

Developing and Deploying Intelligent Chat Bots

Presented by Micheleen Harris

For TRPP24



Main site: aka.ms/botedu
Chat room: aka.ms/botedu-discuss

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

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 😊

Prerequisites for the lab parts

GitHub account

Microsoft account

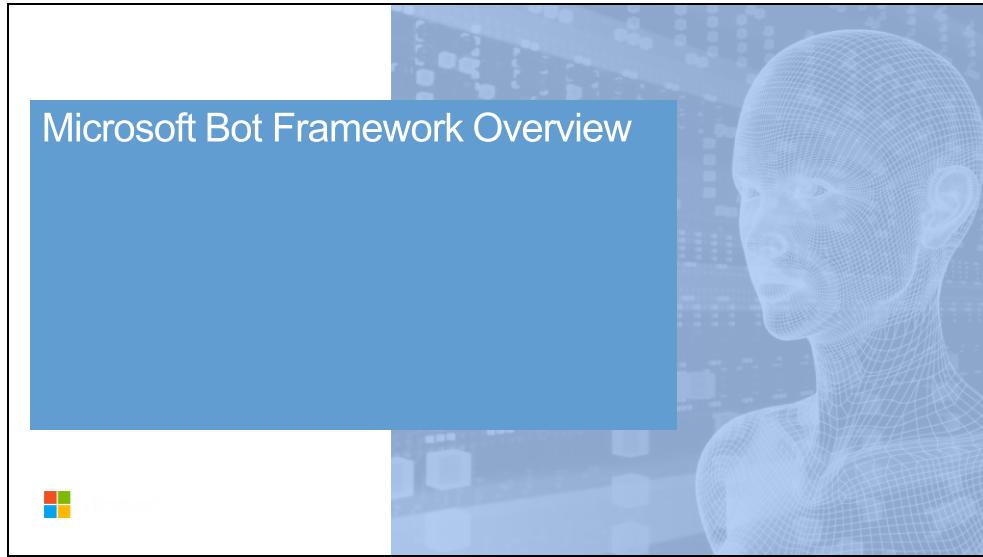
VSCode

Node.js with npm installed locally

Git bash

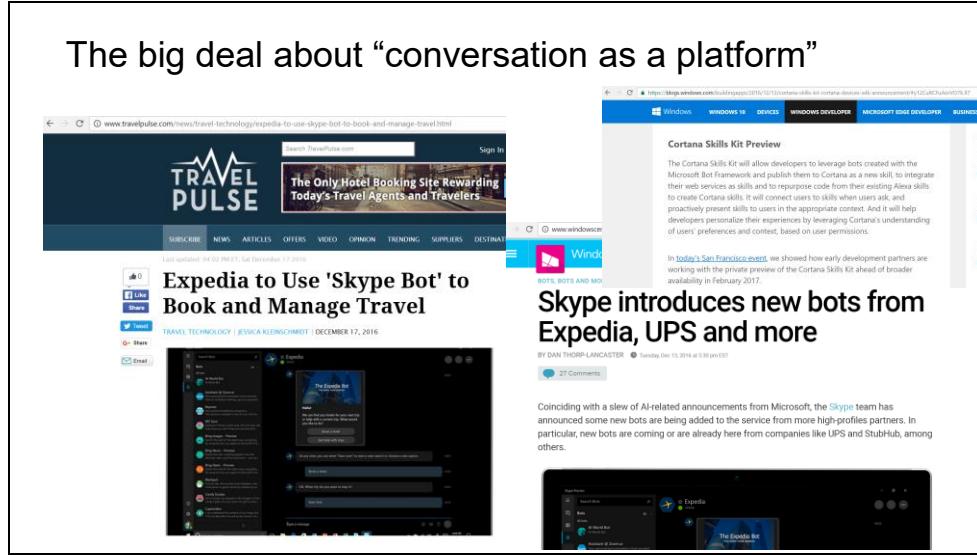
Bot Framework emulator

Azure account[recommended and needed to deploy]



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.

The big deal about “conversation as a platform”



The collage consists of three images arranged in a grid-like layout. The top-left image is a screenshot of the TravelPulse website, showing a news article titled "Expedia to Use 'Skype Bot' to Book and Manage Travel". The top-right image is a screenshot of the Windows Central website, showing a news article titled "Skype introduces new bots from Expedia, UPS and more". The bottom image is a screenshot of the Skype application interface, showing the "Expedia" bot in a conversation window.

<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/>

Demo

The ocrbot (Computer Vision bot)

Launch the ocrbot from the developer portal using Skype

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

What is a bot?



What a bot is not

It's not AI



It's not natural language processing only



It's not text interfaces only



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:

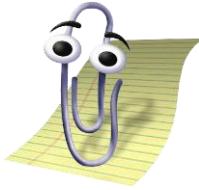
- Natural language processing has limitations, still. The more your bot depends on NLP, the worse the experience gets.
Hint: Typing isn't always the best option.
- Move away from NLP as quickly as possible
- "Drive" the user as much as you can (menus, choices, etc). Less typing = better

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 be the best experience for everything.
- Examples:
 - Skype allows audio and 3D bots as well.
 - Slack, Facebook and Skype have buttons/custom UIs

What is a bot?

Simply put, a bot is an **application** that performs an automated task. That's it.



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.

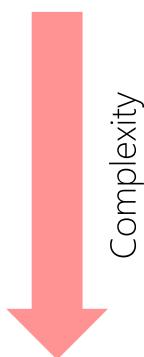
They can:

- Exist in different channels and across platforms.
- Do anything from simple task automation like taking food orders to leveraging sophisticated deep learning algs 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.

What types of bots are there?



- Notifiers
- Reactors
- Responders
- Conversationalist

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:

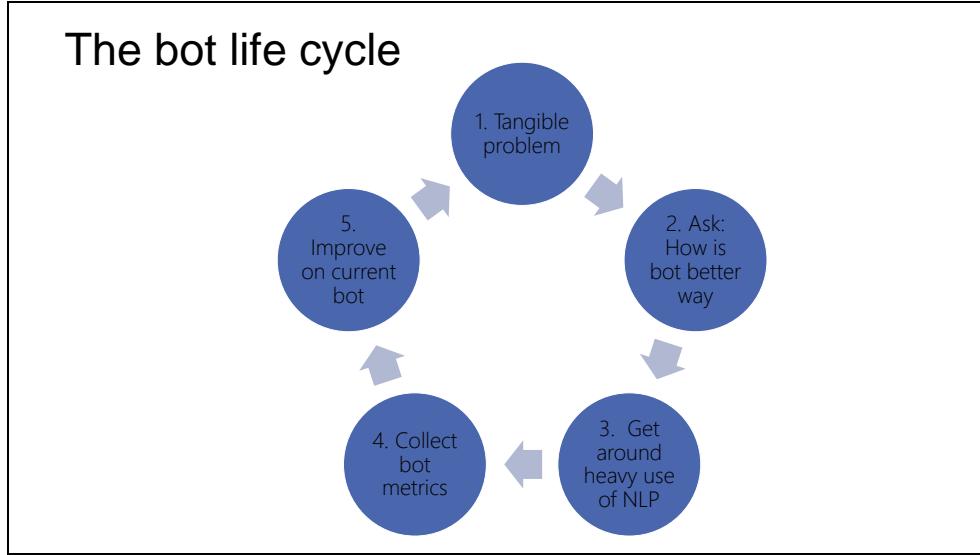
Do they react to messages?

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?



- 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



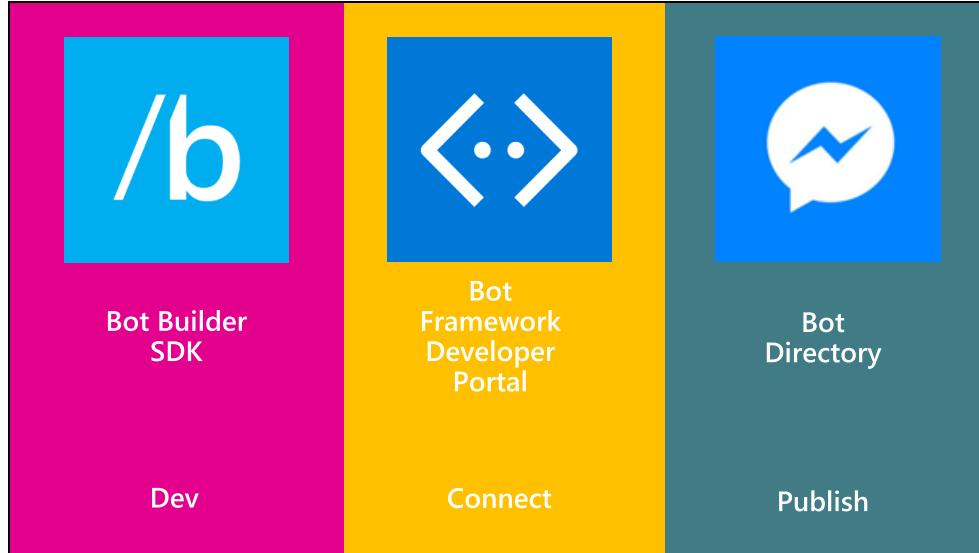
From: <https://docs.botframework.com/en-us/directory/best-practices>

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>

Here enters the Bot
Framework

What is the Bot Framework

A development tool and much more...



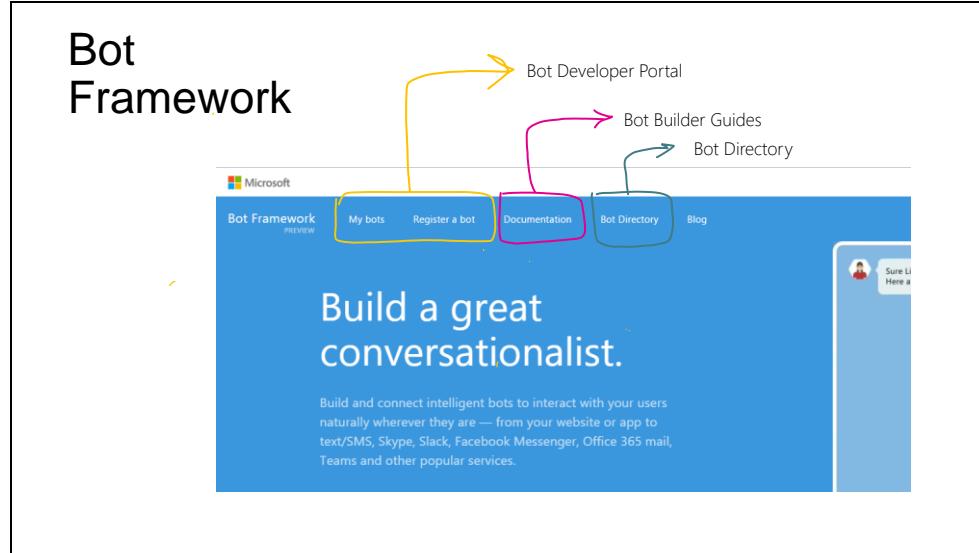
Bot Builder is itself a framework for building conversational applications ("Bots").

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>



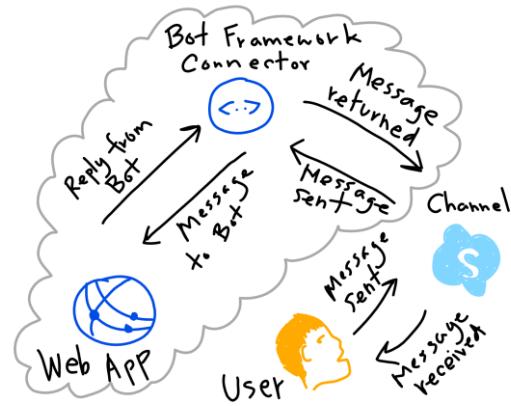
Main page: <https://botframework.com>

What is the Bot Framework

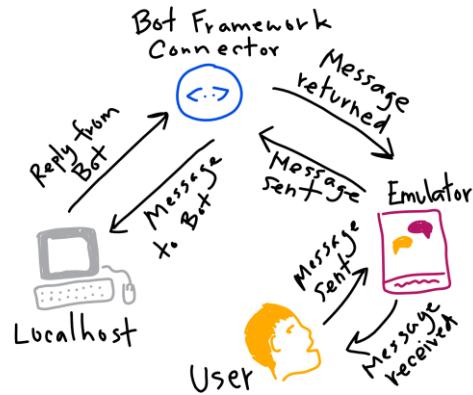
It's also a messaging platform

Let's get a picture of the way it works

What is the Bot Framework: as a cloud solution



What is the Bot Framework: as a development solution



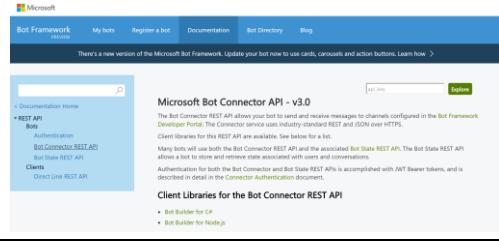
Bot Builder: Development Kits and REST

Bot Builder SDKs for:

- .NET framework for C#
- Node.js

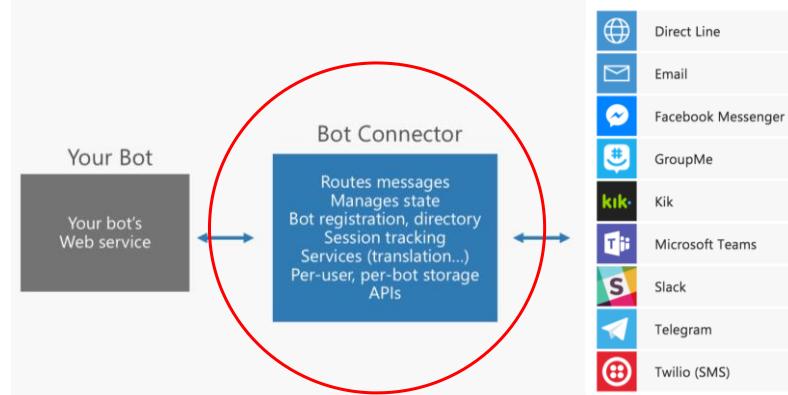
And there's also

- REST and REST State APIs



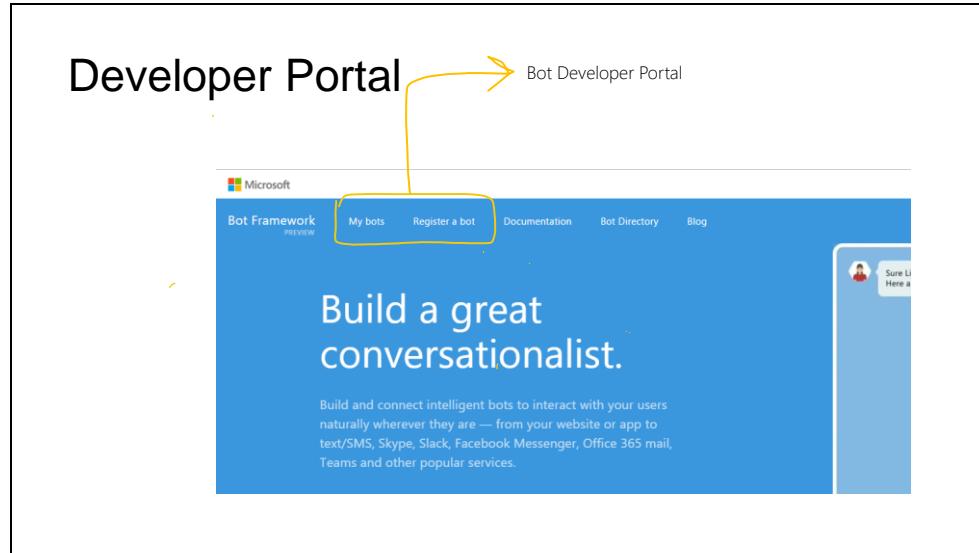
SDKs infographic: http://docs.botframework.com/en-us/images/faq-overview/bot_builder_sdk_july.png

Bot Connector (part of Bot Builder)



It's part of the Bot Builder SDK

<http://docs.botframework.com/en-us/csharp/builder/sdkreference/gettingstarted.html#channels>



Developer Portal: what registration does for you

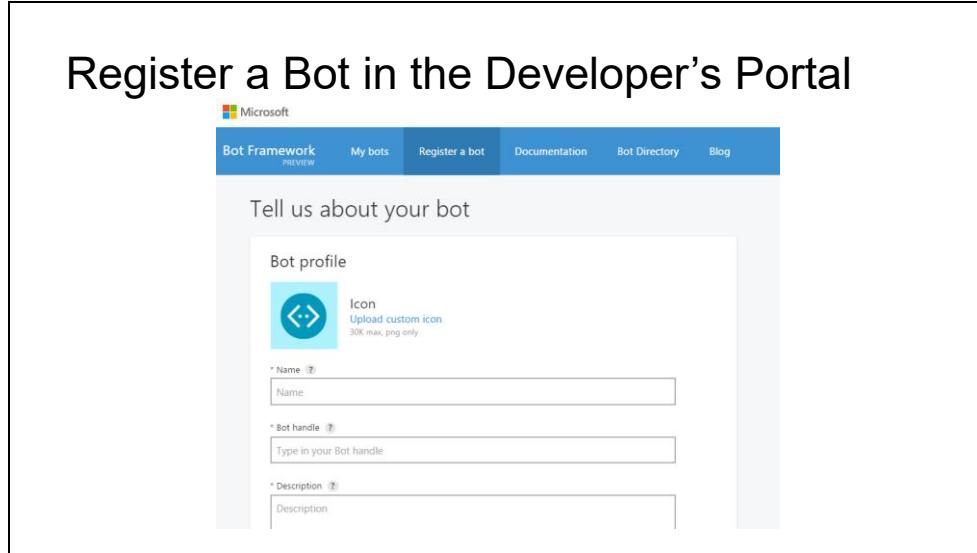
Your bot's web service in the cloud



MS Bot Framework Connector



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.



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.

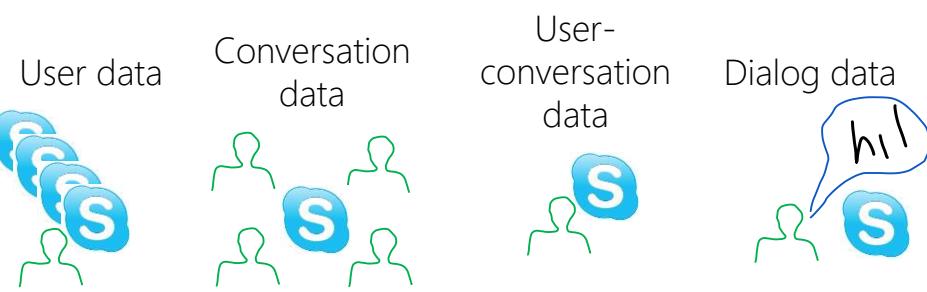
Go back and edit this profile anytime

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

State Service: types of bot data stored for us



- This data is currently stored for free for you within the Bot Framework State Service.
- However, you may bring in your own data source (format: key-value store)

User data – globally available for user across all conversations

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

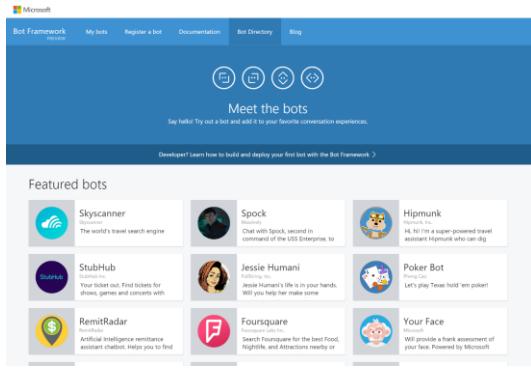
User-conversation data – stores globally conversation data for a user (But private to just that user)

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

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

The Benefits of the Bot Framework

• For developers

- Bots are more capable nowadays so more functions
- Bot Builder SDKs or custom code – you have choices
- Faster testing, dev and deployment
- Easy integration with the cloud
- Growing community

• For end users

- User choice of channels
- Users have trust and control of their data
- New experiences

• For businesses

- Broad access to their customers and new experience
- Reduced cost of development
- High quality bots

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

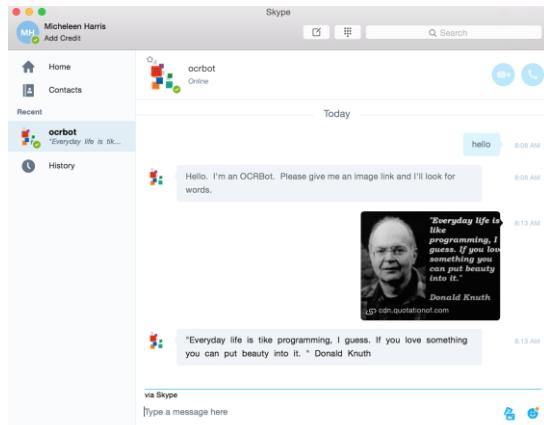
Broad access to their customers where they already are conversing giving them new experiences

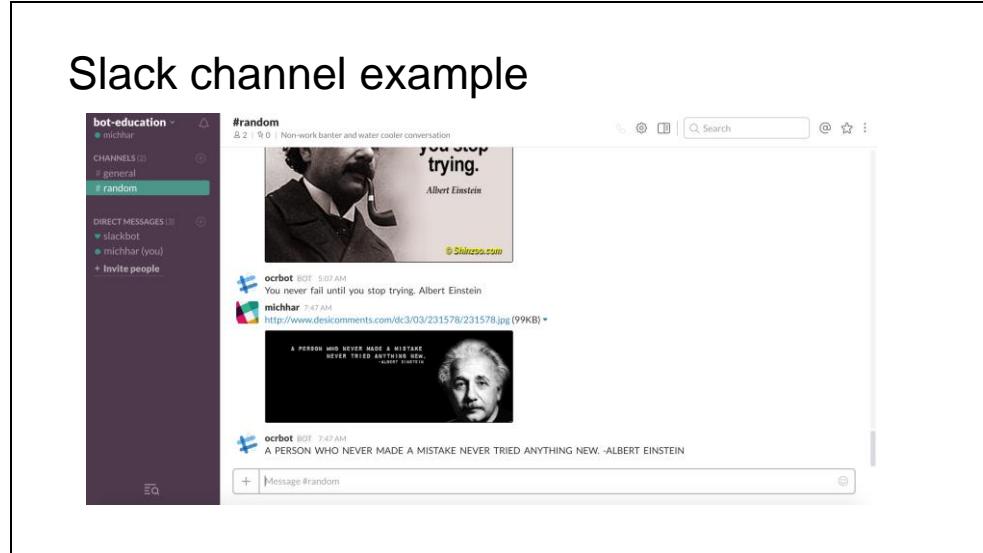
Reduced cost of development – just faster with SDKs and builtin functionality like dialog handling and language understanding

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

Working with channels

Skype channel example



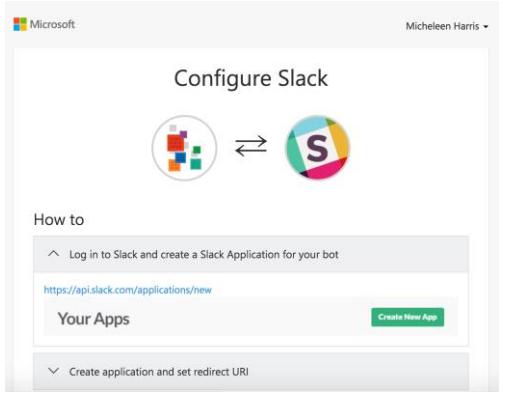


Adding a channel

- Skype is added by default
- Instructions are laid out

Add another channel

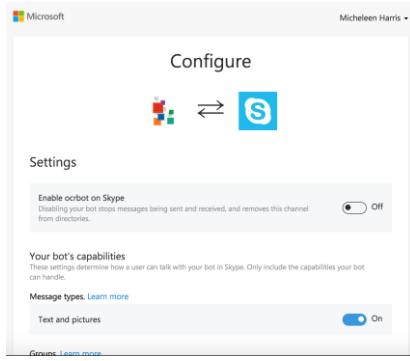
 Direct Line
 Email
 Facebook Messenger
 GroupMe
 Kik
 Microsoft Teams
 Slack
 Telegram
 Twilio (SMS)



The dialog box is titled 'Configure Slack' and shows a Microsoft Bot Framework icon connected to a Slack icon with a double-headed arrow. Below this, there's a 'How to' section with a link to 'Log in to Slack and create a Slack Application for your bot' (<https://api.slack.com/applications/new>) and a 'Create New App' button. There are also sections for 'Your Apps' and 'Create application and set redirect URI'.

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

Editing a channel



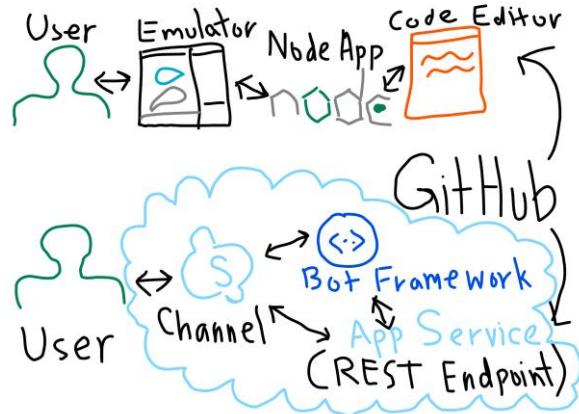
Skype for instance, through configuring we can:

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

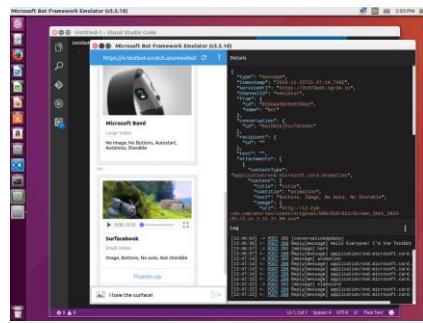
Toolbox

Let's see some tech stuff and code!

Dev process



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

Node App: get the Bot Builder Node.js module

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

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

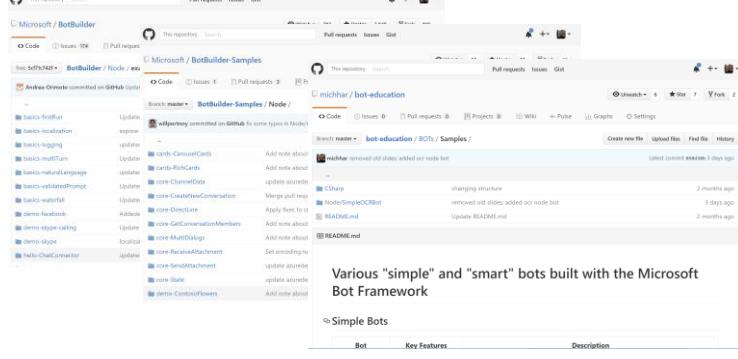
Or to clone the latest entire repository (big, but has samples as well):

```
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>

Tons of sample code on GitHub

On github from Microsoft (and elsewhere):



Various "simple" and "smart" bots built with the Microsoft Bot Framework

Bot	Key Features	Description
Simple Bots		

<https://github.com/Microsoft/BotBuilder-Samples>

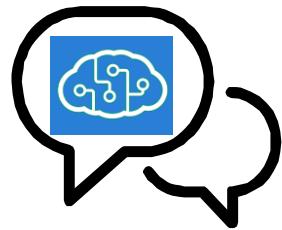
<https://github.com/Microsoft/BotBuilder/tree/master/CSharp/Samples>
<https://github.com/Microsoft/BotBuilder/tree/master/Node/examples>

<https://github.com/Azure/bot-education/tree/master/Student-Resources/BOTs>

<https://github.com/MicrosoftDX/botFramework-proactiveMessages>

More out there...so many, can't list...

Integration with cognitive services

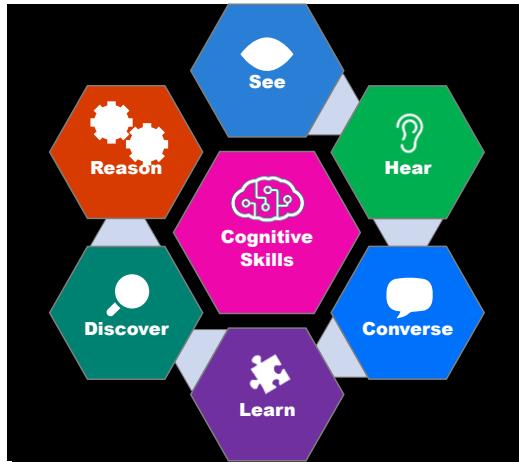


Cognitive Services meet Bots

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



Build smarter
experiences that
delight and engage
your users



Cognitive services vs. bots

We can have a bot
without cognitive
services.



We can have an
intelligent app
without a bot.



Features:

Feature Name	Value
Description	["Type":0,"Caption":1,"Text": "A bird is perched on the bark of a tree."]
Tags	[{"name": "tree", "confidence": 0.999999324028111}, {"name": "bird", "confidence": 0.999000317070279}, {"name": "wood", "confidence": 0.999999324028111}, {"name": "trunk", "confidence": 0.999999324028111}, {"name": "standing", "confidence": 0.999999324028111}, {"name": "perched", "confidence": 0.999999324028111}, {"name": "bark", "confidence": 0.999999324028111}, {"name": "wood", "confidence": 0.999999324028111}, {"name": "branch", "confidence": 0.999999324028111}, {"name": "resized", "confidence": 0.999999324028111}]
Image Format	JPG
Image Dimensions	1211 x 866
Clip Art Type	0 Non-clipart

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>

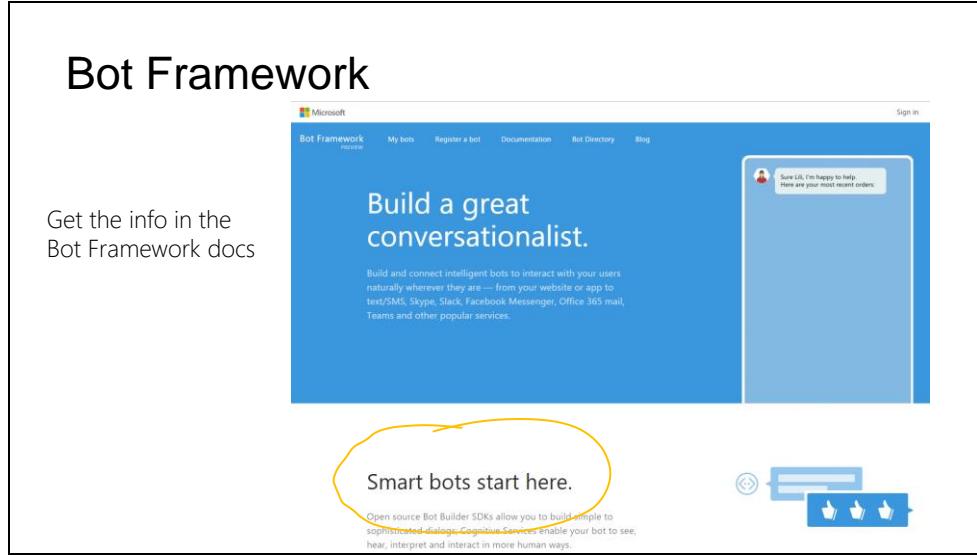
A few use cases

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

Captionbot.ai



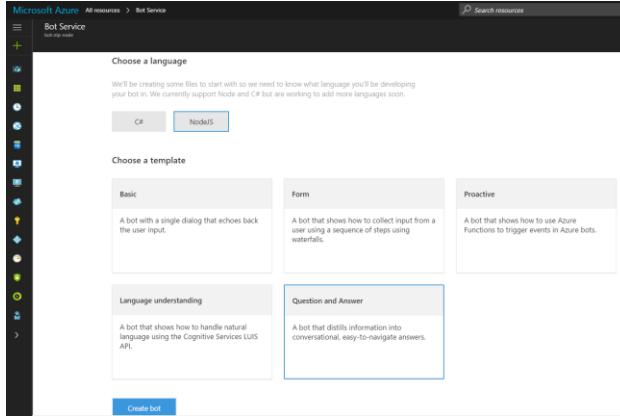
- Computer Vision API - Like CaptionBot.ai – a bot that reports back in human way the contents of an image
- Speech API - Authentication as a user speaks to the bot with a speaker verification profile or “voiceprint”
- LUIS API for analysis of queries such as “What is the weather in Toyko today?” using 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



Main doc page: <https://docs.botframework.com/en-us/>

Azure Bot Service

Bot Framework



From Azure Bot Service:

- Fire up LUIS bot
- Fire up QnA bot
- more templates...

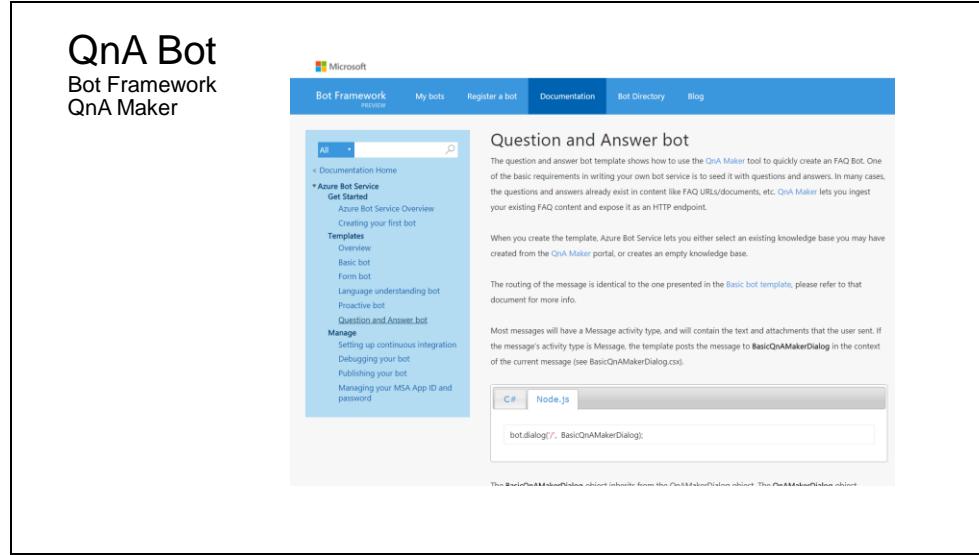
Create your first bot: <https://docs.botframework.com/en-us/azure-bots/build/first-bot/#navtitle>

Done through the Azure Portal

Options as of now for templates are:

- Basic
- Form
- Proactive
- Language understanding
- QnA

Slide 47



The screenshot shows the QnA Bot documentation page on the Bot Framework website. The page is titled "Question and Answer bot". It includes a sidebar with navigation links for "Templates" (Overview, Basic bot, Form bot, Language understanding bot, Proactive bot, Question and Answer bot) and "Manage" (Setting up continuous integration, Debugging your bot, Publishing your bot, Managing your MSA App ID and password). The main content area contains text about the QnA Maker tool and a code snippet for C# and Node.js.

Question and Answer bot

The question and answer bot template shows how to use the QnA Maker tool to quickly create an FAQ Bot. One of the basic requirements in writing your own bot service is to seed it with questions and answers. In many cases, the questions and answers already exist in content like FAQ URLs/documents, etc. QnA Maker lets you ingest your existing FAQ content and expose it as an HTTP endpoint.

When you create the template, Azure Bot Service lets you either select an existing knowledge base you may have created from the QnA Maker portal, or creates an empty knowledge base.

The routing of the message is identical to the one presented in the [Basic bot template](#), please refer to that document for more info.

Most messages will have a Message activity type, and will contain the text and attachments that the user sent. If the message's activity type is Message, the template posts the message to `BasicQnAMakerDialog` in the context of the current message (see `BasicQnAMakerDialog.cs`).

C# Node.js

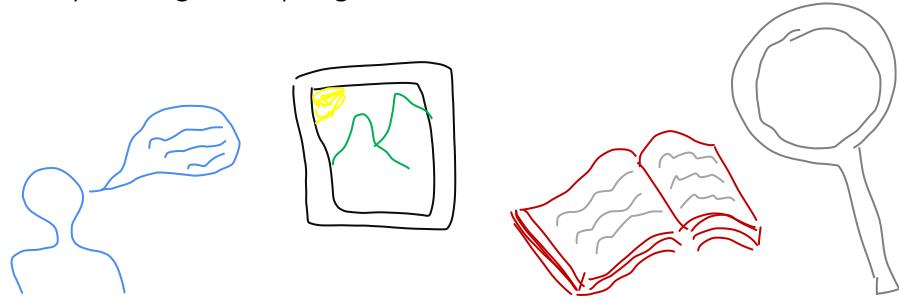
```
bot.dialog('/', BasicQnAMakerDialog);
```

The `BasicQnAMakerDialog` object inherits from the `QnAMakerDialog` object. The `QnAMakerDialog` object

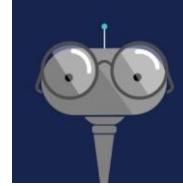
A QnA bot guide (can choose the SDK here): <https://docs.botframework.com/en-us/azure-bot-service/templates/qnamaker/#navtitle>

More integrations...

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



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>

Resources

Support	Contact
Technical FAQ	https://docs.botframework.com/en-us/technical-faq
Bot Builder SDK issues and suggestions	Use the issues tab on our github repo: https://github.com/Microsoft/BotBuilder/
Using a bot	Contact the bot's developer through their publisher e-mail
Community support	Use StackOverflow, with the hashtag #botframework
Reporting Abuse	Contact us at bf-reports@microsoft.com
Gitter discussion forum	https://gitter.im/Microsoft/BotBuilder

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

Lab

Bot dev environment

Instructions:

Questions?

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



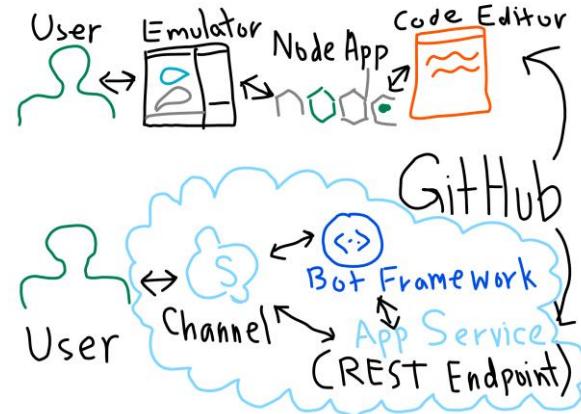
Bot Framework Developer Introduction and Deploying an Intelligent Bot



Getting started

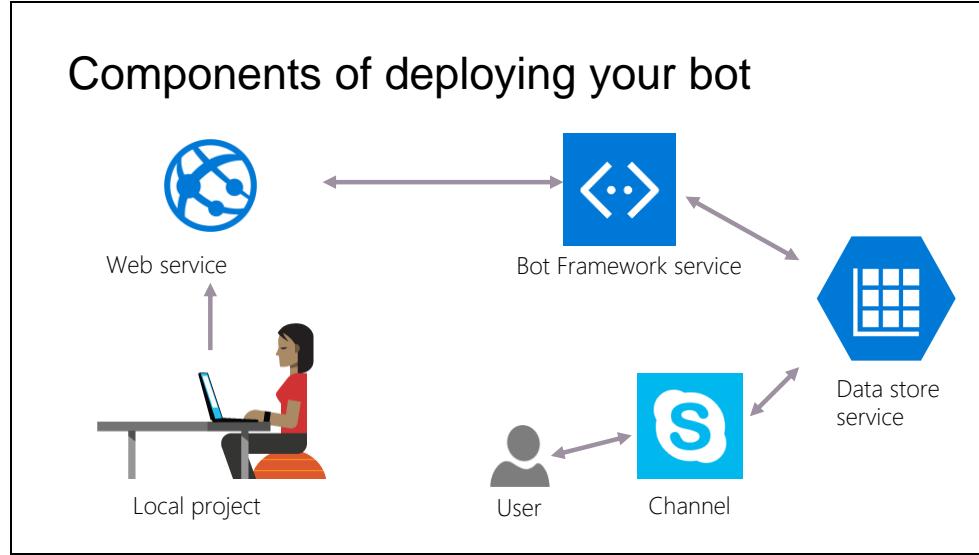
Core concepts: <https://docs.botframework.com/en-us/node/builder/guides/core-concepts/#navtitle>

Dev process/Toolbox



JS and Node.js primer

Whirlwind tour



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

Session outline (and steps to releasing a bot)

1. Develop
 2. Deploy
 3. Register
 4. Publish
 5. Test
- *Deploy a bot to Azure lab*

Develop

in code editor

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/>

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
    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

Here we are working with user data and message data

Concept and code: dialog stack

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

bot.dialog('/', [
  function (session) {
    session.beginDialog('/askName');
  },
  function (session, results) {
    session.send('Hello %s!', results.response);
  }
]);
bot.dialog('/askName', [
  function (session) {
    builder.Prompts.text(session, 'Hi! What is your name?');
  },
  function (session, results) {
    session.endDialogWithResult(results);
  }
]);
```

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

Here let's assume we've created our 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

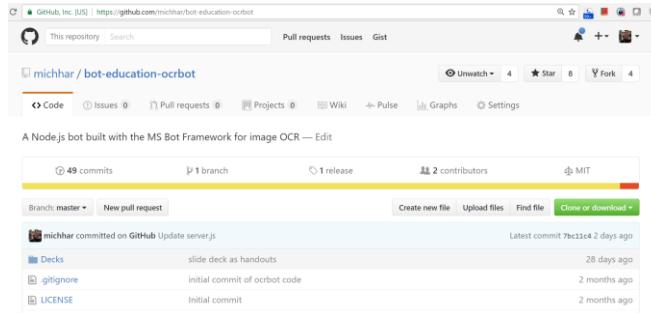
Deploy

to Azure

<https://docs.botframework.com/en-us/node/builder/guides/deploying-to-azure/#step-1-get-a-github-repo>

Step 1: Github repo

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

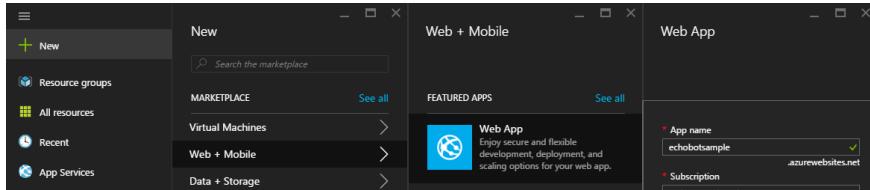


For example: <https://github.com/michhar/bot-education-ocrbot>

Either fork this repo or create your own and place your bot code in it (the server.js and any additional necessary files; I usually like to include the package.json)

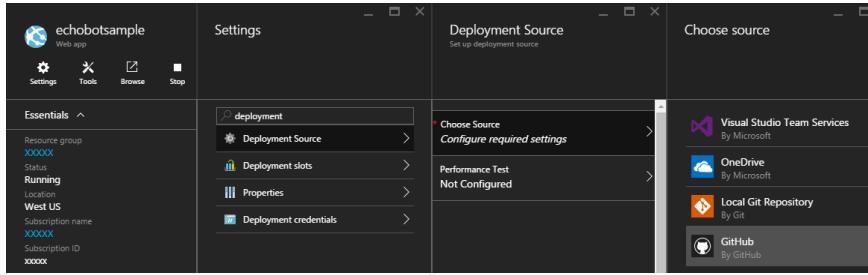
Note: there are other deployment methods

Step 2: Create an Azure Web App



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

Step 3: Set up for continuous deployment

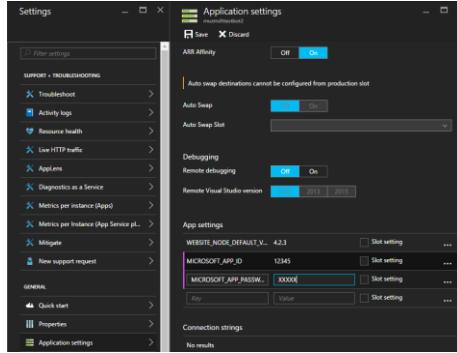


In azure portal: <https://portal.azure.com>

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.

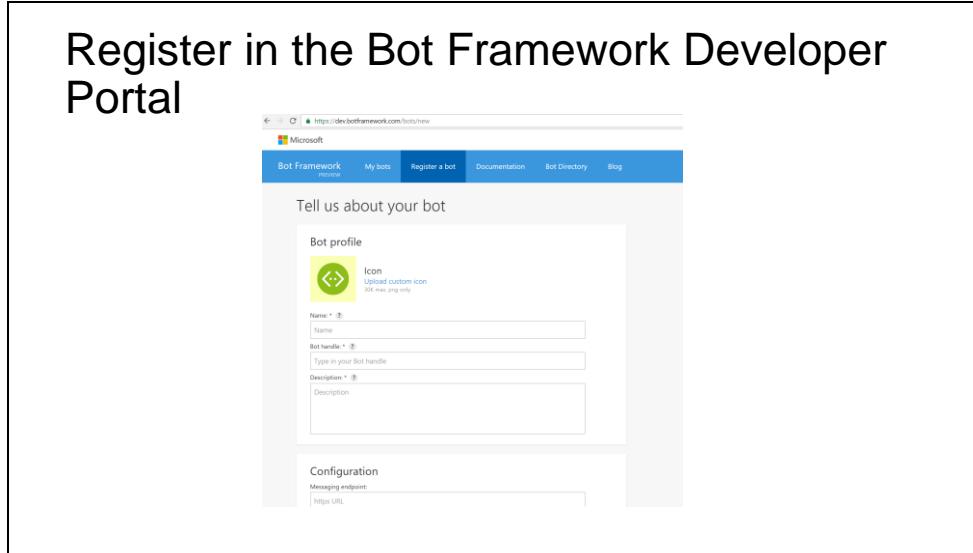
Step 4: Enter in App Id and App Password



This allows the password to be safe and guarded.

Register

in the Developer Portal

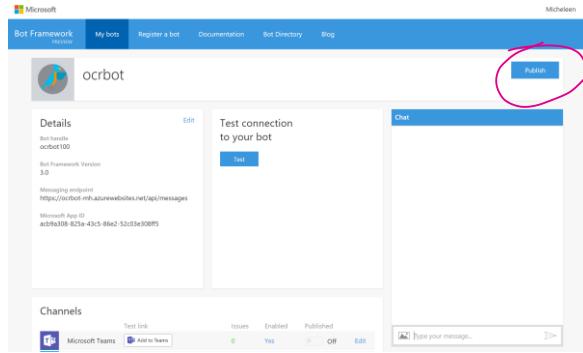


On <https://dev.botframework.com> click on "Register a bot"

Publish to the Bot Directory

In the Developer Portal

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**

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

Test connection
to your bot

Test

Accepted

Test connection with channels

Test bot with a pre-configured channels

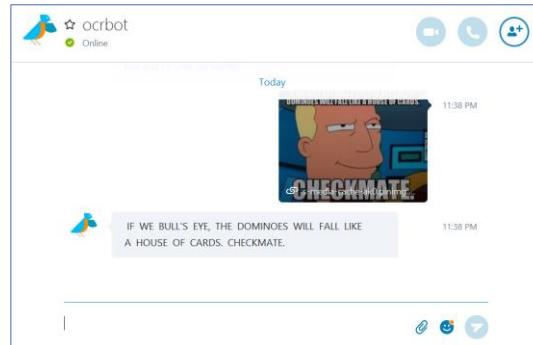
Channels

	Test link	Issues	Enabled	Published	
 Microsoft Teams	Add to Teams	0	Yes	<input type="radio"/> Off	Edit
 Skype	Add to Skype	0	Yes	<input type="radio"/> Off	Edit
 Web Chat		0	Yes	<input type="radio"/> Off	Edit

[Get bot embed codes](#)

Test connection with a channel

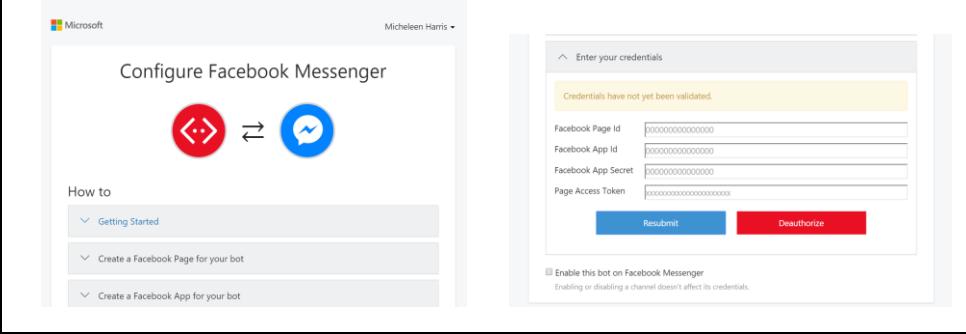
Add bot to
contacts in Skype
and began a
chat...



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

Adding a channel

Many channels will require your credentials as a developer on the service, e.g., Facebook channel



The screenshot shows the 'Configure Facebook Messenger' page. At the top, there's a Microsoft logo and a dropdown menu for 'Micheleen Harris'. The main title is 'Configure Facebook Messenger' with a 'Configure' button. Below the title is a 'How to' section with three items: 'Getting Started', 'Create a Facebook Page for your bot', and 'Create a Facebook App for your bot'. To the right is a 'Enter your credentials' form. It contains four text input fields: 'Facebook Page Id' (0000000000000000), 'Facebook App Id' (0000000000000000), 'Facebook App Secret' (0000000000000000), and 'Page Access Token' (0000000000000000). Below the fields are 'Resubmit' and 'Deauthorize' buttons. A note at the bottom says 'Enable this bot on Facebook Messenger' with a checkbox and a small note: 'Enabling or disabling a channel doesn't affect its credentials.'

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

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!

Questions?

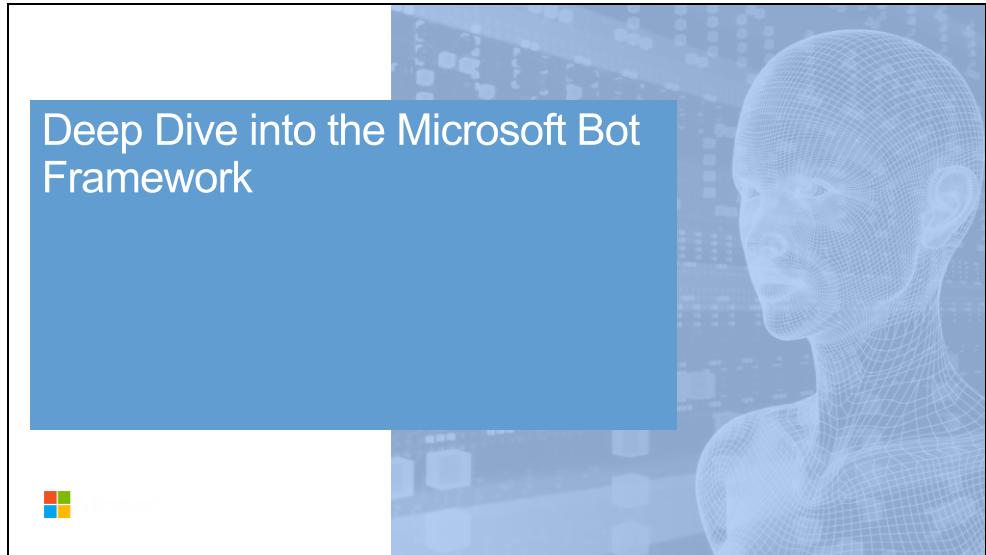
Also, try the course gitter chatroom at
aka.ms/botedu-discuss



Deploy a bot to Azure

For Skype and Teams

Follow along with this article: https://michhar.github.io/notes_posts/ocrbot-makes-a-connection



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.

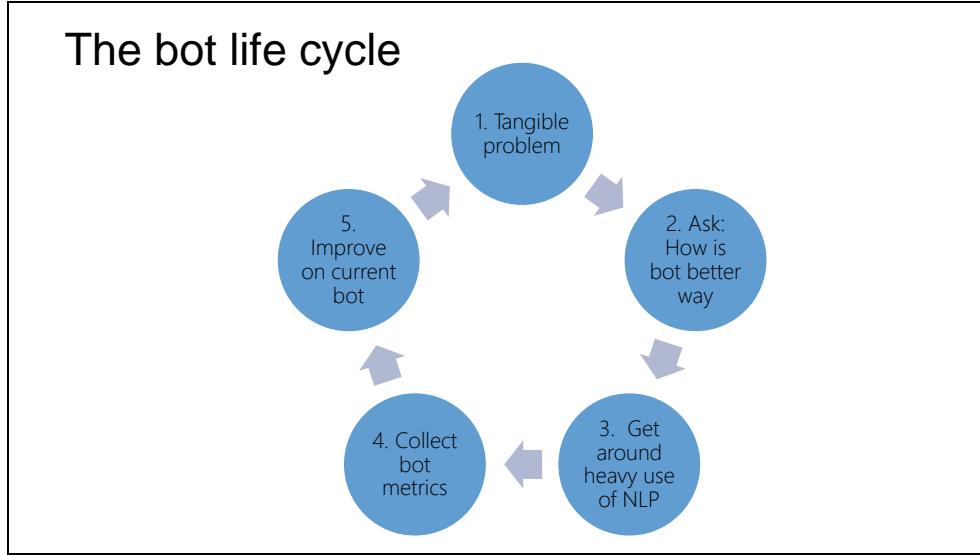
Session outline

- Overview refresher
- 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*

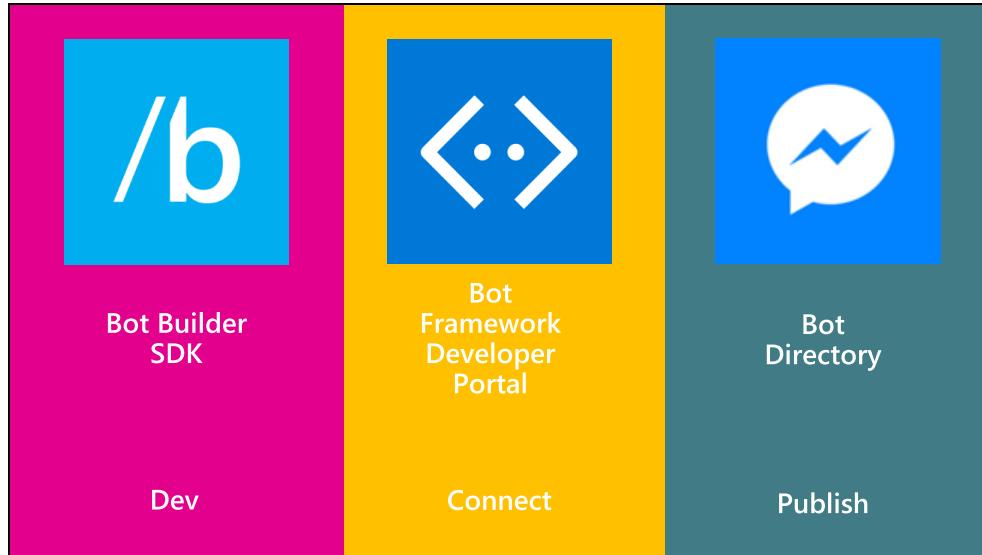
aka.ms/botedu

aka.ms/bot-discuss

Overview Refresher



- 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



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.

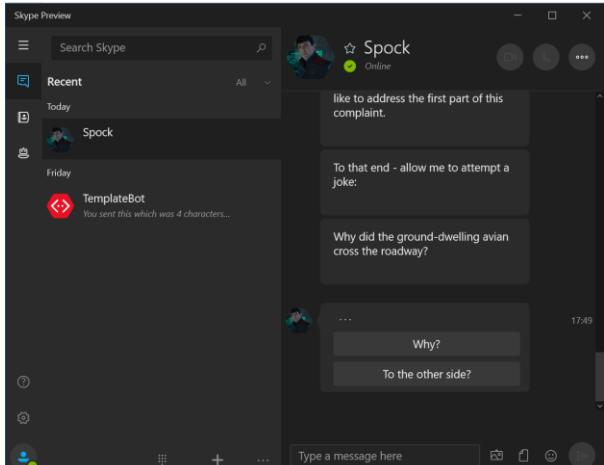
The Bot Framework Developer Portal lets you connect your bot(s) seamlessly text/sms to Skype, Slack, Facebook Messenger, Kik, Office 365 mail and other popular services. Register, configure and publish.

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>

Bot Builder

An open source SDK for .NET, Node.js and a REST API



The screenshot shows a Skype Preview window. At the top, there's a search bar and a contact list with 'Spock' marked as 'Online'. The main area shows a conversation with Spock. Spock has sent a message: 'like to address the first part of this complaint.' The user has responded: 'To that end - allow me to attempt a joke:'. Spock then sends a joke: 'Why did the ground-dwelling avian cross the roadway?'. The user replies with a poll: 'Why?' with options 'To the other side?' and '...'. The poll has 17 votes and 49 responses. At the bottom, there's a text input field 'Type a message here' and various Skype interface buttons.

my notes:

showing a conversation with a publicly registered bot called Spock. It has dialogs and attachments with buttons as a menu as we'll see later.

Bot Builder

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

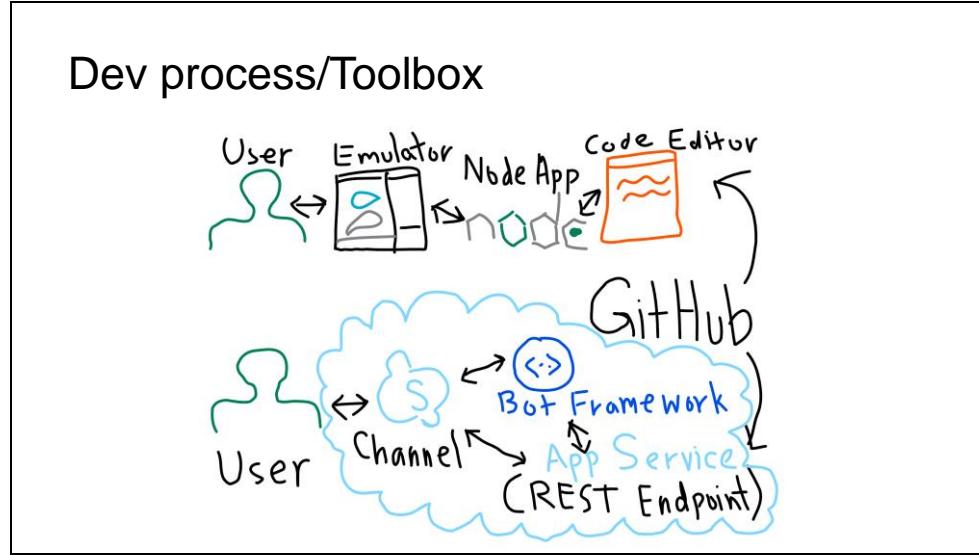
You make any bot app
(anywhere from a simple text-command app to one rich with NLP)



Bot Builder gives you
the features needed
to manage
conversations

my notes:

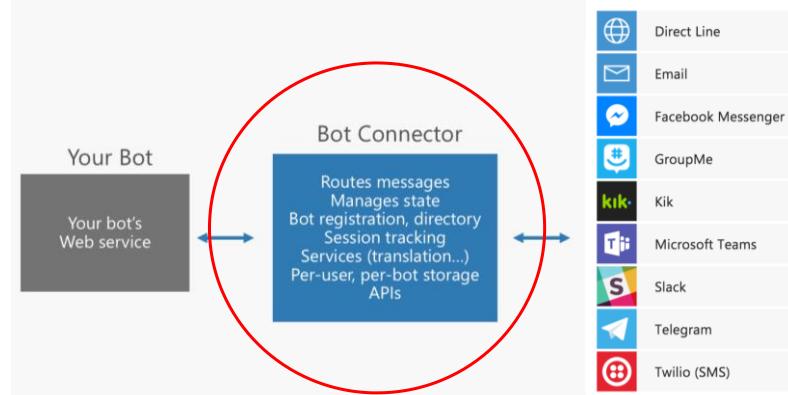
a bot is just an application that addresses the huge effort involved in repetitive tasks (a bot can generalize a problem – provide a fun interface to a game or experience for instance, or collect votes over a choice of opinions – as a bot it's a deployable and repeatable solutions, plus hopefully an enjoyable user experience)



Releasing a bot:

- Develop
- Deploy
- Register
- Publish
- Test

Bot Connector (part of Bot Builder)



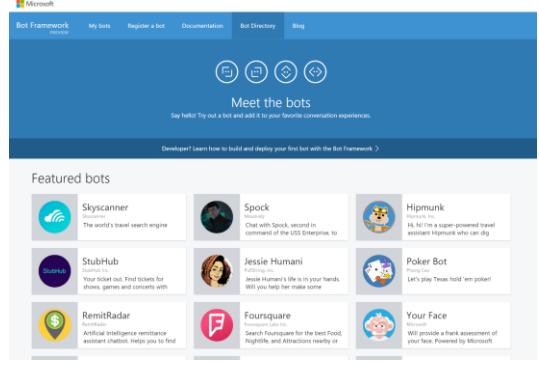
It's part of the Bot Builder SDK

<http://docs.botframework.com/en-us/csharp/builder/sdkreference/gettingstarted.html#channels>

Bot Directory

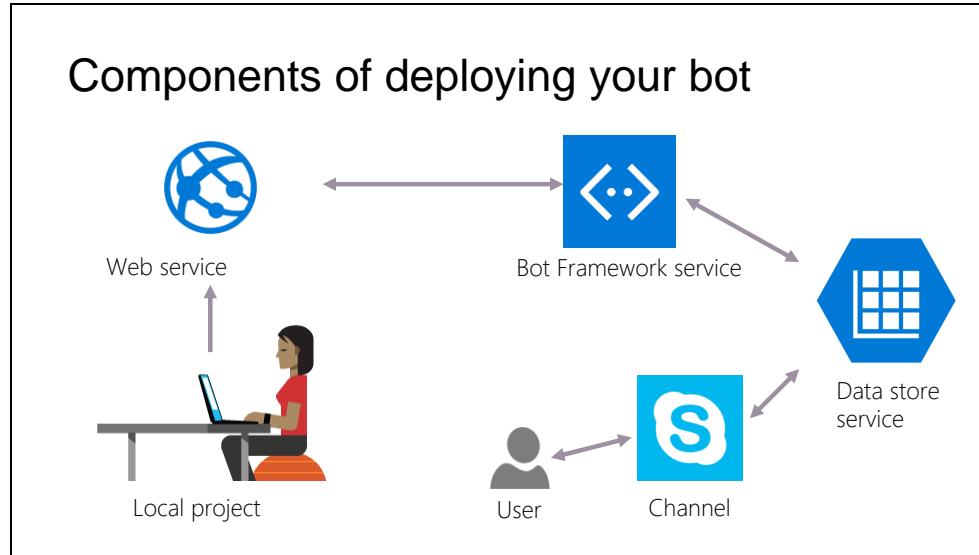
Public Directory of Bot Framework Bots

- 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



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

Basics

Code snippets

my notes:

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

Getting messages, sending replies

At the very least the **post** method is responsible for returning a “task” which in turn is responsible for sending replies

```
// Instantiate the chat connector to route messages
var connector = new builder.ChatConnector(botConnectorOptions);

// Handle Bot Framework messages with a restify server
var server = restify.createServer();
server.post('/api/messages', connector.listen());
server.listen(process.env.port || process.env.PORT || 3978, function () {
  console.log('%s listening to %s', server.name, server.url);
});|
```

Note the explicit creation of the connector client

Getting messages, sending replies

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

```
// Think of a sandwich (the bread is the root dialog and meat/cheese the 'askName' dialog)
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.

Conversation exchange

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

```
// 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);
  }
]);
```

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.

Wait for user input

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 a wide range of built-in prompt types.

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

Conversations and their key features

Overview

Conversation – dialog or stack of dialogs handled by a model

Key feature types in each builder SDK

.NET SDK

1. IDialogContext
2. IDialog
3. Fluent chained dialogs or FormFlow
4. State stores
5. PromptDialog

Features

1. Conversation model
2. Conversation management
3. Conversation management over steps
4. Persistent state data
5. Wait for user input

Node.js SDK

1. Session
2. Dialog handlers
3. Waterfalls
4. State bags
5. Prompts

Both interact with the [Connector Service](#) and we will start with a discussion of this service.

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

Conversation management

Dialog handlers

Conversation exchange

Spock

Hello Micheleen Harris!

Skype Certified

Massively

(The dialog is now suspended until there is a message from Micheleen, the user)...

(The dialog is now resumed and the bot can call a child dialog or send a message back)...

hi

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

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>

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>

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>

Dialogs and handlers

Conversational exchange elements

- Dialog object
- Closure
- Waterfalls
- 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)

Dialog handler example

bot.dialog
object
handles the
message

A child dialog
is made

```
// Think of a sandwich (the bread is the root dialog and meat/cheese the 'askName' dialog)
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>

Closure

Single
function
(one-step
“waterfall”)
– generally a
loop

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

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

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.

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.

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.

With waterfalls you drive the conversation by taking an action that moves the waterfall from one step to the next.

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

Graph dialog bot – an extension

Extending Microsoft's Bot Framework with Graph Based Dialogs

NOV 11, 2016

The problem is that most types of conversation do not look like a waterfall, but more like a graph. Each interaction with the user is represented as a node, and, according to the user's input, the conversation should be able to flow to different nodes in that graph. This flow can be implemented using only the Bot Framework's APIs, but it is not a straightforward or easily reusable process to implement.

Opportunities for Reuse

The `bot-graph-dialog` extension can be plugged in to enhance or completely recreate any new or existing Node.js bot running on top of the Bot Framework. It can be used to dynamically load dialogs from external data sources and embed these dialogs in any of the existing APIs that get a waterfall array of steps. Also, since the extension itself is extendable, it can be extended with custom types of nodes, with external handlers and much more.

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

Conversation model

Sessions or contexts

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)

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

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 Space Needle is an observation tower in Seattle, Washington, a landmark of the Pacific Northwest, and an icon of Seattle")
            .images([
                builder.CardImage.create(session,
                    "https://upload.wikimedia.org/wikipedia/commons/2/2d/Space_Needle_at_night.jpg")
            ])
            .tap(builder.CardAction.openUrl(session, "https://en.wikipedia.org/wiki/Space_Needle"))
    ]);
```

Build the message



Hero Card

Space Needle

The **Space Needle** is an observation tower in Seattle, Washington, a landmark of the Pacific Northwest, and an icon of Seattle.

Card as reply



Dialog lab

Building dialogs

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

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:

State storage

Managed State API and bags

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

State Service: types of bot data stored for us (part of Bot Connector)

User data



Conversation data



User-conversation data



Dialog data



- This data is currently stored for free for you within the Bot Framework State Service.
- However, you may bring in your own data source (format: key-value store)

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.

Storage behind the scenes

Bring your own:
Key-Value based storage



Bot Framework state service:
Azure Table Storage



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. You don't really need to worry about the rest, just take note that table storage is used and you can bring your own if need-be. The Bot Framework manages this default storage for you so you can maintain a stateless bot experience.

User input

Prompts
Recognizers

<https://docs.botframework.com/en-us/node/builder/chat/prompts/#navtitle>

Built-in prompts in both SDKs

Classes

class PromptAttachment Prompt for an attachment. More...	← C#
class PromptChoice Prompt for a choice from a set of choices. More...	
class PromptConfirm Prompt for a confirmation. More...	
class PromptDouble Prompt for a double. More...	
class PromptInt64 Prompt for a confirmation. More...	
class PromptString Prompt for a text string. More...	

Node.js →

Prompt Type	Description
<code>Prompts.text</code>	Asks the user to enter a string of text.
<code>Prompts.confirm</code>	Asks the user to confirm an action.
<code>Prompts.number</code>	Asks the user to enter a number.
<code>Prompts.time</code>	Asks the user for a time or date.
<code>Prompts.choice</code>	Asks the user to choose from a list of choices.
<code>Prompts.attachment</code>	Asks the user to upload a picture or video.

my notes:

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

Additional elements

Localization
Calling
Extensions

<https://docs.botframework.com/en-us/node/builder/chat/prompts/#navtitle>

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>)

Intents lab part 1

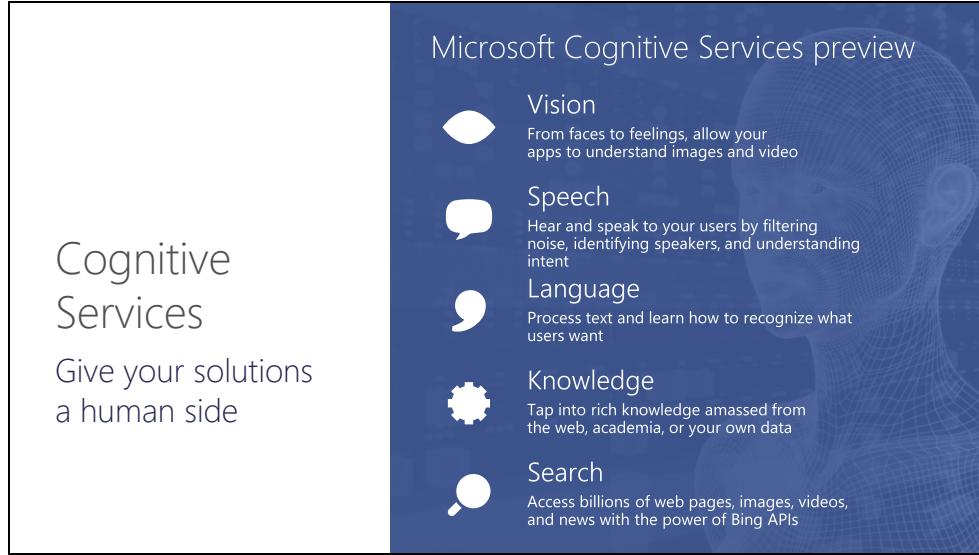
Building a bot to handle user intents

Update this bot () to include a game where the bot picks a number from 1 to 10 and the user has a three tries to guess it correctly. Use intent dialogs and matching, as well as, waterfalls and begin and end dialog logic.

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>

Adding intelligence to your
bot



The slide features a white header section on the left and a dark blue footer section on the right. The header contains the text 'Cognitive Services' and 'Give your solutions a human side'. The footer is titled 'Microsoft Cognitive Services preview' and lists five services: Vision, Speech, Language, Knowledge, and Search, each with a small icon and a brief description.

Microsoft Cognitive Services preview	
	Vision From faces to feelings, allow your apps to understand images and video
	Speech Hear and speak to your users by filtering noise, identifying speakers, and understanding intent
	Language Process text and learn how to recognize what users want
	Knowledge Tap into rich knowledge amassed from the web, academia, or your own data
	Search Access billions of web pages, images, videos, and news with the power of Bing APIs

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.

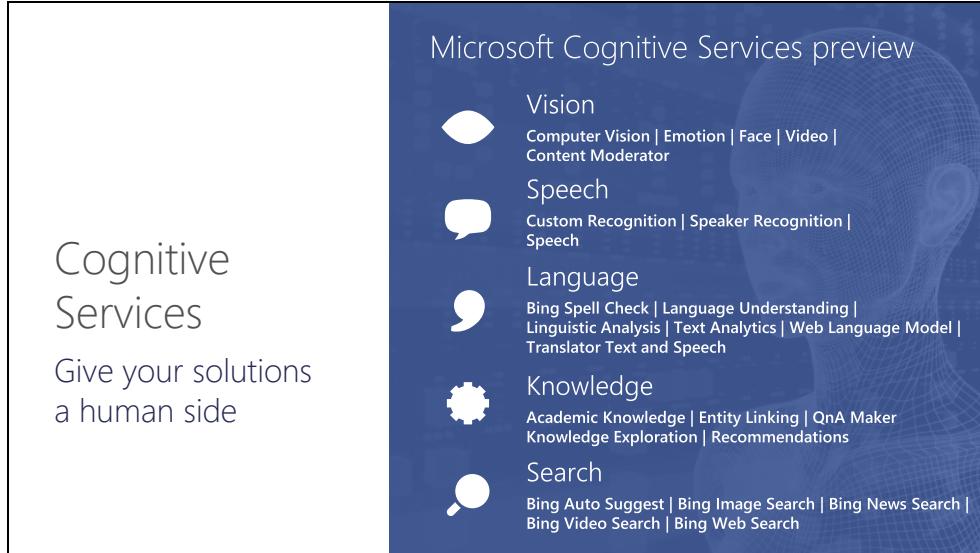
The technology used in our APIs is the same technology that powers our products today. And so, when you think of things like the Bing search APIs, it's the same technology from Bing.

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.



The slide features a white header section on the left and a dark blue footer section on the right. The header contains the text 'Cognitive Services' and 'Give your solutions a human side'. The footer is titled 'Microsoft Cognitive Services preview' and lists five service categories: Vision, Speech, Language, Knowledge, and Search, each with a corresponding icon and a list of APIs.

Microsoft Cognitive Services preview	
	Vision Computer Vision Emotion Face Video Content Moderator
	Speech Custom Recognition Speaker Recognition Speech
	Language Bing Spell Check Language Understanding Linguistic Analysis Text Analytics Web Language Model Translator Text and Speech
	Knowledge Academic Knowledge Entity Linking QnA Maker Knowledge Exploration Recommendations
	Search Bing Auto Suggest Bing Image Search Bing News Search Bing Video Search Bing Web Search

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.

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);
};

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

Intents lab part 2

Building a bot with language understanding

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>

Rate this tutorial!



<https://aka.ms/botedu-survey>

Non-linear waterfall lab

Leveraging the bot-trees graph bot

<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.

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>

Questions?

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

