

Developing ChatBots using Amazon Lex

Written for AllState, Belfast, June 2018

Contents

Chapter 1	Introduction to ChatBots
Chapter 2	ChatBot UX Best Practice
Chapter 3	Introduction to Natural Language Processing
Chapter 4	Cloud Infrastructure Overview
Chapter 5	Security in the Cloud
Chapter 6	Introduction to Amazon Lex
Chapter 7	Processing ChatBot Requests using AWS Lambda
Chapter 8	ChatBot Deployment Options
Chapter 9	Testing Chatbots using Cucumber
	Lab Exercises

©Copyright 15/06/2018 Conygre IT Limited
All rights reserved

All trademarked product and company names are the property of their respective trademark holders.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, or translated into any language, without the prior written consent of the publisher.

Conygre IT Limited shall not be liable in any way in respect of any loss, damages, costs, liabilities, or expenses suffered by the user, whether directly or indirectly as a result of the content, format, presentation, or any other aspect of the materials.

Introduction to ChatBots

Introduction to ChatBots

[conygre] -
f i l i t h e g a p

Objectives

- What is a chatbot?
- Why do chatbots matter?
- Types of Chatbot
- Technology choices

- [conygre] -
f i l l t h o u g h t

What is a ChatBot?

- According to the dictionary, a chat bot is
 - “*a computer program designed to simulate conversation with human users, especially over the Internet.*”
- So a chatbot is a piece of software that can talk to people via chat over a messaging channel

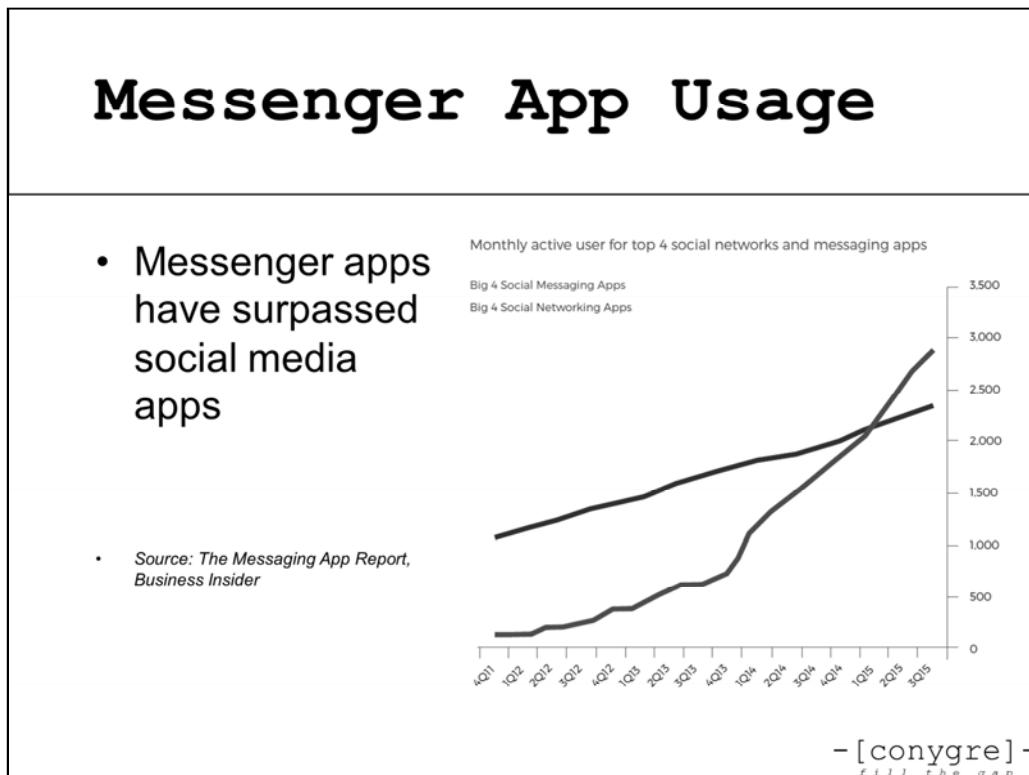
- [conygre] -
f i l l t h o u g h t s

Why do ChatBots Matter?

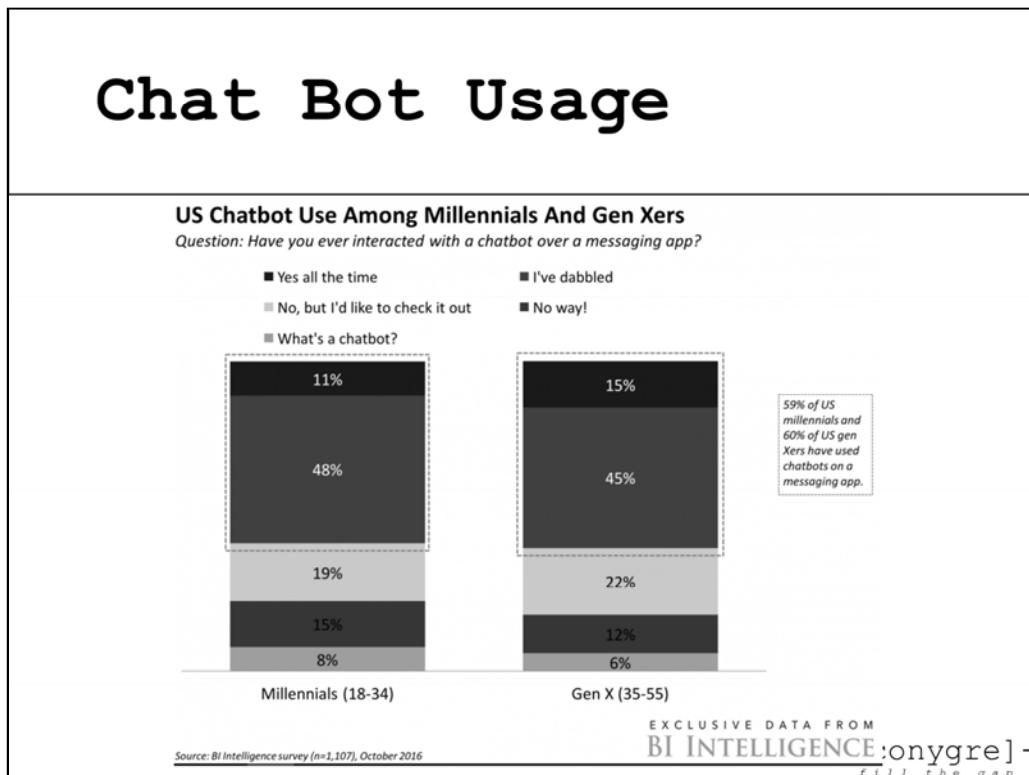
- Messaging apps have become increasingly popular over recent years, supplanting social networks
 - ‘over 2.5 billion people have at least one messaging app installed’ and ‘many teenagers now spend more time on smartphones sending instant messages than perusing social networks. WhatsApp users average nearly 200 minutes each week using the service.’ – The Economist

– [conygre] –
f i l l t h o u g h t s

Introduction to ChatBots



Introduction to ChatBots

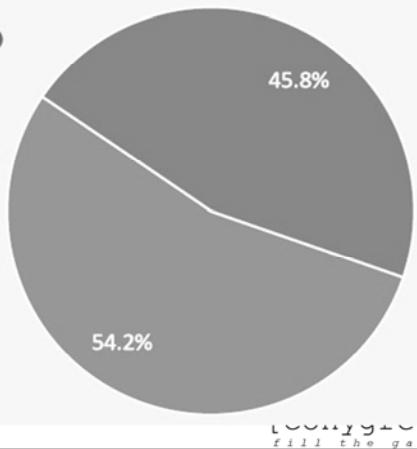


People Like Chat!

Assuming both are free and possible, how would you rather talk to a business?

■ Email

■ Messaging application (FB Messenger, Whatsapp, WeChat, Line etc.)



Chat Matters

- Use of chat is growing all the time and the importance of chatbots is going to be vital if you are going to be available 24/7 over this medium without incurring a massive staff cost

- [conygre] -
f i l l t h o u g h t

Messaging Platforms have APIs for Bots

- Facebook Messenger
- Slack
- Skype
- Twitter
- What'sApp

- [conygre] -
f i l l t h o u g h t s

Types of ChatBot

- **Rule based** designed to allow people to complete a specific set of tasks
- **AI based** designed to dynamically learn from user responses

- [conygre] -
f i l l t h o u g h t

Introduction to ChatBots

AI ChatBots

- Facebook had some issues with their AI driven chatbots

Facebook's artificial intelligence robots shut down after they start ...
[https://www.independent.co.uk/.../facebook-artificial-intelligence-ai-chatbot-new-lang... ▾](https://www.independent.co.uk/.../facebook-artificial-intelligence-ai-chatbot-new-lang...)
31 Jul 2017 - The two chatbots came to create their own changes to English that made it easier for them to work – but which remained mysterious to the ...

Chinese chatbots shut down after anti-government posts - BBC News
[www.bbc.co.uk/news/world-asia-china-40815024 ▾](http://www.bbc.co.uk/news/world-asia-china-40815024)
3 Aug 2017 - The pair of Chinese chatbots were removed after they sent anti-government messages.

Facebook shuts down chatbot experiment after AIs spontaneously ...
[https://www.standard.co.uk/News/Technology ▾](https://www.standard.co.uk/News/Technology)
1 Aug 2017 - Facebook shut down a chatbot experiment after two artificial intelligence creations began communicating in a language which evolved between them. The social media giant was attempting to teach two AIs, dubbed Bob and Alice, the art of negotiation. But, after leaving the pair alone ...

No, Facebook Did Not Panic and Shut Down an AI Program That Was ...
[https://gizmodo.com/no-facebook-did-not-panic-and-shut-down-an-ai-program-1797... ▾](https://gizmodo.com/no-facebook-did-not-panic-and-shut-down-an-ai-program-1797...)
31 Jul 2017 - The intent was to develop a chatbot which could learn from human interaction to negotiate deals with an end user so fluently said user would ...

Facebook shuts down robots after they invent their own language
[https://www.telegraph.co.uk/Technology/Intelligence ▾](https://www.telegraph.co.uk/Technology/Intelligence/)
1 Aug 2017 - Facebook shut down a pair of its artificial intelligence robots after they ... The chatbot conversation "led to divergence from human language as ...

Facebook manages to shut down chatbot just before it could become evil
[https://news.avclub.com/facebook-manages-to-shut-down-chatbot-just-before-it-co-17... ▾](https://news.avclub.com/facebook-manages-to-shut-down-chatbot-just-before-it-co-17...)
31 Jul 2017 - The moment when an artificial intelligence becomes smarter than the humans who created it is usually the turning point in a sci-fi movie, and it ...

F J J E n g a p

Modern AI Chatbots

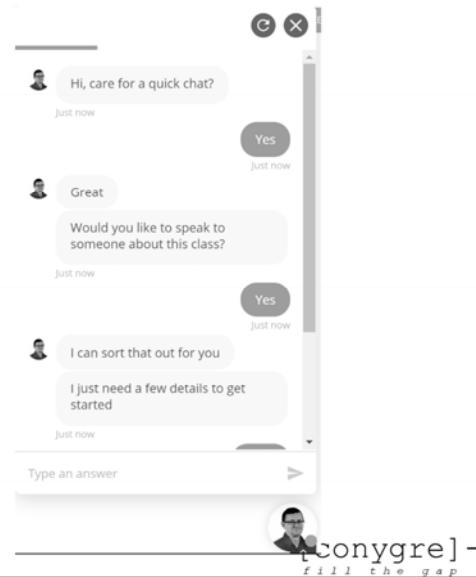
- Modern AI Chatbots are available such as
 - DialogFlow
 - GupShup
 - Amazon Lex



- [conygre] -
f i l l t h o g a p

Rules Based ChatBots

- Rules based chatbots are only as clever as the rules you create
 - Collect.chat
- These are good for simple task oriented bots



Technology Choices

- There is the choice around
 - AI vs. Rules based
- Here are three key decisions you need to make
 1. Rule based or AI
 - Rule based is easier but less flexible
 2. Engine to use
 - There are many choices of engine to drive your chatbot
 3. Platform to deploy to
 - WhatsApp, Twitter, Messenger

- [conygre] -
f i l l t h o u g h t

Exercise

- Sign up to collect.chat to see a basic example of a chat bot
- In pairs, using this simple interface, create a chatbot to handle a problem of your choice
- Once complete, switch with someone else in the class to see how they get on with your chat bot and elicit feedback

- [conygre] -
f i l l t h o u g h t s

Summary

- What is a chatbot?
- Why do chatbots matter?
- Types of Chatbot
- Technology choices

- [conygre] -
f i l l t h o u g h t s

ChatBot UX

[conygre] -
f.i.l.t.h.e.g.a.p

Objectives

- Review a number of best practices identified for the creation of chatbots
 - Source: <https://chatbotsmagazine.com/19-best-practices-for-building-chatbots-3c46274501b2>

- [conygre] -
f i l l t h o u g h t

#1 Expectation Engineering

- Don't give people the impression that this is a person
- Don't give people the impression that it is a chat bot with sophisticated capabilities if it doesn't have them

Hi, my name is Nick and I am sat by my computer in case you have any questions



- [conygre] -
f i l l t h o g a p

#2 Provide a Way Back

- Create a mechanism for users to return to the initial options
 - *How do I get back to the beginning?*

Thanks for entering all that information. Please submit your order. I assume you don't want to change anything.



- [conygre] -
f i l l t h o g a p

#3 Avoid tl;dr

- AVOID Too long, didn't read
- Big scrolling messages are a turn off to most people
 - They just give up

We offer training in all sorts of topics, leadership, programming, cloud computing. You can do the course at home and remote in or come to our office. Our terms are 30 days from the end of the course and all our charges are visible at our terms and conditions page which you are welcome to read



- [conygre] -
f i l l t h e g a p

#4 Test on Different Devices

- Make sure it works as expected on devices of different sizes and resolutions

We offer training in all sorts of subjects such as leadership, programming, computing. You can do the training at your convenience either at home and remote in or coming to our office. Our terms are 30 days long and you will receive a certificate at the end of the course and all course materials. All our training materials are visible at our terms and conditions page which you are welcome to visit.

- [conygre] -
f i l l t h o u g h t s

#5 Consistency

- Make sure you use the same format every time for links or questions requiring specific responses and so on
- Don't confuse people with inconsistent options

- [conygre] -
f i l l t h o u g h t

#6 Redirect with Web Views

- Don't just link out to other web sites, but rather display Web views within the chat itself

Searching for a flight from Bristol, United Kingdom to Rome, Italy, flying out on Wednesday, June 13 and flying back on Friday, June 15. Replying with results shortly.

→ Wed, Jun 13 ← Fri, Jun 15

12:10 BRS Direct 2h 35m	16:20 FCO Direct 2h 45m
15:45 FCO	18:05 BRS

Best flights (\$239; 2h 35m and 2h 45m return)
Flying with easyJet
www.skyscanner.net

Book
Show top 10 best
Set price alerts

- [conygre] -
f i l l t h o u g h t s

#7 Ensure Redirects Capture Information

- When users fill in various options in the chat, make sure those options are propagated through to any Web pages or views that you link to
 - DO NOT make them enter it all again!

- [conygre] -
f i l l t h o u g h t

#8 Provide Guard Rails

- Providing specific options can make it easier for people and keeps them on track

Hi! 😊 Welcome to the Skyscanner bot. We'll help you find your next trip 🎈. What would you like to do?

Find a flight

Best deals

Tell me more

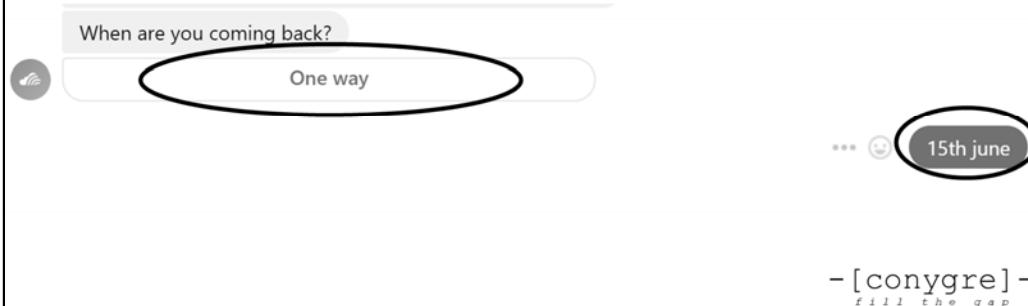


- [conygre] -
f i l l t h e g a p



#9 Mixing Options and Free Text?

- Be careful if providing specific options and ALSO the option to type in a value
- People don't notice that there is an option to type in a value, they just see buttons



#10 Clarity over Elegance

- Always value clarity over elegance
- **Outbound / Inbound**
 - instead of
- **Departure / Return**
- You want to avoid **semantic satiation** where constant use of the same words causes them to lose meaning

- [conygre] -
f i l l t h o u g h t s

#11 Confirm by Asking not Stating

- Ensure that any confirmations are asked for not simply stated
 - “You ordered flowers for Wednesday”
vs
 - “Is it correct you want flowers on Wednesday?”
- Stating can make the user feel like they are being corrected if it is wrong

F I I I - t h o - g a p

#12 Don't Leave User Hanging

- If the ChatBot will take a while to respond, make sure that the user knows
- Long delays will cause people to give up if they don't know what is going on

- [conygre] -
f i l l t h o u g h t

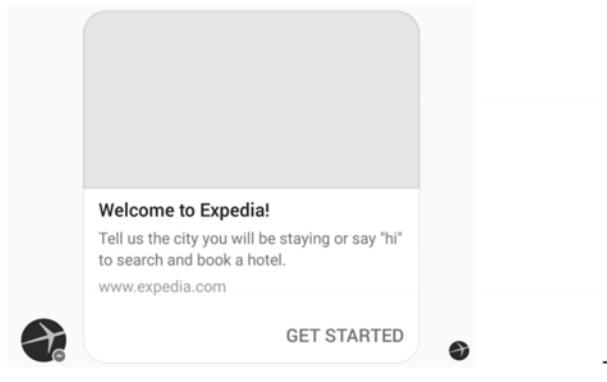
#13 Provide an Undo/Cancel

- Ensure people can undo or cancel what they said
- It is easy to make a mistake in a chat so a simple undo would be useful for users

- [conygre] -
f i l l t h o u g h t s

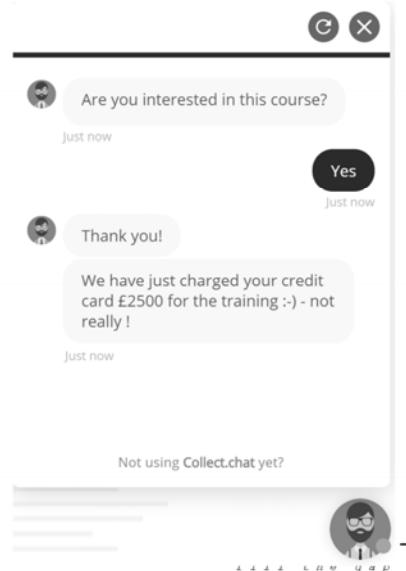
#14 Beware of Captions

- The caption here contains the actual instructions!
 - You could be forgiven for missing it



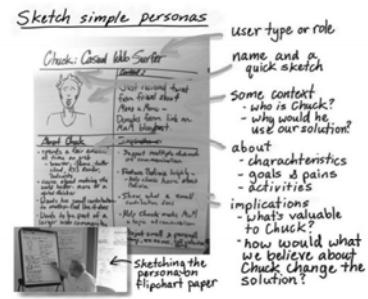
#15 Be Careful with Humour

- Make sure your humour is not too off putting



#16 Consider a Voice

- It can be helpful to create a persona for your chatbot
- It allows you to think of your chatbot as a person and therefore how they might respond to the user
- It can make it more believable and engaging for the audience



- [conygre] -
f i l l t h o u g h t s

#17 Consider the Icon

- Consider the chatbot icon
 - People respond best to faces
 - Don't just use a logo or an icon

- [conygre] -
f i l l t h o u g h t s

Summary

- Bear in mind these best practices when designing chat bots
- Constantly review and iterate over existing bots

- [conygre] -
f i l l t h o u g h t s

Natural Language Processing (NLP)

[conygre] -
f.i.l.t.h.e.g.a.p

Objectives

- What is NLP?
- What can you do with NLP?
- Overview of Natural Language Processing
- Worked Python example of Sentiment analysis
- NLP and Chatbots

- [conygre] -
f i l l t h o u g h t

What is NLP?

- “The field of study that focuses on the interactions between human language and computers is called NLP. It sits at the intersection of computer science, artificial intelligence, and computational linguistics”
– *Wikipedia*

- [conygre] -
f i l l t h o u g h t

What Can You Do With NLP?

- Automate summarization
- Translation
- Entity recognition
- Sentiment analysis
- Topic segmentation
- All of these could be useful in a chatbot!

- [conygre] -
f i l l t h o u g h t s

How Hard Can It Be?

- NLP is tricky
- It is one thing to understand a word, but totally different to infer meaning from the way people might construct a sentence, let alone use of humour, irony, sarcasm, local dialects
- What does this mean?
 - “where’s he to?”

- [conygre] -
f i l l t h o g a p

NLP Fundamentals

- To understand the discipline, we will use an example
 - Processing a set of movie reviews to decide if they are positive reviews or negative reviews
- To complete this we will need to do some
 - Data cleaning and PreProcessing
 - Tokenisation
 - Creating a learning model
 - Learning from the learning model
 - Assessing our reviews

- [conygre] -
f i l l t h o u g h t s

Our DataSet

- Our dataset is a set of reviews which appear as follows in a TSV file consisting of a review id and a review

id	review
12311_10	Naturally, a film who's main ...
8348_2	This movie is a disaster within a disaster film ...

- Are these reviews positive or negative?

- [conygre] -
f i l l t h o u g h t s

Our Learning Model

- In order to complete sentiment analysis, we will need some data for our **machine to learn** from

id	sentiment	review
5814_8	1	With all this stuff going down at the ...
2381_9	0	The Classic War of the Worlds by ...

- These reviews have already been marked as either
 - positive 1
 - negative 0

- [conygre] -
f i l l t h o g a p

Loading the Data

- We can load the data sets using the Python library **pandas**

```
train = pd.read_csv(os.path.join(os.path.dirname(__file__), 'data', 'labeledTrainData.tsv'),  
header=0,\n    delimiter="\t", quoting=3)  
test = pd.read_csv(os.path.join(os.path.dirname(__file__), 'data', 'testData.tsv'), header=0,  
delimiter="\t",\n    quoting=3 )
```

- [conygre] -
f i l l t h e g a p

Data Cleansing

- Before you can process text you might have to tidy it up
 - Fix any spelling errors
 - Remove HTML markup (if processing web content)
- When using Python
 - **BeautifulSoup** for HTML removal
 - SpellChecker
 - <https://github.com/pirate/spellchecker> based on work by Peter Norvig

- [conygre] -
f i l l t h o u g h t s

Splitting into Words

- Once we have cleaned up the text, we can then split the text up into units
- A simple splitting is to
 - Separate by the space characters into words
 - Remove any non word characters
 - Change all text to lower case
- The act of splitting is called **tokenisation**

- [conygre] -
f i l l t h o g a p

Tokenization in Python

- To complete the tasks in Python

```
# remove the HTML  
review_text = BeautifulSoup(review, "html.parser").get_text()  
# remove the non-letters  
review_text = re.sub("[^a-zA-Z]", " ", review_text)  
#convert words to lower case and split  
words = review_text.lower().split()
```

- [conygre] -
f i l l t h o u g h t

Removing Stop Words

- Stop words carry no meaning of their own and are used to link other words together
- These words are often called **stop words**
- Stop words would be things like
 - Articles (the, a, an)
 - Pronouns (I, me, you)
 - Prepositions (in, on to)

```
from nltk.corpus import stopwords  
...  
stops = set(stopwords.words("english"))  
words = [w for w in words if not w in stops]
```

- [conygre] -
f i l l t h o u g h t s

Complete Cleaning Example

- Below is a Python function to return fully cleansed list of words

```
def review_to_wordlist( review, remove_stopwords=False ):
    review_text = BeautifulSoup(review, "html.parser").get_text()
    words = review_text.lower().split()
    if remove_stopwords:
        stops = set(stopwords.words("english"))
        words = [w for w in words if not w in stops]
    return(words)
```

- [conygre] -
f i l l t h o u g h t

Create a Bag of Words

- So far we have one long (cleansed) list of all the words in the review
- Now we need to find out how many times these words are appearing in the reviews
- This list of words along with the frequencies is called a **bag of words**

- [conygre] -
f i l l t h o u g h t

Simple Bag of Words Example

- Consider the following short reviews
 - the movie was really bad
 - the movie was quite bad in places
- The bag of words for the first one would be
 - {the, movie, was, really, bad, quite, in, places}
 - {1,1,1,1,1,0,0,0}
- The bag of words for the second one would be
 - {the, movie, was, really, bad, quite, in, places}
 - {1,1,1,0,1,1,1,1}

- [conygre] -
f i l l t h o g a p

Using scikit-learn

- The scikit-learn library can create a bag of words for each review
- There are loads of words in the reviews so we will limit it to the top 5000
 - Note the **CountVectorizer** could have been used to tokenize for us

```
vectorizer = CountVectorizer(analyzer = "word", \
                            tokenizer = None, \
                            preprocessor = None, \
                            stop_words = None, \
                            max_features = 5000)
# create the bag of words here
train_data_features = vectorizer.fit_transform(clean_train_reviews)
# convert it into a numpy array for convenience
np.asarray(train_data_features)
```

- [conygre] -
f i l l t h e g a p

Now Machine Learn!

- Now we have a model for the sample data with the number of times each word occurs and whether it was a positive or negative review
- We can now use this to work out whether our new reviews are positive or negative
- One relatively straightforward solution is the **Random Forest Algorithm**

- [conygre] -
f i l l t h o u g h t s

The Random Forest Algorithm

- The random forest will look at all the sets of words in the data we need to look at and then apply this algorithm to see if they are positive or negative
- For information on the algorithm, see
 - <https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d>

- [conygre] -
f i l l t h e g a p

Applying the Algorithm

- To apply the algorithm using Python, you can use the **sklearn** package

```
forest = RandomForestClassifier(n_estimators = 100) # 100 trees
# Fit the forest to the training set, using the bag of words as
# features and the sentiment labels as the response variable
forest = forest.fit( train_data_features, train["sentiment"] )
# Create an empty list and append the clean reviews one by one
clean_test_reviews = []
print ("Cleaning and parsing the test set movie reviews...\n")
for i in range(0,len(test["review"])):
    clean_test_reviews.append(" ".join(Word2VecUtility.review_to_wordlist(test["review"][i], True)))
# Get a bag of words for the test set, and convert to a numpy array
test_data_features = vectorizer.transform(clean_test_reviews)
np.asarray(test_data_features)
# Use the random forest to make sentiment label predictions
result = forest.predict(test_data_features)
```

- [conygre] -
f i l l t h e g a p

Create the Output

- Finally, using the panda library, an output file can be created with the predicted sentiment

```
output = pd.DataFrame( data={"id":test["id"], "sentiment":result} )
output.to_csv(os.path.join(os.path.dirname(__file__), 'data', 'Bag_of_Words_model.csv'),
index=False, quoting=3)
```

- [conygre] -
f i l l t h e g a p

NLP and ChatBots

- The concepts introduced here can be used when
 - Creating chatbots that need to understand user input
 - Is the user making a negative or positive comment on your product?
 - Analysing chatbot logs for positive and negative experiences in order to improve the chatbot

- [conygre] -
f i l l t h o u g h t s

Summary

- What is NLP?
- What can you do with NLP?
- Overview of Natural Language Processing
- Worked Python example of Sentiment analysis
- NLP and Chatbots

- [conygre] -
f i l l t h o u g h t s

Exercise

- Modify the sample application to work on some chatbot logs from chat responses that need to be followed up
 - flagged
 - not_flagged
- Flagging is based on seriousness of response to question:
 - “Describe a time when you have acted as a resource for someone else”
- You can split the sample data into two and use one for training and one for analysis
 - Since you have the actual correct response for each, you can see how good your algorithm is!

- [conygre] -
f i l l t h o u g h t s

Platform Overview

Platform Overview

- [conygre] -
f i l l t h e g a p

Platform Overview

Objectives

- Regions
- Zones
- Servers
- Storage
- Containers
- Serverless Architectures

- [conygre] -
fill the gap

Platform Overview

Regions

- AWS and others divide the globe up into regions
 - Each region is an implementation of the cloud platform
 - AWS regions are only connected by the Internet, whilst other cloud providers such as Google have their own infrastructure connecting regions together

- [conygre] -
fill the gap

Region Selection

- For very high availability you would deploy to multiple regions
- Region selection is based on things like
 - Latency
 - Availability of cloud features
 - Not every region will have every feature
 - Regulatory and Legal reasons
 - Copyright or data protection issues for example

- [conygre] -
fill the gap

Availability Zones

- Each region is then further divided into availability zones
- Each zone is a data centre or cluster of data centres
- Zones have very low latency high speed connections between each other
- Zones are geographically placed to minimize region wide outages

- [conygre] -
fill the gap

Running Applications

- Cloud providers allow for a number of ways to run your applications
 - Run traditional servers in the cloud
 - Run containers in the cloud
 - Run totally serverless architectures using features such as AWS Lambda

- [conygre] -
fill the gap

Platform Overview

Running Servers – EC2

- **Elastic Compute Cloud (EC2)** servers can be launched
- Servers run in availability zones and are launched from **Amazon Machines Images (AMI)**
- AMIs can be
 - Standard AWS provided Linux and Windows variations
 - Third Party Provided – lots to choose from!
 - Created by you

- [conygre] -
fill the gap

Platform Overview

Server Types

- When servers are launched they can be of varying sizes from
 - 0.5gb RAM with a single CPU
 - Hundreds of CPUs with almost 2000gb RAM
 - Anything in between
- Servers can be resized if required
 - Stop / Resize / Start
- Options are optimised for different tasks
 - Compute, Memory, Disk, Graphical etc.

- [conygre] -
fill the gap

Hard Drives

- EC2 servers use **Elastic Block Store** drives (**EBS**)
- EBS volumes can be
 - Magnetic
 - Solid State
 - Provisioned with a guaranteed read / write speed
- EC2 instances can have multiple EBS drives
- Drives can be moved between instances
 - Just like physical drives

- [conygre] -
fill the gap

Data Storage

- There are many storage options
 - EBS – hard drives
 - Instance store – temporary SSD storage physically attached to EC2 instances
 - Elasticache – Memcached / Redis as a service
 - DynamoDB – NoSQL database as a service
 - RDS – relational database as a service
 - S3 – unlimited object storage
 - Glacier – archive service

- [conygre] -
fill the gap

Containers

- **Containers run as a group of namespaced processes within a single Linux operating system**
- Containers have a standard packaging format
 - Providing portability
- Containers have shorter startup times
- Containers have better resource utilization of servers
- Containers are ideal for Continuous Integration/Delivery

- [conygre] -
fill the gap

Platform Overview

Container Services

- With the increasingly popular adoption of microservice architectures, containers are very popular with two of the main platforms
 - Docker
 - Kubernetes
- Both Docker and Kubernetes can be used on AWS for example, docker can be used with **EC2 Container Services (ECS)**

- [conygre] -
fill the gap

Serverless Architectures

- Removing the need for servers altogether is **AWS Lambda**
- AWS Lambda allows you to deploy functions to the cloud
- Functions can be triggered by things like
 - REST API calls
 - Messaging
 - File uploads
 - Scheduled events
 - Alexa voice commands

- [conygre] -
fill the gap

Platform Overview

AWS Lambda Benefits

- You only pay in 100ms chunks for the time the function is actually running
- There is no infrastructure to support
- You can specify the amount of memory the lambda runs with
- Lambdas can be written in
 - Java
 - JavaScript
 - Python
 - C#

- [conygre] -
fill the gap

Platform Overview

Summary

- Regions
- Zones
- Servers
- Storage
- Containers
- Serverless Architectures

- [conygre] -
fill the gap

Security

Cloud Security

- [conygre] -
f i l l t h e g a p

Objectives

- Security concerns
- Securing your Account
- Security your infrastructure
- Security your data
- Encryption
- Auditing tools

- [conygre] -
fill the gap

Security Concerns

- Of all the reasons people give for not moving to the cloud, probably the most common one given is security
 - Sometimes this can be more out of ignorance than fact
- However, clearly security is of vital importance, for example
 - Securing accounts
 - Securing servers
 - Securing data

- [conygre]
fill the gap

Securing Accounts



- When you set up an AWS account, you will have a username and password
 - This should **NEVER** be used
- Instead of using your root account, AWS provides
 - **Identity and Access Management (IAM)**

- [conygre] -
fill the gap

Understanding IAM



- IAM allows you to create
 - Users
 - Groups
 - Roles
 - Policies



Users



Groups



Policies



Roles

- [conygre] -
fill the gap

Users



- **Users** are created for those individuals who need some kind of access to your AWS infrastructure
- They are for things like
 - Creating resources on AWS
 - Viewing resources on AWS
- They are NOT for
 - Database accounts
 - Operating system accounts
 - Application accounts

- [conygre] -
fill the gap

Groups



- **Groups** are as the name implies, groups of users
- Groups might be
 - Devs
 - Testers
 - Infrastructure Engineers
 - DevOps
 - SysOps

- [conygre] -
fill the gap

Roles



- **Roles** are used as
 - Temporary permissions for individuals
 - Temporary permissions for applications
 - Eg. A mobile app requiring permission to upload to S3
 - Services in AWS to have permission to do things
 - An EC2 instance to listen to a queue
 - A Lambda function to be allowed to stop and start servers

- [conygre] -
fill the gap

Policies and Permissions

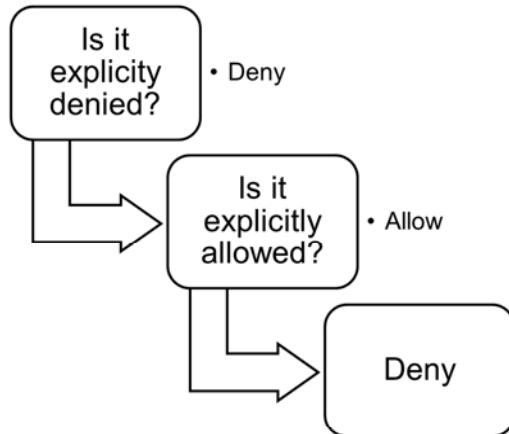


- Users, Groups and Roles can be assigned permissions in the form of **Policies**
- Policies are used to grant or deny permission to AWS resources and services

- [conygre] -
fill the gap

Default Permissions

- Everything is denied by default unless explicitly allowed
- Anything explicitly denied can never be allowed



- [conygre] -
fill the gap

Policies

- Policies can be
 - Custom policies created by you
 - Hand crafted with extremely fine grained access defined
 - AWS Provided
 - Simple policies you can use when required
 - EC2 Full Access
 - S3 Full Access

- [conygre] -
fill the gap

Securing Servers - YOU

- **YOU** are responsible for
 - Patching and updating your servers
 - Malware protection and antivirus
 - Ensuring that you only open the appropriate ports through security groups
 - Ensuring that you place them in appropriate subnets with the correct routing rules

- [conygre] -
fill the gap

Securing Servers - AWS

- Amazon are responsible for ensuring
 - Ensuring that all your security groups and routing rules work as you have specified
 - EC2 instances are completely isolated from each other
 - The security of the physical data centres
 - That all disks at end of life are destroyed to the appropriate international standards

- [conygre] -
fill the gap

Securing Data

- You are responsible for things like
 - For putting the appropriate permissions and access rights onto your S3 buckets, DynamoDB tables, RDS databases etc.
- AWS are responsible for things like
 - Ensuring the permissions you have set actually work
 - Backups and patching of RDS instances happens according to your schedule

- [conygre] -
fill the gap

Encrypting Data

- Many AWS data services support encryption for example, S3 buckets and EC2 drives can be encrypted at rest with keys
 - Provided by AWS
 - Provided by you
- AWS even provide a hardware based key management solution that you can take advantage of at extra cost

- [conygre] -
fill the gap

CloudTrail



- CloudTrail tracks all your IAM user and root user actions within your account
 - Who does what and when
 - Who stopped that server?
 - Who created that queue?
- You need to enable it within your account
- All data goes into an S3 bucket and can be viewed via the console or queried through the CLI
- Great for audit trails

- [conygre] -
fill the gap

AWS Config

- AWS Config allows you to track configuration changes to your resources over time
 - How has this server been changed over time?
 - Resized
 - Stopped / Started
 - Changed security groups
- Also great for audit trails

- [conygre] -
fill the gap

Trusted Adviser

- AWS also has a trusted adviser which will review all of your resources and highlight potential security issues

- [conygre] -
fill the gap

Security Questions

- The AWS Five Pillars also highlight a number of questions you can use to critically look at your security

- [conygre] -
f i l l t h e g a p

Exercise

- Introduction to Identity and Access Management
 - <https://qwiklabs.com/focuses/6211>

- [conygre] -
fill the gap

Summary

- Security concerns
- Securing your Account
- Security your infrastructure
- Security your data
- Encryption
- Auditing tools

- [conygre] -
fill the gap

Introduction to Amazon Lex

Introduction to

Amazon Lex

[conygre] -
f.i.l.i.t.h.g.a.p

Objectives

- What is Lex?
- Why use Lex in particular
- Examples of Lex
- Integration with Amazon Lambda

- [conygre] -
f i l l t h o u g h t

What is Lex?

- Amazon describes Lex as a ‘service for building conversational interfaces into applications using voice and text’.
- Essentially Lex takes all the **Natural Language Understanding** and **Automatic Speech Recognition** from the Amazon Echo and makes it available to developers in order to build a more engaging **UX**.



- [conygre] -
f i l l t h o g a p

Why choose Lex

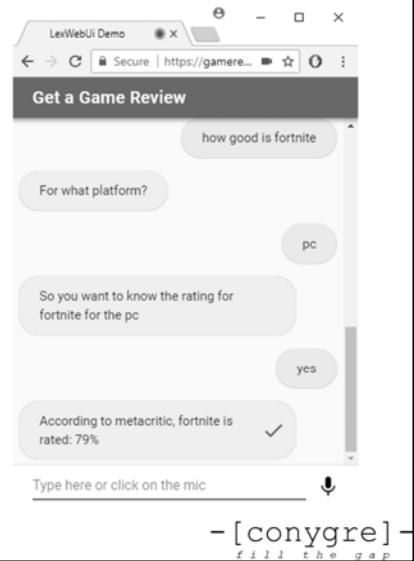
- Full AWS integration
 - Lambda, Cloudwatch, Logging
- All the Machine Learning has been done for you
- Seamless integration into Facebook, Slack, Kik and Twilio SMS

- [conygre] -
f i l l t h o g a p

Introduction to Amazon Lex

Example

- To demonstrate how Lex works you will see how to create a game review chatbot



Building a Chat Bot to Rate Games

- Need to establish how the conversation will flow.

Customer	Bot Response
Hello, I would like to get a rating for a game	Okay, what game would you like to rate?
Minecraft	Okay Minecraft for which platform?
PC	You want me to get a rating for Minecraft for the PC
Yes	
	According to Metacritic, Minecraft for PC is rated 93%

- [conygre] -
f i l l t h o g a p

Introduction to Amazon Lex

Creating the Bot

- Bots can be created through the AWS Web Console

Create your Lex bot

Amazon Lex enables any developer to build conversational chatbots quickly and easily. With Amazon Lex, no deep learning expertise is necessary—you just specify the basic conversational flow directly from the console, and then Amazon Lex manages the dialogue and dynamically adjusts the response. To get started, you can choose one of the sample bots provided below or build a new custom bot from scratch.

CREATE YOUR OWN TRY A SAMPLE

Custom bot BookTrip OrderFlowers ScheduleAppointment

Bot name: GameRatingExample

Language: English (US)

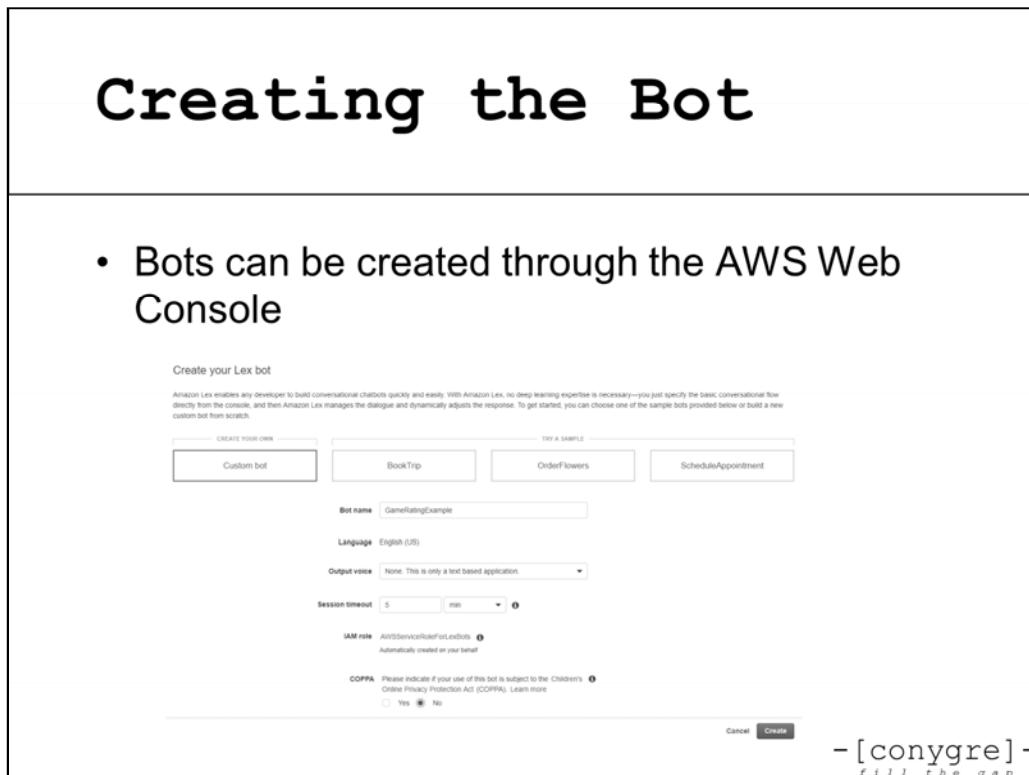
Output voice: None. This is only a text based application

Session timeout: 5 min

IAM role: AWSServiceRoleforLexBots
Automatically created on your behalf

COPPA: Please indicate if your use of this bot is subject to the Children's Online Privacy Protection Act (COPPA). Learn more
 Yes No

Cancel Create - [conygre] - f i l l t h e g a p



Sample Bots

- There are some sample bots you can use to get started quickly
 - Order Flowers
 - Book a trip
 - Schedule an appointment
- Or you can create your own from scratch

- [conygre] -
f i l l t h o u g h t s

Voice Control

- Since Lex is the engine behind the Amazon Echo with Alexa you can optionally select that your chatbot can also work with voice as well as text

Output voice ▾

- If you do enable this, the red dot appears in the browser bar if the user enables it



- [conygre] -
f i l l t h e g a p

Roles

- ChatBots require permissions in order to invoke anything
- If voice is enabled, the Polly SynthesizeSpeech permission is required

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "polly:SynthesizeSpeech"  
            ],  
            "Resource": [  
                "*"  
            ]  
        }  
    ]  
}
```

- [conygre] -
f i l l t h o u g h t s

Intents and Slots

- You now need to set up **Intents** and **Slots**
 - Intent – what the user wants to do
 - Slot – specific pieces of information provided by the user

Customer	Bot Response	Intents and Slots
Hello, I would like to get a rating for a game	Okay, what game would you like to rate?	Intent: Rate Game established
Minecraft	Okay Minecraft for which platform ?	Slot: Game = Minecraft
PC	You want me to get a rating for Minecraft for the PC	Slot: Platform = PC
Yes		Initialise Lambda function to get rating from Metacritic
	According to Metacritic, Minecraft for PC is rated 93%	Intent is now fulfilled

- [conygre] -
final_thoughts

Creating an intent

- To create an Intent, click **Create Intent**
- You can have multiple intents
- In the example we require one intent and our **Intent** is to get a **Game Rating**.



- [conygre] -
f i l l t h e g a p

Invoking the Intent

- We need to tell Lex what kinds of things the customer might enter in order to invoke our Intent
- Lex calls these phrases **utterances**

▼ Sample utterances 

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

How do people feel about {Game}

How good is {Game}

I would like to know the rating of a game

- The words in curly brackets are slots – discussed next

- [conygre] -
f i l l t h o g a p

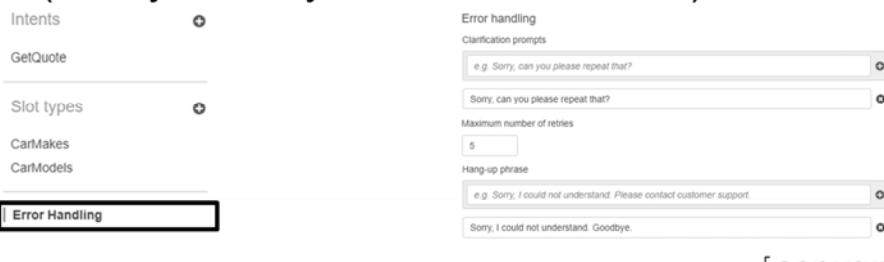
Getting Information from the User

- Almost always, in order to fulfil our intent, we will need to information from the user
 - In our case, we want to know the **Game** that they want to rate and also the **Platform** which it's on
- Each piece of information is referred to as a **Slot** by Lex

- [conygre] -
f i l l t h o u g h t

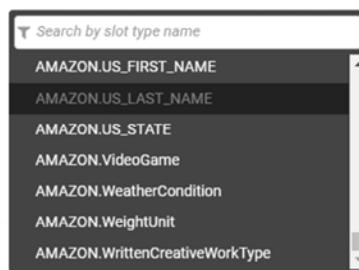
Error Handling

- Sometimes a user will enter something that Lex doesn't understand.
- In this scenario, Lex comes with features to prompt a user to re-enter some information (ideally in a way Lex **will** understand)



Slots and Slot Types

- User data is placed into **Slots**
- Slots always have a Slot Type
- Lex defines some commonly used types
 - There are lots of them!



- [conygre] -
f i l l t h e g a p

Custom Slot Types

- For the **Platform** slot we need to define a custom slot type which lists all the possible platforms that our bot is able to get game ratings for
- In this case we want to make sure that the slot type is **not** set to expandable, as we only want Lex to populate the Platform slot with valid platforms
- Note that slot names are case sensitive



- [conygre] -
f i l l t h e g a p

Custom Slot Types

- Slots also have a Prompt which is the phrase the bot will use to elicit the information from the customer

How do people feel about {Game}

How good is {Game}

I would like to know the rating of a game

- Always place your slot names in curly brackets when adding the prompts

- [conygre] -
f i l l t h o u g h t s

Creating the Slots

- Notice that now the slots have been defined the References to them in the **Utterances** section have been **{color-coded}**

How do people feel about {Game}

How good is {Game}

I would like to know the rating of a game

- [conygre] -
f i l l t h e g a p

Confirmation

- An important ‘good practice’ of Bot Design is to make sure that the user knows what is going on
- Repetition, Confirmation and Repetition are some excellent ways to incorporate this.
- Lex provides a useful **Confirmation Prompt** section to easily ensure the user is happy with the information they have given

The screenshot shows the Amazon Lex console interface. On the left, there is a configuration panel with a 'Confirmation prompt' section checked. It includes fields for 'Confirm' (containing 'So you want to know the rating for (Game) for the (Platform)') and 'Cancel (If the user says "no")' (containing 'Okay, no worries. Let me know if you change your mind'). On the right, a simulated conversation is shown between a user and a bot. The user asks 'how good is minecraft'. The bot responds with 'For what platform?'. The user replies 'pc'. The bot then asks 'So you want to know the rating for minecraft for the pc.' A red box highlights this message. At the bottom, there is a footer with the Conygre logo and the text 'f i l l t h e g a p'.

Fulfilling the Intent

- In Lex an intent can either be **Fulfilled** or **Failed**
- In order to determine this, we can either
 - Use the built in ‘return parameters to client’ function. In which case the intent will be **fulfilled** once Lex has populated all the required **Slots**
Or
– We can utilise Amazon **Lambda** in order to do some logic on the filled **Slots** and determine for ourselves whether or not the intent has *actually* been fulfilled
[Similarly to above, the **lambda** will be called once all the required **slots** are filled.]

- [conygre] -
f i l l t h e g a p

Adding a lambda to Lex

- Incorporating the lambda into our lex bot is straightforward
 - In the fulfilment tab we just tell Lex to call our lambda function
 - The less straightforward part is how exactly Lex interacts with our lambda
- When the lambda is invoked it will be passed a Lex **Event**
 - This **event** is a JSON construct which describes the current intent and the slots that are filled

```
{
  "messageVersion": "1.0",
  "invocationSource": "FulfillmentCodeHook",
  "userId": "pcp241vzjsv7m97i2og0qtv12gbjiuob",
  "sessionAttributes": {},
  "requestAttributes": None,
  "bot": {
    "name": "GameRating",
    "alias": "$LATEST",
    "version": "$LATEST"
  },
  "outputDialogMode": "Text",
  "currentIntent": {
    "name": "GetGameRating",
    "slots": {
      "Game": {"minecraft", "Platform": "pc"}
    },
    "slotDetails": {
      "Game": {
        "resolutions": [],
        "originalValue": "minecraft"
      },
      "Platform": {
        "resolutions": [{"value": "pc"}],
        "originalValue": "pc"
      }
    },
    "confirmationStatus": "Confirmed"
  },
  "inputTranscript": "yes"
}
```

Event example

- [conygre] -

Adding a lambda to Lex

- A simple python script should do the trick for getting our game rating from Metacritic
- First we import some libraries for dealing with HTML data and requests

```
from urllib.request import Request, urlopen  
from urllib.error import HTTPError  
from bs4 import BeautifulSoup
```

- [conygre] -
f i l l t h o u g h t s

Adding a lambda to Lex

- Then we add in the **lambda_handler** function which deals with the stuff from Lex

```
def lambda_handler(event, context):
```

- The **event** parameter contains our JSON from the chatbot
 - In the function we can extract our **Game** and **Platform** information

```
game = event['currentIntent']['slots']['Game']
platform = event['currentIntent']['slots']['Platform']
```

- [conygre] -
f i l l t h o u g h t s

Adding a lambda to Lex

- Then we need some logic for dealing with games with multiple words in their title
 - The Metacritic Web site uses hyphens to deal with this, so we need to hyphenate the game name

```
words = game.split()  
  
gameName = words[0]  
for word in range(1,len(words)):  
    gameName = gameName+'-'+words[word]  
  
print("Normalized Game name: "+gameName)  
url = baseUrl+platform+"/"+gameName
```

- [conygre] -
f i l l t h o u g h p

Adding a lambda to Lex

```
#make request to metacritic
request = Request(url,headers={'User-Agent': 'Mozilla/5.0'}) #set a browser useragent
fulfillmentType = 'Failed' #assume failure unless we can get a rating
try:
    htmlData = urlopen(request).read()

    soup = BeautifulSoup(htmlData, 'html.parser')

    #find the rating from the metacritic page if it exists
    try:
        rating = soup.findAll("span", {"itemprop": "ratingValue"})[0].string
        result = 'According to metacritic, '+game+' is rated: '+rating+'%'
        fulfillmentType = 'Fulfilled' #update fulfillment type to success
    except IndexError:
        result = 'Metacritic doesn\'t seem to have a rating for '+game+' yet'
    except HTTPError:
        result = 'I wasn\'t able to find a rating for: '+gameName'
```

- [conygre] -
f i l l t h e g a p

Adding a lambda to Lex

- Finally we can create the JSON **response** to send back to Lex
- And tell Lex to use our Lambda as the fulfilment criteria

```
response = {
    "dialogAction": {
        "fulfillmentState": "fulfillmentType",
        "type": "Close", "message": [
            {
                "contentType": "PlainText",
                "content": "result"
            }
        ]
    }
}
return response
```



- [conygre] -
f i l l t h e g a p

Testing the Bot

- In the Lex console make sure you remember to click the **Save Intent** button
- To test, expand Test Chatbot menu on the right



- [conygre] -
f i l l t h e g a p

Some finishing touches

- When your Lex bot prompts users for input it currently will not necessarily be able to cope with all sensible inputs

– Consider the following

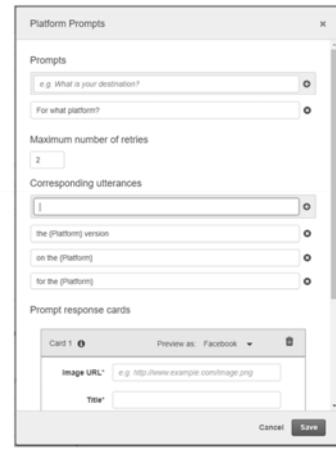


- Lex didn't know how to respond to this longer phrase, however...

- [conygre] -
f i l l t h o g a p

Some finishing touches

- Lex allows for the customisation of expected responses to prompts. In this case we can go into the prompt settings for **Platform** and put in some **Corresponding Utterances**.
- Now the conversation works.



- [conygre] -
f i l l t h e g a p

Summary

- Lex is an AWS Environment for **Natural Language Understanding**
- **Intents** describe what your bot will do
- **Slots** describe the different pieces of information your bot will need to perform the **intent**
- Slots can be explicitly **confirmed** to improve **UX**
- **Lambda** is implemented to create advanced intents that your bot can **Fulfil**

- [conygre] -
f i l l t h o u g h t s

Introduction to AWS Lambda

Introduction to Lambda Functions

- [conygre] -
f i l l t h e g a p

Introduction to AWS Lambda

Objectives

- What are Lambda Functions?
- Why use them?
- How to create a Lambda
- Example Lambda Functions

- [conygre] -
fill the gap

Running Servers

- Consider the code that might need to be invoked when someone uses a chat bot
 - You might run a server somewhere hosting an API used by the chat bot
 - You would be paying for that server 24/7 and would need to ensure that it was highly available (so you might even run two servers)
 - You would also need to ensure that it could scale in response to load
 - New deployments would involve updating these servers every time

- [conygre] -
fill the gap

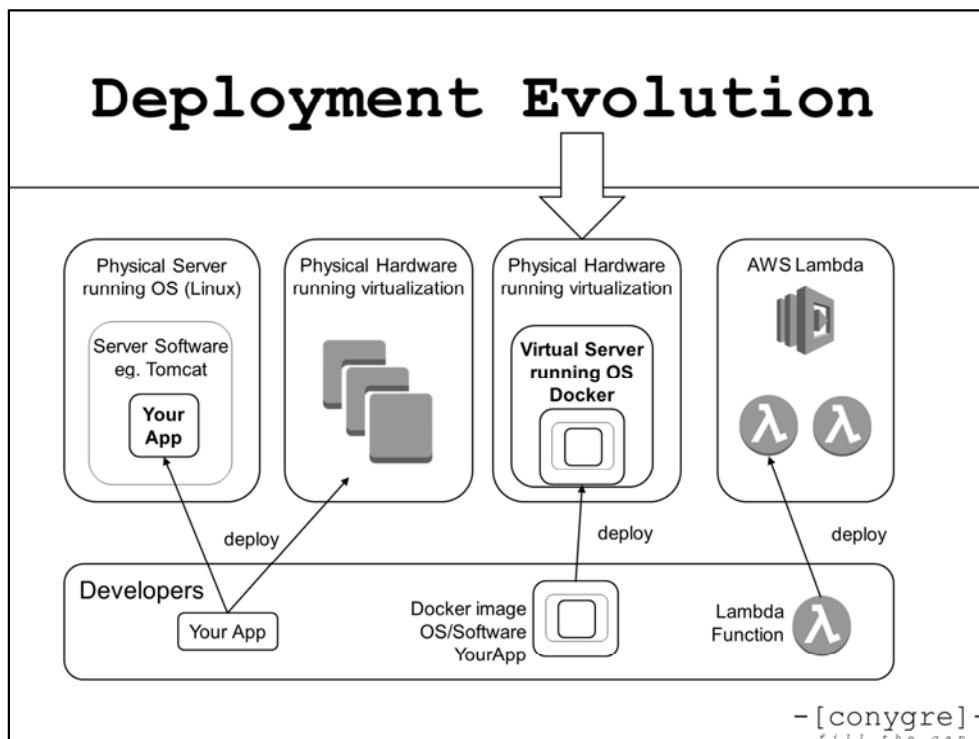
Introduction to AWS Lambda

Introducing Lambda

- Lambda Functions allow you to create **serverless architectures**
- In the serverless paradigm, you would create a function that could be invoked when someone uses your chatbot
- The function is all that you deploy!
 - No servers
 - No maintenance
 - No ongoing server costs
 - No hassle!

- [conygre] -
fill the gap

Introduction to AWS Lambda



Introduction to AWS Lambda

Lambda Technology

- Currently on AWS, Lambdas can be written using
 - Java
 - Python 2.x or 3.x
 - NodeJS various versions
 - C# with .NET Core
 - Go

- [conygre] -
fill the gap

Introduction to AWS Lambda

Triggering Lambda

- A direct invocation
 - A REST API call via the API Gateway service
- An event such as
 - S3 Bucket upload
 - Alarm
 - Message arriving on a queue
 - Notification
- Updates to streams such as
 - Kinesis streams
 - DynamoDB streams
- Run on a schedule

- [conygre] -
fill the gap

Introduction to AWS Lambda

Permissions

- Lambda functions have two lots of permissions to consider
 - Who can call the lambda
 - What can the lambda do
- This is all controlled through Identity and Access Management (IAM)

- [conygre] -
fill the gap

Introduction to AWS Lambda

Lambda Configuration

- Lambda functions are configured with
 - The code to run
 - The event to trigger them
 - The amount of memory to be allocated which is currently between 256mb and 3000mb
 - A timeout
 - IAM roles to specify permissions

- [conygre] -
fill the gap

Introduction to AWS Lambda

Monitoring and Logging

- All Lambda metric data and logging goes to **CloudWatch**
- Cloudwatch is the AWS monitoring service
- To facilitate this you ensure that Lambdas run in a role with access to CloudWatch and CloudWatch Logs



Introduction to AWS Lambda

The Lambda Code

- Lambda code can be created
 - Using the inline editor in the browser or
 - In an IDE and then
 - A zip of your code can be uploaded
 - A zip of your code can be uploaded to an S3 bucket

Code entry type

Edit code inline
Edit code inline
Upload a ZIP file
Upload a file from Amazon S3

- Note for Java and C# it cannot be edited inline since it must be compiled

- [conygre] -
fill the gap

Introduction to AWS Lambda

Steps to Creating a Lambda

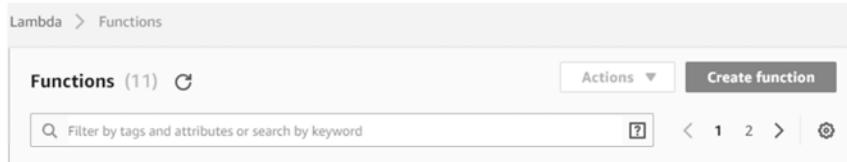
- Using the Web console, you must
 1. Create the Lambda
 2. Set up the trigger
 3. Set up the permissions
 4. Upload / Edit the code
 5. Configure the timeout / memory

- [conygre] -
fill the gap

Introduction to AWS Lambda

Creating a 'HelloWorld' Lambda

- At the Lambda Console, click **Create function**



- There are many blueprints and there is also a marketplace of predefined lambdas, but to do your own, select **Author from Scratch**

- [conygre] -
fill the gap

Introduction to AWS Lambda

Enter the Basic Details

Author from scratch Info

Name

Runtime

Role
Defines the permissions of your function. Note that new roles may not be available for a few minutes after creation. Learn more about Lambda execution roles.

Existing role
You may use an existing role with this function. Note that the role must be assumable by Lambda and must have Cloudwatch Logs permissions.

- At a minimum, the role will need CloudWatch Logs

- [conygre] -
fill the gap

Introduction to AWS Lambda

View the Code

- Click **Create Function**
- Switch the language to Python 3.x and note the code provided already has a function the name of which is in the Handler field

The screenshot shows the AWS Lambda code editor interface. At the top, there are three dropdown menus: 'Code entry type' (set to 'Edit code inline'), 'Runtime' (set to 'Python 3.6'), and 'Handler' (set to 'lambda_function.lambda_'). Below these is a toolbar with standard file operations: File, Edit, Find, View, Goto, Tools, Window. On the left, there's a sidebar labeled 'Environment' with a 'HelloWorldExample' folder containing a 'lambda_function.py' file. The main area is titled 'lambda_function' and contains the following Python code:

```
1 def lambda_handler(event, context):
2     # TODO: Implement
3     return 'Hello from Lambda'
```

At the bottom right of the editor window, there is a watermark that reads '- [conygre] - fill the gap'.

Create a Test

- This Lambda already returns some text, so you can run it as is
- To test it, click **Test** at the top of the function
- We need to create a sample **test event**
 - If we were doing a chatbot, then we might provide an example event that could come from Lex, but for hello world, the event is irrelevant since we are not really using it
- Enter a name for your test event such as **HelloEvent**
 - Leave the JSON as is and click **Create**

- [conygre] -
fill the gap

Introduction to AWS Lambda

Run the Test

- To run the test, click **Test** again, and this time it will run the function

The screenshot shows the AWS Lambda console for a function named "HelloWorldExample". At the top, there are tabs for "Throttle", "Qualifiers", "Actions", "HelloEvent", "Test", and "Save". The "Test" tab is currently selected. Below the tabs, a modal window titled "Execution result: succeeded (logs)" is open. The modal has two sections: "Details" and "Log output". The "Details" section contains a message box with the text "Hello from Lambda!". The "Summary" section provides execution statistics:

- Code SHA-256: qUJymYXveeRNouOfS/YY2gt2dhdzXCBUleyhPrmY1HB=
- Duration: 0.61 ms
- Resources configured: 128 MB
- Request ID: b39fe369-6db8-11e8-ac55-47d5945a4be9
- Billed duration: 100 ms
- Max memory used: 21 MB

The "Log output" section displays CloudWatch logs for the function's execution:

```
START RequestId: b39fe369-6db8-11e8-ac55-47d5945a4be9 Version: $LATEST
END RequestId: b39fe369-6db8-11e8-ac55-47d5945a4be9
REPORT RequestId: b39fe369-6db8-11e8-ac55-47d5945a4be9 Duration: 0.61 ms
Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 21 MB
```

At the bottom right of the modal, there is a watermark: "nygre] -- fail the gap".

Introduction to AWS Lambda

View the Monitoring

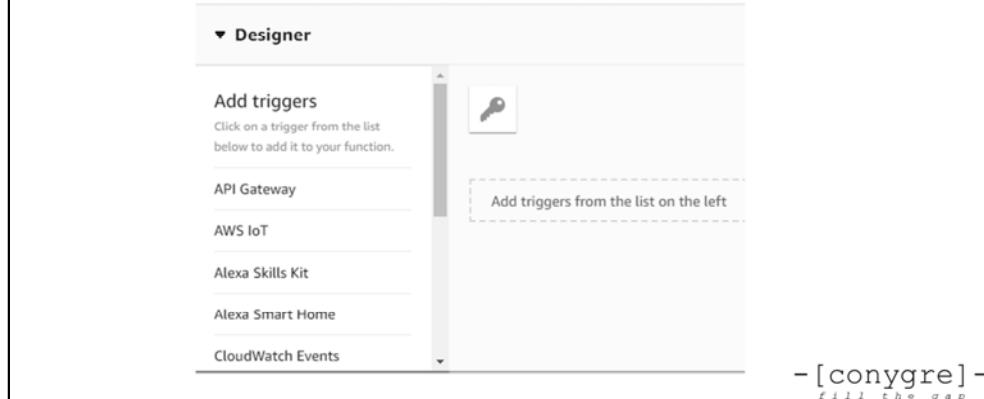
- You can easily see invocation metrics and logs by clicking on the **Monitoring** tab

The screenshot shows the AWS Lambda function configuration interface for 'HelloWorldExample'. The 'Monitoring' tab is selected. Below it, two line charts are displayed: 'Invocations' and 'Duration'. Both charts cover the last 24 hours. The 'Invocations' chart shows a single sharp peak at approximately 12:00 on June 11. The 'Duration' chart shows a similar sharp peak at the same time, with values ranging from 0 to 0.6 milliseconds. At the bottom of each chart, there are buttons for 'Jump to Metrics' and 'Jump to Logs'. A watermark '- [conygre] - fill the gap' is visible at the bottom right of the screenshot.

Introduction to AWS Lambda

Adding an Event Trigger

- Triggers can be added graphically from the list of possible triggers in the Web console



Introduction to AWS Lambda

A Scheduled Event Example

- Scheduled Events can be the basis of a trigger
 - Same time every day
 - Every x minutes or hours
- Scheduled events come from Cloudwatch
 - Schedules are specified using the **cron** standard



CloudWatch Events

OnHourAnd5MinsPast

arn:aws:events:eu-west-1:963778699255:rule/OnHourAnd5MinsPast

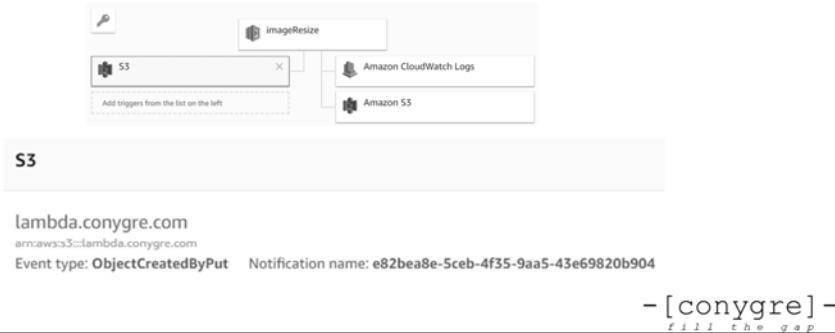
Schedule expression: cron(0,5 * ? * MON-FRI *) Description: On the hour and 5 minutes past the hour

- [conygre] -
f i l l t h e g a p

Introduction to AWS Lambda

A Bucket Upload Example

- Lambdas can be triggered by uploads to buckets
 - Notice the permissions on the right are S3 and the trigger on the left is S3



Introduction to AWS Lambda

Using Libraries

- It is possible to use additional libraries within your lambdas
- All the code will need to be uploaded for the Lambda in order for it to work
- Many popular libraries are already available so will not need to be uploaded
 - Check the Lambda documentation for the library you wish to use

- [conygre] -
fill the gap

Introduction to AWS Lambda

Lab Exercise

- To create your own basic Lambda complete the following exercise
- Introduction to AWS Lambda
 - <https://www.qwiklab.com/focuses/284?parent=catalog>

- [conygre] -
fill the gap

Introduction to AWS Lambda

Summary

- What are Lambda Functions?
- Why use them?
- How to create a Lambda
- Example Lambda Functions

- [conygre] -
fill the gap

Chatbot Deployment using Amazon Lex

[conygre] -
f i l i t h o g a p

Objectives

- Where can Lex be deployed
- Deployment to a Web UI example

- [conygre] -
f i l l t h o u g h t s

ChatBot Deployment Options

- ChatBots built using Lex have a variety of deployment options
- Alternative ChatBot technologies will have a similar range of choices

- [conygre] -
f i l l t h o u g h t s

Deployment Platforms

- Lex can be deployed to any of the Following Platforms
 - Web UI
 - Android
 - Slack
 - Facebook Messenger (Text Only)
 - Twilio (SMS api)



- [conygre] -
f i l l t h o u g h t s

Basic Web Deployment

- A deployed Bot can be programmatically accessed and tested for Web sites
- The deployment process for Lex is very streamlined, so you can easily make updates to your bot and redeploy
 - This facilitates behaviour driven development – more later

- [conygre] -
f i l l t h o u g h t s

ChatBot Training

ChatBot Deployment

Web UI - Example

The image displays two side-by-side screenshots of a web-based chatbot interface. Both screenshots have a dark header bar with the text "Get a Game Review". Below the header, there is a message box containing the text: "You can ask for a game review. Please enter your question below." The left screenshot shows a user input "I would like to get a game rating" followed by a bot response "No problem, what game?". The user then inputs "trine" and the bot asks "For what platform?". The user inputs "pc" and the bot responds "So you want to know the rating for trine for the pc". The user inputs "yes" and the bot replies "According to metacritic, trine is rated: 80%" with a checkmark icon. At the bottom, there is a text input field with the placeholder "Type here or click on the mic" and a microphone icon. The right screenshot follows a similar pattern but with "destiny" instead of "trine", resulting in a rating of 76%.

- [conygre] -
f i l l t h e g a p

Deployment Using CloudFormation

- CloudFormation is a service from Amazon to allow you to deploy entire infrastructures using a script
 - The details are beyond the scope of this course
- During the chapter you will see the fundamentals of how you can use it to successfully deploy your chatbot
- Full details can be found in this Amazon Blog post
 - <https://aws.amazon.com/blogs/machine-learning/deploy-a-web-ui-for-your-chatbot/>

- [conygre] -
f i l l t h o u g h t s

The Deployment Process

1. Cloudformation is used to create the Stack which the Bot will be hosted on
2. Permissions are set up in Cognito to allow the anonymous user to make requests
3. A set of pages and scripts are then created for you by the cloudformation template
4. A provided script snippet can then be pasted into your website

- [conygre] -
f i l l t h o u g h t s

Creating the Stack

- AWS provide **CloudFormation** templates to quickly get a stack setup for your Lex Bot
- These work most easily in **N. Virginia** due to the location of the Bootstrap Bucket referenced in the template.

- [conygre] -
f i l l t h o u g h t

ChatBot Deployment

The CloudFormation Parameters

Specify Details

Specify a stack name and parameter values. You can use or change the default parameter values, which are defined in the AWS CloudFormation template. Learn more.

Stack name [x]

Parameters

Deployment Parameters

CodeBuildName	<input type="text" value="lex-web-ui"/>	Name of the CodeBuild project to be created. Used to configure and directly deploy the web app to S3. Must be unique per region.
CleanupBuckets	<input type="text" value="true"/>	If set to True, buckets created for the Pipeline and to store the web application will be deleted on CloudFormation stack delete. If set to False, S3 buckets will be retained.
BootstrapBucket	<input type="text" value="aws-bigdata-blog"/>	S3 bucket containing pre-staged nested templates and source artifacts
BootstrapPrefix	<input type="text" value="artifacts/aws-lex-web-ui/artifacts"/>	S3 prefix where the templates and source are stored under

Lex Bot Configuration Parameters

BotName	<input type="text"/>	Name of an existing Lex Bot to be used by the web ui. This is an optional parameter. If left empty, a Bot based on the OrderFlowers sample will be automatically created.
BotNamePrefix	<input type="text" value="WebUI"/>	Prefix to add to Lex resource names when using the sample bot. Ignored if you provide your own bot. Must conform to the permitted Lex Bot name syntax (alpha characters).
ShouldDeleteBot	<input type="text" value="true"/>	If set to True, the Lex bot and associated resources will be deleted when the stack is deleted. Otherwise, the bot will be preserved. Only applies if the bot is created by this stack.

-[conygre]-
f i l l t h e g a p

Managing Cognito

- Cognito is a service typically used on AWS for user authentication, often for mobile applications
- When a user interacts with your bot, it will need to have permissions specified
- Lex Bots run the permissions of an **Unauthenticated Cognito User**
- In **IAM** we must ensure that a **Cognito Identity Pool** is available where the unauthenticated user has **Lex Post** permissions

- [conygre] -
f i l l t h o u g h t s

ChatBot Deployment

Managing Cognito

The screenshot shows the AWS Lambda policy editor interface. At the top, there's a 'Policy name' dropdown set to 'LexPost'. Below it is a 'Policy summary' section with tabs for 'JSON' and 'Edit policy'. The main area displays a JSON-based policy document:

```
1  {
2      "Version": "2012-10-17",
3      "Statement": [
4          {
5              "Sid": "VisualEditor0",
6              "Effect": "Allow",
7              "Action": [
8                  "lex:PostText",
9                  "lex:PostContent"
10             ],
11             "Resource": [
12                 "arn:aws:lex:us-east-1:963778699255:bot:WebUiOrderFlowers:*",
13                 "arn:aws:lex:us-east-1:963778699255:bot:GameRatingExample:*",
14                 "arn:aws:lex:us-east-1:963778699255:bot:insurancebotExample:*"
15             ]
16         }
17     ]
18 }
```

A red box highlights the 'Effect', 'Action', and 'Resource' fields of the first statement. At the bottom right of the editor window, there's a watermark: '- [conygre] - f i l l t h e g a p'.

Using the ChatBot from HTML

- Once the template has successfully run, a set of Web pages are created within an S3 bucket
- The **URLs** to these files are outputs from the cloudformation template

▼ Outputs	
Key	Value
ParentPageUrl	https://gamereviewstack-codebuilddeploy-xmvz-webappbucket-71jwxn1oy8i0.s3.amazonaws.com/parent.html
SnippetUrl	https://gamereviewstack-codebuilddeploy-xmvz-webappbucket-71jwxn1oy8i0.s3.amazonaws.com/iframe-snippet.html
CodeBuildUrl	https://console.aws.amazon.com/codestar/home?region=us-east-1#/projects/GameReviewCodeBuild/view
WebAppUrl	https://gamereviewstack-codebuilddeploy-xmvz-webappbucket-71jwxn1oy8i0.s3.amazonaws.com/index.html

- [conygre] -
f i l l t h e g a p

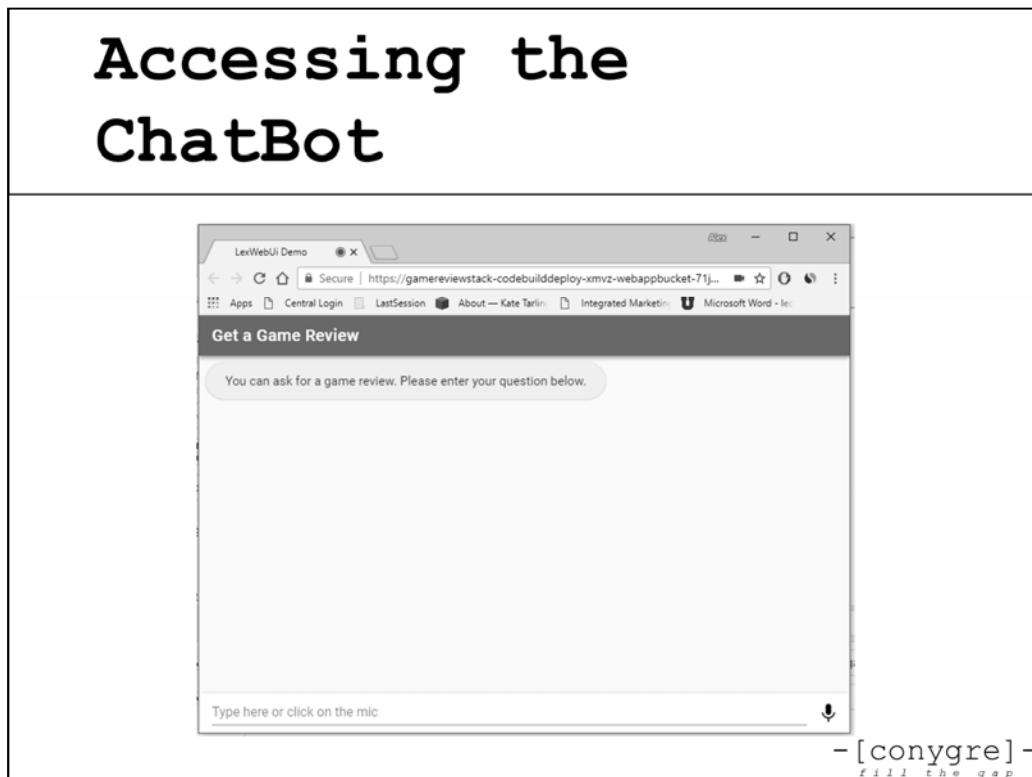
Using your ChatBot in a Web Site

- And the **code snippet** to implement it into your **own webpage** is found [here](#) too

▼ Outputs	
Key	Value
ParentPageUrl	https://gamereviewstack-codebuilddeploy-xmvz-webappbucket-71jwxn1oy8l0.s3.amazonaws.com/parent.html
SnippetUrl	https://gamereviewstack-codebuilddeploy-xmvz-webappbucket-71jwxn1oy8l0.s3.amazonaws.com/frame-snippet.html
CodeBuildUrl	https://console.aws.amazon.com/codebuild/home?region=us-east-1#/projects/GameReviewCodeBuild/view
WebAppUrl	https://gamereviewstack-codebuilddeploy-xmvz-webappbucket-71jwxn1oy8l0.s3.amazonaws.com/index.html

- [conygre] -
f i l l t h e g a p

ChatBot Deployment



Summary

- Where can Lex be deployed
- Deployment to a Web UI example

- [conygre] -
f i l l t h o u g h t s

Testing ChatBots using Cucumber

[conygre] -
f i l l t h e g a p

Objectives

- What is Behaviour Driven Development (BDD)
- The Benefits of BDD
- The Relationship between BDD and Other Agile Practices
- What is Cucumber?
- Defining Features and Scenarios
- Linking Features and Scenarios to Tests
- Testing a ChatBot Example
- When to Run Tests

- [conygre] -
f i l l _ t h e _ g a p

What is BDD?

- Behavior-driven development (BDD) is a software development methodology in which an application is specified and designed by describing how its behavior should appear to an outside observer.
– *techtarget.com*

- [conygre] -
f i l l t h e g a p

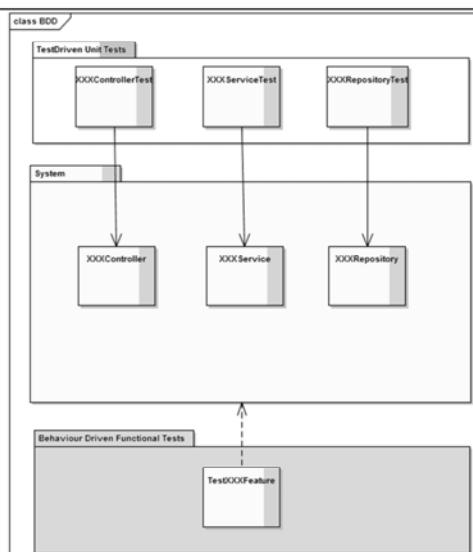
Benefits of BDD

- Automated functional testing
- Completeness of functional testing
- Application only does what is required
- Behavioural requirements are captured in a format understood by technologists and business people alike
- Test cases are defined up front and the requirements are also the test cases

- [conygre] -
fill the gap

Relationship with TDD

- Behavioural tests are written first (BDD)
- The system is then built using Test Driven Development
- At the end, you have two sets of **automated** tests
 - Unit tests (TDD)
 - Acceptance tests (BDD)



BDD and User Stories

- **User Stories**, which are a common way of capturing requirements in Agile projects, have **acceptance criteria** specifying the testable requirements for a feature
- Acceptance criteria can be captured as the BDD tests
 - But how? This is where testing tools like Cucumber come in

- [conygre] -
fill the gap

Introducing Cucumber

- Cucumber is a tool that facilitates the creation and running of BDD tests
- Cucumber supports many different languages and platforms including
 - Java
 - C#
 - C++
 - Scala
 - Python

- [conygre] -
fill the gap

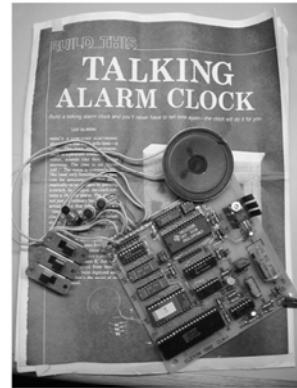
Cucumber Feature Files

- The key artefact in Cucumber is the **Feature File**
- The Feature file specifies a **feature** as a series of **scenarios** which provide **examples** of how the system must work
 - This results in **example based specifications**

- [conygre] -
fill the gap

Worked Example – Time To Text

- Consider the example of a Speaking Clock system that provides a text value for a given time of day
 - 12pm returns the text "midday"
 - 12am returns the text "midnight"
- This can be captured in the form of a feature file



- [conygre] -
fill the gap

Feature File

- Below is a Feature file for our system

Feature: SpeakingClock

In order to get the time as text
I want to be able to pass in a time
So that I can get a suitable text value back

Scenario: ConvertMidnightToText

Given the time is 0 hours and 0 minutes
When I request the time
Then the result should be midnight

Scenario: ConvertMiddayToText

Given the time is 12 hours and 0 minutes
When I request the time
Then the result should be midday

SpeakingClock.feature

- [conygre] -
file thre gap

Feature File Explained

- The top level Feature describes the overall functionality to be implemented

```
Feature: SpeakingClock
In order to get the time as text
I want to be able to pass in a Datetime
So that I can get a suitable String back
```

- This is not read by a machine, but will appear in your automated test results

- [conygre] -
fill the gap

The Scenarios

- Each Scenario follows a standard convention

Keyword	Purpose	
Given	Arrange the test	Arrange
And	Additional aspects for arranging the test (if required)	Arrange
When	Specify the action that is to be done that requires testing	Act
Then	Assert what you expect the result to be	Assert

- For those of you familiar with unit testing, you might recognise the sections as equivalents of
 - **Arrange, Act, Assert**

- [conygre] -
fill the gap

Scenario Example

- Below is an example scenario for our text to time engine

```
Scenario: ConvertMidnightToText
Given the time is 0 hours and 0 minutes
When I request the time
Then the result should be midnight
```

- [conygre] -
fill the gap

Creating Feature Files

- Each feature will have multiple scenarios that capture examples of how the system must work
- Scenarios can be created by business people and technical people
- Those involved in the process agree terminology for the domain in which they are working (**domain driven design**)
 - The agreed terms are then used consistently in the scenario text (important for automation)

- [conygre] -
f i l l _ t h e _ g a p

What about the Code?

- To run as an automated test against the system, a code artefact needs to be created
- These code artefacts are called **Step Flow Definitions**
- These can be written in one of the supported languages

- [conygre] -
fill the gap

Step Flow Definition

- Below is the step flow definition for our example written in C#

```
public class SpeakingClockSteps {
    [Given(@"the time is (.*) hours and (.*) minutes")]
    public void GivenTheTimeIsHoursAndMinutes(int hours, int minutes) {
        // set up a clock
    }
    [When(@"I request the time")]
    public void WhenIRquestTheTime() {
        //request the time from the clock
    }
    [Then(@"the result should be (.*)")]
    public void ThenTheResultShouldBeTimeAsText(string expectedTimeAsText) {
        // assert what you expect to happen
    }
}
```

- [conygre] -
fill the gap

Steps Explained

- Each step is processed by Given / When / Then attributes
- Each attribute contains a regular expression with a pattern matching the text in the scenario
- Parameter values in the text are placed in parentheses which are then automatically available as method parameters

```
[Given(@"the time is (.*) hours and (.*) minutes")]
public void GivenTheTimeIsHoursAndMinutes(int hours, int minutes)
```

- [conygre] -
fill the gap

Within the Methods

1. Arrange the system so it is ready for testing

```
[Given(@"the time is (.*) hours and (.*) minutes")]
public void GivenTheTimeIsHoursAndMinutes(int hours, int minutes) {
    dateTime = new DateTime(2016,10,27,hours,minutes,0);
    speakingClock = new SpeakingClock();
}
```

2. Exercise the system to see what happens

```
[When(@"I request the time")]
public void WhenIRequestTheTime() {
    actualTimeAsText = speakingClock.GetTimeAsText(dateTime);
}
```

3. Verify the result

```
[Then(@"the result should be (.*)")]
public void ThenTheResultShouldBe(string expectedTimeAsText) {
    Assert.AreEqual(expectedTimeAsText, actualTimeAsText);
}
```

- [conygre] -
fill the gap

Finally - What was Tested

- Below is the simple class we were testing

```
public class SpeakingClock : ISpeakingClock
{
    public string GetTimeAsText(DateTime dateTime)
    {
        if (dateTime.Hour == 0 && dateTime.Minute == 0)
            return "midnight";
        else if (dateTime.Hour == 12 && dateTime.Minute == 0)
            return "midday";
        return null;
    }
}
```

- [conygre] -
fill the gap

Setting up Cucumber in C#

- In Visual Studio you can install the SpecFlow Cucumber tools
- In the .NET project then add the following references
 - TechTalk.SpecFlow
 - TechTalk.SpecFlowRunner
- You will then have wizards to help create the Feature files
 - The StepFlow class stubs can be generated automatically from the feature file

- [conygre] -
fill the gap

Setting up Cucumber in Java

- Both IntelliJ and Eclipse have plugins and extensions for Cucumber
- Projects built with Maven can automatically run your cucumber tests

```
<dependency>
<groupId>info.cukes</groupId>
<artifactId>cucumber-java</artifactId>
<version>${cucumber.version}</version>
<scope>test</scope>
</dependency>
<dependency>
<groupId>info.cukes</groupId>
<artifactId>cucumber-junit</artifactId>
<version>${cucumber.version}</version>
<scope>test</scope>
</dependency>
```

- [conygre] -
fill the gap

Java Step Flow Definitions

- Java step flows are almost identical to the C# ones except annotations are used instead of attributes

```
@When("^the customer searches for items published between (\\d+) and (\\d+)$")
public void setSearchParameters(@Format("yyyy") final Date from, @Format("yyyy") final Date to) {
    // act
}
```

- [conygre] -
fill the gap

Testing Chatbots using Cucumber

Working with DataSets

- When using Cucumber you can also work with datasets that can be defined in the feature file
- Consider the time converter example
 - If working with lots of different time examples, you wouldn't want lots of scenarios, a table of possible values would be easier

- [conygre] -
fill the gap

DataSet Example C#

- Below is a scenario written with a set of data

```
Scenario: ConvertMultipleTimesToText
Given I have been provided this set of times and expected results
```

Hours	Minutes	ExpectedText
0	0	midnight
12	0	midday

```
When I request all the times as text
Then all the text should match
```

- Create a C# class to match the table

```
public class TimeTable {
    public int Hours { get; set; }
    public int Minutes { get; set; }
    public string ExpectedText { get; set; }
}
```

- [conygre] -
fill the gap

Processing the Set

- To process the set, the method with the Given attribute is different

```
private List<TimeTable> timeTables;  
[Given(@"I have been provided this set of times and expected results")]  
public void GivenIHaveBeenProvidedThisSetOfTimesAndExpectedResults(Table tableOfTimes)  
{  
    timeTables = tableOfTimes.CreateSet<TimeTable>().ToList();  
}
```

- SpecFlow provides a **Table** class that has methods to convert your table into a regular List<YourType> so long as the properties of the type match the table columns

- [conygre] -
fill the gap

DataSet Example Java

- Java is even easier than C#, just create a class matching your table with getters/setters
- Then have the @Given method to take in a parameter of a List<YourType>

```
@Given("^the system is initialized with the following data$")
public void initializeTheSystem(final List<TimeTable> timeTables) {
    // arrange
}
```

- [conygre] -
fill the gap

Testing Chatbots using Cucumber

Integration with Selenium

- One of the most useful aspects of Cucumber is that you can combine it with almost any UI technology
- A common test platform for Web is selenium and Cucumber integrates easily with Selenium

- [conygre] -
fill the gap

Testing Chatbots using Cucumber

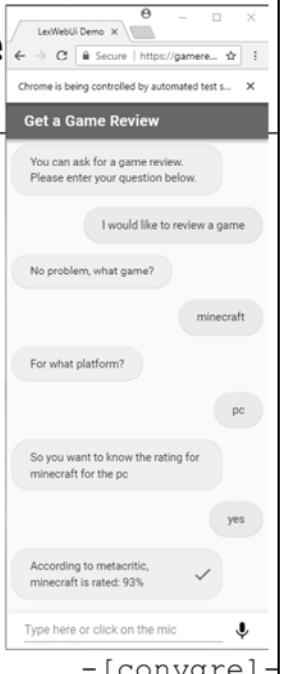
Selenium Example

- Our example will be a behavioural test for a chatbot that provides game reviews

Feature: Requesting a game review from a chat bot
Provided with the name of the game and the platform the chatbot should respond with a rating

Scenario: Get a Rating for Minecraft

Given the user enters 'I would like to review a game'
When asked for the game title the user enters 'minecraft'
And when asked for the game platform the user enters 'pc'
And when prompted to confirm the user enters 'yes'
Then the chatbot should respond with 'According to metacritic, minecraft is rated: 93%'



The screenshot shows a web browser window titled 'LexWebUI Demo' with the URL 'https://gamer...'. The page has a dark header with the text 'Get a Game Review'. Below it, a message box says 'You can ask for a game review. Please enter your question below.' A user message 'I would like to review a game' is shown, followed by a bot response 'No problem, what game?'. The user replies 'minecraft'. The bot asks 'For what platform?' and the user replies 'pc'. The bot then says 'So you want to know the rating for minecraft for the pc' and the user replies 'yes'. Finally, the bot provides the rating: 'According to metacritic, minecraft is rated: 93%' with a checkmark icon.

Testing Chatbots using Cucumber

The Selenium Drivers

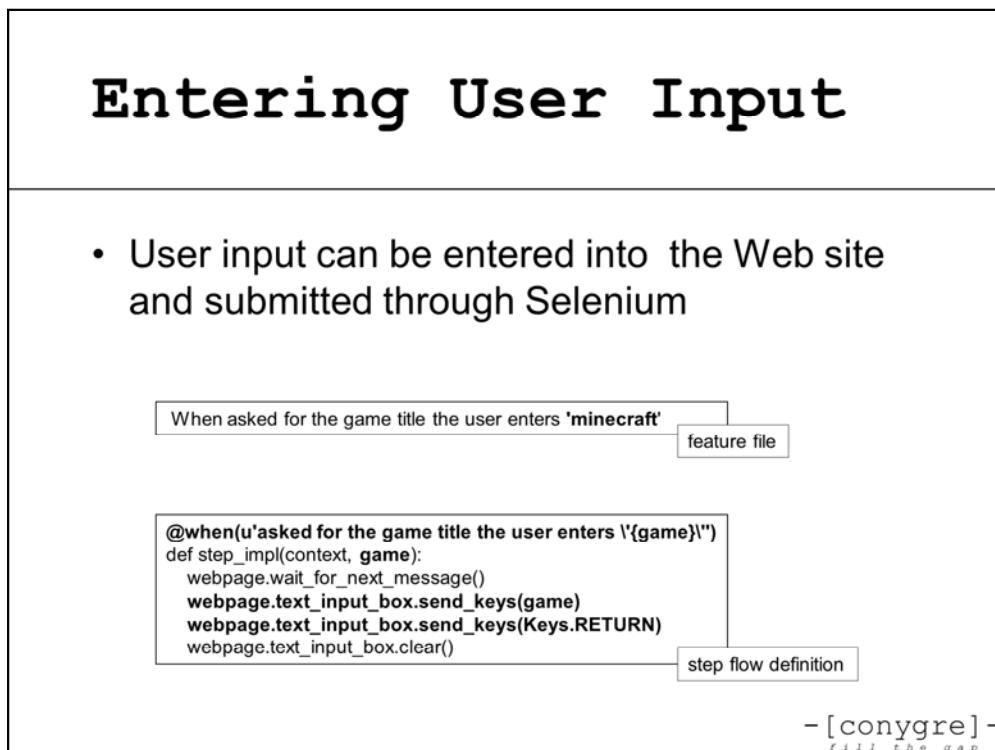
- Selenium uses WebDrivers to run the test
- In our example, it will use Python and the ChromeDriver

```
def load_chatbot(self):
    self.driver = webdriver.Chrome()
    self.website = self.driver.get('https://locationofwebpage.com')
    ...
    self.text_input_box = self.driver.find_element_by_id('text-input')
```

- Via the driver, elements on the page can be located
 - The text-input box in this case

- [conygre] -
fill the gap

Testing Chatbots using Cucumber



Testing Chatbots using Cucumber

Capturing Page Content

- In this example, we want to confirm that the correct score is returned

Then the chatbot should respond with 'According to metacritic, minecraft is rated: 93%'

```
@then(u'the chatbot should respond with '{fulfillment}')
def step_implementation(context, fulfillment):
    ..
    result_message = webpage.get_current_message_again()
    assert_that(result_message, starts_with(fulfillment))

def get_current_message_again(self):
    css_selector = '.message-bot:nth-child(' + str(self.number_of_messages-2) + ')'
    latest_message_element = \
        self.driver.find_element_by_css_selector(css_selector=css_selector)
    latest_message_from_bot = latest_message_element.text
    return latest_message_from_bot
```

- [conygre] -
full_thor gap

Testing Chatbots using Cucumber

ChatBot Example: Closer Look

- In the Python example we create a class as a singleton to represent the Web page hosting the chat bot

```
class WebPage:  
    instance = None  
  
    def __init__(self):  
        pass  
  
    @classmethod  
    def get_instance(cls):  
        if cls.instance is None:  
            cls.instance = WebPage()  
        return cls.instance
```

- [conygre] -
fill the gap

Testing Chatbots using Cucumber

Initialisation

- Our WebPage class has a function to initialise itself

```
def load_chatbot(self):
    chrome_options = Options()
    chrome_options.add_argument("--use-fake-ui-for-media-stream") # options turn off prompt for mic
    self.driver = webdriver.Chrome(chrome_options=chrome_options)
    self.website = self.driver.get('https://gamereviewstack-codebuilddeploy-xmvz-webappbucket-71jwxn1oy8i0.s3.amazonaws.com/index.html')
    self.number_of_messages = 1
    self.wait_for_next_message()
    self.text_input_box = self.driver.find_element_by_id('text-input')
```

- Initialise the driver (turn off the mic prompt)
- Set the Web page
- Set the initial number of messages (1) this is increased as we use the chatbot
- Wait for the initial message (see later)
- Locate the input text box

- [conygre] -
fill the gap

Testing Chatbots using Cucumber

The Step Flow Definitions

- The first definition is to initialise the bot and supply the first chat message to it

Given the user enters 'I would like to review a game'

```
@given(u'the user enters \'{intent}\')"  
def step_implementation(context, intent):  
    webpage.load_chatbot() # initialise the WebPage class  
    webpage.text_input_box.send_keys(intent)  
    webpage.text_input_box.send_keys(Keys.RETURN)  
    webpage.text_input_box.clear()
```

- [conygre] -
fill the gap

Testing Chatbots using Cucumber

Enter the Game and Platform

- Two similar step flow definitions are now used to enter the game and platform from the scenario

```
@when(u'asked for the game title the user enters '{game}'")  
def step_implementation(context, game):  
    webpage.wait_for_next_message()  
    webpage.text_input_box.send_keys(game)  
    webpage.text_input_box.send_keys(Keys.RETURN)  
    webpage.text_input_box.clear()
```

- [conygre] -
fill the gap

Testing Chatbots using Cucumber

Waiting for a Response

- A function has been added to get a response
 - The important bit is the CSS selector getting the next message from the bot

```
def wait_for_next_message(self):
    latest_message_from_bot = ""
    try:
        delay = 3
        css_selector = '.message-bot:nth-child(' + str(self.number_of_messages) + ')'
        WebDriverWait(self.driver,
delay).until(EC.presence_of_element_located((By.CSS_SELECTOR, css_selector)))
        time.sleep(2) # brief pause to allow the chatbot time to put some content in the field
        latest_message_element =
self.driver.find_element_by_css_selector(css_selector=css_selector)
        latest_message_from_bot = latest_message_element.text
        self.number_of_messages=self.number_of_messages+2 # 2 since every other is from human
    except TimeoutException:
        print("Loading took too much time!")
    return latest_message_from_bot
```

- [conygre] -
fill the gap

Testing Chatbots using Cucumber

Processing the Result

- The final step definition is to check if the final message from the bot matches the scenario

Then the chatbot should respond with 'According to metacritic, minecraft is rated: 93%'

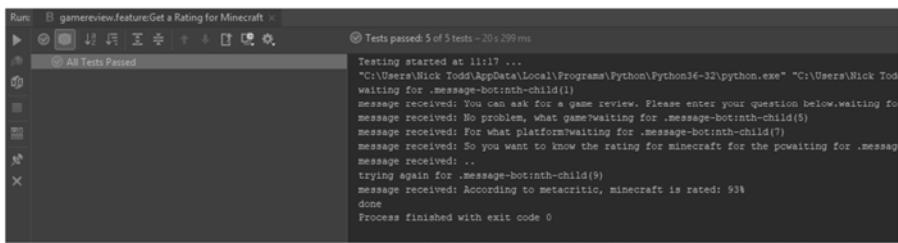
```
@then(u'the chatbot should respond with '{fulfillment}\")\ndef step_implementation(context, fulfillment):\n    ..\n        result_message = webpage.get_current_message_again()\n        assert_that(result_message, starts_with(fulfillment))\n\n\ndef get_current_message_again(self):\n    css_selector = '.message-bot:nth-child(' + str(self.number_of_messages-2) + ')\n    latest_message_element = \\\n        self.driver.find_element_by_css_selector(css_selector=css_selector)\n    latest_message_from_bot = latest_message_element.text\n    return latest_message_from_bot
```

- [conygre] -
fill the gap

Testing Chatbots using Cucumber

Running the Test

- Below is the test being run from IntelliJ



The screenshot shows the IntelliJ IDEA interface during a test run. The title bar says "Run: gamereview.feature:Get a Rating for Minecraft". The status bar at the top right indicates "Tests passed: 5 of 5 tests - 20s.299 ms". The main window displays the terminal output of the test execution. The output shows the bot interacting with the user to get a game review and rating. The terminal window has a dark background with light-colored text.

```
Testing started at 11:17 ...
"C:\Users\Nick Todd\AppData\Local\Programs\Python\Python36-32\python.exe" "C:\Users\Nick Todd\PycharmProjects\ChatBot\src\main\java\com\conygre\chatbot\GamereviewBot.java"
waiting for .message-bot:nth-child(1)
message received: You can ask for a game review. Please enter your question below.waiting for .message-bot:nth-child(5)
message received: No problem, what game?waiting for .message-bot:nth-child(5)
message received: For what platform?waiting for .message-bot:nth-child(7)
message received: So you want to know the rating for minecraft for the p
message received: ..
trying again for .message-bot:nth-child(9)
message received: According to metacritic, minecraft is rated: 95%
done
Process finished with exit code 0
```

- [conygre] -
fill the gap

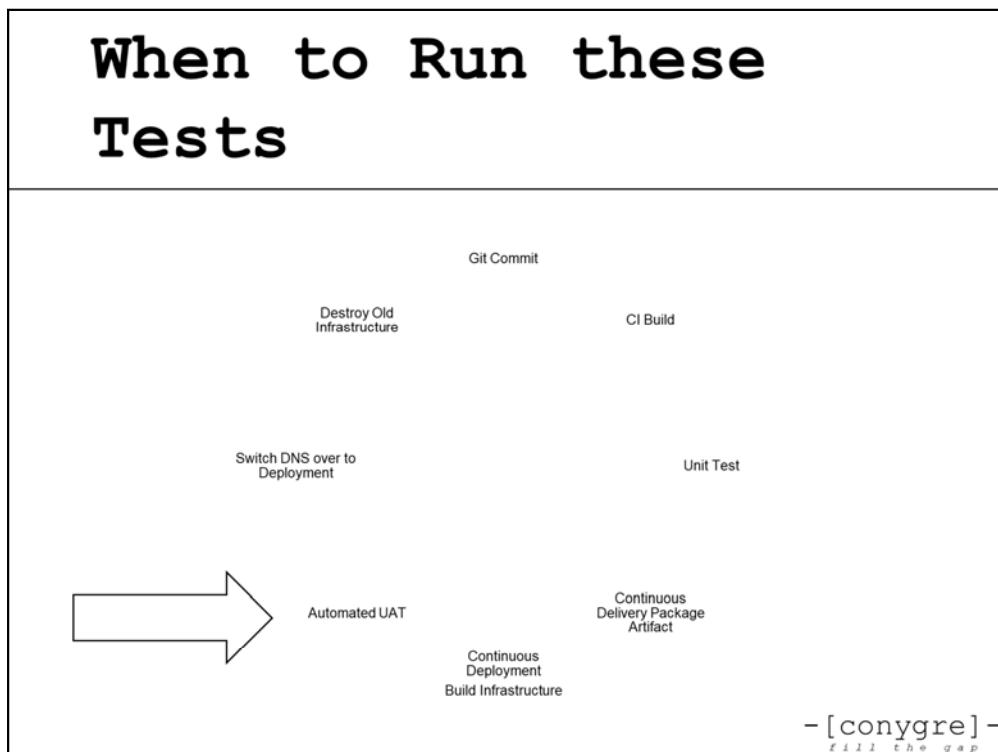
Testing Chatbots using Cucumber

When to Run Tests

- In a continuous delivery pipeline, these tests would be run before making any new deployment live

- [conygre] -
fill the gap

Testing Chatbots using Cucumber



Summary

- What is Behaviour Driven Development (BDD)
- The Benefits of BDD
- The Relationship between BDD and Other Agile Practices
- What is Cucumber?
- Defining Features and Scenarios
- Linking Features and Scenarios to Tests
- Testing a ChatBot Example
- When to Run Tests

- [conygre] -
f i l l _ t h e _ g a p

Amazon Lex

Course Labs Contents

Amazon Lex.....	1
Course Labs Contents	1
Chapter 1 Building a Basic Lex Bot	2
Aims.....	2
Part 1 Logging in to Amazon Web Services.....	2
Part 2 Creating the Lex Bot	2
Part 3 Creating your first Intent	2
Part 4 Creating the Slot Types	2
Part 5 Creating the Slots	3
.....	4
Part 6 Confirmation.....	4
Part 7 Fulfilment with Lambda.....	4
Part 8 Finishing off	5
Deployment to Web UI.....	6
Aims.....	6
Part 1 Launching the ChatBot.....	6
Part 2 Configuring the IAM roles for the servers	6
Part 3 Testing the Web UI	7
Appendix Writing a Lambda Function for Lex	9
Aims:.....	9
Part 1 Creating the function	9
Part 2 Writing the code	9

Building a Basic Lex Bot

Aims

In this exercise you will create a basic Amazon Lex bot to handle a simple request from a user. The context for this lab will be getting a quote for car insurance.

Part 1 Logging in to Amazon Web Services

1. Go to <http://aws.amazon.com>
2. Click the ‘Create an aws account’ button in the top right corner
3. Then click ‘Sign in to an existing AWS account’
4. Sign into AWS using the account credentials provided by the instructor

Part 2 Creating the Lex Bot

1. From the **AWS console** search for ‘Lex’ in the services tab
2. In the top right corner, ensure that your Region is set to **N. Virginia**.
3. Click the blue **Create** button to create your first Lex bot
4. Select **Custom Bot** and give your bot a name. Use your first and last name in the Bot name so it does not get confused with other peoples’ bots. (e.g. **JohnSmithInsuranceBot**)
5. You can also select an output voice if you would like your bot to use text-to-speech.
6. In the **Session Timeout** field, enter 5 minutes which will be plenty of time.
7. At the **COPPA** radio buttons select **No**.

Part 3 Creating your first Intent

1. Click the **Create Intent** Button
2. Remember that an **Intent** is the specific goal the user will want to achieve so name your intent **GetQuote**.
3. Add some **utterances** to the Intent in order to trigger it. E.g. ‘I would like to get a quote’. (Remember you can have multiple **utterances** which trigger the same intent.)



Part 4 Creating the Slot Types

We need to get some information from the user in order to fulfil the GetQuote Intent. Assuming we are talking about car insurance, we will need to know:

- a) The make of car
- b) The model of car

Conygre IT Limited - Amazon Lex

- c) The Engine size
- d) The Date of Birth of the User
- e) The year the user passed their test

Lex calls these pieces of information **SLOTS** which we need to fill.

We will first need to tell Lex the different types of information that the user could enter. We will need to create a **slot type** for each piece of information a-e.

For some of the slot types in this section, it will be easier to use the python script provided in order to generate files to upload to Lex to create the slot types for you. You don't *have* to use this script as it is possible to enter the information in manually but getting familiar with it will prove a time saver later on.

1. In the slot types tab on the left-hand side, click the blue + sign to add a type.
2. If you have used the python script, then you can click 'upload' and upload the zip file created by the script.
3. **Otherwise:**
 - a. Click **Create Slot Type** and give the slot type a name. The first of these will be **CarMakes**.
 - b. Add a description that briefly describes the slot. (e.g. **available car manufacturers**)
 - c. It is best to select this field to **Restrict to slot values and synonyms** in this case. This is to ensure that the user is only able to select from the car manufacturers that our insurance company is able to deal with.
 - d. Using a list from the internet or your own knowledge, go ahead and enter in all of the car manufacturers that you have patience to.

Note that for some of the Slot Types, amazon has built-in types. For example, amazon has a built in AMAZON.DATE type which we can use for the user's pass date and birthday.

4. Once you have created the **CarMakes** slot type, repeat the process for **CarModels, DOB, PassDate and EngineSize**.

Part 5 Creating the Slots

For each of the slots we defined we tell Lex how to prompt the user for that piece of information as well as what that information might look like.

1. In the Name section, type the name of the slot (This is case sensitive so try to stick to a convention) e.g. **CarMake**
2. In the slot type section, search for the slot type that fits the slot the best. For the **CarMake** slot this will be the **CarMakes** slot type you have created above.
3. In the Prompt section, type the question that Lex will ask the user, in order to get this piece of information. E.g. **What make of car do you drive?**
4. Do this for all of the pieces of information that you need for your bot and you should have something similar to below.

Conygre IT Limited - Amazon Lex

Priority	Required	Name	Slot type	Version	Prompt
		e.g. Location	e.g. AMAZON.US_CITY		e.g. What city?
1.	✓	CarMake	CarMakes	1	What make of car?
2.	✓	CarModels	CarModels	1	Okay, What model is that?
3.	✓	DateOfBirth	AMAZON.DATE	Built-in	In order for us to process your request, we ne
4.	✓	PassDate	AMAZON.DATE	Built-in	When did you pass your test?
5.	✓	EngineSize	AMAZON.NUMBER	Built-in	What size engine is your {CarMake} {CarMod

Part 6 Confirmation

It is good practice to ensure that the user knows that the bot has acquired all the information correctly. We can use the **confirmation prompt** tab to do this.

1. In the **Confirmation Prompt** tab, check the box to enable the confirmation prompt
2. In the confirm section we can write the confirmation message that will tell the client all of the information that the bot has acquired. We can link to the slots by using the following syntax: **{SLOT_NAME}** in the confirmation message.

Example: 'So you would like a quote for a {CarMake} {CarModel} {EngineSize} litre? And you were born on {DateOfBirth} and passed your test on {PassDate}?'

3. In the cancel section, we can put a simple message to confirm to the user that their request will not be processed. E.g. Okay, no problem. See you again soon.

Part 7 Fulfilment with Lambda

Once Lex has elicited all the information from the user, we are now in a position to **do something with it**. This is where lambda comes in.

1. Ensure you've saved your intent in Lex
2. In the **AWS Services**, Search for **Lambda**
3. Ensure you're in the same region as your Lex Bot (N. Virginia)
4. At the Lambda homepage, click **Create** and then select **Author from scratch**.
5. Enter the name **FirstNameLastNameInsuranceFunction** and select **Python 3** as the runtime.
6. Assign it a role of **lambda_basic_execution**.
7. In the **Function Code** section, from the dropdown **Code entry type**, select **Upload a .ZIP file** and go ahead and upload the **lambda.zip** file from the Lex Labs directory.



8. Go back to the **Lex** console and in the **Fulfillment** tab, select **AWS Lambda Function**.

9. Find your **lambda** in the drop-down menu and make sure it's set to the **latest** version.

10. You won't need to worry about the response tab in this lab as the response is handled by your lambda function.

Part 8 Finishing off

1. At the bottom of the screen, click **Save** to save your intent.
2. Click **Build** to build your Bot.
3. **Test it** using the testing panel on the right-hand side of the screen.

Deployment to Web UI

Aims

By the end of this exercise, you will have a fully-working chatbot deployed within a Web site. This will facilitate testing your chatbot programmatically as well as provide a way to implement your chatbot into your website of choice.

Part 1 Launching the ChatBot

Note: Additional documentation can be found here:

<https://aws.amazon.com/blogs/machine-learning/deploy-a-web-ui-for-your-chatbot/>

1. Follow the [link](#) to go to **CloudFormation** with the correct template already populated and click **Next**
2. Enter a name for your stack **InsuranceBotStack**
3. Enter a code build name **InsuranceBotCodeBuild**

IMPORTANT: DO NOT CHANGE THE BOOTSTRAP BUCKET OR BOOTSTRAP PREFIX as these refer to standard locations where boilerplate code is going to be copied from

5. Enter the **Bot Name** exactly as it appears in your Lex configuration.
6. Enter the **Cognito Identity Pool ID** if one is already set up.
(You can get this from Cognito / Manage Pools / Click on the graph of the pool already there and copy its id)
7. Set the **WebAppConfBotInitialText** to something appropriate. For Example:
'You can ask for an insurance Quote. Please enter your question below'
8. Set the **WebAppConfToolbarTitle** to 'Insurance Quote'.
9. Click **Next**.
10. At the **Options** dialog, click **Next**.
11. At the Review dialog, scroll to the end and tick the **I acknowledge that AWS CloudFormation might create IAM resources with custom names** option.
12. Click **Create**.
14. If successful you will see the following:

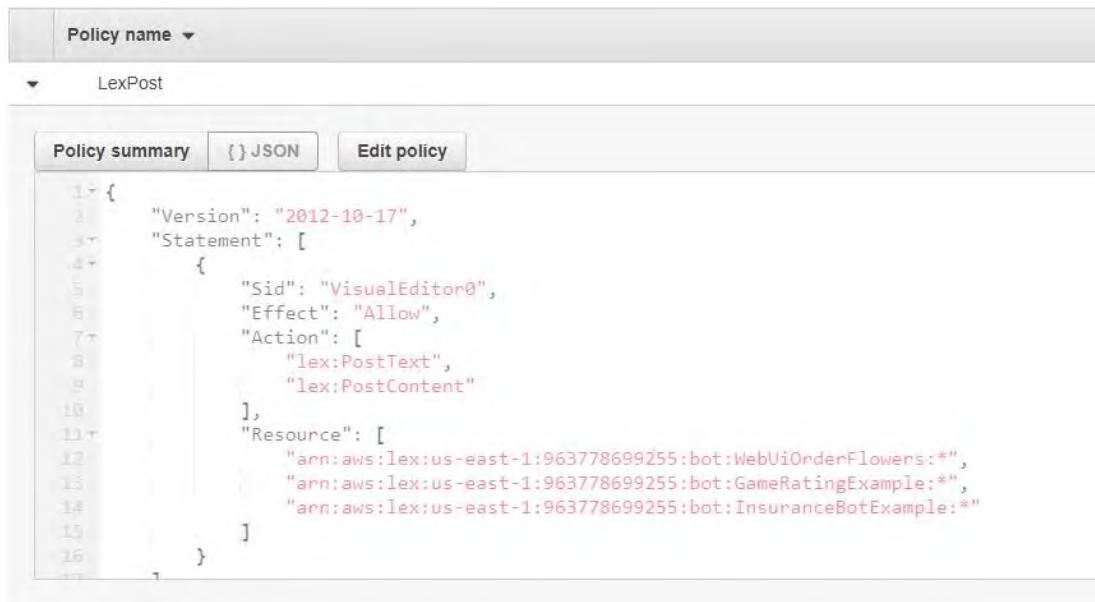
Stack Name	Created Time	Status	Description
InsuranceBotStack-CodeBuild...	2018-06-14 11:55:06 UTC+0100	CREATE_COMPLETE	This template creates a CodeBuild project used to configure and deploy the chatbot UI
InsuranceBotStack	2018-06-14 11:55:01 UTC+0100	CREATE_COMPLETE	Master Lex Web UI CloudFormation template. It deploys - S3 buckets to host the web app

Part 2 Configuring the IAM roles for the servers

ChatBot Training

Conygre IT Limited - Amazon Lex

1. To ensure that the role that the **Cognito unauthenticated user** is assigned allows access to your chatbot, click **Services**, and then type **IAM**, right click on the IAM link and choose **Open in a new tab**.
2. In the IAM console, click Roles, and then find and then select the role with a name like:
lex-web-ui-CognitoIdentityPool-1-CognitoUnauthRole-xxxxxxxxxxxx.
3. Expand the LexPost policy and then check that your LexBot is listed in the Allow section of the JSON



```
1  {
2      "Version": "2012-10-17",
3      "Statement": [
4          {
5              "Sid": "VisualEditor0",
6              "Effect": "Allow",
7              "Action": [
8                  "lex:PostText",
9                  "lex:PostContent"
10             ],
11             "Resource": [
12                 "arn:aws:lex:us-east-1:963778699255:bot:WebUiOrderFlowers:*",
13                 "arn:aws:lex:us-east-1:963778699255:bot:GameRatingExample:*",
14                 "arn:aws:lex:us-east-1:963778699255:bot:InsuranceBotExample:/*"
15             ]
16         }
17     ]
18 }
```

4. If it is not present, then click **Edit Policy**, and add into the list of resources, your Bots Amazon Resource Name (arn). It will be identical to the other present but with the different name on the end. Don't forget the :* which means any version of it.
5. **Save** the changes.

Part 3 Testing the Web UI

1. You are now ready to test your chatbot, go back to CloudFormation and click on the **InsuranceBotStack**, and then click **Outputs**

Outputs		
Key	Value	Description
ParentPageUrl	https://insurancebotstack-codebuilddeploy-1r-webappbucket-3yawgbv1ue3.s3.amazonaws.com/parent.html	URL of the iframe based sample web application. This page will be a...
SnippetUrl	https://insurancebotstack-codebuilddeploy-1r-webappbucket-3yawgbv1ue3.s3.amazonaws.com/iframe-snippet.html	URL of a page showing the snippet to load the chatbot UI as an iframe
CodeBuildUrl	https://console.aws.amazon.com/codebuild/home?region=us-east-1#/projects/InsuranceBotCodeBuild/view	Monitor the pipeline URL to see when the application has been fully ...
WebAppUrl	https://insurancebotstack-codebuilddeploy-1r-webappbucket-3yawgbv1ue3.s3.amazonaws.com/index.html	URL of the stand-alone sample web application. This page will be a...
LoaderScriptUrl	https://insurancebotstack-codebuilddeploy-1r-webappbucket-3yawgbv1ue3.s3.amazonaws.com/lex-web-ui-loader.min.js	URL of the loader script. This script will be available after the pipeli...

2. To do a basic test, click on the **WebAppUri** link to see your chatbot in action. You can try it out from here.
3. Finally, to test in your own Web page, click on the **SnippetUrl** link above which will show you the instructions for how to create your own Web page containing the chat bot.

ChatBot Training

Conygre IT Limited - Amazon Lex

4. Create a simple HTML file and add in the relevant snippet as per the instructions in the Web page.
5. Upload the page into the same S3 bucket as all the rest of your chat bot links are from and ensure that is set to **public visibility**.
6. Visit the page in a browser.

Appendix Writing a Lambda Function for Lex

Aims:

By the end of this lab you will have written a lambda function that will interface with the Lex bot you built in Chapter 1.

Part 1 Creating the function

This is the same as above:

2. Open Amazon Lambda from the AWS Console
3. Ensure that you're in the same region as your Lex Bot (N. Virginia)
4. Create a new lambda Function and choose 'Author from Scratch'
5. Give your Lambda a **name** and select the **Python 2.7/3** runtime (whichever you are most comfortable with)
6. Assign a suitable role- in this case probably (lambda_basic_execution) and click **create**

Part 2 Writing the code

This section will mostly be your **own code** written in **Python**, with only a small amount of guidance. You are free to either write your code in an **IDE of your choice** and upload it as a **.ZIP** file, or alternatively you can edit it from within the **built-in IDE** in amazon Lambda. Consider the following steps more as *hints* than as a step-by-step guide.

1. The function that will be called within your code is the **lambda_handler** function which takes in two parameters: an **event** and a **context**. Make sure you have this function.
2. The event that the **lambda_handler** is passes is in the following **JSON** format

```
{
  "currentIntent": {
    "slots": {
      "CarMake": "bmw",
      "CarModel": "m5",
      "DateOfBirth": "1975-5-16",
      "PassDate": "1997-8-12",
      "EngineSize": "5.0"
    },
    "name": "GetQuote",
    "confirmationStatus": "None"
  },
  "bot": {
    "alias": null,
    "version": "$LATEST",
    "name": "JohnSmithInsuranceBot"
  },
  "userId": "bijt6rovckwecnzesbthrr1d7lv3ja3n",
  "invocationSource": "DialogCodeHook",
  "outputDialogMode": "Text",
  "messageVersion": "1.0",
  "sessionAttributes": {}
}
```

ions

3. Lex will receive back the response from the **Lambda_handler** function so you must ensure that it has the appropriate **JSON** format in order to **fulfill** the **intent**.
4. This format is best achieved using python **dict** notation (see below)

```
response = {  
    "dialogAction":  
    {  
        "fulfillmentState":fulfillmentType,  
        "type":"Close","message":  
        {  
            "contentType":"PlainText",  
            "content": result  
        }  
    }  
}
```

ChatBot Training

Conygre IT Limited - Amazon Lex