 자신의 고유한 권한을 볼 수 있도록 허용

이 예제는 IAM 사용자가 자신의 사용자 ID에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 AWS CLI 또는 AWS API를 사용하여 프로그래밍 방식으로이 작업을 완료할 수 있는 권한이 포함되어 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

모든 아마존 Lex 봇 삭제

이 예제 정책은 AWS 계정의 사용자에게 계정의 봇을 삭제할 수 있는 권한을 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "lex:DeleteBot"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

사용자가 봇을 Amazon Lex V2 API로 마이그레이션하도록 허용

다음 IAM 권한 정책은 사용자가 Amazon Lex에서 Amazon Lex V2 API로 봇 마이그레이션을 시작하고 마이그레이션 목록 및 진행 상황을 확인할 수 있도록 허용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "startMigration",
            "Effect": "Allow",
            "Action": "lex:StartMigration",
            "Resource": "arn:aws:lex:<Region>:<123456789012>:bot:*"
        },
        {
            "Sid": "passRole",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::<123456789012>:role/<v2 bot role>"
        },
        {
            "Sid": "allowOperations",
            "Effect": "Allow",
            "Action": [
```

```
                "lex:CreateBot",
                "lex:CreateIntent",
                "lex:UpdateSlot",
                "lex:DescribeBotLocale",
                "lex:UpdateBotAlias",
                "lex:CreateSlotType",
                "lex:DeleteBotLocale",
                "lex:DescribeBot",
                "lex:UpdateBotLocale",
                "lex:CreateSlot",
                "lex:DeleteSlot",
                "lex:UpdateBot",
                "lex:DeleteSlotType",
                "lex:DescribeBotAlias",
                "lex:CreateBotLocale",
                "lex:DeleteIntent",
                "lex:StartImport",
                "lex:UpdateSlotType",
                "lex:UpdateIntent",
                "lex:DescribeImport",
                "lex:CreateCustomVocabulary",
                "lex:UpdateCustomVocabulary",
                "lex:DeleteCustomVocabulary",
                "lex:DescribeCustomVocabulary",
                "lex:DescribeCustomVocabularyMetadata"
            ],
            "Resource": [
                "arn:aws:lex:<Region>:<123456789012>:bot/*",
                "arn:aws:lex:<Region>:<123456789012>:bot-alias/*/*"
            ]
        },
        {
            "Sid": "showBots",
            "Effect": "Allow",
            "Action": [
                "lex:CreateUploadUrl",
                "lex:ListBots"
            ],
            "Resource": "*"
        },
        {
            "Sid": "showMigrations",
            "Effect": "Allow",
            "Action": [
```

```
                "lex:GetMigration",
                "lex:GetMigrations"
            ],
            "Resource": "*"
        }
    ]
}
```

태그를 사용하여 리소스 액세스

이 예제 정책은 **Department** 키와 **Support** 값으로 태그가 지정된 모든 리소스에 대해 PostText 작업을 사용할 수 있는 권한을 AWS 계정의 사용자 또는 역할에 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "lex:PostText",
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "lex:ResourceTag/Department": "Support"
                }
            }
        }
    ]
}
```

AWS Amazon Lex에 대한 관리형 정책

AWS 관리형 정책은에서 생성하고 관리하는 독립 실행형 정책입니다 AWS. AWS 관리형 정책은 사용자, 그룹 및 역할에 권한 할당을 시작할 수 있도록 많은 일반적인 사용 사례에 대한 권한을 제공하도록 설계되었습니다.

AWS 관리형 정책은 모든 AWS 고객이 사용할 수 있으므로 특정 사용 사례에 대해 최소 권한 권한을 부여하지 않을 수 있습니다. 사용 사례에 고유한 고객 관리형 정책을 정의하여 권한을 줄이는 것이 좋습니다.

AWS 관리형 정책에 정의된 권한은 변경할 수 없습니다. 가 AWS 관리형 정책에 정의된 권한을 AWS 업데이트하면 정책이 연결된 모든 보안 주체 자격 증명(사용자, 그룹 및 역할)에 영향을 미칩니다. AWS 는 새 AWS 서비스 가 시작되거나 기존 서비스에 새 API 작업을 사용할 수 있게 되면 AWS 관리형 정책을 업데이트할 가능성이 높습니다.

자세한 내용은 IAM 사용 설명서의 AWS 관리형 정책을 참조하세요.

AWS 관리형 정책: AmazonLexReadOnly

`AmazonLexReadOnly` 정책을 IAM 보안 인증에 연결할 수 있습니다.

이 정책은 사용자가 Amazon Lex 및 Amazon Lex V2 모델 구축 서비스의 모든 작업을 볼 수 있도록 허용하는 읽기 전용 권한을 부여합니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `lex` – 모델 구축 서비스의 Amazon Lex 및 Amazon Lex V2 리소스에 대한 읽기 전용 액세스입니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "lex:GetBot",
                "lex:GetBotAlias",
                "lex:GetBotAliases",
                "lex:GetBots",
                "lex:GetBotChannelAssociation",
                "lex:GetBotChannelAssociations",
                "lex:GetBotVersions",
                "lex:GetBuiltinIntent",
                "lex:GetBuiltinIntents",
                "lex:GetBuiltinSlotTypes",
                "lex:GetIntent",
                "lex:GetIntents",
```

```
                "lex:GetIntentVersions",
                "lex:GetSlotType",
                "lex:GetSlotTypes",
                "lex:GetSlotTypeVersions",
                "lex:GetUtterancesView",
                "lex:DescribeBot",
                "lex:DescribeBotAlias",
                "lex:DescribeBotChannel",
                "lex:DescribeBotLocale",
                "lex:DescribeBotVersion",
                "lex:DescribeExport",
                "lex:DescribeImport",
                "lex:DescribeIntent",
                "lex:DescribeResourcePolicy",
                "lex:DescribeSlot",
                "lex:DescribeSlotType",
                "lex:ListBots",
                "lex:ListBotLocales",
                "lex:ListBotAliases",
                "lex:ListBotChannels",
                "lex:ListBotVersions",
                "lex:ListBuiltInIntents",
                "lex:ListBuiltInSlotTypes",
                "lex:ListExports",
                "lex:ListImports",
                "lex:ListIntents",
                "lex:ListSlots",
                "lex:ListSlotTypes",
                "lex:ListTagsForResource"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 관리형 정책: AmazonLexRunBotsOnly

`AmazonLexRunBotsOnly` 정책을 IAM 보안 인증에 연결할 수 있습니다.

이 정책은 Amazon Lex 및 Amazon Lex V2 대화형 봇 실행에 대한 액세스를 허용하는 읽기 전용 권한을 부여합니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `lex` – Amazon Lex 및 Amazon Lex V2 런타임의 모든 작업에 대한 읽기 전용 액세스입니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "lex:PostContent",
                "lex:PostText",
                "lex:PutSession",
                "lex:GetSession",
                "lex:DeleteSession",
                "lex:RecognizeText",
                "lex:RecognizeUtterance",
                "lex:StartConversation"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 관리형 정책: AmazonLexFullAccess

`AmazonLexFullAccess` 정책을 IAM 보안 인증에 연결할 수 있습니다.

이 정책은 사용자에게 Amazon Lex 및 Amazon Lex V2 리소스를 생성, 읽기, 업데이트 및 삭제하고 Amazon Lex 및 Amazon Lex V2 대화형 봇을 실행할 수 있는 관리 권한을 부여합니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `lex` – 보안 주체에게 Amazon Lex 및 Amazon Lex V2 모델 구축 및 런타임 서비스의 모든 작업에 대한 읽기 및 쓰기 액세스를 허용합니다.
- `cloudwatch` – 보안 주체가 Amazon CloudWatch 지표와 경보를 볼 수 있도록 허용합니다.
- `iam` – 보안 주체가 서비스 연결 역할을 생성 및 삭제하고, 역할을 전달하고, 정책을 역할에 연결 및 분리할 수 있습니다. 권한은 Amazon Lex 운영의 경우 "lex.amazonaws.com"으로, Amazon Lex V2 운영의 경우 "lexv2.amazonaws.com"으로 제한됩니다.

- `kendra` – 보안 주체가 Amazon Kendra 색인을 나열하도록 허용합니다.
- `kms` - 보안 주체가 AWS KMS 키와 별칭을 설명할 수 있습니다.
- `lambda` - 보안 주체가 AWS Lambda 함수를 나열하고 Lambda 함수에 연결된 권한을 관리할 수 있습니다.
- `polly` – 보안 주체가 Amazon Polly 음성을 설명하고 음성을 합성하도록 허용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:DescribeAlarmsForMetric",
                "kms:DescribeKey",
                "kms:ListAliases",
                "lambda:GetPolicy",
                "lambda:ListFunctions",
                "lex:*",
                "polly:DescribeVoices",
                "polly:SynthesizeSpeech",
                "kendra:ListIndices",
                "iam:ListRoles",
                "s3:ListAllMyBuckets",
                "logs:DescribeLogGroups",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:AddPermission",
                "lambda:RemovePermission"
            ],
            "Resource": "arn:aws:lambda:*:*:function:AmazonLex*",
            "Condition": {
                "StringEquals": {
```

```
                    "lambda:Principal": "lex.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots",
                "arn:aws:iam::*:role/aws-service-role/channels.lex.amazonaws.com/
AWSServiceRoleForLexChannels",
                "arn:aws:iam::*:role/aws-service-role/lexv2.amazonaws.com/
AWSServiceRoleForLexV2Bots*",
                "arn:aws:iam::*:role/aws-service-role/channels.lexv2.amazonaws.com/
AWSServiceRoleForLexV2Channels*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "lex.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/channels.lex.amazonaws.com/
AWSServiceRoleForLexChannels"
            ],
```

```
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "channels.lex.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/lexv2.amazonaws.com/
AWSServiceRoleForLexV2Bots*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "lexv2.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/channels.lexv2.amazonaws.com/
AWSServiceRoleForLexV2Channels*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "channels.lexv2.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:DeleteServiceLinkedRole",
                "iam:GetServiceLinkedRoleDeletionStatus"
            ],
            "Resource": [
```

```
                "arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots",
                "arn:aws:iam::*:role/aws-service-role/channels.lex.amazonaws.com/
AWSServiceRoleForLexChannels",
                "arn:aws:iam::*:role/aws-service-role/lexv2.amazonaws.com/
AWSServiceRoleForLexV2Bots*",
                "arn:aws:iam::*:role/aws-service-role/channels.lexv2.amazonaws.com/
AWSServiceRoleForLexV2Channels*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "lex.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/lexv2.amazonaws.com/
AWSServiceRoleForLexV2Bots*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "lexv2.amazonaws.com"
                    ]
                }
            }
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/channels.lexv2.amazonaws.com/
AWSServiceRoleForLexV2Channels*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "channels.lexv2.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

AWS 관리형 정책에 대한 Amazon Lex 업데이트

이 서비스가 이러한 변경 사항을 추적하기 시작한 이후 Amazon Lex의 AWS 관리형 정책 업데이트에 대한 세부 정보를 봅니다. 이 페이지의 변경 사항에 대한 자동 알림을 받아보려면 Amazon Lex Amazon Lex 문서 기록 페이지에서 RSS 피드를 구독하세요.

변경 사항	설명	날짜
AmazonLexFullAccess – 기존 정책에 대한 업데이트	Amazon Lex 에는 Amazon Lex V2 모델 빌드 서비스 작업에 대한 읽기 전용 액세스를 허용하는 새로운 권한이 추가되었습니다.	2021년 8월 18일
AmazonLexReadOnly – 기존 정책에 대한 업데이트	Amazon Lex에는 Amazon Lex V2 모델 빌드 서비스 작업에 대한 읽기 전용 액세스를 허용하	2021년 8월 18일

변경 사항	설명	날짜
	는 새로운 권한이 추가되었습니다.	
AmazonLexRunBotsOnly – 기존 정책에 대한 업데이트	Amazon Lex 에는 Amazon Lex V2 런타임 서비스 작업에 대한 읽기 전용 액세스를 허용하는 새로운 권한이 추가되었습니다.	2021년 8월 18일
Amazon Lex가 변경 내용 추적 시작	Amazon Lex가 AWS 관리형 정책에 대한 변경 사항 추적을 시작했습니다.	2021년 8월 18일

Amazon Lex에 서비스 연결 역할 사용

Amazon Lex는 AWS Identity and Access Management (IAM) 서비스 연결 역할을 사용합니다. 서비스 연결 역할은 Amazon Lex에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 Amazon Lex에서 사전 정의하며 서비스가 사용자를 대신하여 다른 AWS 서비스를 호출하는 데 필요한 모든 권한을 포함합니다.

필요한 권한을 수동으로 추가할 필요가 없으므로 서비스 연결 역할로 Amazon Lex를 더 쉽게 설정할 수 있습니다. Amazon Lex에서 서비스 연결 역할의 권한을 정의하므로 다르게 정의되지 않은 한, Amazon Lex에만 해당 역할을 수임할 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함되며 이 권한 정책은 다른 IAM 엔티티에 연결할 수 없습니다.

먼저 관련 리소스를 삭제해야만 서비스 연결 역할을 삭제할 수 있습니다. 이렇게 하면 리소스에 대한 액세스 권한을 실수로 삭제할 수 없기 때문에 Amazon Lex 리소스가 보호됩니다.

Amazon Lex에 대한 서비스 연결 역할 권한

Amazon Lex는 두 가지 서비스 연결 역할을 사용합니다.

- AWSServiceRoleForLexBots – Amazon Lex는 이 서비스 연결 역할을 사용하여 Amazon Polly를 간접 호출하여 봇에 대한 음성 응답을 합성하고, 감정 분석을 위해 Amazon Comprehend를 호출하고, 선택적으로 인덱스 검색을 위해 Amazon Kendra를 호출합니다.
- AWSServiceRoleForLexChannels – Amazon Lex는 채널을 관리할 때 이 서비스 연결 역할을 사용하여 봇에 텍스트를 게시합니다.

IAM 엔터티(사용자, 그룹, 역할 등)가 서비스 링크 역할을 생성하고 편집하거나 삭제할 수 있도록 권한을 구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 권한을 참조하세요.

Amazon Lex에 대한 서비스 연결 역할 생성

서비스 링크 역할은 수동으로 생성할 필요가 없습니다. 에서 봇, 봇 채널 또는 Amazon Kendra 검색 의도를 생성하면 AWS Management Console Amazon Lex가 서비스 연결 역할을 생성합니다.

이 서비스 연결 역할을 삭제했다가 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. 새 봇, 채널 연결 또는 Amazon Kendra 검색 의도를 생성하면 Amazon Lex가 서비스 연결 역할을 다시 생성합니다.

또한 AWS CLI 를 사용하여 AWSServiceRoleForLexBots 사용 사례로 서비스 연결 역할을 생성할 수 있습니다. 에서 Amazon Lex 서비스 이름이 인 서비스 연결 역할을 AWS CLI 생성합니다`lex.amazonaws.com`. 자세한 내용은 1단계: 서비스 연결 역할 생성(AWS CLI) 단원을 참조하십시오. 이 서비스 연결 역할을 삭제하면 동일한 프로세스를 사용하여 역할을 다시 생성할 수 있습니다.

Amazon Lex에 대한 서비스 연결 역할 편집

Amazon Lex에서는 Amazon Lex 서비스 연결 역할을 편집하는 것을 허용하지 않습니다. 서비스 링크 역할을 생성한 후에는 다양한 개체가 역할을 참조할 수 있기 때문에 역할 이름을 변경할 수 없습니다. 하지만 IAM을 사용하여 역할의 설명을 편집할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 편집을 참조하세요.

Amazon Lex에 대한 서비스 연결 역할 삭제

서비스 연결 역할이 필요한 기능 또는 서비스가 더 이상 필요 없는 경우에는 해당 역할을 삭제하는 것이 좋습니다. 따라서 적극적으로 모니터링하거나 유지하지 않는 미사용 엔터티가 없도록 합니다. 단, 서비스 링크 역할에 대한 리소스를 먼저 정리해야 수동으로 삭제할 수 있습니다.

> **Note**
>
> 리소스를 삭제하려고 할 때 Amazon Lex 서비스가 역할을 사용 중이면 삭제에 실패할 수 있습니다. 이 문제가 발생하면 몇 분 기다렸다가 작업을 다시 시도하세요.

서비스 연결 역할에서 사용되는 Amazon Lex 리소스를 삭제하려면

1. 사용 중인 모든 봇 채널을 삭제합니다.
2. 계정의 모든 봇을 삭제합니다.

IAM을 사용하여 수동으로 서비스 연결 역할을 삭제하려면 다음을 수행하세요.

IAM 콘솔, AWS CLI또는 AWS API를 사용하여 Amazon Lex 서비스 연결 역할을 삭제합니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 삭제 섹션을 참조하십시오.

Amazon Lex 서비스 연결 역할에 대해 지원되는 리전

Amazon Lex는 서비스가 제공되는 모든 리전에서 서비스 연결 역할 사용을 지원합니다. 자세한 내용을 알아보려면 Amazon Lex 엔드포인트 및 할당량을 참조하세요.

Amazon Lex 자격 증명 및 액세스 문제 해결

다음 정보를 사용하여 Amazon Lex 및 IAM으로 작업할 때 발생할 수 있는 일반적인 문제를 진단하고 수정할 수 있습니다.

주제

- Amazon Lex에서 작업을 수행할 권한이 없음
- iam:PassRole을 수행하도록 인증되지 않음
- 내 외부의 사람이 내 Amazon Lex 리소스 AWS 계정 에 액세스하도록 허용하고 싶습니다.

Amazon Lex에서 작업을 수행할 권한이 없음

작업을 수행할 권한이 없다는 오류가 수신되면, 작업을 수행할 수 있도록 정책을 업데이트해야 합니다.

다음의 예제 오류는 `mateojackson` IAM 사용자가 콘솔을 사용하여 가상 *`my-example-widget`* 리소스에 대한 세부 정보를 보려고 하지만 가상 `lex:`*`GetWidget`* 권한이 없을 때 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 lex:GetWidget on resource: my-example-widget
```

이 경우, `lex:`*`GetWidget`* 작업을 사용하여 *`my-example-widget`* 리소스에 액세스할 수 있도록 `mateojackson` 사용자 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

iam:PassRole을 수행하도록 인증되지 않음

`iam:PassRole` 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 Amazon Lex에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

일부 AWS 서비스 에서는 새 서비스 역할 또는 서비스 연결 역할을 생성하는 대신 기존 역할을 해당 서비스에 전달할 수 있습니다. 이렇게 하려면 사용자가 서비스에 역할을 전달할 수 있는 권한을 가지고 있어야 합니다.

다음 예제 오류는 `marymajor`라는 IAM 사용자가 콘솔을 사용하여 Amazon Lex에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 있어야 합니다. Mary는 서비스에 역할을 전달할 수 있는 권한을 가지고 있지 않습니다.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

이 경우, Mary가 `iam:PassRole` 작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

내 외부의 사람이 내 Amazon Lex 리소스 AWS 계정 에 액세스하도록 허용하고 싶습니다.

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제어 목록(ACL)을 지원하는 서비스의 경우, 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세히 알아보려면 다음을 참조하세요.

- Amazon Lex에서 이러한 기능을 지원하는지 여부를 알아보려면 Amazon Lex에서 IAM을 사용하는 방법을 참조하세요.
- 소유 AWS 계정 한의 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 소유 AWS 계정 한 다른의 IAM 사용자에게 액세스 권한 제공을 참조하세요.
- 타사에 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 타사 AWS 계정 소유에 대한 액세스 권한 제공을 AWS 계정참조하세요.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 외부에서 인증된 사용자에게 액세스 권한 제공(ID 페더레이션)을 참조하세요.
- 크로스 계정 액세스에 대한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 크로스 계정 리소스 액세스를 참조하세요.

Amazon Lex에 사용되는 모니터링

모니터링은 챗봇의 신뢰성, 가용성 및 성능을 유지하는 데 중요합니다. 이 주제에서는 Amazon CloudWatch Logs 및를 사용하여 Amazon Lex AWS CloudTrail 를 모니터링하는 방법과 Amazon Lex 런타임 및 채널 연결 지표에 대해 설명합니다.

주제

- Amazon CloudWatch 를 사용한 Amazon Lex 모니터링
- AWS CloudTrail 로그를 사용하여 Amazon Lex API 호출 모니터링

Amazon CloudWatch 를 사용한 Amazon Lex 모니터링

Amazon Lex 봇의 상태를 추적하려면 Amazon CloudWatch 를 사용하십시오. CloudWatch를 사용하면 계정에 대한 개별 Amazon Lex 작업 또는 전역 Amazon Lex 작업에 대한 지표를 얻을 수 있습니다. 또한 하나 이상의 지표가 정의한 임계값을 초과하는 경우 알리도록 CloudWatch 경보를 설정할 수도 있습니다. 예를 들어 특정 기간 동안 봇에 대한 요청 횟수를 모니터링하거나, 성공적인 요청의 지연 시간을 보거나, 오류가 임계값을 초과한 경우 경보를 실행할 수 있습니다.

Amazon Lex의 CloudWatch 지표

Amazon Lex 작업에 대한 지표를 얻으려면 다음 정보를 지정해야 합니다.

- 지표 차원. 차원은 지표를 식별하는 데 사용하는 이름-값 페어 집합입니다. Amazon Lex에는 세 가지 차원이 있습니다.
  - `BotAlias, BotName, Operation`
  - `BotAlias, BotName, InputMode, Operation`
  - `BotName, BotVersion, InputMode, Operation`
- `MissedUtteranceCount` 또는 `RuntimeRequestCount`와 같은 지표 이름

AWS CLI, 또는 CloudWatch API를 사용하여 Amazon Lex에 대한 지표 AWS Management Console를 가져올 수 있습니다. Amazon AWS 소프트웨어 개발 키트(SDK) 또는 CloudWatch API 도구 중 하나를 통해 CloudWatch API를 사용할 수 있습니다. Amazon Lex 콘솔에는 CloudWatch API의 원시 데이터를 기초로 하는 그래프가 표시됩니다.

CloudWatch를 사용하여 Amazon Lex를 모니터링하려면 적절한 권한이 있어야 합니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 Amazon CloudWatch에 대한 인증 및 액세스 제어를 참조하세요.

Amazon Lex 지표 보기

Amazon Lex 콘솔 또는 CloudWatch 콘솔을 사용하여 Amazon Lex 지표를 볼 수 있습니다.

지표 보기(Amazon Lex 콘솔)

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lex/ Amazon Lex 콘솔을 엽니다.
2. 봇 목록에서 확인할 지표 중 하나를 선택합니다.
3. 모니터링을 선택합니다. 지표가 그래프로 표시됩니다.

지표 보기(CloudWatch 콘솔)

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/cloudwatch/ CloudWatch 콘솔을 엽니다.
2. Metrics, All Metrics, AWS/Lex를 차례로 선택합니다.
3. 차원과 지표 이름을 선택한 다음 그래프에 추가를 선택합니다.
4. 날짜 범위 값을 선택합니다. 선택한 날짜 범위에 대한 지표 개수가 그래프에 표시됩니다.

알림 생성

CloudWatch 경보는 지정한 기간 동안 단일 지표를 감시하고, Amazon Simple Notification Service (Amazon SNS) 주제 또는 Auto Scaling 정책에 알림 보내기와 같은 하나 이상의 작업을 수행합니다. 이러한 작업은 지정한 여러 기간 동안 지정된 임계값에 따른 지표의 값을 기반으로 합니다. 경보로 인해 상태가 변경되면 Amazon SNS 메시지를 전송하는 CloudWatch 경보를 생성할 수 있습니다.

CloudWatch 경보는 상태가 변경되어 지정한 기간 동안 지속되는 경우에만 작업을 간접 호출합니다.

경보 설정

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/cloudwatch/ CloudWatch 콘솔을 엽니다.
2. 알람를 선택한 다음 알람 생성을 선택합니다.
3. AWS/Lex Metrics를 선택한 후 지표를 선택합니다.
4. 시간 범위에서 모니터링할 시간 범위를 선택한 후, 다음를 선택합니다.
5. 이름 및 설명을 입력합니다.
6. Whenever에서 >=를 선택하고 최대 값을 입력합니다.

7. 경보 상태에 도달하면 CloudWatch가 이메일을 보내기를 원한다면, 작업 섹션에서 경보가 발생할 경우 항상에 대해 상태가 ALARM입니다를 선택합니다. 알림 보내기 대상에서 메일 발송 목록을 선택하거나 새 목록을 선택하여 새 목록을 만듭니다.
8. 알람 미리보기 섹션에서 경보를 미리 볼 수 있습니다. 경보가 만족스러우면 경보 생성을 선택합니다.

Amazon Lex 런타임 의 CloudWatch 측정치

다음 표에서는 Amazon Lex 런타임 지표를 설명합니다.

지표	설명
`KendraIndexAccessError`	Amazon Lex가 Amazon Kendra 인덱스에 액세스할 수 없는 횟수입니다. `PostContent` 작업에 대한 유효한 차원(`Text` 또는 `Speech` `InputMode` 사용): • `BotName, BotAlias, Operation, InputMode` `PostText` 작업에 대한 유효 차원: • `BotName, BotAlias, Operation` 단위: 수
`KendraLatency`	Amazon Kendra가 `AMAZON.KendraSearchIntent` 의 요청에 응답하는 데 걸리는 시간입니다. `PostContent` 작업에 대한 유효한 차원(`Text` 또는 `Speech` `InputMode` 사용): • `BotName, BotVersion, Operation, InputMode` • `BotName, BotAlias, Operation, InputMode` `PostText` 작업에 대한 유효한 차원:

지표	설명
	• BotName, BotVersion, Operation • BotName, BotAlias, Operation 단위: 밀리초
KendraSuccess	AMAZON.KendraSearchIntent 에서 Amazon Kendra 인덱스로 성공적으로 보낸 요청 수입니다. PostContent 작업에 대한 유효한 차원(Text 또는 SpeechInputMode 사용): • BotName, BotVersion, Operation, InputMode • BotName, BotAlias, Operation, InputMode PostText 작업에 대한 유효한 차원: • BotName, BotVersion, Operation • BotName, BotAlias, Operation 단위: 수
KendraSystemErrors	Amazon Lex가 Amazon Kendra 인덱스에 액세스할 수 없는 횟수입니다. PostContent 작업에 대한 유효한 차원(Text 또는 SpeechInputMode 사용): • BotName, BotAlias, Operation, InputMode PostText 작업에 대한 유효 차원: • BotName, BotAlias, Operation 단위: 수

지표	설명
KendraThrottledEvents	Amazon Kendra가 AMAZON.KendraSearchIntent 의 요청을 제한한 횟수입니다. PostContent 작업에 대한 유효한 차원(Text 또는 Speech InputMode 사용): • BotName, BotAlias, Operation, InputMode PostText 작업에 대한 유효 차원: • BotName, BotAlias, Operation 단위: 수
MissedUtteranceCount	지정한 기간에 인식되지 않은 표현 수입니다. PostContent 작업에 대한 유효한 차원(Text 또는 SpeechInputMode 사용): • BotName, BotVersion, Operation, InputMode • BotName, BotAlias, Operation, InputMode PostText 작업에 대한 유효한 차원: • BotName, BotVersion, Operation • BotName, BotAlias, Operation

지표	설명
`RuntimeConcurrency`	지정된 기간 동안 동시 연결 수. `RuntimeConcurrency` 은 `StatisticSet` 로 보고됩니다. `PostContent` 작업에 대한 유효한 차원(`Text` 또는 `Speech` `InputMode` 사용): • Operation, BotName, BotVersion, InputMode • Operation, BotName, BotAlias, InputMode 작업에 대한 유효 차원: • Operation, BotName, BotVersion • Operation, BotName, BotAlias 단위: 수
`RuntimeInvalidLambdaResponses`	지정된 기간 동안 유효하지 않은 AWS Lambda (Lambda) 응답 수입니다. `PostContent` 작업에 대한 유효한 차원(`Text` 또는 `Speech` `InputMode` 사용): • `BotName, BotAlias, Operation, InputMode` `PostText` 작업에 대한 유효 차원: • `BotName, BotAlias, Operation`

지표	설명
RuntimeLambdaErrors	지정된 기간에 발생한 런타임 오류 수입니다. PostContent 작업에 대한 유효한 차원(Text 또는 Speech InputMode 사용): • BotName, BotAlias, Operation, InputMode PostText 작업에 대한 유효 차원: • BotName, BotAlias, Operation
RuntimePollyErrors	지정된 기간 동안 유효하지 않은 Amazon Polly 응답 수입니다. PostContent 작업에 대한 유효한 차원(Text 또는 Speech InputMode 사용): • BotName, BotAlias, Operation, InputMode PostText 작업에 대한 유효 차원: • BotName, BotAlias, Operation
RuntimeRequestCount	지정된 기간의 런타임 요청 수입니다. PostContent 작업에 대한 유효한 차원(Text 또는 Speech InputMode 사용): • BotName, BotVersion, Operation, InputMode • BotName, BotAlias, Operation, InputMode PostText 작업에 대한 유효한 차원: • BotName, BotVersion, Operation • BotName, BotAlias, Operation 단위: 수

지표	설명
`RuntimeSucessfulRequestLatency` **Important** 이 지표는 `RuntimeSucessfulRequestLatency` 이며, `RuntimeSuccessfulRequestLatency` 가 아닙니다.	요청 시간과 응답이 다시 전달된 시간 사이의 성공한 요청에 대한 지연 시간입니다. `PostContent` 작업에 대한 유효한 차원(`Text` 또는 `Speech` `InputMode` 사용): • `BotName, BotVersion, Operation, InputMode` • `BotName, BotAlias, Operation, InputMode` `PostText` 작업에 대한 유효한 차원: • `BotName, BotVersion, Operation` • `BotName, BotAlias, Operation` 단위: 밀리초
`RuntimeSystemErrors`	지정된 기간에 발생한 시스템 오류 수입니다. 시스템 오류의 응답 코드 범위는 500~599입니다. `PostContent` 작업에 대한 유효한 차원(`Text` 또는 `Speech` `InputMode` 사용): • `BotName, BotAlias, Operation, InputMode` `PostText` 작업에 대한 유효 차원: • `BotName, BotAlias, Operation` 단위: 수

지표	설명
`RuntimeThrottledEvents`	제한된 요청 수. Amazon Lex는 계정에 대해 설정된 초당 트랜잭션 한도 이상의 요청이 수신되면 요청을 제한합니다. 계정에 대해 설정된 한도가 자주 초과되면 한도 증가를 요청할 수 있습니다. 증가를 요청하려면 AWS 서비스 한도를 참조하십시오. `PostContent` 작업에 대한 유효한 차원(`Text` 또는 `Speech` `InputMode` 사용): • BotName, BotAlias, Operation, InputMode `PostText` 작업에 대한 유효 차원: • `BotName, BotAlias, Operation` 단위: 수
`RuntimeUserErrors`	지정된 기간에 발생한 사용자 오류 수입니다. 사용자 오류의 응답 코드 범위는 400~499입니다. `PostContent` 작업에 대한 유효한 차원(`Text` 또는 `Speech` `InputMode` 사용): • `BotName, BotAlias, Operation, InputMode` `PostText` 작업에 대한 유효 차원: • `BotName, BotAlias, Operation` 단위: 수

Amazon Lex 런타임 지표는 `AWS/Lex` 네임스페이스를 사용하며, 다음 차원의 지표를 제공합니다. 지표는 CloudWatch 콘솔에서 차원별로 그룹화할 수 있습니다.

차원	설명
BotName, BotAlias, Operation, InputMode	봇 별칭, 봇 이름, 작업(PostContent), 텍스트 입력 또는 음성 입력별로 지표를 그룹화합니다.
BotName, BotVersion, Operation, InputMode	봇 이름, 봇 버전, 작업(PostContent)과 텍스트 입력 또는 음성 입력별로 지표를 그룹화합니다.
BotName, BotVersion, Operation	봇 이름, 봇 버전 및 작업(PostText)별로 지표를 그룹화합니다.
BotName, BotAlias, Operation	봇 이름, 봇 별칭 및 작업(PostText)별로 지표를 그룹화합니다.

Amazon Lex 채널 연결에 사용되는 CloudWatch 지표

채널 연결은 Amazon Lex 와 메시징 채널(예: Facebook) 간의 연결입니다. 다음 표에서는 Amazon Lex 채널 연결 지표를 설명합니다.

지표	설명
BotChannelAuthErrors	지정한 기간에 메시징 채널에서 반환한 인증 오류 수입니다. 인증 오류는 채널 생성 중 제공된 비밀 토큰이 유효하지 않거나 만료되었음을 나타냅니다.
BotChannelConfigurationErrors	지정된 기간에 발생한 구성 오류 수입니다. 구성 오류는 채널에 대한 하나 이상의 구성 항목이 유효하지 않음을 나타냅니다.
BotChannelInboundThrottledEvents	지정된 기간에 메시징 채널이 보낸 메시지를 Amazon Lex가 제한한 수입니다.
BotChannelOutboundThrottledEvents	지정된 기간에 Amazon Lex에서 메시징 채널로 전송되는 아웃바운드 이벤트가 제한된 수입니다.
BotChannelRequestCount	지정된 기간에 채널에 대한 요청 수입니다.

지표	설명
BotChannelResponseCardErrors	지정된 기간에 Amazon Lex가 응답 카드를 게시하지 못한 횟수입니다.
BotChannelSystemErrors	지정된 기간에 Amazon Lex에서 채널에 대해 발생한 내부 오류 수입니다.

Amazon Lex 채널 연결 지표는 AWS/Lex 네임스페이스를 사용하며, 다음 차원에 대한 지표를 제공합니다. 지표는 CloudWatch 콘솔에서 차원별로 그룹화할 수 있습니다.

차원	설명
BotAlias, BotChannelName, BotName, Source	봇 별칭, 채널 이름, 봇 이름 및 트래픽 출처별로 지표를 그룹화합니다.

대화 로그에 대한 CloudWatch 지표

Amazon Lex는 대화 로깅에 다음 지표를 사용합니다.

지표	설명
ConversationLogsAudioDeliverySuccess	지정된 기간 동안 S3 버킷에 성공적으로 전달된 오디오 로그 수입니다. 단위: 개
ConversationLogsAudioDeliveryFailure	지정된 기간 동안 S3 버킷에 전달하지 못한 오디오 로그 수입니다. 전달 실패는 대화 로그에 대해 구성된 리소스에 오류가 있음을 나타냅니다. 오류에는 IAM 권한 부족, 액세스할 수 없는 AWS KMS 키 또는 액세스할 수 없는 S3 버킷이 포함될 수 있습니다. 단위: 개

지표	설명
ConversationLogsTextDeliverySuccess	지정된 기간 동안 CloudWatch에 성공적으로 전달된 텍스트 로그 수입니다. 단위: 개
ConversationLogsTextDeliveryFailure	지정된 기간 동안 CloudWatch에 전달하지 못한 텍스트 로그 수입니다. 전달 실패는 대화 로그에 대해 구성된 리소스에 오류가 있음을 나타냅니다. 오류에는 IAM 권한 부족, 액세스할 수 없는 AWS KMS 키 또는 액세스할 수 없는 CloudWatch Logs 로그 그룹이 포함될 수 있습니다. 단위: 개

Amazon Lex 대화 로그 지표는 AWS/Lex 네임스페이스를 사용하며 다음 차원에 대한 지표를 제공합니다. 지표는 CloudWatch 콘솔에서 차원별로 그룹화할 수 있습니다.

차원	설명
BotAlias	봇의 별칭별로 지표를 그룹화.
BotName	봇의 이름별로 지표를 그룹화.
BotVersion	봇의 버전별로 지표를 그룹화.

AWS CloudTrail 로그를 사용하여 Amazon Lex API 호출 모니터링

Amazon Lex는 Amazon Lex의 사용자 AWS CloudTrail, 역할 또는 서비스가 수행한 작업에 대한 레코드를 제공하는 AWS 서비스와 통합됩니다. CloudTrail은 Amazon Lex 콘솔의 호출 및 Amazon Lex API에 대한 코드 호출을 포함하여 Amazon Lex에 대한 API 직접 호출의 하위 집합을 이벤트로 캡처합니다. 추적을 생성하면 Amazon Lex 이벤트를 포함한 CloudTrail 이벤트를 지속적으로 Amazon S3 버킷에 배포할 수 있습니다. 트레일을 구성하지 않은 경우에도 CloudTrail 콘솔의 이벤트 기록에서 최신 이벤트를 볼 수 있습니다. CloudTrail에서 수집한 정보를 사용하여 Amazon Lex에 수행된 요청, 요청이 수행된 IP 주소, 요청을 수행한 사람, 요청이 수행된 시간 및 추가 세부 정보를 확인할 수 있습니다.

구성 및 사용 방법을 포함하여 CloudTrail에 대한 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하세요.

CloudTrail의 Amazon Lex 정보

AWS 계정을 생성할 때 계정에서 CloudTrail이 활성화됩니다. 지원되는 이벤트 활동이 Amazon Lex에서 발생하면, 해당 활동이 이벤트 기록의 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. AWS 계정에서 최근 이벤트를 보고 검색하고 다운로드할 수 있습니다. 자세한 정보는 CloudTrail 이벤트 기록을 사용하여 이벤트 보기를 참조하세요.

Amazon Lex에 대한 이벤트를 포함하여 AWS 계정의 이벤트를 지속적으로 기록하려면 추적을 생성합니다. CloudTrail은 추적(trail)을 사용하여 Amazon Simple Storage Service (Amazon S3) 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 추적을 생성하면 기본적으로 모든 AWS Regions에 추적이 적용됩니다. 추적은 AWS 파티션의 모든 리전에서 이벤트를 로깅하고 지정한 S3 버킷으로 로그 파일을 전송합니다. 또한 CloudTrail 로그에서 수집된 이벤트 데이터를 추가로 분석하고 조치를 취하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 다음을 참조하세요.

- 트레일 생성 개요
- CloudTrail 지원 서비스 및 통합
- CloudTrail에서 Amazon SNS 알림 구성
- 여러 리전으로부터 CloudTrail 로그 파일 받기 및 여러 계정으로부터 CloudTrail 로그 파일 받기

Amazon Lex는 CloudTrail 로그 파일의 이벤트로 다음 작업의 로깅을 지원합니다.

- CreateBotVersion
- CreateIntentVersion
- CreateSlotTypeVersion
- DeleteBot
- DeleteBotAlias
- DeleteBotChannelAssociation
- DeleteBotVersion
- DeleteIntent
- DeleteIntentVersion
- DeleteSlotType

- DeleteSlotTypeVersion
- DeleteUtterances
- GetBot
- GetBotAlias
- GetBotAliases
- GetBotChannelAssociation
- GetBotChannelAssociations
- GetBots
- GetBotVersions
- GetBuiltinIntent
- GetBuiltinIntents
- GetBuiltinSlotTypes
- GetSlotTypeVersions
- GetUtterancesView
- PutBot
- PutBotAlias
- PutIntent
- PutSlotType

모든 이벤트 또는 로그 항목에는 요청을 생성했던 사용자에 대한 정보가 포함됩니다. 이 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청을 루트로 했는지 아니면 사용자 보안 인증으로 했는지 여부
- 역할 또는 연합된 사용자에 대한 임시 보안 인증을 사용하여 요청이 생성되었는지 여부
- 다른 AWS 서비스에서 요청했는지 여부

자세한 내용은 CloudTrail userIdentity 요소를 참조하세요.

CloudTrail 로그에 로깅되는 작업에 대한 자세한 내용은 Amazon Lex 모델 구축 서비스를 참조하십시오. 예를 들어, PutBot, GetBot 및 DeleteBot작업에 대한 호출은 CloudTrail 로그 파일의 항목을 생성합니다. Amazon Lex Runtime Service, PostContent 및 PostText에 기록되는 작업은 로깅되지 않습니다.

예: Amazon Lex로그 파일 항목

추적이란 지정한 S3 버킷에 이벤트를 로그 파일로 입력할 수 있도록 하는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함될 수 있습니다. 이벤트는 모든 소스의 단일 요청을 나타내며 요청된 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보를 포함하고 있습니다. CloudTrail 로그 파일은 퍼블릭 API 직접 호출에 대한 순서 지정된 스택 추적이 아니기 때문에 특정 순서로 표시되지 않습니다.

다음 CloudTrail 로그 예시는 `PutBot` 작업에 대한 호출 결과를 보여줍니다.

```
    {
                "eventVersion": "1.05",
                "userIdentity": {
                    "type": "AssumedRole | FederatedUser | IAMUser | Root | SAMLUser |
WebIdentityUser",
                    "principalId": "principal ID",
                    "arn": "ARN",
                    "accountId": "account ID",
                    "accessKeyId": "access key ID",
                    "userName": "user name"
                },
                "eventTime": "timestamp",
                "eventSource": "lex.amazonaws.com",
                "eventName": "PutBot",
                "awsRegion": "region",
                "sourceIPAddress": "source IP address",
                "userAgent": "user agent",
                "requestParameters": {
                    "name": "CloudTrailBot",
                    "intents": [
                        {
                            "intentVersion": "11",
                            "intentName": "TestCloudTrail"
                        }
                    ],
                    "voiceId": "Salli",
                    "childDirected": false,
                    "locale": "en-US",
                    "idleSessionTTLInSeconds": 500,
                    "processBehavior": "BUILD",
                    "description": "CloudTrail test bot",
                    "clarificationPrompt": {
                        "messages": [
```

```
                            {
                                "contentType": "PlainText",
                                "content": "I didn't understand you. What would you
like to do?"
                            }
                        ],
                        "maxAttempts": 2
                    },
                    "abortStatement": {
                        "messages": [
                            {
                                "contentType": "PlainText",
                                "content": "Sorry. I'm not able to assist at this
time."
                            }
                        ]
                    }
                },
                "responseElements": {
                    "voiceId": "Salli",
                    "locale": "en-US",
                    "childDirected": false,
                    "abortStatement": {
                        "messages": [
                            {
                                "contentType": "PlainText",
                                "content": "Sorry. I'm not able to assist at this
time."
                            }
                        ]
                    },
                    "status": "BUILDING",
                    "createdDate": "timestamp",
                    "lastUpdatedDate": "timestamp",
                    "idleSessionTTLInSeconds": 500,
                    "intents": [
                        {
                            "intentVersion": "11",
                            "intentName": "TestCloudTrail"
                        }
                    ],
                    "clarificationPrompt": {
                        "messages": [
                            {
```

```
                                "contentType": "PlainText",
                                "content": "I didn't understand you. What would you
like to do?"
                            }
                        ],
                        "maxAttempts": 2
                    },
                    "version": "$LATEST",
                    "description": "CloudTrail test bot",
                    "checksum": "checksum",
                    "name": "CloudTrailBot"
                },
                "requestID": "request ID",
                "eventID": "event ID",
                "eventType": "AwsApiCall",
                "recipientAccountId": "account ID"
            }
        }
```

Amazon Lex의 규정 준수 검증

타사 감사자는 여러 규정 준수 프로그램의 일환으로 Amazon Lex의 보안 및 AWS 규정 준수를 평가합니다. Amazon Lex는 HIPAA 적격 서비스입니다. PCI, SOC 및 ISO 규정을 준수합니다. 를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다 AWS Artifact. 자세한 내용은 AWS Artifact에서 보고서 다운로드를 참조하십시오.

Amazon Lex 사용 시 귀하의 규정 준수 책임은 데이터의 민감도, 조직의 규정 준수 목표, 관련 법률과 규정에 따라 결정됩니다. Amazon Lex 사용 시 PCI와 같은 표준으로 규정 준수해야 하는 경우 AWS 는 다음과 같은 유용한 리소스를 제공합니다.

- 보안 및 규정 준수 빠른 시작 안내서 - 아키텍처 고려 사항을 설명하고에 보안 및 규정 준수 중심 기준 환경을 배포하기 위한 단계를 제공하는 배포 안내서 AWS
- HIPAA 보안 및 규정 준수 기술 백서 설계 – 이 백서는 기업에서 AWS를 사용하여 HIPAA를 준수하는 애플리케이션을 만드는 방법을 설명합니다.
- AWS 규정 준수 리소스 – 사용자의 산업 및 위치에 적용될 수 있는 워크북 및 가이드 컬렉션
- AWS Config – 리소스 구성이 내부 관행, 업계 지침 및 규정을 얼마나 잘 준수하고 있는지 평가하는 서비스입니다.

- AWS Security Hub - 보안 업계 표준 및 모범 사례 준수를 확인하는 데 도움이 AWS 되는 내 보안 상태에 대한 포괄적인 보기

특정 규정 준수 프로그램의 범위에 속하는 AWS 서비스 목록은 규정 준수 프로그램의 범위에 속하는 AWS 서비스를 참조하세요. 일반 정보는 AWS 규정 준수 프로그램을 참조하세요.

Amazon Lex의 복원력

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 기반으로 구축됩니다. AWS 리전은 지연 시간이 짧고 처리량이 높으며 중복성이 높은 네트워킹과 연결된 물리적으로 분리되고 격리된 여러 가용 영역을 제공합니다. 가용 영역을 사용하면 중단 없이 가용 영역 간에 자동으로 장애 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 복수 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 AWS 글로벌 인프라를 참조하세요.

AWS 글로벌 인프라 외에도 Amazon Lex는 데이터 복원력 및 백업 요구 사항을 지원하는 데 도움이 되는 몇 가지 기능을 제공합니다.

Amazon Lex의 인프라 보안

관리형 서비스인 Amazon Lex는 Amazon Web Services: 보안 프로세스 개요 백서에 설명된 AWS 글로벌 네트워크 보안 절차로 보호됩니다.

게시된 AWS API 호출을 사용하여 네트워크를 통해 Amazon Lex에 액세스합니다. 클라이언트가 TLS(전송 계층 보안) 1.0을 지원해야 합니다. TLS 1.2 이상을 권장합니다. 클라이언트는 DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Diffie-Hellman Ephemeral)와 같은 PFS(전달 완전 보안)가 포함된 암호 제품군도 지원해야 합니다. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다. 또한 요청은 액세스 키 ID 및 IAM 위탁자와 관련된 보안 암호 액세스 키를 사용하여 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용하여 임시 자격 증명을 생성하여 요청에 서명할 수 있습니다.

이러한 API 작업은 모든 네트워크 위치에서 호출할 수 있지만, Amazon Lex는 원본 IP 주소를 기반으로 하는 제한을 포함할 수 있는 리소스 수준 액세스 정책을 지원합니다. Amazon Lex 정책을 사용하여 특정 Amazon Virtual Private Cloud(Amazon VPC) 엔드포인트 또는 특정 VPC에서 액세스를 제어할 수도 있습니다. 이렇게 하면 네트워크 내의 특정 VPC에서만 지정된 Amazon Lex 리소스에 대한 AWS 네트워크 액세스가 효과적으로 격리됩니다.

Amazon Lex의 가이드라인 및 할당량

다음 섹션에서는 Amazon Lex 사용과 관련된 가이드라인과 할당량이 나와 있습니다.

주제

- 지원되는 리전
- 일반 가이드라인
- 할당량

지원되는 리전

Amazon Lex를 사용할 수 있는 AWS 리전 목록은 Amazon Web Services 일반 참조의 AWS 리전 및 엔드포인트를 참조하세요.

일반 가이드라인

이 섹션에서는 Amazon Lex 사용에 대한 일반적인 가이드라인을 설명합니다.

- 서명 요청 – API 참조 내의 Amazon Lex의 모든 모델 구축 및 런타임 API 작업에서는 요청 인증을 위해 서명 V4를 사용합니다. 요청 인증에 대한 자세한 내용은 Amazon Web Services 일반 참조의 서명 버전 4 서명 프로세스를 참조하십시오.

  PostContent의 경우 Amazon Lex는 Amazon Simple Storage Service(S3) API 참조의 권한 부여 헤더에 대한 서명 계산: 단일 청크에서 페이로드 전송(AWS 서명 버전 4)에 설명된 서명되지 않은 페이로드 옵션을 사용합니다.

  서명되지 않은 페이로드 옵션을 사용하는 경우 표준 요청에 페이로드의 해시를 포함시키지 마세요. 대신 페이로드의 해시로 리터럴 문자열 "UNSIGNED-PAYLOAD"를 사용합니다. 또한 `PostContent` 요청에 이름이 `x-amz-content-sha256`이고 값이 `UNSIGNED-PAYLOAD`인 헤더를 포함시킵니다.

- Amazon Lex가 사용자 표현에서 슬롯 값을 캡처하는 방식에 대해서는 아래를 참고하십시오.

Amazon Lex는 슬롯 유형 정의에 제공한 열거 값을 사용하여 기계 학습 모델을 교육합니다. 다음과 같은 샘플 표현을 사용하여 `GetPredictionIntent`라는 의도를 정의한다고 가정합시다.

```
"Tell me the prediction for {Sign}"
```

여기서 `{Sign}`은 사용자 지정 슬롯 유형인 `ZodiacSign`입니다. 열거 값은 `Aries`부터 `Pisces`까지 12개입니다. "...에 대한 예측을 말해줘"라는 사용자 표현에서 Amazon Lex는 다음에 오는 것이 별자리라는 것을 이해합니다.

PutSlotType 작업을 사용하여 `valueSelectionStrategy` 필드를 `ORIGINAL_VALUE`로 설정하거나, 콘솔에서 값 확장을 선택한 경우, 사용자가 "지구에 대한 예측을 말해줘"라고 말하면 Amazon Lex는 "지구"가 `ZodiacSign`라고 유추하고 이 값을 클라이언트 애플리케이션 또는 Lambda 함수로 전달합니다. 이행 활동에서 슬롯 값을 사용하기 전에 슬롯 값에 유효한 값이 들어 있는지 확인해야 합니다.

`valueSelectionStrategy` 필드를 PutSlotType 작업을 사용하여 `TOP_RESOLUTION`으로 설정하거나, 콘솔에서 슬롯 값 및 동의어로 제한을 선택한 경우 반환되는 값은 슬롯 유형에 대해 정의한 값으로 제한됩니다. 예를 들어 사용자가 "지구에 대한 예측을 말해줘"라고 말하는 경우, 슬롯 유형에 대해 정의된 값 중 하나가 아니기 때문에 이를 인식하지 못합니다. 슬롯 값에 대한 동의어를 정의하면 슬롯 값과 동일하게 인식하지만 동의어 대신 슬롯 값을 반환합니다.

Amazon Lex가 Lambda 함수를 호출하거나 클라이언트 애플리케이션과의 음성 상호작용 결과를 반환하는 경우, 슬롯 값의 대/소문자는 보장되지 않습니다. 예를 들어 AMAZON.Movie 기본 제공 슬롯 유형에 대한 값을 유도하고 있는 경우 사용자가 "Gone with the wind"라고 말하거나 입력하면 Amazon Lex는 "Gone with the Wind," "gone with the wind" 또는 "Gone With The Wind"를 반환할 수 있습니다. 텍스트 상호 작용 시, 슬롯 값은 입력한 텍스트와 일치하거나 `valueResolutionStrategy` 필드 값에 따라 달라집니다.

- 약자가 포함된 슬롯 값을 정의할 때는 다음 패턴을 사용하십시오.

- 마침표로 구분된 대문자(D.V.D.)
- 스페이스로 구분된 대문자(D V D)

- Amazon Lex는 Alexa Skills Kit가 지원하는 AMAZON.LITERAL 기본 제공 슬롯 유형을 지원하지 않습니다. 그러나 Amazon Lex는 이 기능을 구현하는 데 사용할 수 있는 사용자 지정 슬롯 유형 생성을 지원합니다. 이전 항목에서 설명한 것처럼 사용자 지정 슬롯 유형 정의 외부의 값을 캡처할 수 있습니다. 더 많은 다양한 열거 값을 추가하여 자동 음성 인식(ASR)및 자연 언어 이해(NLU)의 정확도를 높일 수 있습니다.

- AMAZON.DATE 및 AMAZON.TIME 기본 제공 슬롯 유형은 절대 및 상대 날짜와 시간을 둘 다 캡처합니다. 상대 날짜 및 시간은 Amazon Lex가 요청을 처리하는 리전에서 확인됩니다.

  `AMAZON.TIME` 기본 제공 슬롯 유형의 경우 시간이 오전인지 또는 오후인지를 사용자가 지정하지 않으면 시간이 모호하므로 Amazon Lex는 사용자에게 메시지를 다시 표시합니다. 따라서 절대 시간을 유도하는 프롬프트를 사용하는 것이 좋습니다. 예를 들어 "언제 피자가 배달되기를 원하시나요? 오후 6시 또는 저녁 6시로 답할 수 있습니다"와 같은 프롬프트를 사용합니다.

- 혼동을 일으킬 수 있는 교육 데이터를 봇에 제공하면 사용자 입력을 이해하는 Amazon Lex의 기능이 감소됩니다. 아래 예를 참조하십시오:

  봇에 두 가지 의도(`OrderPizza` 및 `OrderDrink`)가 있다고 가정합니다. 여기에서 두 가지 의도 모두 "주문 하고 싶어"라는 표현으로 구성됩니다. 이 표현은 Amazon Lex가 빌드 시간에 봇용 언어 모델을 구축하는 동안 학습할 수 있는 특정 의도에 매핑되지 않습니다. 따라서 런타임에 사용자가 이 표현을 입력하면 Amazon Lex는 높은 신뢰도로 의도를 선택할 수 없습니다.

  사용자 확인을 얻기 위해 사용자 지정 의도를 정의하는 또 다른 예(`MyCustomConfirmationIntent`)를 가정하고, "예" 및 "아니오" 표현으로 의도를 구성합니다. Amazon Lex에는 사용자 확인을 이해하기 위한 언어 모델도 있습니다. 이는 충돌하는 상황을 초래할

수 있습니다. 사용자가 "예"라고 응답하는 경우, 이것은 지속적인 의도를 확인하는 것이거나 생성한 사용자 지정 의도를 사용자가 요청하는 중임을 뜻합니다.

일반적으로 사용자가 제공한 샘플 표현은 특정 의도에, 그리고 선택 사항으로, 특정 슬롯 값에 매핑되어야 합니다.

- 런타임 API 작업 PostContent 및 PostText에서는 사용자 ID를 필수 파라미터로 받아들입니다. 개발자는 이 값을 API에 설명된 제약 조건을 충족하는 어떤 값으로도 설정할 수 있습니다. 사용자 로그인, 이메일 또는 주민등록번호와 같은 기밀 정보를 전송할 때는 이 파라미터를 사용하지 않는 것이 좋습니다. 이 ID는 봇과의 대화를 식별하는 데 주로 사용됩니다(피자를 주문하는 사용자가 여럿일 수 있음).

- 클라이언트 애플리케이션에서 인증을 위해 Amazon Cognito를 사용하는 경우, Amazon Cognito 사용자 ID와 Amazon Lex 사용자 ID를 사용할 수 있습니다. 봇 용으로 구성된 모든 Lambda 함수에는 Amazon Lex가 대신하여 Lambda 함수를 호출하는 사용자를 식별하기 위한 자체 인증 메커니즘이 있어야 합니다.

- Amazon은 대화를 중단하고자 하는 사용자의 의도를 캡처하는 의도를 정의하도록 장려합니다. 예를 들어, 실험 표현("나는 아무것도 원하지 않아", "종료", "갈게") 포함, 슬롯 미포함, 코드 후크로 구성된 Lambda 함수 미포함 의도(`NothingIntent`)를 정의할 수 있습니다. 이렇게 하면 사용자가 편안하게 대화를 종료할 수 있습니다.

할당량

이 섹션에서는 Amazon Lex의 현재 할당량에 대해 설명합니다. 이러한 할당량은 범주별로 그룹화됩니다.

서비스 할당량을 조정하거나 늘릴 수 있습니다. 할당량을 늘리려면 AWS 고객 지원팀에 문의하세요. 서비스 할당량을 늘리는 데 며칠이 걸릴 수 있습니다. 대규모 프로젝트의 일환으로 할당량을 늘리는 경우 이 시간을 계획에 추가하세요.

주제

- 런타임 서비스 할당량
- 모델 구축 할당량

런타임 서비스 할당량

API 참조에 설명된 할당량 외에도 다음에 유의하십시오.

API 할당량

- PostContent 작업의 음성 입력 길이는 최대 15초입니다.
- PostContent 및 PostText 런타임 API 작업 모두에서 입력 텍스트 크기는 최대 1,024개 유니코드 문자입니다.
- `PostContent` 헤더의 최대 크기는 16KB입니다. 요청 및 세션 헤더를 합한 최대 크기는 12KB입니다.
- 텍스트 모드에서 `PostContent` 또는 `PostText` 작업을 사용하는 경우 봇과 동시에 대화할 수 있는 최대 수는 `$LATEST` 별칭의 경우 2개, 다른 모든 별칭의 경우 50개입니다. 할당량은 각 API에 개별적으로 적용됩니다.
- 음성 모드에서 `PostContent` 작업을 사용하는 경우 봇과의 최대 동시 텍스트 모드 대화 수는 `$LATEST` 별칭의 경우 2개이고 다른 모든 별칭의 경우 125개입니다. 할당량은 각 API에 개별적으로 적용됩니다.
- 동시 세션 관리 호출(PutSession, GetSession, DeleteSession)의 최대 수는 봇 `$LATEST` 별칭의 경우 2개, 기타 모든 별칭의 경우 50개입니다.
- Lambda 함수에 대한 최대 입력 크기는 12KB입니다. 최대 출력 크기는 25KB인데, 이 중 12KB는 세션 속성일 수 있습니다.

$LATEST 버전 사용

- 봇 $LATEST 버전은 수동 테스트에만 사용해야 합니다. Amazon Lex는 봇 $LATEST 버전에 대해 실행할 수 있는 런타임 요청 수를 제한합니다.

- $LATEST 버전의 봇을 업데이트하면 Amazon Lex는 $LATEST 버전의 봇을 사용하여 클라이언트 애플리케이션에 대해 진행 중인 모든 대화를 종료합니다. 일반적으로, $LATEST 버전은 업데이트할 수 있으므로 프로덕션 환경에서 $LATEST 버전의 봇을 사용하면 안 됩니다. 대신에 버전을 게시하고 이를 사용해야 합니다.

- 별칭을 업데이트하면 Amazon Lex에서 변경 사항을 적용하는 데 몇 분 정도 걸릴 수 있습니다. $LATEST 버전의 봇을 수정하는 경우에는 변경 사항이 즉시 적용됩니다.

세션 제한 시간

- 봇을 생성했을 때 설정한 세션 제한 시간은 봇이 현재 사용자 의도 및 슬롯 데이터와 같은 대화 컨텍스트를 유지할 수 있는 시간을 결정합니다.

- 다른 버전을 가리키도록 봇 별칭을 업데이트하더라도 사용자가 봇과 대화를 시작한 후 세션이 만료될 때까지 Amazon Lex는 동일한 봇 버전을 사용합니다.

모델 구축 할당량

모델 구축이란 봇을 생성하고 관리하는 것을 말합니다. 봇, 의도, 슬롯 유형, 슬롯, 봇 채널 연결을 생성하고 관리하는 것이 여기에 포함됩니다.

주제

- 봇 할당량
- 의도 할당량
- 슬롯 유형 할당량

봇 할당량

- 모델 구축 API 전체에서 프롬프트 및 문장을 구성합니다. 이러한 각 프롬프트 또는 문장에는 메시지가 최대 5개 있을 수 있고, 각 메시지는 UTF-8 문자 1~1,000개를 포함할 수 있습니다.

- 메시지 그룹을 사용하는 경우 각 메시지마다 최대 다섯 개 메시지 그룹을 정의할 수 있습니다. 각 메시지 그룹에는 최대 다섯 개 메시지를 포함할 수 있으므로, 모든 메시지 그룹에서 15개 메시지로 제한됩니다.

- 의도 및 슬롯에 대한 샘플 표현을 정의할 수 있습니다. 모든 표현에는 최대 200,000자를 사용할 수 있습니다.

- 각 슬롯 유형은 값 및 동의어를 최대 10,000개까지 정의할 수 있습니다. 각 봇은 슬롯 유형 값 및 동의어를 최대 50,000개까지 포함할 수 있습니다.

- 봇, 별칭, 봇 채널 연결 이름은 생성 시 대/소문자를 구분하지 않습니다. 따라서 `PizzaBot`을 생성한 후 `pizzaBot`을 생성하려고 하면 오류가 발생합니다. 그러나 리소스에 액세스할 때 리소스 이름은 대/소문자를 구분하므로 `pizzaBot`이 아닌 `PizzaBot`을 지정해야 합니다. 이러한 이름은 ASCII 문자 2~50개로 되어 있어야 합니다.

- 모든 리소스 유형에 대해 게시할 수 있는 최대 버전 개수는 100개입니다. 단, 별칭 버전 관리 기능은 없습니다.

- 봇 내의 의도 이름과 슬롯 이름은 고유해야 하므로 의도와 슬롯 이름이 동일할 수 없습니다.

- 여러 의도를 지원하도록 구성된 봇을 생성할 수 있습니다. 두 가지 의도에 동일한 이름의 슬롯이 있는 경우, 해당 슬롯 이름이 동일해야 합니다.

  예를 들어, 두 가지 의도(`OrderPizza` 및 `OrderDrink`)를 지원하도록 봇을 생성한다고 가정해 보겠습니다. 이 두 의도에 모두 `size` 슬롯이 있는 경우, 이 슬롯 유형은 두 위치 모두에서 동일해야 합니다.

  또한 이들 의도 중 하나에 있는 슬롯에 제공하는 샘플 표현은 다른 의도의 이름이 같은 슬롯에 적용됩니다.

- 의도는 최대 250개까지 봇과 연결할 수 있습니다.

- 봇 생성 시 세션 제한 시간을 지정합니다. 세션 제한 시간은 1분에서 하루일 수 있습니다. 기본값은 5분입니다.

- 봇에 대해 별칭을 최대 다섯 개 만들 수 있습니다.

- AWS 계정당 봇을 최대 250개까지 만들 수 있습니다.

- 동일한 기본 제공 의도에서 확장된 여러 개의 의도를 만들 수 없습니다.

의도 할당량

- 의도 및 슬롯 이름은 생성 시 대/소문자를 구분하지 않습니다. 따라서 `OrderPizza` 의도를 생성한 후 다른 `orderPizza` 의도를 다시 생성하려고 하면 오류가 발생합니다. 그러나 이러한 리소스에 액세스할 때 리소스 이름은 대/소문자를 구분합니다(`orderPizza`가 아닌 `OrderPizza`를 지정해야 함). 이러한 이름은 ASCII 문자 1~100개로 되어 있어야 합니다.

- 의도 한 개는 최대 1,500개의 샘플 표현을 보유할 수 있습니다. 샘플 표현은 최소 하나가 필요합니다. 각 샘플 표현은 최대 200자의 UTF-8 문자일 수 있습니다. 봇에서 모든 의도 및 슬롯 표현에 대해 최대 200,000자를 사용할 수 있습니다. 의도의 샘플 표현:
  - 0개 이상의 슬롯 이름을 참조할 수 있습니다.
  - 슬롯 이름은 한 번만 참조할 수 있습니다.

  예시:

  ```
  I want a pizza
  I want a {pizzaSize} pizza
  I want a {pizzaSize} {pizzaTopping} pizza
  ```

- 각 의도는 표현을 최대 1,500개까지 지원하지만, 표현을 적게 사용할수록 Amazon Lex는 제공된 집합 이외의 입력을 더 잘 인식할 수 있습니다.

- 의도에 각 메시지에 대한 최대 다섯 개 메시지 그룹을 생성할 수 있습니다. 메시지 하나에 대한 모든 메시지 그룹에 총 15개 메시지가 있을 수 있습니다.

- 이 콘솔은 `conclusionStatement` 및 `followUpPrompt` 메시지에 대해서만 메시지 그룹을 만들 수 있습니다. Amazon Lex API를 사용하여 다른 메시지에 대한 메시지 그룹을 생성할 수 있습니다.

- 각 슬롯은 샘플 표현을 최대 10개까지 보유할 수 있습니다. 각 샘플 표현은 슬롯 이름을 한 번만 표시해야 합니다. 예시:

  ```
  {pizzaSize} please
  ```

- 각 봇에는 의도 및 슬롯 표현을 합해 최대 200,000자를 사용할 수 있습니다.

- 기본 제공 의도에서 연장하는 의도에 표현을 제공할 수 없습니다. 다른 모든 의도의 경우 하나 이상의 샘플 표현을 제공해야 합니다. 의도에는 슬롯이 포함되어 있지만, 슬롯 수준 샘플 표현은 선택 사항입니다.

- 기본 제공 의도
  - 현재 Amazon Lex는 기본 제공 의도에 대한 슬롯 유도를 지원하지 않습니다. 기본 제공 의도에서 파생된 의도가 있는 응답에 `ElicitSlot` 지시문 반환하는 Lambda 함수를 생성할 수 없습니다. 자세한 내용은 응답 형식을 참조하세요.
  - 이 서비스는 기본 제공 의도에 샘플 표현을 추가하는 기능을 지원하지 않습니다. 이와 마찬가지로 기본 제공 의도에 슬롯을 추가하거나 제거할 수 없습니다.

- AWS 계정당 의도를 최대 1,000개까지 만들 수 있습니다. 의도에 슬롯을 최대 100개까지 만들 수 있습니다.

슬롯 유형 할당량

- 슬롯 유형 이름은 생성 시 대/소문자를 구분하지 않습니다. `PizzaSize` 슬롯 유형을 생성한 후 `pizzaSize` 슬롯 유형을 다시 생성하면 오류가 발생합니다. 그러나 이러한 리소스에 액세스할 때 리소스 이름은 대/소문자를 구분합니다(`pizzaSize`가 아닌 `PizzaSize`를 지정해야 함). 이름은 ASCII 문자 1~100자여야 합니다.

- 생성한 사용자 지정 슬롯 유형은 열거 값 및 동의어를 최대 10,000개까지 보유할 수 있습니다. 각 값은 UTF-8 문자로 최대 140자일 수 있습니다. 열거 값 및 동의어는 중복 항목을 포함할 수 없습니다.

- 해당되는 경우 슬롯 유형 값에 대해 대문자와 소문자를 모두 지정합니다. 예를 들어 `Procedure`라는 슬롯 유형의 경우 값이 `MRI`이면 "MRI"와 "mri"를 모두 값으로 지정합니다.

- 기본 제공 슬롯 유형 – 현재 Amazon Lex는 기본 제공 슬롯 유형에 대해 열거 값 또는 동의어를 추가하는 기능을 지원하지 않습니다.

API 참조

이 단원에서는 Amazon Lex API 작업에 대한 설명서를 제공합니다. 현재 Amazon Lex 사용 가능한 모든 AWS 리전 목록은 Amazon Web Services 일반 참조의 AWS 리전 및 엔드포인트를 참조하세요.

주제

- 작업
- 데이터 타입

작업

다음 작업이 Amazon Lex 모델 구축 서비스에서 지원됩니다.

- CreateBotVersion
- CreateIntentVersion
- CreateSlotTypeVersion
- DeleteBot
- DeleteBotAlias
- DeleteBotChannelAssociation
- DeleteBotVersion
- DeleteIntent
- DeleteIntentVersion
- DeleteSlotType
- DeleteSlotTypeVersion
- DeleteUtterances
- GetBot
- GetBotAlias
- GetBotAliases
- GetBotChannelAssociation
- GetBotChannelAssociations
- GetBots

- GetBotVersions
- GetBuiltinIntent
- GetBuiltinIntents
- GetBuiltinSlotTypes
- GetExport
- GetImport
- GetIntent
- GetIntents
- GetIntentVersions
- GetMigration
- GetMigrations
- GetSlotType
- GetSlotTypes
- GetSlotTypeVersions
- GetUtterancesView
- ListTagsForResource
- PutBot
- PutBotAlias
- PutIntent
- PutSlotType
- StartImport
- StartMigration
- TagResource
- UntagResource

다음 작업이 Amazon Lex 런타임 서비스에서 지원됩니다.

- DeleteSession
- GetSession
- PostContent
- PostText

- PutSession

Amazon Lex 모델 구축 서비스

다음 작업이 Amazon Lex 모델 구축 서비스에서 지원됩니다.

- CreateBotVersion
- CreateIntentVersion
- CreateSlotTypeVersion
- DeleteBot
- DeleteBotAlias
- DeleteBotChannelAssociation
- DeleteBotVersion
- DeleteIntent
- DeleteIntentVersion
- DeleteSlotType
- DeleteSlotTypeVersion
- DeleteUtterances
- GetBot
- GetBotAlias
- GetBotAliases
- GetBotChannelAssociation
- GetBotChannelAssociations
- GetBots
- GetBotVersions
- GetBuiltinIntent
- GetBuiltinIntents
- GetBuiltinSlotTypes
- GetExport
- GetImport
- GetIntent
- GetIntents

- GetIntentVersions
- GetMigration
- GetMigrations
- GetSlotType
- GetSlotTypes
- GetSlotTypeVersions
- GetUtterancesView
- ListTagsForResource
- PutBot
- PutBotAlias
- PutIntent
- PutSlotType
- StartImport
- StartMigration
- TagResource
- UntagResource

CreateBotVersion

서비스: Amazon Lex Model Building Service

$LATEST 버전에 따라 새 버전의 봇을 생성합니다. 이 리소스의 $LATEST 버전이 마지막 버전 생성 후 변경되지 않은 경우 Amazon Lex는 새 버전을 생성하지 않고 마지막으로 생성된 버전을 반환합니다. 마지막으로 생성된 버전을 반환합니다.

> **Note**
>
> $LATEST 버전의 봇만 업데이트할 수 있습니다. `CreateBotVersion` 작업을 통해 만든 번호가 매겨진 버전은 업데이트할 수 없습니다.

봇의 첫 번째 버전을 지정할 때 Amazon Lex는 버전을 1로 설정합니다. 후속 버전은 1씩 증가합니다. 자세한 내용은 버전 관리을 참조하세요.

이 작업에는 `lex:CreateBotVersion` 작업에 대한 권한이 필요합니다.

Request Syntax

```
POST /bots/name/versions HTTP/1.1
Content-type: application/json

{
   "checksum": "string"
}
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

name

새 버전을 생성하려는 봇의 이름. 이름은 대/소문자를 구분합니다.

길이 제한: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

요청 본문

요청은 JSON 형식으로 다음 데이터를 받습니다.

checksum

봇 $LATEST 버전의 특정 개정 버전을 식별합니다. 체크섬을 지정하였는데 봇 $LATEST 버전의 체크섬이 다른 경우 PreconditionFailedException 예외가 반환되고 Amazon Lex는 새 버전을 게시하지 않습니다. 체크섬을 지정하지 않으면 Amazon Lex에서 $LATEST 버전을 게시합니다.

유형: 문자열

필수사항: 아니요

응답 구문

```
HTTP/1.1 201
Content-type: application/json

{
   "abortStatement": {
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "checksum": "string",
   "childDirected": boolean,
   "clarificationPrompt": {
      "maxAttempts": number,
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
```

```
    "createdDate": number,
    "description": "string",
    "detectSentiment": boolean,
    "enableModelImprovements": boolean,
    "failureReason": "string",
    "idleSessionTTLInSeconds": number,
    "intents": [
      {
         "intentName": "string",
         "intentVersion": "string"
      }
    ],
    "lastUpdatedDate": number,
    "locale": "string",
    "name": "string",
    "status": "string",
    "version": "string",
    "voiceId": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 201 응답을 다시 전송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

abortStatement

Amazon Lex에서 대화를 취소하는 데 사용하는 메시지입니다. 자세한 내용은 PutBot을 참조하세요.

유형: Statement 객체

checksum

생성된 봇의 버전을 식별하는 체크섬입니다.

유형: 문자열

childDirected

Amazon Lex Model Building Service로 만든 각 Amazon Lex 봇에 대해, Amazon Lex 사용이, 전체 또는 일부가, 13세 미만 어린이를 대상으로 하거나 대상으로 하는 웹 사이트, 프로그램 또는 기타 애플리케이션과 관련이 있는지 여부 및 어린이 온라인 개인정보 보호 법률(COPPA)을 준수하는지

를 `childDirected` 필드의 `true` 또는 `false`를 지정하여 지정해야 합니다. `childDirected` 필드에 `true`을 지정함으로써 사용자는 Amazon Lex 사용이 13세 미만 및 COPPA 적용 대상 아동 전체 또는 일부를 대상으로 하는 웹 사이트, 프로그램 또는 기타 애플리케이션과 관련이 있음을 확인합니다. `childDirected` 필드에 `false`을 지정함으로써 사용자는 Amazon Lex 사용이 13세 미만 및 COPPA 적용 대상 아동 전체 또는 일부를 대상으로 하는 웹 사이트, 프로그램 또는 기타 애플리케이션과 관련이 없음을 확인합니다. 사용자는 Amazon Lex 사용이 13세 미만 및 COPPA 적용 대상 아동 전체 또는 일부를 대상으로 하는 웹 사이트, 프로그램 또는 기타 애플리케이션과 관련이 있는지 여부를 정확하게 반영하지 않는 `childDirected` 필드에 대한 기본값을 지정할 수 없습니다.

Amazon Lex의 사용이 전체 또는 부분적으로 13세 미만 어린이를 대상으로 하는 웹 사이트, 프로그램 또는 기타 애플리케이션과 관련된 경우 COPPA에 따라 검증 가능한 필수 부모 동의를 얻어야 합니다. 전체 또는 일부를 13세 미만 어린이를 대상으로 하거나 대상으로 하는 웹 사이트, 프로그램 또는 기타 애플리케이션과 관련된 Amazon Lex 사용에 관한 정보는 Amazon Lex FAQ를 참조하십시오.

유형: 부울

clarificationPrompt

Amazon Lex가 사용자의 요청을 이해하지 못할 때 사용하는 메시지입니다. 자세한 내용은 PutBot을 참조하세요.

유형: Prompt 객체

createdDate

봇 버전이 생성된 날짜입니다.

유형: 타임스탬프

description

봇에 대한 설명입니다.

유형: 문자열

길이 제한: 최소 길이는 0. 최대 길이는 200.

detectSentiment

감정 분석을 위해 사용자가 입력한 표현을 Amazon Comprehend 로 전송해야 하는지 여부를 나타냅니다.

유형: 부울

enableModelImprovements

봇이 정확도 개선을 사용하는지 여부를 나타냅니다. `true`은 봇이 개선 사항을 사용하고 있음을 나타내며, 그렇지 않으면, `false`입니다.

유형: 부울

failureReason

만약 `status`이 `FAILED`라면, Amazon Lex는 봇 구축에 실패한 이유를 제공합니다.

유형: 문자열

idleSessionTTLInSeconds

Amazon Lex가 대화에서 수집된 데이터를 유지하는 최대 시간(초) 입니다. 자세한 내용은 PutBot을 참조하세요.

유형: 정수

유효한 범위: 최소값은 60. 최대값은 86,400.

intents

`Intent` 객체 어레이. 자세한 내용은 PutBot을 참조하세요.

유형: Intent 객체 배열

lastUpdatedDate

이 봇의 `$LATEST` 버전이 업데이트된 날짜.

유형: 타임스탬프

locale

봇의 타겟 로캘을 지정합니다.

타입: 문자열

유효 값: `de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR`

name

봇의 이름.

유형: 문자열

길이 제한: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

status

봇을 만들거나 업데이트하라는 요청을 보내면 Amazon Lex는 `status` 응답 요소를 `BUILDING`로 설정합니다. Amazon Lex가 봇을 빌드한 후에는 `status`을 `READY`로 설정합니다. Amazon Lex가 봇을 빌드하지 못하는 경우에는 `status`을 `FAILED`로 설정합니다. Amazon Lex는 `failureReason` 응답 요소에 실패 이유를 반환합니다.

타입: 문자열

유효 값: `BUILDING | READY | READY_BASIC_TESTING | FAILED | NOT_BUILT`

version

봇의 버전.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

voiceId

Amazon Lex가 사용자와의 음성 상호 작용에 사용하는 Amazon Polly 음성 ID입니다.

유형: 문자열

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

ConflictException

요청을 처리하는 동안 충돌이 발생했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 409

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

PreconditionFailedException

변경하려는 리소스의 체크섬이 요청의 체크섬과 일치하지 않습니다. 리소스의 체크섬을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 412

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

CreateIntentVersion

서비스: Amazon Lex Model Building Service

의도의 `$LATEST` 버전을 기반으로 의도의 새 버전을 생성합니다. 이 의도의 `$LATEST` 버전이 마지막 업데이트 이후 변경되지 않은 경우 Amazon Lex는 새 버전을 생성하지 않습니다. 마지막으로 생성된 버전을 반환합니다.

> **Note**
>
> `$LATEST` 버전의 의도만 업데이트할 수 있습니다. `CreateIntentVersion` 작업을 통해 만든 번호가 매겨진 버전은 업데이트할 수 없습니다.

의도 버전을 생성하면 Amazon Lex는 버전을 1로 설정합니다. 후속 버전은 1씩 증가합니다. 자세한 내용은 버전 관리을 참조하세요.

이 작업에는 `lex:CreateIntentVersion` 조치를 수행할 권한이 요구됩니다.

Request Syntax

```
POST /intents/name/versions HTTP/1.1
Content-type: application/json

{
   "checksum": "string"
}
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

name

새 버전을 생성하려는 봇의 버전. 이름은 대/소문자를 구분합니다.

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

요청 본문

요청은 JSON 형식으로 다음 데이터를 받습니다.

checksum

새 버전을 생성하는 데 사용해야 하는 의도 `$LATEST` 버전의 체크섬. 체크섬을 지정했는데 의도 `$LATEST` 버전의 체크섬이 다른 경우 Amazon Lex는 `PreconditionFailedException` 예외를 반환하고 새 버전을 게시하지 않습니다. 체크섬을 지정하지 않으면 Amazon Lex 에서 `$LATEST` 버전을 게시합니다.

유형: 문자열

필수사항: 아니요

응답 구문

```
HTTP/1.1 201
Content-type: application/json

{
   "checksum": "string",
   "conclusionStatement": {
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "confirmationPrompt": {
      "maxAttempts": number,
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
```

```
    "createdDate": number,
    "description": "string",
    "dialogCodeHook": {
       "messageVersion": "string",
       "uri": "string"
    },
    "followUpPrompt": {
       "prompt": {
          "maxAttempts": number,
          "messages": [
             {
                "content": "string",
                "contentType": "string",
                "groupNumber": number
             }
          ],
          "responseCard": "string"
       },
       "rejectionStatement": {
          "messages": [
             {
                "content": "string",
                "contentType": "string",
                "groupNumber": number
             }
          ],
          "responseCard": "string"
       }
    },
    "fulfillmentActivity": {
       "codeHook": {
          "messageVersion": "string",
          "uri": "string"
       },
       "type": "string"
    },
    "inputContexts": [
       {
          "name": "string"
       }
    ],
    "kendraConfiguration": {
       "kendraIndex": "string",
       "queryFilterString": "string",
```

```
        "role": "string"
    },
    "lastUpdatedDate": number,
    "name": "string",
    "outputContexts": [
        {
            "name": "string",
            "timeToLiveInSeconds": number,
            "turnsToLive": number
        }
    ],
    "parentIntentSignature": "string",
    "rejectionStatement": {
        "messages": [
            {
                "content": "string",
                "contentType": "string",
                "groupNumber": number
            }
        ],
        "responseCard": "string"
    },
    "sampleUtterances": [ "string" ],
    "slots": [
        {
            "defaultValueSpec": {
                "defaultValueList": [
                    {
                        "defaultValue": "string"
                    }
                ]
            },
            "description": "string",
            "name": "string",
            "obfuscationSetting": "string",
            "priority": number,
            "responseCard": "string",
            "sampleUtterances": [ "string" ],
            "slotConstraint": "string",
            "slotType": "string",
            "slotTypeVersion": "string",
            "valueElicitationPrompt": {
                "maxAttempts": number,
                "messages": [
```

```
                {
                   "content": "string",
                   "contentType": "string",
                   "groupNumber": number
                }
             ],
             "responseCard": "string"
          }
       }
    ],
    "version": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 201 응답을 다시 전송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

checksum

의도 버전의 체크섬 생성.

유형: 문자열

conclusionStatement

`fulfillmentActivity` 필드에 지정된 Lambda 함수가 의도를 이행한 후 Amazon Lex는 이 명령문을 사용자에게 전달합니다.

유형: Statement객체

confirmationPrompt

의도에 정의된 경우 Amazon Lex는 사용자에게 의도를 이행하기 전에 의도를 확인하라는 메시지를 표시합니다.

유형: Prompt객체

createdDate

의도가 생성된 날짜입니다.

유형: 타임스탬프

description

의도에 대한 설명.

유형: 문자열

길이 제한: 최소 길이는 0. 최대 길이 200.

dialogCodeHook

정의된 경우 Amazon Lex는 각 사용자 입력에 대해 이 Lambda 함수를 호출합니다.

유형: CodeHook객체

followUpPrompt

정의된 경우 Amazon Lex는 이 프롬프트를 사용하여 의도가 이행된 후 추가 사용자 활동을 요청합니다.

유형: FollowUpPrompt객체

fulfillmentActivity

의도가 이행되는 방법을 설명합니다.

유형: FulfillmentActivity객체

inputContexts

Amazon Lex가 사용자와의 대화에서 의도를 선택하기 위해 활성화되어야 하는 컨텍스트를 목록화하는 `InputContext` 객체 배열입니다.

타입: InputContext객체 배열

배열 멤버: 최소 항목 수 0개. 최대 항목 수는 5개.

kendraConfiguration

Amazon Kendra 인덱스를 `AMAZON.KendraSearchIntent` 의도와 연결하기 위한 구성 정보(있는 경우)입니다.

유형: KendraConfiguration객체

lastUpdatedDate

의도가 업데이트된 날짜.

유형: 타임스탬프

name

의도의 이름.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

outputContexts

의도가 이행될 때 의도가 활성화하는 컨텍스트를 목록화하는 `OutputContext` 객체의 배열입니다.

타입: OutputContext객체 배열

배열 멤버: 최소 항목 수는 0개. 최대 항목 수는 10개.

parentIntentSignature

기본 제공 의도의 고유 식별자입니다.

유형: 문자열

rejectionStatement

사용자가 `confirmationPrompt`에 정의된 질문에 "아니요"라고 답하면 Amazon Lex는 의도가 취소되었음을 확인하기 위해 이 문장으로 답합니다.

유형: Statement객체

sampleUtterances

의도에 대해 구성된 샘플 표현 배열.

유형: 문자열 배열

배열 구성원: 최소수는 0개입니다. 최대 항목 수 1,500개.

길이 제약 조건: 최소 길이는 1입니다. 최대 길이는 200입니다.

slots

의도를 충족하는 데 필요한 정보를 정의하는 다양한 슬롯 유형.

타입: Slot객체 배열

배열 구성원: 최소수는 0개입니다. 최대 항목 수는 100개.

version

새 버전의 의도에 할당된 버전 번호입니다.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

ConflictException

요청을 처리하는 동안 충돌이 발생했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 409

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

PreconditionFailedException

변경하려는 리소스의 체크섬이 요청의 체크섬과 일치하지 않습니다. 리소스의 체크섬을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 412

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

CreateSlotTypeVersion

서비스: Amazon Lex Model Building Service

지정된 슬롯 유형의 `$LATEST` 버전을 기반으로 새 버전을 생성합니다. 이 리소스의 `$LATEST` 버전이 생성된 마지막 버전 이후 변경되지 않은 경우 Amazon Lex는 새 버전을 생성하지 않습니다. 마지막으로 생성된 버전을 반환합니다.

> **Note**
>
> 슬롯 유형의 `$LATEST` 버전만 업데이트할 수 있습니다. `CreateSlotTypeVersion` 작업을 통해 만든 번호가 매겨진 버전은 업데이트할 수 없습니다.

의도 버전을 생성하면 Amazon Lex는 버전을 1로 설정합니다. 후속 버전은 1씩 증가합니다. 자세한 내용은 버전 관리을 참조하세요.

이 작업에는 `lex:CreateSlotTypeVersion` 액션에 대한 권한이 필요합니다.

Request Syntax

```
POST /slottypes/name/versions HTTP/1.1
Content-type: application/json

{
   "checksum": "string"
}
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

name

새 버전을 생성하려는 슬롯 유형의 이름. 이름은 대/소문자를 구분합니다.

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

요청 본문

요청은 JSON 형식으로 다음 데이터를 받습니다.

checksum

게시하려는 슬롯 유형 `$LATEST` 버전의 체크섬. 체크섬을 지정했는데 슬롯 유형 `$LATEST` 버전의 체크섬이 다른 경우 Amazon Lex는 `PreconditionFailedException` 예외를 반환하고 새 버전을 게시하지 않습니다. 체크섬을 지정하지 않으면 Amazon Lex 에서 `$LATEST` 버전을 게시합니다.

유형: 문자열

필수사항: 아니요

응답 구문

```
HTTP/1.1 201
Content-type: application/json

{
   "checksum": "string",
   "createdDate": number,
   "description": "string",
   "enumerationValues": [
      {
         "synonyms": [ "string" ],
         "value": "string"
      }
   ],
   "lastUpdatedDate": number,
   "name": "string",
   "parentSlotTypeSignature": "string",
   "slotTypeConfigurations": [
      {
         "regexConfiguration": {
            "pattern": "string"
         }
      }
   ],
   "valueSelectionStrategy": "string",
   "version": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 201 응답을 다시 전송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

checksum

슬롯 유형의 `$LATEST` 버전의 체크섬.

유형: 문자열

createdDate

슬롯 유형이 생성된 날짜.

유형: 타임스탬프

description

슬롯 유형에 대한 설명.

유형: 문자열

길이 제한: 최소 길이는 0. 최대 길이는 200.

enumerationValues

슬롯 유형이 사용할 수 있는 값을 정의하는 `EnumerationValue` 객체의 목록.

타입: EnumerationValue객체 배열

배열 멤버: 최소 항목 수 0개. 최대 항목 수 10,000개.

lastUpdatedDate

슬롯 유형이 업데이트된 날짜. 리소스를 생성할 때 생성 날짜 및 최종 업데이트 날짜가 동일합니다.

유형: 타임스탬프

name

슬롯 유형의 이름.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

parentSlotTypeSignature

기본 제공 슬롯 유형은 슬롯 유형의 상위 유형을 사용했습니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^((AMAZON\.)_?|[A-Za-z]_?)+`

slotTypeConfigurations

상위 기본 제공 슬롯 유형을 확장하는 구성 정보.

타입: SlotTypeConfiguration객체 배열

배열 멤버: 최소 항목 수는 0개. 최대 항목 수는 10개.

valueSelectionStrategy

Amazon Lex가 슬롯 값을 결정하는 데 사용하는 전략입니다. 자세한 내용은 PutSlotType을 참조하세요.

타입: 문자열

유효 값: `ORIGINAL_VALUE | TOP_RESOLUTION`

version

새 슬롯 유형 버전에 할당된 버전.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

ConflictException

요청을 처리하는 동안 충돌이 발생했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 409

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

PreconditionFailedException

변경하려는 리소스의 체크섬이 요청의 체크섬과 일치하지 않습니다. 리소스의 체크섬을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 412

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3

- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

DeleteBot

서비스: Amazon Lex Model Building Service

`$LATEST` 버전을 포함한 모든 버전의 봇을 삭제합니다. 특정 버전의 봇을 삭제하려면 DeleteBotVersion 작업을 사용하세요. `DeleteBot` 작업을 수행해도 봇 스키마가 즉시 제거되지는 않습니다. 대신 삭제 표시되고 나중에 제거됩니다.

Amazon Lex는 사용자 입력에 응답하는 봇의 능력을 향상시키기 위해 표현을 무기한 저장합니다. 봇이 삭제되어도 이러한 표현은 제거되지 않습니다. 표현을 삭제하려면 DeleteUtterances 작업을 사용하십시오.

봇에 별칭이 있으면 삭제할 수 없습니다. 대신 `DeleteBot` 작업은 봇을 참조하는 별칭에 대한 참조가 포함된 `ResourceInUseException` 예외를 반환합니다. 봇에 대한 참조를 제거하려면 별칭을 삭제하십시오. 동일한 예외가 다시 발생하는 경우 `DeleteBot` 작업이 성공할 때까지 참조 별칭을 삭제하십시오.

이 작업에는 `lex:DeleteBot` 액션에 대한 권한이 필요합니다.

Request Syntax

```
DELETE /bots/name HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

name

봇의 이름. 이름은 대/소문자를 구분합니다.

길이 제한: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

액션이 성공하면 해당 서비스는 빈 HTTP 본문과 함께 HTTP 204 응답을 되돌려줍니다.

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

ConflictException

요청을 처리하는 동안 충돌이 발생했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 409

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

ResourceInUseException

삭제하려는 리소스가 다른 리소스에서 참조되고 있습니다. 이 정보를 사용하여 삭제하려는 리소스에 대한 참조를 제거할 수 있습니다.

예외 본문은 리소스를 설명하는 JSON 객체를 포함하고 있습니다.

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,
"resourceReference": {
"name": string, "version": string } }
```

HTTP 상태 코드: 400

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

DeleteBotAlias

서비스: Amazon Lex Model Building Service

지정된 봇의 별칭을 삭제합니다.

봇과 메시징 채널 간의 연결에 사용되는 별칭은 삭제할 수 없습니다. 별칭이 채널 연결에 사용되는 경우 `DeleteBot` 작업은 봇을 참조하는 채널 연결에 대한 참조를 포함하는 `ResourceInUseException` 예외를 반환합니다. 채널 연결을 삭제하여 별칭에 대한 참조를 제거할 수 있습니다. 동일한 예외가 다시 발생하면 `DeleteBotAlias` 작업이 성공할 때까지 참조 연결을 삭제하세요.

Request Syntax

```
DELETE /bots/botName/aliases/name HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

botName

별칭이 가리키는 봇의 이름.

길이 제약: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

name

삭제할 별칭의 이름 이름은 대/소문자를 구분합니다.

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

액션이 성공하면 해당 서비스는 빈 HTTP 본문과 함께 HTTP 204 응답을 되돌려줍니다.

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

ConflictException

요청을 처리하는 동안 충돌이 발생했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 409

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

ResourceInUseException

삭제하려는 리소스가 다른 리소스에서 참조되고 있습니다. 이 정보를 사용하여 삭제하려는 리소스에 대한 참조를 제거할 수 있습니다.

예외 본문은 리소스를 설명하는 JSON 객체를 포함하고 있습니다.

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,
"resourceReference": {
"name": string, "version": string } }
```

HTTP 상태 코드: 400

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go용 SDK v2
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

DeleteBotChannelAssociation

서비스: Amazon Lex Model Building Service

Amazon Lex 봇과 메시징 플랫폼 간의 연결을 삭제합니다.

이 작업에는 `lex:DeleteBotChannelAssociation` 작업에 대한 권한이 필요합니다.

Request Syntax

```
DELETE /bots/botName/aliases/aliasName/channels/name HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

aliasName

이 연결이 이루어지는 Amazon Lex 봇의 특정 버전을 가리키는 별칭입니다.

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

botName

Amazon Lex 봇의 이름.

길이 제약: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

name

연결의 이름입니다. 이름은 대/소문자를 구분합니다.

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

액션이 성공하면 해당 서비스는 빈 HTTP 본문과 함께 HTTP 204 응답을 되돌려줍니다.

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

ConflictException

요청을 처리하는 동안 충돌이 발생했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 409

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

DeleteBotVersion

서비스: Amazon Lex Model Building Service

봇의 특정 버전을 삭제합니다. 봇의 모든 버전을 삭제하려면 DeleteBot 작업을 사용하세요.

이 작업에는 `lex:DeleteBotVersion` 액션에 대한 권한이 필요합니다.

Request Syntax

```
DELETE /bots/name/versions/version HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

name

봇의 이름.

길이 제약: 최소 길이 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

version

삭제할 봇의 버전. 슬롯 유형의 `$LATEST` 버전은 삭제할 수 없습니다. `$LATEST` 버전을 삭제하려면 DeleteBot 작업을 사용하십시오.

길이 제약: 최소 길이는 1. 최대 길이는 64.

패턴: `[0-9]+`

필수 여부: 예

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

액션이 성공하면 해당 서비스는 빈 HTTP 본문과 함께 HTTP 204 응답을 되돌려줍니다.

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

ConflictException

요청을 처리하는 동안 충돌이 발생했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 409

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

ResourceInUseException

삭제하려는 리소스가 다른 리소스에서 참조되고 있습니다. 이 정보를 사용하여 삭제하려는 리소스에 대한 참조를 제거할 수 있습니다.

예외 본문은 리소스를 설명하는 JSON 객체를 포함하고 있습니다.

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,

"resourceReference": {
```

```
"name": string, "version": string } }
```

HTTP 상태 코드: 400

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

DeleteIntent

서비스: Amazon Lex Model Building Service

`$LATEST` 버전을 포함한 모든 버전의 의도를 삭제합니다. 특정 버전의 의도를 삭제하려면 DeleteIntentVersion 작업을 사용하세요.

참조하지 않는 의도 버전을 삭제할 수 있습니다. 하나 이상의 봇에서 참조되는 의도를 삭제하려면 (Amazon Lex: 작동 방식 참조) 먼저 해당 참조를 제거해야 합니다.

> **Note**
>
> `ResourceInUseException` 예외가 발생하면 의도가 참조된 위치를 보여주는 예제 참조가 제공됩니다. 의도에 대한 참조를 제거하려면 봇을 업데이트하거나 삭제하세요. 의도를 다시 삭제하려고 시도했을 때 같은 예외가 발생하는 경우, 의도에 참조가 없어지고 `DeleteIntent` 호출이 성공할 때까지 반복하세요.

이 작업에는 `lex:DeleteIntent` 작업에 대한 권한이 필요합니다.

Request Syntax

```
DELETE /intents/name HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

name

의도의 이름. 이름은 대/소문자를 구분합니다.

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

액션이 성공하면 해당 서비스는 빈 HTTP 본문과 함께 HTTP 204 응답을 되돌려줍니다.

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

ConflictException

요청을 처리하는 동안 충돌이 발생했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 409

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

ResourceInUseException

삭제하려는 리소스가 다른 리소스에서 참조되고 있습니다. 이 정보를 사용하여 삭제하려는 리소스에 대한 참조를 제거할 수 있습니다.

예외 본문은 리소스를 설명하는 JSON 객체를 포함하고 있습니다.

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,
"resourceReference": {
"name": string, "version": string } }
```

HTTP 상태 코드: 400

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

DeleteIntentVersion

서비스: Amazon Lex Model Building Service

의도의 특정 버전을 삭제합니다. 의도의 모든 버전을 삭제하려면 DeleteIntent 작업을 사용하세요.

이 작업에는 `lex:DeleteIntentVersion` 액션에 대한 권한이 필요합니다.

Request Syntax

```
DELETE /intents/name/versions/version HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

name

의도의 이름.

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

version

삭제할 의도의 버전. 슬롯 유형의 `$LATEST` 버전은 삭제할 수 없습니다. `$LATEST` 버전을 삭제하려면 DeleteIntent 작업을 사용하십시오.

길이 제약: 최소 길이는 1. 최대 길이는 64.

패턴: `[0-9]+`

필수 여부: 예

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

액션이 성공하면 해당 서비스는 빈 HTTP 본문과 함께 HTTP 204 응답을 되돌려줍니다.

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

ConflictException

요청을 처리하는 동안 충돌이 발생했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 409

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

ResourceInUseException

삭제하려는 리소스가 다른 리소스에서 참조되고 있습니다. 이 정보를 사용하여 삭제하려는 리소스에 대한 참조를 제거할 수 있습니다.

예외 본문은 리소스를 설명하는 JSON 객체를 포함하고 있습니다.

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,

"resourceReference": {
```

```
"name": string, "version": string } }
```

HTTP 상태 코드: 400

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

DeleteSlotType

서비스: Amazon Lex Model Building Service

`$LATEST` 버전을 포함한 모든 버전의 슬롯을 삭제합니다. 특정 버전의 봇을 삭제하려면 DeleteSlotTypeVersion 작업을 사용하세요.

참조하지 않는 슬롯 유형 버전을 삭제할 수 있습니다. 하나 이상의 의도에서 참조되는 슬롯 유형을 삭제하려면 먼저 해당 참조를 제거해야 합니다.

> **Note**
>
> `ResourceInUseException` 예외가 발생하면 슬롯 유형이 참조된 위치를 보여주는 예제 참조가 제공됩니다. 슬롯 유형에 대한 참조를 제거하려면 의도를 업데이트하거나 삭제하세요. 슬롯 유형을 다시 삭제하려고 할 때 동일한 예외가 발생하면 슬롯 유형에 참조가 없어 `DeleteSlotType` 호출이 성공할 때까지 반복하십시오.

이 작업에는 `lex:DeleteSlotType` 작업에 대한 권한이 필요합니다.

Request Syntax

```
DELETE /slottypes/name HTTP/1.1
```

URI 요청 파라미터

다음의 URI 파라미터를 사용하여 요청합니다.

name

슬롯 유형의 이름. 이름은 대/소문자를 구분합니다.

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

액션이 성공하면 해당 서비스는 빈 HTTP 본문과 함께 HTTP 204 응답을 되돌려줍니다.

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

ConflictException

요청을 처리하는 동안 충돌이 발생했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 409

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

ResourceInUseException

삭제하려는 리소스가 다른 리소스에서 참조되고 있습니다. 이 정보를 사용하여 삭제하려는 리소스에 대한 참조를 제거할 수 있습니다.

예외 본문은 리소스를 설명하는 JSON 객체를 포함하고 있습니다.

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,

"resourceReference": {

"name": string, "version": string } }
```

HTTP 상태 코드: 400

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

DeleteSlotTypeVersion

서비스: Amazon Lex Model Building Service

슬롯 유형의 특정 버전을 삭제합니다. 슬롯 유형의 모든 버전을 삭제하려면 DeleteSlotType 작업을 사용하세요.

이 작업에는 `lex:DeleteSlotTypeVersion` 액션에 대한 권한이 필요합니다.

Request Syntax

```
DELETE /slottypes/name/version/version HTTP/1.1
```

URI 요청 파라미터

다음의 URI 파라미터를 사용하여 요청합니다.

name

슬롯 유형의 이름입니다.

길이 제한: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

version

삭제할 슬롯 유형의 버전입니다. 슬롯 유형의 `$LATEST` 버전은 삭제할 수 없습니다. `$LATEST`버전을 삭제하려면 DeleteSlotType 작업을 사용하십시오.

길이 제약: 최소 길이는 1. 최대 길이는 64.

패턴: `[0-9]+`

필수 여부: 예

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

액션이 성공하면 해당 서비스는 빈 HTTP 본문과 함께 HTTP 204 응답을 되돌려줍니다.

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

ConflictException

요청을 처리하는 동안 충돌이 발생했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 409

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

ResourceInUseException

삭제하려는 리소스가 다른 리소스에서 참조되고 있습니다. 이 정보를 사용하여 삭제하려는 리소스에 대한 참조를 제거할 수 있습니다.

예외 본문은 리소스를 설명하는 JSON 객체를 포함하고 있습니다.

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,
"resourceReference": {
"name": string, "version": string } }
```

HTTP 상태 코드: 400

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

DeleteUtterances

서비스: Amazon Lex Model Building Service

저장된 표현을 삭제합니다.

Amazon Lex는 사용자가 봇에 보내는 표현을 저장합니다. 표현은 GetUtterancesView 작업에 사용하기 위해 15일 동안 저장되며, 이후에는 봇이 사용자 입력에 응답하는 기능을 개선하는 데 사용하기 위해 무기한 저장됩니다.

특정 사용자의 저장된 표현을 삭제하려면 `DeleteUtterances` 작업을 사용하십시오. `DeleteUtterances` 작업을 사용하면 봇이 사용자 입력에 응답하는 능력을 향상시키기 위해 저장된 표현은 즉시 삭제됩니다. `GetUtterancesView` 작업에 사용하기 위해 저장된 표현은 15일 후에 삭제됩니다.

이 작업에는 `lex:DeleteUtterances` 액션에 대한 권한이 필요합니다.

Request Syntax

```
DELETE /bots/botName/utterances/userId HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

botName

표현을 저장한 봇의 이름입니다.

길이 제약: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

userId

표현을 수행한 사용자의 고유 식별자입니다. 표현이 포함된 PostContent 또는 PostText 작업 요청에서 전송된 사용자 ID입니다.

길이 제약: 최소 길이는 2. 최대 길이는 100.

필수 여부: 예

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

액션이 성공하면 해당 서비스는 빈 HTTP 본문과 함께 HTTP 204 응답을 되돌려줍니다.

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

참고

언어별 AWS SDKs

- AWS Command Line Interface

- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

GetBot

서비스: Amazon Lex Model Building Service

특정 봇에 대한 메타데이터를 반환합니다. 봇 이름 및 봇 버전 또는 별칭도 제공해야 합니다.

이 작업에는 `lex:GetBot` 액션에 대한 권한이 필요합니다.

Request Syntax

```
GET /bots/name/versions/versionoralias HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

name

봇의 이름. 이름은 대/소문자를 구분합니다.

길이 제한: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

versionoralias

봇의 버전 또는 별칭입니다.

필수 여부: 예

Request Body

해당 요청에는 본문이 없습니다.

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",
   "detectSentiment": boolean,
   "enableModelImprovements": boolean,
   "failureReason": "string",
   "idleSessionTTLInSeconds": number,
   "intents": [
      {
         "intentName": "string",
         "intentVersion": "string"
      }
   ],
   "lastUpdatedDate": number,
   "locale": "string",
   "name": "string",
   "nluIntentConfidenceThreshold": number,
   "status": "string",
   "version": "string",
   "voiceId": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

abortStatement

사용자가 대화를 완료하지 않고 종료하기로 선택할 때 Amazon Lex가 반환하는 메시지입니다. 자세한 내용은 PutBot을 참조하세요.

유형: Statement객체

checksum

봇의 `$LATEST` 버전의 특정 버전을 식별하는 데 사용되는 봇의 체크섬입니다.

유형: 문자열

childDirected

Amazon Lex Model Building Service로 만든 각 Amazon Lex 봇에 대해, Amazon Lex 사용이, 전체 또는 일부가, 13세 미만 어린이를 대상으로 하거나 대상으로 하는 웹 사이트, 프로그램 또는 기타 애플리케이션과 관련이 있는지 여부 및 어린이 온라인 개인정보 보호 법률(COPPA)을 준수하는지를 `childDirected` 필드의 `true` 또는 `false`를 지정하여 지정해야 합니다. `childDirected` 필드에 `true`을 지정함으로써 사용자는 Amazon Lex 사용이 13세 미만 및 COPPA 적용 대상 아동 전체 또는 일부를 대상으로 하는 웹 사이트, 프로그램 또는 기타 애플리케이션과 관련이 있음을 확인합니다. `childDirected` 필드에 `false`을 지정함으로써 사용자는 Amazon Lex 사용이 13세 미만 및 COPPA 적용 대상 아동 전체 또는 일부를 대상으로 하는 웹 사이트, 프로그램 또는 기타 애플리케이션과 관련이 없음을 확인합니다. 사용자는 Amazon Lex 사용이 13세 미만 및 COPPA 적용 대상 아동 전체 또는 일부를 대상으로 하는 웹 사이트, 프로그램 또는 기타 애플리케이션과 관련이 있는지 여부를 정확하게 반영하지 않는 `childDirected` 필드에 대한 기본값을 지정할 수 없습니다.

Amazon Lex의 사용이 전체 또는 부분적으로 13세 미만 어린이를 대상으로 하는 웹 사이트, 프로그램 또는 기타 애플리케이션과 관련된 경우 COPPA에 따라 검증 가능한 필수 부모 동의를 얻어야 합니다. 전체 또는 일부를 13세 미만 어린이를 대상으로 하거나 대상으로 하는 웹 사이트, 프로그램 또는 기타 애플리케이션과 관련된 Amazon Lex 사용에 관한 정보는 Amazon Lex FAQ를 참조하십시오.

유형: 부울

clarificationPrompt

Amazon Lex가 사용자의 요청을 이해하지 못할 때 사용하는 메시지입니다. 자세한 내용은 PutBot 단원을 참조하십시오.

유형: Prompt객체

createdDate

봇이 생성된 날짜.

유형: 타임스탬프

description

봇에 대한 설명입니다.

유형: 문자열

길이 제한: 최소 길이는 0. 최대 길이는 200.

detectSentiment

감정 분석을 위해 사용자가 입력한 표현을 Amazon Comprehend로 전송해야 하는지 여부를 나타냅니다.

유형: 부울

enableModelImprovements

봇이 정확도 개선을 사용하는지 여부를 나타냅니다. `true`은 봇이 개선 사항을 사용하고 있음을 나타내며, 그렇지 않으면, `false`입니다.

유형: 부울

failureReason

`status`이 `FAILED`인 경우, Amazon Lex가 봇 구축에 실패한 이유를 설명합니다.

유형: 문자열

idleSessionTTLInSeconds

Amazon Lex가 대화에서 수집된 데이터를 유지하는 최대 시간(초) 입니다. 자세한 내용은 PutBot을 참조하세요.

유형: 정수

유효한 범위: 최소값은 60. 최대값은 86,400.

intents

`intent` 객체 어레이. 자세한 내용은 PutBot을 참조하세요.

유형: Intent 객체 배열

lastUpdatedDate

봇이 업데이트된 날짜. 리소스를 생성할 때 생성 날짜 및 최종 업데이트 날짜가 동일합니다.

유형: 타임스탬프

locale

봇의 타겟 로캘입니다.

타입: 문자열

유효 값: `de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR`

name

봇의 이름.

유형: 문자열

길이 제한: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

nluIntentConfidenceThreshold

PostContent 또는 PostText 응답에서 대체 의도를 반환할 때 Amazon Lex가 `AMAZON.FallbackIntent`, `AMAZON.KendraSearchIntent` 또는 둘 다를 삽입하는 위치를 결정하는 점수입니다. `AMAZON.FallbackIntent`는 모든 의도에 대한 신뢰도 점수가 이 값보다 낮은 경우 삽입됩니다. `AMAZON.KendraSearchIntent`은 봇에 대해 구성된 경우에만 삽입됩니다.

유형: 더블

유효한 범위: 최소값 0. 최댓값은 1.

status

봇의 상태.

상태가 `BUILDING`인 경우 Amazon Lex는 테스트 및 사용을 위해 봇을 구축하고 있습니다.

봇 상태가 `READY_BASIC_TESTING`인 경우 봇의 의도에 지정된 정확한 표현을 사용하여 봇을 테스트할 수 있습니다. 봇을 완전히 테스트하거나 실행할 준비가 되면 상태는 `READY`입니다.

봇을 구축하는 데 문제가 있는 경우 상태는 `FAILED`이며 `failureReason` 필드에는 봇이 빌드되지 않은 이유가 설명되어 있습니다.

봇이 저장되었지만 빌드되지 않은 경우 상태는 `NOT_BUILT`입니다.

타입: 문자열

유효 값: `BUILDING | READY | READY_BASIC_TESTING | FAILED | NOT_BUILT`

version

봇의 버전. 새 봇의 경우 버전은 항상 `$LATEST`입니다.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

voiceId

Amazon Lex가 사용자와의 음성 상호 작용에 사용하는 Amazon Polly 음성 ID입니다. 자세한 내용은 PutBot 단원을 참조하십시오.

유형: 문자열

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go용 SDK v2
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

GetBotAlias

서비스: Amazon Lex Model Building Service

Amazon Lex 봇 별칭에 대한 정보를 반환합니다. 별칭에 대한 자세한 내용은 버전 관리 및 별칭을 참조하십시오.

이 작업에는 `lex:GetBotAlias` 액션에 대한 권한이 필요합니다.

Request Syntax

```
GET /bots/botName/aliases/name HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

botName

봇의 이름.

길이 제약: 최소 길이 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

name

봇 별칭의 이름. 이름은 대/소문자를 구분합니다.

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
```

```
Content-type: application/json

{
   "botName": "string",
   "botVersion": "string",
   "checksum": "string",
   "conversationLogs": {
      "iamRoleArn": "string",
      "logSettings": [
         {
            "destination": "string",
            "kmsKeyArn": "string",
            "logType": "string",
            "resourceArn": "string",
            "resourcePrefix": "string"
         }
      ]
   },
   "createdDate": number,
   "description": "string",
   "lastUpdatedDate": number,
   "name": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

botName

별칭이 가리키는 봇의 이름.

유형: 문자열

길이 제한: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

botVersion

별칭이 가리키는 봇의 버전.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

checksum

봇 별칭의 체크섬입니다.

유형: 문자열

conversationLogs

Amazon Lex가 별칭에 대한 대화 로그를 사용하는 방법을 결정하는 설정입니다.

유형: ConversationLogsResponse객체

createdDate

봇이 별칭이 생성된 날짜.

유형: 타임스탬프

description

별칭에 대한 설명입니다.

유형: 문자열

길이 제한: 최소 길이는 0. 최대 길이는 200.

lastUpdatedDate

봇 별칭이 업데이트된 날짜. 리소스를 생성할 때 생성 날짜 및 최종 업데이트 날짜가 동일합니다.

유형: 타임스탬프

name

봇 별칭의 이름.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

GetBotAliases

서비스: Amazon Lex Model Building Service

지정된 Amazon Lex 봇에 대한 별칭 목록을 반환합니다.

이 작업에는 `lex:GetBotAliases` 액션에 대한 권한이 필요합니다.

Request Syntax

```
GET /bots/botName/aliases/?
maxResults=maxResults&nameContains=nameContains&nextToken=nextToken HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

botName

봇의 이름.

길이 제약: 최소 길이 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

maxResults

응답에 반환될 최대 별칭 수입니다. 기본 값은 50.

유효한 범위: 최소값은 1. 최대값은 50.

nameContains

봇 별칭 이름에서 일치시킬 하위 문자열. 이름 중 일부가 하위 문자열과 일치하면 별칭이 반환됩니다. 예를 들어, "xyz"는 "xyzabc"와 "abcxyz"와 모두 일치합니다.

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

nextToken

별칭의 다음 페이지를 가져오기 위한 페이지 매김 토큰입니다. 이 호출에 대한 응답이 잘린 경우, Amazon Lex는 응답에서 페이지 매김 토큰을 반환합니다. 별칭의 다음 페이지를 가져오려면 다음 요청에서 페이지 매김 토큰을 지정하십시오.

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "BotAliases": [
      {
         "botName": "string",
         "botVersion": "string",
         "checksum": "string",
         "conversationLogs": {
            "iamRoleArn": "string",
            "logSettings": [
               {
                  "destination": "string",
                  "kmsKeyArn": "string",
                  "logType": "string",
                  "resourceArn": "string",
                  "resourcePrefix": "string"
               }
            ]
         },
         "createdDate": number,
         "description": "string",
         "lastUpdatedDate": number,
         "name": "string"
      }
   ],
   "nextToken": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

BotAliases

각각 봇 별칭을 설명하는 `BotAliasMetadata` 객체 배열.

타입: BotAliasMetadata객체 배열

nextToken

별칭의 다음 페이지를 가져오기 위한 페이지 매김 토큰입니다. 이 호출에 대한 응답이 잘린 경우, Amazon Lex는 응답에서 페이지 매김 토큰을 반환합니다. 별칭의 다음 페이지를 가져오려면 다음 요청에서 페이지 매김 토큰을 지정하십시오.

유형: 문자열

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

참고 항목

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++

- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

GetBotChannelAssociation

서비스: Amazon Lex Model Building Service

Amazon Lex 봇과 메시징 플랫폼 간의 연결에 대한 정보를 반환합니다.

이 작업에는 `lex:GetBotChannelAssociation` 액션에 대한 권한이 필요합니다.

Request Syntax

```
GET /bots/botName/aliases/aliasName/channels/name HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

aliasName

이 연결이 이루어지는 Amazon Lex 봇의 특정 버전을 가리키는 별칭입니다.

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

botName

Amazon Lex 봇의 이름.

길이 제약: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

name

봇과 채널 간의 연결 이름. 이름은 대/소문자를 구분합니다.

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

Request Body

해당 요청에는 본문이 없습니다.

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"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

botAlias

이 연결이 이루어지는 Amazon Lex 봇의 특정 버전을 가리키는 별칭입니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

botConfiguration

메시징 플랫폼이 Amazon Lex 봇과 통신하는 데 필요한 정보를 제공합니다.

유형: 문자열 간 맵

맵 항목: 최대 항목 수는 10개.

botName

Amazon Lex 봇의 이름.

유형: 문자열

길이 제한: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

createdDate

봇과 채널 간의 연결이 생성된 날짜입니다.

유형: 타임스탬프

description

봇과 채널 간의 연결에 대한 설명입니다.

유형: 문자열

길이 제한: 최소 길이는 0. 최대 길이는 200.

failureReason

만약 `status`이 `FAILED`라면, Amazon Lex는 봇 구축에 실패한 이유를 제공합니다.

유형: 문자열

name

봇과 채널 간의 연결 이름.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

status

봇 채널의 상태입니다.

- `CREATED` - 채널이 생성되어 사용할 준비가 되었습니다.
- `IN_PROGRESS` - 채널 생성이 진행 중입니다.

- `FAILED` - 채널을 만드는 중 오류가 발생했습니다. 필드에 대한 자세한 내용은 `failureReason` 필드를 참조하세요.

타입: 문자열

유효 값: `IN_PROGRESS | CREATED | FAILED`

type

메시징 플랫폼 유형.

타입: 문자열

유효 값: `Facebook | Slack | Twilio-Sms | Kik`

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

GetBotChannelAssociations

서비스: Amazon Lex Model Building Service

지정된 봇과 연결된 모든 채널의 목록을 반환합니다.

`GetBotChannelAssociations` 작업에는 `lex:GetBotChannelAssociations` 작업에 대한 권한이 필요합니다.

Request Syntax

```
GET /bots/botName/aliases/aliasName/channels/?
maxResults=maxResults&nameContains=nameContains&nextToken=nextToken HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

aliasName

이 연결이 이루어지는 Amazon Lex 봇의 특정 버전을 가리키는 별칭입니다.

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^(-|^([A-Za-z]_?)+$)$`

필수 여부: 예

botName

연결에 있는 Amazon Lex 봇의 이름.

길이 제약: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

maxResults

응답에 반환될 최대 연결 수입니다. 기본값은 50.

유효한 범위: 최소값은 1. 최대값은 50.

nameContains

채널 연결 이름과 일치하는 하위 문자열. 이름 중 일부가 하위 문자열과 일치하면 별칭이 반환됩니다. 예를 들어, "xyz"는 "xyzabc"와 "abcxyz"와 모두 일치합니다. 모든 봇 채널 연결을 반환하려면 하이픈("-")을 `nameContains` 파라미터로 사용하십시오.

길이 제약: 최소 길이 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

nextToken

연결의 다음 페이지를 가져오기 위한 페이지 매김 토큰입니다. 이 호출에 대한 응답이 잘린 경우, Amazon Lex는 응답에서 페이지 매김 토큰을 반환합니다. 연결의 다음 페이지를 가져오려면 다음 요청에서 페이지 매김 토큰을 지정하십시오.

Request Body

해당 요청에는 본문이 없습니다.

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"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

botChannelAssociations

Amazon Lex 봇 및 채널과의 연결에 대한 정보를 제공하는 객체 배열(각 연결마다 하나씩)입니다.

타입: BotChannelAssociation객체 배열

nextToken

연결의 다음 페이지를 가져오기 위한 페이지 매김 토큰입니다. 이 호출에 대한 응답이 잘린 경우, Amazon Lex는 응답에서 페이지 매김 토큰을 반환합니다. 연결의 다음 페이지를 가져오려면 다음 요청에서 페이지 매김 토큰을 지정하십시오.

유형: 문자열

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

참고 항목

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

GetBots

서비스: Amazon Lex Model Building Service

다음과 같이 봇 정보를 반환합니다.

- `nameContains` 필드를 제공하는 경우 응답에는 이름에 지정된 문자열이 포함된 모든 봇의 `$LATEST` 버전에 대한 정보가 포함됩니다.
- `nameContains` 필드를 지정하지 않으면 작업은 모든 봇의 `$LATEST` 버전에 대한 정보를 반환합니다.

이 작업에는 `lex:GetBots` 작업에 대한 권한이 필요합니다.

Request Syntax

```
GET /bots/?maxResults=maxResults&nameContains=nameContains&nextToken=nextToken HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

maxResults

요청이 반환할 응답으로 반환할 최대 봇 수입니다. 기본값은 10.

유효한 범위: 최소값은 1. 최대값은 50.

nameContains

봇 이름에서 일치하는 하위 문자열. 이름 중 일부가 하위 문자열과 일치하면 별칭이 반환됩니다. 예를 들어, "xyz"는 "xyzabc"와 "abcxyz"와 모두 일치합니다.

길이 제약: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

nextToken

봇의 다음 페이지를 가져오기 위한 페이지 매김 토큰입니다. 이 호출에 대한 응답이 잘린 경우, Amazon Lex는 응답에서 페이지 매김 토큰을 반환합니다. 봇의 다음 페이지를 가져오려면 다음 요청에서 페이지 매김 토큰을 지정하십시오.

Request Body

해당 요청에는 본문이 없습니다.

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"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

bots

각 봇에 한 개의 항목이 있는 `botMetadata` 객체 배열.

타입: BotMetadata객체 배열

nextToken

응답이 잘린 경우 다음 요청 시 봇의 다음 페이지를 가져오도록 지정할 수 있는 페이지 매김 토큰이 포함됩니다.

유형: 문자열

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

GetBotVersions

서비스: Amazon Lex Model Building Service

봇의 모든 버전에 대한 정보를 가져옵니다.

이 `GetBotVersions` 작업은 봇의 각 버전에 대한 `BotMetadata` 객체를 반환합니다. 예를 들어 봇에 번호가 매겨진 버전이 3개 있는 경우, `GetBotVersions` 작업은 번호가 매겨진 버전마다 하나씩 및 `$LATEST` 버전에 한 개, 총 4개의 `BotMetadata` 객체를 응답으로 반환합니다.

`GetBotVersions` 작업은 항상 하나 이상의 버전, 즉 `$LATEST` 버전을 반환합니다.

이 작업에는 `lex:GetBotVersions` 액션에 대한 권한이 필요합니다.

Request Syntax

```
GET /bots/name/versions/?maxResults=maxResults&nextToken=nextToken HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

maxResults

응답으로 반환할 최대 봇 버전 수입니다. 기본값은 10.

유효한 범위: 최소값은 1. 최대값은 50.

name

버전을 반환해야 할 봇의 이름

길이 제약: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

nextToken

봇 버전의 다음 페이지를 가져오기 위한 페이지 매김 토큰입니다. 이 호출에 대한 응답이 잘린 경우, Amazon Lex는 응답에서 페이지 매김 토큰을 반환합니다. 버전의 다음 페이지를 가져오려면 다음 요청에서 페이지 매김 토큰을 지정하십시오.

Request Body

해당 요청에는 본문이 없습니다.

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"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

bots

번호가 매겨진 각 버전의 봇에 대해 하나씩 및 `$LATEST` 버전에 대한 하나에 해당하는 `BotMetadata` 객체 배열입니다.

타입: BotMetadata객체 배열

nextToken

봇 버전의 다음 페이지를 가져오기 위한 페이지 매김 토큰입니다. 이 호출에 대한 응답이 잘린 경우, Amazon Lex는 응답에서 페이지 매김 토큰을 반환합니다. 버전의 다음 페이지를 가져오려면 다음 요청에서 페이지 매김 토큰을 지정하십시오.

유형: 문자열

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

GetBuiltinIntent

서비스: Amazon Lex Model Building Service

기본 제공 의도에 관한 정보를 반환합니다.

이 작업에는 `lex:GetBuiltinIntent` 작업에 대한 권한이 필요합니다.

Request Syntax

```
GET /builtins/intents/signature HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

signature

기본 제공 의도의 고유 식별자입니다. 의도의 서명을 찾으려면 Alexa Skills Kit의 표준 기본 제공 의도를 참조하십시오.

필수 여부: 예

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "signature": "string",
   "slots": [
      {
         "name": "string"
      }
   ],
   "supportedLocales": [ "string" ]
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

signature

기본 제공 의도의 고유 식별자입니다.

유형: 문자열

slots

의도의 각 슬롯 유형당 하나의 항목으로 구성된 `BuiltinIntentSlot` 객체 배열.

타입: BuiltinIntentSlot객체 배열

supportedLocales

의도가 지원하는 로캘의 목록입니다.

유형: 문자열 배열

유효 값: `de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR`

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

GetBuiltinIntents

서비스: Amazon Lex Model Building Service

지정된 기준에 맞는 기본 제공 의도의 목록을 가져옵니다.

이 작업에는 `lex:GetBuiltinIntents` 작업에 대한 권한이 필요합니다.

Request Syntax

```
GET /builtins/intents/?
locale=locale&maxResults=maxResults&nextToken=nextToken&signatureContains=signatureContains
 HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

locale

의도가 지원하는 로캘의 목록입니다.

유효 값: `de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR`

maxResults

응답에 반환되는 최대 의도 수입니다. 기본값은 10.

유효한 범위: 최소값은 1. 최대값은 50.

nextToken

의도의 다음 페이지를 가져오기 위한 페이지 매김 토큰입니다. 이 API 직접 호출이 잘린 경우, Amazon Lex는 응답에서 페이지 매김 토큰을 반환합니다. 의도의 다음 페이지를 가져오려면 다음 요청에서 페이지 매김 토큰을 지정하십시오.

signatureContains

기본 제공 의도 서명과 일치하는 하위 문자열. 서명의 일부가 하위 문자열과 일치하면 의도가 반환됩니다. 예를 들어, "xyz"는 "xyzabc"와 "abcxyz"와 모두 일치합니다. 의도의 서명을 찾으려면 Alexa Skills Kit의 표준 기본 제공 의도를 참조하십시오.

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "intents": [
      {
         "signature": "string",
         "supportedLocales": [ "string" ]
      }
   ],
   "nextToken": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

intents

응답의 각 의도에 대해 하나씩 있는 `builtinIntentMetadata` 객체의 배열입니다.

타입: BuiltinIntentMetadata객체 배열

nextToken

의도의 다음 페이지를 가져오기 위한 페이지 매김 토큰입니다. 이 API 직접 호출에 대한 응답이 잘린 경우, Amazon Lex는 응답에서 페이지 매김 토큰을 반환합니다. 의도의 다음 페이지를 가져오려면 다음 요청에서 페이지 매김 토큰을 지정하십시오.

유형: 문자열

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

참고 항목

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

GetBuiltinSlotTypes

서비스: Amazon Lex Model Building Service

지정된 기준에 맞는 기본 제공 슬롯 유형을 가져옵니다.

사용 가능한 기본 제공 슬롯 유형의 목록은 Alexa Skills Kit의 슬롯 유형 참고 자료를 참조하십시오.

이 작업에는 `lex:GetBuiltInSlotTypes` 작업에 대한 권한이 필요합니다.

Request Syntax

```
GET /builtins/slottypes/?
locale=locale&maxResults=maxResults&nextToken=nextToken&signatureContains=signatureContains
 HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

locale

슬롯 유형이 지원하는 로캘의 목록입니다.

유효 값: `de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR`

maxResults

응답에 반환되는 최대 슬롯 유형 수입니다. 기본값은 10.

유효한 범위: 최소값은 1. 최대값은 50.

nextToken

슬롯 유형의 다음 페이지를 가져오기 위한 페이지 매김 토큰입니다. 이 API 직접 호출에 대한 응답이 잘린 경우, Amazon Lex는 응답에서 페이지 매김 토큰을 반환합니다. 의도의 다음 페이지를 가져오려면 다음 요청에서 페이지 매김 토큰을 지정하십시오.

signatureContains

기본 제공 슬롯 유형 서명과 일치하는 하위 문자열. 서명의 일부가 하위 문자열과 일치하면 슬롯 유형이 반환됩니다. 예를 들어, "xyz"는 "xyzabc"와 "abcxyz"와 모두 일치합니다.

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "nextToken": "string",
   "slotTypes": [
      {
         "signature": "string",
         "supportedLocales": [ "string" ]
      }
   ]
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

nextToken

응답이 잘린 경우 다음 요청에서 슬롯 유형의 다음 페이지를 가져오는 데 사용할 수 있는 페이지 매김 토큰이 응답에 포함됩니다.

유형: 문자열

slotTypes

의도의 각 슬롯 유형당 하나의 항목으로 구성된 `BuiltInSlotTypeMetadata` 객체 배열.

타입: BuiltinSlotTypeMetadata 객체 배열

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

참고 항목

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

GetExport

서비스: Amazon Lex Model Building Service

Amazon Lex 리소스의 콘텐츠를 지정된 형식으로 내보냅니다.

Request Syntax

```
GET /exports/?exportType=exportType&name=name&resourceType=resourceType&version=version
 HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

exportType

내보내진 데이터의 형식입니다.

유효 값: `ALEXA_SKILLS_KIT | LEX`

필수 사항 여부: 예

name

내보낼 봇의 이름.

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `[a-zA-Z_]+`

필수 여부: 예

resourceType

내보내기 할 리소스의 유형.

유효 값: `BOT | INTENT | SLOT_TYPE`

필수 사항 여부: 예

version

내보낼 봇의 버전.

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `[0-9]+`

필수 여부: 예

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "exportStatus": "string",
   "exportType": "string",
   "failureReason": "string",
   "name": "string",
   "resourceType": "string",
   "url": "string",
   "version": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

exportStatus

내보내기 작업 상태를 가져옵니다.

- `IN_PROGRESS` - 내보내기가 진행 중입니다.
- `READY` - 내보내기가 완료되었습니다.
- `FAILED` - 내보내기를 완료할 수 없습니다.

타입: 문자열

유효 값: `IN_PROGRESS | READY | FAILED`

exportType

내보내진 데이터의 형식입니다.

타입: 문자열

유효 값: `ALEXA_SKILLS_KIT | LEX`

failureReason

만약 `status`이 `FAILED`라면, Amazon Lex는 리소스 내보내기에 실패한 이유를 제공합니다.

유형: 문자열

name

내보내진 봇의 이름.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `[a-zA-Z_]+`

resourceType

내보내진 리소스의 유형.

타입: 문자열

유효 값: `BOT | INTENT | SLOT_TYPE`

url

내보낸 리소스의 위치를 제공하는 S3 미리 서명된 URL입니다. 내보내기의 리소스는 내보내진 리소스가 JSON 형식으로 포함된 ZIP 아카이브 파일입니다. 아카이브 구조는 변경될 수 있습니다. 코드는 아카이브 구조에 의존해서는 안 됩니다.

유형: 문자열

version

내보내진 봇의 버전.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `[0-9]+`

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK

- AWS Python용 SDK
- AWS SDK for Ruby V3

GetImport

서비스: Amazon Lex Model Building Service

`StartImport` 작업으로 시작된 가져오기 작업에 대한 정보를 가져옵니다.

Request Syntax

```
GET /imports/importId HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

importId

반환할 가져오기 작업 정보의 식별자입니다.

필수 여부: 예

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "createdDate": number,
   "failureReason": [ "string" ],
   "importId": "string",
   "importStatus": "string",
   "mergeStrategy": "string",
   "name": "string",
   "resourceType": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

createdDate

가져오기 작업이 생성된 날짜와 시간에 대한 타임스탬프입니다.

유형: 타임스탬프

failureReason

가져오기 작업이 완료되지 못한 이유를 설명하는 문자열입니다.

유형: 문자열 배열

importId

특정 가져오기 작업의 식별자입니다.

유형: 문자열

importStatus

가져오기 작업의 상태입니다. 상태가 `FAILED` 인 경우, `failureReason` 필드로부터 실패 원인을 얻을 수 있습니다.

타입: 문자열

유효 값: `IN_PROGRESS | COMPLETE | FAILED`

mergeStrategy

기존 리소스와 가져오기 파일의 리소스 간에 충돌이 있을 때 취해진 작업입니다.

타입: 문자열

유효 값: `OVERWRITE_LATEST | FAIL_ON_CONFLICT`

name

가져오기 작업의 이름입니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `[a-zA-Z_]+`

resourceType

가져올 리소스의 유형.

타입: 문자열

유효 값: `BOT | INTENT | SLOT_TYPE`

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go용 SDK v2

- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

GetIntent

서비스: Amazon Lex Model Building Service

의도에 대한 정보를 반환합니다. 의도 이름 뿐만 아니라 의도 버전 또한 지정해야 합니다.

이 작업에는 `lex:GetIntent` 조치를 수행할 권한이 요구됩니다.

Request Syntax

```
GET /intents/name/versions/version HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

name

의도의 이름. 이름은 대/소문자를 구분합니다.

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

version

의도의 버전.

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

필수 여부: 예

Request Body

해당 요청에는 본문이 없습니다.

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"
},
"inputContexts": [
    {
        "name": "string"
    }
],
"kendraConfiguration": {
    "kendraIndex": "string",
    "queryFilterString": "string",
    "role": "string"
},
"lastUpdatedDate": number,
"name": "string",
"outputContexts": [
    {
        "name": "string",
        "timeToLiveInSeconds": number,
        "turnsToLive": number
    }
],
"parentIntentSignature": "string",
"rejectionStatement": {
    "messages": [
        {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
        }
    ],
    "responseCard": "string"
},
```

```
    "sampleUtterances": [ "string" ],
    "slots": [
       {
          "defaultValueSpec": {
             "defaultValueList": [
                {
                   "defaultValue": "string"
                }
             ]
          },
          "description": "string",
          "name": "string",
          "obfuscationSetting": "string",
          "priority": number,
          "responseCard": "string",
          "sampleUtterances": [ "string" ],
          "slotConstraint": "string",
          "slotType": "string",
          "slotTypeVersion": "string",
          "valueElicitationPrompt": {
             "maxAttempts": number,
             "messages": [
                {
                   "content": "string",
                   "contentType": "string",
                   "groupNumber": number
                }
             ],
             "responseCard": "string"
          }
       }
    ],
    "version": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

checksum

의도의 체크섬입니다.

유형: 문자열

conclusionStatement

`fulfillmentActivity` 요소에 지정된 Lambda 함수가 의도를 이행한 후, Amazon Lex는 이 명령문을 사용자에게 전달합니다.

유형: Statement객체

confirmationPrompt

봇에 정의된 경우 Amazon Lex는 사용자의 요청을 이행하기 전에 의도를 확인하기 위해 프롬프트를 사용합니다. 자세한 내용은 PutIntent을 참조하세요.

유형: Prompt객체

createdDate

의도가 생성된 날짜입니다.

유형: 타임스탬프

description

의도에 대한 설명.

유형: 문자열

길이 제한: 최소 길이는 0. 최대 길이는 200.

dialogCodeHook

봇이 정의된 경우 Amazon Lex는 각 사용자 입력에 대해 이 Lambda 함수를 호출합니다. 자세한 내용은 PutIntent 단원을 참조하십시오.

유형: CodeHook객체

followUpPrompt

의도에 정의된 경우 Amazon Lex는 이 프롬프트를 사용하여 의도가 충족된 후 추가 사용자 활동을 요청합니다. 자세한 내용은 PutIntent 단원을 참조하십시오.

유형: FollowUpPrompt객체

fulfillmentActivity

의도가 이행되는 방식을 설명합니다. 자세한 내용은 PutIntent 단원을 참조하십시오.

유형: FulfillmentActivity객체

inputContexts

Amazon Lex가 사용자와의 대화에서 의도를 선택하기 위해 활성화되어야 하는 컨텍스트를 목록화하는 `InputContext` 객체 배열입니다.

타입: InputContext객체 배열

배열 멤버: 최소 항목 수 0개. 최대 항목 수는 5개.

kendraConfiguration

Amazon Kendra 인덱스를 `AMAZON.KendraSearchIntent` 의도와 연결하기 위한 구성 정보(있는 경우)입니다.

유형: KendraConfiguration객체

lastUpdatedDate

의도가 업데이트된 날짜. 리소스를 생성할 때 생성 날짜 및 최종 업데이트 날짜가 동일합니다.

유형: 타임스탬프

name

의도의 이름.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

outputContexts

의도가 이행될 때 의도가 활성화하는 컨텍스트를 목록화하는 `OutputContext` 객체의 배열입니다.

타입: OutputContext객체 배열

배열 멤버: 최소 항목 수는 0개. 최대 항목 수는 10개.

parentIntentSignature

기본 제공 의도의 고유 식별자입니다.

유형: 문자열

rejectionStatement

사용자가 `confirmationPrompt`에 정의된 질문에 "아니요"라고 답하면 Amazon Lex는 의도가 취소되었음을 확인하기 위해 이 문장으로 답합니다.

유형: Statement객체

sampleUtterances

의도에 대해 구성된 샘플 표현 배열.

유형: 문자열 배열

배열 구성원: 최소수는 0개입니다. 최대 항목 수 1,500개.

길이 제약 조건: 최소 길이는 1입니다. 최대 길이는 200입니다.

slots

의도에 대해 구성된 의도 슬롯 배열.

타입: Slot객체 배열

배열 구성원: 최소수는 0개입니다. 최대 항목 수 100개.

version

의도의 버전.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

GetIntents

서비스: Amazon Lex Model Building Service

다음과 같이 의도 정보를 반환합니다.

- `nameContains` 필드를 지정하는 경우, 지정된 문자열이 포함된 모든 의도의 `$LATEST` 버전을 반환합니다.
- `nameContains` 필드를 지정하지 않으면 모든 의도의 `$LATEST` 버전에 대한 정보가 반환됩니다.

이 작업에는 `lex:GetIntents` 작업에 대한 권한이 필요합니다.

Request Syntax

```
GET /intents/?maxResults=maxResults&nameContains=nameContains&nextToken=nextToken
 HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

maxResults

응답에 반환되는 최대 의도 수입니다. 기본값은 10.

유효한 범위: 최소값은 1. 최대값은 50.

nameContains

의도 이름과 일치하는 하위 문자열. 의도 이름의 일부가 하위 문자열과 일치하면 의도가 반환됩니다. 예를 들어, "xyz"는 "xyzabc"와 "abcxyz"와 모두 일치합니다.

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

nextToken

의도의 다음 페이지를 가져오기 위한 페이지 매김 토큰입니다. 이 API 직접 호출에 대한 응답이 잘린 경우, Amazon Lex는 응답에서 페이지 매김 토큰을 반환합니다. 의도의 다음 페이지를 가져오려면 다음 요청에서 페이지 매김 토큰을 지정하십시오.

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "intents": [
      {
         "createdDate": number,
         "description": "string",
         "lastUpdatedDate": number,
         "name": "string",
         "version": "string"
      }
   ],
   "nextToken": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

intents

`Intent` 객체 어레이. 자세한 내용은 PutBot을 참조하세요.

유형: IntentMetadata 객체 배열

nextToken

응답이 잘린 경우 응답에는 의도의 다음 페이지를 가져오기 위해 다음 요청에서 지정할 수 있는 페이지 매김 토큰이 포함됩니다.

유형: 문자열

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

GetIntentVersions

서비스: Amazon Lex Model Building Service

의도의 모든 버전에 대한 정보를 반환합니다.

이 `GetIntentVersions` 작업은 의도의 각 버전에 대한 `IntentMetadata` 객체를 반환합니다. 예를 들어 의도에 번호가 매겨진 버전이 3개 있는 경우, `GetIntentVersions` 작업은 번호가 매겨진 버전마다 하나씩 및 `$LATEST` 버전에 한 개, 총 4개의 `IntentMetadata` 객체를 응답으로 반환합니다.

`GetIntentVersions` 작업은 항상 하나 이상의 버전, 즉 `$LATEST` 버전을 반환합니다.

이 작업에는 `lex:GetIntentVersions` 액션에 대한 권한이 필요합니다.

Request Syntax

```
GET /intents/name/versions/?maxResults=maxResults&nextToken=nextToken HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

maxResults

응답으로 반환할 최대 의도 버전 수입니다. 기본값은 10.

유효한 범위: 최소값은 1. 최대값은 50.

name

버전을 반환해야 할 의도의 이름.

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

nextToken

의도 버전의 다음 페이지를 가져오기 위한 페이지 매김 토큰입니다. 이 호출에 대한 응답이 잘린 경우, Amazon Lex는 응답에서 페이지 매김 토큰을 반환합니다. 버전의 다음 페이지를 가져오려면 다음 요청에서 페이지 매김 토큰을 지정하십시오.

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "intents": [
      {
         "createdDate": number,
         "description": "string",
         "lastUpdatedDate": number,
         "name": "string",
         "version": "string"
      }
   ],
   "nextToken": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

intents

번호가 매겨진 각 버전의 의도에 대해 하나씩 및 `$LATEST` 버전에 대한 하나에 해당하는 `IntentMetadata` 객체 배열입니다.

타입: IntentMetadata객체 배열

nextToken

의도 버전의 다음 페이지를 가져오기 위한 페이지 매김 토큰입니다. 이 호출에 대한 응답이 잘린 경우, Amazon Lex는 응답에서 페이지 매김 토큰을 반환합니다. 버전의 다음 페이지를 가져오려면 다음 요청에서 페이지 매김 토큰을 지정하십시오.

유형: 문자열

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go용 SDK v2
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

GetMigration

서비스: Amazon Lex Model Building Service

Amazon Lex V1 봇에서 Amazon Lex V2 봇으로의 진행 중인 또는 전체 마이그레이션에 대한 세부 정보를 제공합니다. 이 작업을 사용하면 마이그레이션과 관련된 마이그레이션 알림 및 경고를 볼 수 있습니다.

Request Syntax

```
GET /migrations/migrationId HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

migrationId

조회할 마이그레이션의 고유 식별자입니다. StartMigration 작업에서 `migrationID`가 반환됩니다.

길이 제약 조건: 고정 길이는 10.

패턴: `^[0-9a-zA-Z]+$`

필수 여부: 예

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "alerts": [
      {
         "details": [ "string" ],
         "message": "string",
         "referenceURLs": [ "string" ],
         "type": "string"
      }
```

```
    ],
    "migrationId": "string",
    "migrationStatus": "string",
    "migrationStrategy": "string",
    "migrationTimestamp": number,
    "v1BotLocale": "string",
    "v1BotName": "string",
    "v1BotVersion": "string",
    "v2BotId": "string",
    "v2BotRole": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

alerts

Amazon Lex V1 봇을 Amazon Lex V2로 마이그레이션할 때 발생하는 문제를 나타내는 경고 및 경고 목록입니다. Amazon Lex V2에서 Amazon Lex V1 기능이 다르게 구현되면 경고 메시지가 표시됩니다.

자세한 내용은 Amazon Lex V2 개발자 가이드의 봇 마이그레이션을 참조하십시오.

타입: MigrationAlert객체 배열

migrationId

마이그레이션의 고유 식별자입니다. `GetMigration` 작업을 호출할 때 사용되는 식별자와 동일합니다.

유형: 문자열

길이 제약 조건: 고정 길이는 10.

패턴: `^[0-9a-zA-Z]+$`

migrationStatus

마이그레이션 상태를 나타냅니다. 상태 `COMPLETE`이 되면 마이그레이션이 완료되고 Amazon Lex V2에서 봇을 사용할 수 있게 됩니다. 마이그레이션을 완료하기 위해 해결해야 하는 알림 및 경고가 있을 수 있습니다.

타입: 문자열

유효 값: `IN_PROGRESS | COMPLETED | FAILED`

migrationStrategy

마이그레이션을 수행하는 데 사용된 전략.

- `CREATE_NEW` - 새 Amazon Lex V2 봇을 만들고 Amazon Lex V1 봇을 새 봇으로 마이그레이션합니다.
- `UPDATE_EXISTING` - 기존 Amazon Lex V2 봇 메타데이터와 마이그레이션 중인 로캘을 덮어씁니다. Amazon Lex V2 봇의 다른 로캘은 변경되지 않습니다. 로캘이 존재하지 않는 경우 Amazon Lex V2 봇에 새 로캘이 생성됩니다.

타입: 문자열

유효 값: `CREATE_NEW | UPDATE_EXISTING`

migrationTimestamp

마이그레이션이 시작된 날짜와 시간.

유형: 타임스탬프

v1BotLocale

Amazon Lex V2로 마이그레이션하려는 아마존 Lex V1 로캘의 이름.

타입: 문자열

유효 값: `de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR`

v1BotName

Amazon Lex V2로 마이그레이션하려는 아마존 Lex V1의 이름.

유형: 문자열

길이 제한: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

v1BotVersion

Amazon Lex V2로 마이그레이션하려는 Amazon Lex V1 버전.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

v2BotId

Amazon Lex V1이 마이그레이션되는 대상 Amazon Lex V2 봇의 고유 식별자입니다.

유형: 문자열

길이 제약 조건: 고정 길이는 10.

패턴: `^[0-9a-zA-Z]+$`

v2BotRole

Amazon Lex가 Amazon Lex V2 봇을 실행하는 데 사용하는 IAM 역할.

유형: 문자열

길이 제약: 최소 길이는 20. 최대 길이는 2,048.

패턴: `^arn:[\w\-]+:iam::[\d]{12}:role/.+$`

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

GetMigrations

서비스: Amazon Lex Model Building Service

아마존 Lex V1과 아마존 Lex V2 간의 마이그레이션 목록을 가져옵니다.

Request Syntax

```
GET /migrations?
maxResults=maxResults&migrationStatusEquals=migrationStatusEquals&nextToken=nextToken&sortByAtt
 HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

maxResults

응답에 반환되는 최대 마이그레이션 수입니다. 기본값은 10.

유효한 범위: 최소값은 1. 최대값은 50.

migrationStatusEquals

지정된 상태의 마이그레이션만 포함하도록 목록을 필터링합니다.

유효 값: `IN_PROGRESS | COMPLETED | FAILED`

nextToken

마이그레이션의 다음 페이지를 가져오기 위한 페이지 매김 토큰입니다. 이 작업에 대한 응답이 잘린 경우, Amazon Lex는 응답에서 페이지 매김 토큰을 반환합니다. 마이그레이션의 다음 페이지를 가져오려면 요청에서 페이지 매김 토큰을 지정하십시오.

sortByAttribute

마이그레이션 목록을 정렬하는 데 사용할 필드입니다. Amazon Lex V1 봇 이름 또는 마이그레이션이 시작된 날짜 및 시간을 기준으로 정렬할 수 있습니다.

유효 값: `V1_BOT_NAME | MIGRATION_DATE_TIME`

sortByOrder

순서에 따라 목록이 정렬됩니다.

유효 값: `ASCENDING | DESCENDING`

v1BotNameContains

이름에 지정된 문자열이 포함된 봇만 포함하도록 목록을 필터링합니다. 문자열은 봇 이름의 모든 위치와 일치합니다.

길이 제약: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

요청 본문

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "migrationSummaries": [
      {
         "migrationId": "string",
         "migrationStatus": "string",
         "migrationStrategy": "string",
         "migrationTimestamp": number,
         "v1BotLocale": "string",
         "v1BotName": "string",
         "v1BotVersion": "string",
         "v2BotId": "string",
         "v2BotRole": "string"
      }
   ],
   "nextToken": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

migrationSummaries

Amazon Lex V1에서 Amazon Lex V2로의 마이그레이션에 대한 요약의 배열입니다. 마이그레이션의 세부 정보를 보려면 GetMigration 작업에 대한 호출 내 요약의 `migrationId`을 사용하십시오.

타입: MigrationSummary객체 배열

nextToken

응답이 잘린 경우 다음 요청 시 마이그레이션의 다음 페이지를 가져오도록 지정할 수 있는 페이지 매김 토큰이 포함됩니다.

유형: 문자열

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

참고 항목

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++

- AWS Go용 SDK v2
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

GetSlotType

서비스: Amazon Lex Model Building Service

슬롯 유형의 특정 버전에 대한 정보를 반환합니다. 슬롯 유형 이름을 지정할 뿐만 아니라 슬롯 유형 버전 또한 지정해야 합니다.

이 작업에는 `lex:GetSlotType` 액션에 대한 권한이 필요합니다.

Request Syntax

```
GET /slottypes/name/versions/version HTTP/1.1
```

URI 요청 파라미터

다음의 URI 파라미터를 사용하여 요청합니다.

name

슬롯 유형의 이름. 이름은 대/소문자를 구분합니다.

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

version

슬롯 유형의 버전.

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

필수 여부: 예

Request Body

해당 요청에는 본문이 없습니다.

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",
   "parentSlotTypeSignature": "string",
   "slotTypeConfigurations": [
      {
         "regexConfiguration": {
            "pattern": "string"
         }
      }
   ],
   "valueSelectionStrategy": "string",
   "version": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

checksum

슬롯 유형의 $LATEST 버전의 체크섬.

유형: 문자열

createdDate

슬롯 유형이 생성된 날짜.

유형: 타임스탬프

description

슬롯 유형에 대한 설명.

유형: 문자열

길이 제한: 최소 길이는 0. 최대 길이는 200.

enumerationValues

슬롯 유형이 사용할 수 있는 값을 정의하는 `EnumerationValue` 객체의 목록.

타입: EnumerationValue객체 배열

배열 멤버: 최소 항목 수 0개. 최대 항목 수 10,000개.

lastUpdatedDate

슬롯 유형이 업데이트된 날짜. 리소스를 생성할 때 생성 날짜 및 최종 업데이트 날짜가 동일합니다.

유형: 타임스탬프

name

슬롯 유형의 이름.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

parentSlotTypeSignature

슬롯 유형의 상위 슬롯 유형으로 사용되는 기본 제공 슬롯 유형입니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^((AMAZON\.)_?|[A-Za-z]_?)+`

slotTypeConfigurations

상위 기본 제공 슬롯 유형을 확장하는 구성 정보.

타입: SlotTypeConfiguration객체 배열

배열 멤버: 최소 항목 수는 0개. 최대 항목 수는 10개.

valueSelectionStrategy

Amazon Lex가 슬롯 값을 결정하는 데 사용하는 전략입니다. 자세한 내용은 PutSlotType을 참조하세요.

타입: 문자열

유효 값: `ORIGINAL_VALUE | TOP_RESOLUTION`

version

슬롯 유형의 버전.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

GetSlotTypes

서비스: Amazon Lex Model Building Service

다음과 같이 슬롯 유형 정보를 반환합니다.

- `nameContains` 필드를 지정하는 경우, 지정된 문자열이 포함된 모든 슬롯 유형의 `$LATEST` 버전을 반환합니다.
- `nameContains` 필드를 지정하지 않으면 모든 슬롯 유형의 `$LATEST` 버전에 대한 정보가 반환됩니다.

이 작업에는 `lex:GetSlotTypes` 작업에 대한 권한이 필요합니다.

Request Syntax

```
GET /slottypes/?maxResults=maxResults&nameContains=nameContains&nextToken=nextToken
 HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

maxResults

응답에 반환되는 최대 슬롯 유형 수입니다. 기본값은 10.

유효한 범위: 최소값은 1. 최대값은 50.

nameContains

슬롯 유형 이름에서 일치하는 하위 문자열. 이름 중 일부가 하위 문자열과 일치하면 슬롯 유형이 반환됩니다. 예를 들어, "xyz"는 "xyzabc"와 "abcxyz"와 모두 일치합니다.

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

nextToken

슬롯 유형의 다음 페이지를 가져오기 위한 페이지 매김 토큰입니다. 이 API 호출에 대한 응답이 잘린 경우, Amazon Lex는 응답에서 페이지 매김 토큰을 반환합니다. 의도의 다음 페이지를 가져오려면 다음 요청에서 페이지 매김 토큰을 지정하십시오.

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "nextToken": "string",
   "slotTypes": [
      {
         "createdDate": number,
         "description": "string",
         "lastUpdatedDate": number,
         "name": "string",
         "version": "string"
      }
   ]
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

nextToken

응답이 잘린 경우 다음 요청 시 슬롯 유형의 다음 페이지를 가져오도록 지정할 수 있는 페이지 매김 토큰이 포함됩니다.

유형: 문자열

slotTypes

슬롯 유형 이름, 버전, 설명 등의 정보를 제공하는 객체 배열(각 슬롯 유형당 하나씩).

타입: SlotTypeMetadata 객체 배열

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

GetSlotTypeVersions

서비스: Amazon Lex Model Building Service

슬롯 유형의 모든 버전에 대한 정보를 얻습니다.

이 `GetSlotTypeVersions` 작업은 슬롯 유형의 각 버전에 대한 `SlotTypeMetadata` 객체를 반환합니다. 예를 들어 슬롯 유형에 번호가 매겨진 버전이 3개 있는 경우, `GetSlotTypeVersions` 작업은 번호가 매겨진 버전마다 하나씩 및 `$LATEST` 버전에 한 개, 총 4개의 `SlotTypeMetadata` 객체를 응답으로 반환합니다.

`GetSlotTypeVersions` 작업은 항상 하나 이상의 버전, 즉 `$LATEST` 버전을 반환합니다.

이 작업에는 `lex:GetSlotTypeVersions` 액션에 대한 권한이 필요합니다.

Request Syntax

```
GET /slottypes/name/versions/?maxResults=maxResults&nextToken=nextToken HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

maxResults

응답에 반환되는 최대 슬롯 유형 버전 수입니다. 기본값은 10.

유효한 범위: 최소값은 1. 최대값은 50.

name

버전을 반환해야 할 슬롯 유형의 이름.

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

nextToken

슬롯 유형 버전의 다음 페이지를 가져오기 위한 페이지 매김 토큰입니다. 이 호출에 대한 응답이 잘린 경우, Amazon Lex는 응답에서 페이지 매김 토큰을 반환합니다. 버전의 다음 페이지를 가져오려면 다음 요청에서 페이지 매김 토큰을 지정하십시오.

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "nextToken": "string",
   "slotTypes": [
      {
         "createdDate": number,
         "description": "string",
         "lastUpdatedDate": number,
         "name": "string",
         "version": "string"
      }
   ]
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

nextToken

슬롯 유형 버전의 다음 페이지를 가져오기 위한 페이지 매김 토큰입니다. 이 호출에 대한 응답이 잘린 경우, Amazon Lex는 응답에서 페이지 매김 토큰을 반환합니다. 버전의 다음 페이지를 가져오려면 다음 요청에서 페이지 매김 토큰을 지정하십시오.

유형: 문자열

slotTypes

번호가 매겨진 각 버전의 봇에 대해 하나씩 및 `$LATEST` 버전에 대한 하나에 해당하는 `SlotTypeMetadata` 객체 배열입니다.

타입: SlotTypeMetadata 객체 배열

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

GetUtterancesView

서비스: Amazon Lex Model Building Service

`GetUtterancesView` 작업을 사용하여 사용자가 봇에 대해 한 표현 정보를 가져옵니다. 이 목록을 사용하여 봇이 반응하는 표현을 조정할 수 있습니다.

예를 들어 꽃을 주문하는 봇을 만들었다고 가정합니다. 사용자가 한동안 봇을 사용한 후 `GetUtterancesView` 작업을 사용하여 요청한 내용과 성공 여부를 확인하세요. "꽃을 원해요"라는 표현이 인식되지 않을 수도 있습니다. 이 표현을 `OrderFlowers` 의도에 추가하여 봇이 해당 표현을 인식하도록 할 수 있습니다.

새 버전의 봇을 게시한 후에는 이전 버전과 새 버전에 대한 정보를 얻을 수 있으므로 두 버전의 성능을 비교할 수 있습니다.

표현 통계는 매일 한 번 생성됩니다. 데이터는 지난 15일 동안 사용할 수 있습니다. 각 요청에서 최대 5개 버전의 봇에 대한 정보를 요청할 수 있습니다. Amazon Lex는 봇이 지난 15일 동안 가장 자주 받은 표현을 반환합니다. 응답에는 각 버전에 대한 최대 100개의 표현에 대한 정보가 포함됩니다.

다음 조건에서는 표현 통계가 생성되지 않습니다.

- 봇이 생성될 때 `childDirected` 필드는 참으로 설정되었습니다.
- 하나 이상의 슬롯에서 슬롯 난독화 기능을 사용하고 있습니다.
- Amazon Lex 개선에 참여하지 않기로 선택했습니다.

이 작업에는 `lex:GetUtterancesView` 액션에 대한 권한이 필요합니다.

Request Syntax

```
GET /bots/botname/utterances?
view=aggregation&bot_versions=botVersions&status_type=statusType HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

botname

표현 정보를 반환해야 하는 봇의 이름.

길이 제약: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

botVersions

표현 정보를 반환해야 하는 봇 버전의 배열. 요청당 5개 버전으로 제한됩니다.

배열 멤버: 최소 항목 수는 1개. 최대 항목 수는 5개.

길이 제약: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

필수 여부: 예

statusType

인식되고 처리된 표현을 반환하려면 `Detected`을 사용하십시오. 인식되지 않은 표현을 반환하려면 `Missed`을 사용하십시오.

유효 값: `Detected | Missed`

필수 사항 여부: 예

Request Body

해당 요청에는 본문이 없습니다.

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"
            }
        ]
    }
  ]
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

botName

표현 정보를 반환해야 하는 봇의 이름.

유형: 문자열

길이 제한: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

utterances

각 UtteranceList 객체에는 봇이 처리한 표현을 설명하는 UtteranceData 객체 목록이 들어 있는 객체 배열입니다. 응답에는 각 버전당 최대 100개의 `UtteranceData` 객체가 포함됩니다. Amazon Lex는 봇이 지난 15일 동안 가장 자주 받은 표현을 반환합니다.

타입: UtteranceList 객체 배열

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

참고 항목

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

ListTagsForResource

서비스: Amazon Lex Model Building Service

지정된 리소스와 연결된 태그를 가져옵니다. 봇, 봇 별칭, 봇 채널에만 태그를 연결할 수 있습니다.

Request Syntax

```
GET /tags/resourceArn HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

resourceArn

태그 목록을 가져올 리소스의 Amazon 리소스 이름(ARN)입니다.

길이 제약: 최소 길이는 1. 최대 길이는 1011입니다.

필수 여부: 예

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "tags": [
      {
         "key": "string",
         "value": "string"
      }
   ]
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

tags

리소스와 연결된 태그를 봅니다.

타입: Tag객체 배열

어레이 멤버: 최소 항목 수 0개. 최대 항목 수 200개.

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET

- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

PutBot

서비스: Amazon Lex Model Building Service

Amazon Lex 대화형 봇을 생성하거나 기존 봇을 교체합니다. 봇을 생성하거나 업데이트할 때는 이름, 로캘, 봇이 13세 미만의 어린이를 대상으로 하는지 여부만 지정하면 됩니다. 이를 사용하여 나중에 의도를 추가하거나 기존 봇에서 의도를 제거할 수 있습니다. 최소 정보로 봇을 생성하면 봇이 생성되거나 업데이트되지만 Amazon Lex는 응답을 반환합니다`FAILED`. 하나 이상의 의도를 추가한 후 봇을 구축할 수 있습니다. Amazon Lex에 대한 자세한 내용은 Amazon Lex: 작동 방식를 참조하십시오.

기존 봇의 이름을 지정하는 경우, 요청의 필드가 봇 `$LATEST` 버전의 기존 값을 대체합니다. Amazon Lex는 요청에서 값을 제공하지 않은 모든 필드를 제거합니다. 단, 기본값으로 설정된 `idleTTLInSeconds` 및 `privacySettings` 필드는 예외입니다. 필수 필드 값을 지정하지 않으면 Amazon Lex에서 예외가 발생합니다.

이 작업에는 `lex:PutBot` 액션에 대한 권한이 필요합니다. 자세한 내용은 Amazon Lex의 Identity and Access Management을 참조하세요.

요청 구문

```
PUT /bots/name/versions/$LATEST HTTP/1.1
Content-type: application/json

{
   "abortStatement": {
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "checksum": "string",
   "childDirected": boolean,
   "clarificationPrompt": {
      "maxAttempts": number,
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
```

```
            }
        ],
        "responseCard": "string"
    },
    "createVersion": boolean,
    "description": "string",
    "detectSentiment": boolean,
    "enableModelImprovements": boolean,
    "idleSessionTTLInSeconds": number,
    "intents": [
        {
            "intentName": "string",
            "intentVersion": "string"
        }
    ],
    "locale": "string",
    "nluIntentConfidenceThreshold": number,
    "processBehavior": "string",
    "tags": [
        {
            "key": "string",
            "value": "string"
        }
    ],
    "voiceId": "string"
}
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

name

봇의 이름. 이름은 대/소문자를 구분하지 않습니다.

길이 제약: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

요청 본문

요청은 JSON 형식으로 다음 데이터를 받습니다.

abortStatement

Amazon Lex는 컨텍스트에서 사용자의 입력을 이해할 수 없는 경우 정보를 추출하려고 몇 번 시도합니다. 그런 다음 Amazon Lex는 `abortStatement`에서 정의한 메시지를 사용자에게 보낸 다음 대화를 취소합니다. 재시도 횟수를 설정하려면 슬롯 유형의 `valueElicitationPrompt` 필드를 사용하십시오.

예를 들어 피자 주문 봇에서 Amazon Lex는 사용자에게 "어떤 종류의 크러스트를 원하시나요?"라고 물을 수 있습니다. 사용자의 응답이 예상 응답 중 하나가 아닌 경우(예: "얇은 크러스트", "딥 디쉬" 등), Amazon Lex는 올바른 응답을 몇 번 더 유도하려고 합니다.

예를 들어 피자 주문 애플리케이션에서는, `OrderPizza`은 이러한 의도 중 하나일 수 있습니다. 이 의도에는 `CrustType` 슬롯이 필요할 수 있습니다. `CrustType` 슬롯을 생성할 때 `valueElicitationPrompt` 필드를 지정합니다.

폴백 의도를 정의한 경우 취소 명령문이 사용자에게 전송되지 않고 대신 폴백 의도가 사용됩니다. 자세한 내용은 AMAZON.FallbackIntent를 참조하십시오.

유형: Statement객체

필수 여부: 아니요

checksum

`$LATEST` 버전의 특정 개정 버전을 식별합니다.

새 봇을 만들 때는 `checksum` 필드를 비워 두십시오. 체크섬을 지정하면 `BadRequestException` 예외가 발생합니다.

봇을 업데이트하려면 `checksum` 필드를 해당 버전의 최신 수정 `$LATEST` 버전의 체크섬으로 설정하십시오. `checksum` 필드를 지정하지 않거나 체크섬이 `$LATEST` 버전과 일치하지 않으면 `PreconditionFailedException` 예외가 발생합니다.

유형: 문자열

필수 항목 여부: 아니요

childDirected

Amazon Lex Model Building Service로 만든 각 Amazon Lex 봇에 대해, Amazon Lex 사용이, 전체 또는 일부가, 13세 미만 어린이를 대상으로 하거나 대상으로 하는 웹 사이트, 프로그램 또는 기타 애플리케이션과 관련이 있는지 여부 및 어린이 온라인 개인정보 보호 법률(COPPA)을 준수하는지를 `childDirected` 필드의 `true` 또는 `false`를 지정하여 지정해야 합니다. `childDirected` 필

드에 `true`을 지정함으로써 사용자는 Amazon Lex 사용이 13세 미만 및 COPPA 적용 대상 아동 전체 또는 일부를 대상으로 하는 웹 사이트, 프로그램 또는 기타 애플리케이션과 관련이 있음을 확인합니다. `childDirected` 필드에 `false`을 지정함으로써 사용자는 Amazon Lex 사용이 13세 미만 및 COPPA 적용 대상 아동 전체 또는 일부를 대상으로 하는 웹 사이트, 프로그램 또는 기타 애플리케이션과 관련이 없음을 확인합니다. 사용자는 Amazon Lex 사용이 13세 미만 및 COPPA 적용 대상 아동 전체 또는 일부를 대상으로 하는 웹 사이트, 프로그램 또는 기타 애플리케이션과 관련이 있는지 여부를 정확하게 반영하지 않는 `childDirected` 필드에 대한 기본값을 지정할 수 없습니다.

Amazon Lex의 사용이 전체 또는 부분적으로 13세 미만 어린이를 대상으로 하는 웹 사이트, 프로그램 또는 기타 애플리케이션과 관련된 경우 COPPA에 따라 검증 가능한 필수 부모 동의를 얻어야 합니다. 전체 또는 일부를 13세 미만 어린이를 대상으로 하거나 대상으로 하는 웹 사이트, 프로그램 또는 기타 애플리케이션과 관련된 Amazon Lex 사용에 관한 정보는 Amazon Lex FAQ를 참조하십시오.

유형: 부울

필수 여부: 예

clarificationPrompt

Amazon Lex는 사용자의 의도를 이해하지 못할 경우 이 메시지를 사용하여 확인을 구합니다. Amazon Lex가 설명 프롬프트를 반복해야 하는 횟수를 지정하려면 `maxAttempts` 필드를 사용하십시오. Amazon Lex가 여전히 이해하지 못하면 `abortStatement` 필드로 메시지를 보냅니다.

설명 프롬프트를 생성할 때는 사용자의 올바른 응답을 제안하는지 확인하십시오. 예를 들어 피자와 음료를 주문하는 봇의 경우 다음과 같은 설명 프롬프트를 생성할 수 있습니다. "어떻게 하시겠습니까? '음로 주문' 또는 '피자 주문'라고 말할 수 있습니다.

폴백 의도를 정의한 경우 `maxAttempts` 필드에 정의된 횟수만큼 설명 프롬프트가 반복되면 해당 의도가 호출됩니다. 자세한 내용은 AMAZON.FallbackIntent를 참조하십시오.

확인 프롬프트를 정의하지 않으면 Amazon Lex는 런타임 시 다음과 같은 세 가지 경우에 400 잘못된 요청 예외를 반환합니다.

- 후속 프롬프트 - 사용자가 후속 프롬프트에 응답하지만, 의도를 제공하지 않는 경우. 예를 들어, "오늘 다른 것을 원하시나요?"라는 후속 프롬프트에 대한 응답으로 사용자가 "예"라고 한 경우 Amazon Lex에 사용자의 의도를 파악하기 위한 확인 프롬프트가 없기 때문에 400 잘못된 요청 예외가 반환됩니다.
- Lambda 함수 - Lambda 함수를 사용하는 경우 `ElicitIntent` 대화 상자 유형을 반환합니다. Amazon Lex에 사용자의 의도를 파악하기 위한 확인 프롬프트가 없기 때문에 400 잘못된 요청 예외가 반환됩니다.

- PutSession 작업 - `PutSession` 작업을 사용 중인 경우 `ElicitIntent` 대화 유형을 전송합니다. Amazon Lex에 사용자의 의도를 파악하기 위한 확인 프롬프트가 없기 때문에 400 잘못된 요청 예외가 반환됩니다.

유형: Prompt객체

필수 여부: 아니요

createVersion

`true`로 설정하면 번호가 매겨진 새 버전의 봇이 생성됩니다. 이는 `CreateBotVersion` 오퍼레이션을 호출하는 것과 같습니다. `createVersion`를 지정하지 않으면 기본값은 `false`입니다.

유형: 부울

필수 항목 여부: 아니요

description

봇에 대한 설명입니다.

유형: 문자열

길이 제한: 최소 길이는 0. 최대 길이는 200.

필수 여부: 아니요

detectSentiment

`true`로 설정하면 감정 분석을 위해 사용자 표현이 Amazon Comprehend로 전송됩니다. `detectSentiment`를 지정하지 않으면 기본값은 `false`입니다.

유형: 부울

필수 항목 여부: 아니요

enableModelImprovements

자연어 이해 개선에 대한 액세스를 가능하게 `true`로 설정합니다.

`enableModelImprovements` 매개변수를 `true`로 설정하면 `nluIntentConfidenceThreshold` 매개변수를 사용하여 신뢰도 점수를 구성할 수 있습니다. 자세한 내용은 신뢰도 점수를 참조하세요.

특정 지역에서만 `enableModelImprovements` 매개변수를 설정할 수 있습니다. 매개변수를 `true`로 설정하면 봇이 정확도 개선에 액세스합니다.

en-US 로캘에 `enableModelImprovements` 매개변수를 `false`로 설정할 수 있는 지역은 다음과 같습니다.

- 미국 동부(버지니아 북부)(us-east-1)
- 미국 서부(오레곤)(us-west-2)
- 아시아 태평양(시드니)(ap-southeast-2)
- EU(아일랜드)(eu-west-1)

다른 지역 및 로캘에서는 `enableModelImprovements` 매개변수가 기본적으로 `true`로 설정됩니다. 이러한 지역 및 로캘에서 매개변수를 `false`로 설정하면 `ValidationException` 예외가 발생합니다.

유형: 부울

필수 항목 여부: 아니요

idleSessionTTLInSeconds

Amazon Lex가 대화에서 수집된 데이터를 유지하는 최대 시간(초) 입니다.

사용자 상호 작용 세션은 지정된 시간 동안 활성 상태로 유지됩니다. 이 시간 동안 대화가 발생하지 않으면 세션이 만료되고 Amazon Lex는 제한 시간 전에 제공된 모든 데이터를 삭제합니다.

예를 들어, 사용자가 OrderPizza 의도를 선택했지만 주문을 하는 도중에 다른 곳으로 이동했다고 가정하겠습니다. 사용자가 지정된 시간 내에 주문을 완료하지 않으면 Amazon Lex는 수집한 슬롯 정보를 삭제하므로 사용자는 처음부터 다시 시작해야 합니다.

`PutBot` 작업 요청에 `idleSessionTTLInSeconds` 요소를 포함하지 않는 경우 Amazon Lex는 기본값을 사용합니다. 요청이 기존 봇을 대체하는 경우에도 마찬가지입니다.

기본값은 300초(5분)입니다.

유형: 정수

유효한 범위: 최소값은 60. 최대값은 86,400.

필수 여부: 아니요

intents

`Intent` 객체 어레이. 각 의도는 사용자가 표현할 수 있는 명령을 나타냅니다. 예를 들어 피자 주문 봇은 OrderPizza 의도를 지원할 수 있습니다. 자세한 내용은 Amazon Lex: 작동 방식 단원을 참조하십시오.

유형: Intent 객체 배열

필수: 아니요

locale

봇의 타겟 로캘을 지정합니다. 봇에서 사용되는 모든 의도는 봇의 로캘과 호환되어야 합니다.

기본값은 `en-US`입니다.

타입: 문자열

유효 값: `de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR`

필수 사항 여부: 예

nluIntentConfidenceThreshold

PostContent 또는 PostText 응답에서 대체 의도를 반환할 때 Amazon Lex가 `AMAZON.FallbackIntent`, `AMAZON.KendraSearchIntent`, 또는 둘 다를 삽입할 임계값을 결정합니다. `AMAZON.FallbackIntent` 및 `AMAZON.KendraSearchIntent`은 봇에 대해 구성된 경우만 삽입됩니다.

다음 지역의 신뢰도 점수를 사용하려면 `enableModelImprovements` 매개변수를 `true`로 설정해야 합니다.

- 미국 동부(버지니아 북부)(us-east-1)
- 미국 서부(오레곤)(us-west-2)
- 아시아 태평양(시드니)(ap-southeast-2)
- EU(아일랜드)(eu-west-1)

다른 지역 및 로캘에서는 `enableModelImprovements` 매개변수가 기본적으로 `true`로 설정됩니다.

예를 들어, 봇이 신뢰 임계값 0.80 및 `AMAZON.FallbackIntent`으로 구성되어 있다고 가정하겠습니다. Amazon Lex는 IntentA(0.70), IntentB(0.60), IntentC(0.50) 라는 신뢰도 점수를 가진 세 가지 대체 의도를 반환합니다. `PostText` 작업에 대한 응답은 다음과 같습니다.

- AMAZON.FallbackIntent
- IntentA

- IntentB
- IntentC

유형: Double

유효한 범위: 최소값 0. 최댓값은 1.

필수 여부: 아니요

processBehavior

`processBehavior` 요소를 `BUILD`로 설정하면 Amazon Lex는 봇이 실행될 수 있도록 봇을 빌드합니다. 요소를 `SAVE`로 설정하면, Amazon Lex는 봇을 저장하지만 빌드하지는 않습니다.

값을 지정하지 않을 경우 기본값은 `BUILD`입니다.

타입: 문자열

유효 값: `SAVE | BUILD`

필수 여부: 아니요

tags

봇에 추가할 태그의 목록입니다. 봇을 생성할 때만 태그를 추가할 수 있으며 봇에서 태그를 업데이트하는 `PutBot` 작업을 사용할 수는 없습니다. 태그를 업데이트하려면 `TagResource` 작업을 사용합니다.

유형: Tag 객체의 배열

어레이 멤버: 최소 항목 수 0개. 최대 항목 수 200개.

필수 여부: 아니요

voiceId

Amazon Lex가 사용자와의 음성 상호 작용에 사용하는 Amazon Polly 음성 ID입니다. 음성에 구성된 로캘은 봇의 로캘과 일치해야 합니다. 자세한 내용은 Amazon Polly 개발자 안내서의 Amazon Polly 음성을 참조하세요.

유형: 문자열

필수사항: 아니요

응답 구문

```
HTTP/1.1 200
Content-type: application/json

{
   "abortStatement": {
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "checksum": "string",
   "childDirected": boolean,
   "clarificationPrompt": {
      "maxAttempts": number,
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "createdDate": number,
   "createVersion": boolean,
   "description": "string",
   "detectSentiment": boolean,
   "enableModelImprovements": boolean,
   "failureReason": "string",
   "idleSessionTTLInSeconds": number,
   "intents": [
      {
         "intentName": "string",
         "intentVersion": "string"
      }
   ],
   "lastUpdatedDate": number,
   "locale": "string",
```

```
    "name": "string",
    "nluIntentConfidenceThreshold": number,
    "status": "string",
    "tags": [
       {
          "key": "string",
          "value": "string"
       }
    ],
    "version": "string",
    "voiceId": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

abortStatement

Amazon Lex에서 대화를 취소하는 데 사용하는 메시지입니다. 자세한 내용은 PutBot을 참조하세요.

유형: Statement객체

checksum

생성한 봇의 체크섬.

유형: 문자열

childDirected

Amazon Lex Model Building Service로 만든 각 Amazon Lex 봇에 대해, Amazon Lex 사용이, 전체 또는 일부가, 13세 미만 어린이를 대상으로 하거나 대상으로 하는 웹 사이트, 프로그램 또는 기타 애플리케이션과 관련이 있는지 여부 및 어린이 온라인 개인정보 보호 법률(COPPA)을 준수하는지를 `childDirected` 필드의 `true` 또는 `false`를 지정하여 지정해야 합니다. `childDirected` 필드에 `true`을 지정함으로써 사용자는 Amazon Lex 사용이 13세 미만 및 COPPA 적용 대상 아동 전체 또는 일부를 대상으로 하는 웹 사이트, 프로그램 또는 기타 애플리케이션과 관련이 있음을 확인합니다. `childDirected` 필드에 `false`을 지정함으로써 사용자는 Amazon Lex 사용이 13세 미만 및 COPPA 적용 대상 아동 전체 또는 일부를 대상으로 하는 웹 사이트, 프로그램 또는 기타 애플리케이션과 관련이 없음을 확인합니다. 사용자는 Amazon Lex 사용이 13세 미만 및 COPPA 적용 대

상 아동 전체 또는 일부를 대상으로 하는 웹 사이트, 프로그램 또는 기타 애플리케이션과 관련이 있는지 여부를 정확하게 반영하지 않는 `childDirected` 필드에 대한 기본값을 지정할 수 없습니다.

Amazon Lex의 사용이 전체 또는 부분적으로 13세 미만 어린이를 대상으로 하는 웹 사이트, 프로그램 또는 기타 애플리케이션과 관련된 경우 COPPA에 따라 검증 가능한 필수 부모 동의를 얻어야 합니다. 전체 또는 일부를 13세 미만 어린이를 대상으로 하거나 대상으로 하는 웹 사이트, 프로그램 또는 기타 애플리케이션과 관련된 Amazon Lex 사용에 관한 정보는 Amazon Lex FAQ를 참조하십시오.

유형: 부울

clarificationPrompt

Amazon Lex 가 사용자의 의도를 이해하지 못할 때 사용하는 프롬프트입니다. 자세한 내용은 PutBot 단원을 참조하십시오.

유형: Prompt객체

createdDate

봇이 생성된 날짜.

유형: 타임스탬프

createVersion

봇의 새 버전이 생성된 경우 `True`. 요청에서 `createVersion` 필드를 지정하지 않은 경우 응답에서 `createVersion` 필드는 거짓으로 설정됩니다.

유형: 부울

description

봇에 대한 설명입니다.

유형: 문자열

길이 제한: 최소 길이는 0. 최대 길이는 200.

detectSentiment

봇이 감정 분석을 위해 사용자 발언을 Amazon Comprehend로 보내도록 구성된 경우 `true`. 요청에서 `detectSentiment` 필드를 지정하지 않은 경우 응답에서 `detectSentiment` 필드는 `false`입니다.

유형: 부울

enableModelImprovements

봇이 정확도 개선을 사용하는지 여부를 나타냅니다. `true`은 봇이 개선 사항을 사용하고 있음을 나타내며, 그렇지 않으면, `false`입니다.

유형: 부울

failureReason

만약 `status`이 `FAILED`라면, Amazon Lex는 봇 구축에 실패한 이유를 제공합니다.

유형: 문자열

idleSessionTTLInSeconds

Amazon Lex가 대화에서 수집된 데이터를 유지하는 최대 시간입니다. 자세한 내용은 PutBot 단원을 참조하십시오.

유형: 정수

유효한 범위: 최소값은 60. 최대값은 86,400.

intents

`Intent` 객체 어레이. 자세한 내용은 PutBot을 참조하세요.

유형: Intent 객체 배열

lastUpdatedDate

봇이 업데이트된 날짜. 리소스를 생성할 때 생성 날짜 및 최종 업데이트 날짜가 동일합니다.

유형: 타임스탬프

locale

봇의 타겟 로캘입니다.

타입: 문자열

유효 값: `de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR`

name

봇의 이름.

유형: 문자열

길이 제한: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

nluIntentConfidenceThreshold

PostContent 또는 PostText 응답에서 대체 의도를 반환할 때 Amazon Lex가 `AMAZON.FallbackIntent`, `AMAZON.KendraSearchIntent` 또는 둘 다를 삽입하는 위치를 결정하는 점수입니다. `AMAZON.FallbackIntent`는 모든 의도에 대한 신뢰도 점수가 이 값보다 낮은 경우 삽입됩니다. `AMAZON.KendraSearchIntent`은 봇에 대해 구성된 경우에만 삽입됩니다.

유형: 더블

유효한 범위: 최소값 0. 최댓값은 1.

status

`processBehavior`이 `BUILD`로 설정된 봇 생성 요청을 보내면 Amazon Lex는 `status` 응답 요소를 `BUILDING`로 설정합니다.

`READY_BASIC_TESTING` 상태에서는 슬롯 유형에서 봇의 의도와 값에 대해 구성된 표현과 정확히 일치하는 사용자 입력으로 봇을 테스트할 수 있습니다.

Amazon Lex가 봇을 빌드하지 못하는 경우에는 `status`을 `FAILED`로 설정합니다. Amazon Lex 는 `failureReason` 응답 요소에 실패 이유를 반환합니다.

`processBehavior`을 `SAVE`로 설정하면 Amazon Lex에서 상태 코드를 `NOT BUILT`로 설정합니다.

봇이 `READY` 상태가 되면 봇을 테스트하고 게시할 수 있습니다.

타입: 문자열

유효 값: `BUILDING | READY | READY_BASIC_TESTING | FAILED | NOT_BUILT`

tags

봇과 연결된 태그 목록입니다.

타입: Tag객체 배열

어레이 멤버: 최소 항목 수 0개. 최대 항목 수 200개.

version

봇의 버전. 새 봇의 경우 버전은 항상 `$LATEST`입니다.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

voiceId

Amazon Lex가 사용자와의 음성 상호 작용에 사용하는 Amazon Polly 음성 ID입니다. 자세한 내용은 PutBot 단원을 참조하십시오.

유형: 문자열

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

ConflictException

요청을 처리하는 동안 충돌이 발생했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 409

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

PreconditionFailedException

변경하려는 리소스의 체크섬이 요청의 체크섬과 일치하지 않습니다. 리소스의 체크섬을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 412

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

PutBotAlias

서비스: Amazon Lex Model Building Service

지정된 버전의 봇에 대한 별칭을 만들거나 지정된 버전의 봇에 대한 별칭을 대체합니다. 별칭이 가리키는 봇 버전을 변경하려면 별칭을 바꾸십시오. 별칭에 대한 자세한 내용은 버전 관리 및 별칭을 참조하십시오.

이 작업에는 `lex:PutBotAlias` 액션에 대한 권한이 필요합니다.

Request Syntax

```
PUT /bots/botName/aliases/name HTTP/1.1
Content-type: application/json

{
   "botVersion": "string",
   "checksum": "string",
   "conversationLogs": {
      "iamRoleArn": "string",
      "logSettings": [
         {
            "destination": "string",
            "kmsKeyArn": "string",
            "logType": "string",
            "resourceArn": "string"
         }
      ]
   },
   "description": "string",
   "tags": [
      {
         "key": "string",
         "value": "string"
      }
   ]
}
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

botName

봇의 이름.

길이 제약: 최소 길이 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

name

별칭의 이름. 이름은 대/소문자를 구분하지 않습니다.

길이 제약: 최소 길이 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

요청 본문

요청은 JSON 형식으로 다음 데이터를 받습니다.

botVersion

봇의 버전.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

필수 여부: 예

checksum

`$LATEST` 버전의 특정 개정 버전을 식별합니다.

새 봇을 만들 때는 `checksum` 필드를 비워 두십시오. 체크섬을 지정하면 `BadRequestException` 예외가 발생합니다.

봇을 업데이트하려면 checksum 필드를 해당 버전의 최신 수정 $LATEST 버전의 체크섬으로 설정하십시오. checksum 필드를 지정하지 않거나 체크섬이 $LATEST 버전과 일치하지 않으면 PreconditionFailedException 예외가 발생합니다.

유형: 문자열

필수 항목 여부: 아니요

conversationLogs

별칭의 대화 로그 설정.

유형: ConversationLogsRequest객체

필수 여부: 아니요

description

별칭에 대한 설명.

유형: 문자열

길이 제한: 최소 길이는 0. 최대 길이는 200.

필수 여부: 아니요

tags

봇에 추가할 태그의 목록입니다. 봇을 생성할 때만 태그를 추가할 수 있으며, 봇 별칭에서 태그를 업데이트하기 위해 PutBotAlias 작업을 사용할 수 없습니다. 태그를 업데이트하려면 TagResource 작업을 사용합니다.

유형: Tag 객체의 배열

어레이 멤버: 최소 항목 수 0개. 최대 항목 수 200개.

필수 여부: 아니요

응답 구문

```
HTTP/1.1 200
Content-type: application/json

{
   "botName": "string",
```

```
    "botVersion": "string",
    "checksum": "string",
    "conversationLogs": {
       "iamRoleArn": "string",
       "logSettings": [
          {
             "destination": "string",
             "kmsKeyArn": "string",
             "logType": "string",
             "resourceArn": "string",
             "resourcePrefix": "string"
          }
       ]
    },
    "createdDate": number,
    "description": "string",
    "lastUpdatedDate": number,
    "name": "string",
    "tags": [
       {
          "key": "string",
          "value": "string"
       }
    ]
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

botName

별칭이 가리키는 봇의 이름.

유형: 문자열

길이 제한: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

botVersion

별칭이 가리키는 봇의 버전.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

checksum

별칭의 현재 버전에 대한 체크섬.

유형: 문자열

conversationLogs

Amazon Lex가 별칭에 대한 대화 로그를 사용하는 방법을 결정하는 설정입니다.

유형: ConversationLogsResponse객체

createdDate

봇이 별칭이 생성된 날짜.

유형: 타임스탬프

description

별칭에 대한 설명.

유형: 문자열

길이 제한: 최소 길이는 0. 최대 길이는 200.

lastUpdatedDate

봇 별칭이 업데이트된 날짜. 리소스를 생성할 때 생성 날짜 및 최종 업데이트 날짜가 동일합니다.

유형: 타임스탬프

name

별칭의 이름입니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

tags

봇과 연결된 태그 목록.

타입: Tag객체 배열

어레이 멤버: 최소 항목 수 0개. 최대 항목 수 200개.

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

ConflictException

요청을 처리하는 동안 충돌이 발생했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 409

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

PreconditionFailedException

변경하려는 리소스의 체크섬이 요청의 체크섬과 일치하지 않습니다. 리소스의 체크섬을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 412

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

PutIntent

서비스: Amazon Lex Model Building Service

의도를 생성하거나 기존 의도를 대체합니다.

사용자와 봇 간의 상호작용을 정의하려면 하나 이상의 의도를 사용합니다. 예를 들어 피자 주문 봇의 경우 `OrderPizza` 의도를 생성할 수 있습니다.

의도를 생성하거나 기존 의도를 대체하려면 다음을 제공해야 합니다.

- 의도 이름. 예를 들어 `OrderPizza`입니다.
- 샘플 표현. 예를 들어, "피자 하나 주문할 수 있을까요?" 및 "피자를 주문하고 싶어요."
- 수집해야 할 정보. 봇이 사용자에게 요청할 정보의 슬롯 유형을 지정합니다. 날짜 또는 시간과 같은 표준 슬롯 유형을 지정하거나 피자의 크기 및 크러스트와 같은 사용자 지정 슬롯 유형을 지정할 수 있습니다.
- 의도가 이행되는 방식. Lambda 함수를 제공하거나 의도 정보를 클라이언트 애플리케이션에 반환하도록 의도를 구성할 수 있습니다. Lambda 함수를 사용하는 경우 모든 의도 정보를 사용할 수 있게 되면 Amazon Lex는 Lambda 함수를 간접적으로 호출합니다. 의도 정보를 클라이언트 애플리케이션에 반환하도록 의도를 구성하는 경우.

요청에 다음과 같은 기타 선택적 정보를 지정할 수 있습니다.

- 사용자에게 의도를 확인하도록 요청하는 확인 프롬프트입니다. 예를 들어, "피자 주문할까요?"
- 의도가 이행된 후 사용자에게 보낼 결론 설명. 예를 들어, "피자를 주문했습니다."
- 사용자에게 추가 활동을 요청하는 후속 프롬프트. 예를 들어 "피자와 함께 음료를 주문하시겠습니까?"

의도를 업데이트하기 위해 기존 의도 이름을 지정하는 경우 Amazon Lex는 의도 `$LATEST` 버전의 값을 요청의 값으로 대체합니다. Amazon Lex는 요청에서 제공하지 않은 필드를 제거합니다. 필수 필드 값을 지정하지 않으면 Amazon Lex에서 예외가 발생합니다. 의도의 `$LATEST` 버전을 업데이트하면 해당 의도 `$LATEST` 버전을 사용하는 모든 봇의 `status` 필드는 `NOT_BUILT`로 설정됩니다.

자세한 내용은 Amazon Lex: 작동 방식을 참조하세요.

이 작업에는 `lex:PutIntent` 액션에 대한 권한이 필요합니다.

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"
    },
    "inputContexts": [
        {
            "name": "string"
        }
    ],
    "kendraConfiguration": {
        "kendraIndex": "string",
        "queryFilterString": "string",
        "role": "string"
    },
    "outputContexts": [
        {
            "name": "string",
            "timeToLiveInSeconds": number,
            "turnsToLive": number
        }
    ],
    "parentIntentSignature": "string",
    "rejectionStatement": {
        "messages": [
            {
                "content": "string",
                "contentType": "string",
                "groupNumber": number
```

```
            }
        ],
        "responseCard": "string"
    },
    "sampleUtterances": [ "string" ],
    "slots": [
        {
            "defaultValueSpec": {
                "defaultValueList": [
                    {
                        "defaultValue": "string"
                    }
                ]
            },
            "description": "string",
            "name": "string",
            "obfuscationSetting": "string",
            "priority": number,
            "responseCard": "string",
            "sampleUtterances": [ "string" ],
            "slotConstraint": "string",
            "slotType": "string",
            "slotTypeVersion": "string",
            "valueElicitationPrompt": {
                "maxAttempts": number,
                "messages": [
                    {
                        "content": "string",
                        "contentType": "string",
                        "groupNumber": number
                    }
                ],
                "responseCard": "string"
            }
        }
    ]
}
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

name

의도의 이름. 이름은 대/소문자를 구분하지 않습니다.

이름은 기본 제공 의도 이름 또는 "AMAZON"이라고 표시된 기본 제공 의도 이름과 일치할 수 없습니다. 제거됨. 예를 들어 `AMAZON.HelpIntent`라는 기본 제공 의도가 있기 때문에 `HelpIntent`라는 사용자 지정 의도를 생성할 수 없습니다.

기본 제공 의도 목록은 Alexa Skills Kit의 표준 기본 제공 의도를 참조하십시오.

길이 제약: 최소 길이 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

요청 본문

요청은 JSON 형식으로 다음 데이터를 받습니다.

checksum

`$LATEST` 버전의 특정 개정 버전을 식별합니다.

새 의도를 만들 때는 `checksum` 필드를 비워 두십시오. 체크섬을 지정하면 `BadRequestException` 예외가 발생합니다.

의도를 업데이트하려면 `checksum` 필드를 해당 버전의 가장 최신 수정 `$LATEST` 버전의 체크섬으로 설정합니다. `checksum` 필드를 지정하지 않거나 체크섬이 `$LATEST` 버전과 일치하지 않으면 `PreconditionFailedException` 예외가 발생합니다.

유형: 문자열

필수 항목 여부: 아니요

conclusionStatement

Lambda 함수가 의도를 성공적으로 이행한 후 Amazon Lex가 사용자에게 전달하기를 원하는 설명.

이 요소는 `fulfillmentActivity`에서 Lambda 함수를 제공하는 경우에만 관련이 있습니다. 의도를 클라이언트 애플리케이션에 반환하는 경우 이 요소를 지정할 수 없습니다.

> **Note**
>
> followUpPrompt 및 conclusionStatement는 함께 사용할 수 없습니다. 하나만 지정할 수 있습니다.

유형: Statement객체

필수 여부: 아니요

confirmationPrompt

사용자에게 의도를 확인하라는 프롬프트를 표시합니다. 이 질문에 예 또는 아니요로 답해야 합니다.

Amazon Lex는 의도가 이행될 준비가 되었음을 사용자가 인지하도록 하기 위해 이 메시지를 사용합니다. 예를 들어 OrderPizza 의도가 있는 경우 주문을 하기 전에 올바른지 확인해야 할 수 있습니다. 사용자 질문에 간단히 응답하는 의도와 같은 다른 의도의 경우 정보를 제공하기 전에 사용자에게 확인을 요청하지 않아도 될 수 있습니다.

> **Note**
>
> rejectionStatement와 confirmationPrompt를 모두 입력하거나 둘 다 입력하지 않아야 합니다.

유형: Prompt객체

필수 여부: 아니요

createVersion

true로 설정하면 번호가 매겨진 새 버전의 의도가 생성됩니다. 이는 CreateIntentVersion 작업을 호출하는 것과 같습니다. createVersion를 지정하지 않으면 기본값은 false입니다.

유형: 부울

필수 항목 여부: 아니요

description

의도에 대한 설명.

유형: 문자열

길이 제한: 최소 길이는 0. 최대 길이는 200.

필수 여부: 아니요

dialogCodeHook

각 사용자 입력에 대해 간접 호출하도록 Lambda 함수를 지정합니다. 이 Lambda 함수를 간접 호출하여 사용자 상호 작용을 개인화할 수 있습니다.

예를 들어, 봇이 사용자를 John이라고 판단한다고 가정해 보겠습니다. Lambda 함수는 백엔드 데이터베이스에서 John의 정보를 검색하고 일부 값을 미리 채울 수 있습니다. 예를 들어, John이 글루텐 불내증이 있는 것으로 확인되면 해당하는 `GlutenIntolerant` 의도 슬롯을 참으로 설정할 수 있습니다. John의 전화번호를 찾아 해당 세션 속성을 설정할 수 있습니다.

유형: CodeHook객체

필수 여부: 아니요

followUpPrompt

Amazon Lex는 이 프롬프트를 사용하여 의도를 이행한 후 추가 활동을 요청합니다. 예를 들어 `OrderPizza` 의도가 이행된 후 사용자에게 음료를 주문하라는 프롬프트를 표시할 수 있습니다.

Amazon Lex가 취하는 조치는 다음과 같이 사용자의 응답에 따라 달라집니다.

- 사용자가 "예"라고 대답하면 봇에 대해 구성된 설명 프롬프트로 응답합니다.
- 사용자가 "예"라고 답하고 의도를 트리거하는 표현을 계속하면 해당 의도에 대한 대화가 시작됩니다.
- 사용자가 "아니요"라고 말하면 후속 조치 프롬프트에 대해 구성된 거부 문구로 응답합니다.
- 표현을 인식하지 못하면 후속 조치 프롬프트가 다시 반복됩니다.

`followUpPrompt` 필드와 `conclusionStatement` 필드는 상호 배타적입니다. 하나만 지정할 수 있습니다.

유형: FollowUpPrompt객체

필수 여부: 아니요

fulfillmentActivity

필수 사항입니다. 의도가 이행되는 방식을 설명합니다. 예를 들어, 사용자가 피자 주문에 필요한 모든 정보를 제공한 후 `fulfillmentActivity`이 봇이 현지 피자 가게에 주문하는 방법을 정의합니다.

모든 의도 정보를 클라이언트 애플리케이션에 반환하거나, 의도(예: 피자 가게에서 주문)를 처리할 수 있는 Lambda 함수를 간접 호출하도록 Amazon Lex를 구성할 수 있습니다 .

유형: FulfillmentActivity객체

필수 여부: 아니요

inputContexts

Amazon Lex가 사용자와의 대화에서 의도를 선택하기 위해 활성화되어야 하는 컨텍스트를 목록화하는 `InputContext` 객체 배열입니다.

타입: InputContext객체 배열

배열 멤버: 최소 항목 수 0개. 최대 항목 수 5개.

필수 여부: 아니요

kendraConfiguration

`AMAZON.KendraSearchIntent` 의도를 사용하여 Amazon Kendra 인덱스에 연결하는 데 필요한 구성 정보입니다. 자세한 내용은 AMAZON.KendraSearchIntent를 참조하세요.

유형: KendraConfiguration객체

필수 여부: 아니요

outputContexts

의도가 이행될 때 의도가 활성화하는 컨텍스트를 목록화하는 `OutputContext` 객체의 배열입니다.

타입: OutputContext객체 배열

배열 멤버: 최소 항목 수는 0개. 최대 항목 수 10개.

필수 여부: 아니요

parentIntentSignature

이 의도의 기반이 되는 기본 제공 의도의 고유 식별자입니다. 의도의 서명을 찾으려면 Alexa Skills Kit의 표준 기본 제공 의도를 참조하십시오.

유형: 문자열

필수 항목 여부: 아니요

rejectionStatement

사용자가 `confirmationPrompt`에 정의된 질문에 "아니요"라고 답하면 Amazon Lex는 의도가 취소되었음을 확인하기 위해 이 문장으로 답합니다.

> **Note**
>
> `rejectionStatement`와 `confirmationPrompt`를 모두 입력하거나 둘 다 입력하지 않아야 합니다.

유형: Statement객체

필수 여부: 아니요

sampleUtterances

사용자가 의도를 알리기 위해 말할 수 있는 표현(문자열)의 목록입니다. 예: "{PizzaSize} 피자를 원해요", "{수량} {PizzaSize} 피자를 주문하세요."

각 표현에서 슬롯 이름은 중괄호로 묶여 있습니다.

유형: 문자열 배열

배열 구성원: 최소수는 0개입니다. 최대 항목 수 1,500개.

길이 제약 조건: 최소 길이는 1입니다. 최대 길이는 200.

필수 여부: 아니요

slots

의도 슬롯 배열. Amazon Lex는 런타임 시 슬롯에 정의된 프롬프트를 사용하여 사용자로부터 필요한 슬롯 값을 끌어냅니다. 자세한 내용은 Amazon Lex: 작동 방식 단원을 참조하십시오.

타입: Slot객체 배열

배열 구성원: 최소수는 0개입니다. 최대 항목 수 100개.

필수 항목 여부: 아니요

응답 구문

```
HTTP/1.1 200
Content-type: application/json

{
   "checksum": "string",
   "conclusionStatement": {
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "confirmationPrompt": {
      "maxAttempts": number,
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "createdDate": number,
   "createVersion": boolean,
   "description": "string",
   "dialogCodeHook": {
      "messageVersion": "string",
      "uri": "string"
   },
   "followUpPrompt": {
      "prompt": {
         "maxAttempts": number,
         "messages": [
            {
               "content": "string",
               "contentType": "string",
               "groupNumber": number
            }
```

```
            ],
            "responseCard": "string"
        },
        "rejectionStatement": {
            "messages": [
                {
                    "content": "string",
                    "contentType": "string",
                    "groupNumber": number
                }
            ],
            "responseCard": "string"
        }
    },
    "fulfillmentActivity": {
        "codeHook": {
            "messageVersion": "string",
            "uri": "string"
        },
        "type": "string"
    },
    "inputContexts": [
        {
            "name": "string"
        }
    ],
    "kendraConfiguration": {
        "kendraIndex": "string",
        "queryFilterString": "string",
        "role": "string"
    },
    "lastUpdatedDate": number,
    "name": "string",
    "outputContexts": [
        {
            "name": "string",
            "timeToLiveInSeconds": number,
            "turnsToLive": number
        }
    ],
    "parentIntentSignature": "string",
    "rejectionStatement": {
        "messages": [
            {
```

```
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "sampleUtterances": [ "string" ],
   "slots": [
      {
         "defaultValueSpec": {
            "defaultValueList": [
               {
                  "defaultValue": "string"
               }
            ]
         },
         "description": "string",
         "name": "string",
         "obfuscationSetting": "string",
         "priority": number,
         "responseCard": "string",
         "sampleUtterances": [ "string" ],
         "slotConstraint": "string",
         "slotType": "string",
         "slotTypeVersion": "string",
         "valueElicitationPrompt": {
            "maxAttempts": number,
            "messages": [
               {
                  "content": "string",
                  "contentType": "string",
                  "groupNumber": number
               }
            ],
            "responseCard": "string"
         }
      }
   ],
   "version": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

checksum

생성 또는 업데이트된 `$LATEST` 버전의 의도 체크섬.

유형: 문자열

conclusionStatement

`fulfillmentActivity` 의도에 지정된 Lambda 함수가 의도를 이행하면 Amazon Lex는 이 설명을 사용자에게 전달합니다.

유형: Statement객체

confirmationPrompt

의도에 정의된 경우 Amazon Lex는 사용자에게 의도를 이행하기 전에 의도를 확인하라는 메시지를 표시합니다.

유형: Prompt객체

createdDate

의도가 생성된 날짜입니다.

유형: 타임스탬프

createVersion

의도의 새 버전이 생성된 경우 `True`. 요청에서 `createVersion` 필드를 지정하지 않은 경우 응답에서 `createVersion` 필드는 거짓으로 설정됩니다.

유형: 부울

description

의도에 대한 설명.

유형: 문자열

길이 제한: 최소 길이는 0. 최대 길이는 200.

dialogCodeHook

의도가 정의된 경우 Amazon Lex는 각 사용자 입력에 대해 이 Lambda 함수를 간접 호출합니다.

유형: CodeHook객체

followUpPrompt

의도에 정의된 경우 Amazon Lex는 이 프롬프트를 사용하여 의도가 충족된 후 추가 사용자 활동을 요청합니다.

유형: FollowUpPrompt객체

fulfillmentActivity

의도에 정의된 경우 Amazon Lex는 사용자가 의도에 필요한 모든 정보를 제공한 후 이 Lambda 함수를 간접 호출하여 의도를 수행합니다.

유형: FulfillmentActivity객체

inputContexts

Amazon Lex가 사용자와의 대화에서 의도를 선택하기 위해 활성화되어야 하는 컨텍스트를 목록화하는 `InputContext` 객체 배열입니다.

타입: InputContext객체 배열

배열 멤버: 최소 항목 수 0개. 최대 항목 수 5개.

kendraConfiguration

Amazon Kendra 인덱스에 연결하고 `AMAZON.KendraSearchIntent` 의도를 사용하는 데 필요한 구성 정보입니다.

유형: KendraConfiguration객체

lastUpdatedDate

의도가 업데이트된 날짜. 리소스를 생성할 때 생성 날짜 및 최종 업데이트 날짜가 동일합니다.

유형: 타임스탬프

name

의도의 이름.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

outputContexts

의도가 이행될 때 의도가 활성화하는 컨텍스트를 목록화하는 `OutputContext` 객체의 배열입니다.

타입: OutputContext객체 배열

배열 멤버: 최소 항목 수는 0개. 최대 항목 수는 10개.

parentIntentSignature

이 의도의 기반이 되는 기본 제공 의도의 고유 식별자입니다.

유형: 문자열

rejectionStatement

사용자가 `confirmationPrompt`에 정의된 질문에 "아니요"라고 답하면 Amazon Lex는 의도가 취소되었음을 확인하기 위해 이 문장으로 답합니다.

유형: Statement객체

sampleUtterances

의도에 맞게 구성된 샘플 표현 배열.

유형: 문자열 배열

배열 구성원: 최소수는 0개입니다. 최대 항목 수 1,500개.

길이 제약 조건: 최소 길이는 1입니다. 최대 길이는 200입니다.

slots

의도에 맞게 구성된 의도 슬롯 배열.

타입: Slot객체 배열

배열 구성원: 최소수는 0개입니다. 최대 항목 수 100개.

version

의도의 버전. 새 의도의 경우 버전은 항상 `$LATEST`입니다.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

ConflictException

요청을 처리하는 동안 충돌이 발생했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 409

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

PreconditionFailedException

변경하려는 리소스의 체크섬이 요청의 체크섬과 일치하지 않습니다. 리소스의 체크섬을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 412

참고

언어별 AWS SDKs

- AWS Command Line Interface

- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

PutSlotType

서비스: Amazon Lex Model Building Service

사용자 지정 슬롯 유형을 생성하거나 기존 사용자 지정 슬롯 유형을 교체합니다.

사용자 정의 슬롯 유형을 생성하려면 슬롯 유형의 이름과 이 유형의 슬롯이 가정할 수 있는 값인 열거값 세트를 지정합니다. 자세한 내용은 Amazon Lex: 작동 방식을 참조하세요.

기존 슬롯 유형의 이름을 지정하는 경우, 요청의 필드가 슬롯 유형의 `$LATEST` 버전의 기존 값을 대체합니다. Amazon Lex는 요청에서 제공하지 않은 필드를 제거합니다. 필수 필드 값을 지정하지 않으면 Amazon Lex에서 예외가 발생합니다. 슬롯 유형의 `$LATEST` 버전을 업데이트할 때 봇이 슬롯 유형을 포함하는 의도 `$LATEST` 버전을 사용하는 경우 봇의 `status` 필드는 `NOT_BUILT`로 설정됩니다.

이 작업에는 `lex:PutSlotType` 액션에 대한 권한이 필요합니다.

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"
      }
   ],
   "parentSlotTypeSignature": "string",
   "slotTypeConfigurations": [
      {
         "regexConfiguration": {
            "pattern": "string"
         }
      }
   ],
   "valueSelectionStrategy": "string"
}
```

URI 요청 파라미터

다음의 URI 파라미터를 사용하여 요청합니다.

name

슬롯 유형의 이름. 이름은 대/소문자를 구분하지 않습니다.

이름은 기본 제공 의도 이름 또는 "AMAZON"이라고 표시된 기본 제공 의도 이름과 일치할 수 없습니다. 제거됨. 예를 들어 `AMAZON.DATE`라는 기본 제공 슬롯 유형이 있기 때문에 `DATE` 라는 사용자 지정 슬롯 유형을 생성할 수 없습니다.

사용 가능한 기본 제공 슬롯 유형의 목록은 Alexa Skills Kit의 슬롯 유형 참고 자료를 참조하십시오.

길이 제약: 최소 길이 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

요청 본문

요청은 JSON 형식으로 다음 데이터를 받습니다.

checksum

`$LATEST` 버전의 특정 개정 버전을 식별합니다.

새 봇을 만들 때는 `checksum` 필드를 비워 두십시오. 체크섬을 지정하면 `BadRequestException` 예외가 발생합니다.

봇을 업데이트하려면 `checksum` 필드를 해당 버전의 `$LATEST` 버전의 최신 개정의 체크섬으로 설정하십시오. `checksum` 필드를 지정하지 않거나 체크섬이 `$LATEST` 버전과 일치하지 않으면 `PreconditionFailedException` 예외가 발생합니다.

유형: 문자열

필수 항목 여부: 아니요

createVersion

`true`로 설정하면 번호가 매겨진 새 버전의 슬롯 유형이 생성됩니다. 이는 `CreateSlotTypeVersion` 작업을 호출하는 것과 같습니다. `createVersion`를 지정하지 않으면 기본값은 `false`입니다.

유형: 부울

필수 항목 여부: 아니요

description

슬롯 유형에 대한 설명.

유형: 문자열

길이 제한: 최소 길이는 0. 최대 길이는 200.

필수 여부: 아니요

enumerationValues

슬롯 유형이 사용할 수 있는 값을 정의하는 `EnumerationValue` 객체의 목록. 각 값에는 슬롯에 대해 확인되는 값에 대해 기계 학습 모델을 훈련하는 데 도움이 되는 추가 값인 `synonyms` 목록이 있을 수 있습니다.

정규 표현식 슬롯 유형에는 열거형 값이 필요하지 않습니다. 다른 모든 슬롯 유형에는 열거값 목록이 필요합니다.

Amazon Lex는 슬롯 값을 확인할 때 슬롯에 사용할 수 있는 값을 최대 5개까지 포함하는 확인 목록을 생성합니다. Lambda 함수를 사용하는 경우 이 확인 목록이 함수에 전달됩니다. Lambda 함수를 사용하지 않는 경우 사용자가 입력한 값을 반환하거나 확인 목록의 첫 번째 값을 슬롯 값으로 반환하도록 선택할 수 있습니다. `valueSelectionStrategy` 필드는 사용할 옵션을 나타냅니다.

타입: EnumerationValue객체 배열

배열 멤버: 최소 항목 수 0개. 최대 항목 수 10,000개.

필수 여부: 아니요

parentSlotTypeSignature

이 슬롯 유형의 상위 슬롯 유형으로 사용되는 기본 제공 슬롯 유형. 상위 슬롯 유형을 정의하면 새 슬롯 유형은 상위 슬롯 유형과 구성이 모두 동일합니다.

`AMAZON.AlphaNumeric`만 지원됩니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^((AMAZON\.)_?|[A-Za-z]_?)+`

필수 여부: 아니요

slotTypeConfigurations

상위 기본 제공 슬롯 유형을 확장하는 구성 정보. 구성이 상위 슬롯 유형의 설정에 추가됩니다.

타입: SlotTypeConfiguration객체 배열

배열 멤버: 최소 항목 수는 0개. 최대 항목 수 10개.

필수 여부: 아니요

valueSelectionStrategy

Amazon Lex가 슬롯 유형 값을 반환하는 데 사용하는 슬롯 확인 전략을 결정합니다. 다음 값 중 하나로 필드를 설정할 수 있습니다.

- `ORIGINAL_VALUE` - 사용자 값이 슬롯 값과 유사한 경우 사용자가 입력한 값을 반환합니다.
- `TOP_RESOLUTION` - 슬롯에 대한 확인 목록이 있는 경우 확인 목록의 첫 번째 값을 슬롯 유형 값으로 반환합니다. 확인 목록이 없으면 null이 반환됩니다.

`valueSelectionStrategy`를 지정하지 않으면 기본값은 `ORIGINAL_VALUE`입니다.

타입: 문자열

유효 값: `ORIGINAL_VALUE | TOP_RESOLUTION`

필수 항목 여부: 아니요

응답 구문

```
HTTP/1.1 200
Content-type: application/json

{
   "checksum": "string",
   "createdDate": number,
   "createVersion": boolean,
   "description": "string",
   "enumerationValues": [
      {
         "synonyms": [ "string" ],
```

```
            "value": "string"
        }
    ],
    "lastUpdatedDate": number,
    "name": "string",
    "parentSlotTypeSignature": "string",
    "slotTypeConfigurations": [
        {
            "regexConfiguration": {
                "pattern": "string"
            }
        }
    ],
    "valueSelectionStrategy": "string",
    "version": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

checksum

슬롯 유형의 `$LATEST` 버전의 체크섬.

유형: 문자열

createdDate

슬롯 유형이 생성된 날짜.

유형: 타임스탬프

createVersion

슬롯 유형의 새 버전이 생성된 경우 `True`. 요청에서 `createVersion` 필드를 지정하지 않은 경우 응답에서 `createVersion` 필드는 거짓으로 설정됩니다.

유형: 부울

description

슬롯 유형에 대한 설명.

유형: 문자열

길이 제한: 최소 길이는 0. 최대 길이는 200.

enumerationValues

슬롯 유형이 사용할 수 있는 값을 정의하는 `EnumerationValue` 객체의 목록.

타입: EnumerationValue객체 배열

배열 멤버: 최소 항목 수 0개. 최대 항목 수 10,000개.

lastUpdatedDate

슬롯 유형이 업데이트된 날짜. 슬롯 유형을 생성할 때 생성 날짜 및 최종 업데이트 날짜가 동일합니다.

유형: 타임스탬프

name

슬롯 유형의 이름.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

parentSlotTypeSignature

이 슬롯 유형의 상위 슬롯 유형으로 사용되는 기본 제공 슬롯 유형.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^((AMAZON\.)_?|[A-Za-z]_?)+`

slotTypeConfigurations

상위 기본 제공 슬롯 유형을 확장하는 구성 정보.

타입: SlotTypeConfiguration객체 배열

배열 멤버: 최소 항목 수는 0개. 최대 항목 수는 10개.

valueSelectionStrategy

Amazon Lex가 슬롯 값을 결정하는 데 사용하는 슬롯 확인 전략입니다. 자세한 내용은 PutSlotType 단원을 참조하십시오.

타입: 문자열

유효 값: `ORIGINAL_VALUE | TOP_RESOLUTION`

version

슬롯 유형의 버전. 새 슬롯 유형의 경우 버전은 항상 `$LATEST`입니다.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

ConflictException

요청을 처리하는 동안 충돌이 발생했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 409

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

PreconditionFailedException

변경하려는 리소스의 체크섬이 요청의 체크섬과 일치하지 않습니다. 리소스의 체크섬을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 412

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

StartImport

서비스: Amazon Lex Model Building Service

Amazon Lex로 리소스를 가져오는 작업을 시작합니다.

Request Syntax

```
POST /imports/ HTTP/1.1
Content-type: application/json

{
   "mergeStrategy": "string",
   "payload": blob,
   "resourceType": "string",
   "tags": [
      {
         "key": "string",
         "value": "string"
      }
   ]
}
```

URI 요청 파라미터

요청은 URI 파라미터를 사용하지 않습니다.

요청 본문

요청은 JSON 형식으로 다음 데이터를 받습니다.

mergeStrategy

이름이 같은 기존 리소스가 있는 경우 작업에서 수행해야 하는 `StartImport` 작업을 지정합니다.

- FAIL_ON_CONFLICT - 가져오기 파일의 리소스와 기존 리소스 간의 첫 번째 충돌 시 가져오기 작업이 중지됩니다. 충돌을 일으키는 리소스 이름은 `GetImport` 작업에 대한 응답 `failureReason` 필드에 있습니다.

  OVERWRITE_LATEST - 기존 리소스와 충돌이 있더라도 가져오기 작업이 진행됩니다. 기존 리소스의 $LASTEST 버전을 가져오기 파일의 데이터로 덮어씁니다.

타입: 문자열

유효 값: `OVERWRITE_LATEST | FAIL_ON_CONFLICT`

필수 사항 여부: 예

payload

바이너리 포맷의 zip 아카이브입니다. 아카이브에는 가져올 리소스가 들어 있는 JSON 파일 한 개가 포함되어야 합니다. 리소스는 `resourceType` 필드에 지정된 유형과 일치해야 합니다.

타입: Base64로 인코딩된 이진 데이터 객체

필수 여부: 예

resourceType

내보내기 할 리소스 매핑의 유형을 지정합니다. 또한 각 리소스는 해당 리소스가 종속된 모든 리소스를 내보냅니다.

- 봇은 종속 의도를 내보냅니다.
- 의도는 종속 슬롯 유형을 내보냅니다.

타입: 문자열

유효 값: `BOT | INTENT | SLOT_TYPE`

필수 사항 여부: 예

tags

봇에 추가할 태그의 목록입니다. 봇을 가져올 때만 태그를 추가할 수 있으며 의도 또는 슬롯 유형에는 태그를 추가할 수 없습니다.

타입: Tag객체 배열

어레이 멤버: 최소 항목 수 0개. 최대 항목 수 200개.

필수 여부: 아니요

응답 구문

```
HTTP/1.1 201
Content-type: application/json

{
   "createdDate": number,
```

```
    "importId": "string",
    "importStatus": "string",
    "mergeStrategy": "string",
    "name": "string",
    "resourceType": "string",
    "tags": [
       {
          "key": "string",
          "value": "string"
       }
    ]
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 201 응답을 다시 전송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

createdDate

가져오기 작업이 요청된 날짜와 시간에 대한 타임스탬프입니다.

유형: 타임스탬프

importId

특정 가져오기 작업의 식별자입니다.

유형: 문자열

importStatus

가져오기 작업의 상태입니다. 상태가 `FAILED` 인 경우 `GetImport` 작업을 사용하여 실패 원인을 확인할 수 있습니다.

타입: 문자열

유효 값: `IN_PROGRESS | COMPLETE | FAILED`

mergeStrategy

병합 충돌 발생 시 실행할 작업입니다.

타입: 문자열

유효 값: `OVERWRITE_LATEST | FAIL_ON_CONFLICT`

name

가져오기 작업의 이름입니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `[a-zA-Z_]+`

resourceType

가져올 리소스의 유형.

타입: 문자열

유효 값: `BOT | INTENT | SLOT_TYPE`

tags

가져온 봇에 추가된 태그의 목록입니다.

타입: Tag객체 배열

어레이 멤버: 최소 항목 수 0개. 최대 항목 수 200개.

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

참고 항목

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

StartMigration

서비스: Amazon Lex Model Building Service

Amazon Lex V1에서 Amazon Lex V2로 봇을 마이그레이션하기 시작합니다. Amazon Lex V2의 새 기능을 활용하려고 할 때 봇을 마이그레이션하십시오.

자세한 내용은 Amazon Lex 개발자 가이드의 봇 마이그레이션을 참조하십시오.

Request Syntax

```
POST /migrations HTTP/1.1
Content-type: application/json

{
   "migrationStrategy": "string",
   "v1BotName": "string",
   "v1BotVersion": "string",
   "v2BotName": "string",
   "v2BotRole": "string"
}
```

URI 요청 파라미터

요청은 URI 파라미터를 사용하지 않습니다.

요청 본문

요청은 JSON 형식으로 다음 데이터를 받습니다.

migrationStrategy

마이그레이션을 수행하는 데 사용된 전략.

- `CREATE_NEW` - 새 Amazon Lex V2 봇을 만들고 Amazon Lex V1 봇을 새 봇으로 마이그레이션합니다.
- `UPDATE_EXISTING` - 기존 Amazon Lex V2 봇 메타데이터와 마이그레이션 중인 로캘을 덮어씁니다. Amazon Lex V2 봇의 다른 로캘은 변경되지 않습니다. 로캘이 존재하지 않는 경우 Amazon Lex V2 봇에 새 로캘이 생성됩니다.

타입: 문자열

유효 값: `CREATE_NEW | UPDATE_EXISTING`

필수 사항 여부: 예

v1BotName

Amazon Lex V2로 마이그레이션하려는 아마존 Lex V1 봇의 이름.

유형: 문자열

길이 제한: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 예

v1BotVersion

Amazon Lex V2로 마이그레이션할 수 있는 봇의 버전입니다. 번호가 매겨진 모든 `$LATEST` 버전뿐만 아니라 버전도 마이그레이션할 수 있습니다.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

필수 여부: 예

v2BotName

Amazon Lex V2로 마이그레이션하려는 아마존 Lex V1 봇의 이름.

- Amazon Lex V2 봇이 없는 경우 `CREATE_NEW` 마이그레이션 전략을 사용해야 합니다.
- Amazon Lex V2 봇이 있는 경우 `UPDATE_EXISTING` 마이그레이션 전략을 사용하여 Amazon Lex V2 봇의 콘텐츠를 변경해야 합니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([0-9a-zA-Z][_-]?)+$`

필수 여부: 예

v2BotRole

Amazon Lex가 Amazon Lex V2 봇을 실행하는 데 사용하는 IAM 역할.

유형: 문자열

길이 제약: 최소 길이는 20. 최대 길이는 2,048.

패턴: `^arn:[\w\-]+:iam::[\d]{12}:role/.+$`

필수 여부: 예

응답 구문

```
HTTP/1.1 202
Content-type: application/json

{
   "migrationId": "string",
   "migrationStrategy": "string",
   "migrationTimestamp": number,
   "v1BotLocale": "string",
   "v1BotName": "string",
   "v1BotVersion": "string",
   "v2BotId": "string",
   "v2BotRole": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 202 응답을 다시 전송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

migrationId

Amazon Lex가 마이그레이션에 할당한 고유 식별자입니다.

유형: 문자열

길이 제약 조건: 고정 길이는 10.

패턴: `^[0-9a-zA-Z]+$`

migrationStrategy

마이그레이션을 수행하는 데 사용된 전략.

타입: 문자열

유효 값: `CREATE_NEW | UPDATE_EXISTING`

migrationTimestamp

마이그레이션이 시작된 날짜와 시간.

유형: 타임스탬프

v1BotLocale

Amazon Lex V1 봇에 사용되는 로캘입니다.

타입: 문자열

유효 값: `de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR`

v1BotName

Amazon Lex V2로 마이그레이션하려는 아마존 Lex V1 봇의 이름.

유형: 문자열

길이 제한: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

v1BotVersion

Amazon Lex V2로 마이그레이션할 수 있는 봇의 버전입니다.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

v2BotId

Amazon Lex V2 봇의 고유 식별자입니다.

유형: 문자열

길이 제약 조건: 고정 길이는 10.

패턴: `^[0-9a-zA-Z]+$`

v2BotRole

Amazon Lex가 Amazon Lex V2 봇을 실행하는 데 사용하는 IAM 역할.

유형: 문자열

길이 제약: 최소 길이는 20. 최대 길이는 2,048.

패턴: `^arn:[\w\-]+:iam::[\d]{12}:role/.+$`

오류

AccessDeniedException

IAM 사용자 또는 역할에는 봇을 마이그레이션하는 데 필요한 Amazon Lex V2 API를 호출할 권한이 없습니다.

HTTP 상태 코드: 403

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go용 SDK v2
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

TagResource

서비스: Amazon Lex Model Building Service

지정된 태그를 지정된 리소스에 추가합니다. 태그 키가 이미 존재하는 경우 기존 값이 새 값으로 대체됩니다.

Request Syntax

```
POST /tags/resourceArn HTTP/1.1
Content-type: application/json

{
   "tags": [
      {
         "key": "string",
         "value": "string"
      }
   ]
}
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

resourceArn

태그할 봇, 봇 별칭 또는 봇 채널의 Amazon 리소스 이름(ARN) 입니다.

길이 제약: 최소 길이는 1. 최대 길이는 1011입니다.

필수 여부: 예

요청 본문

요청은 JSON 형식으로 다음 데이터를 받습니다.

tags

리소스에 추가할 태그 키 목록입니다. 태그 키가 이미 존재하는 경우 기존 값이 새 값으로 대체됩니다.

타입: Tag객체 배열

어레이 멤버: 최소 항목 수 0개. 최대 항목 수 200개.

필수 항목 여부: 예

응답 구문

```
HTTP/1.1 204
```

Response Elements

액션이 성공하면 해당 서비스는 빈 HTTP 본문과 함께 HTTP 204 응답을 되돌려줍니다.

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

ConflictException

요청을 처리하는 동안 충돌이 발생했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 409

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

UntagResource

서비스: Amazon Lex Model Building Service

봇, 봇 별칭 또는 봇 채널에서 태그를 제거합니다.

Request Syntax

```
DELETE /tags/resourceArn?tagKeys=tagKeys HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

resourceArn

태그를 제거할 리소스의 Amazon 리소스 이름(ARN)입니다.

길이 제약: 최소 길이는 1. 최대 길이는 1011입니다.

필수 여부: 예

tagKeys

리소스에서 제거할 태그 키의 목록입니다. 리소스에 태그 키가 없으면 무시됩니다.

배열 멤버: 최소 항목 수는 0개. 최대 항목 수 200개.

길이 제약: 최소 길이 1. 최대 길이는 128.

필수 여부: 예

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

액션이 성공하면 해당 서비스는 빈 HTTP 본문과 함께 HTTP 204 응답을 되돌려줍니다.

Errors

BadRequestException

요청이 제대로 구성되지 않았습니다. 예를 들어, 값이 유효하지 않거나 필수 필드가 누락된 경우입니다. 필드 값을 확인한 후 다시 시도하세요.

HTTP 상태 코드: 400

ConflictException

요청을 처리하는 동안 충돌이 발생했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 409

InternalFailureException

내부 Amazon Lex 오류가 발생했습니다. 요청을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

요청이 한도를 초과했습니다. 다시 요청해 보세요.

HTTP 상태 코드: 429

NotFoundException

요청에 지정된 리소스를 찾을 수 없습니다. 리소스를 확인한 후 다시 시도하세요.

HTTP 상태 코드: 404

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3

- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

Amazon Lex 런타임 서비스

다음 작업이 Amazon Lex 런타임 서비스에서 지원됩니다.

- DeleteSession
- GetSession
- PostContent
- PostText
- PutSession

DeleteSession

서비스: Amazon Lex Runtime Service

지정된 봇, 별칭 및 사용자 ID에 대한 세션 정보를 제거합니다.

Request Syntax

```
DELETE /bot/botName/alias/botAlias/user/userId/session HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

botAlias

세션 데이터를 포함하는 봇에 사용되는 별칭입니다.

필수 여부: 예

botName

세션 데이터가 들어있는 봇의 명칭.

필수 여부: 예

userId

세션 데이터와 연결된 사용자의 식별자입니다.

길이 제약: 최소 길이는 2. 최대 길이는 100.

패턴: `[0-9a-zA-Z._:-]+`

필수 여부: 예

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
```

```
    "botAlias": "string",
    "botName": "string",
    "sessionId": "string",
    "userId": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

botAlias

세션 데이터와 연결된 봇에 사용되는 별칭입니다.

유형: 문자열

botName

세션 데이터와 연결된 봇 이름.

유형: 문자열

sessionId

세션의 고유 식별자입니다.

유형: 문자열

userId

클라이언트 애플리케이션 사용자의 ID입니다.

유형: 문자열

길이 제약: 최소 길이는 2. 최대 길이는 100.

패턴: `[0-9a-zA-Z._:-]+`

오류

BadRequestException

요청 검증이 실패했거나, 컨텍스트에 사용 가능한 메시지가 없거나, 봇 빌드가 실패했거나, 아직 진행 중이거나, 빌드되지 않은 변경 사항이 포함되어 있습니다.

HTTP 상태 코드: 400

ConflictException

두 클라이언트가 동일한 AWS 계정, Amazon Lex 봇 및 사용자 ID를 사용하고 있습니다.

HTTP 상태 코드: 409

InternalFailureException

내부 서비스 오류. 호출을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

제한 초과함.

HTTP 상태 코드: 429

NotFoundException

참조된 리소스(예: Amazon Lex 봇 또는 별칭)를 찾을 수 없습니다.

HTTP 상태 코드: 404

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

GetSession

서비스: Amazon Lex Runtime Service

지정된 봇, 별칭 및 사용자 ID에 대한 세션 정보를 반환합니다.

Request Syntax

```
GET /bot/botName/alias/botAlias/user/userId/session/?
checkpointLabelFilter=checkpointLabelFilter HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

botAlias

세션 데이터를 포함하는 봇에 사용되는 별칭입니다.

필수 여부: 예

botName

세션 데이터가 들어있는 봇의 명칭.

필수 여부: 예

checkpointLabelFilter

`recentIntentSummaryView` 구조에서 반환된 의도를 필터링하는 데 사용되는 문자열입니다.

필터를 지정하면 `checkpointLabel` 필드가 해당 문자열로 설정된 의도만 반환됩니다.

길이 제약: 최소 길이는 1. 최대 길이는 255.

패턴: `[a-zA-Z0-9-]+`

userId

클라이언트 애플리케이션 사용자의 ID입니다. Amazon Lex는 이를 사용하여 사용자와 봇의 대화를 식별합니다.

길이 제약: 최소 길이는 2. 최대 길이는 100.

패턴: `[0-9a-zA-Z._:-]+`

필수 여부: 예

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "activeContexts": [
      {
         "name": "string",
         "parameters": {
            "string" : "string"
         },
         "timeToLive": {
            "timeToLiveInSeconds": number,
            "turnsToLive": number
         }
      }
   ],
   "dialogAction": {
      "fulfillmentState": "string",
      "intentName": "string",
      "message": "string",
      "messageFormat": "string",
      "slots": {
         "string" : "string"
      },
      "slotToElicit": "string",
      "type": "string"
   },
   "recentIntentSummaryView": [
      {
         "checkpointLabel": "string",
         "confirmationStatus": "string",
         "dialogActionType": "string",
         "fulfillmentState": "string",
         "intentName": "string",
         "slots": {
```

```
            "string" : "string"
         },
         "slotToElicit": "string"
      }
   ],
   "sessionAttributes": {
      "string" : "string"
   },
   "sessionId": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

activeContexts

세션에 대해 활성화된 컨텍스트 목록. 의도가 이행될 때 또는 `PostContent`, `PostText`, 또는 `PutSession` 작업을 호출하여 컨텍스트를 설정할 수 있습니다.

컨텍스트를 사용하여 의도를 추적할 수 있는 의도를 제어하거나 애플리케이션 작업을 수정할 수 있습니다.

타입: ActiveContext객체 배열

배열 항목: 최소 항목 수는 0개. 최대 항목 수는 20개.

dialogAction

봇의 현재 상태를 설명합니다.

유형: DialogAction객체

recentIntentSummaryView

세션에서 사용된 의도에 대한 정보 배열. 배열에는 최대 세 개의 요약이 포함될 수 있습니다. 세션에서 세 개 이상의 의도를 사용하는 경우 `recentIntentSummaryView` 작업에는 마지막으로 사용한 세 개의 의도에 대한 정보가 포함됩니다.

요청에서 `checkpointLabelFilter` 파라미터를 설정하는 경우 배열에는 지정된 레이블의 의도만 포함됩니다.

타입: IntentSummary객체 배열

배열 구성원: 최소수는 0개입니다. 최대 항목 수는 3개.

sessionAttributes

세션별 컨텍스트 정보를 나타내는 키/값 페어의 맵입니다. Amazon Lex와 클라이언트 애플리케이션 간에 전달되는 요청별 정보를 포함합니다.

유형: 문자열 간 맵

sessionId

세션에 대한 고유 식별자입니다.

유형: 문자열

오류

BadRequestException

요청 검증이 실패했거나, 컨텍스트에 사용 가능한 메시지가 없거나, 봇 빌드가 실패했거나, 아직 진행 중이거나, 빌드되지 않은 변경 사항이 포함되어 있습니다.

HTTP 상태 코드: 400

InternalFailureException

내부 서비스 오류. 호출을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

제한 초과함.

HTTP 상태 코드: 429

NotFoundException

참조된 리소스(예: Amazon Lex 봇 또는 별칭)를 찾을 수 없습니다.

HTTP 상태 코드: 404

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go용 SDK v2
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

PostContent

서비스: Amazon Lex Runtime Service

Amazon Lex에 사용자 입력(텍스트 또는 스피치)을 전송합니다. 클라이언트는 이 API를 사용하여 런타임에 Amazon Lex에 텍스트 및 오디오 요청을 보냅니다. Amazon Lex는 봇용으로 구축한 기계 학습 모델을 사용하여 사용자 입력을 해석합니다.

`PostContent` 작업은 8kHz 및 16kHz에서 오디오 입력을 지원합니다. 전화 오디오 애플리케이션에서 8kHz 오디오를 사용하여 음성 인식 정확도를 높일 수 있습니다.

이에 대한 응답으로 Amazon Lex는 사용자에게 전달할 다음 메시지를 반환합니다. 다음 예제를 검토하십시오.

- 사용자가 "피자를 먹고 싶어요"라고 입력하면 Amazon Lex는 슬롯 데이터(예: `PizzaSize`: "어떤 피자 크기를 원하시나요?")를 유도하는 메시지가 포함된 응답을 반환할 수 있습니다.
- 사용자가 모든 피자 주문 정보를 제공한 후 Amazon Lex는 "피자를 주문하시겠습니까?"라는 사용자 확인 메시지가 포함된 응답을 반환할 수 있습니다.
- 사용자가 확인 메시지에 "예"라고 응답하면 Amazon Lex에서 "감사합니다. 치즈 피자를 주문했습니다."라는 결론을 반환할 수 있습니다.

모든 Amazon Lex 메시지에 사용자의 응답이 필요한 것은 아닙니다. 예를 들어, 결론문에는 응답이 필요하지 않습니다. 일부 메시지에는 예 또는 아니오 응답만 필요합니다. Amazon Lex는 `message` 외에도 적절한 클라이언트 사용자 인터페이스를 표시하는 등 클라이언트 동작을 개선하는 데 사용할 수 있는 메시지에 대한 추가 컨텍스트를 제공합니다. 다음 예제를 살펴보세요.

- 메시지가 슬롯 데이터를 끌어내라는 것이면 Amazon Lex는 다음 컨텍스트 정보를 반환합니다.
  - `x-amz-lex-dialog-state` 헤더는 `ElicitSlot`로 설정되었습니다.
  - `x-amz-lex-intent-name` 현재 컨텍스트의 의도 이름으로 설정된 헤더
  - `x-amz-lex-slot-to-elicit` 헤더는 `message`가 정보를 이끌어내는 슬롯 이름으로 설정됩니다.
  - `x-amz-lex-slots` 헤더는 현재 값을 사용하여 의도에 맞게 구성된 슬롯의 맵으로 설정됩니다.
- 메시지가 확인 프롬프트인 경우 `x-amz-lex-dialog-state` 헤더는 `Confirmation`로 설정되고 `x-amz-lex-slot-to-elicit` 헤더는 생략됩니다.
- 메시지가 의도에 맞게 구성된 설명 프롬프트로서 사용자 의도를 이해할 수 없음을 나타내는 경우 `x-amz-dialog-state`헤더는 `ElicitIntent`로 설정되고 `x-amz-slot-to-elicit` 헤더는 생략됩니다.

또한 Amazon Lex는 애플리케이션별 `sessionAttributes` 정보도 반환합니다. 자세한 내용은 대화 컨텍스트 관리를 참조하십시오.

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
x-amz-lex-active-contexts: activeContexts

inputStream
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

accept

이것은 `Accept` HTTP 헤더의 값으로 전달됩니다.

Amazon Lex가 응답에서 반환하는 메시지는 요청의 `Accept` HTTP 헤더 값을 기반으로 하는 텍스트 또는 음성일 수 있습니다.

- 값이 `text/plain; charset=utf-8`인 경우 Amazon Lex는 응답에서 텍스트를 반환합니다.
- 값이 `audio/`로 시작하는 경우 Amazon Lex는 응답으로 음성을 반환합니다. Amazon Lex는 Amazon Polly를 사용하여 음성을 생성합니다 (`Accept` 헤더에 지정한 구성 사용). 예를 들어 값을 `audio/mpeg`로 지정하면 Amazon Lex는 음성을 MPEG 형식으로 반환합니다.
- 값이 `audio/pcm`인 경우 반환되는 음성은 16비트 리틀 엔디안 형식의 `audio/pcm`입니다.
- 허용되는 값은 다음과 같습니다.
  - 오디오/mpeg
  - 오디오/ogg
  - 오디오/pcm
  - 텍스트/플레인, 문자셋=utf-8
  - audio/* (기본값은 mpeg)

activeContexts

요청에 대해 활성화된 컨텍스트 목록. 이전 의도가 이행될 때 또는 요청에 컨텍스트를 포함시켜 컨텍스트를 활성화할 수 있습니다.

컨텍스트 목록을 지정하지 않으면 Amazon Lex는 세션의 현재 컨텍스트 목록을 사용합니다. 빈 목록을 지정하면 세션의 모든 컨텍스트가 지워집니다.

botAlias

Amazon Lex 봇의 별칭

필수 여부: 예

botName

Amazon Lex 봇의 이름.

필수 여부: 예

contentType

이것은 `Content-Type` HTTP 헤더의 값으로 전달됩니다.

오디오 형식 또는 텍스트를 나타냅니다. 헤더 값은 다음 접두사 중 하나로 시작해야 합니다.

- PCM 형식의 오디오 데이터는 리틀 엔디안 바이트 순서여야 합니다.
  - 오디오/l16, 속도=16000, 채널=1
  - 오디오/x-l16, 샘플-레이트=16000, 채널-수=1
  - 오디오/lpcm, 샘플-레이트=8000, 샘플-크기-비트=16, 채널-수=1, is-big-endian=거짓
- Opus 포맷
  - 오디오/x-cbr-opus-with 프리앰블, 프리앰블-크기=0, 비트-레이트=256000, 프레임-크기-밀리초=4
- 날짜 형식
  - 텍스트/플레인, 문자셋=utf-8

필수 여부: 예

requestAttributes

이것은 `x-amz-lex-request-attributes` HTTP 헤더의 값으로 전달됩니다.

Amazon Lex와 클라이언트 애플리케이션 간에 전달되는 요청별 정보. 값은 문자열 키와 값이 있는 JSON으로 직렬화되고 base64로 인코딩된 맵이어야 합니다. `requestAttributes` 및 `sessionAttributes` 헤더의 총 크기는 12KB로 제한됩니다.

네임스페이스 `x-amz-lex:`는 특수 속성용으로 남겨둡니다. 접두사 `x-amz-lex:`를 사용하여 요청 속성을 생성하지 마세요.

요청 속성에 대한 자세한 내용은 요청 속성 설정을 참조하십시오.

sessionAttributes

이것은 `x-amz-lex-session-attributes` HTTP 헤더의 값으로 전달됩니다.

Amazon Lex와 클라이언트 애플리케이션 간에 전달되는 요청별 정보. 값은 문자열 키와 값이 있는 JSON으로 직렬화되고 base64로 인코딩된 맵이어야 합니다. `sessionAttributes` 및 `requestAttributes` 헤더의 총 크기는 12KB로 제한됩니다.

요청 속성에 대한 자세한 내용은 요청 속성 설정을 참조하십시오.

userId

클라이언트 애플리케이션 사용자의 ID입니다. Amazon Lex는 이를 사용하여 사용자와 봇의 대화를 식별합니다. 런타임 시 각 요청에는 `userID` 필드가 포함되어야 합니다.

애플리케이션에 사용할 사용자 ID를 결정하려면 다음 요소를 고려하십시오.

- `userID` 필드에는 사용자의 개인 식별 정보 (예: 이름, 개인 식별 번호 또는 기타 최종 사용자 개인 정보)가 포함되어서는 안 됩니다.
- 사용자가 한 기기에서 대화를 시작하고 다른 기기에서 계속하도록 하려면 사용자별 식별자를 사용하세요.
- 동일한 사용자가 서로 다른 두 기기에서 독립적인 대화를 두 번 할 수 있게 하려면 기기별 식별자를 선택하세요.
- 사용자는 같은 봇의 서로 다른 두 가지 버전과 독립적인 대화를 두 번 할 수 없습니다. 예를 들어 사용자는 동일한 봇의 PROD 및 BETA 버전과 대화할 수 없습니다. 예를 들어 테스트 중에 사용자가 서로 다른 두 버전과 대화해야 할 것으로 예상되는 경우 사용자 ID에 봇 별칭을 포함하여 두 대화를 구분하십시오.

길이 제약: 최소 길이는 2. 최대 길이는 100.

패턴: `[0-9a-zA-Z._:-]+`

필수 여부: 예

요청 본문

요청은 다음의 이진 데이터를 허용합니다.

inputStream

`Content-Type` HTTP 헤더에 설명된 대로 PCM 또는 Opus 오디오 형식 또는 텍스트 형식의 사용자 입력입니다.

오디오 데이터를 Amazon Lex로 스트리밍하거나 오디오 데이터를 전송하기 전에 모든 오디오 데이터를 캡처하는 로컬 버퍼를 생성할 수 있습니다. 일반적으로 로컬에서 데이터를 버퍼링하는 대신 오디오 데이터를 스트리밍하면 성능이 향상됩니다.

필수 사항 여부: 예

응답 구문

```
HTTP/1.1 200
Content-Type: contentType
x-amz-lex-intent-name: intentName
x-amz-lex-nlu-intent-confidence: nluIntentConfidence
x-amz-lex-alternative-intents: alternativeIntents
x-amz-lex-slots: slots
x-amz-lex-session-attributes: sessionAttributes
x-amz-lex-sentiment: sentimentResponse
x-amz-lex-message: message
x-amz-lex-encoded-message: encodedMessage
x-amz-lex-message-format: messageFormat
x-amz-lex-dialog-state: dialogState
x-amz-lex-slot-to-elicit: slotToElicit
x-amz-lex-input-transcript: inputTranscript
x-amz-lex-encoded-input-transcript: encodedInputTranscript
x-amz-lex-bot-version: botVersion
x-amz-lex-session-id: sessionId
x-amz-lex-active-contexts: activeContexts

audioStream
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

응답에 다음 HTTP 헤더가 반환됩니다.

activeContexts

세션에 대해 활성화된 컨텍스트 목록. 의도가 이행될 때 또는 `PostContent`, `PostText`, 또는 `PutSession` 작업을 호출하여 컨텍스트를 설정할 수 있습니다.

컨텍스트를 사용하여 의도를 추적할 수 있는 의도를 제어하거나 애플리케이션 작업을 수정할 수 있습니다.

alternativeIntents

사용자의 의도에 적용할 수 있는 1~4개의 대체 의도.

각 대안에는 Amazon Lex가 의도가 사용자의 의도와 일치한다고 얼마나 확신하는 지를 나타내는 점수가 포함되어 있습니다. 의도는 신뢰도 점수를 기준으로 정렬됩니다.

botVersion

대화에 응답한 봇의 버전입니다. 이 정보를 사용하여 한 버전의 봇이 다른 버전보다 성능이 좋은지 확인할 수 있습니다.

길이 제약: 최소 길이는 1. 최대 길이는 64.

패턴: `[0-9]+|\$LATEST`

contentType

요청의 `Accept` HTTP 헤더에 지정된 콘텐츠 유형입니다.

dialogState

사용자 상호 작용의 현재 상태를 식별합니다. Amazon Lex는 다음 값 중 하나를 `dialogState`로 반환합니다. 클라이언트는 선택적으로 이 정보를 사용하여 사용자 인터페이스를 사용자 정의할 수 있습니다.

- `ElicitIntent` - Amazon Lex는 사용자의 의도를 이끌어내고자 합니다. 다음 예제를 살펴보세요.

  예를 들어, 사용자가 의도 ("피자를 주문하고 싶어요")를 말할 수 있습니다. Amazon Lex가 이 발언에서 사용자 의도를 유추할 수 없는 경우 이 대화상자 상태를 반환합니다.

- `ConfirmIntent` - Amazon Lex는 "예" 또는 "아니요"라는 응답을 기대하고 있습니다.

  예를 들어 Amazon Lex는 의도를 이행하기 전에 사용자 확인을 원합니다. 사용자는 단순한 "예" 또는 "아니요" 응답 대신 추가 정보로 응답할 수 있습니다. 예를 들어 "네, 하지만 두꺼운 크러스트

피자로 만드세요" 또는 "아니요, 음료를 주문하고 싶어요." Amazon Lex는 이러한 추가 정보를 처리할 수 있습니다(이 예시에서는 크러스트 유형 슬롯을 업데이트하거나 의도를 OrderPizza에서 OrderDrink로 변경).

- `ElicitSlot` - Amazon Lex는 현재 의도에 사용할 슬롯의 값을 예상하고 있습니다.

  예를 들어 Amazon Lex가 응답에서 "어떤 크기의 피자를 원하시나요?" 라는 메시지를 보낸다고 가정해 보겠습니다. 사용자가 슬롯 값(예: "중간")으로 응답할 수 있습니다. 사용자는 응답 시 추가 정보(예: "중간 두께의 크러스트 피자") 를 제공할 수도 있습니다. Amazon Lex는 이러한 추가 정보를 적절하게 처리할 수 있습니다.
- `Fulfilled` - Lambda 함수가 의도를 성공적으로 수행했음을 전달합니다.
- `ReadyForFulfillment` - 클라이언트가 요청을 이행해야 함을 전달합니다.
- `Failed` - 사용자와의 대화가 실패했음을 전달합니다.

  이는 사용자가 서비스의 프롬프트에 적절한 응답을 제공하지 않는 경우(Amazon Lex가 사용자에게 특정 정보를 요청할 수 있는 횟수를 구성할 수 있음) 또는 Lambda 함수가 의도를 이행하지 못하는 경우를 포함하여 다양한 이유로 발생할 수 있습니다.

유효 값: `ElicitIntent | ConfirmIntent | ElicitSlot | Fulfilled | ReadyForFulfillment | Failed`

encodedInputTranscript

요청을 처리하는 데 사용되는 텍스트입니다.

입력이 오디오 스트림인 경우 `encodedInputTranscript` 필드에는 오디오 스트림에서 추출한 텍스트가 포함됩니다. 이 텍스트는 의도 및 슬롯 값을 인식하기 위해 실제로 처리되는 텍스트입니다. 이 정보를 사용하여 Amazon Lex가 전송되는 오디오를 올바르게 처리하고 있는지 확인할 수 있습니다.

`encodedInputTranscript` 필드는 base-64로 인코딩됩니다. 값을 사용하려면 먼저 필드를 디코딩해야 합니다.

encodedMessage

사용자에게 전달하는 메시지입니다. 메시지는 봇의 구성 또는 Lambda 함수에서 올 수 있습니다.

의도가 Lambda 함수로 구성되지 않았거나 Lambda 함수가 응답으로 `Delegate`을 `dialogAction.type`로 반환하는 경우, Amazon Lex는 다음 액션 코스를 결정하고 현재 상호 작용 컨텍스트를 기반으로 봇의 구성에서 적절한 메시지를 선택합니다. 예를 들어 Amazon Lex가 사용자 입력을 이해할 수 없는 경우 설명 프롬프트 메시지를 사용합니다.

의도를 생성하면 그룹에 메시지를 할당할 수 있습니다. 메시지가 그룹에 할당되면 Amazon Lex는 응답의 각 그룹에서 메시지를 하나씩 반환합니다. 메시지 필드는 메시지가 포함된 이스케이프된 JSON 문자열입니다. 반환된 JSON 문자열의 구조에 대한 자세한 내용은 지원되는 메시지 형식을 참조하세요.

Lambda 함수가 메시지를 반환하면 Amazon Lex는 응답으로 이를 클라이언트에 전달합니다.

`encodedMessage` 필드는 base-64로 인코딩됩니다. 값을 사용하려면 먼저 필드를 디코딩해야 합니다.

길이 제약: 최소 길이는 1. 최대 길이는 1,366입니다.

inputTranscript

이 헤더는 더 이상 사용되지 않습니다.

요청을 처리하는 데 사용되는 텍스트입니다.

이 필드는 de-de, en-AU, en-GB, en-US, es-419, es-ES, es-US, fr-CA, fr-FR 및 it-IT 로캘에서만 사용할 수 있습니다. 다른 모든 로캘에서는 이 `inputTranscript` 필드가 null입니다. 대신에 `encodedInputTranscript` 필드를 사용해야 합니다.

입력이 오디오 스트림인 경우 `inputTranscript` 필드에는 오디오 스트림에서 추출한 텍스트가 포함됩니다. 이 텍스트는 의도 및 슬롯 값을 인식하기 위해 실제로 처리되는 텍스트입니다. 이 정보를 사용하여 Amazon Lex가 전송되는 오디오를 올바르게 처리하고 있는지 확인할 수 있습니다.

intentName

Amazon Lex가 알고 있는 현재 사용자 의도를 나타냅니다.

message

이 헤더는 더 이상 사용되지 않습니다.

이 필드는 de-de, en-AU, en-GB, en-US, es-419, es-ES, es-US, fr-CA, fr-FR 및 it-IT 로캘에서만 사용할 수 있습니다. 다른 모든 로캘에서는 이 `message` 필드가 null입니다. 대신에 `encodedMessage` 필드를 사용해야 합니다.

사용자에게 전달하는 메시지입니다. 메시지는 봇의 구성 또는 Lambda 함수에서 올 수 있습니다.

의도가 Lambda 함수로 구성되지 않았거나 Lambda 함수가 응답으로 `Delegate`을 `dialogAction.type`로 반환하는 경우, Amazon Lex는 다음 액션 코스를 결정하고 현재 상호 작용 컨텍스트를 기반으로 봇의 구성에서 적절한 메시지를 선택합니다. 예를 들어 Amazon Lex가 사용자 입력을 이해할 수 없는 경우 설명 프롬프트 메시지를 사용합니다.

의도를 생성하면 그룹에 메시지를 할당할 수 있습니다. 메시지가 그룹에 할당되면 Amazon Lex는 응답의 각 그룹에서 메시지를 하나씩 반환합니다. 메시지 필드는 메시지가 포함된 이스케이프된 JSON 문자열입니다. 반환된 JSON 문자열의 구조에 대한 자세한 내용은 지원되는 메시지 형식을 참조하세요.

Lambda 함수가 메시지를 반환하면 Amazon Lex는 응답으로 이를 클라이언트에 전달합니다.

길이 제약: 최소 길이는 1. 최대 길이는 1,024.

messageFormat

응답 메시지의 형식. 다음 값 중 하나입니다.

- `PlainText` - 메시지에 일반 UTF-8 텍스트가 포함됩니다.
- `CustomPayload` - 메시지는 클라이언트의 사용자 지정 형식입니다.
- `SSML` - 메시지에 음성 출력용으로 서식이 지정된 텍스트가 포함됩니다.
- `Composite` - 메시지에는 의도 생성 시 메시지가 할당된 그룹의 메시지가 하나 이상 포함된 이스케이프된 JSON 개체가 포함되어 있습니다.

유효 값: `PlainText | CustomPayload | SSML | Composite`

nluIntentConfidence

Amazon Lex가 반환된 의도가 사용자의 의도와 일치한다고 얼마나 확신하는지 나타내는 점수를 제공합니다. 점수는 0.0~1.0 사이입니다.

점수는 상대 점수이며 절대 점수가 아닙니다. 점수는 Amazon Lex의 개선 사항에 따라 변경될 수 있습니다.

sentimentResponse

감정이 하나의 표현으로 표현되었습니다.

봇이 감정 분석을 위해 Amazon Comprehend로 표현을 보내도록 구성된 경우 이 필드에 분석 결과가 포함됩니다.

sessionAttributes

세션별 컨텍스트 정보를 나타내는 키/값 페어의 맵입니다.

sessionId

세션의 고유 식별자입니다.

slots

Amazon Lex가 대화 중 사용자 입력에서 감지한 0개 이상의 의도 슬롯(이름/값 페어) 맵입니다. 필드는 base-64로 인코딩됩니다.

Amazon Lex는 슬롯에 대한 예상 값을 가진 해결 목록을 생성합니다. 반환되는 값은 슬롯 유형이 생성되거나 업데이트될 때 선택된 `valueSelectionStrategy`에 따라 결정됩니다. `valueSelectionStrategy`을 `ORIGINAL_VALUE`로 설정하면 사용자 값이 슬롯 값과 유사한 경우 사용자가 제공한 값이 반환됩니다. `valueSelectionStrategy`가 `TOP_RESOLUTION`로 설정된 경우, Amazon Lex는 해결 목록의 첫 번째 값을 반환하고, 해결 목록이 없는 경우 null을 반환합니다. `valueSelectionStrategy`를 지정하지 않으면 기본값은 `ORIGINAL_VALUE`입니다.

slotToElicit

`dialogState`값이 `ElicitSlot`인 경우 Amazon Lex가 값을 추출하는 슬롯의 이름을 반환합니다.

응답은 다음 내용을 HTTP 본문으로 반환합니다.

audioStream

사용자에게 전달할 프롬프트(또는 명령문). 이는 봇 구성 및 컨텍스트를 기반으로 합니다. 예를 들어 Amazon Lex가 사용자 의도를 이해하지 못한 경우 봇을 위해 구성된 `clarificationPrompt`을 보냅니다. 이행 작업을 수행하기 전에 의도에서 확인이 필요한 경우 `confirmationPrompt`를 보냅니다. 또 다른 예: Lambda 함수가 의도를 성공적으로 수행하고 사용자에게 전달할 메시지를 보냈다고 가정해 보겠습니다. 그러면 Amazon Lex가 응답으로 해당 메시지를 보냅니다.

오류

BadGatewayException

Amazon Lex 봇이 아직 구축 중이거나 종속 서비스 중 하나(Amazon Polly, AWS Lambda) 가 내부 서비스 오류로 인해 장애가 발생했습니다.

HTTP 상태 코드: 502

BadRequestException

요청 검증이 실패했거나, 컨텍스트에 사용 가능한 메시지가 없거나, 봇 빌드가 실패했거나, 아직 진행 중이거나, 빌드되지 않은 변경 사항이 포함되어 있습니다.

HTTP 상태 코드: 400

ConflictException

두 클라이언트가 동일한 AWS 계정, Amazon Lex 봇 및 사용자 ID를 사용하고 있습니다.

HTTP 상태 코드: 409

DependencyFailedException

AWS Lambda 또는 Amazon Polly와 같은 종속 서비스 중 하나에서 예외가 발생했습니다. 예를 들어,

- Amazon Lex에 Lambda 함수를 호출할 수 있는 충분한 권한이 없는 경우.
- Lambda 함수를 실행하는 데 30초 이상 걸리는 경우.
- 이행 Lambda 함수가 슬롯 값을 제거하지 않고 `Delegate` 대화 작업을 반환하는 경우.

HTTP 상태 코드: 424

InternalFailureException

내부 서비스 오류. 호출을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

제한 초과함.

HTTP 상태 코드: 429

LoopDetectedException

이 예외는 사용되지 않습니다.

HTTP 상태 코드: 508

NotAcceptableException

요청의 수락 헤더에 유효한 값이 없습니다.

HTTP 상태 코드: 406

NotFoundException

참조된 리소스(예: Amazon Lex 봇 또는 별칭)를 찾을 수 없습니다.

HTTP 상태 코드: 404

RequestTimeoutException

입력 음성이 너무 깁니다.

HTTP 상태 코드: 408

UnsupportedMediaTypeException

콘텐츠 유형 헤더(`PostContentAPI`)에 유효하지 않은 값이 있습니다.

HTTP 상태 코드: 415

예시

예시 1

이 요청에서 URI는 봇(트래픽), 봇 버전($LATEST) 및 최종 사용자 이름(일부 사용자)을 식별합니다. `Content-Type` 헤더는 본문의 오디오 형식을 식별합니다. Amazon Lex는 다른 형식도 지원합니다. 필요한 경우 SoX 오픈 소스 소프트웨어를 사용하여 오디오를 한 형식에서 다른 형식으로 변환할 수 있습니다. `Accept` HTTP 헤더를 추가하여 응답을 받을 형식을 지정합니다.

응답의 `x-amz-lex-message` 헤더에는 Amazon Lex가 반환한 응답이 표시됩니다. 그러면 클라이언트는 이 응답을 사용자에게 보낼 수 있습니다. 청크 인코딩을 통해 동일한 메시지가 오디오/MPEG 형식으로 전송됩니다(요청에 따라).

샘플 요청

```
"POST /bot/Traffic/alias/$LATEST/user/someuser/content HTTP/1.1[\r][\n]"
"x-amz-lex-session-attributes: eyJ1c2VyTmFtZSI6IkJvYiJ9[\r][\n]"
"Content-Type: audio/x-l16; channel-count=1; sample-rate=16000f[\r][\n]"
"Accept: audio/mpeg[\r][\n]"
"Host: runtime.lex.us-east-1.amazonaws.com[\r][\n]"
"Authorization: AWS4-HMAC-SHA256 Credential=BLANKED_OUT/20161230/us-east-1/lex/
aws4_request,
SignedHeaders=accept;content-type;host;x-amz-content-sha256;x-amz-date;x-amz-lex-
session-attributes,
 Signature=78ca5b54ea3f64a17ff7522de02cd90a9acd2365b45a9ce9b96ea105bb1c7ec2[\r][\n]"
"X-Amz-Date: 20161230T181426Z[\r][\n]"
"X-Amz-Content-Sha256:
 e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855[\r][\n]"
"Transfer-Encoding: chunked[\r][\n]"
```

```
"Connection: Keep-Alive[\r][\n]"
"User-Agent: Apache-HttpClient/4.5.x (Java/1.8.0_112)[\r][\n]"
"Accept-Encoding: gzip,deflate[\r][\n]"
"[\r][\n]"
"1000[\r][\n]"
"[0x7][0x0][0x7][0x0][\n]"
"[0x0][0x7][0x0][0xfc][0xff][\n]"
"[0x0][\n]"
…
```

샘플 응답

```
"HTTP/1.1 200 OK[\r][\n]"
"x-amzn-RequestId: cc8b34af-cebb-11e6-a35c-55f3a992f28d[\r][\n]"
"x-amz-lex-message: Sorry, can you repeat that?[\r][\n]"
"x-amz-lex-dialog-state: ElicitIntent[\r][\n]"
"x-amz-lex-session-attributes: eyJ1c2VyTmFtZSI6IkJvYiJ9[\r][\n]"
"Content-Type: audio/mpeg[\r][\n]"
"Transfer-Encoding: chunked[\r][\n]"
"Date: Fri, 30 Dec 2016 18:14:28 GMT[\r][\n]"
"[\r][\n]"
"2000[\r][\n]"
"ID3[0x4][0x0][0x0][0x0][0x0][0x0]#TSSE[0x0][0x0][0x0][0xf][0x0][0x0]
[0x3]Lavf57.41.100[0x0][0x0][0x0][0x0][0x0][0x0][0x0][0x0][0x0][0x0][0x0][0xff]
[0xf3]`[0xc4][0x0][0x1b]{[0x8d][0xe8][0x1]C[0x18][0x1][0x0]J[0xe0]`b[0xdd][0xd1]
[0xb][0xfd][0x11][0xdf][0xfe]";[0xbb][0xbb][0x9f][0xee][0xee][0xee][0xee]|DDD/[0xff]
[0xff][0xff][0xff]www?D[0xf7]w^?[0xff][0xfa]h[0x88][0x85][0xfe][0x88][0x88][0x88]
[[0xa2]'[0xff][0xfa]"{[0x9f][0xe8][0x88]]D[0xeb][0xbb][0xbb][0xa2]!u[0xfd][0xdd][0xdf]
[0x88][0x94][0x0]F[0xef][0xa1]8[0x0][0x82]w[0x88]N[0x0][0x0][0x9b][0xbb][0xe8][0xe
…
```

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK

- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

PostText

서비스: Amazon Lex Runtime Service

Amazon Lex에 사용자 입력을 전송합니다. 클라이언트 애플리케이션은 이 API를 사용하여 런타임에 Amazon Lex에 요청을 보낼 수 있습니다. Amazon Lex는 봇용으로 구축한 기계 학습 모델을 사용하여 사용자 입력을 해석합니다.

이에 대한 응답으로 Amazon Lex는 다음 `message`을 반환하여 표시할 선택적 `responseCard`를 사용자에게 전달합니다. 다음 예제를 검토하십시오.

- 사용자가 "피자를 먹고 싶어요"라고 입력하면 Amazon Lex는 슬롯 데이터(예: PizzaSize: "어떤 피자 크기를 원하시나요?")를 유도하는 메시지가 포함된 응답을 반환할 수 있습니다.
- 사용자가 모든 피자 주문 정보를 제공한 후 Amazon Lex는 사용자 확인을 얻기 위해 "피자 주문을 진행하시겠습니까?"라는 메시지와 함께 응답을 반환할 수 있습니다.
- 사용자가 확인 프롬프트에 "예"라고 응답하면 Amazon Lex에서 "감사합니다. 치즈 피자가 주문되었습니다."라는 결론문을 반환할 수 있습니다.

모든 Amazon Lex 메시지에 사용자의 응답이 필요한 것은 아닙니다. 예를 들어, 결론문에는 응답이 필요하지 않습니다. 일부 메시지에는 "예" 또는 "아니오" 응답만 필요합니다. Amazon Lex는 `message` 외에도, 예를 들어, 적절한 클라이언트 사용자 인터페이스를 표시하는 등 클라이언트 동작을 개선하는 데 사용할 수 있는 메시지에 대한 추가 컨텍스트를 제공합니다. 응답의`slotToElicit`, `dialogState`, `intentName`, 및 `slots` 필드는 다음과 같습니다. 다음 예제를 살펴보세요.

- 메시지가 슬롯 데이터를 끌어내라는 것이면 Amazon Lex는 다음 컨텍스트 정보를 반환합니다.
  - `dialogState`을 ElicitSlot으로 설정합니다.
  - `intentName`을 현재 컨텍스트의 의도 이름으로 설정합니다.
  - `slotToElicit`은 `message`가 정보를 이끌어내는 슬롯 이름으로 설정됩니다.
  - `slots`은 현재 알려진 값을 사용하여 의도에 맞게 구성된 슬롯의 맵으로 설정됩니다.
- 메시지가 확인 프롬프트인 경우, `dialogState`는 ConfirmIntent로 설정되고 `SlotToElicit`는 null로 설정됩니다.
- 메시지가 사용자 의도가 이해되지 않았음을 나타내는 설명 프롬프트(의도에 대해 구성됨)인 경우 `dialogState`은 ElicitIntent로 설정되고 `slotToElicit`는 null로 설정됩니다.

또한 Amazon Lex는 애플리케이션별 `sessionAttributes` 정보도 반환합니다. 자세한 내용은 대화 컨텍스트 관리를 참조하십시오.

Request Syntax

```
POST /bot/botName/alias/botAlias/user/userId/text HTTP/1.1
Content-type: application/json

{
   "activeContexts": [
      {
         "name": "string",
         "parameters": {
            "string" : "string"
         },
         "timeToLive": {
            "timeToLiveInSeconds": number,
            "turnsToLive": number
         }
      }
   ],
   "inputText": "string",
   "requestAttributes": {
      "string" : "string"
   },
   "sessionAttributes": {
      "string" : "string"
   }
}
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

botAlias

Amazon Lex 봇의 별칭.

필수 여부: 예

botName

Amazon Lex 봇의 이름.

필수 여부: 예

userId

클라이언트 애플리케이션 사용자의 ID입니다. Amazon Lex는 이를 사용하여 사용자와 봇의 대화를 식별합니다. 런타임 시 각 요청에는 `userID` 필드가 포함되어야 합니다.

애플리케이션에 사용할 사용자 ID를 결정하려면 다음 요소를 고려하십시오.

- `userID` 필드에는 사용자의 개인 식별 정보 (예: 이름, 개인 식별 번호 또는 기타 최종 사용자 개인 정보)가 포함되어서는 안 됩니다.
- 사용자가 한 기기에서 대화를 시작하고 다른 기기에서 계속하도록 하려면 사용자별 식별자를 사용하세요.
- 동일한 사용자가 서로 다른 두 기기에서 독립적인 대화를 두 번 할 수 있게 하려면 기기별 식별자를 선택하세요.
- 사용자는 같은 봇의 서로 다른 두 가지 버전과 독립적인 대화를 두 번 할 수 없습니다. 예를 들어 사용자는 동일한 봇의 PROD 및 BETA 버전과 대화할 수 없습니다. 예를 들어 테스트 중에 사용자가 서로 다른 두 버전과 대화해야 할 것으로 예상되는 경우 사용자 ID에 봇 별칭을 포함하여 두 대화를 구분하십시오.

길이 제약: 최소 길이는 2. 최대 길이는 100.

패턴: `[0-9a-zA-Z._:-]+`

필수 여부: 예

요청 본문

요청은 JSON 형식으로 다음 데이터를 받습니다.

activeContexts

요청에 대해 활성화된 컨텍스트 목록. 이전 의도가 이행될 때 또는 요청에 컨텍스트를 포함시켜 컨텍스트를 활성화할 수 있습니다.

컨텍스트 목록을 지정하지 않으면 Amazon Lex는 세션의 현재 컨텍스트 목록을 사용합니다. 빈 목록을 지정하면 세션의 모든 컨텍스트가 지워집니다.

타입: ActiveContext객체 배열

배열 항목: 최소 항목 수는 0개. 최대 항목 수는 20개.

필수 여부: 아니요

inputText

사용자가 입력한 텍스트(Amazon Lex는 이 텍스트를 해석함).

AWS CLI를 사용하는 경우 `--input-text` 파라미터에 URL을 전달할 수 없습니다. 대신 `--cli-input-json` 파라미터를 사용하여 URL을 전달하십시오.

유형: 문자열

길이 제약: 최소 길이 1. 최대 길이는 1,024.

필수 여부: 예

requestAttributes

Amazon Lex와 클라이언트 애플리케이션 간에 전달되는 요청별 정보.

네임스페이스 `x-amz-lex:`는 특수 속성용으로 남겨둡니다. 접두사 `x-amz-lex:`를 사용하여 요청 속성을 생성하지 마세요.

요청 속성에 대한 자세한 내용은 요청 속성 설정을 참조하십시오.

유형: 문자열 대 문자열 맵

필수 여부: 아니요

sessionAttributes

Amazon Lex와 클라이언트 애플리케이션 간에 전달되는 요청별 정보.

요청 속성에 대한 자세한 내용은 요청 속성 설정을 참조하십시오.

유형: 문자열 대 문자열 맵

필수 항목 여부: 아니요

응답 구문

```
HTTP/1.1 200
Content-type: application/json

{
   "activeContexts": [
      {
         "name": "string",
         "parameters": {
```

```
            "string" : "string"
         },
         "timeToLive": {
            "timeToLiveInSeconds": number,
            "turnsToLive": number
         }
      }
   ],
   "alternativeIntents": [
      {
         "intentName": "string",
         "nluIntentConfidence": {
            "score": number
         },
         "slots": {
            "string" : "string"
         }
      }
   ],
   "botVersion": "string",
   "dialogState": "string",
   "intentName": "string",
   "message": "string",
   "messageFormat": "string",
   "nluIntentConfidence": {
      "score": number
   },
   "responseCard": {
      "contentType": "string",
      "genericAttachments": [
         {
            "attachmentLinkUrl": "string",
            "buttons": [
               {
                  "text": "string",
                  "value": "string"
               }
            ],
            "imageUrl": "string",
            "subTitle": "string",
            "title": "string"
         }
      ],
      "version": "string"
```

```
    },
    "sentimentResponse": {
       "sentimentLabel": "string",
       "sentimentScore": "string"
    },
    "sessionAttributes": {
       "string" : "string"
    },
    "sessionId": "string",
    "slots": {
       "string" : "string"
    },
    "slotToElicit": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

activeContexts

세션에 대해 활성화된 컨텍스트 목록. 의도가 이행될 때 또는 `PostContent`, `PostText`, 또는 `PutSession` 작업을 호출하여 컨텍스트를 설정할 수 있습니다.

컨텍스트를 사용하여 의도를 추적할 수 있는 의도를 제어하거나 애플리케이션 작업을 수정할 수 있습니다.

타입: ActiveContext객체 배열

배열 항목: 최소 항목 수는 0개. 최대 항목 수는 20개.

alternativeIntents

사용자의 의도에 적용할 수 있는 1~4개의 대체 의도.

각 대안에는 Amazon Lex가 의도가 사용자의 의도와 일치한다고 얼마나 확신하는 지를 나타내는 점수가 포함되어 있습니다. 의도는 신뢰도 점수를 기준으로 정렬됩니다.

타입: PredictedIntent객체 배열

배열 멤버: 최대 항목 수는 4개입니다.

botVersion

대화에 응답한 봇의 버전입니다. 이 정보를 사용하여 한 버전의 봇이 다른 버전보다 성능이 좋은지 확인할 수 있습니다.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `[0-9]+|\$LATEST`

dialogState

사용자 상호 작용의 현재 상태를 식별합니다. Amazon Lex는 다음 값 중 하나를 `dialogState`로 반환합니다. 클라이언트는 선택적으로 이 정보를 사용하여 사용자 인터페이스를 사용자 정의할 수 있습니다.

- `ElicitIntent` - Amazon Lex는 사용자의 의도를 이끌어내고자 합니다.

  예를 들어, 사용자가 의도("피자를 주문하고 싶어요")를 말할 수 있습니다. Amazon Lex가 이 표현에서 사용자 의도를 유추할 수 없는 경우 이 dialogState를 반환합니다.

- `ConfirmIntent` - Amazon Lex는 "예" 또는 "아니요"라는 응답을 기대하고 있습니다.

  예를 들어 Amazon Lex는 의도를 이행하기 전에 사용자 확인을 원합니다.

  사용자는 단순한 "예" 또는 "아니요" 대신 추가 정보로 응답할 수 있습니다. 예를 들어 "네, 하지만 두꺼운 크러스트 피자로 만드세요" 또는 "아니요, 음료를 주문하고 싶어요." Amazon Lex는 이러한 추가 정보를 처리할 수 있습니다(이 예시에서는 크러스트 유형 슬롯을 업데이트하거나 의도를 OrderPizza에서 OrderDrink로 변경).

- `ElicitSlot` - Amazon Lex는 현재 의도에 사용할 슬롯의 값을 예상하고 있습니다.

  예를 들어 Amazon Lex가 응답에서 "어떤 크기의 피자를 원하시나요?"라는 메시지를 보낸다고 가정해 보겠습니다. 사용자가 슬롯 값(예: "중간")으로 응답할 수 있습니다. 사용자는 응답 시 추가 정보(예: "중간 두께의 크러스트 피자") 를 제공할 수도 있습니다. Amazon Lex는 이러한 추가 정보를 적절하게 처리할 수 있습니다.

- `Fulfilled` - 의도에 대해 구성된 Lambda 함수가 성공적으로 이행되었음을 전달합니다.
- `ReadyForFulfillment` - 클라이언트가 의도를 이행해야 한다는 것을 전달합니다.
- `Failed` - 사용자와의 대화가 실패했음을 전달합니다.

이는 사용자가 서비스의 프롬프트에 적절한 응답을 제공하지 않았거나(Amazon Lex가 사용자에게 특정 정보에 대한 메시지를 표시할 수 있는 횟수를 구성할 수 있음) Lambda 함수가 의도를 이행하지 못한 경우 등 다양한 이유로 발생할 수 있습니다.

타입: 문자열

유효 값: `ElicitIntent | ConfirmIntent | ElicitSlot | Fulfilled | ReadyForFulfillment | Failed`

intentName

Amazon Lex가 알고 있는 현재 사용자 의도를 나타냅니다.

유형: 문자열

message

사용자에게 전달하는 메시지입니다. 메시지는 봇의 구성 또는 Lambda 함수에서 올 수 있습니다.

의도가 Lambda 함수로 구성되지 않았거나 Lambda 함수가 응답으로 `Delegate`을 `dialogAction.type`로 반환하는 경우, Amazon Lex는 다음 액션 코스를 결정하고 현재 상호 작용 컨텍스트를 기반으로 봇의 구성에서 적절한 메시지를 선택합니다. 예를 들어 Amazon Lex가 사용자 입력을 이해할 수 없는 경우 설명 프롬프트 메시지를 사용합니다.

의도를 생성하면 그룹에 메시지를 할당할 수 있습니다. 메시지가 그룹에 할당되면 Amazon Lex는 응답의 각 그룹에서 메시지를 하나씩 반환합니다. 메시지 필드는 메시지가 포함된 이스케이프된 JSON 문자열입니다. 반환된 JSON 문자열의 구조에 대한 자세한 내용은 지원되는 메시지 형식을 참조하세요.

Lambda 함수가 메시지를 반환하면 Amazon Lex는 응답으로 이를 클라이언트에 전달합니다.

유형: 문자열

길이 제약: 최소 길이 1. 최대 길이는 1,024.

messageFormat

응답 메시지의 형식. 다음 값 중 하나입니다.

- `PlainText` - 메시지에 일반 UTF-8 텍스트가 포함됩니다.
- `CustomPayload` - 메시지는 Lambda 함수에서 정의되는 사용자 지정 형식입니다.
- `SSML` - 메시지에 음성 출력용으로 서식이 지정된 텍스트가 포함됩니다.

- `Composite` - 메시지에는 의도 생성 시 메시지가 할당된 그룹의 메시지가 하나 이상 포함된 이스케이프된 JSON 개체가 포함되어 있습니다.

타입: 문자열

유효 값: `PlainText | CustomPayload | SSML | Composite`

nluIntentConfidence

Amazon Lex가 반환된 의도가 사용자의 의도와 일치한다고 얼마나 확신하는지 나타내는 점수를 제공합니다. 점수는 0.0~1.0 사이입니다. 자세한 내용은 신뢰도 점수를 참조하세요.

점수는 상대 점수이며 절대 점수가 아닙니다. 점수는 Amazon Lex의 개선 사항에 따라 변경될 수 있습니다.

유형: IntentConfidence객체

responseCard

사용자가 현재 프롬프트에 응답해야 하는 옵션을 나타냅니다. 응답 카드는 봇 구성 (Amazon Lex 콘솔에서, 슬롯 옆의 설정 버튼 선택) 또는 코드 후크(Lambda 함수)에서 가져올 수 있습니다.

유형: ResponseCard객체

sentimentResponse

표현된 감정과 표현.

봇이 감정 분석을 위해 Amazon Comprehend로 표현을 보내도록 구성된 경우 이 필드에 분석 결과가 포함됩니다.

유형: SentimentResponse객체

sessionAttributes

세션별 컨텍스트 정보를 나타내는 키-값 페어의 맵입니다.

유형: 문자열 간 맵

sessionId

세션에 대한 고유 식별자입니다.

유형: 문자열

slots

Amazon Lex가 대화의 사용자 입력에서 감지한 의도 슬롯.

Amazon Lex는 슬롯에 대한 예상 값을 가진 해결 목록을 생성합니다. 반환되는 값은 슬롯 유형이 생성되거나 업데이트될 때 선택된 `valueSelectionStrategy`에 따라 결정됩니다. `valueSelectionStrategy`을 `ORIGINAL_VALUE`로 설정하면 사용자 값이 슬롯 값과 유사한 경우 사용자가 제공한 값이 반환됩니다. `valueSelectionStrategy`가 `TOP_RESOLUTION`로 설정된 경우, Amazon Lex는 해결 목록의 첫 번째 값을 반환하고, 해결 목록이 없는 경우 null을 반환합니다. `valueSelectionStrategy`를 지정하지 않으면 기본값은 `ORIGINAL_VALUE`입니다.

유형: 문자열 간 맵

slotToElicit

`dialogState` 값이 `ElicitSlot`인 경우 Amazon Lex가 값을 추출하는 슬롯의 이름을 반환합니다.

유형: 문자열

오류

BadGatewayException

Amazon Lex 봇이 아직 구축 중이거나 종속 서비스 중 하나(Amazon Polly, AWS Lambda) 가 내부 서비스 오류로 인해 장애가 발생했습니다.

HTTP 상태 코드: 502

BadRequestException

요청 검증이 실패했거나, 컨텍스트에 사용 가능한 메시지가 없거나, 봇 빌드가 실패했거나, 아직 진행 중이거나, 빌드되지 않은 변경 사항이 포함되어 있습니다.

HTTP 상태 코드: 400

ConflictException

두 클라이언트가 동일한 AWS 계정, Amazon Lex 봇 및 사용자 ID를 사용하고 있습니다.

HTTP 상태 코드: 409

DependencyFailedException

AWS Lambda 또는 Amazon Polly와 같은 종속 서비스 중 하나에서 예외가 발생했습니다. 예를 들어,

- Amazon Lex에 Lambda 함수를 호출할 수 있는 충분한 권한이 없는 경우.

- Lambda 함수를 실행하는 데 30초 이상 걸리는 경우.
- 이행 Lambda 함수가 슬롯 값을 제거하지 않고 `Delegate` 대화 작업을 반환하는 경우.

HTTP 상태 코드: 424

InternalFailureException

내부 서비스 오류. 호출을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

제한 초과함.

HTTP 상태 코드: 429

LoopDetectedException

이 예외는 사용되지 않습니다.

HTTP 상태 코드: 508

NotFoundException

참조된 리소스(예: Amazon Lex 봇 또는 별칭)를 찾을 수 없습니다.

HTTP 상태 코드: 404

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK

- AWS SDK for Ruby V3

PutSession

서비스: Amazon Lex Runtime Service

Amazon Lex 봇을 사용하여 새 세션을 생성하거나 기존 세션을 수정합니다. 이 작업을 사용하면 애플리케이션이 봇의 상태를 설정할 수 있습니다.

자세한 내용은 세션 관리를 참조하세요.

Request Syntax

```
POST /bot/botName/alias/botAlias/user/userId/session HTTP/1.1
Accept: accept
Content-type: application/json

{
   "activeContexts": [
      {
         "name": "string",
         "parameters": {
            "string" : "string"
         },
         "timeToLive": {
            "timeToLiveInSeconds": number,
            "turnsToLive": number
         }
      }
   ],
   "dialogAction": {
      "fulfillmentState": "string",
      "intentName": "string",
      "message": "string",
      "messageFormat": "string",
      "slots": {
         "string" : "string"
      },
      "slotToElicit": "string",
      "type": "string"
   },
   "recentIntentSummaryView": [
      {
         "checkpointLabel": "string",
         "confirmationStatus": "string",
         "dialogActionType": "string",
```

```
            "fulfillmentState": "string",
            "intentName": "string",
            "slots": {
               "string" : "string"
            },
            "slotToElicit": "string"
         }
      ],
      "sessionAttributes": {
         "string" : "string"
      }
   }
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

accept

Amazon Lex가 응답으로 반환하는 메시지는 이 필드의 값에 따라 텍스트 또는 음성일 수 있습니다.

- 값이 `text/plain; charset=utf-8`인 경우 Amazon Lex는 응답에서 텍스트를 반환합니다.
- 값이 `audio/`로 시작하는 경우, Amazon Lex는 응답으로 음성을 반환합니다. Amazon Lex는 Amazon Polly를 사용하여 사용자가 지정한 구성으로 음성을 생성합니다. 예를 들어, 값을 `audio/mpeg`로 지정하면 Amazon Lex는 음성을 MPEG 형식으로 반환합니다.
- 값이 `audio/pcm`인 경우 반환되는 음성은 16비트 리틀 엔디안 형식의 `audio/pcm`입니다.
- 허용되는 값은 다음과 같습니다.
  - `audio/mpeg`
  - `audio/ogg`
  - `audio/pcm`
  - `audio/*`(기본값은 mpeg)
  - `text/plain; charset=utf-8`

botAlias

세션 데이터를 포함하는 봇에 사용되는 별칭입니다.

필수 여부: 예

botName

세션 데이터가 들어있는 봇의 명칭.

필수 여부: 예

userId

클라이언트 애플리케이션 사용자의 ID입니다. Amazon Lex는 이를 사용하여 사용자와 봇의 대화를 식별합니다.

길이 제약: 최소 길이는 2. 최대 길이는 100.

패턴: `[0-9a-zA-Z._:-]+`

필수 여부: 예

요청 본문

요청은 JSON 형식으로 다음 데이터를 받습니다.

activeContexts

요청에 대해 활성화된 컨텍스트 목록. 이전 의도가 이행될 때 또는 요청에 컨텍스트를 포함시켜 컨텍스트를 활성화할 수 있습니다.

컨텍스트 목록을 지정하지 않으면 Amazon Lex는 세션의 현재 컨텍스트 목록을 사용합니다. 빈 목록을 지정하면 세션의 모든 컨텍스트가 지워집니다.

타입: ActiveContext객체 배열

배열 항목: 최소 항목 수는 0개. 최대 항목 수는 20개.

필수 여부: 아니요

dialogAction

봇이 대화를 수행하기 위해 취해야 할 다음 조치를 설정합니다.

유형: DialogAction객체

필수 여부: 아니요

recentIntentSummaryView

봇에 대한 최근 의도 요약. 의도 요약 보기를 사용하여 의도에 체크포인트 라벨을 설정하고 의도의 속성을 수정할 수 있습니다. 또한 이를 사용하여 의도 요약 개체를 제거하거나 목록에서 추가할 수 있습니다.

목록에 수정하거나 추가하는 의도는 봇에 적합해야 합니다. 예를 들어 의도 이름은 봇에 대해 유효해야 합니다. 다음의 값을 제공해야 합니다.

- `intentName`
- 슬롯 이름
- `slotToElict`

`PutSession` 요청에 `recentIntentSummaryView` 파라미터를 보내면 새 요약 보기의 콘텐츠가 이전 요약 보기를 대체합니다. 예를 들어 `GetSession` 요청이 요약 보기에서 세 개의 의도를 반환하고 요약 보기에서 하나의 의도로 `PutSession`를 호출하는 경우 다음 `GetSession`에 대한 호출에서는 하나의 의도만 반환됩니다.

타입: IntentSummary객체 배열

배열 구성원: 최소수는 0개입니다. 최대 항목 수는 3개.

필수 여부: 아니요

sessionAttributes

세션별 컨텍스트 정보를 나타내는 키/값 페어의 맵입니다. Amazon Lex와 클라이언트 애플리케이션 간에 전달되는 요청별 정보를 포함합니다.

유형: 문자열 간 맵

필수 항목 여부: 아니요

응답 구문

```
HTTP/1.1 200
Content-Type: contentType
x-amz-lex-intent-name: intentName
x-amz-lex-slots: slots
x-amz-lex-session-attributes: sessionAttributes
x-amz-lex-message: message
x-amz-lex-encoded-message: encodedMessage
x-amz-lex-message-format: messageFormat
x-amz-lex-dialog-state: dialogState
x-amz-lex-slot-to-elicit: slotToElicit
x-amz-lex-session-id: sessionId
x-amz-lex-active-contexts: activeContexts
```

```
audioStream
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

응답에 다음 HTTP 헤더가 반환됩니다.

activeContexts

세션에 대해 활성화된 컨텍스트 목록.

contentType

요청의 `Accept` HTTP 헤더에 지정된 콘텐츠 유형입니다.

dialogState

- `ConfirmIntent` - Amazon Lex는 의도를 이행하기 전에 의도를 확인하기 위해 “예” 또는 “아니요” 응답을 기대하고 있습니다.
- `ElicitIntent` - Amazon Lex는 사용자의 의도를 이끌어내고자 합니다.
- `ElicitSlot`- Amazon Lex는 현재 의도에 사용할 슬롯의 값을 예상하고 있습니다.
- `Failed` - 사용자와의 대화가 실패했음을 전달합니다. 이는 사용자가 서비스의 프롬프트에 적절한 응답을 제공하지 않거나 Lambda 함수가 의도를 이행하지 못하는 경우 등 다양한 이유로 발생할 수 있습니다.
- `Fulfilled` - Lambda 함수가 의도를 성공적으로 수행했음을 전달합니다.
- `ReadyForFulfillment` - 클라이언트가 의도를 이행해야 한다는 것을 전달합니다.

유효 값: `ElicitIntent | ConfirmIntent | ElicitSlot | Fulfilled | ReadyForFulfillment | Failed`

encodedMessage

사용자에게 제시해야 할 다음 메시지입니다.

`encodedMessage` 필드는 base-64로 인코딩됩니다. 값을 사용하려면 먼저 필드를 디코딩해야 합니다.

길이 제약: 최소 길이는 1. 최대 길이는 1,366.

intentName

현재 의도의 이름.

message

이 헤더는 더 이상 사용되지 않습니다.

사용자에게 제시해야 할 다음 메시지입니다.

이 필드는 de-de, en-AU, en-GB, en-US, es-419, es-ES, es-US, fr-CA, fr-FR 및 it-IT 로캘에서만 사용할 수 있습니다. 다른 모든 로캘에서는 이 `message` 필드가 null입니다. 대신에 `encodedMessage` 필드를 사용해야 합니다.

길이 제약: 최소 길이 1. 최대 길이는 1,024.

messageFormat

응답 메시지의 형식. 다음 값 중 하나입니다.

- `PlainText` - 메시지에 일반 UTF-8 텍스트가 포함됩니다.
- `CustomPayload` - 메시지는 클라이언트의 사용자 지정 형식입니다.
- `SSML` - 메시지에 음성 출력용으로 서식이 지정된 텍스트가 포함됩니다.
- `Composite` - 메시지에는 의도 생성 시 메시지가 할당된 그룹의 메시지가 하나 이상 포함된 이스케이프된 JSON 개체가 포함되어 있습니다.

유효 값: `PlainText | CustomPayload | SSML | Composite`

sessionAttributes

세션별 컨텍스트 정보를 나타내는 키/값 페어의 맵입니다.

sessionId

세션에 대한 고유 식별자입니다.

slots

Amazon Lex가 대화 중에 사용자 입력에서 감지한 0개 이상의 의도 슬롯의 맵입니다.

Amazon Lex는 슬롯에 대한 예상 값을 가진 해결 목록을 생성합니다. 반환되는 값은 슬롯 유형이 생성되거나 업데이트될 때 선택된 `valueSelectionStrategy`에 따라 결정됩니다. `valueSelectionStrategy`을 `ORIGINAL_VALUE`로 설정하면 사용자 값이 슬롯 값과 유사한 경우 사용자가 제공한 값이 반환됩니다. `valueSelectionStrategy`가 `TOP_RESOLUTION`로 설정

된 경우, Amazon Lex는 해결 목록의 첫 번째 값을 반환하고, 해결 목록이 없는 경우 null을 반환합니다. `valueSelectionStrategy`를 지정하지 않으면 기본값은 `ORIGINAL_VALUE`입니다.

slotToElicit

`dialogState`이 `ElicitSlot`인 경우 Amazon Lex가 값을 추출하는 슬롯의 이름을 반환합니다.

응답은 다음 내용을 HTTP 본문으로 반환합니다.

audioStream

사용자에게 전달할 메시지의 오디오 버전입니다.

오류

BadGatewayException

Amazon Lex 봇이 아직 구축 중이거나 종속 서비스 중 하나(Amazon Polly, AWS Lambda) 가 내부 서비스 오류로 인해 장애가 발생했습니다.

HTTP 상태 코드: 502

BadRequestException

요청 검증이 실패했거나, 컨텍스트에 사용 가능한 메시지가 없거나, 봇 빌드가 실패했거나, 아직 진행 중이거나, 빌드되지 않은 변경 사항이 포함되어 있습니다.

HTTP 상태 코드: 400

ConflictException

두 클라이언트가 동일한 AWS 계정, Amazon Lex 봇 및 사용자 ID를 사용하고 있습니다.

HTTP 상태 코드: 409

DependencyFailedException

AWS Lambda 또는 Amazon Polly와 같은 종속 서비스 중 하나에서 예외가 발생했습니다. 예를 들어,

- Amazon Lex에 Lambda 함수를 호출할 수 있는 충분한 권한이 없는 경우.
- Lambda 함수를 실행하는 데 30초 이상 걸리는 경우.
- 이행 Lambda 함수가 슬롯 값을 제거하지 않고 `Delegate` 대화 작업을 반환하는 경우.

HTTP 상태 코드: 424

InternalFailureException

내부 서비스 오류. 호출을 다시 시도하세요.

HTTP 상태 코드: 500

LimitExceededException

제한 초과함.

HTTP 상태 코드: 429

NotAcceptableException

요청의 수락 헤더에 유효한 값이 없습니다.

HTTP 상태 코드: 406

NotFoundException

참조된 리소스(예: Amazon Lex 봇 또는 별칭)를 찾을 수 없습니다.

HTTP 상태 코드: 404

참고

언어별 AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2용 SDK
- AWS Java V2용 SDK
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS SDK for Ruby V3

데이터 타입

다음 데이터 유형이 Amazon Lex 모델 구축 서비스에서 지원됩니다.

- BotAliasMetadata
- BotChannelAssociation
- BotMetadata
- BuiltinIntentMetadata
- BuiltinIntentSlot
- BuiltinSlotTypeMetadata
- CodeHook
- ConversationLogsRequest
- ConversationLogsResponse
- EnumerationValue
- FollowUpPrompt
- FulfillmentActivity
- InputContext
- Intent
- IntentMetadata
- KendraConfiguration
- LogSettingsRequest
- LogSettingsResponse
- Message
- MigrationAlert
- MigrationSummary
- OutputContext
- Prompt
- ResourceReference
- Slot
- SlotDefaultValue
- SlotDefaultValueSpec

- SlotTypeConfiguration
- SlotTypeMetadata
- SlotTypeRegexConfiguration
- Statement
- Tag
- UtteranceData
- UtteranceList

다음 데이터 유형이 Amazon Lex 런타임 서비스에서 지원됩니다.

- ActiveContext
- ActiveContextTimeToLive
- Button
- DialogAction
- GenericAttachment
- IntentConfidence
- IntentSummary
- PredictedIntent
- ResponseCard
- SentimentResponse

Amazon Lex 모델 구축 서비스

다음 데이터 유형이 Amazon Lex 모델 구축 서비스에서 지원됩니다.

- BotAliasMetadata
- BotChannelAssociation
- BotMetadata
- BuiltinIntentMetadata
- BuiltinIntentSlot
- BuiltinSlotTypeMetadata
- CodeHook

- ConversationLogsRequest
- ConversationLogsResponse
- EnumerationValue
- FollowUpPrompt
- FulfillmentActivity
- InputContext
- Intent
- IntentMetadata
- KendraConfiguration
- LogSettingsRequest
- LogSettingsResponse
- Message
- MigrationAlert
- MigrationSummary
- OutputContext
- Prompt
- ResourceReference
- Slot
- SlotDefaultValue
- SlotDefaultValueSpec
- SlotTypeConfiguration
- SlotTypeMetadata
- SlotTypeRegexConfiguration
- Statement
- Tag
- UtteranceData
- UtteranceList

BotAliasMetadata

서비스: Amazon Lex Model Building Service

봇 별칭에 대한 정보를 제공합니다.

콘텐츠

botName

별칭이 가리키는 봇의 이름.

유형: 문자열

길이 제한: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

Required: No

botVersion

별칭이 가리키는 Amazon Lex 봇의 버전.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

Required: No

checksum

봇 별칭의 체크섬입니다.

유형: 문자열

필수 항목 여부: 아니요

conversationLogs

Amazon Lex가 별칭에 대한 대화 로그를 사용하는 방법을 결정하는 설정입니다.

유형: ConversationLogsResponse 객체

필수 여부: 아니요

createdDate

봇 별칭이 생성된 날짜.

유형: 타임스탬프

필수 여부: 아니요

description

별칭에 대한 설명입니다.

유형: 문자열

길이 제한: 최소 길이는 0. 최대 길이는 200.

필수 여부: 아니요

lastUpdatedDate

봇 별칭이 업데이트된 날짜. 리소스를 생성할 때 생성 날짜 및 최종 업데이트 날짜가 동일합니다.

유형: 타임스탬프

필수 여부: 아니요

name

봇 별칭의 이름.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2

- AWS SDK for Ruby V3

BotChannelAssociation

서비스: Amazon Lex Model Building Service

Amazon Lex 봇과 외부 메시징 플랫폼 간의 연결을 나타냅니다.

콘텐츠

botAlias

이 연결이 이루어지는 Amazon Lex 봇의 특정 버전을 가리키는 별칭입니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

Required: No

botConfiguration

메시징 플랫폼과 통신하는 데 필요한 정보를 제공합니다.

유형: 문자열 간 맵

맵 항목: 최대 항목 수는 10개.

필수 여부: 아니요

botName

이 연결이 이루어지는 Amazon Lex 봇의 이름.

> **Note**
> 현재 Amazon Lex는 페이스북, 슬랙, 트윌리오와의 제휴를 지원하고 있습니다.

유형: 문자열

길이 제한: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

Required: No

createdDate

Amazon Lex 봇과 채널 간의 연결이 생성된 날짜입니다.

유형: 타임스탬프

필수 여부: 아니요

description

생성 중인 연결에 대한 텍스트 설명입니다.

유형: 문자열

길이 제한: 최소 길이는 0. 최대 길이는 200.

필수 여부: 아니요

failureReason

만약 `status`이 `FAILED`라면, Amazon Lex는 봇 구축에 실패한 이유를 제공합니다.

유형: 문자열

필수 항목 여부: 아니요

name

봇과 채널 간의 연결 이름.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

Required: No

status

봇 채널의 상태입니다.

- `CREATED` - 채널이 생성되어 사용할 준비가 되었습니다.
- `IN_PROGRESS` - 채널 생성이 진행 중입니다.

- `FAILED` - 채널을 만드는 중 오류가 발생했습니다. 필드에 대한 자세한 내용은 `failureReason` 필드를 참조하세요.

타입: 문자열

유효 값: `IN_PROGRESS | CREATED | FAILED`

필수 여부: 아니요

type

Amazon Lex 봇과 외부 메시징 플랫폼 간에 설정되는 채널 유형을 표시하여 연결 유형을 지정합니다.

타입: 문자열

유효 값: `Facebook | Slack | Twilio-Sms | Kik`

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

BotMetadata

서비스: Amazon Lex Model Building Service

봇에 대한 정보를 제공합니다.

콘텐츠

createdDate

봇이 생성된 날짜.

유형: 타임스탬프

필수 여부: 아니요

description

봇에 대한 설명입니다.

유형: 문자열

길이 제한: 최소 길이는 0. 최대 길이는 200.

필수 여부: 아니요

lastUpdatedDate

봇이 생성된 날짜입니다. 봇을 생성할 때 생성 날짜 및 최종 업데이트 날짜가 동일합니다.

유형: 타임스탬프

필수 여부: 아니요

name

봇의 이름.

유형: 문자열

길이 제한: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

Required: No

status

봇의 상태.

타입: 문자열

유효 값: `BUILDING | READY | READY_BASIC_TESTING | FAILED | NOT_BUILT`

필수 여부: 아니요

version

봇의 버전. 새 봇의 경우 버전은 항상 `$LATEST`입니다.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

BuiltinIntentMetadata

서비스: Amazon Lex Model Building Service

기본 제공 의도에 관한 메타데이터를 제공합니다.

콘텐츠

signature

기본 제공 의도의 고유 식별자. 의도의 서명을 찾으려면 Alexa Skills Kit의 표준 기본 제공 의도를 참조하십시오.

유형: 문자열

필수 항목 여부: 아니요

supportedLocales

의도가 지원하는 로캘의 목록입니다.

유형: 문자열 배열

유효 값: `de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR`

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

BuiltinIntentSlot

서비스: Amazon Lex Model Building Service

기본 제공 의도에 사용되는 슬롯에 관한 정보를 제공합니다.

콘텐츠

name

의도에 정의된 슬롯 목록.

유형: 문자열

필수 항목 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

BuiltinSlotTypeMetadata

서비스: Amazon Lex Model Building Service

기본 제공 슬롯 유형에 관한 정보를 제공합니다.

콘텐츠

signature

기본 제공 슬롯 유형의 고유 식별자. 슬롯 유형의 서명을 찾으려면 Alexa Skills Kit의 슬롯 유형 참조를 참조하십시오.

유형: 문자열

필수 항목 여부: 아니요

supportedLocales

슬롯의 대상 로캘의 목록입니다.

유형: 문자열 배열

유효 값: `de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR`

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

CodeHook

서비스: Amazon Lex Model Building Service

봇에 대한 요청을 확인하거나 봇에 대한 사용자의 요청을 이행하는 Lambda 함수를 지정합니다.

콘텐츠

messageVersion

Amazon Lex가 Lambda 함수를 간접 호출하는 데 사용할 요청-응답의 버전입니다. 자세한 내용은 Lambda 함수 사용을 참조하세요.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 5.

필수 여부: 예

uri

Lambda 함수의 Amazon 리소스 이름(ARN)입니다.

유형: 문자열

길이 제약: 최소 길이는 20. 최대 길이는 2,048.

Pattern: `arn:aws[a-zA-Z-]*:lambda:[a-z]+-[a-z]+(-[a-z]+)*-[0-9]:[0-9]{12}:function:[a-zA-Z0-9-_]+(\/[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12})?(:[a-zA-Z0-9-_]+)?`

필수 여부: 예

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ConversationLogsRequest

서비스: Amazon Lex Model Building Service

대화 로그에 필요한 설정을 제공합니다.

내용

iamRoleArn

텍스트 로그의 경우 CloudWatch Logs에, 오디오 로그의 경우 S3 버킷에 쓸 수 있는 권한이 있는 IAM 역할의 Amazon 리소스 이름(ARN)입니다. 오디오 암호화가 활성화된 경우 이 역할은 오디오 로그를 암호화하는 데 사용되는 AWS KMS 키에 대한 액세스 권한도 제공합니다. 자세한 내용은 대화 로그를 위한 IAM 역할 및 정책 생성을 참조하십시오.

유형: 문자열

길이 제약: 최소 길이는 20. 최대 길이는 2,048.

Pattern: `^arn:[\w\-]+:iam::[\d]{12}:role/.+$`

필수 여부: 예

logSettings

대화 로그 설정. 대화 텍스트, 대화 오디오 또는 둘 다를 기록할 수 있습니다.

유형: LogSettingsRequest객체 어레이

필수 여부: 예

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ConversationLogsResponse

서비스: Amazon Lex Model Building Service

대화 로그 설정에 대한 정보가 포함되어있습니다.

내용

iamRoleArn

CloudWatch Logs 또는 S3 버킷에 로그를 작성하는 데 사용되는 IAM 역할의 Amazon 리소스 이름 (ARN) 입니다.

유형: 문자열

길이 제약: 최소 길이는 20. 최대 길이는 2,048.

패턴: `^arn:[\w\-]+:iam::[\d]{12}:role/.+$`

Required: No

logSettings

대화 로그 설정. 텍스트, 오디오 또는 둘 다를 기록할 수 있습니다.

유형: LogSettingsResponse 객체 배열

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

EnumerationValue

서비스: Amazon Lex Model Building Service

각 슬롯 유형에는 값 세트가 있을 수 있습니다. 각 열거 값은 슬롯 유형에 사용할 수 있는 값을 나타냅니다.

예를 들어 피자 주문 봇은 피자에 있어야 하는 크러스트 유형을 지정하는 슬롯 유형을 가질 수 있습니다. 슬롯 유형에는 값이 포함될 수 있습니다.

- 두꺼운
- 얇은
- 속을 채운

내용

value

슬롯 유형의 값.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 140.

필수 여부: 예

synonyms

슬롯 유형 항목과 관련된 추가 값입니다.

유형: 문자열 배열

길이 제약: 최소 길이는 1. 최대 길이는 140.

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2

- AWS SDK for Ruby V3

FollowUpPrompt

서비스: Amazon Lex Model Building Service

의도가 이행된 후 추가 활동에 대한 프롬프트입니다. 예를 들어 `OrderPizza` 의도가 충족된 후 사용자에게 음료를 주문하고 싶은지 여부를 확인하라는 프롬프트를 표시할 수 있습니다.

내용

prompt

사용자가 제공하는 정보를 위한 프롬프트.

유형: Prompt 객체

필수 여부: 예

rejectionStatement

사용자가 `prompt` 필드에 정의된 질문에 "아니요"라고 답하면 Amazon Lex는 의도가 취소되었음을 확인하기 위해 이 문장으로 답합니다.

유형: Statement 객체

필수 여부: 예

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

FulfillmentActivity

서비스: Amazon Lex Model Building Service

사용자가 의도에 필요한 모든 정보를 제공한 후 의도가 이행되는 방식을 설명합니다. Lambda 함수를 제공하여 의도를 처리하거나 의도 정보를 클라이언트 애플리케이션에 반환할 수 있습니다. 관련 로직이 클라우드에 있도록 Lambda 함수를 사용하고 클라이언트 측 코드를 주로 프레젠테이션으로만 제한하는 것이 좋습니다. 로직을 업데이트해야 하는 경우 Lambda 함수만 업데이트하면 되며 클라이언트 애플리케이션을 업그레이드할 필요는 없습니다.

다음 예제를 살펴보세요.

- 피자 주문 애플리케이션에서는 사용자가 주문을 위한 모든 정보를 제공한 후 Lambda 함수를 사용하여 피자 가게에 주문을 합니다.
- 게임 애플리케이션에서는 사용자가 “돌을 주워”라고 말하면 클라이언트 애플리케이션이 작업을 수행하고 그래픽을 업데이트할 수 있도록 이 정보가 클라이언트 애플리케이션에 다시 전달되어야 합니다. 이 경우 사용자는 Amazon Lex가 의도 데이터를 클라이언트에 반환하기를 원합니다.

내용

type

Lambda 함수를 실행하거나 슬롯 데이터를 클라이언트 애플리케이션에 반환하여 의도를 이행하는 방법.

타입: 문자열

유효 값: `ReturnIntent | CodeHook`

필수 사항 여부: 예

codeHook

의도를 이행하기 위해 실행되는 Lambda 함수에 대한 설명.

유형: CodeHook 객체

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

InputContext

서비스: Amazon Lex Model Building Service

Amazon Lex에서 의도를 선택하려면 활성화되어야 하는 컨텍스트의 이름입니다.

내용

name

컨텍스트의 이름.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

Pattern: `^([A-Za-z]_?)+$`

필수 여부: 예

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Intent

서비스: Amazon Lex Model Building Service

의도의 특정 개정 버전을 식별합니다.

내용

intentName

의도의 이름.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

Pattern: `^([A-Za-z]_?)+$`

필수 여부: 예

intentVersion

의도의 버전.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

Pattern: `\$LATEST|[0-9]+`

필수 여부: 예

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

IntentMetadata

서비스: Amazon Lex Model Building Service

의도에 대한 정보를 제공합니다.

내용

createdDate

의도가 생성된 날짜입니다.

유형: 타임스탬프

필수 여부: 아니요

description

의도에 대한 설명.

유형: 문자열

길이 제한: 최소 길이는 0. 최대 길이는 200.

필수 여부: 아니요

lastUpdatedDate

의도가 업데이트된 날짜. 봇을 생성할 때 생성 날짜 및 최종 업데이트 날짜가 동일합니다.

유형: 타임스탬프

필수 여부: 아니요

name

의도의 이름.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

Required: No

version

의도의 버전.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

KendraConfiguration

서비스: Amazon Lex Model Building Service

AMAZON.KendraSearchInten 의도에 대한 구성 정보를 제공합니다. 이 의도를 사용할 때 Amazon Lex는 지정된 Amazon Kendra 인덱스를 검색하고 인덱스에서 사용자의 표현과 일치하는 문서를 반환합니다. 자세한 내용은 AMAZON.KendraSearchIntent를 참조하세요.

내용

kendraIndex

AMAZON.KendraSearchIntent 의도가 검색할 Amazon Kendra 인덱스의 Amazon 리소스 이름 (ARN)입니다. 인덱스는 Amazon Lex 봇과 동일한 계정 및 리전에 있어야 합니다. Amazon Kendra 인덱스가 없는 경우 `PutIntent` 작업을 호출할 때 예외가 발생합니다.

유형: 문자열

길이 제약: 최소 길이는 20. 최대 길이는 2,048.

Pattern: `arn:aws:kendra:[a-z]+-[a-z]+-[0-9]:[0-9]{12}:index\/[a-zA-Z0-9][a-zA-Z0-9_-]*`

필수 여부: 예

role

Amazon Kendra 인덱스를 검색할 권한이 있는 IAM 역할의 Amazon 리소스 이름 (ARN) 입니다. 인덱스는 Amazon Lex 봇과 동일한 계정 및 리전에 있어야 합니다. 역할이 없는 경우 `PutIntent` 작업을 호출할 때 예외가 발생합니다.

유형: 문자열

길이 제약: 최소 길이는 20. 최대 길이는 2,048.

Pattern: `arn:aws:iam::[0-9]{12}:role/.*`

필수 여부: 예

queryFilterString

쿼리의 응답을 필터링하기 위해 Amazon Lex에서 Amazon Kendra에 전송하는 쿼리 필터입니다. 필터는 Amazon Kendra에서 정의한 형식입니다. 자세한 내용은 쿼리 필터링을 참조하세요.

런타임 시 이 필터 문자열을 새 필터 문자열로 재정의할 수 있습니다.

유형: 문자열

길이 제약: 최소 길이는 0.

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

LogSettingsRequest

서비스: Amazon Lex Model Building Service

대화 로그의 전달 모드 및 대상을 구성하는 데 사용되는 설정입니다.

내용

destination

로그가 전송될 위치. 텍스트 로그는 CloudWatch Logs 로그 그룹에 전송됩니다. S3 버킷으로 오디오 로그가 전송됩니다.

타입: 문자열

유효 값: `CLOUDWATCH_LOGS | S3`

필수 사항 여부: 예

logType

활성화할 로그 유형의 목록입니다. 텍스트 로그는 CloudWatch Logs 로그 그룹에 전송됩니다. S3 버킷으로 오디오 로그가 전송됩니다.

타입: 문자열

유효 값: `AUDIO | TEXT`

필수 사항 여부: 예

resourceArn

로그가 전송되어야 하는 S3 버킷이나 CloudWatch Logs 로그 그룹의 Amazon 리소스 이름 (ARN) 입니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

Pattern: `^arn:[\w\-]+:(?:logs:[\w\-]+:[\d]{12}:log-group:[\.\-_/#A-Za-z0-9]{1,512}(?::\*)?|s3:::[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9])$`

필수 여부: 예

kmsKeyArn

S3 버킷으로 전송되는 오디오 로그를 암호화하기 위한 AWS KMS 고객 관리 키의 Amazon 리소스 이름 (ARN) 입니다. 키는 CloudWatch Logs에는 적용되지 않으며 S3 버킷의 경우 선택 사항입니다.

유형: 문자열

길이 제약: 최소 길이는 20. 최대 길이는 2,048.

패턴: `^arn:[\w\-]+:kms:[\w\-]+:[\d]{12}:(?:key\/[\w\-]+|alias\/[a-zA-Z0-9:\/_\-]{1,256})$`

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

LogSettingsResponse

서비스: Amazon Lex Model Building Service

대화 로그 설정.

내용

destination

로그가 전달되는 대상.

타입: 문자열

유효 값: `CLOUDWATCH_LOGS | S3`

필수 여부: 아니요

kmsKeyArn

S3 버킷의 오디오 로그 파일을 암호화하기 위한 키의 Amazon 리소스 이름(ARN).

유형: 문자열

길이 제약: 최소 길이는 20. 최대 길이는 2,048.

패턴: `^arn:[\w\-]+:kms:[\w\-]+:[\d]{12}:(?:key\/[\w\-]+|alias\/[a-zA-Z0-9:\/_\-]{1,256})$`

Required: No

logType

활성화된 로깅 유형.

타입: 문자열

유효 값: `AUDIO | TEXT`

필수 여부: 아니요

resourceArn

로그가 전송되어야 하는 S3 버킷이나 CloudWatch Logs 로그 그룹의 Amazon 리소스 이름 (ARN).

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

패턴: `^arn:[\w\-]+:(?:logs:[\w\-]+:[\d]{12}:log-group:[\.\-_/#A-Za-z0-9]{1,512}(?::\*)?|s3:::[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9])$`

Required: No

resourcePrefix

리소스 접두사는 오디오 로그를 포함하도록 지정한 S3 버킷 내 S3 객체 키의 첫 번째 부분입니다. CloudWatch Logs의 경우, 지정한 로그 그룹 내 로그 스트림 이름의 접두사입니다.

유형: 문자열

길이 제약: 최대 길이는 1,024.

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Message

서비스: Amazon Lex Model Building Service

메시지 텍스트와 유형을 제공하는 객체입니다.

내용

content

메시지의 텍스트.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 1,000.

필수 여부: 예

contentType

메시지 문자열의 콘텐츠 유형.

타입: 문자열

유효 값: `PlainText | SSML | CustomPayload`

필수 사항 여부: 예

groupNumber

메시지가 속한 메시지 그룹을 식별합니다. 메시지가 그룹에 할당되면 Amazon Lex는 응답의 각 그룹에서 메시지 하나를 반환합니다.

타입: 정수

유효한 범위: 최소값은 1. 최댓값은 5.

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2

- AWS SDK for Ruby V3

MigrationAlert

서비스: Amazon Lex Model Building Service

Amazon Lex가 마이그레이션 중에 보내는 알림 및 경고에 대한 정보를 제공합니다. 알림에는 문제를 해결하는 방법에 대한 정보가 포함됩니다.

내용

details

알림에 대한 추가 세부 정보.

유형: 문자열 배열

필수 여부: 아니요

message

경고가 발생한 이유를 설명하는 메시지.

유형: 문자열

필수 항목 여부: 아니요

referenceURLs

알림 해결 방법을 설명하는 Amazon Lex 설명서 링크입니다.

유형: 문자열 배열

필수 여부: 아니요

type

오류의 유형. 다음 두 가지 종류의 알림이 있습니다.

- `ERROR` - 마이그레이션 중에 해결할 수 없는 문제가 발생했습니다. 마이그레이션이 중지됩니다.
- `WARN` - 마이그레이션 중에 새 Amazon Lex V2 봇을 수동으로 변경해야 하는 문제가 발생했습니다. 마이그레이션은 계속됩니다.

타입: 문자열

유효 값: `ERROR | WARN`

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

MigrationSummary

서비스: Amazon Lex Model Building Service

Amazon Lex V1에서 Amazon Lex V2로 봇을 마이그레이션하는 방법에 대한 정보를 제공합니다.

내용

migrationId

Amazon Lex가 마이그레이션에 할당한 고유 식별자입니다.

유형: 문자열

길이 제약 조건: 고정 길이는 10.

패턴: `^[0-9a-zA-Z]+$`

Required: No

migrationStatus

작업의 상태. 상태가 `COMPLETE`이면 Amazon Lex V2에서 봇을 사용할 수 있습니다. 마이그레이션을 완료하기 위해 해결해야 하는 알림 및 경고가 있을 수 있습니다.

타입: 문자열

유효 값: `IN_PROGRESS | COMPLETED | FAILED`

필수 여부: 아니요

migrationStrategy

마이그레이션을 수행하는 데 사용된 전략.

타입: 문자열

유효 값: `CREATE_NEW | UPDATE_EXISTING`

필수 여부: 아니요

migrationTimestamp

마이그레이션이 시작된 날짜와 시간.

유형: 타임스탬프

필수 여부: 아니요

v1BotLocale

마이그레이션의 소스가 되는 Amazon Lex V1 봇의 로캘.

타입: 문자열

유효 값: `de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR`

필수 여부: 아니요

v1BotName

마이그레이션의 소스가 되는 Amazon Lex V1 봇의 이름.

유형: 문자열

길이 제한: 최소 길이는 2. 최대 길이는 50.

패턴: `^([A-Za-z]_?)+$`

Required: No

v1BotVersion

마이그레이션의 소스가 되는 Amazon Lex V1 봇의 버전.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

Required: No

v2BotId

마이그레이션의 대상이 되는 Amazon Lex V2의 고유 식별자입니다.

유형: 문자열

길이 제약 조건: 고정 길이는 10.

패턴: `^[0-9a-zA-Z]+$`

Required: No

v2BotRole

Amazon Lex가 Amazon Lex V2 봇을 실행하는 데 사용하는 IAM 역할.

유형: 문자열

길이 제약: 최소 길이는 20. 최대 길이는 2,048.

패턴: `^arn:[\w\-]+:iam::[\d]{12}:role/.+$`

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

OutputContext

서비스: Amazon Lex Model Building Service

의도가 이행될 때 출력 컨텍스트의 사양.

내용

name

컨텍스트의 이름.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

Pattern: `^([A-Za-z]_?)+$`

필수 여부: 예

timeToLiveInSeconds

`PostContent`또는 `PostText` 응답으로 컨텍스트가 처음 전송된 후 컨텍스트가 활성화되어야 하는 시간(초). 5초에서 86,400초(24시간) 사이를 설정할 수 있습니다.

유형: 정수

유효한 범위: 최소값은 5. 최대값은 86,400.

필수 여부: 예

turnsToLive

컨텍스트가 활성 상태로 유지되어야 하는 대화 턴 수입니다. 대화 턴은 하나의 `PostContent` 또는 `PostText` 요청 및 Amazon Lex로부터의 관련 응답입니다.

타입: 정수

유효한 범위: 최소값은 1. 최대값은 20.

필수 여부: 예

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Prompt

서비스: Amazon Lex Model Building Service

사용자로부터 정보를 얻습니다. 프롬프트를 정의하려면 메시지를 하나 이상 제공하고 사용자로부터 정보를 얻으려는 시도 횟수를 지정합니다. 메시지를 두 개 이상 제공하는 경우 Amazon Lex는 메시지 중 하나를 선택하여 사용자에게 메시지를 표시합니다. 자세한 내용은 Amazon Lex: 작동 방식을 참조하십시오.

내용

maxAttempts

사용자에게 정보를 요청하는 횟수입니다.

타입: 정수

유효한 범위: 최소값은 1. 최대값은 5.

필수 여부: 예

messages

각각 메시지 문자열과 유형을 제공하는 객체의 배열입니다. 일반 텍스트 또는 음성 합성 마크업 언어(SSML)로 메시지 문자열을 지정할 수 있습니다.

유형: Message객체 어레이

배열 멤버: 최소 항목 수는 1개. 최대 항목 수는 15개.

필수 여부: 예

responseCard

응답 카드. Amazon Lex는 런타임 시 `PostText` API 응답에서 이 프롬프트를 사용합니다. 응답 카드의 자리 표시자를 세션 속성과 슬롯 값으로 대체합니다. 자세한 내용은 응답 카드 사용 단원을 참조하십시오.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 50,000.

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ResourceReference

서비스: Amazon Lex Model Building Service

삭제하려는 리소스를 참조하는 리소스를 설명합니다. 이 개체는 `ResourceInUseException` 예외의 일부로 반환됩니다.

내용

name

삭제하려는 리소스를 사용 중인 리소스의 이름.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `[a-zA-Z_]+`

Required: No

version

삭제하려는 리소스를 사용 중인 리소스의 버전.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Slot

서비스: Amazon Lex Model Building Service

슬롯의 특정 버전을 식별합니다.

내용

name

슬롯의 이름.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

Pattern: `^([A-Za-z](-|_|.)?)+$`

필수 여부: 예

slotConstraint

슬롯이 필수인지 옵션인지 여부를 지정합니다.

타입: 문자열

유효 값: `Required | Optional`

필수 사항 여부: 예

defaultValueSpec

슬롯의 기본값 목록입니다. Amazon Lex가 슬롯에 대한 값을 결정하지 않은 경우 기본값이 사용됩니다. 컨텍스트 변수, 세션 속성 및 정의된 값에서 기본값을 지정할 수 있습니다.

유형: SlotDefaultValueSpec 객체

필수 여부: 아니요

description

슬롯 유형에 대한 설명.

유형: 문자열

길이 제한: 최소 길이는 0. 최대 길이는 200.

필수 여부: 아니요

obfuscationSetting

대화 로그와 저장된 표현 내용에서 슬롯이 난독화되는지 여부를 결정합니다. 슬롯을 난독 처리하면 값이 중괄호 ({})로 묶인 슬롯 이름으로 대체됩니다. 예를 들어 슬롯 이름이 “full_name”인 경우 난독화된 값은 "{full_name}"으로 대체됩니다. 자세한 내용은 슬롯 난독화 를 참조하십시오.

타입: 문자열

유효 값: `NONE | DEFAULT_OBFUSCATION`

필수 여부: 아니요

priority

사용자로부터 이 슬롯 값을 추출하는 순서를 Amazon Lex에 지시합니다. 예를 들어 의도에 우선 순위가 1과 2인 슬롯 두 개가 있는 경우 AWS Amazon Lex는 우선 순위가 1인 슬롯의 값을 먼저 추출합니다.

여러 슬롯이 동일한 우선 순위를 공유하는 경우 Amazon Lex가 값을 도출하는 순서는 임의적입니다.

유형: 정수

유효 범위: 최소값은 0입니다. 최댓값은 100입니다.

필수 여부: 아니요

responseCard

텍스트 기반 클라이언트가 사용하는 슬롯 유형에 대해 가능한 응답 세트입니다. 사용자는 텍스트를 사용하여 회신하는 대신 응답 카드에서 옵션을 선택합니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 50,000.

필수 여부: 아니요

sampleUtterances

사용자가 슬롯 값에 대한 Amazon Lex 요청에 응답할 수 있는 특정 패턴을 알고 있는 경우 이러한 표현을 제공하여 정확도를 높일 수 있습니다. 이는 선택 사항입니다. 대부분의 경우 Amazon Lex는 사용자의 표현을 이해할 수 있습니다.

유형: 문자열 배열

배열 멤버: 최소 항목 수는 0개. 최대 항목 수는 10개.

길이 제약 조건: 최소 길이는 1입니다. 최대 길이는 200.

필수 여부: 아니요

slotType

슬롯 유형(정의한 사용자 정의 슬롯 유형 또는 기본 제공 슬롯 유형 중 하나)입니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^((AMAZON\.)_?|[A-Za-z]_?)+`

Required: No

slotTypeVersion

슬롯 유형의 버전.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

Required: No

valueElicitationPrompt

Amazon Lex가 사용자로부터 슬롯 값을 추출하는 데 사용하는 프롬프트입니다.

유형: Prompt 객체

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++

- AWS SDK for Java V2
- AWS SDK for Ruby V3

SlotDefaultValue

서비스: Amazon Lex Model Building Service

슬롯의 기본값.

내용

defaultValue

슬롯의 기본값. 다음 중 한 가지를 지정할 수 있습니다.

- `#context-name.slot-name` - "컨텍스트 이름" 컨텍스트의 슬롯 값 "슬롯 이름".
- `{attribute}` - 세션 속성 "속성"의 슬롯 값.
- `'value'` - 불연속 값 "값".

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 202.

필수 여부: 예

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

SlotDefaultValueSpec

서비스: Amazon Lex Model Building Service

슬롯의 기본값을 포함합니다. Amazon Lex가 슬롯에 대한 값을 결정하지 않은 경우 기본값이 사용됩니다.

내용

defaultValueList

슬롯의 기본값. 둘 이상의 값을 지정할 수 있습니다. 예를 들어 일치하는 컨텍스트 변수, 세션 속성 또는 고정 값에서 사용할 기본값을 지정할 수 있습니다.

선택한 기본값은 목록에서 지정한 순서에 따라 선택됩니다. 예를 들어 컨텍스트 변수와 고정 값을 순서대로 지정하는 경우 Amazon Lex는 컨텍스트 변수를 사용하고, 사용 가능한 경우 고정 값을 사용합니다.

유형: SlotDefaultValue객체 어레이

배열 멤버: 최소 항목 수는 0개. 최대 항목 수는 10개.

필수 여부: 예

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

SlotTypeConfiguration

서비스: Amazon Lex Model Building Service

슬롯 유형에 대한 구성 정보를 제공합니다.

내용

regexConfiguration

슬롯 값을 검증하는 데 사용되는 정규식.

유형: SlotTypeRegexConfiguration 객체

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

SlotTypeMetadata

서비스: Amazon Lex Model Building Service

슬롯 유형에 관한 정보를 제공합니다.

내용

createdDate

슬롯 유형이 생성된 날짜.

유형: 타임스탬프

필수 여부: 아니요

description

슬롯 유형에 대한 설명.

유형: 문자열

길이 제한: 최소 길이는 0. 최대 길이는 200.

필수 여부: 아니요

lastUpdatedDate

슬롯 유형이 업데이트된 날짜. 리소스를 생성할 때 생성 날짜 및 최종 업데이트 날짜가 동일합니다.

유형: 타임스탬프

필수 여부: 아니요

name

슬롯 유형의 이름.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

패턴: `^([A-Za-z]_?)+$`

Required: No

version

슬롯 유형의 버전.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

SlotTypeRegexConfiguration

서비스: Amazon Lex Model Building Service

슬롯 값을 검증하는 데 사용되는 정규식을 제공합니다.

내용

pattern

슬롯 값을 검증하는 데 사용되는 정규식.

표준 정규 표현식을 사용합니다. Amazon Lex는 정규 표현식에서 다음 문자를 지원합니다.

- A~Z, a~z
- 0~9
- 유니코드 문자("\u<Unicode>")

4자리 숫자가 포함된 유니코드 문자를 나타냅니다(예: "\u0041" 또는 "\u005A").

다음 정규식 연산자는 지원되지 않습니다.

- 무한 반복자: 상한이 없는 *, + 또는 {x,}
- 와일드 카드(.)

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

필수 여부: 예

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Statement

서비스: Amazon Lex Model Building Service

사용자에게 정보를 전달하는 메시지 모음. Amazon Lex는 런타임 시 전달할 메시지를 선택합니다.

내용

messages

메시지 객체의 모음.

유형: Message객체 어레이

배열 멤버: 최소 항목 수는 1개. 최대 항목 수는 15개.

필수 여부: 예

responseCard

런타임 시 클라이언트가 PostText API를 사용하는 경우 Amazon Lex는 응답에 응답 카드를 포함합니다. 응답 카드의 자리 표시자를 세션 속성과 슬롯 값으로 대체합니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 50,000.

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Tag

서비스: Amazon Lex Model Building Service

봇, 봇 별칭 또는 봇 채널을 식별하는 키/값 쌍의 목록. 키와 값의 문자로는 Unicode 문자, 숫자, 공백 그리고 다음 기호가 허용됩니다:_ . : / = + - @.

내용

key

태그의 키. 키는 대/소문자를 구분하지 않으며 고유해야 합니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 128.

필수 여부: 예

value

키에 연결할 값입니다. 값은 빈 문자열일 수 있지만 null일 수는 없습니다.

유형: 문자열

길이 제약: 최소 길이는 0. 최대 길이는 256.

필수 여부: 예

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

UtteranceData

서비스: Amazon Lex Model Building Service

봇에 생성된 단일 표현에 대한 정보를 제공합니다

내용

count

표현이 처리된 횟수.

유형: 정수

필수 항목 여부: 아니요

distinctUsers

해당 표현을 사용한 총 개인 수.

유형: 정수

필수 항목 여부: 아니요

firstUtteredDate

표현이 처음 기록된 날짜.

유형: 타임스탬프

필수 여부: 아니요

lastUtteredDate

표현이 마지막으로 기록된 날짜.

유형: 타임스탬프

필수 여부: 아니요

utteranceString

사용자가 입력한 텍스트 또는 오디오 클립의 텍스트 표현.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,000.

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

UtteranceList

서비스: Amazon Lex Model Building Service

봇의 특정 버전에 대한 표현 목록을 제공합니다. 목록에는 최대 100개의 표현이 포함됩니다.

내용

botVersion

목록을 처리한 봇의 버전입니다.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: `\$LATEST|[0-9]+`

Required: No

utterances

봇에게 한 표현에 대한 정보가 들어 있는 하나 이상의 UtteranceData 개체. 최대 객체 개수는 100.

유형: UtteranceData 객체 배열

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Amazon Lex 런타임 서비스

다음 데이터 유형이 Amazon Lex 런타임 서비스에서 지원됩니다.

- ActiveContext
- ActiveContextTimeToLive

- Button
- DialogAction
- GenericAttachment
- IntentConfidence
- IntentSummary
- PredictedIntent
- ResponseCard
- SentimentResponse

ActiveContext

서비스: Amazon Lex Runtime Service

컨텍스트는 사용자와 Amazon Lex 간의 현재 대화 상태에 대한 정보를 포함하는 변수입니다. 컨텍스트는 의도가 이행될 때 Amazon Lex에서 자동으로 설정하거나 `PutContent`, `PutText`, 또는 `PutSession` 작업을 사용하여 런타임 시 설정할 수 있습니다.

내용

name

컨텍스트의 이름.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 100.

Pattern: `^([A-Za-z]_?)+$`

필수 여부: 예

parameters

현재 컨텍스트의 상태 변수. 이 값을 후속 이벤트에서 슬롯의 기본값으로 사용할 수 있습니다.

유형: 문자열 간 맵

맵 항목: 최소 항목 수는 0개. 최대 항목 수는 10개.

키 길이 제약 조건: 최소 길이는 1. 최대 길이는 100.

값 길이 제약 조건: 최소 길이는 1. 최대 길이는 1,024.

필수 여부: 예

timeToLive

컨텍스트가 활성 상태로 유지되는 시간 또는 턴 수.

유형: ActiveContextTimeToLive 객체

필수 여부: 예

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ActiveContextTimeToLive

서비스: Amazon Lex Runtime Service

컨텍스트가 활성 상태로 유지되는 시간 또는 턴 수.

내용

timeToLiveInSeconds

`PostContent` 또는 `PostText` 응답으로 컨텍스트가 처음 전송된 후 컨텍스트가 활성화되어야 하는 시간(초). 5초에서 86,400초(24시간) 사이를 설정할 수 있습니다.

유형: 정수

유효한 범위: 최소값은 5. 최대값은 86,400.

필수 여부: 아니요

turnsToLive

컨텍스트가 활성 상태로 유지되어야 하는 대화 턴 수입니다. 대화 턴은 하나의 `PostContent` 또는 `PostText` 요청 및 Amazon Lex로부터의 관련 응답입니다.

타입: 정수

유효한 범위: 최소값은 1. 최대값은 20.

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Button

서비스: Amazon Lex Runtime Service

클라이언트 플랫폼(페이스북, 슬랙 등)에 표시할 옵션을 나타냅니다.

내용

text

버튼에서 사용자에게 표시되는 텍스트.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 15.

필수 여부: 예

value

사용자가 버튼을 선택할 때 Amazon Lex로 전송되는 값입니다. 버튼 텍스트 “NYC”를 예로 들어 보겠습니다. 사용자가 버튼을 선택하면 전송되는 값은 “뉴욕 시”가 될 수 있습니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 1,000.

필수 여부: 예

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

DialogAction

서비스: Amazon Lex Runtime Service

봇이 사용자와의 상호작용에서 취해야 하는 다음 조치를 설명하고 작업이 발생하는 상황에 대한 정보를 제공합니다. `DialogAction` 데이터 유형을 사용하여 상호 작용을 특정 상태로 설정하거나 상호 작용을 이전 상태로 되돌릴 수 있습니다.

내용

type

봇이 사용자와의 상호 작용에서 수행해야 하는 다음 작업입니다. 가능한 값은 다음과 같습니다.

- `ConfirmIntent` - 다음 작업은 사용자에게 의도가 완료되었고 이행할 준비가 되었는지 묻는 것입니다. “주문하세요?”와 같은 예/아니요 질문입니다.
- `Close` - 사용자로부터 응답이 없을 것임을 나타냅니다. 예를 들어 "주문 되었습니다"라는 문장에는 응답이 필요 없습니다.
- `Delegate` - Amazon Lex가 다음 작업을 결정합니다.
- `ElicitIntent` - 다음 작업은 사용자가 이행하고자 하는 의도를 결정하는 것입니다.
- `ElicitSlot` - 다음 작업은 사용자로부터 슬롯 값을 추출하는 것입니다.

타입: 문자열

유효 값: `ElicitIntent | ConfirmIntent | ElicitSlot | Close | Delegate`

필수 사항 여부: 예

fulfillmentState

의도에 대한 이행 상태. 가능한 값은 다음과 같습니다.

- `Failed` - 의도와 연결된 Lambda 함수가 의도를 이행하지 못했습니다.
- `Fulfilled` - 의도는 의도와 연결된 Lambda 함수에 의해 이행되었습니다.
- `ReadyForFulfillment` - 의도에 필요한 모든 정보가 존재하며 클라이언트 애플리케이션에서 의도를 이행할 준비가 되어 있습니다.

타입: 문자열

유효 값: `Fulfilled | Failed | ReadyForFulfillment`

필수 여부: 아니요

intentName

의도의 이름.

유형: 문자열

필수 항목 여부: 아니요

message

사용자에게 표시해야 하는 메시지입니다. 사용자가 메시지를 지정하지 않으면 Amazon Lex는 의도에 대해 구성된 메시지를 사용합니다.

유형: 문자열

길이 제약: 최소 길이 1. 최대 길이 1024.

필수 여부: 아니요

messageFormat

- `PlainText` - 메시지에 일반 UTF-8 텍스트가 포함됩니다.
- `CustomPayload` - 메시지는 클라이언트의 사용자 지정 형식입니다.
- `SSML` - 메시지에 음성 출력용으로 서식이 지정된 텍스트가 포함됩니다.
- `Composite` - 메시지에 하나 이상의 메시지가 포함된 이스케이프된 JSON 객체가 포함되어 있습니다. 더 자세한 내용은, 메시지 그룹을 참조하십시오.

타입: 문자열

유효 값: `PlainText | CustomPayload | SSML | Composite`

필수 여부: 아니요

slots

수집된 슬롯과 해당 값의 맵.

유형: 문자열 간 맵

필수 여부: 아니요

slotToElicit

사용자로부터 추출해야 하는 슬롯의 이름입니다.

유형: 문자열

필수 항목 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

GenericAttachment

서비스: Amazon Lex Runtime Service

프롬프트가 표시될 때 사용자에게 렌더링되는 옵션을 나타냅니다. 이미지, 버튼, 링크 또는 텍스트일 수 있습니다.

내용

attachmentLinkUrl

응답 카드에 있는 첨부 파일의 URL.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 여부: 아니요

buttons

사용자에게 표시할 옵션 목록.

유형: Button객체 어레이

배열 멤버: 최소 항목 수 0개. 최대 항목 수 5개.

필수 여부: 아니요

imageUrl

사용자에게 표시되는 이미지의 URL.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 여부: 아니요

subTitle

제목 아래에 표시된 자막.

유형: 문자열

길이 제약: 최소 길이 1. 최대 길이는 80.

필수 여부: 아니요

title

옵션의 제목.

유형: 문자열

길이 제약: 최소 길이 1. 최대 길이는 80.

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

IntentConfidence

서비스: Amazon Lex Runtime Service

의도가 사용자의 의도를 만족하는 것인지 여부에 대한 Amazon Lex의 신뢰도를 나타내는 점수를 제공합니다.

내용

score

Amazon Lex가 의도가 사용자의 의도를 만족시키는지 신뢰하는 정도를 나타내는 점수입니다. 범위는 0.00에서 1.00 사이입니다. 점수가 높을수록 신뢰도가 높습니다.

유형: 더블

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

IntentSummary

서비스: Amazon Lex Runtime Service

의도의 상태에 대한 정보를 제공합니다. 이 정보를 사용하여 의도의 현재 상태를 파악하여 의도를 처리하거나 의도를 이전 상태로 되돌릴 수 있습니다.

내용

dialogActionType

봇이 사용자와의 상호 작용에서 수행해야 하는 다음 작업입니다. 가능한 값은 다음과 같습니다.

- `ConfirmIntent` - 다음 작업은 사용자에게 의도가 완료되었고 이행할 준비가 되었는지 묻는 것입니다. “주문하세요?”와 같은 예/아니요 질문입니다.
- `Close` - 사용자로부터 응답이 없을 것임을 나타냅니다. 예를 들어 "주문 되었습니다"라는 문장에는 응답이 필요 없습니다.
- `ElicitIntent` - 다음 작업은 사용자가 이행하고자 하는 의도를 결정하는 것입니다.
- `ElicitSlot` - 다음 작업은 사용자로부터 슬롯 값을 추출하는 것입니다.

타입: 문자열

유효 값: `ElicitIntent | ConfirmIntent | ElicitSlot | Close | Delegate`

필수 사항 여부: 예

checkpointLabel

특정 의도를 식별하는 사용자 정의 라벨입니다. 이 레이블을 사용하여 이전 의도로 돌아갈 수 있습니다.

`GetSessionRequest` 작업의 `checkpointLabelFilter` 매개변수를 사용하여 작업에서 반환된 의도를, 지정된 라벨만 있는 의도로 필터링합니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 255.

패턴: `[a-zA-Z0-9-]+`

Required: No

confirmationStatus

사용자가 확인 프롬프트에 응답한 이후의 의도 상태입니다. 사용자가 의도를 확인하면 Amazon Lex는 이 필드를 `Confirmed`로 설정합니다. 사용자가 의도를 거부하면 Amazon Lex는 이 값을 `Denied`로 설정합니다. 가능한 값은 다음과 같습니다.

- `Confirmed` - 사용자가 확인 프롬프트에 “예”라고 응답하여 의도가 완료되었으며 이행할 준비가 되었음을 확인했습니다.
- `Denied` - 사용자가 확인 프롬프트에 “아니요”라고 응답했습니다.
- `None` - 사용자에게 확인 메시지가 표시되지 않거나 사용자에게 메시지가 표시되었지만 메시지를 확인하거나 거부하지 않았습니다.

타입: 문자열

유효 값: `None | Confirmed | Denied`

필수 여부: 아니요

fulfillmentState

의도의 이행 상태. 가능한 값은 다음과 같습니다.

- `Failed` - 의도와 연결된 Lambda 함수가 의도를 이행하지 못했습니다.
- `Fulfilled` - 의도는 의도와 연결된 Lambda 함수에 의해 이행되었습니다.
- `ReadyForFulfillment` - 의도에 필요한 모든 정보가 존재하며 클라이언트 애플리케이션에서 의도를 이행할 준비가 되어 있습니다.

타입: 문자열

유효 값: `Fulfilled | Failed | ReadyForFulfillment`

필수 여부: 아니요

intentName

의도의 이름.

유형: 문자열

필수 항목 여부: 아니요

slots

수집된 슬롯과 해당 값의 맵.

유형: 문자열 간 맵

필수 여부: 아니요

slotToElicit

사용자로부터 유도할 다음 슬롯. 유도할 슬롯이 없는 경우 해당 필드는 비어 있습니다.

유형: 문자열

필수 항목 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

PredictedIntent

서비스: Amazon Lex Runtime Service

Amazon Lex에서 제안하는 의도는 사용자의 의도를 만족시킵니다. 의도의 이름, Amazon Lex가 사용자의 의도가 만족되었다는 신뢰도, 의도에 대해 정의된 슬롯을 포함합니다.

내용

intentName

Amazon Lex에서 제안하는 의도 이름은 사용자의 의도를 만족합니다.

유형: 문자열

필수 항목 여부: 아니요

nluIntentConfidence

Amazon Lex가 의도가 사용자의 의도를 만족한다고 얼마나 신뢰하는지 나타냅니다.

유형: IntentConfidence 객체

필수 여부: 아니요

slots

예측된 의도와 관련된 슬롯 및 슬롯 값.

유형: 문자열 간 맵

필수 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ResponseCard

서비스: Amazon Lex Runtime Service

봇을 생성할 때 응답 카드를 구성하면 Amazon Lex는 사용 가능한 세션 속성과 슬롯 값을 대체한 다음 이를 반환합니다. 응답 카드는 Lambda 함수 (`dialogCodeHook` 및 `fulfillmentActivity` 인텐트)에서 가져올 수도 있습니다.

내용

contentType

응답의 콘텐츠 유형.

타입: 문자열

유효 값: `application/vnd.amazonaws.card.generic`

필수 여부: 아니요

genericAttachments

옵션을 나타내는 첨부 객체 배열.

유형: GenericAttachment객체 어레이

배열 멤버: 최소 항목 수는 0개. 최대 항목 수 10개.

필수 여부: 아니요

version

응답 카드 형식의 버전.

유형: 문자열

필수 항목 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2

- AWS SDK for Ruby V3

SentimentResponse

서비스: Amazon Lex Runtime Service

감정이 하나의 표현으로 표현되었습니다.

봇이 감정 분석을 위해 Amazon Comprehend로 표현을 보내도록 구성된 경우 이 필드에 분석 결과가 포함됩니다.

내용

sentimentLabel

Amazon Comprehend가 가장 신뢰하는 추론된 감정입니다.

유형: 문자열

필수 항목 여부: 아니요

sentimentScore

감정이 올바르게 추론되었을 가능성.

유형: 문자열

필수 항목 여부: 아니요

참고

언어별 AWS SDKs

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Amazon Lex 문서 기록

- 설명서 최종 업데이트: 2021년 9월 9일

다음 표에서는 Amazon Lex의 각 릴리스에서 변경된 중요 사항에 대해 설명합니다. 이 설명서에 대한 업데이트 알림을 받으려면 RSS 피드를 구독하세요.

변경 사항	설명	날짜
새로운 특성	Amazon Lex는 이제 한국어(Ko-KR) 로캘을 지원합니다. 자세한 내용은 Amazon Lex에서 지원되는 언어를 참조하세요.	2021년 9월 9일
새 기능	Amazon Lex는 이제 영어(인도) 로캘을 지원합니다. 자세한 내용은 Amazon Lex에서 지원되는 언어를 참조하세요.	2021년 7월 15일
새 기능	Amazon Lex는 이제 봇을 Amazon Lex V2 API로 마이그레이션할 수 있는 도구를 제공합니다. 자세한 내용은 마이그레이션 봇을 참조하세요.	2021년 7월 13일
새 기능	Amazon Lex는 이제 일본어(일본) 로캘을 지원합니다. 자세한 내용은 Amazon Lex에서 지원되는 언어를 참조하세요.	2021년 4월 1일
새 기능	Amazon Lex는 이제 독일어(독일)(de-DE) 및 스페인어(라틴 아메리카)(es-419) 로캘을 지원합니다. 자세한 내용은 Amazon Lex에서 지원되는 언어를 참조하세요.	2020년 11월 23일

새 기능	Amazon Lex는 이제 컨텍스트를 사용하여 활성화된 의도를 관리할 수 있습니다. 자세한 내용은 의도 컨텍스트 설정을 참조하세요.	2020년 11월 19일
새 기능	Amazon Lex는 이제 프랑스어(fr-FR), 프랑스령 캐나다(fr-CA), 이탈리아어 (IT-it) 및 스페인어(es-ES) 로캘을 지원합니다. 지원되는 전체 로캘 목록은 Amazon Lex에서 지원하는 언어를 참조하세요.	2020년 11월 11일
새 기능	Amazon Lex는 이제 스페인어(미국)(ES-미국) 로캘을 지원합니다. 자세한 내용은 Amazon Lex에서 지원되는 언어를 참조하세요.	2020년 9월 22일
새 기능	Amazon Lex는 이제 영어(영국)(en-GB) 로캘을 지원합니다. 자세한 내용은 Amazon Lex에서 지원되는 언어를 참조하세요.	2020년 9월 15일
새 기능	Amazon Lex는 이제 영어(오스트레일리아)(en-AU) 로캘을 지원합니다. 자세한 내용은 Amazon Lex에서 지원되는 언어를 참조하세요.	2020년 9월 8일

새 기능	Amazon Lex는 이제 7개의 새로운 기본 제공 의도와 9개의 새로운 기본 제공 슬롯 유형을 제공합니다. 자세한 내용은 기본 제공 의도 및 슬롯 유형을 참조하세요.	2020년 9월 8일
새 예제	고객 지원 에이이전트가 Amazon Kendra 에서 답변을 검색하여 고객 질문에 답변하는 데 사용할 수 있는 Amazon Lex 봇을 만드는 방법을 알아봅니다. 자세한 내용은 예: 콜 센터 에이전트 어시스턴트를 참조하세요.	2020년 8월 10일
새 기능	Amazon Lex는 이제 신뢰도 점수를 기반으로 최대 4개의 대체 의도를 반환할 수 있습니다. 자세한 내용은 신뢰도 점수를 참조하세요.	2020년 8월 6일
리전 확장	Amazon Lex는 이제 아시아 태평양(도쿄)(ap-Northast-1)에서 사용할 수 있습니다.	2020년 6월 30일
새 기능	Amazon Lex에서는 이제 자주 묻는 질문에 대한 답변을 찾기 위한 Amazon Kendra 인덱스 검색을 지원합니다. 자세한 내용은 AMAZON.KendraSearchIntent를 참조하세요.	2020년 6월 11일

새 기능	Amazon Lex는 이제 대화 로그의 더 많은 정보를 반환합니다. 자세한 내용은 Amazon CloudWatch Logs 텍스트 로그 보기를 참조하세요.	2020년 6월 9일
리전 확장	이제 아시아 태평양(싱가포르)(ap-southeast-1), 유럽(프랑크푸르트)(eu-wetheast-1) 및 유럽(런던)(eu-west-2) 에서 Amazon Lex를 사용할 수 있습니다.	2020년 4월 23일
새 기능	Amazon Lex는 이제 태그를 지원합니다. 태그 지정을 사용하여 리소스를 식별하고 비용을 할당하며 액세스를 제어할 수 있습니다. 자세한 내용은 Amazon Lex 리소스 태그를 참조하세요.	2020년 3월 12일
새 기능	Amazon Lex에서는 AMAZON.AlphaNumeric 기본 제공 슬롯 유형에 정규식을 지원합니다. 자세한 내용은 AMAZON.AlphaNumeric을 참조하세요.	2020년 2월 6일
새 기능	Amazon Lex는 이제 대화 정보를 기록하고 해당 로그의 슬롯 값을 난독화할 수 있습니다. 자세한 내용은 대화 로그 생성 및 슬롯 난독화를 참조하세요.	2019년 12월 19일
리전 확장	이제 아시아 태평양(시드니)(ap-southeast-2)에서 Amazon Lex를 사용할 수 있습니다.	2019년 12월 17일

새 기능	Amazon Lex는 이제 HIPAA를 준수합니다. 자세한 내용은 Amazon Lex 준수 검증을 참조하세요.	2019년 12월 10일
새 기능	Amazon Lex는 이제 사용자 표현을 Amazon Comprehend 에 보내 표현의 감정을 분석할 수 있습니다. 자세한 내용은 감정 분석을 참조하세요.	2019년 11월 21일
새 기능	Amazon Lex는 이제 SOC를 준수합니다. 자세한 내용은 Amazon Lex 준수 검증을 참조하세요.	2019년 11월 19일
새 기능	Amazon Lex는 이제 PCI를 준수합니다. 자세한 내용은 Amazon Lex 준수 검증을 참조하세요.	2019년 10월 17일
새 기능	대화 중 의도로 손쉽게 돌아갈 수 있도록 의도에 체크포인트를 추가하는 기능을 추가했습니다. 자세한 내용은 세션 관리를 참조하세요.	2019년 10월 10일
새 기능	사용자 입력이 예상과 다를 때 봇이 이 상황을 처리할 수 있도록 `AMAZON.FallbackIntent` 에 대한 지원을 추가했습니다. 자세한 내용은 AMAZON.FallbackIntent를 참조하세요.	2019년 10월 3일

새 기능	Amazon Lex를 사용하면 봇의 세션 정보를 관리할 수 있습니다. 자세한 내용은 Amazon Lex API로 세션 관리를 참조하세요.	2019년 8월 8일
리전 확장	이제 미국 서부(오레곤)(us-west-2)에서 Amazon Lex를 사용할 수 있습니다.	2018년 5월 8일
새 기능	Amazon Lex 형식으로 내보내고 가져오는 작업에 대한 지원을 추가했습니다. 자세한 내용은 Amazon Lex 봇, 의도 및 슬롯 유형 가져오기 및 내보내기를 참조하세요.	2018년 2월 13일
새 기능	이제 Amazon Lex에서는 봇에 대한 추가 응답 메시지를 지원합니다. 자세한 내용은 응답을 참조하세요.	2018년 2월 8일
리전 확장	이제 유럽(아일랜드)(eu-west-1)에서 Amazon Lex를 사용할 수 있습니다.	2017년 11월 21일
새 기능	Kirk에서 Amazon Lex 봇을 배포하는 작업에 대한 지원을 추가했습니다. 자세한 내용은 Amazon Lex 봇을 Kik과 통합을 참조하세요.	2017년 11월 20일
새 기능	새로운 기본 제공 슬롯 유형과 요청 속성에 대한 지원을 추가했습니다. 자세한 내용은 기본 제공 슬롯 유형과 요청 속성 설정을 참조하세요.	2017년 11월 3일

새 기능	Alexa Skills Kit로 내보내기 기능을 추가했습니다. 자세한 내용은 Alexa Skill로 내보내기를 참조하세요.	2017년 9월 7일
새 기능	슬롯 유형 값에 대한 동의어 지원을 추가했습니다. 자세한 내용은 사용자 지정 슬롯 유형을 참조하세요.	2017년 8월 31일
새 기능	AWS CloudTrail 통합이 추가되었습니다. 자세한 내용은 AWS CloudTrail 로그를 사용한 Amazon Lex API 호출 모니터링을 참조하세요.	2017년 8월 15일
확장 설명서	에 대한 시작하기 예제가 추가되었습니다 AWS CLI. 자세한 내용은 https://docs.aws.amazon.com/lex/latest/dg/gs-cli.html 단원을 참조하세요.	2017년 5월 22일
새 가이드	이 설명서는 Amazon Lex 사용 설명서의 최초 릴리스입니다.	2017년 4월 19일

AWS 용어집

최신 AWS 용어는 AWS 용어집 참조의 AWS 용어집을 참조하세요.