

## AWS MAJOR PROJECT

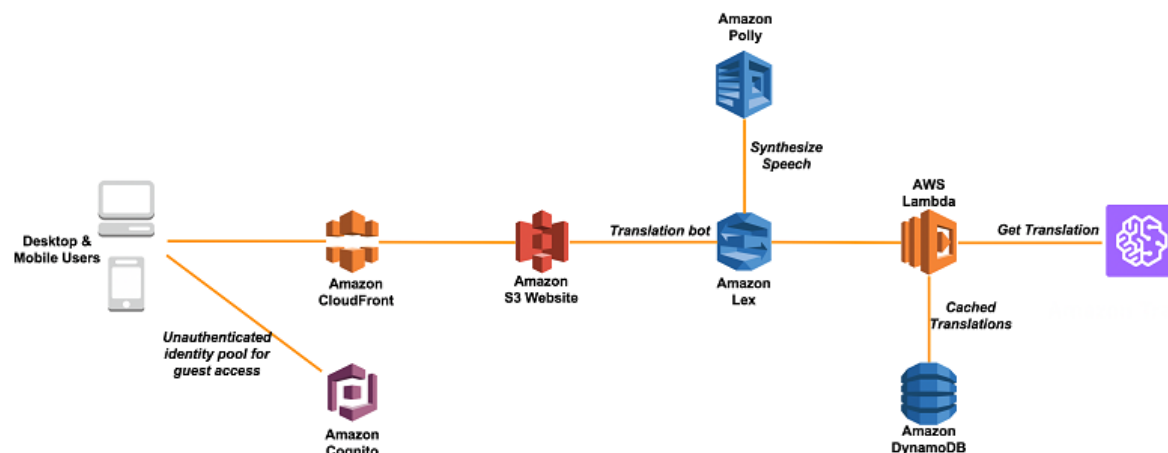
Name: katamoni koundinya goud

Google classroom: AWS-04-SPB1

### Abstract

Creating an text translate bot using amazon translate and amazon lex

The working model of the translate bot



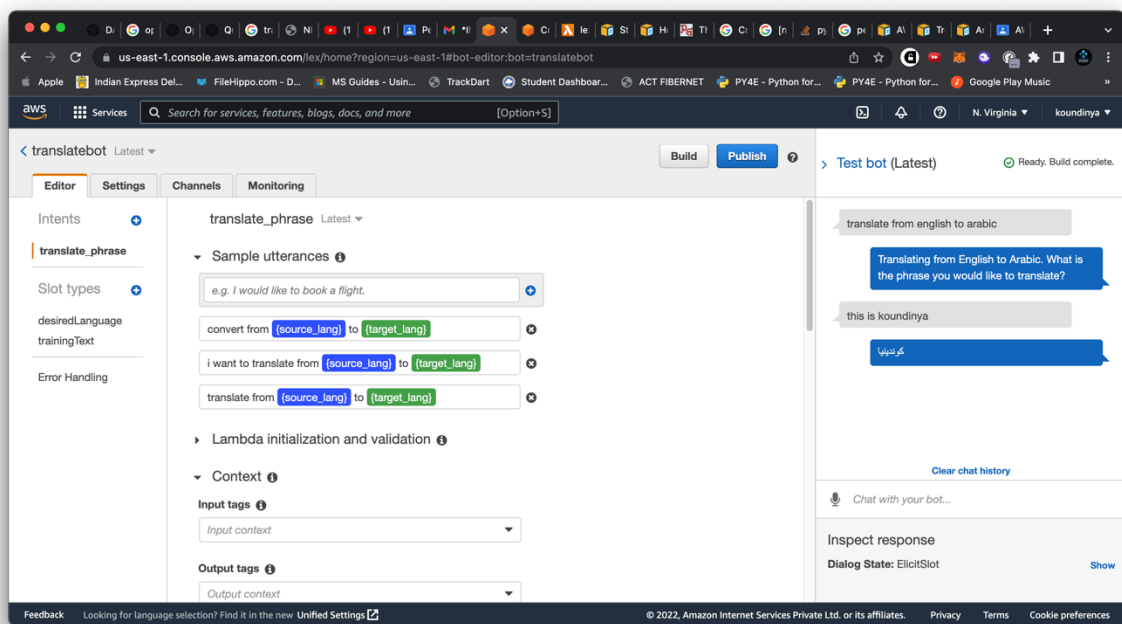
The user can request to translate the language in the chatbot and this entire architecture is based on serverless computing

### Downloading and processing dataset

I used a book called sherlock.txt for my dataset processing and to clean the data set I used the following code to phrase the text and save it using the zip format  
I also used the neural language toolkit for cleaning the dataset and phrasing the text which tokenises the sentences

```
home -- bash -- 161x39
Bharaths-MacBook-Pro:~ bharathkatamoni$ cd /Library/Frameworks/Python.framework/Versions/3.9/bin/python3
-bash: cd: /Library/Frameworks/Python.framework/Versions/3.9/bin/python3: Not a directory
Bharaths-MacBook-Pro:~ bharathkatamoni$ -m pip install --upgrade pip
-bash: -m: command not found
Bharaths-MacBook-Pro:~ bharathkatamoni$ pip install --upgrade pip
Requirement already satisfied: pip in /Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages (22.0.3)
Collecting pip
  Downloading pip-22.1.2-py3-none-any.whl (2.1 MB)
    2.1/2.1 MB 1.6 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 22.0.3
    Uninstalling pip-22.0.3:
      Successfully uninstalled pip-22.0.3
  Successfully installed pip-22.1.2
Bharaths-MacBook-Pro:~ bharathkatamoni$ cd ..
Bharaths-MacBook-Pro:~/Users/bharathkatamoni$ ls
Shared
Bharaths-MacBook-Pro:~/Users/bharathkatamoni$ cd ..
Bharaths-MacBook-Pro:~/Users/bharathkatamoni$ ls
Applications  System  Volumes  cores  etc  home  opt  private  sbin  tmp  usr  var
Library
Bharaths-MacBook-Pro:~/Users/bharathkatamoni$ cd home
Bharaths-MacBook-Pro:~/home bharathkatamoni$ ls
Library
Bharaths-MacBook-Pro:~/home bharathkatamoni$ ls
Library
Bharaths-MacBook-Pro:~/home bharathkatamoni$ sudo pip install -U nltk
Password:
WARNING: The directory '/Users/bharathkatamoni/Library/Caches/pip' or its parent directory is not owned or is not writable by the current user. The cache has been disabled. Check the permissions and owner of that directory. If executing pip with sudo, you should use sudo's -H flag.
Requirement already satisfied: nltk in /Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages (3.7)
Requirement already satisfied: joblib in /Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages (from nltk) (1.1.0)
Requirement already satisfied: regex<=2021.8.3 in /Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages (from nltk) (2022.6.2)
Requirement already satisfied: tqdm in /Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages (from nltk) (4.64.0)
Requirement already satisfied: click in /Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages (from nltk) (8.1.3)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a venv
  WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a venv instead: https://pip.pypa.io/warnings/venv
Bharaths-MacBook-Pro:~/home bharathkatamoni$ ls
Bharaths-MacBook-Pro:~/home bharathkatamoni$
```

### Creating an amazon lex bot



Creating a lambda function to grab the api from the amazon translate and serve the bot with the specified permission's I managed to create a function and choose python 3.9 for the writing code which as follows

```
# -*- coding: utf-8 -*-
"""
```

This function interacts with Amazon Lex and the Amazon Translate AWS services to receive input speech and then translate it to the requested language. Current supported translations as of July 2018 are:

English to

- Arabic (ar)
- Chinese (Simplified) (zh)
- Chinese (Traditional) (zh-TW)
- Czech (cs)
- French (fr)
- German (de)
- Italian (it)
- Japanese (ja)
- Portuguese (pt)
- Russian (ru)
- Spanish (es)
- Turkish (tr)

Please see the Amazon Translate API docs for an up to date list:  
[https://docs.aws.amazon.com/translate/latest/dg/API\\_TranslateText.html](https://docs.aws.amazon.com/translate/latest/dg/API_TranslateText.html)

```

from __future__ import print_function
import boto3
import logging

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

# boto3 translate client
translate = boto3.client('translate')

# Use a dictionary to map supported languages with their ISO639-1 codes
lang_map = {
    'arabic': 'ar',
    'simplified chinese': 'zh',
    'traditional chinese': 'zh-TW',
    'czech': 'cs',
    'english': 'en',
    'french': 'fr',
    'german': 'de',
    'italian': 'it',
    'japanese': 'ja',
    'portuguese': 'pt',
    'russian': 'ru',
    'spanish': 'es',
    'turkish': 'tr'
}

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

def get_slots(intent_request):
    return intent_request['currentIntent']['slots']

def elicit_slot(session_attributes, intent_name, slots, slot_to_elicit, message):
    logger.debug('Elicit Slot function intentName: {}'.format(intent_name))
    logger.debug('Elicit Slot function slots: {}'.format(slots))
    logger.debug('Elicit Slot function slotToElicit: {}'.format(slot_to_elicit))
    logger.debug('Elicit Slot function message: {}'.format(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
        }
    }
}

```

# ----- Functions that control the skill's behavior -----

```

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

# return JSON-formatted descriptive response if validation fails
def build_validation_result(isvalid, violated_slot, message_content):
    return {
        'isValid': isvalid,
        'violatedSlot': violated_slot,
        'message': {'contentType': 'PlainText', 'content': message_content}
    }

# validate source and target languages
def validate_languages(source_lang, target_lang):
    if source_lang is not None and source_lang not in lang_map:
        return build_validation_result(False,
                                       'source_lang',
                                       'We do not currently support {} as a source
language.'.format(source_lang))

    if target_lang is not None and target_lang not in lang_map:
        return build_validation_result(False,
                                       'target_lang',
                                       'We do not currently support {} as a target
language.'.format(source_lang))

    return build_validation_result(True, None, None)

# Check phrase is not empty
def validate_phrase(phrase, source_lang, target_lang):
    if phrase is not None:
        logger.debug('***** phrase is not none *****')
        return build_validation_result(True, None, None)
    else:
        logger.debug('***** phrase is none *****')
        return build_validation_result(False,
                                       'phrase',
                                       'Translating from {} to {}. What is the phrase you would like to
translate?'.format(source_lang.capitalize(), target_lang.capitalize()))

# ----- Main Translator function -----

def translatePhrase(intent_request):
    """

```

Performs dialog management and fulfillment for translating a phrase.

Uses Amazon Translate to translate a phrase from a source -> destination phrase and provide to customer

```
"""
```

```
source_lang = get_slots(intent_request)['source_lang'].lower()
target_lang = get_slots(intent_request)['target_lang'].lower()
phrase = get_slots(intent_request)['phrase']
source = intent_request['invocationSource']
```

```
logger.debug('Intent Request={}'.format(intent_request))
logger.debug('source_lang={}'.format(source_lang))
logger.debug('target_lang={}'.format(target_lang))
```

```
logger.debug('Phrase={}'.format(phrase))
logger.debug('inputTranscript={}'.format(intent_request['inputTranscript']))
```

```
# Set session attributes for the value of source and target languages
session_attributes = intent_request['sessionAttributes'] if
intent_request['sessionAttributes'] is not None else {}
```

```
if source == 'DialogCodeHook':
    # Validate any slots which have been specified. If any are invalid, re-elicite for their value
    slots = get_slots(intent_request)
```

```
    validation_result = validate_languages(source_lang, target_lang) #check languages are
    valid and supported
```

```
    if not validation_result['isValid']:
        slots[validation_result['violatedSlot']] = None
        return elicit_slot(intent_request['sessionAttributes'],
                           intent_request['currentIntent']['name'],
                           slots,
                           validation_result['violatedSlot'],
                           validation_result['message'])
```

```
    logger.debug('Validated Languages')
```

```
# Convert languages to ISO639-1 codes for Amazon Translate
sourceISO = lang_map[source_lang]
targetISO = lang_map[target_lang]
```

```
logger.debug('sourceISO={}'.format(sourceISO))
logger.debug('targetISO={}'.format(targetISO))
#Set session attributes for the value of source and target ISO codes
session_attributes['source_lang'] = sourceISO
```

```

session_attributes['target_lang'] = targetISO

phrase = get_slots(intent_request)['phrase']
logger.debug('Phrase in dialogCodeHook code={}'.format(phrase))
valid_phrase = validate_phrase(phrase, source_lang, target_lang)
if not valid_phrase['isValid']:
    slots[valid_phrase['violatedSlot']] = None
    return elicit_slot(intent_request['sessionAttributes'],
                      intent_request['currentIntent']['name'],
                      slots,
                      valid_phrase['violatedSlot'],
                      valid_phrase['message'])
logger.debug('Validated Phrase')

# Phrase is valid. Convert phrase to translated text then reset slot to none for next
translation
if phrase is not None:
    translation = try_ex(lambda: translate.translate_text(Text=phrase,
SourceLanguageCode=sourceISO, TargetLanguageCode=targetISO))
    translatedText = str(translation.get('TranslatedText'))
    logger.debug('Translated Text is={}'.format(translatedText))
    print ("Translation is: ", translatedText)

    intent_request['currentIntent']['phrase'] = None
    logger.debug('Set phrase slot to None and returning elicit slot with translation')
    logger.debug('Phrase={}'.format(intent_request['currentIntent']['phrase']))

    return elicit_slot(intent_request['sessionAttributes'],
                      intent_request['currentIntent']['name'],
                      slots,
                      'phrase',
                      {
                        'contentType': 'PlainText',
                        'content': format(translatedText)
                      })
    logger.debug('Translated Phrase')
    return delegate(session_attributes, intent_request['currentIntent']['slots'])

# Convert phrase to translated text
translation = try_ex(lambda: translate.translate_text(Text=phrase,
SourceLanguageCode=sourceISO, TargetLanguageCode=targetISO))
translatedText = str(translation.get('TranslatedText'))

logger.debug('Translated Text is={}'.format(translatedText))
print ("Translation is: ", translatedText)

```

```

return close(
    session_attributes,
    'Fulfilled',
    {
        'contentType': 'PlainText',
        'content': 'Thank you for using the translator bot'
    }
)

```

# --- Lex intent capture ---

```

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 == 'translate_phrase':
        return translatePhrase(intent_request)
    raise Exception('Intent with name ' + intent_name + ' not supported')

```

# --- Main lambda handler ---

```

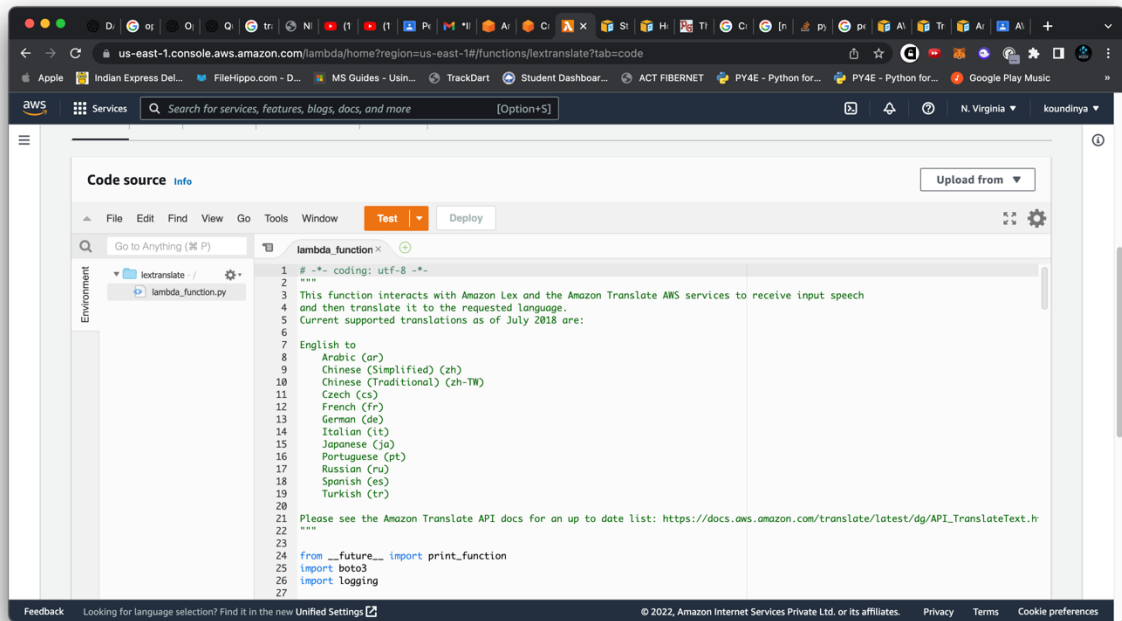
def lambda_handler(event, context):
    """
    Route the incoming request based on intent.
    The JSON body of the request is provided in the event slot.
    """

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

    return dispatch(event)

```





## Test bot in action

### > Test bot (Latest)

✓ Ready. Build complete.

translate from english to arabic

Translating from English to Arabic. What is the phrase you would like to translate?

this is koundinya

کوندینیا

Clear chat history

Chat with your bot...

Inspect response

Dialog State: ElicitSlot

Show