Slide 1

Today's Agenda

- Bot Framework Overview
- Bot Framework Developer Introduction and Deploying an Intelligent Bot
- Deep Dive into the Microsoft Bot Framework

Be ready for fun labs throughout ☺

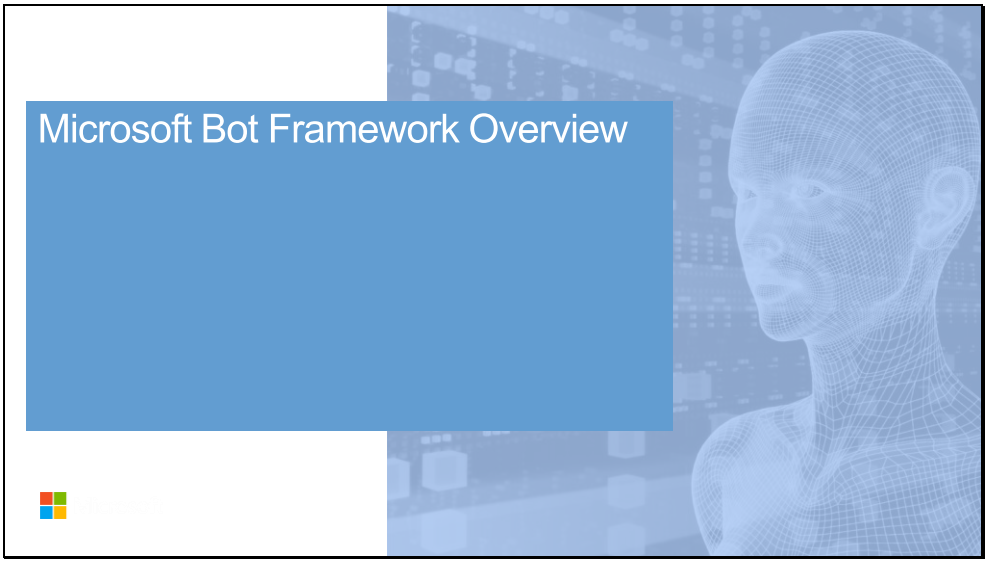
Welcome to Developing and Deploying Intelligent Chat Bots

Day 2

Slide 2

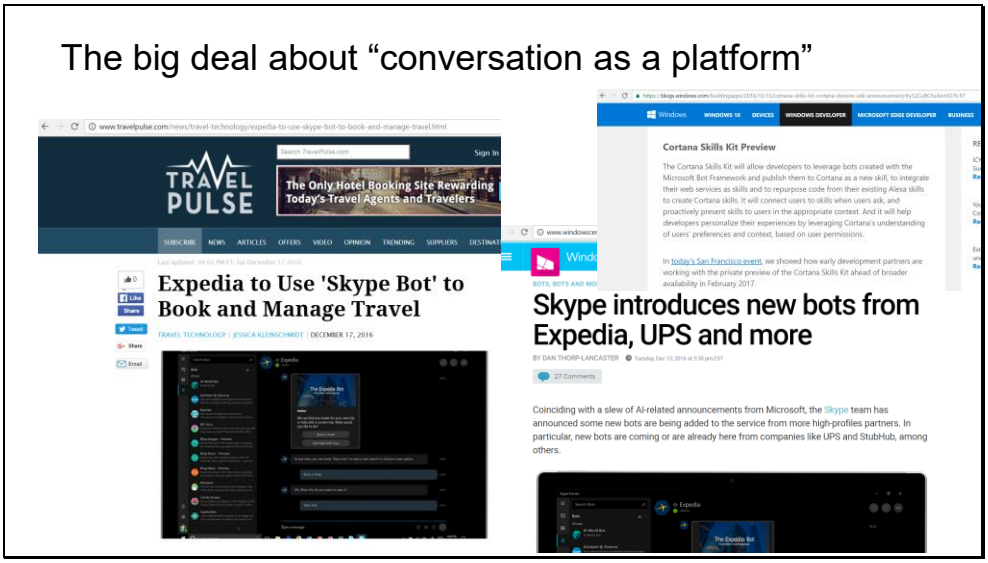
This link contains additional resources on the bot framework and related topics. mailto: michhar<at>microsoft.com for questions/comments. Show sites

Slide 3



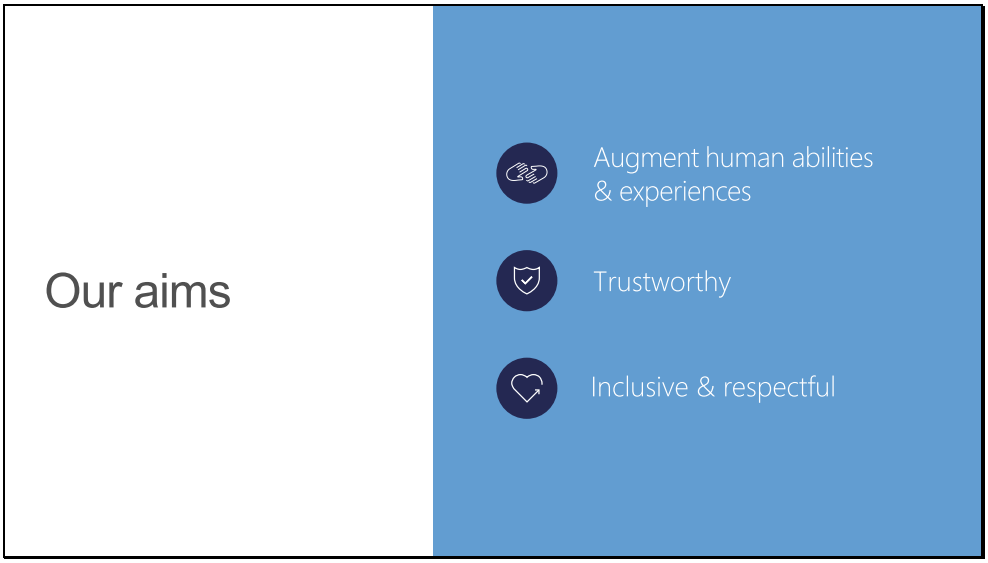
If the links in this deck are broken please let us know (mailto: michhar<at>Microsoft.com). Thanks in advance and enjoy learning about bots and the Microsoft Bot Framework.

Slide 4

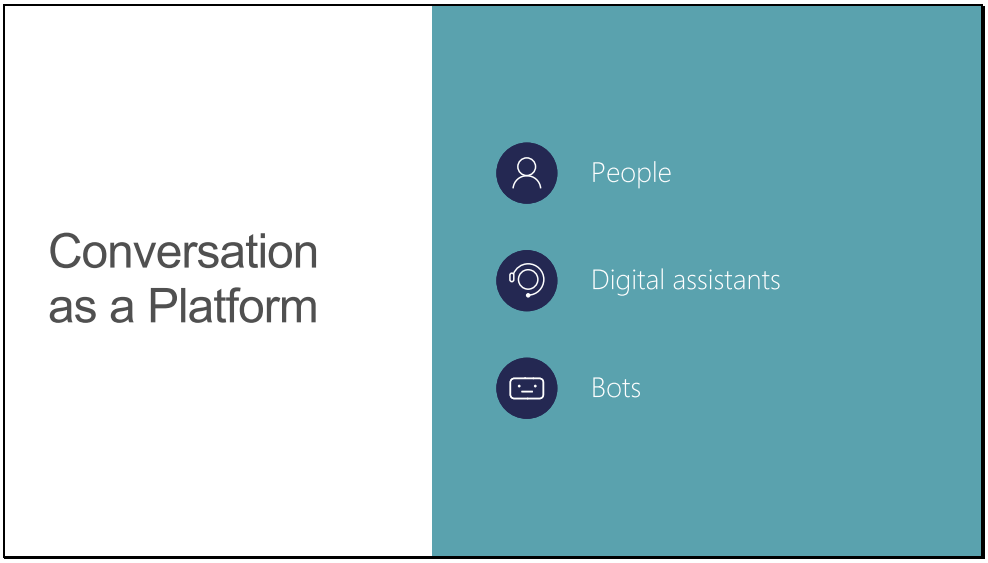


http://www.windowscentral.com/skype-introduces-new-bots-expedia-ups-and-more
http://www.travelpulse.com/news/travel-technology/expedia-to-use-skype-bot-to-book-and-manage-travel.html
https://blogs.windows.com/buildingapps/2016/12/13/cortana-skills-kit-cortana-devices-sdk-announcement

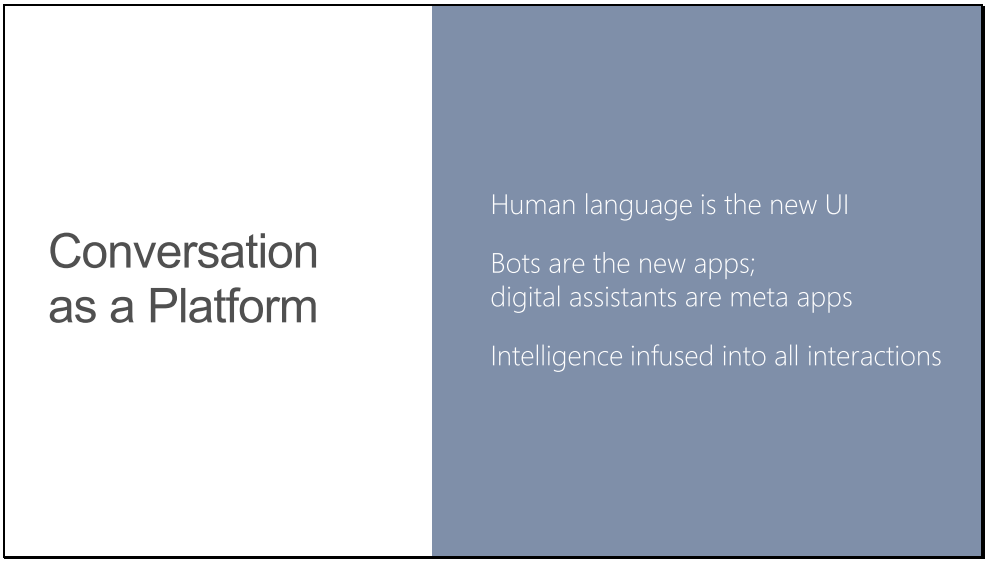
Slide 5


Our aims
Augment human abilities & experiences
Trustworthy
Inclusive & respectful

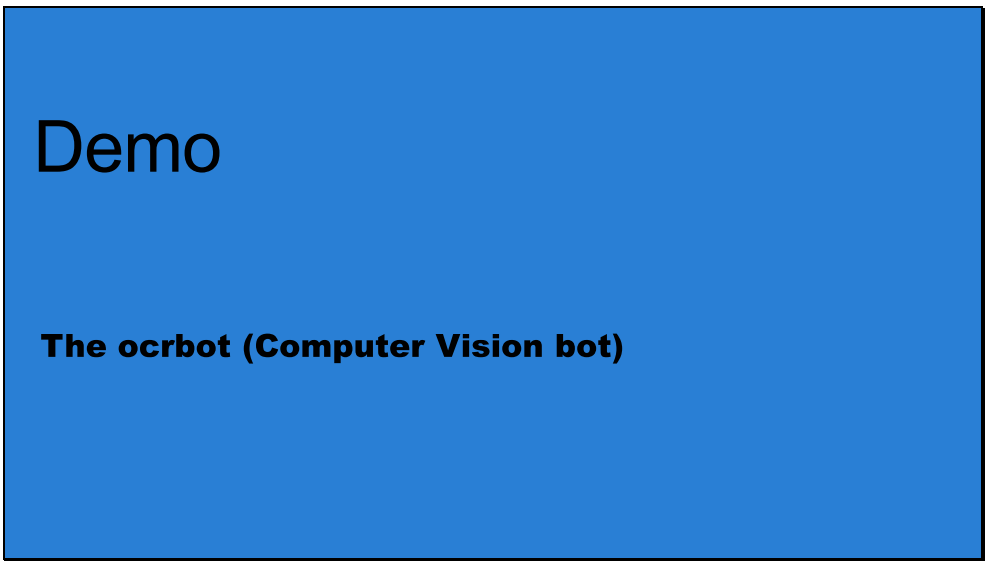
Slide 6


Conversation as a Platform
People
Digital assistants
Bots

Slide 7


Conversation
as a Platform
Human language is the new UI
Bots are the new apps;
digital assistants are meta apps
Intelligence infused into all interactions

Slide 8

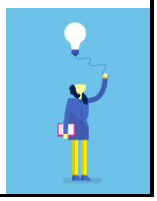


Launch the ocrbot from the developer portal using Skype

Slide 9

Session outline

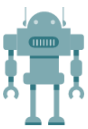
- What **a bot is** and is **not**
- The **major components** of the Bot Framework
- Deploying and working with **channels**
- Your **arsenal** or **toolbox**
- *Dev environment lab*



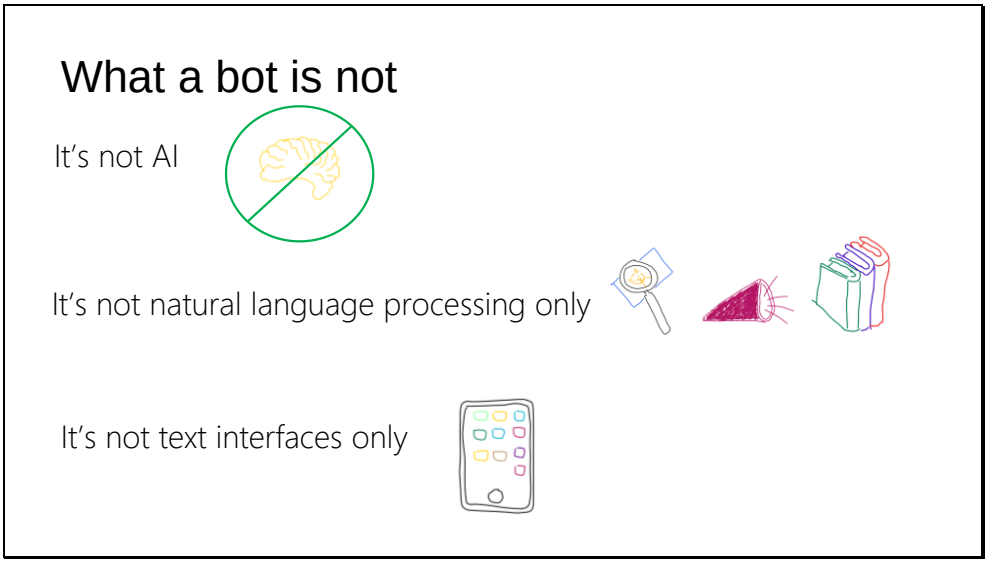
Learning objectives for this overview module on the Bot Framework

Slide 10

What is a bot?



Slide 11



Not AI:

- Bots can be simple task automation utilities.
- Example: Password reset bot. There's no AI here. Just ask a couple of security validation questions, then reset the password.
- They may have AI as well, if the scenario applies

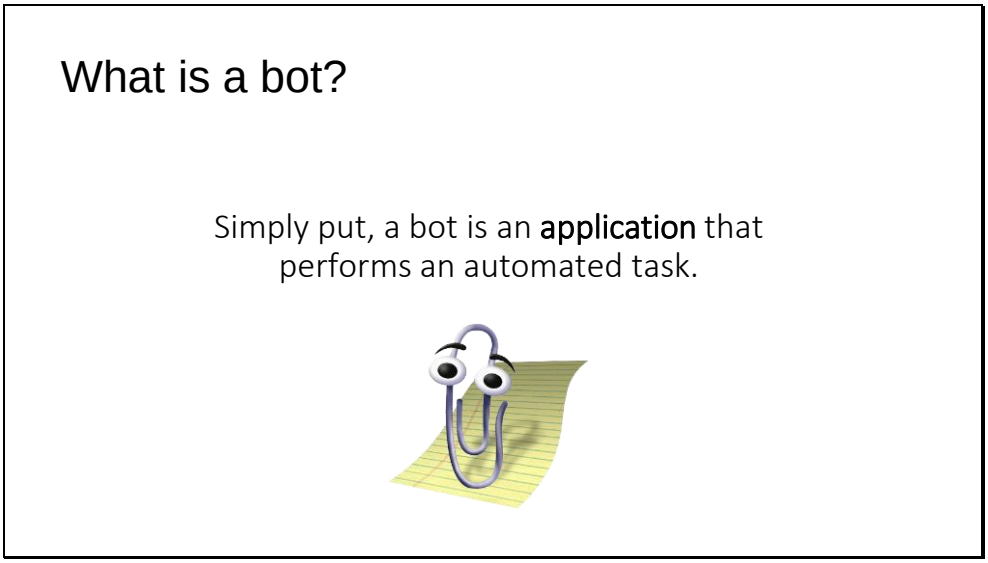
Not only NLP:

- Natural language processing has limitations, still. The more your bot depends on NLP, the worse the experience gets. Hint: Typing isn't always the best option.
- Move away from NLP as quickly as possible
- "Drive" the user as much as you can (menus, choices, etc). Less typing = better

Not only text interfaces:

- Bot channels are evolving quickly to support richer experiences: Media, buttons, custom controls. These are here or on their way. Text is not known to the best experience for everything.
- Examples:
  - Skype allows audio and 3D bots as well.
  - Slack, Facebook and Skype have buttons/custom UIs

Slide 12



Siri, Cortana, the old-school MS Clippy and even AOL's SmarterChild are some examples. Essentially, bots perform automated tasks that are generally REPETITIVE for humans to do. We want to make life easier for the end user of the bot.

Bots are apps.

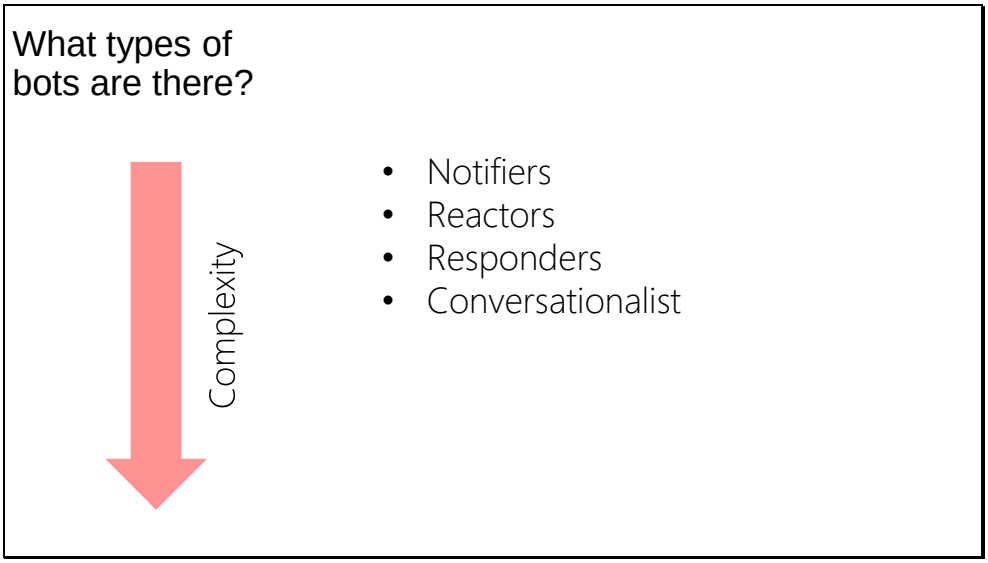
They can:

- Exist in different channels and across platforms.
- Do anything from simple task automation like taking food orders to leveraging sophisticated deep learning algos as is used by CaptionBot (https://www.captionbot.ai/ which describes the contents of an image how a human would) and otherAI-esque capabilities.

What a bot can do is only limited to the APIs your bot uses.

Bots don't have to leverage the MS Bot Framework (e.g. MimikerAlarm https://www.microsoft.com/cognitive-services/en-us/mimickeralarm, an app for waking you up), but the Framework makes dev and deploy much simpler and faster for.

Slide 13

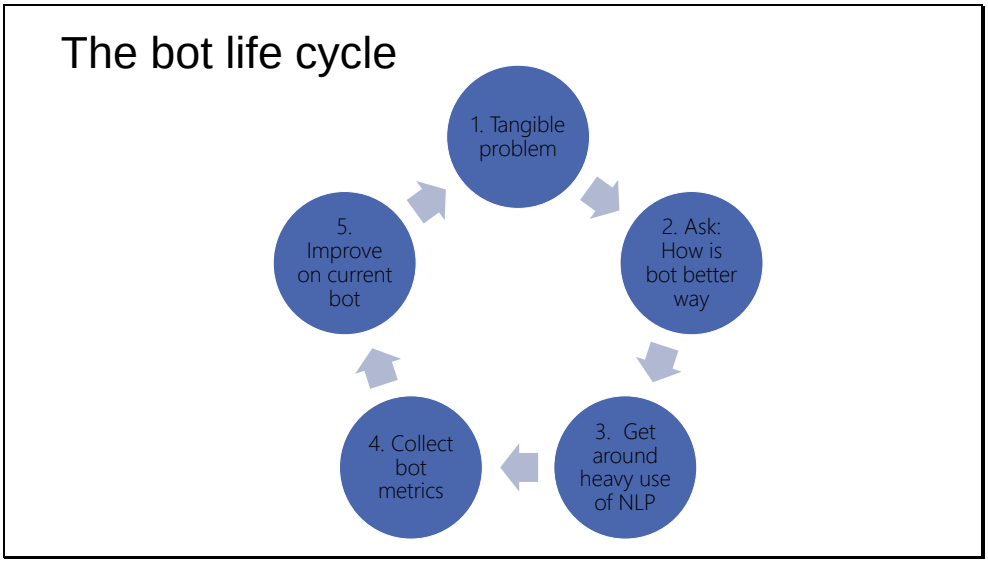


Based on this blog post: http://willschenk.com/bot-design-patterns/?imm_mid=0e50a2&cmp=em-data-na-na-newsltr_20160622 about different bot types and the definitions of these.

- Notifier - simply broadcast messages aka push bot e.g. ping me when there's a interesting tweet about Hadley Wickam
- Reactor - reacts to messages on service, but doesn't persist anything (message, user state, location) e.g. send me the stock price for a stock I specify, but don't remember me or what I say
- Responder - reacts to messages on service, persists message and knows who I am e.g. send me today's weather forecast for my city, use my user name on this channel, and remember what cities I choose
- Conversationalist – reacts to messages, persists messages, knows who I am, knows about the "place" I'm at (channel, room,...), knows the state of the conversation e.g. send me today's weather forecast for a city, use my user name on this channel, remember what cities I choose, format it nicely for this channel, pick up where I left off if I go away for a minute, and if the conversation is very old, archive it and send as email.

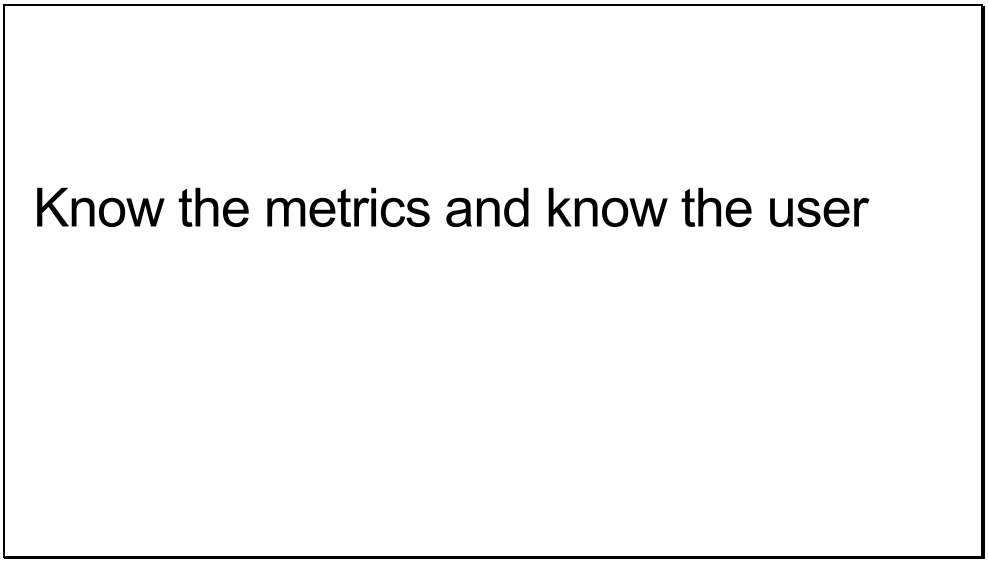
From "Bot Design Patterns": Questions that help us formulate what kind of bot we might want or need:
Do they react to messages?
Do they know who they are talking to?
Can they learn from what was said?
Do they know where the conversation is taking place?
Do they remember the overall conversation?

Slide 14



- Start by asking what problem are we trying to solve. Refine until it looks like a tangible problem and not "magic"
- Ask how a bot will be a better experience. User experience is EVERYTHING
- Avoid too much natural language. Careful with unrealistic expectations. Natural language recognition is limited. Menus work great. Commands work great. Buttons, etc.
- You can only analyze and improve your bot if you're collecting metrics for it
- Iterate, improve

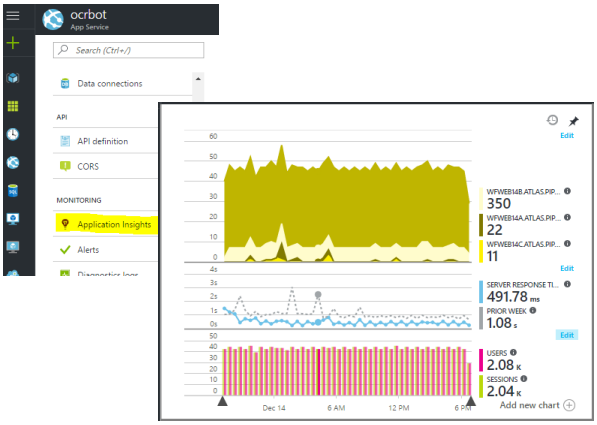
Slide 15



Slide 16

Add Application Insights

In the Azure portal, activate **Application Insights** for your bot's web service

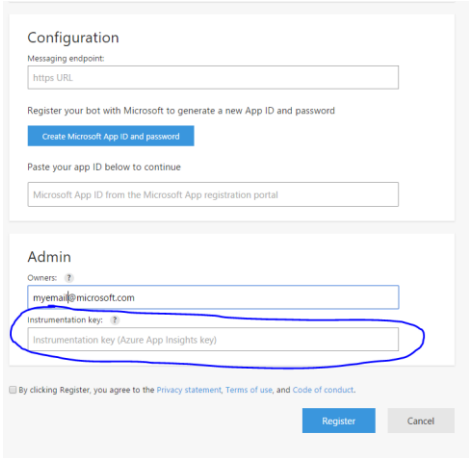


Slide 17

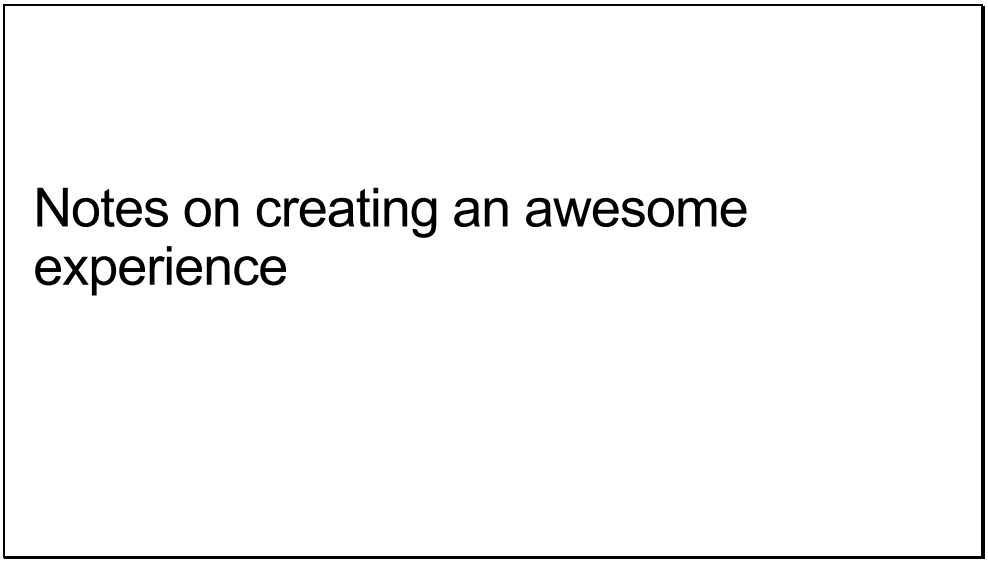
Connect App Insights to the Bot Framework

In the Bot Framework Developer's portal, plug in your key from App Insights

It's that easy to collect metrics!



Slide 18



Slide 19

From: https://docs.botframework.com/en-us/directory/best-practices

This may have been in your mind before this tutorial. Ethical and societal considerations as well in an article by Satya Nadella: https://www.linkedin.com/pulse/partnership-future-how-humans-ai-can-work-together-solve-nadella

Slide 20

Creating an awesome experience: the AI

Key considerations around AI and design today:

Outcomes are determined as much by the human element as by the software element.

The quality of the user experience determines both the usefulness of the product and its rate of adoption, and this is why [it is believed] design is the next frontier of AI.

What does this mean?

Credit for paraphrased quotes Manoj Saxena: https://www.fastcodesign.com/3068005/whats-still-missing-from-the-ai-revolution

Slide 21

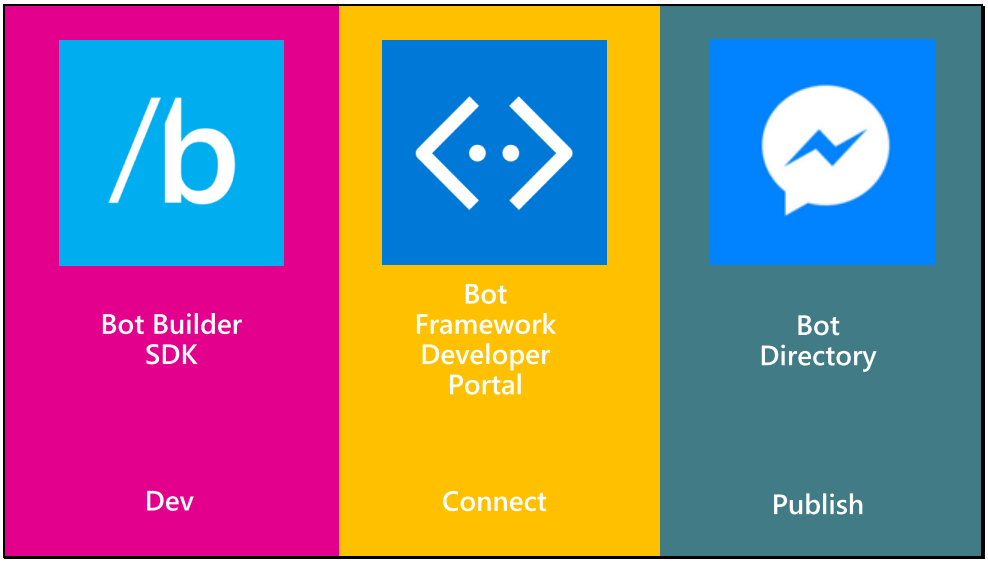
Here enters the Bot Framework

Slide 22

What is the Bot Framework

A development tool and much more…

Slide 23



Bot Builder is itself a framework for building conversational applications ("Bots").

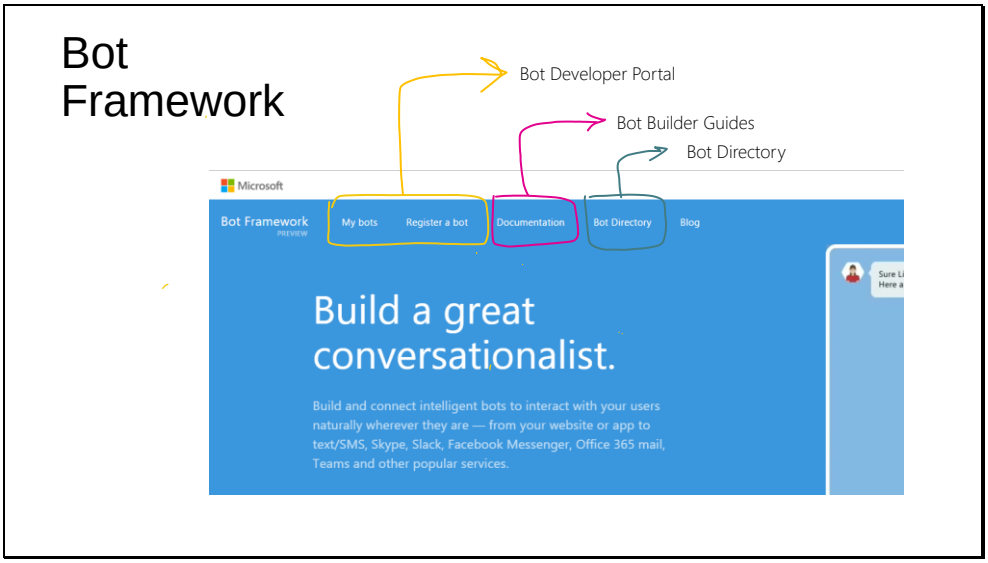
The Bot Builder SDK is an open source SDK hosted on GitHub that provides everything you need to build great dialogs within your Node.js-, .NET- or REST API-based bot.

The Bot Framework Developer Portal lets you connect your bot(s) seamlessly text/sms to Skype, Slack, Facebook Messenger, Kik, Office 365 mail and other popular services. Register, configure and publish.

The Bot Directory is a public directory of all reviewed bots registered through the Developer Portal.

NB: Bot builder and bot connector SDK now one in V3 of framework: http://docs.botframework.com/en-us/support/upgrade-to-v3/#botbuilder-and-connector-are-now-one-sdk

Slide 24



Main page: https://botframework.com

Slide 25

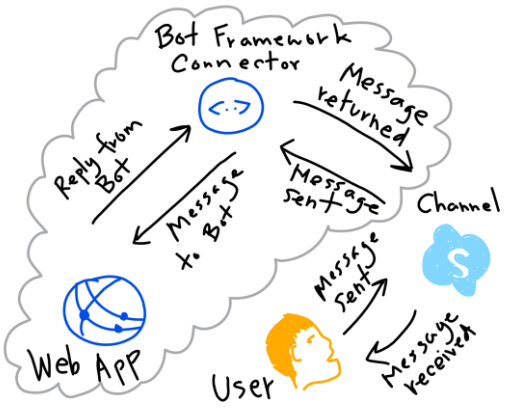
What is the Bot Framework

It’s also a messaging platform

Let’s get a picture of the way it works

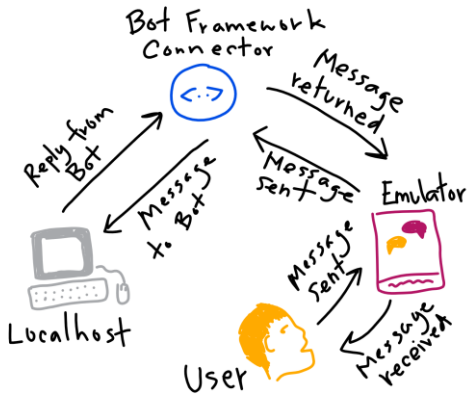
Slide 26

What is the Bot Framework: as a cloud solution



Slide 27

What is the Bot Framework: as a development solution



Slide 28

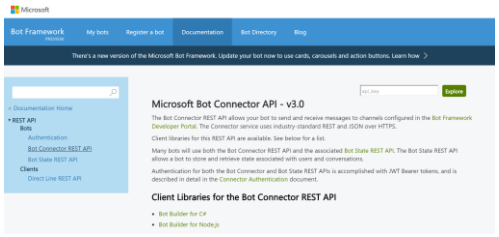
Bot Builder: Development Kits and REST

Bot Builder SDKs for:

- .NET framework for C#
- Node.js

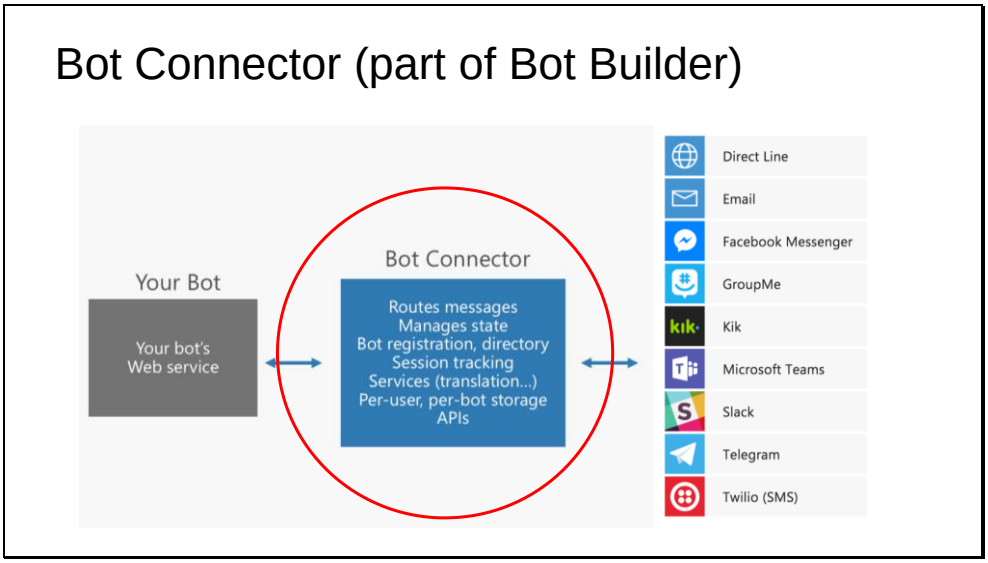
And there's also

- REST and REST State APIs



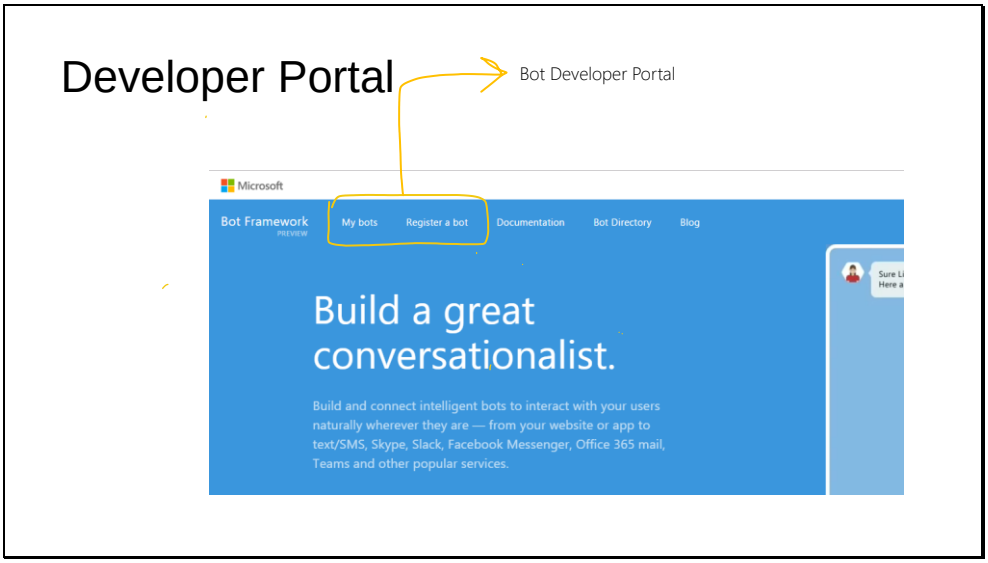
SDKs infographic: http://docs.botframework.com/en-us/images/faq-overview/bot_builder_sdk_july.png

Slide 29

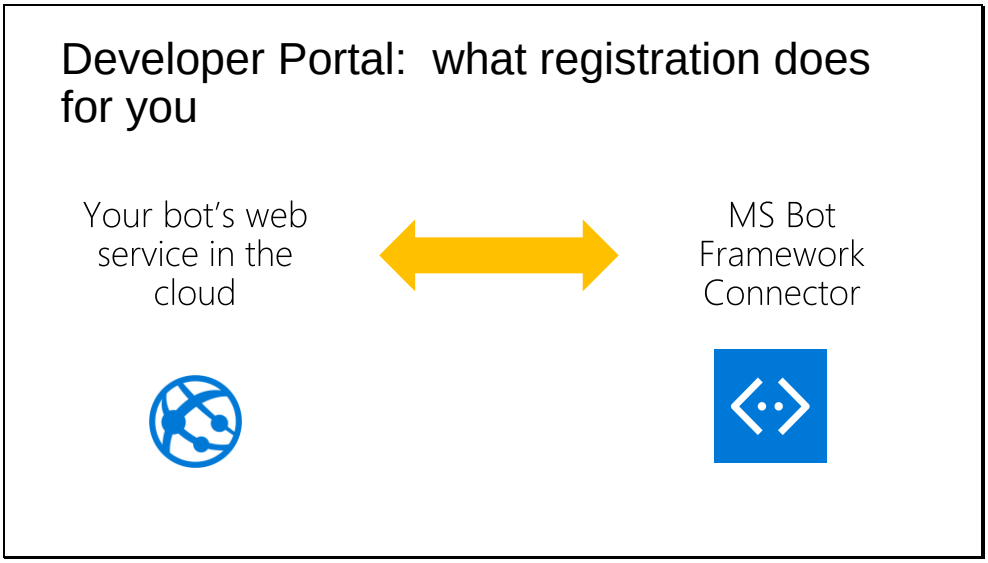


It's part of the Bot Builder SDK
http://docs.botframework.com/en-us/csharp/builder/sdkreference/gettingstarted.html#channels

Slide 30

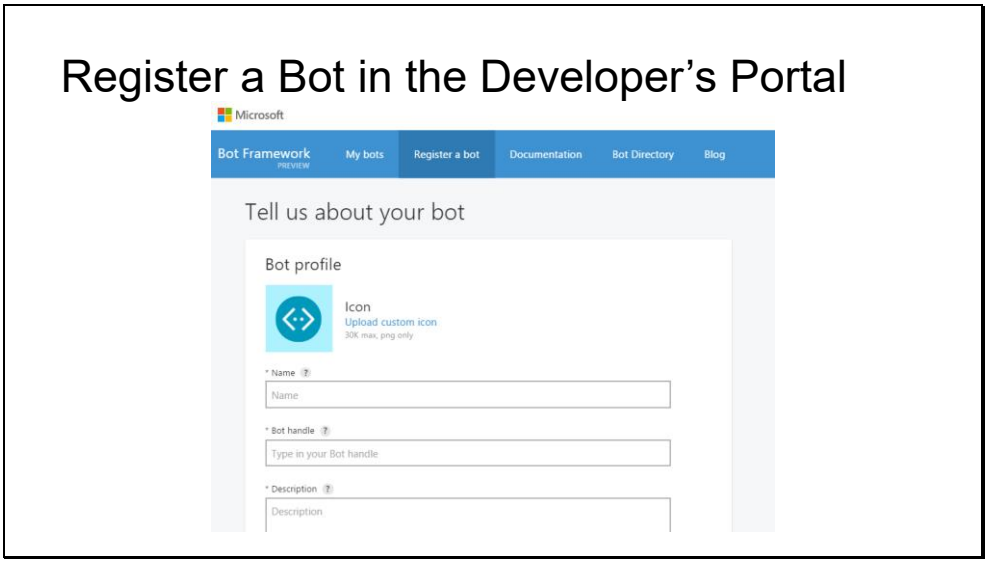

Developer Portal
Bot Developer Portal
Microsoft
Bot Framework
PREVIEW
My bots
Register a bot
Documentation
Bot Directory
Blog
Build a great conversationalist.
Build and connect intelligent bots to interact with your users naturally wherever they are — from your website or app to text/SMS, Skype, Slack, Facebook Messenger, Office 365 mail, Teams and other popular services.

Slide 31



Expose a Microsoft Bot Framework-compatible API on the Internet, then the Bot Framework Connector service will forward messages from your Bot to a user, and will send user messages back to your Bot.

Slide 32



Register on the developer portal by clicking the 'Register a bot' link: https://dev.botframework.com/bots/new

Name: TemplateBot
Bot Handle: templatebot (for referencing in Bot Directory and name for bot on web chat, NOT the app's URL used as endpoint)
Also, add a description here

Configuration

Remember the URL endpoint from deploying endpoint step. Should be something like:
"https://*botwebappname.azurewebsites.net*/api/messages"
You'll go through the "Generate App ID and password" wizard, then return to the registration page.

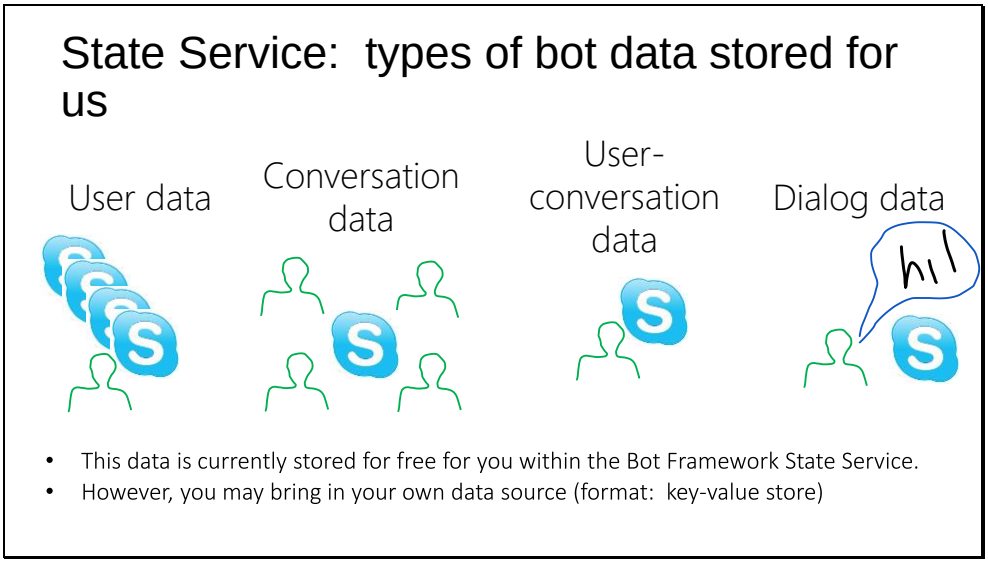
Go back and edit this profile anytime

What about all of the message data?

The Bot Framework manages the conversation, user and dialog data

This data is called State and the manager is the State Service

Slide 34



User data – globally available for user across all conversations

conversation data – stores globally for a single conversation (many users could be involved)

User-conversation data – stores globally conversation data for a user (But private to just that user)

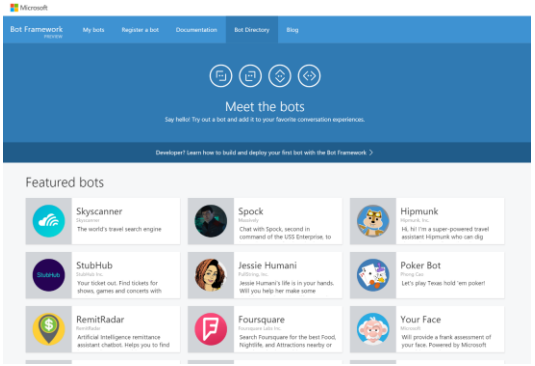
Dialog data as well – persists for a single dialog (helpful for temp data in a waterfall set of steps)

Slide 35

Bot Directory

Public Directory of Bot Framework Bots

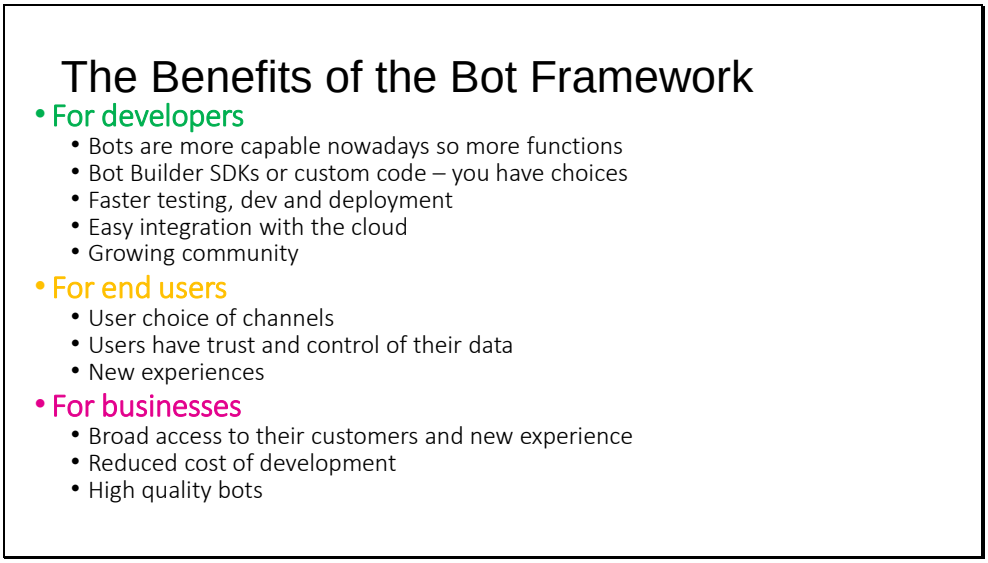
- Discover, try, and add bots from here with no added configuration
- Bots are public at developer discretion; must be reviewed
- Searchable here



Bots must be submitted for review and approved in order to appear in the directory

As of Nov. 17, 2016 there are 94 bots in the directory from MS and other companies

Slide 36



For developers

Bots are more capable because of supporting services i.e. MS Cognitive Services and BF State Service

Bring your own bot or build your own bot with the Bot Builder SDKs

With SDKs and sample code on github

Smooth cloud deployment and integration

Big community (open source community, issues, gitters, stackoverflow – active and responsive)

For end users

Users can choose from a variety of conversation channels

Users have trust and control of their data – encrypted and accessible anytime

Frictionless fun or at least makes something much easier

For businesses

Broad access to their customers where they already are conversing giving them new experiences

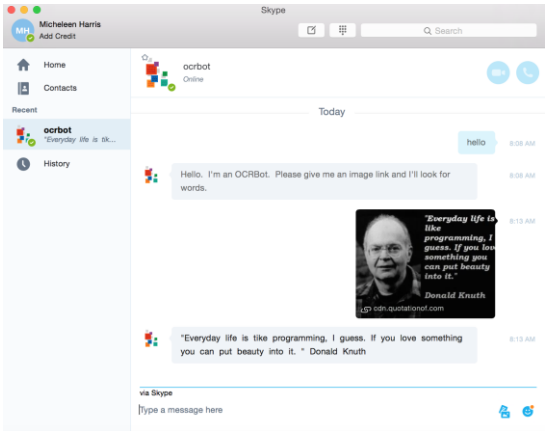
Reduced cost of development – just faster with SDKs and builtin functionality like dialog handling and language understanding

High quality bots (big support and dev community) as well as, bots are reviewed after publishing and surfaced on Bot Directory

Working with channels

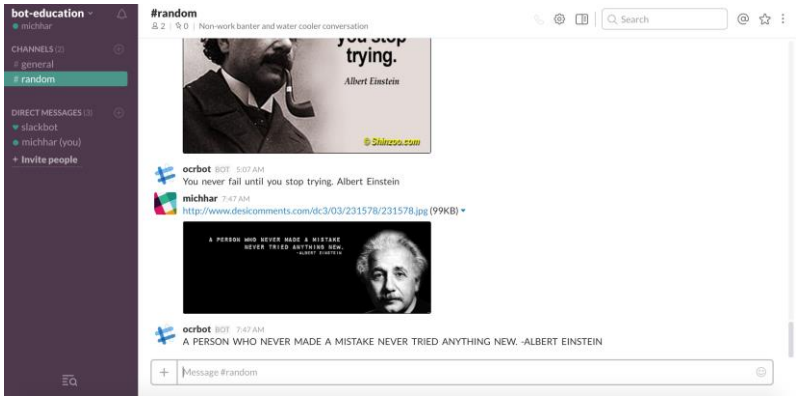
Slide 38

Skype channel example

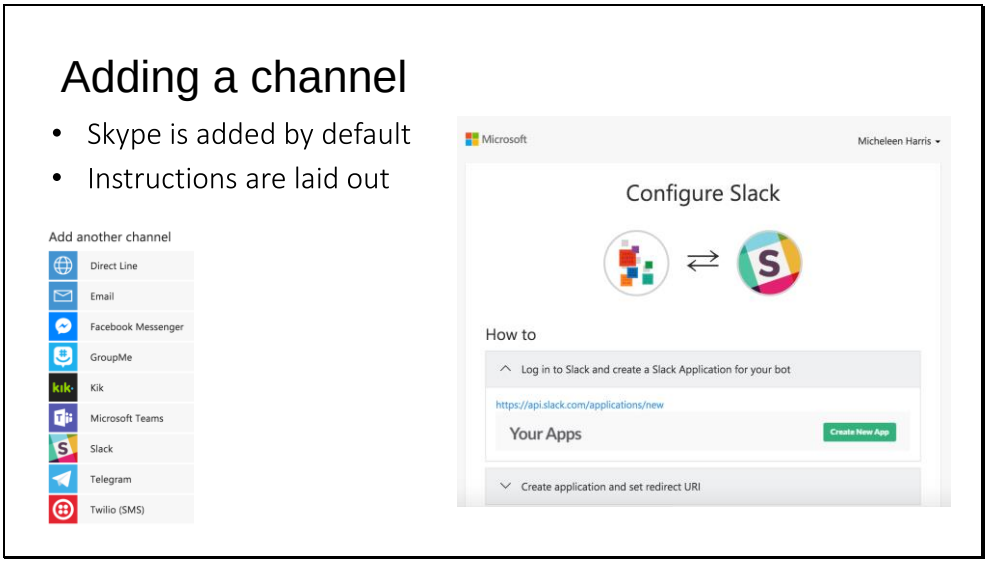


Slide 39

Slack channel example



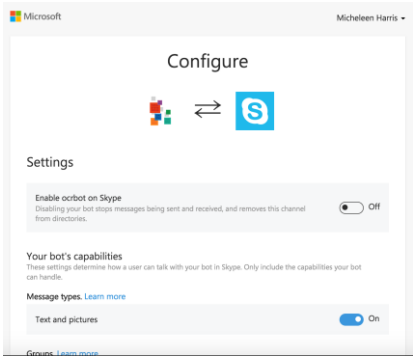
Slide 40



Often, the most time will be spent configuring your credentials as a developer on the target service, registering your app, and getting a set of Oauth keys that Microsoft Bot Framework can use on your behalf

Slide 41

Editing a channel



Skype for instance, through configuring we can:

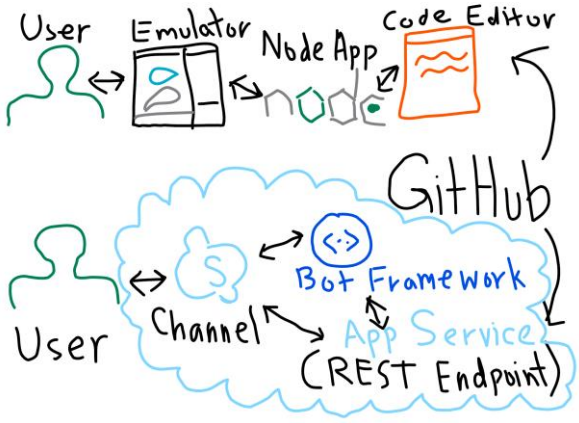
- Disable/enable globally
- Turn on/off group messaging
- and more

Slide 42

Let's see some tech stuff and code!

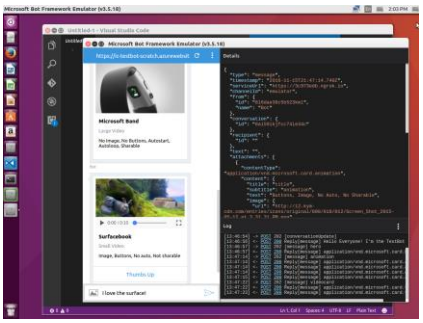
Slide 43

Dev process



Slide 44

The BF Emulator – New Unix-Compatibility



- **New** Support for Mac, Linux and Windows
- **New** All the Bot Framework card types are supported
- **New** Save multiple profiles for when you're working online and off
- **New** Simplifies configuration when you're working with ngrok
- **New** Uses the webchat control for higher fidelity layout and consistency with the webchat experience

Emulator purpose:
Send requests and receive responses to/from your bot endpoint on localhost
Inspect the Json response
Emulate a specific user and/or conversation

Slide 45

Node App: Bot Builder Node.js SDK

Get the package as easy as this one line in the terminal:

```
npm install --save botbuilder
```

Or to clone the latest entire repository (also where the SDKs are):

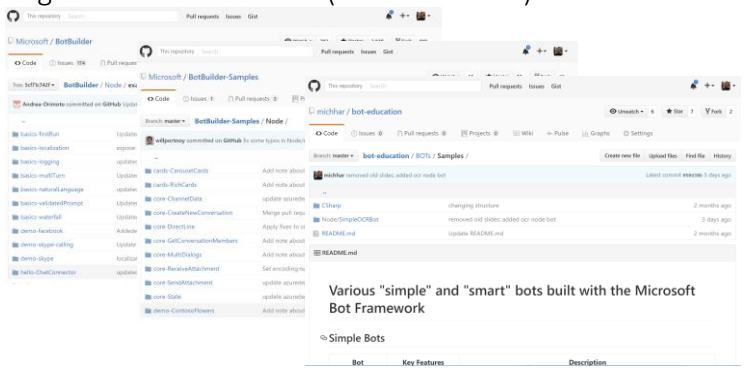
```
git clone https://github.com/Microsoft/BotBuilder.git
cd BotBuilder/Node
npm install
```

We will set up after the Cognitive Services Overview
https://docs.botframework.com/en-us/node/builder/guides/core-concepts

Slide 46

Tons of sample code on GitHub

On github from Microsoft (and elsewhere):



Go to repository link and in Git Bash type: **`git clone <http cloning link from page>`**

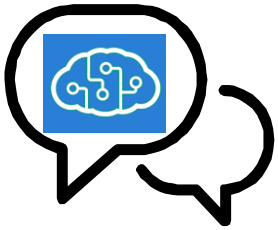
https://github.com/Microsoft/BotBuilder-Samples
https://github.com/Microsoft/BotBuilder/tree/master/CSharp/Samples
https://github.com/Microsoft/BotBuilder/tree/master/Node/examples
https://github.com/Azure/bot-education/tree/master/Student-Resources/BOTs
https://github.com/MicrosoftDX/botFramework-proactiveMessages

And many more to be found

More out there…so many, can't list…

Slide 47

Integration with cognitive services



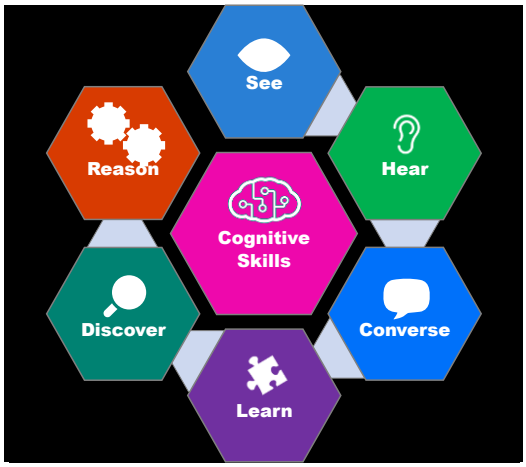
Slide 48

Cognitive Services meet Bots

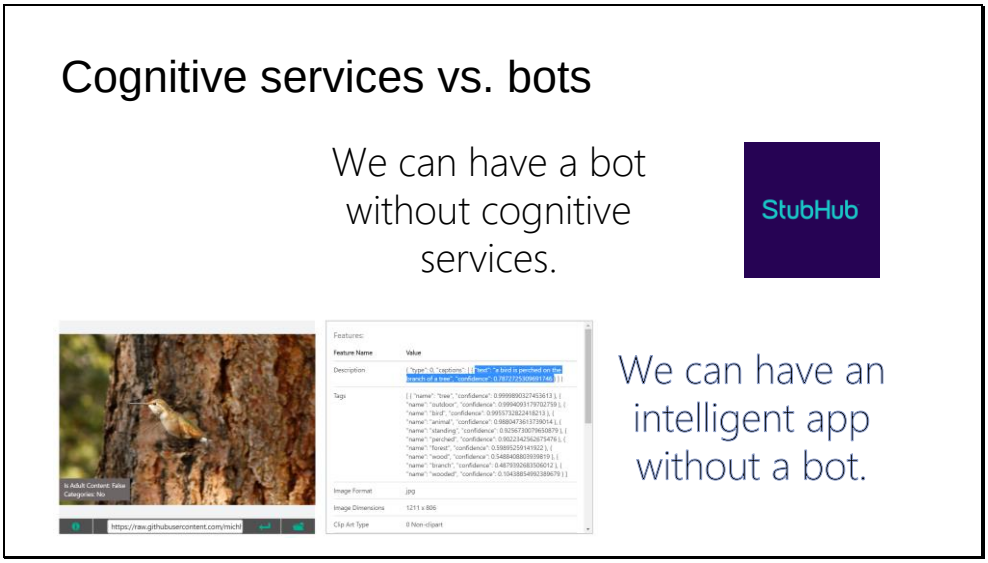
We'll learn how to add Cognitive Skills to your Bots

GOAL

Build smarter experiences that delight and engage your users



Slide 49

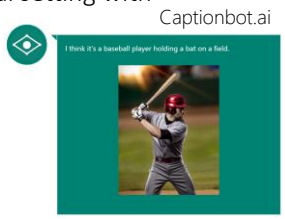


Stubhub: Find tickets for shows, games and concerts with StubHub.
Computer vision API demo on this page: https://www.microsoft.com/cognitive-services/en-us/computer-vision-api

Slide 50

A few use cases

1. Create human readable captions for the content of an image
2. Authenticate users using a voiceprint
3. Recognize the intent of a user
4. Recommend products frequently bought together
5. Ease the burden of typing queries in a conversational setting with autosuggest



- Computer Vision API - Like CaptionBot.ai – a bot that reports back in human way the contents of an image
- Speech API - Authentication as a user speaks to the bot with a speaker verification profile or "voiceprint"
- LUIS API for analysis of queries such as "What is the weather in Toyko today?" using entities to parse out intent (what the user is asking for)
- Knowledge API. Recommendations based on our knowledge and/or a user's history. Also, could search through a graph or user-defined database to return relevant academic papers from a natural language query
- Bing Search API for autosuggest - Search also has other capabilities such as returning the latest trending news on a topic for example

Slide 51

More complex examples

1. Ensuring proper advice and experience for customers of wealth and investment management
2. Monitoring potential changes to relevant laws and alerting lawyers when changes occur
3. Presenting the right products to customers at the most appropriate point in the buying cycle
4. Providing trip-planning advice that travelers find relevant

Would you build a bot or not for these?

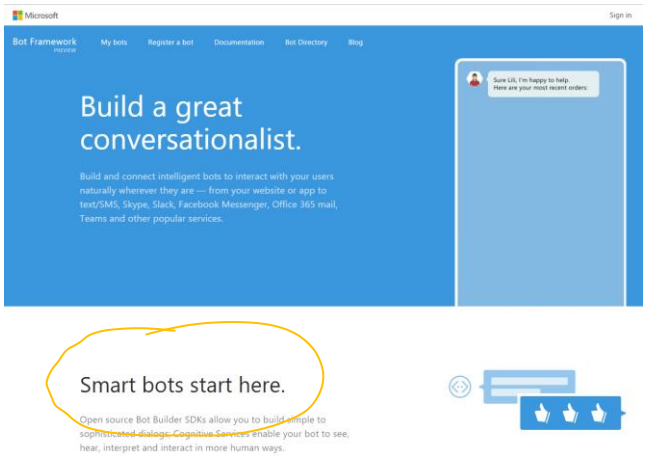
Credit: Post by Manoj Saxena at https://www.fastcodesign.com/3068005/whats-still-missing-from-the-ai-revolution

Slide 52

Bot Framework

Add Bot intelligence

Get the info in the Bot Framework docs



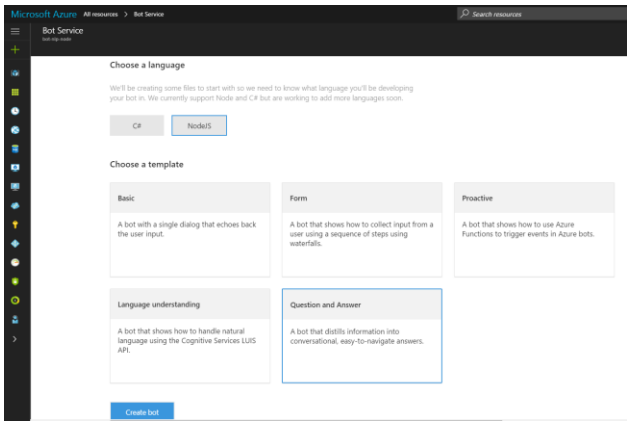
Landing page for the bot framework: https://botframework.com
Excellent Documentation: https://docs.botframework.com

Slide 53

Azure Bot Service

Bot Framework

From Azure Bot Service:
- Fire up LUIS bot
- Fire up QnA bot
- more templates…

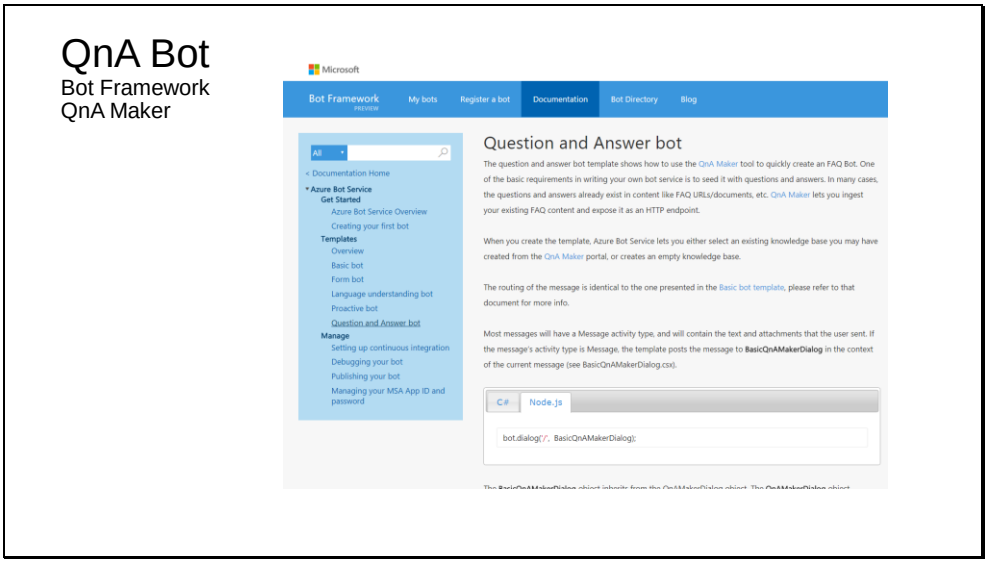


Some instructions for creating your first bot with Azure Bot Service: https://docs.botframework.com/en-us/azure-bots/build/first-bot/#navtitle

This is very easy, but has many limitations.
However, you can download the code for this Bot and create a more advanced solution (recommended)

Done through the Azure Portal
Options as of now for templates are:
- Basic
- Form
- Proactive
- Language understanding
- QnA

Slide 54

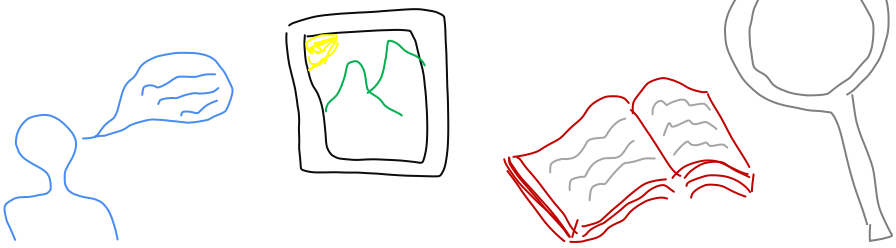


A QnA bot guide (can choose the SDK here): https://docs.botframework.com/en-us/azure-bot-service/templates/qnamaker/#navtitle

Slide 55

More integrations…

- LUIS is tightly coupled in the SDKs
- Easy to integrate any Cognitive Service with it's REST API



Caveat – some chaining of cognitive services may have to be done, however ones like LUIS and QnA maker are easy to integrate as they have similar mechanisms

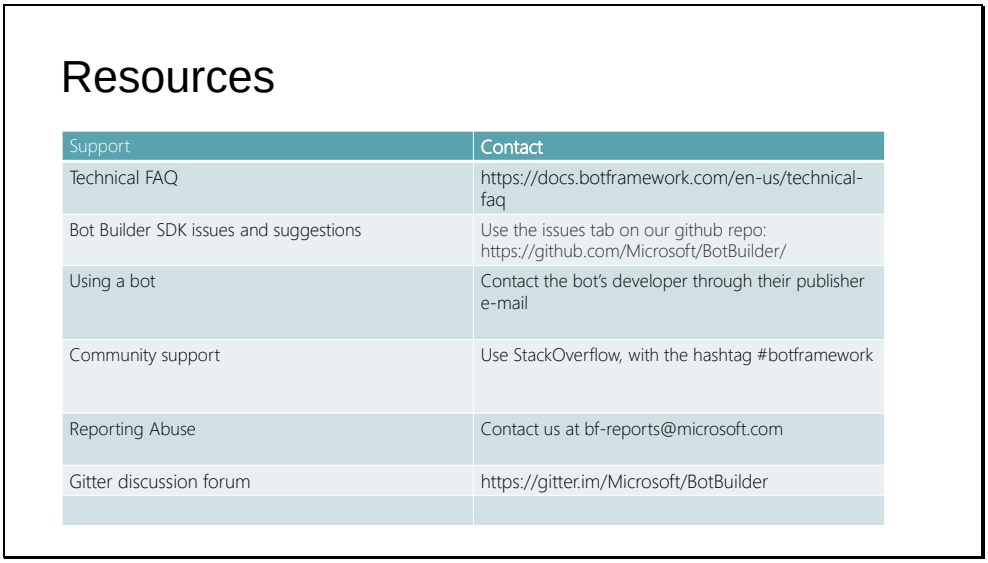
Slide 56

Parting thought (before the lab)

We can aim for our bots to: Be transparent and have algorithmic accountability so that humans can undo unintended harm. (Satya Nadella)

Ethical and societal considerations taken directly from an article by Satya Nadella:
https://www.linkedin.com/pulse/partnership-future-how-humans-ai-can-work-together-solve-nadella

Slide 57



https://docs.botframework.com/en-us/support

Slide 58

Prerequisites for the lab parts

GitHub account
Microsoft account

VSCode
Node.js with npm installed locally
Git bash
Bot Framework emulator
Azure account[recommended and needed to deploy]

Slide 59

Lab

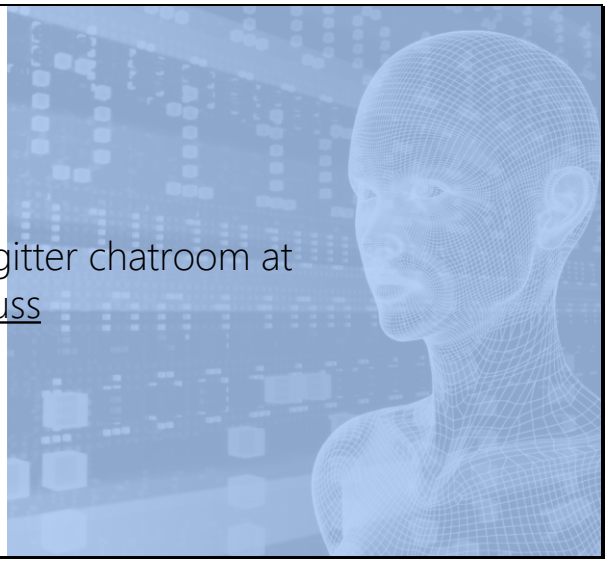
Bot dev environment

Instructions (lab links also in the wiki under the Agenda for the Syllabus)
https://github.com/Azure/bot-education/blob/master/Student-Resources/Labs/Node/Lab1_SetupCheck.md

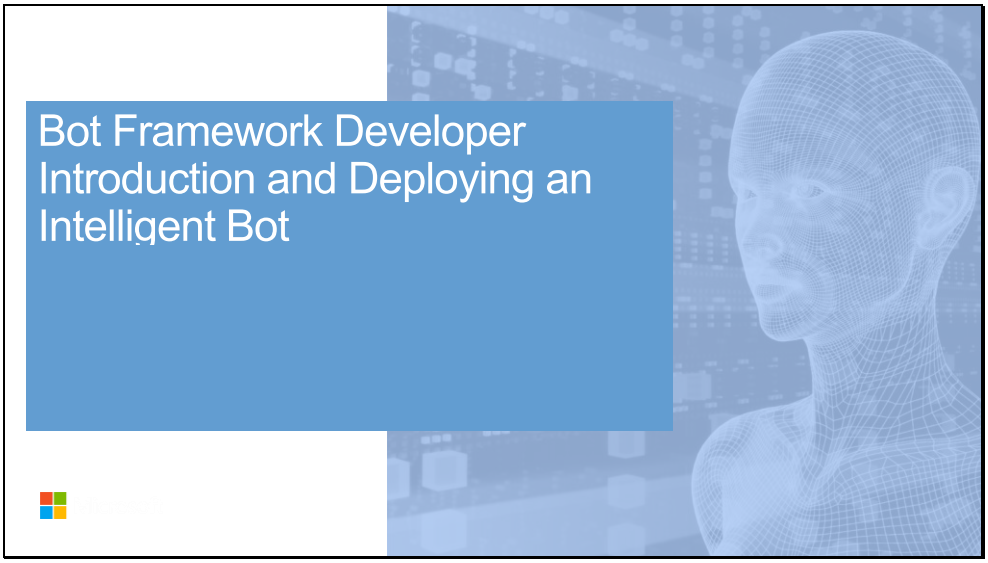
Slide 60

Questions?

Also, try the course gitter chatroom at aka.ms/botedu-discuss



Slide 61

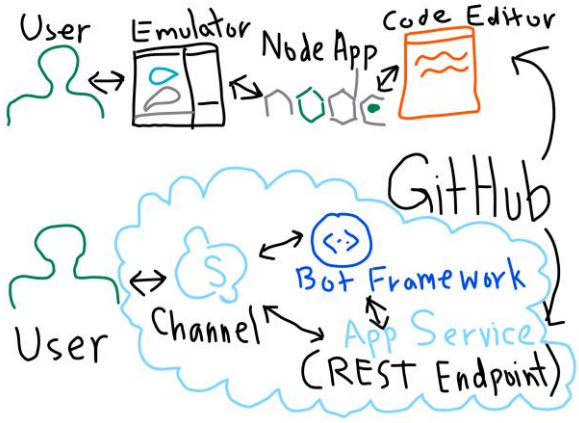

Bot Framework Developer Introduction and Deploying an Intelligent Bot

Slide 62

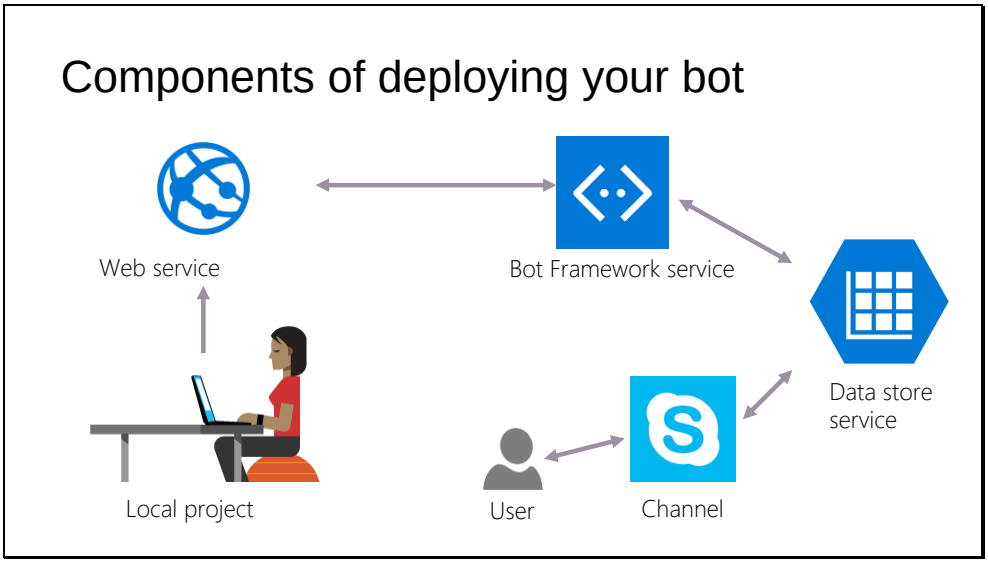
Core concepts: https://docs.botframework.com/en-us/node/builder/guides/core-concepts/#navtitle

Slide 63

Dev process/Toolbox



Slide 64



Steps:
The developer writes their bot code, leveraging the BF libraries in the SDKs and APIs available
Create an endpoint for the bot to talk to in the cloud
Connect to the Bot Framework service (Connector and State services)
Pass user and conversation data between channel and Framework (data store in the state service, managed by the BF)
The user interacts with the bot on a channel of their choosing

Slide 65

Session outline (and steps to releasing a bot)

1. Develop
2. Deploy
3. Register
4. Publish
5. Test

- *Deploy a bot to Azure lab*

Slide 66

Develop

in code editor

Slide 67

Concept and code: create the bot

UniversalBot – a simplified way to connect bots to dialogs

```
// Create bot
var bot = new builder.UniversalBot(connector);

// Dialog handling
bot.dialog('/', function (session) {
       session.send('Hello World');
});
```

You will use your setup environment in VSCode (or elsewhere) to work through these examples. The code is also at: https://github.com/Azure/bot-education/blob/master/Student-Resources/BOTs/Node/bot-playground/server.js

Note on dialogs: Bot Builder breaks conversational applications up into components called dialogs. If you think about building a conversational application in the way you'd think about building a web application, each dialog can be thought of as route within the conversational application.

From: https://docs.botframework.com/en-us/node/builder/guides/core-concepts

UniversalBot (https://docs.botframework.com/en-us/node/builder/chat-reference/classes/_botbuilder_d_.universalbot)
- has a lightweight connector model and includes ChatConnector and ConsoleConnector classes
- your bot can even utilize both the ChatConnector and ConsoleConnector and others at the same time if so desired
- replaces and unifies old classes like BotConnectorBot and TextBot

- updates and changes from https://docs.botframework.com/en-us/node/builder/whats-new/

Slide 68

Concept and code: create bot with handler

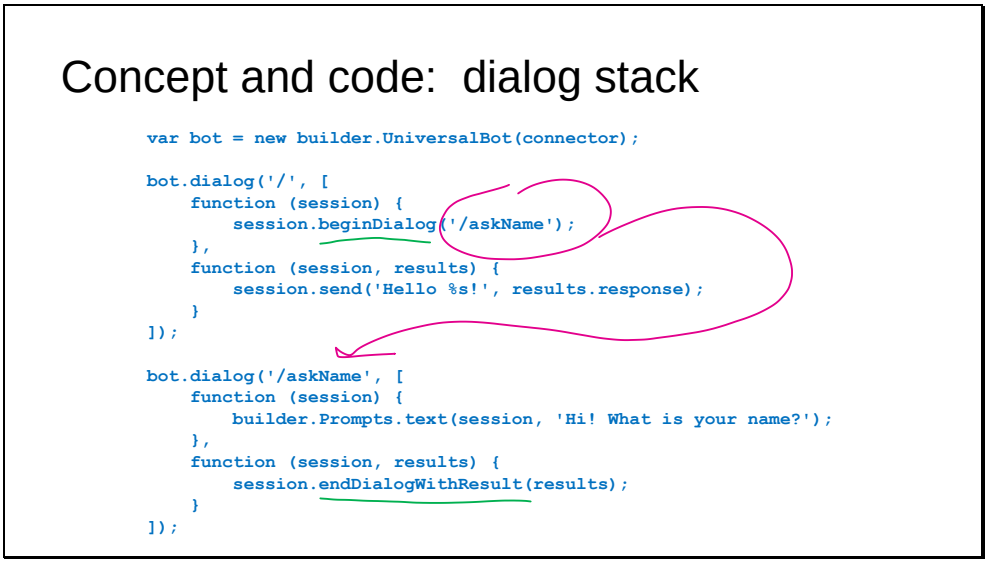
New code style in botbuilder release v3.5.0: create the bot with the root message handler at the start

```
var bot = new builder.UniversalBot(connector, [
    function (session) {
        // Introducing a builtin prompt - first waterfall step
        builder.Prompts.text(session, 'Hi! What is your name?');
    },
    function (session, results) {
        // Step 2 in the waterfall
        session.send('Hello %s!', results.response);
    }
]);
```

You will use your setup environment in VSCode (or elsewhere) to work through these examples.

Really a style choice
This is a more complex sample than just a Hello World bot in the previous slide and has steps – what are the collection of these steps called?
Here we are working with user data and message data

Slide 69



You will use your setup environment in VSCode (or elsewhere) to work through these examples.

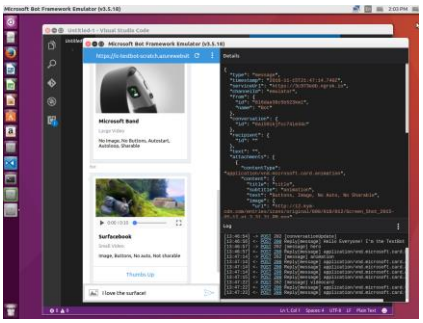
Here let's assume we've created out bot as in the previous slide (var bot = …)

beginDialog – pushes data onto the "stack"
endDialogWithResults – pops data off of the "stack" and in this case returns the results (we could have done some more interesting things with the data of course…)

Note: both dialog "routes" have small waterfalls

Slide 70

The BF Emulator – Test your code locally



- **New** Support for Mac, Linux and Windows
- **New** All the Bot Framework card types are supported
- **New** Save multiple profiles for when you're working online and off
- **New** Simplifies configuration when you're working with ngrok
- **New** Uses the webchat control for higher fidelity layout and consistency with the webchat experience

Emulator purpose:
Send requests and receive responses to/from your bot endpoint on localhost
Inspect the Json response
Emulate a specific user and/or conversation

More info on the open source Emulator project page: https://github.com/microsoft/botframework-emulator/wiki/Getting-Started

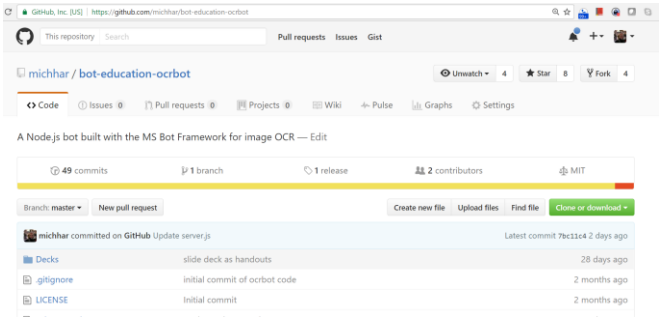
Slide 71

We will do this lab at the end

Slide 72

Step 1: Github repo

- Your bot code and related files will be on one github repo



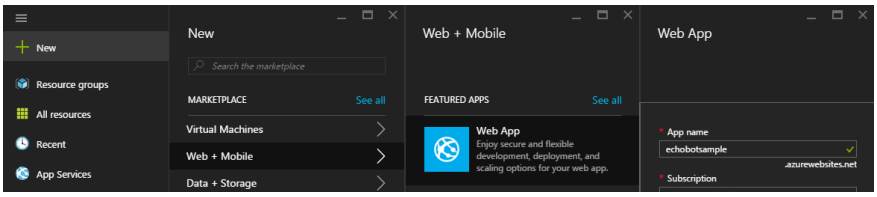
For example you could fork or clone https://github.com/michhar/bot-education-ocrbot
The server.js is the app
The package.json has the specs on what packages to install

On the command line you'd type:
npm install
node server.js

Note: there are other deployment methods other than from GitHub

Slide 73

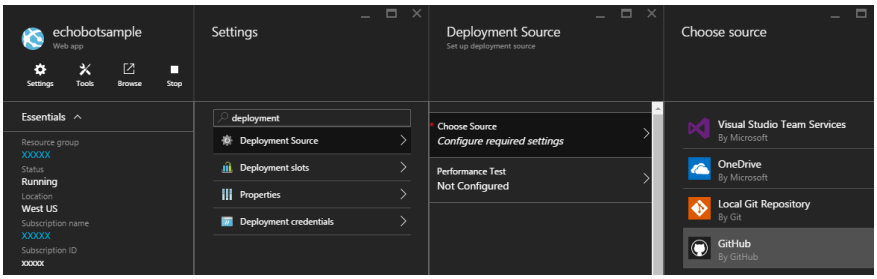
Step 2: Create an Azure Web App



In azure portal: https://portal.azure.com

Slide 74

Step 3: Set up for continuous deployment



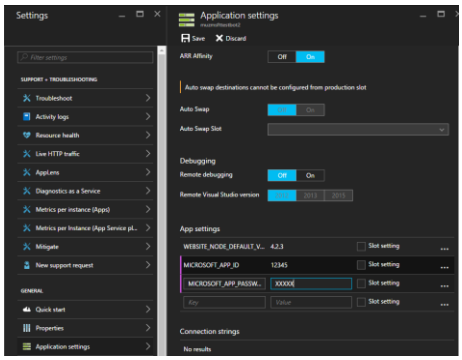
In azure portal: https://portal.azure.com

Select github and enter your credentials for GitHub when asked.

Click on Browse to test the deployment. Should see the message from the index.html file.

Slide 75

Step 4: Enter in App Id and App Password



This allows the password to be safe and guarded.

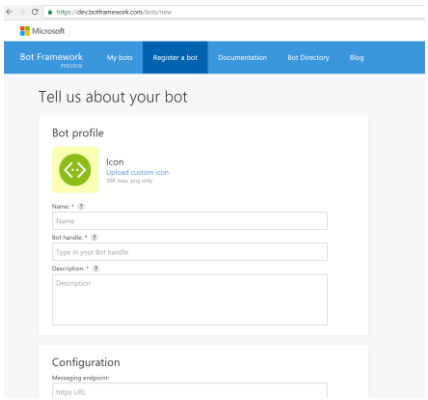
Slide 76

Register

in the Developer Portal

Slide 77

Register in the Bot Framework Developer Portal



On https://dev.botframework.com click on "Register a bot"

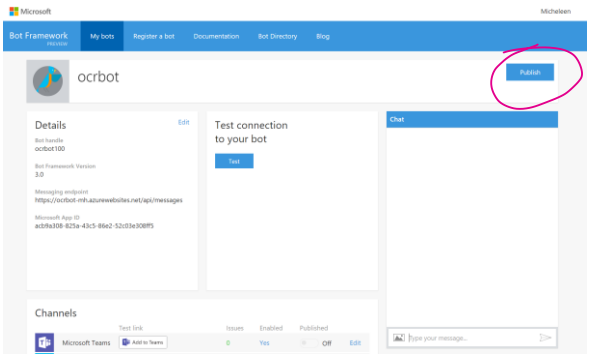
Slide 78

Publish to the Bot Directory

In the Developer Portal

Slide 79

Publish the registered bot



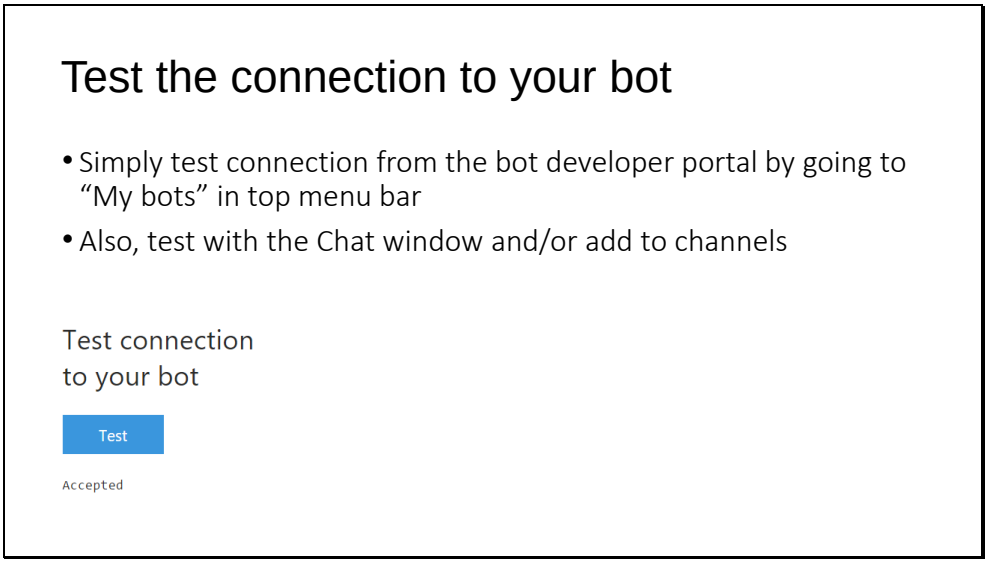
If you wish – it will then be reviewed and surface in the Bot Directory
Review guidelines: https://docs.botframework.com/en-us/directory/review-guidelines/

Test connection and conversation

Slide 81

Test the connection to your bot

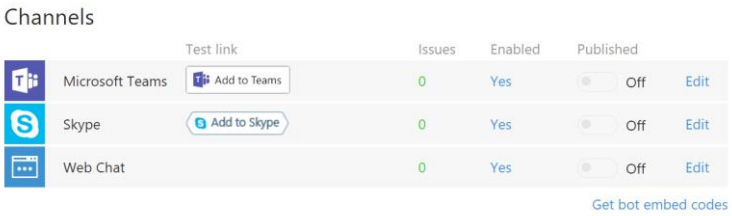
- Simply test connection from the bot developer portal by going to "My bots" in top menu bar
- Also, test with the Chat window and/or add to channels



Slide 82

Test connection with channels

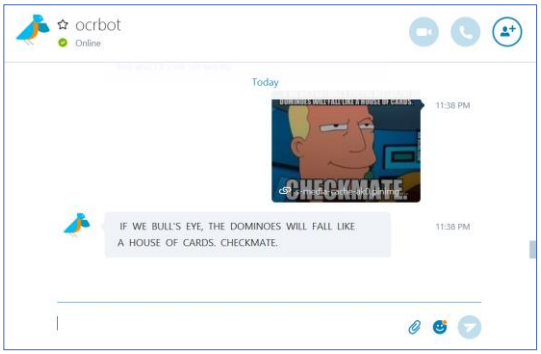
Test bot with a pre-configured channels



Slide 83

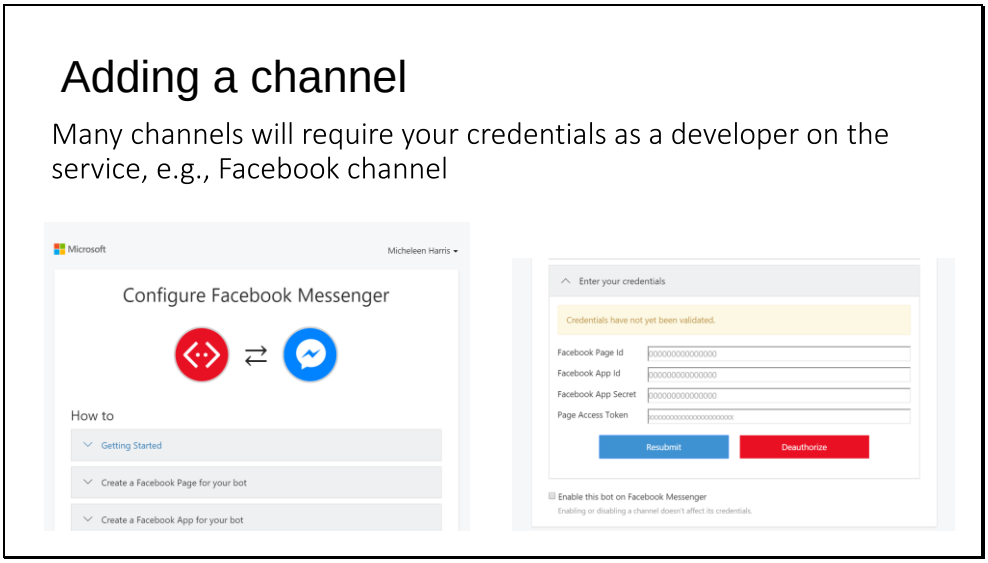
Test connection with a channel

Add bot to contacts in Skype and began a chat…



From developer portal page, clicked on test link "Add to Skype" and added bot to my contacts for testing.

Slide 84



Often, the most time will be spent configuring your credentials as a developer on the target service, registering your app, and getting a set of Oauth keys that Microsoft Bot Framework can use on your behalf

Slide 85

Next steps

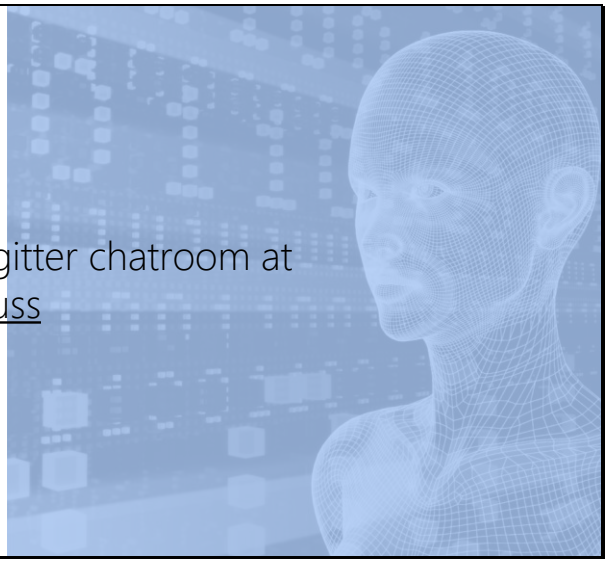
- Publish to Bot Directory
- Add bot diagnostics and telemetry with Azure App Insights
- Sign up as a developer for other channels supported by the Bot Framework and begin chatting on those as well

- Create more bots!

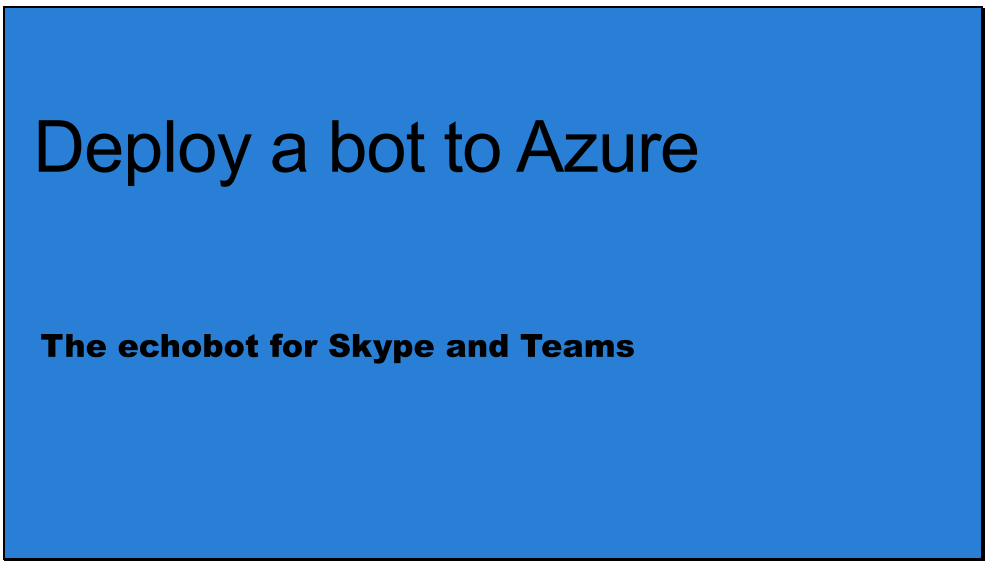
Slide 86

Questions?

Also, try the course gitter chatroom at aka.ms/botedu-discuss



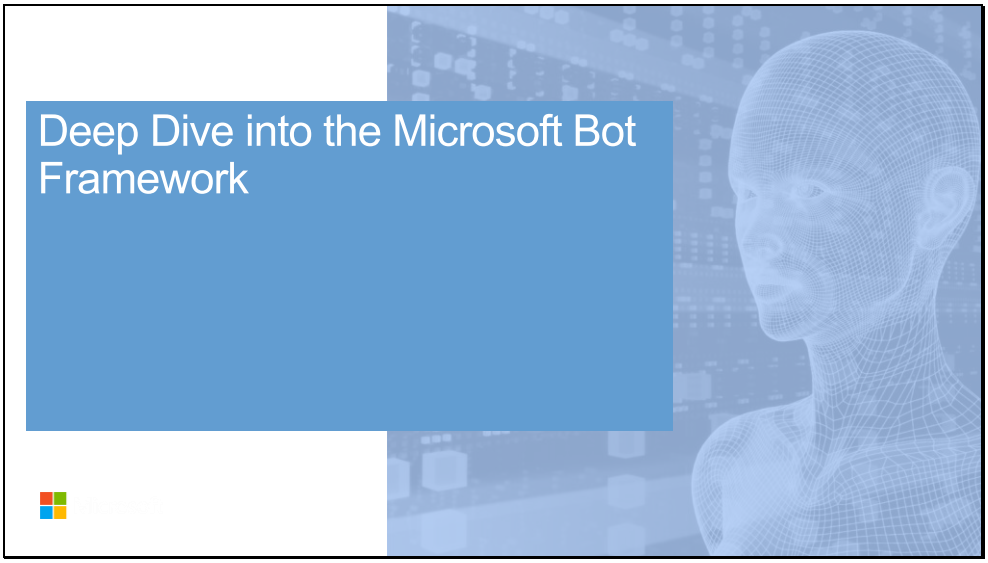
Slide 87



Follow along with the instructor and this guide: https://docs.botframework.com/en-us/node/builder/guides/deploying-to-azure/#i-want-to-setup-continuous-integration-from-github (you may want to "clone" instead of "fork" the repository for echobot)

Also, a useful post on this is at: https://michhar.github.io/posts/ocrbot-makes-a-connection

Slide 88



If the links in this deck are broken please let us know (mailto: michhar@Microsoft.com). Thanks in advance and enjoy learning about bots and the Microsoft Bot Framework.

Slide 89

Session outline

- Basics
- *QnA bot lab*
- Conversations and their features
- Conversation model
- *Dialog lab*
- *Attachments lab*
- State storage
- *Nonlinear waterfall lab*
- *Intents lab part 1*
- Adding intelligence
- *Intents lab part 2*

aka.ms/botedu aka.ms/bot-discuss

Slide 90

my notes:
You've seen some code so far, so let's dive in to the .NET Bot Builder again.

Slide 91

Install BotBuilder into our project

You've downloaded Node.js with npm (Node package manager)

If we have a package specification file (package.json):

```
npm install
```

If we don't have this file:

```
npm init
npm install --save botbuilder
```

From the core concepts here: https://docs.botframework.com/en-us/node/builder/guides/core-concepts/#navtitle

Setup - packages

- Import the Bot Builder SDK package
- Import “restify” to allow use to create the REST server for posting replies

```
// Required packages to import
var builder = require('botbuilder');
var restify = require('restify');
```

Setup –server

Set up the REST API server and set up message routing for the channel

```
// Set up the restify server
var server = restify.createServer();
server.listen(process.env.port || process.env.PORT || 3978, function () {
   //When testing on a local machine, 3978 indicates the port to test on
   console.log('%s listening to %s', server.name, server.url);
});
```

Slide 94

Setup – connecting to the BF

Pull in the registered bot id and password

Create a chat connector

Set up the posting of replies from bot

```
// Connector options
var botConnectorOptions = {
    appId: process.env.MICROSOFT_APP_ID || "",
    appPassword: process.env.MICROSOFT_APP_PASSWORD || ""
};

// Instatiate the chat connector and ready the REST API to send bot replies
var connector = new builder.ChatConnector(botConnectorOptions);
server.post('/api/messages', connector.listen());
```

Hook up the ChatConnector using the registration information which hooks it up to the Bot Framework (technically what's called the Bot Framework Connector) and set up a way for the bot to reply. Basically, we are setting up our REST API here with the restify server we just made.

App id and app password will remain empty during pre-deployed-bot local testing.

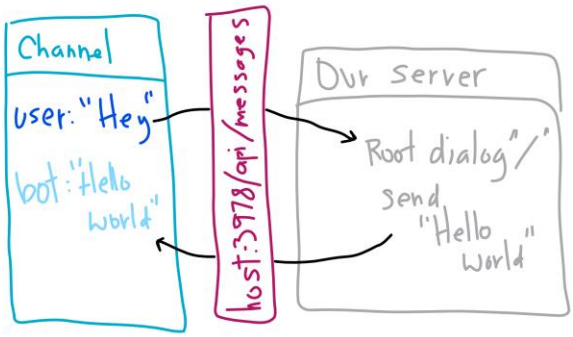
Setup – and finally create the bot

Initiate the bot with the chat connector object

```
// Create our bot and connect it to the BF connector
var bot = new builder.UniversalBot(connector);
```

Slide 96

Messaging and routing – quick view



Slide 97

Messaging and routing – in code

All user messages route first to the root dialog

```
// This is the root dialog
bot.dialog("/", function(session) {
    session.send("Hello world");
});
```

Nothing user-message specific happens here.

Slide 98

Conversation

In cases involving a **conversation**, the message activity gets passed to a dialog "root method" which handles the entire dialog process asynchronously (with "bot.dialog")

"session" is our conversation model

"bot.dialog" is the dialog handler

```
bot.dialog('/', [
    function (session) {
        // "push" onto the dialog stack
        session.beginDialog('/askName');
    },
    function (session, results) {
        session.send('Hello %s!', results.response);
    }
]);

bot.dialog('/askName', [ // this is a small waterfall
    function (session) {
        builder.Prompts.text(session, 'Hi! What is your name?');
    },
    function (session, results) {
        // "pop" off of the dialog stack and return the results
        session.endDialogWithResult(results);
    }
]);
```

The dialog "root method" is called SendAsync in C# and it controls the following process:

- Instantiate the required components.
- Deserialize the dialog state (the dialog stack in the storage "bags" and each dialog's state)
- Resume the conversation processes where the Bot decided to suspend and wait for a message.
- Send the replies.
- Serialize the updated dialog state and storage "bags" and persist it back to the state store.

Notes:
The dialog process can involve sub-dialogs
In between 2 and 3 a dialog object gets made and the message processed, possibly sub-dialogs spawned and further messages obtained from the user, message waiting occurs, dialog state updated, and more.

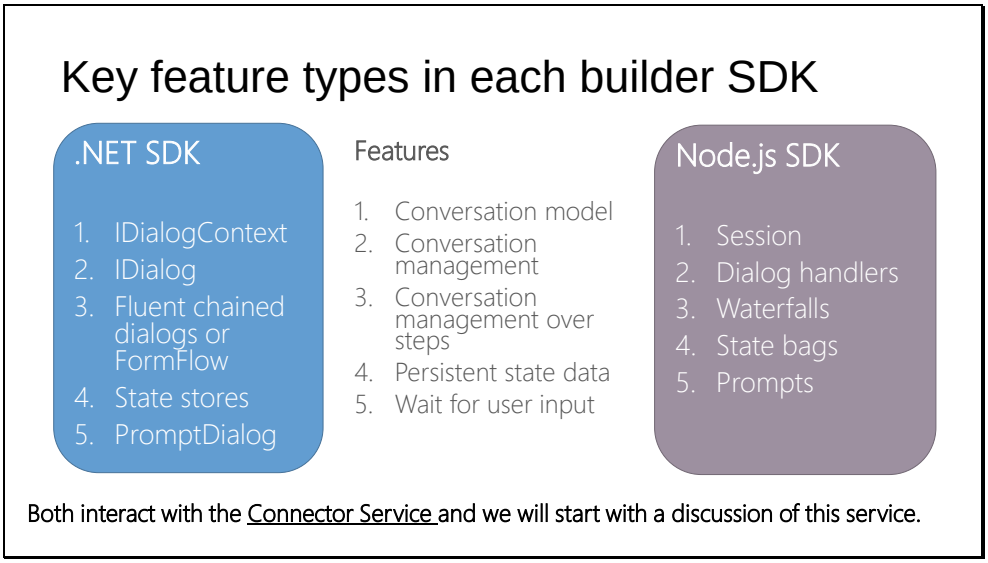
Slide 99

Conversations and their key features

Overview

Conversation – dialog or stack of dialogs handled by a model

Slide 100



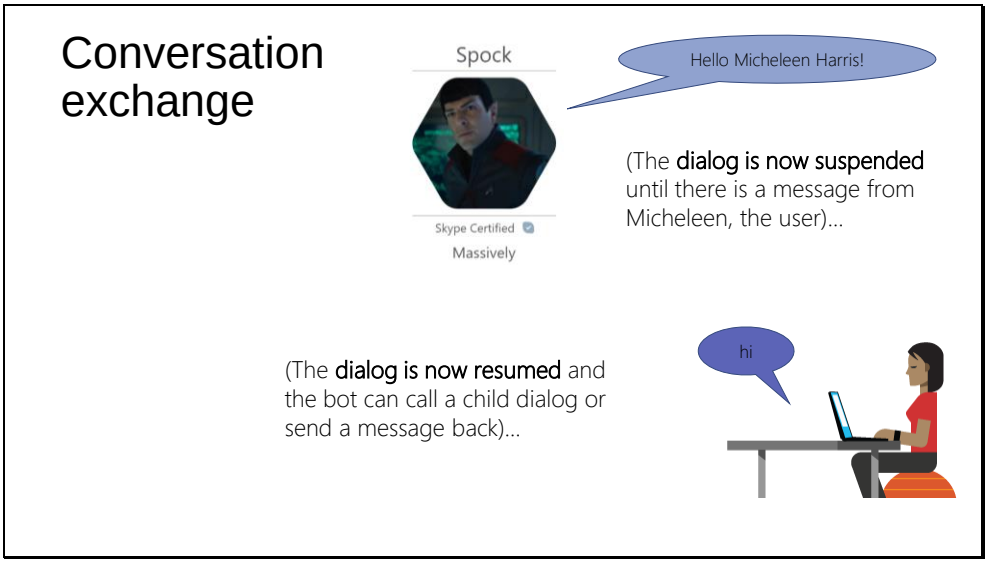
They may use different names and different implementations, but have essentially all of the same key features.

Slide 101

Conversation management

Dialog handlers

Slide 102



Bot Builder lets you break your bots' conversation with a user into parts called dialogs. You can chain dialogs together to have sub-conversations with the user or to accomplish some micro task. node.js – prompts and even waterfalls are implemented internally as dialogs

Slide 103

UniversalBot

Manages all conversations between a bot and user(s)

- Connectors
- Proactive messaging

https://docs.botframework.com/en-us/node/builder/chat/UniversalBot/#navtitle

Connecting the bot to a framework

Two methods

1) Connect to the **local console** (e.g. `ConsoleConnector` class)

2) Connect to the **Bot Framework (or emulate it at least)** (e.g. `ChatConnector` class)

https://docs.botframework.com/en-us/node/builder/chat/UniversalBot/#connectors

Slide 105

Modes of message delivery

Two types

1) Reactive – in response to a user message (e.g. `session.send()`)

2) Proactive – in response to some external event (e.g. `bot.send()` or `bot.beginDialog()`)

Reactive is most common method delivery. A "pull" bot. (We've seen this type of code earlier – it's what we were doing)

But there's the option of "pushing" a message to a user based on an external event like if a product has shipped or there's a weather alert. We first need the "address" of the user so we can find them later.

https://docs.botframework.com/en-us/node/builder/chat/UniversalBot/#proactive-messaging

Slide 106

Starting conversations – the push bot

Options

1) You don't need to wait for a reply (e.g. `bot.send()`)
2) You do need to wait for a reply (e.g. `bot.beginDialog()`)

* Notice, we are not dealing with the session, but rather the bot object

Are we waiting for a reply or not here?

```
server.post('/api/notify', function (req, res) {
    // Process posted notification
    var address = JSON.parse(req.body.address);
    var notification = req.body.notification;

    // Send notification as a proactive message
    var msg = new builder.Message().address(address).text(notification);
    bot.send(msg, function (err) {
        // Return success/failure
        res.status(err ? 500 : 200);
        res.end();
    });
});
```

The differences between bot.send() and bot.beginDialog() are subtle.

Calling bot.send() with a message won't affect the state of any existing conversation between the bot & user so it's generally safe to call at any time.

Calling bot.beginDialog() will end any existing conversation between the bot & user and start a new conversation using the specified dialog.

As a general rule you should call bot.send() if you don't need to wait for a reply from the user and bot.beginDialog()if you do.

https://docs.botframework.com/en-us/node/builder/chat/UniversalBot/#starting-conversations

Slide 107

Dialogs and handlers

Conversational exchange elements

- Dialog object
- Closure
- Waterfalls
- Prompts
- other forms (e.g. graph extension as in bot-trees)

Dialog object – conversational exchange object
Closure – passed in single functions to create a 1-step waterfall which loops
Waterfalls – conversational exchange over steps
other forms (e.g. graph extension as in bot-trees)

Slide 108

Dialog handler example

bot.dialog object handles the message

A child dialog is made

```
bot.dialog('/', [
    function (session) {
        // "push" onto the dialog stack
        session.beginDialog('/askName');
    },
    function (session, results) {
        session.send('Hello %s!', results.response);
    }
]);

bot.dialog('/askName', [ // this is a small waterfall
    function (session) {
        builder.Prompts.text(session, 'Hi! What is your name?');
    },
    function (session, results) {
        // "pop" off of the dialog stack and return the results
        session.endDialogWithResult(results);
    }
]);
```

To understand dialogs its easiest to think of them as the equivalent of routes for a website.

All bots will have at least one root '/' dialog just like all websites typically have at least one root '/' route.

When the framework receives a message from the user it will be routed to this root '/' dialog for processing.

For many bots this single root '/' dialog is all that's needed but just like websites often have multiple routes, bots will often have multiple dialogs.

https://docs.botframework.com/en-us/node/builder/chat/dialogs/#overview

Slide 109

Closure

Single function (one-step “waterfall”)

```
// Create a bot
var bot = new builder.UniversalBot(connector);

// Dialog handling
bot.dialog('/', function (session) {
    session.send('Hello world');
});
```

Here, no matter what the user’s message is, the bot will always return “Hello world”

Slide 110

Waterfall

The **waterfall** is an *array of functions* that chain together to form steps in a dialog

```
// This is a dialog with a waterfall (an array of functions for a conversation)
// It makes use of a storage "bag" called the dialogData used for temp storage
bot.dialog('/waterfall', [
    function(session, args, next) {
        session.dialogData.stepone = "Made it through step one.  ";
        next();
    },
    function (session, results, next) {
        session.dialogData.steptwo = "Made it through step two.  ";
        next();
    },
    function (session, results) {
        var mymsg = session.dialogData.stepone + "<br>" + session.dialogData.steptwo + "<br>";
        session.endDialog(mymsg + "Let's end this waterfall and return control to the root dialog :) .");
    }
]);
```

By passing an array of functions for our dialog handler a waterfall is setup where the results of the first function are passed to the input of the second function. We can chain together a series of these functions into steps that create waterfalls of any length.

This code uses dialog data state "bag" to store some things. We didn't' have to do that, but it gave us a way to pass something from one dialog to the next in a temporary way.

What happens if we don't have a built-in prompt in use? (Ans: we'll need to call next() ourselves)

Slide 111

Waterfall example 1

Waterfalls are the most common form of dialogs

```
// What's different about this bot setup? (this is the new style, btw, for v3.5.0)
var bot = new builder.UniversalBot(connector, [
    function (session) {
        // Introducing a builtin prompt
        builder.Prompts.text(session, 'Hi! What is your name?');
    },
    function (session, results) {
        // Step 2 in the waterfall
        session.send('Hello %s!', results.response);
    }
]);
```

Most common form of dialogs. Fundamental skill.

Bots based on Bot Builder implement something we call "Guided Dialog" meaning that the bot is generally driving (or guiding) the conversation with the user.

Calling a built-in prompt like Prompts.text() moves the conversation along because the users response to the prompt is passed to the input of the next waterfall step.

In .NET FormFlows are used.

Slide 112

Waterfall example 2

Many-step waterfall

```
// Setup bot and root waterfall - os this old or new style?
var bot = new builder.UniversalBot(connector, [
    function (session) {
        builder.Prompts.text(session, "Hello... What's your name?");
    },
    function (session, results) {
        session.userData.name = results.response;
        builder.Prompts.number(session, "Hi " + results.response + ", How many years have you been coding?");
    },
    function (session, results) {
        session.userData.coding = results.response;
        builder.Prompts.choice(session, "What language do you code Node using?", ["JavaScript", "CoffeeScript",
            "TypeScript"]);
    },
    function (session, results) {
        session.userData.language = results.response.entity;
        session.send("Got it... " + session.userData.name +
                     " you've been programming for " + session.userData.coding +
                     " years and use " + session.userData.language + ".");
    }
]);
```

Most common form of dialogs. Fundamental skill.

Hello…What's your name
Hi <results> How many years have you been coding?
"What language do you code Node using?"

etc.

https://docs.botframework.com/en-us/node/builder/chat/dialogs/#waterfall

Slide 113

Prompts

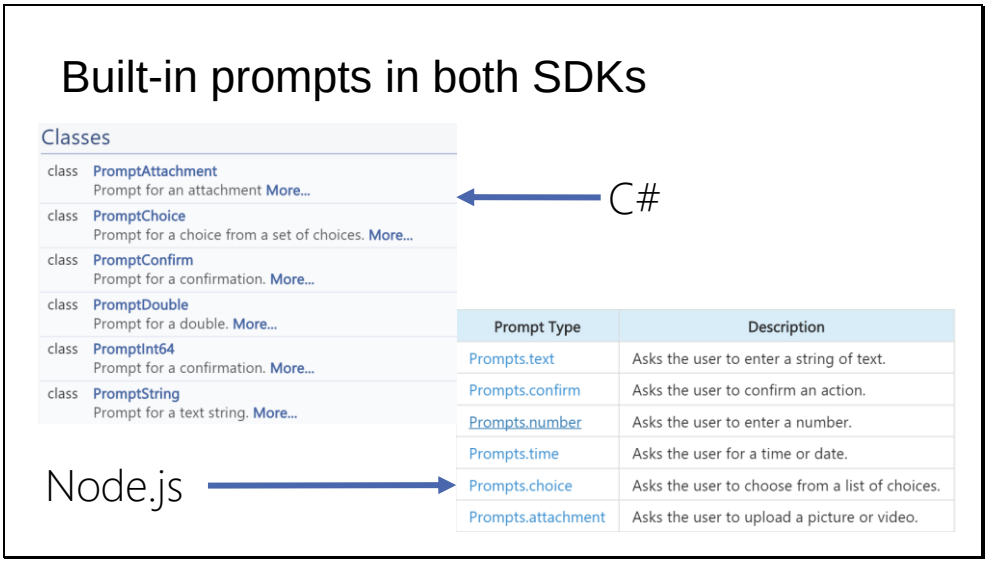
Use a **prompt** to *wait* and obtain user input.

Use a **built-in prompt** for expected types.

```
// Setup bot and root waterfall - os this old or new style?
var bot = new builder.UniversalBot(connector, [
    function (session) {
        builder.Prompts.text(session, "Hello... What's your name?");
    },
    function (session, results) {
        session.userData.name = results.response;
        builder.Prompts.number(session, "Hi " + results.response + ", How many years have you been coding?");
    },
    function (session, results) {
        session.userData.coding = results.response;
        builder.Prompts.choice(session, "What language do you code Node using?", ["JavaScript", "CoffeeScript",
            "TypeScript"]);
    },
    function (session, results) {
        session.userData.language = results.response.entity;
        session.send("Got it... " + session.userData.name +
                     " you've been programming for " + session.userData.coding +
                     " years and use " + session.userData.language + ".");
    }
]);
```

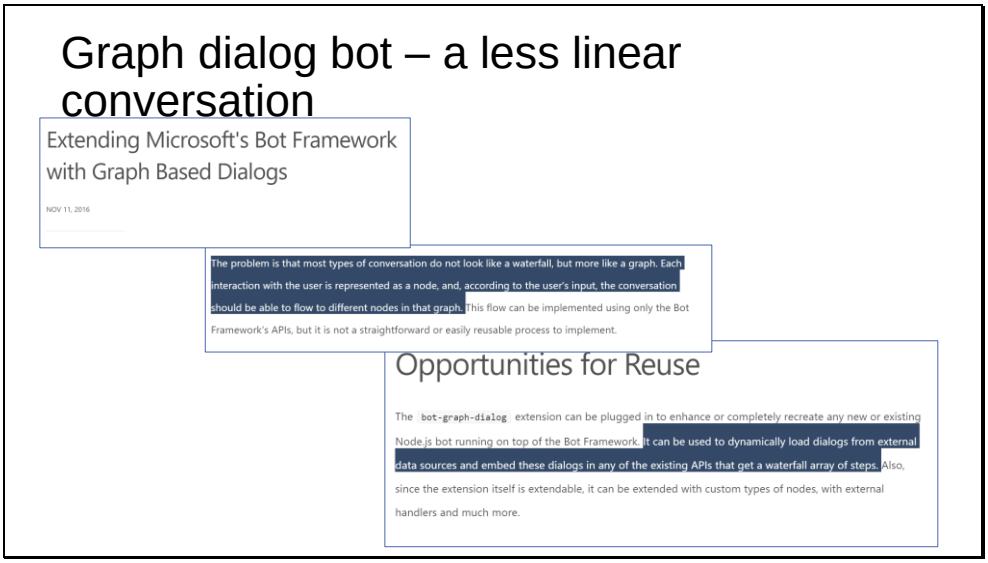
To actually wait for the users input we're using one of the SDK's built in prompts. We're using a simple text prompt which will capture anything the user types but the SDK has a wide range of built-in prompt types.

Slide 114



my notes:
A "dialog factory" for simple prompts to make obtaining user input easier.

Slide 115



This is an extension that helps us do two things: 1) create a more flowing conversation by taking away the constraint that it must be linear as we've seen in waterfalls 2) allows us to load a preset "graph" or dialog flow from a data source like a file.

More at: https://www.microsoft.com/developerblog/real-life-code/2016/11/11/Bot-Graph-Dialog.html

Slide 116

Conversation model

Sessions or contexts

Slide 117

Sessions 

Session contains the stack of active dialogs

- Sending messages
- Dialog stack
- Callbacks

https://docs.botframework.com/en-us/node/builder/chat/session/#navtitle

Sending messages (text, attachments, cards, etc.)
Dialog stack (starting, ending, replacing, etc.)
Callbacks (usage)

Slide 118

Sending messages

A text message is easy. Just `session.send()` for example.

What's more fun is sending attachments.

What type of attachment is being sent here?

```
// Send an image attachment to the user
bot.dialog('/picture', [
    function (session) {
        session.send("You can easily send pictures to a user...");
        var msg = new builder.Message(session)
            .attachments([{
                contentType: "image/jpeg",
                contentUrl: "http://www.theoldrobots.com/images62/Bender-18.JPG"
            }]);
        session.endDialog(msg);
    }
]);
```

Attachments types allowed: image, video, and file

Slide 119

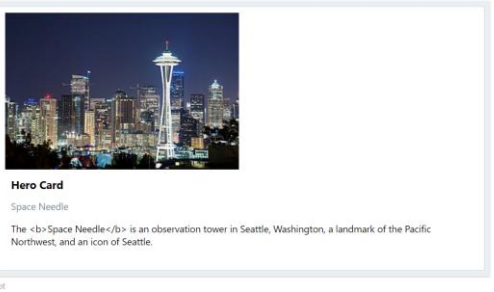
Sending messages cont'd

Cards – hero and thumbnail cards which can have text, images or buttons even

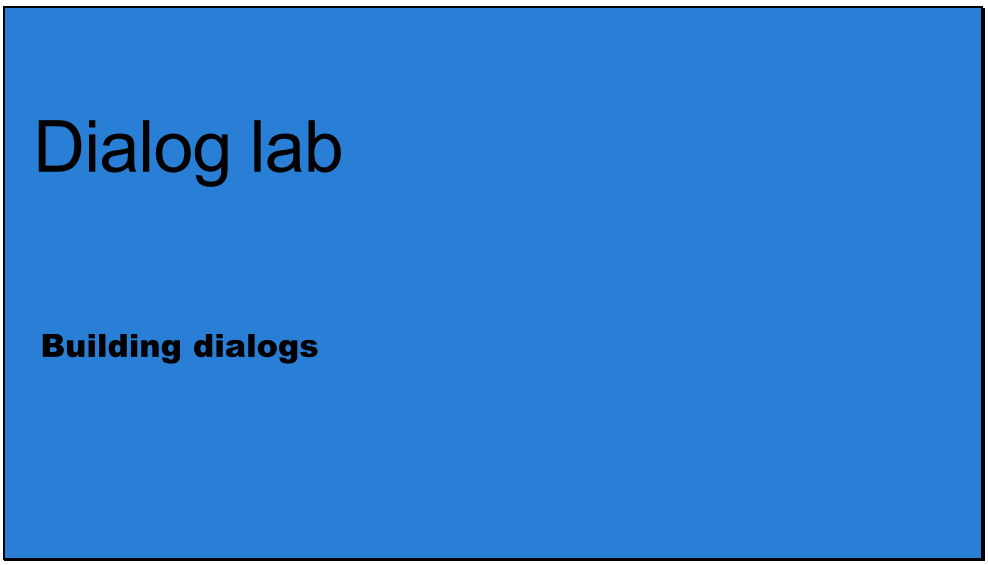
```
var msg = new builder.Message(session)
    .textFormat(builder.TextFormat.xml)
    .attachments([
        new builder.HeroCard(session)
            .title("Hero Card")
            .subtitle("Space Needle")
            .text("The <b>Space Needle</b> is an observation tow
            .images([
                builder.CardImage.create(session,
                    "https://upload.wikimedia.org/wikipedia/comm
            ])
            .tap(builder.CardAction.openUrl(session, "https://en.
    ]);
```

Build the message

Card as reply



Slide 120



Follow the instructions in the code (server.js) for this sample bot: https://github.com/Azure/bot-education/tree/master/Student-Resources/BOTs/Node/bot-playground

Slide 121

Attachments and cards lab

Building a bot that sends an image attachment

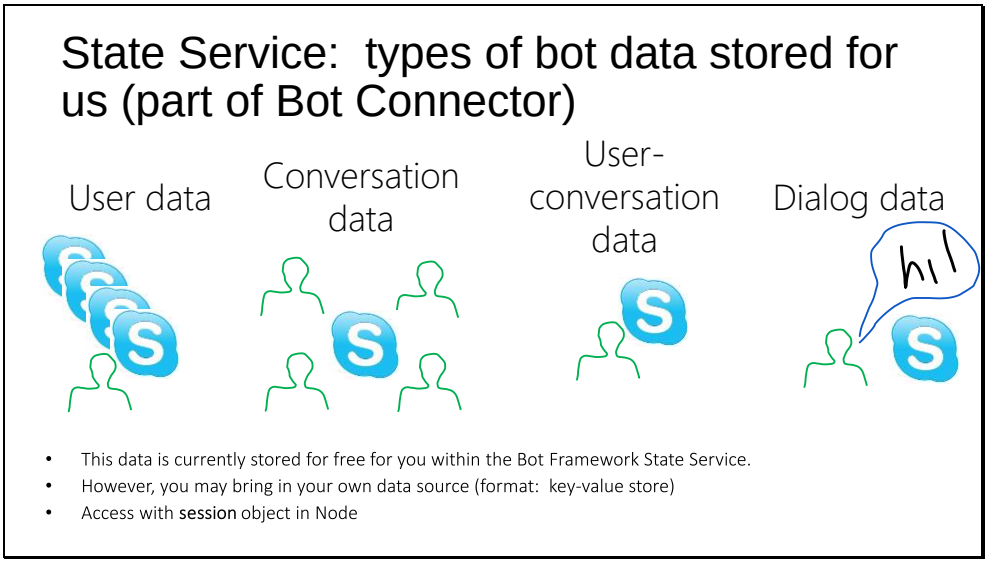
Add an image attachment and a hero card (fill in the proper code) for this bot: https://github.com/Azure/bot-education/tree/master/Student-Resources/BOTs/Node/bot-cardbot

You can check your code with Node code under Instructor-Resources at https://github.com/Azure/bot-education/tree/master/Instructor-Resources/BOTs/Node

Slide 122

https://docs.botframework.com/en-us/node/builder/guides/core-concepts/#adding-dialogs-and-memory

Slide 123



User data – globally available for user across all conversations

conversation data – stores globally for a single conversation (many users could be involved)

User-conversation data – stores globally conversation data for a user (But private to just that user)

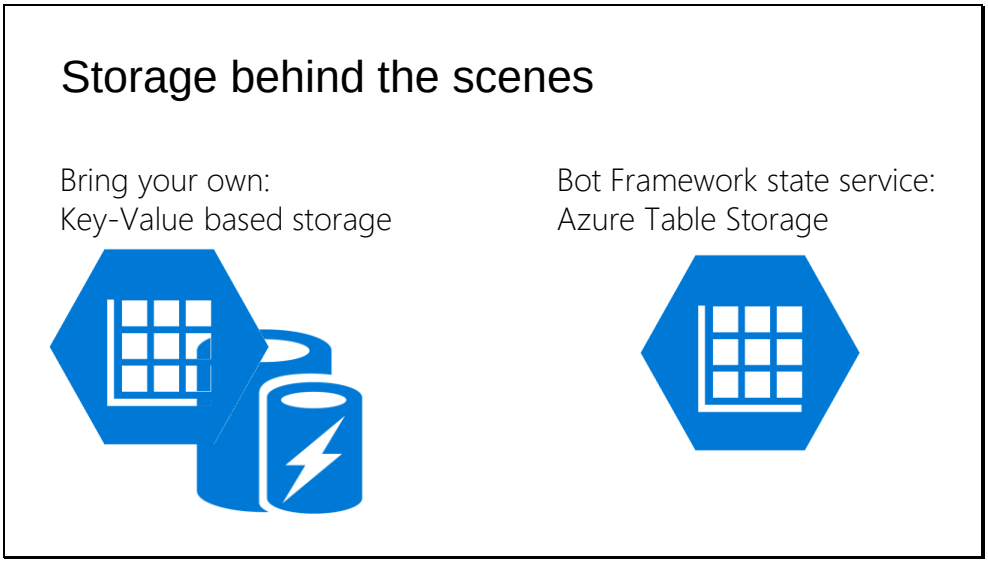
Dialog data as well – persists for a single dialog (helpful for temp data in a waterfall set of steps)

If I have a bot that plays Blackjack with me, my stats would be stored in user data (would follow me around from game to game), the deck information and stats in the conversation data (i.e. other players could use the same deck), and my hand in a game would be in user-conversation data (my immediate game's data).

The dialog data persistence ensures that the dialogs state is properly maintained between each turn of the conversation. Dialog data also ensures that a conversation can be picked up later and even on a different machine.

Anything can be stored in these data stores or bags, however it should be limited to data types that are serializable like primitives.

Slide 124



The State REST API has wrappers built around Azure Table Storage. NB, you can bring your own storage in the form of a key-value store like Redis Cache, Table Storage etc. The Bot Framework manages this default storage for you so you can maintain a stateless bot experience and if you bring your own, you'll need to maintain that store and make sure it scales.

Slide 125

Persistent state data

State storage, managed by State API

Also, called storage "**bags**"

```
// Setup bot and root waterfall - os this old or new style?
var bot = new builder.UniversalBot(connector, [
    function (session) {
        builder.Prompts.text(session, "Hello... What's your name?");
    },
    function (session, results) {
        session.userData.name = results.response;
        builder.Prompts.number(session, "Hi " + results.response + ", How many years have you been coding?");
    },
    function (session, results) {
        session.userData.coding = results.response;
        builder.Prompts.choice(session, "What language do you code Node using?", ["JavaScript", "CoffeeScript",
            "TypeScript"]);
    },
    function (session, results) {
        session.userData.language = results.response.entity;
        session.send("Got it... " + session.userData.name +
                     " you've been programming for " + session.userData.coding +
                     " years and use " + session.userData.language + ".");
    }
]);
```

Remember our:
User data
Conversation data
Private conversation data
Dialog data

Additional elements

Recognizers
Localization
Calling
Extensions

Slide 127

Developer tip

Maintain code in staging spaces:

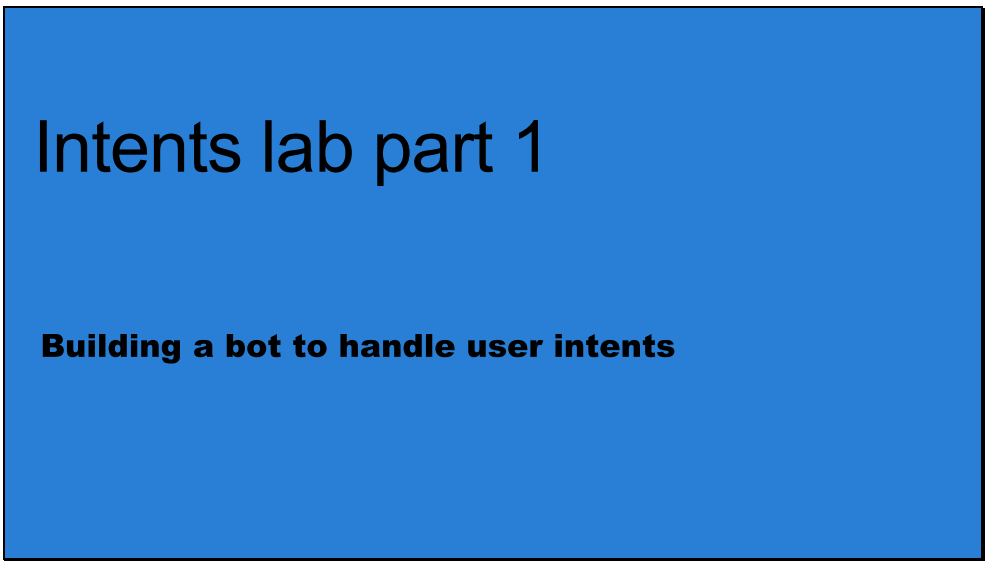
- Local – Bot Framework connects to my local machine

- Dev – push publicly and BF connects in cloud

- Prod – what goes in Bot Directory; final product

Local – e.g. in Node.js can use ngrok as a local server tunnel service (get a free account if wish to ngrok https://dashboard.ngrok.com/user/signup)

Slide 128



Follow instructions here: https://github.com/Azure/bot-education/tree/master/Student-Resources/BOTs/Node/bot-simpleintent

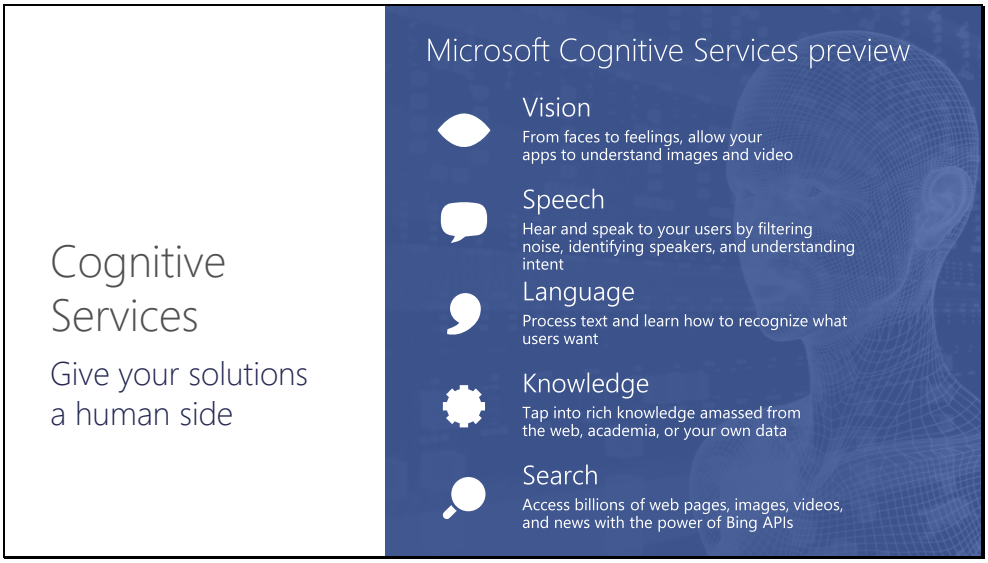
Advanced:

- Can you deal with the date using Date datetime formats? See https://docs.botframework.com/en-us/node/builder/chat/IntentDialog/#resolving-dates--times

Slide 129

Adding intelligence to your bot

Slide 130



What are Cognitive Services? Microsoft Cognitive Services are a new collection of intelligence and knowledge APIs that enable developers to ultimately build smarter apps.

So, what are Cognitive Services? Cognitive Services are a collection of artificial intelligence APIs, and we believe in *democratizing* artificial intelligence. So what that means is, regardless of your skill level -- whether you're a high school student running your first program or working in industry or in a giant enterprise -- that you should be able to use our APIs incredibly quickly in a matter of minutes.

And regardless of your platform -- whether you're on Android or IOS or Windows, or making a website -- all of our APIs are rest APIs, which means you can call them as long as you have an Internet connection. And so that's pretty huge because what we're doing is making it so that everyone can build these smarter, more context-aware applications.

The technology used in our APIs is the same technology that powers our products today. And so, when you think of things like the Bing search APIs, it's the same technology from Bing.

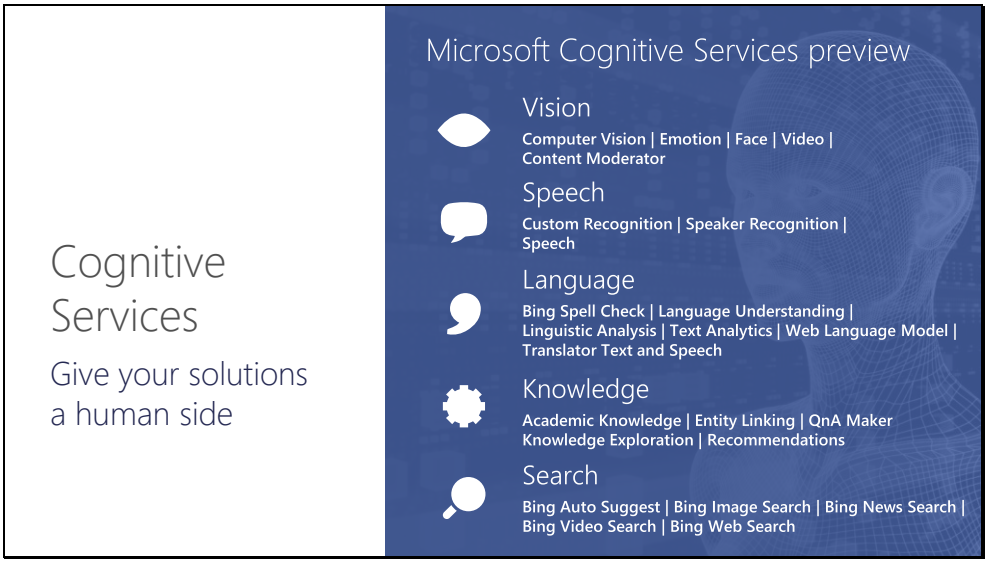
Today I'm going to talk with you about the entire collection spanning vision, speech, language, knowledge, and search.

The other things that I want to point out is that you can get started for free with all of the APIs, but we do have pricing available for a number of them, which are in public preview on Azure.

The other piece is the developer resources. So, all of our documentation is on the website and actually in GitHub as well, so we do welcome community submissions. We have a set of SDKs that are also available on GitHub where we welcome pull requests and post everything on there. The SDKs vary from API to API, but they are all included in this one repository for people to see.

And then we have three different communities that we support. We have our MSDN forums, our Stack Overflow, and we have User Voice that we use for feedback requests.

Slide 131



At Microsoft, we've been offering APIs for a very long time across the company. In delivering Microsoft Cognitive Services API, we started with 4 last year at /build (2015); added 7 more last December, and today we have 25 APIs in our collection.

Cognitive Services are available individually or as a part of the Cortana Intelligence Suite, formerly known as Cortana Analytics, which provides a comprehensive collection of services powered by cutting-edge research into machine learning, perception, analytics and social bots.

These APIs are powered by Microsoft Azure.

Developers and businesses can use this suite of services and tools to create apps that learn about our world and interact with people and customers in personalized, intelligent ways.

Slide 132

Leveraging the API collection

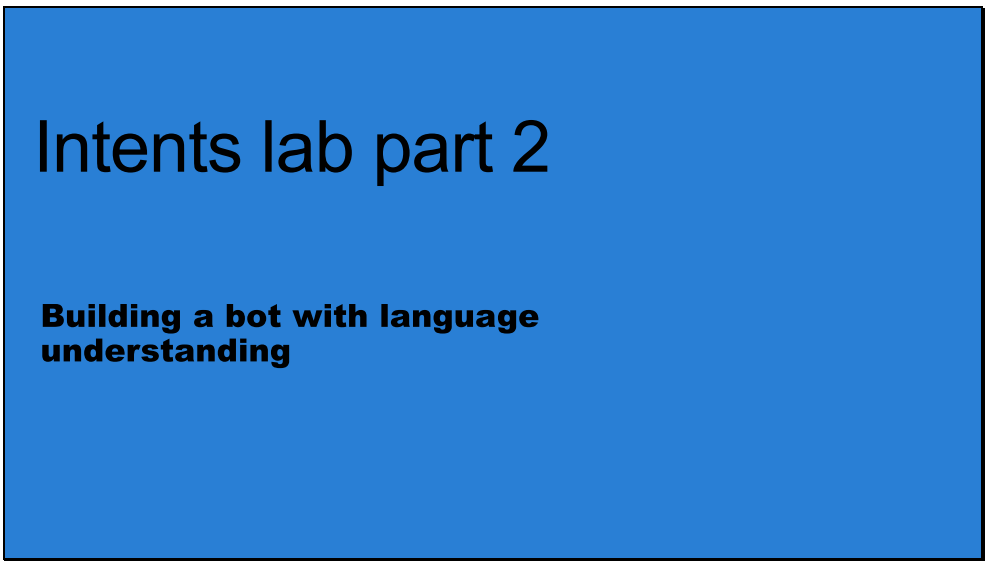
Any of the Cognitive Services APIs can be reached with a simple REST API call

```
var request = require("request");

var readImageText = function _readImageText(url, callback) {

    var options = {
        method: 'POST',
        url: config.CONFIGURATIONS.COMPUTER_VISION_SERVICE.API_URL + "ocr/",
        headers: {
            'ocp-apim-subscription-key': config.CONFIGURATIONS.COMPUTER_VISION_SERVICE.API_KEY,
            'content-type': 'application/json'
        },
        body: {url: url, language: "en"},
        json: true
    };

    request(options, callback);

};
```

Language integration with LUIS (for NLP based in intents) is easiest as a NuGet package exists within the bot builder for easy integration, however any of the Cognitive Services APIs can be reached with a simple REST API call.

Next, let's go straight to an example of implementing a Cognitive Service API (LUIS) within a bot

Slide 133



Follow the instructions and use the resources here: https://github.com/Microsoft/BotBuilder-Samples/tree/master/Node/intelligence-LUIS

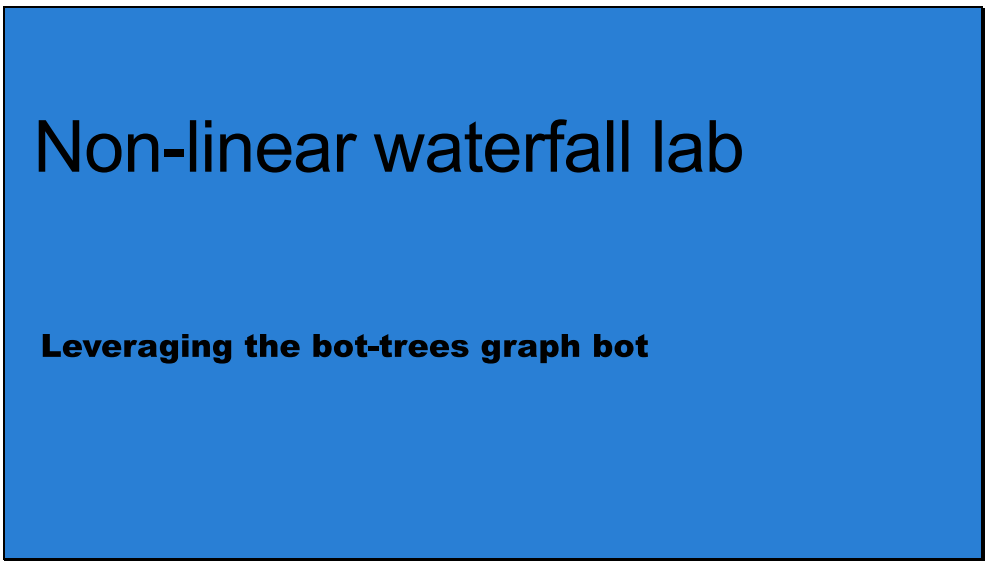
Good docs at https://docs.botframework.com/en-us/node/builder/guides/understanding-natural-language/#luis

Slide 134

Rate this tutorial!

https://aka.ms/botedu-survey

Slide 135



https://github.com/CatalystCode/bot-trees (follow instructions to install extension – this will be part of the Bot Framework in the future)

If you have some extra time try out the Azure Bot Service (https://docs.botframework.com/en-us/azure-bot-service/). Set the bot here or another one up for fun as an Azure Bot Service (use the echobot for instance as a template and then host the code for continuous deployment on GitHub and pulled in to Azure) – basically replace the "template" code given by Azure Bot Service with custom code while retaining the structure.

Slide 136

Resources

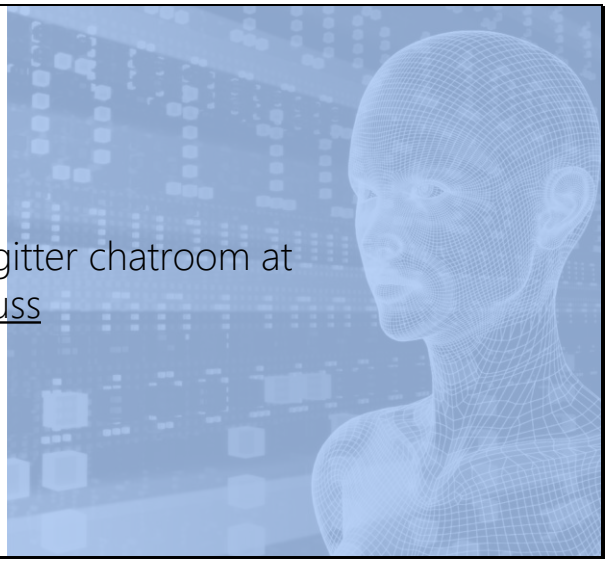
- Github issues
- Gitter
- Stackoverflow with botframework tag (#botframework)

https://github.com/Microsoft/BotBuilder/issues
https://gitter.im/Microsoft/BotBuilder
http://stackoverflow.com/questions/tagged/botframework

Slide 137

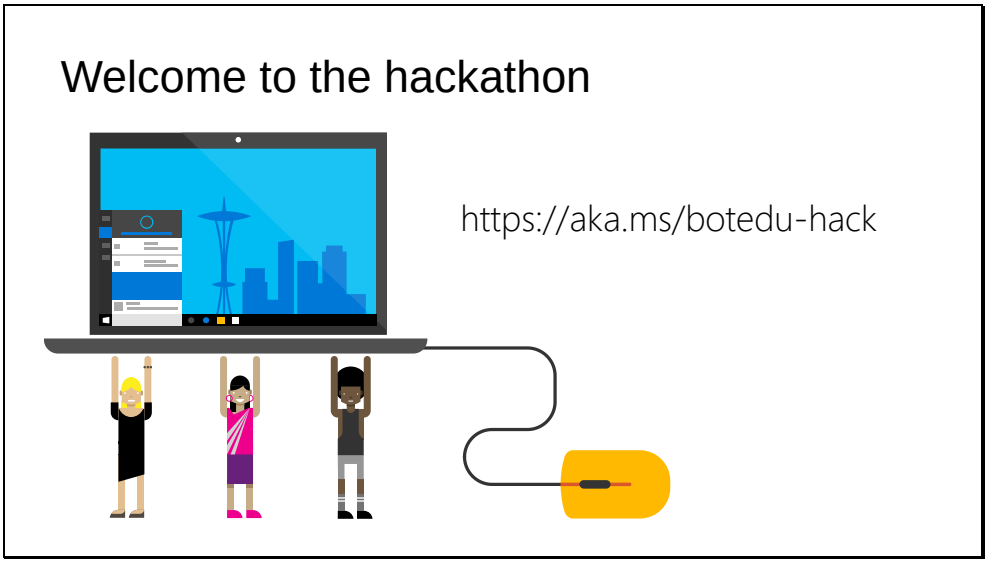
Questions?

Also, try the course gitter chatroom at aka.ms/botedu-discuss



Slide 138

Slide 139



https://aka.ms/botedu-hack

Slide 140

Q and A

Developing and Deploying Intelligent Chat Bots

Day 2

Extras

Slide 142

Hackathon schedule

Proposal (30 min): As a team come up with a concrete flow diagram to communicate the components and ideas involved in your proposal. Present your proposal in 2 minutes or less.

Implement idea using C#, Node.js or leverage the REST API.

Presentations (4:30pm): 4 minute showcase of your implementation (simply what you got done and future plans if need-be)

Example bot project: https://azure.microsoft.com/en-us/blog/bot-framework-made-better-with-azure/

Slide 143

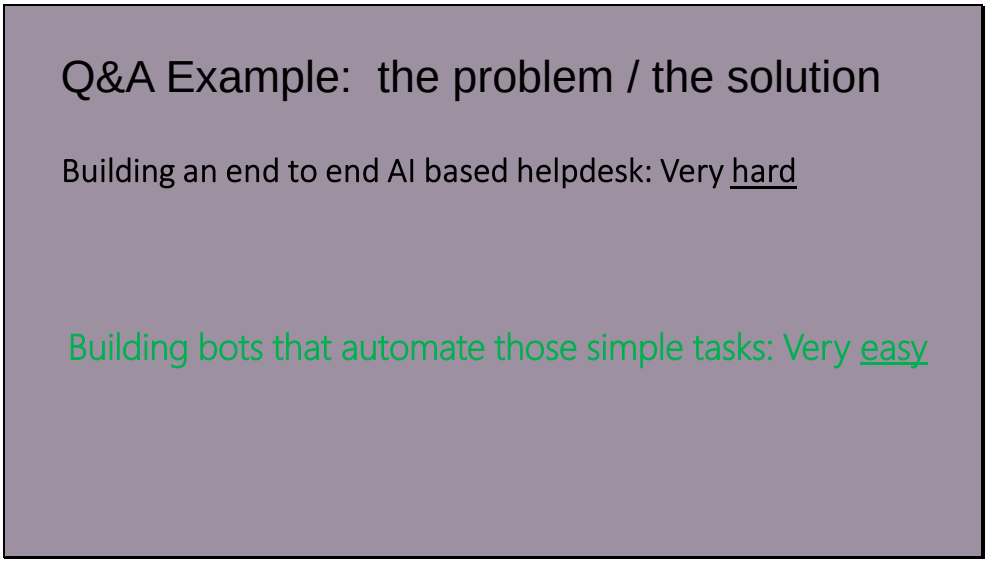
Example scenario for Q&A bot

"I want to automate my helpdesk…"

What is the most repetitive or takes the most of my employees' time…

- Password reset
- Opening a ticket
- Request to install software

Slide 144



So focus on the easy: Narrow down scenarios to tangible things, often repetitive tasks.

If you can see an app doing it, then you can see a bot doing it, too.

Slide 145

Hack Topic 1

Our current expectation is that we will address one or more of the primary challenges in managing humanitarian and medical assistance to the refugees from the Middle East and North Africa (MENA). There is a lack of data for NGOs on the ground working with refugees in Europe & the middle east. Technology could help document the crisis, help NGOs work together and solicit further support. Technical & marketing solutions are needed to evangelize and mobilize volunteers and donors. Procurement for in-kind donations is sporadic and disorganized. Technology solutions could be utilized to build efficiencies in sourcing of humanitarian goods and resources for missions. Awareness for this humanitarian crisis low and there is a need for community dialog and empathy.

Slide 146

Hack Topic 2

The goal is to address the essential education and enrichment needs of kids in foster care. An organization has been working with kids in foster care to help them secure essential education, basic needs and social experiences they deserve. We would like to develop a digital solution that will allow foster kids to transcribe their aspirations and goals, work with their mentors to create a plan to achieve them, and to keep track of their progress and achievements through continuous communication and feedback from their advisors.

Slide 147

Hack Topic 3

We need to create a communications app for non-verbal/emerging verbal kids in K-12 education. Currently Apple occupies this niche by default, but there has never been a true grounds-up effort to adapt existing technology (GPS, speech recognition, word banks, etc.) & marry it to a visual communications app to be a truly effective communications tools for so many kids in need. In essence this app can give a child a 'voice' and help them integrate into their communities and immensely improve their quality of life.

Slide 148

.NET SDK: NuGet packages galore for Cognitive Services

Cognitive services (aka ProjectOxford)

And many, many more – Connector and everything the .NET framework provides