AMAZON LEX: ANATOMIA DI UN CHATBOT

AI | Amazon Lex | AWS Lambda | Deep Learning

beSharp | 9 Agosto 2019

Con il crescente impiego dell'intelligenza artificiale e con l'avvento di nuove interfacce uomo-macchina, diventa sempre più importante esplorare le nuove tecnologie ed impiegarle al meglio per trarne la massima utilità ed il massimo valore.

Uno degli impieghi dell'intelligenza artificiale e del deep learning più orientato all'utente finale è certamente la realizzazione di nuovi tipi di **interfacce uomo-macchina** basate sul riconoscimento degli intenti, sia su messaggi di testo sia da interazioni vocali.

Per avere alcuni esempi di questa tipologia di applicazioni basta pensare agli assistenti vocali sugli smartphone, a quelli domestici come ad esempio Amazon Alexa e a tutti i servizi con cui è possibile interagire via chat.

In questo articolo vi mostreremo come realizzare uno **Slack bot usando Amazon Lex.**

In ambito informatico, riconoscimento vocale e comprensione del linguaggio naturale sono tra i problemi più complessi affrontati, tanto da necessitare algoritmi di deep learning molto complessi. Tali algoritmi richiedono enormi quantità di dati per essere addestrati efficacemente, oltre ad apposite infrastrutture di calcolo dai costi particolarmente elevati. **Amazon Lex rende queste tecnologie accessibili a tutti** aprendo la strada verso una categoria completamente nuova di prodotti.

Amazon Lex è un servizio per la creazione di interfacce di comunicazione tramite voce e testo realizzato per essere sufficientemente generico da essere impiegato in qualsiasi applicazione. Offre funzionalità avanzate di apprendimento per il riconoscimento vocale e la dettatura, nonché per il riconoscimento del linguaggio naturale e per la comprensione di testi, consentendo la creazione di applicazioni con una user-experience coinvolgente, basata su conversazioni realistiche. Con Amazon Lex, le stesse tecnologie di apprendimento approfondito su cui si basa Amazon Alexa sono a disposizione di tutti gli sviluppatori, consentendo così la creazione di **bot di conversazione** ("chatbot") sofisticati e naturali in modo semplice, veloce e a basso effort.

Lex consente una facile integrazione con Facebook Messenger, Slack e Twilio SMS.

Attualmente, Lex supporta solamente la lingua Inglese, ma è possibile realizzare bot in altre lingue costruendo una soluzione ad-hoc che sfrutti Amazon Translate.

Prima di entrare nei dettagli di implementazione occorre definire il gergo ed i concetti fondamentali di Lex.

Un bot realizzato mediante Lex si compone sostanzialmente di 3 parti:

1. **Modello di interazione;**
2. **Funzioni Lambda** (back-end) per elaborare le richieste;
3. **Una o più interfacce/canali di interazione** (Slack, SMS, telefono).

Ora definiamo meglio i concetti sopra citati.

Modello di interazione

Il modello di interazione consiste di un file json che modellizza tutti gli **intenti**, cioè il tipo di azioni rese possibili agli utenti e che il bot può gestire. Per ogni **intento** è necessario definire anche quali siano i **parametri** necessari e il loro **tipo**.

Questo file può essere redatto testualmente e fornito a Lex, oppure essere realizzato per via grafica utilizzando la web console di AWS.

Il modello contiene anche molte frasi di esempio per ogni intento (**utterances**), frasi che verranno utilizzate dalla IA per riconoscere le richieste degli utenti verso il bot ed indirizzarle opportunamente alle specifiche funzioni Lambda per essere processate.

Lambda Back-end

Il back-end di un bot Lex, ovvero la parte di computazione che permette di soddisfare gli intenti e che contiene quindi tutta la business logic, deve essere realizzato con **AWS Lambda.** Attualmente, al contrario di amazon Alexa, non sono previste modalità di integrazione generiche e cross provider.

AWS garantisce la sicurezza delle invocazioni mediante **IAM policy e trust relationship.**

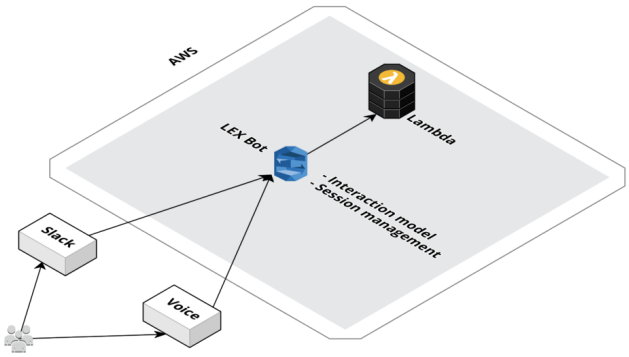
Interfacce/Canali di comunicazioni

I canali sono le vie di comunicazione con il bot. Al momento vengono supportati Facebook Messenger, Slack e Twilio SMS.

È possibile definire uno o più canali direttamente dalle impostazioni del bot; per tutti i canali gestiti non è necessario implementare funzioni, né effettuare il provisioning di risorse. È sufficiente configurare l'integrazione con i servizi esterni.

Sono altresì disponibili guide ufficiali dettagliate per integrare tutti i servizi supportati con Lex.

Infrastruttura



L'infrastruttura è molto semplice e si compone di **soli servizi gestiti.**

Il trigger tra Lex e Lambda è completamente gestito da AWS; per configurarlo basta indicare l'arn della funzione nel modello del bot, ed abbinarlo all'intento desiderato.

Anche i canali supportati non richiedono il provisioning di risorse all'interno dell'account AWS. La gestione di eventuali endpoint o canali di comunicazioni con servizi esterni sono gestiti automaticamente da Lex.

Interazione con il bot

Una volta definito il modello e configurato un canale, il bot è pronto a ricevere input dall'utente. Ogni input verrà scandagliato in cerca dell'intento; Lex guiderà poi la conversazione per reperire tutti gli **slot**, o **parametri**, definiti nel modello per poter elaborare l'intento individuato.

Ottenuti tutti i **parametri** necessari, Lex invocherà la funzione lambda indicata passandogli un oggetto in cui sono definiti **l'intento** ed i **parametri** raccolti.

Il risultato della lambda sarà il messaggio di risposta fornito all'utente.

Un bot triviale

Abbiamo definito tutti i concetti chiave di Lex e possiamo cominciare ad entrare nel merito della realizzazione di un bot.

Quello che segue è un esempio di bot funzionante realizzato utilizzando esclusivamente Amazon Lex.

Si tratta di un bot triviale, al solo fine di dimostrare i concetti ed il principio di funzionamento.

Per questo primo bot non saranno utilizzate funzioni di back-end in quanto ci limiteremo a riconoscere l'intento di salutare o di essere salutati e rispondere con un messaggio preimpostato.

Modello di interazione

Questo è l'aspetto di un **modello di interazione.** Come già accennato si tratta di un file json che contiene l'elenco di tutti gli intenti, nel nostro esempio "greetings".

Per ogni intento vengono definite le frasi di esempio, definite nell'array "sampleUtterances" e anche il tipo di azione da intraprendere per soddisfare l'intento, definito nel campo "fulfillmentActivity" e valorizzato a ReturnIntent" nel nostro esempio, per rispondere al client con un messaggio fisso.

Il messaggio preimpostato viene scelto casualmente tra quelli indicati in "conclusionStatement"."messages".

```
{
 "metadata": {
   "schemaVersion": "1.0",
   "importType": "LEX",
   "importFormat": "JSON"
 },
 "resource": {
   "name": "TestBot",
   "version": "1",
   "intents": [
     {
       "name": "greetings",
       "version": "2",
       "fulfillmentActivity": {
         "type": "ReturnIntent"
       },
       "sampleUtterances": [
         "greet me",
         "please greet me",
         "greetings",
         "make the greetings for me",
         "would you greet me please",
         "would you greet me"
       ],
       "slots": [],
       "conclusionStatement": {
         "messages": [
           {
             "groupNumber": 1,
             "contentType": "PlainText",
             "content": "Hello"
           },
           {
             "groupNumber": 1,
             "contentType": "PlainText",
             "content": "Greetings"
           },
           {
             "groupNumber": 1,
             "contentType": "PlainText",
             "content": "Hi :-)"
           }
         ]
       }
     }
   ],
```

```
    "voiceId": "Salli",
    "childDirected": false,
    "locale": "en-US",
    "idleSessionTTLInSeconds": 300,
    "clarificationPrompt": {
      "messages": [
        {
          "contentType": "PlainText",
          "content": "Sorry, can you please repeat that?"
        }
      ],
      "maxAttempts": 5
    },
    "abortStatement": {
      "messages": [
        {
          "contentType": "PlainText",
          "content": "Sorry, I could not understand. Goodbye."
        }
      ]
    }
  }
}
```

A questo punto, basta configurare un canale, ad esempio Slack, per poi pubblicare e testare il bot.

Per maggiori informazioni su come configurare un canale qui è disponibile una guida passo a passo.

Che ci crediate o no, questo modello è del tutto sufficiente a produrre un bot funzionante 🙂

Un bot con backend

Ora che abbiamo messo in produzione il bot con le funzionalità minime, possiamo passare ad un esempio più completo.

In questo test andremo ad estendere il bot precedentemente illustrato aggiungendo un intento per il calcolo del fattoriale di un numero.

Per poter soddisfare la richiesta, sarà utilizzato un back-end basato su AWS Lambda.

Modello di interazione

Il modello è identico a quello precedente, ma con l'aggiunta di un nuovo intento "factorial".

Del nuovo intento sono anche definiti gli slot nell'apposito array e il tipo di "fulfillmentActivity" è un oggetto "CodeHook" contenente a sua volta il parametro "uri", che è l'arn della funzione lambda da invocare.

Per utilizzare i parametri all'interno delle frasi di esempio, i nomi degli slot vanno indicati tra parentesi graffe {slot}.

Ecco il modello completo:

```
{
 "metadata": {
   "schemaVersion": "1.0",
   "importType": "LEX",
   "importFormat": "JSON"
 },
 "resource": {
   "name": "TestBot",
   "version": "2",
   "intents": [
     {
       "name": "factorial",
       "version": "4",
       "fulfillmentActivity": {
         "codeHook": {
           "uri": "arn:aws:lambda:eu-west-1:XXXXXXXXXXXX:function:besharp-testbot",
           "messageVersion": "1.0"
         },
         "type": "CodeHook"
       },
       "sampleUtterances": [
         "{n} factorial",
         "compute {n} factorial",
         "please calc {n} factorial",
         "factorial of {n}",
         "what is the factorial of {n}",
         "what is {n}"
       ],
       "slots": [
         {
           "sampleUtterances": [],
           "slotType": "AMAZON.NUMBER",
           "slotConstraint": "Required",
           "valueElicitationPrompt": {
             "messages": [
               {
                 "contentType": "PlainText",
                 "content": "What is the input number?"
               }
             ],
             "maxAttempts": 2
           },
           "priority": 1,
           "name": "n"
         }
       ]
     },
     {
       "name": "greetings",
       "version": "3",
       "fulfillmentActivity": {
        "type": "ReturnIntent"
       },
       "sampleUtterances": [
         "greet me",
         "please greet me",
         "greetings",
         "make the greetings for me",
         "would you greet me please",
```

```
        "would you greet me",
        "hello",
        "hi"
      ],
      "slots": [],
      "followUpPrompt": {
        "prompt": {
          "messages": [
            {
              "groupNumber": 1,
              "contentType": "PlainText",
              "content": "Hello"
            },
            {
              "groupNumber": 1,
              "contentType": "PlainText",
              "content": "Greetings"
            },
            {
              "groupNumber": 1,
              "contentType": "PlainText",
              "content": "Hi :-)"
            }
          ],
          "maxAttempts": 3
        },
        "rejectionStatement": {
          "messages": [
            {
              "groupNumber": 1,
              "contentType": "PlainText",
              "content": "Yes, yes, whatever"
            }
          ]
        }
      }
    }
  ],
  "voiceId": "Salli",
  "childDirected": false,
  "locale": "en-US",
  "idleSessionTTLInSeconds": 300,
  "clarificationPrompt": {
    "messages": [
      {
        "contentType": "PlainText",
        "content": "Sorry, can you please repeat that?"
      }
    ],
    "maxAttempts": 5
  },
  "abortStatement": {
    "messages": [
      {
        "contentType": "PlainText",
        "content": "Sorry, I could not understand. Goodbye."
      }
    ]
  }
```

```
 }
}
```

Funzione Lambda

Per poter rispondere con il fattoriale occorre ora programmare il back-end del bot.

Quella che segue è una semplice funzione lambda in Python 2.7.

Abbiamo reso grigio tutto il boilerplate e le funzioni helper, il fulcro del backend sono i metodi: **lambda_handler**, che fa da entry point, la **funzione dispatch** che si occupa di invocare la funzione corretta in base al nome dell'intento individuato da Lex, e **le funzioni factorial e fatt** che si occupano di calcolare il fattoriale e restituirlo nel formato corretto per Lex.

```
import json
import datetime
import time
import os
import dateutil.parser
import logging

logger = logging.getLogger()
logger.setLevel(logging.DEBUG)


# --- Helpers that build all of the responses ---

def elicit_slot(session_attributes, intent_name, slots, slot_to_elicit, message):
    return {
        'sessionAttributes': session_attributes,
        'dialogAction': {
            'type': 'ElicitSlot',
            'intentName': intent_name,
            'slots': slots,
            'slotToElicit': slot_to_elicit,
            'message': message
        }
    }

def confirm_intent(session_attributes, intent_name, slots, message):
    return {
        'sessionAttributes': session_attributes,
        'dialogAction': {
            'type': 'ConfirmIntent',
            'intentName': intent_name,
            'slots': slots,
            'message': message
        }
    }

def close(session_attributes, fulfillment_state, message):
    response = {
        'sessionAttributes': session_attributes,
        'dialogAction': {
            'type': 'Close',
```

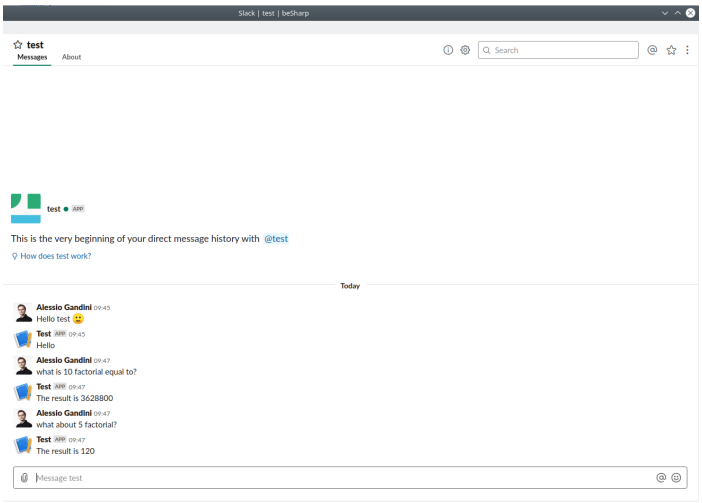
```
            'fulfillmentState': fulfillment_state,
            'message': message
        }
    }

    return response

def delegate(session_attributes, slots):
    return {
        'sessionAttributes': session_attributes,
        'dialogAction': {
            'type': 'Delegate',
            'slots': slots
        }
    }


# --- Helper Functions ---


def safe_int(n):
    """
    Safely convert n value to int.
    """
    if n is not None:
        return int(n)
    return n


def try_ex(func):
    """
    Call passed in function in try block. If KeyError is encountered return None.
    This function is intended to be used to safely access dictionary.

    Note that this function would have negative impact on performance.
    """

    try:
        return func()
    except KeyError:
        return None


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


def factorial(intent_request):

    n = safe_int(try_ex(lambda: intent_request['currentIntent']['slots']['n']))

    return close(
        {},
        'Fulfilled',
        {
            'contentType': 'PlainText',
            'content': 'The result is ' + str(fatt(n))
        }
    )

def fatt(n):
```

```
    r = 1
    for i in range(1, n + 1):
        r = r * i
    return r


# --- Intents ---


def dispatch(intent_request):
    """

    Called when the user specifies an intent for this bot.

    """


    logger.debug('dispatch userId={},
    intentName={}'.format(intent_request['userId'],
    intent_request['currentIntent']['name']))

    intent_name = intent_request['currentIntent']['name']


    # Dispatch to your bot's intent handlers


    if intent_name == 'factorial':
        return factorial(intent_request)

    raise Exception('Intent with name ' + intent_name + ' not supported')


# --- Main handler ---


def lambda_handler(event, context):
    """
    Route the incoming request based on intent.
    The JSON body of the request is provided in the event slot.
    """
    # By default, treat the user request as coming from the America/New_York time zone.
    os.environ['TZ'] = 'America/New_York'
    time.tzset()


    logger.debug('event.bot.name={}'.format(event['bot']['name']))


    return dispatch(event)
```

Anche in questo caso basta pubblicare la lambda e poi aggiornare il modello del bot precedente per poter subito utilizzare le nuove capacità del bot.

Come potete notare, una volta appresi i concetti chiave, la realizzazione di un bot è piuttosto semplice ed assistita dai servizi di Amazon ed AWS.

Un esempio di conversazione:



In questo articolo abbiamo trattato i concetti fondamentali che stanno alla base del funzionamento di Lex. Abbiamo poi analizzato l'anatomia di un bot semplice e funzionante, il primo passo per la realizzazione di bot più complessi.

Restate sintonizzati per altri articoli su Lex e per tutte le ultime novità!

beSharp

Dal 2011 beSharp guida le aziende italiane sul Cloud. Dalla piccola impresa alla grande multinazionale, dal manifatturiero al terziario avanzato, aiutiamo le realtà più all'avanguardia a realizzare progetti innovativi in campo IT.

Get in touch

beSharp.it
proud2becloud@besharp.it