

# Artificially Intelligent Bot Application with Language understanding & Cloud Computing

---

1/25/2017

Subtitle:

Chat Bots for Process Automation

Project Supervisor:

Prof. Dr.-Ing., Dr. rer. pol. Herbert Nosko

Prepared by:

Muhammad Ehsan-Ul-Haq / 1098587

Frankfurt University of Applied  
Sciences

## Abstract

An **Internet bot**, also known as **web robot**, **WWW robot** or simply **bot**, is a software application that runs automated tasks (scripts) over the Internet. Typically, bots perform tasks that are both simple and structurally repetitive, at a much higher rate than would be possible for a human alone. The largest use of bots is in web spidering (*web crawler*), in which an automated script fetches, analyzes and files information from web servers at many times the speed of a human.

Some bots communicate with other users of Internet-based services, via instant messaging (IM), Internet Relay Chat (IRC), or another web interface such as Facebook Bots, Skype Bots and Twitter Bots. These chatterbots may allow people to ask questions in plain English and then formulate a proper response. These bots can often handle many tasks, including reporting weather, zip-code information, sports scores, converting currency or other units, etc. Others are used for entertainment, such as SmarterChild on AOL Instant Messenger and MSN Messenger.

# Table of Contents

Topic	Page
<b>1 Part 1: Bots</b>	<b>2</b>
1.1 Introduction	
1.2 Examples of Chat Bots	
1.3 Why Chatbots are such a big opportunity?	
1.4 How Chatbots works?	
1.5 Available frameworks to build bots	
1.6 Available cloud platforms	
<b>2 Part 2: Pizza Bot</b>	<b>9</b>
2.1 Introduction	
2.2 Build Process	
2.2.1 Getting Started	
2.2.2 Publishing your bot to Microsoft Azure	
2.2.3 Registering your bot with Microsoft bot framework	
<b>3 Part 3: Project Code</b>	<b>22</b>
3.1 Folder Structure	
3.2 MessagesController.cs	
3.3 Pizza.cs	
3.4 PizzaOrderDialog.cs	
<b>4 Part 4: Few famous bots</b>	<b>29</b>
3.1 Wall street journal bot	
3.2 Besure Messenger	
3.3 Fyndy's Fify	
3.4 Poncho	
3.5 HealthTap	
<b>5 References</b>	<b>32</b>

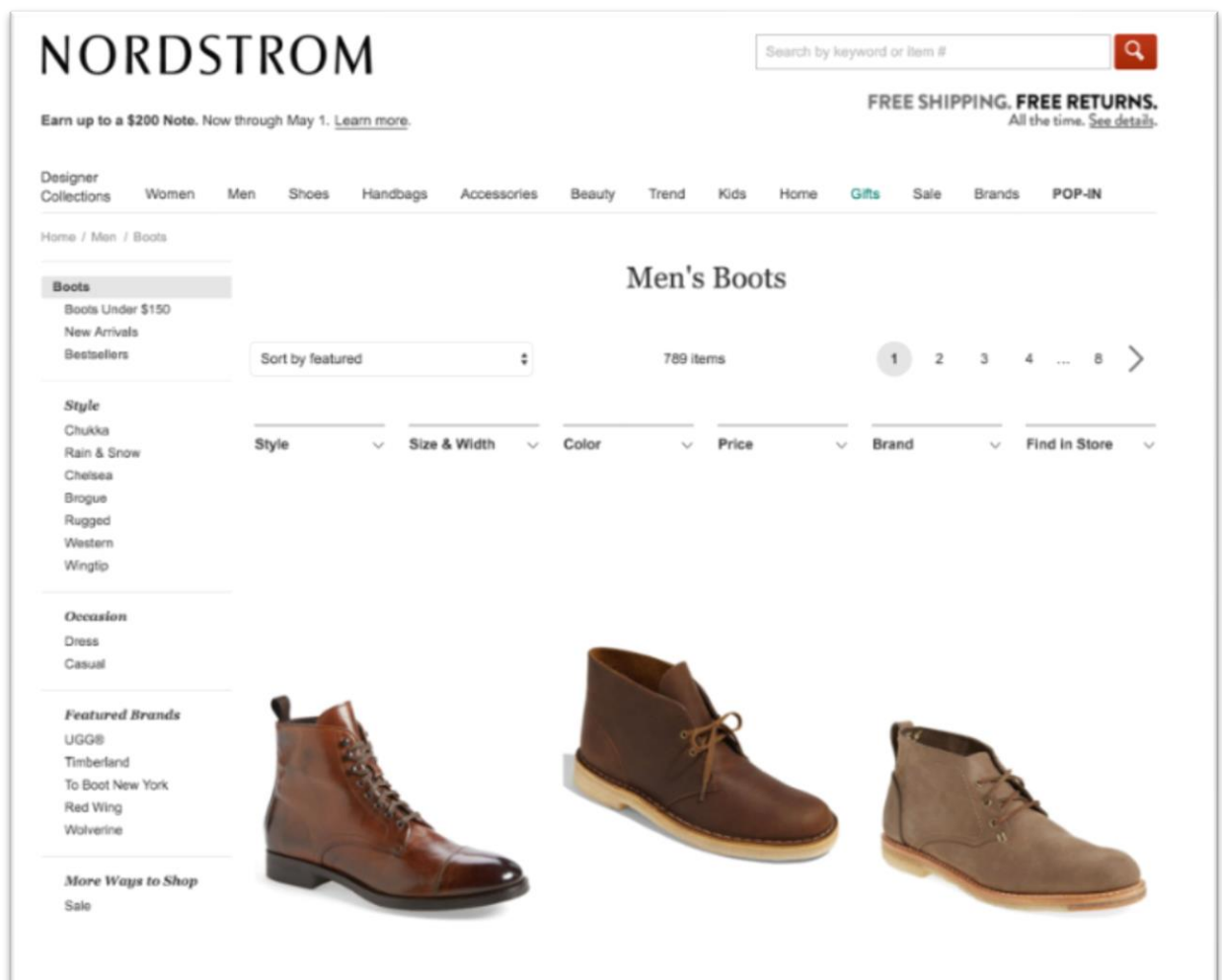
# Part 1: Bots

## 1.1 Introduction

A chatbot is a service, powered by rules and sometimes artificial intelligence, that you interact with via a chat interface. The service could be any number of things, ranging from functional to fun, and it could live in any major chat product (Facebook Messenger, Slack, Telegram, Text Messages, etc.). [1]

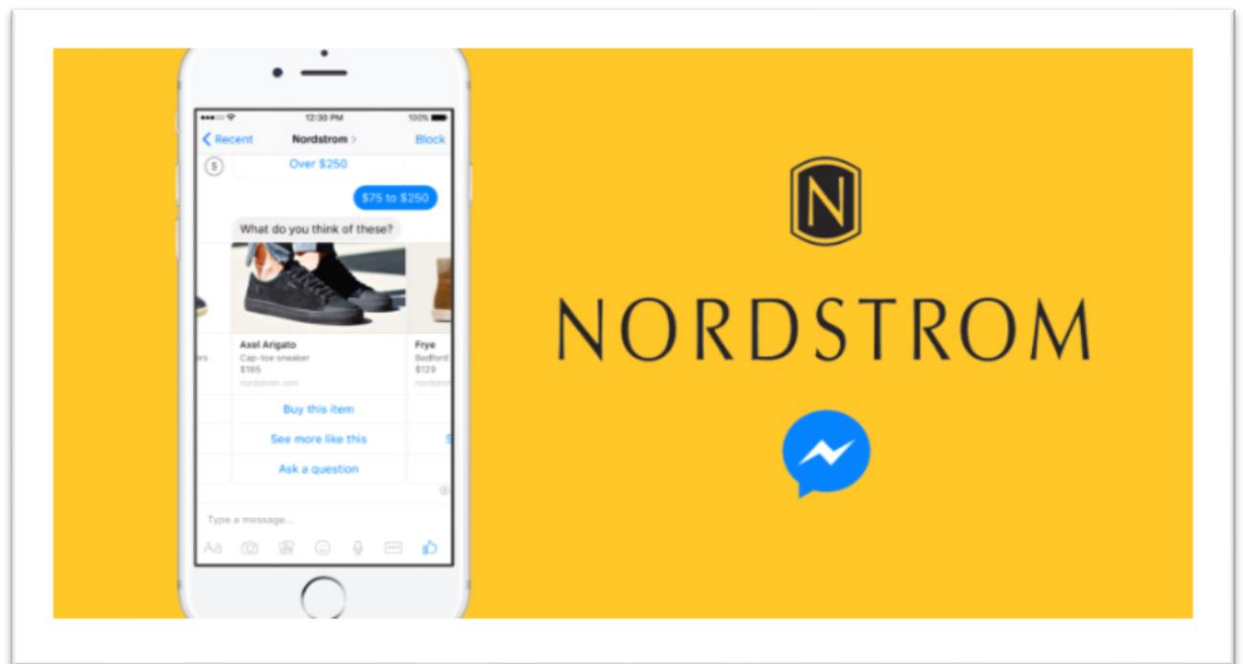
### Example:

If you wanted to buy shoes from Nordstrom online, you would go to their website, look around until you find the shoes you wanted, and then you would purchase them.



If Nordstrom makes a bot, which I'm sure they will, **you would simply be able to message Nordstrom on Facebook.** It would ask you what you're looking for and you would simply... tell it.

Instead of browsing a website, you will have a conversation with the Nordstrom bot, **mirroring the type of experience you would get when you go into the retail store.**



## 1.2 Examples of Chat Bots

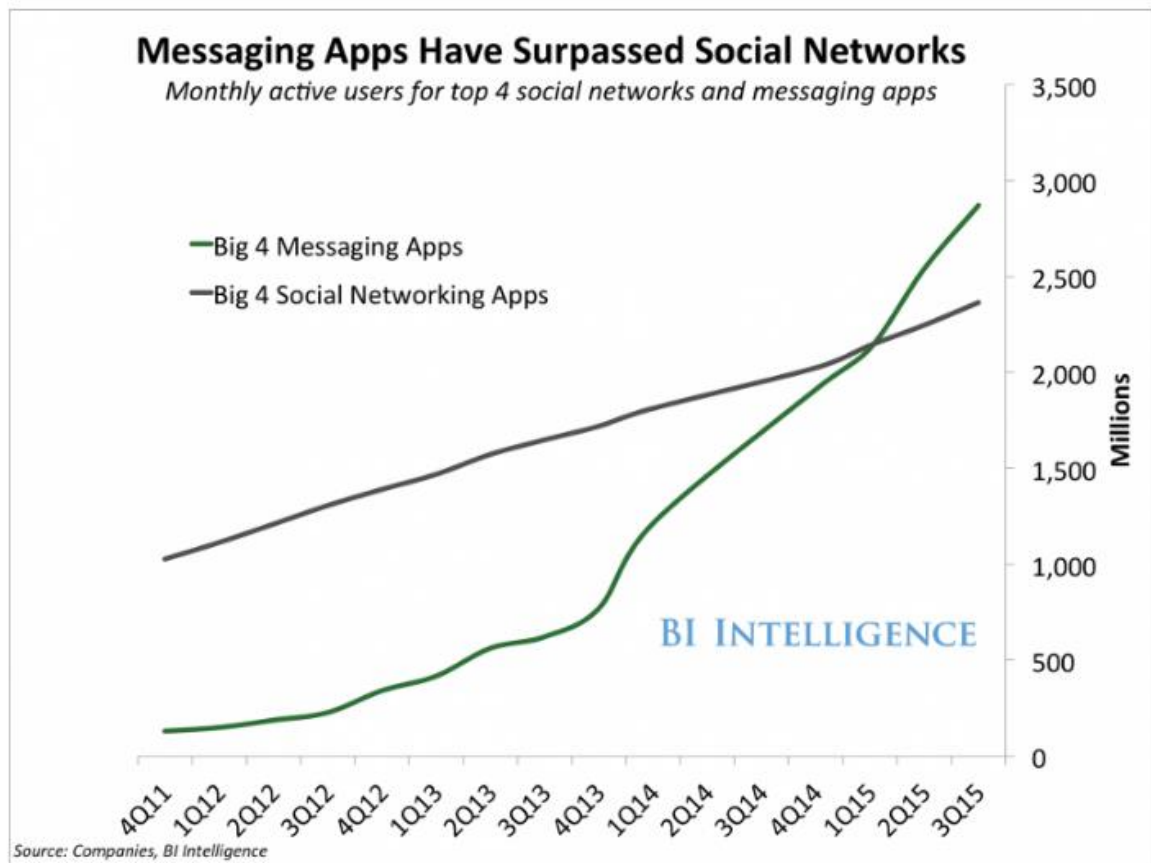
**Buying shoes isn't the only thing chatbot can be used for.** Here are a couple of other examples:

- **Weather bot.** Get the weather whenever you ask.
- **Grocery bot.** Help me pick out and order groceries for the week.
- **News bot.** Ask it to tell you whenever something interesting happens.
- **Life advice bot.** I'll tell it my problems and it helps me think of solutions.
- **Personal finance bot.** It helps me manage my money better.
- **Scheduling bot.** Get me a meeting with someone on the Messenger team at Facebook.
- **A bot that's your friend.** In China there is a bot called Xiaoice, built by Microsoft, that over 20 million people talk to.

## 1.3 Why Chatbot are such a big opportunity?

You are probably wondering **"Why does anyone care about chatbots? They look like simple text based services... what's the big deal?"** [1]

It's because for the first time ever **people are using messenger apps more than they are using social networks.**



“Major shifts on large platforms should be seen as an opportunity for distribution. That said, we need to be careful not to judge the very early prototypes too harshly as the platforms are far from complete. I believe Facebook’s recent launch is the beginning of a new application platform for micro application experiences. The fundamental idea is that customers will interact with just enough UI, whether conversational and/or widgets, to be delighted by a service/brand with immediate access to a rich profile and without the complexities of installing a native app, all fueled by mature advertising products. It’s potentially a massive opportunity.”— Aaron Batalion, Partner at Lightspeed Venture Partners

This is why chatbots are such a big deal. **It’s potentially a huge business opportunity for anyone willing to jump headfirst and build something people want.**

## 1.4 How Chatbots works?

There are two types of chatbots, one functions based on a set of rules, and the other more advanced version uses machine learning.

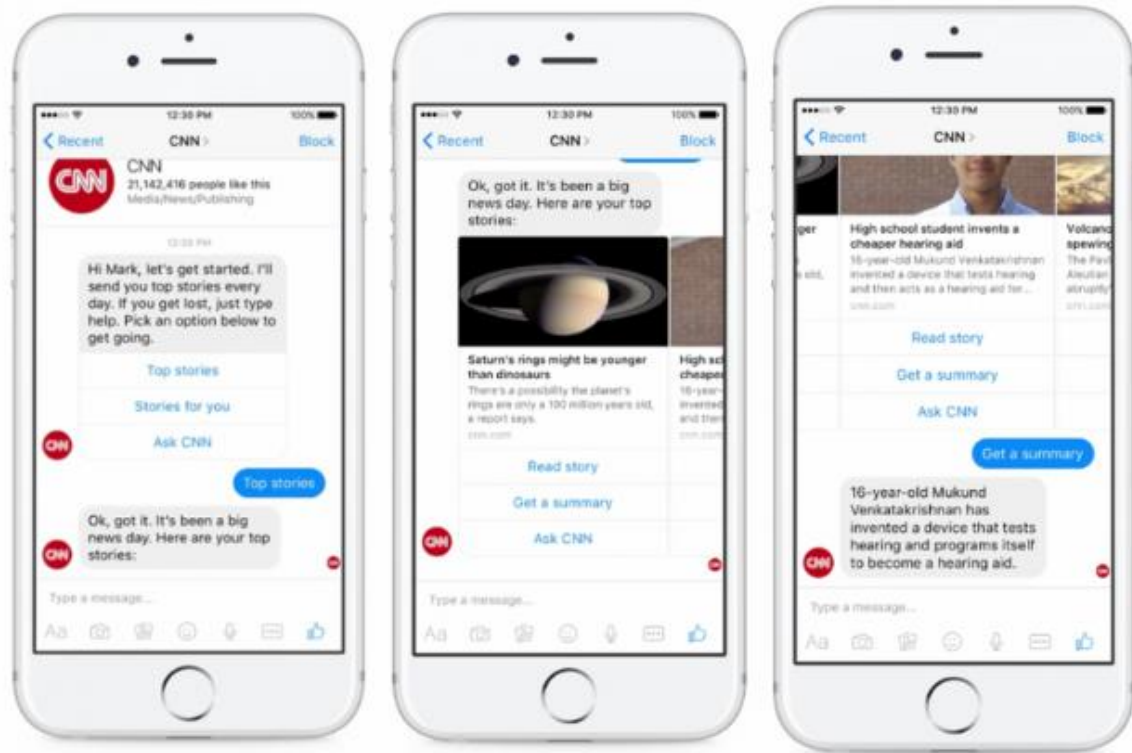
### Chatbot that functions based on rules:

- This bot is very limited. It can only respond to very specific commands. If you say the wrong thing, it doesn’t know what you mean.
- This bot is only as smart as it is programmed to be.

## Chatbot that functions using machine learning:

- This bot has an artificial brain AKA artificial intelligence. You don't have to be ridiculously specific when you are talking to it. It understands language, not just commands.
- This bot continuously gets smarter as it learns from conversations it has with people.

**Bots are created with a purpose.** A store will likely want to create a bot that helps you purchase something, where someone like Comcast might create a bot that can answer customer support questions. You start interacting with it by sending a message. Following is an example of CNN's chatbot for pulling out latest news information.



## 1.5 Available Frameworks to build Bots

Major tech companies have started to build frameworks to make it easier to build Bots. Following are few of them.

### Facebook Bot Engine

Facebook Bot Engine, released in April 2016, is based on the technology of Wit.ai, acquired by Facebook in early 2015. While Wit.ai itself runs from its own server in the cloud, the Bot Engine is a wrapper built to deploy the bots in Facebook Messenger.

Wit.ai offers several options:

1. Extract certain predefined entities (the standard set of date, time, phone, etc.) [listed here](#).
2. Extract intent ("what the user wants to perform").
3. Extract sentiment (per utterance only).
4. Define and extract own entities.

## Microsoft Bot Framework

Microsoft announced its commitment to the conversational commerce approximately at the same time as Facebook. However, Microsoft's needs, philosophy, and approach are somewhat different. Just like the Facebook's offering, Microsoft's SDK can be viewed as two components, which are independent of each other:

- Bot Connector, the integration framework
- LUIS.ai, the natural language understanding component

The integration component is impressive. It plugs into GroupMe, Skype, Slack, SMS, Telegram, web chat, Facebook Messenger, and email. There is a PaaS option on Azure, just for bots.

LUIS.ai is tightly integrated with the language via the use of .NET attributes. In more than one way, the philosophy is similar to that of Wit.ai and API.ai: intents and entities. Being a Microsoft product, it comes with nice deployment capabilities, and to cover the shortcomings of the user-defined models, the developers are given access to Cortana models. As long as Cortana keeps being developed, this library keeps growing. It is more API.ai than Wit.ai. Documentation can be found [here](#).

## API.ai

API.ai is another primarily web-based bot framework. While the general philosophy with the reliance on entities and intents appears similar, the integration options are far more expansive. API.ai bots can be exported as modules, integrated in Facebook, Kik, Slack, Alexa, and even Cortana.

What's more important, API.ai seems to have understood the weakness of letting the users define entities and intents by entering numerous utterances, and provided a large set of domains. An option to define user entities solves the issue with user's titles which Wit.ai does not solve.

Documentation for API.ai can be found [here](#).

## Viv

[Viv.ai](#), a company co-founded by the authors of Siri, demoed its technology in May 2016. The company is yet to release more details, but it appears that the focus is on processing complex queries, using a flexible mechanism juggling prepared recipe components which can be assembled into endless combinations.

The advance over Siri 1.0 type of assistants is not a linguistic one, but rather the new type of application logic. It appears to be similar to the way SQL queries are processed, broken down into constituents, and combined into an execution plan. The founders made it clear that Viv is intended to be a development framework. They did, however, mention multiple times that it's a "virtual assistant", and indeed this type of complex queries are more suitable for this type of applications. It might be too risky and unpredictable to be used in enterprise applications which must adhere to standards and procedures though, unless tight control is exercised (in which case, the advantage of the flexibility and recipes floating in the cloud is somewhat lost).

## IBM Watson

IBM Watson is a technology platform that uses natural language processing and machine learning to reveal insights from large amounts of unstructured data. The Watson Dialog service allows you to upload a conversation script as a "template". You can then make web services calls to converse with the service. The Dialog service will respond based on the script. It will automatically keep track of the conversation

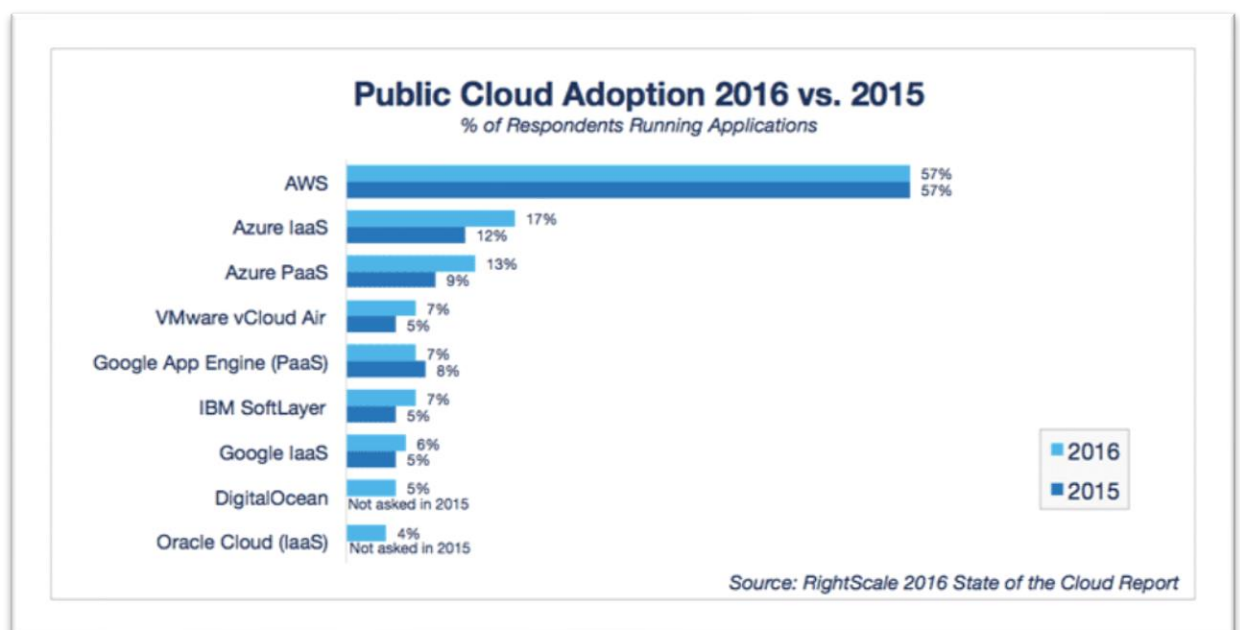


context (e.g., when you say “I want it”—what does “it” refers to?) as well as providing basic natural language processing support within the conversation. For details on how to create a Watson Dialog template, you can refer to its [documentation](#).

The brand name Watson now encompass everything IBM does in the field of artificial intelligence (AI). Watson now has many software services available via IBM’s cloud computing platform Bluemix.

## 1.6 Available Cloud Platforms

The many benefits offered by the cloud – on-demand scalability, reduced costs – have made it an irresistible choice for businesses. According to **RightScale’s Fifth Annual State of the Cloud Survey** of the latest cloud computing trends, AWS continues to lead in public cloud adoption.



But as you can see in the graph above, Microsoft’s Azure is galloping closely behind. A recent report by Morgan Stanley, based on a survey of 100 CIOs, **predicted that Azure would be the largest Infrastructure as a Service vendor by 2019.**

## Part 2: Pizza Ordering Bot

### 2.1 Introduction

It's a simple bot that lets users order a pizza of their own choice using a skype chat service. User can customize the order by choosing size, type, toppings of its own choice. The bot application uses Microsoft Bot Platform and LUIS intent service for understanding the language. Moreover, the application is hosted in Microsoft Azure cloud platform and can be accessed any time and used on any platform (Android, iOS, Windows, Linux platform) within the skype application. Web link to find the application is as following:

<http://pizzabotj.azurewebsites.net/>


It looks as following

## Pizza Ordering Bot

[Click Here](#)

### Skype Bot

It's a simple Bot that resides in your skype contacts and you can access it any time to order a pizza for you. You can build your custom pizza with the size and topping of your own choice. Click on the button to add it your skype contacts!



#### What?

An Internet bot, also known as web robot, which performs tasks that are both simple and structurally repetitive, at a much higher rate than would be possible for a human alone.

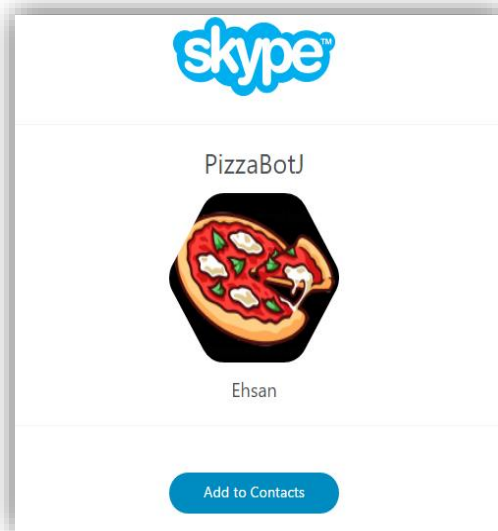
#### Where?

Click on the Red button to find the bot application

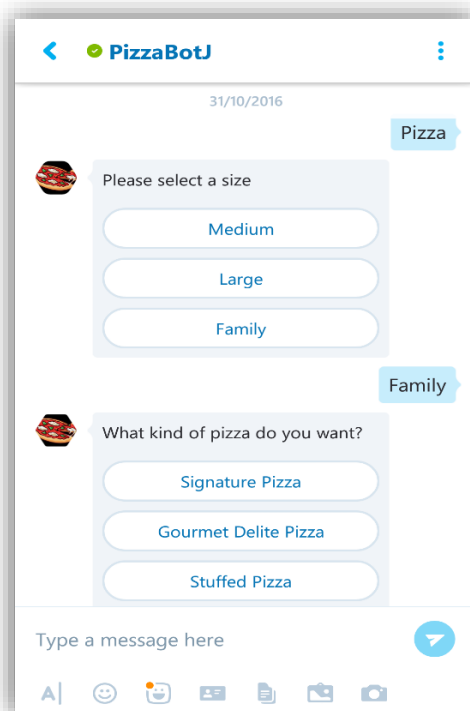
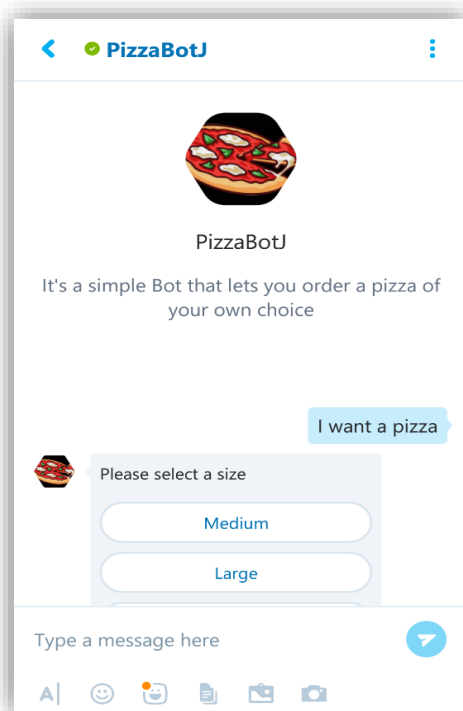
#### How?

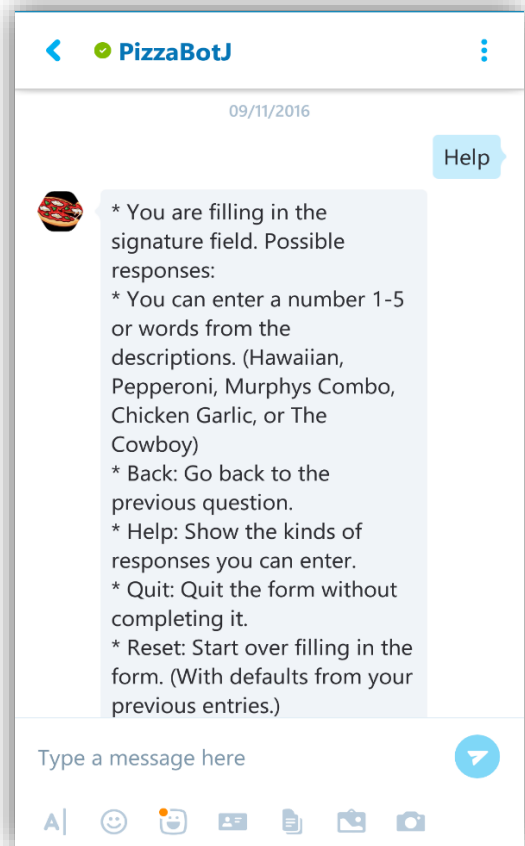
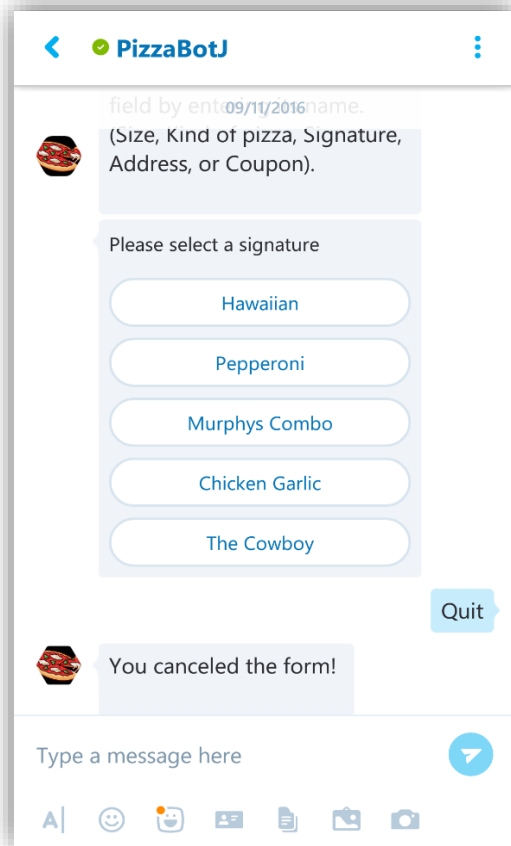
Add it to your skype contacts. It will always be there to help your order the pizza of your choice

After you click the button it will ask you add the bot application to yours skype contacts. It might ask you to login to yours skype if you are not logged in already.



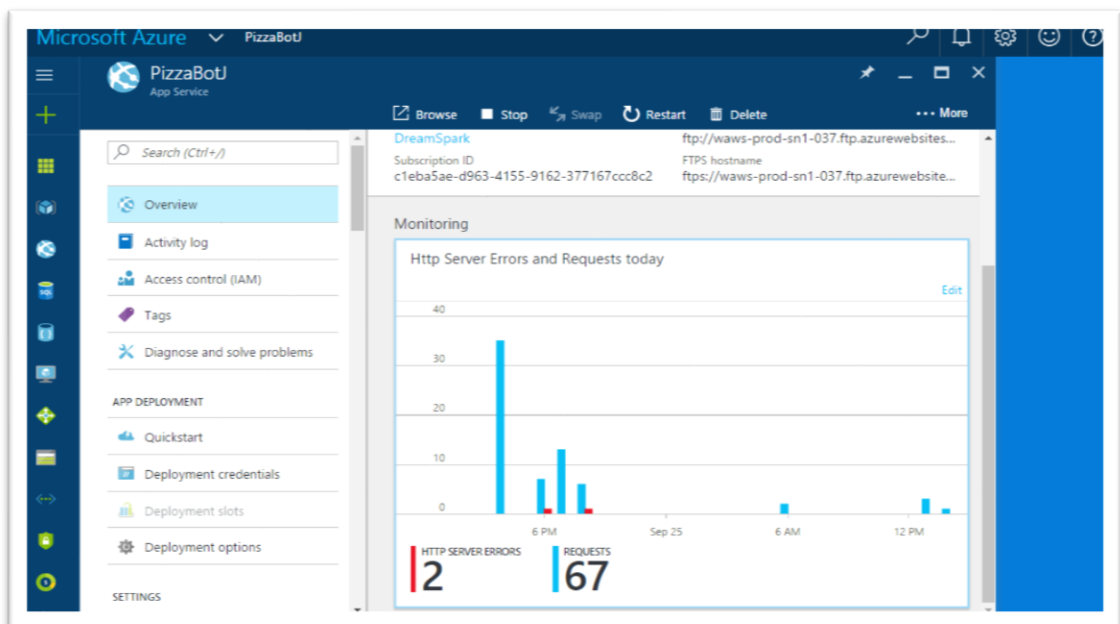
After you logged-in and added the bot in your contacts. Just start conversation with a “I want a Pizza” “Give me a pizza” or just “Pizza”. It will guide you throughout the order. You can type “Back” to go back to previous option, “Help” to find possible responses, “Quit” to quit the form without filling it. “Reset” to start over filling the form again, “status” to show your progress in filling the form etc. etc.





## Cloud portal view

Following is the cloud portal view of running application and it allows us to view the traffic and number of requests in real time. [3]



## 2.2 Build Process

### 2.2.1 Getting Started

#### Overview

Microsoft Bot Builder is a powerful framework for constructing bots that can handle both freeform interactions and more guided ones where the possibilities are explicitly shown to the user. It is easy to use and leverages C# to provide a natural way to write bots.[2]

#### High Level Features:

- Powerful dialog system with dialogs that are isolated and composable.
- Built-in dialogs for simple things like Yes/No, strings, numbers, enumerations.
- Built-in dialogs that utilize powerful AI frameworks like [LUIS](#).
- Bots are stateless which helps them scale.
- [FormFlow](#) for automatically generating a bot from a C# class for filling in the class and that supports help, navigation, clarification and confirmation.
- SDK source code is found on <http://github.com/Microsoft/botbuilder>.

#### Getting started with the Connector

The Microsoft Bot Connector is a communication service that helps you connect your Bot with many different communication channels (Skype, SMS, email, and others). If you write a conversational Bot or agent and expose a Microsoft Bot Framework-compatible API on the Internet, the Bot Framework Connector service will forward messages from your Bot to a user, and will send user messages back to your Bot.

To use the Microsoft Bot Framework Connector, you must have:

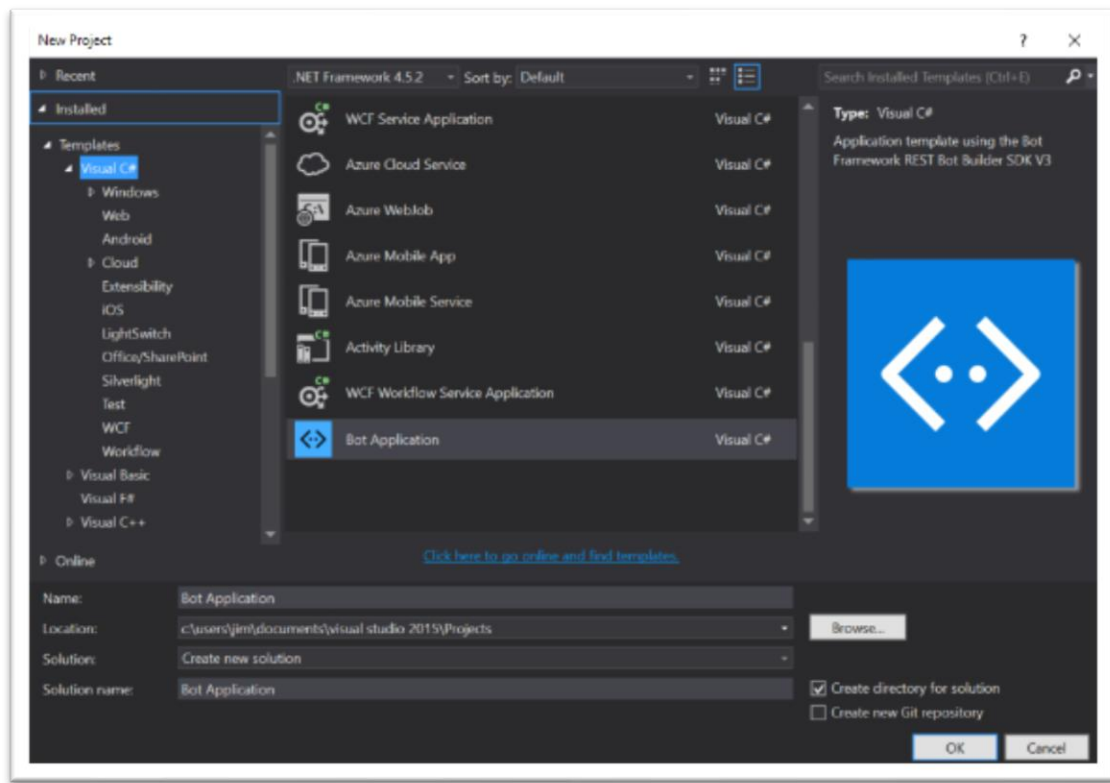
- 1.A Microsoft Account (Hotmail, Live, Outlook.com) to log into the Bot Framework developer portal, which you will use to register your Bot.
- 2.An Azure-accessible REST endpoint exposing a callback for the Connector service.
- 3.Developer accounts on one or more communication services (such as Skype) where your Bot will communicate.

#### Getting started in .NET

This is a step-by-step guide to writing a Bot in C# using the Bot Framework Connector SDK .NET template.[2]

1. Install prerequisite software
  - Visual Studio 2015 (latest update)
  - Important: Please update all VS extensions to their latest versions Tools->Extensions and Updates->Updates
2. Download and install the Bot Application template
  - Download the file from the direct download link [here](#):
  - Save the zip file to your Visual Studio 2015 templates directory which is traditionally in "%USERPROFILE%\Documents\Visual Studio 2015\Templates\ProjectTemplates\Visual C#\"
3. Open Visual Studio

4. Create a new C# project using the new Bot Application template.

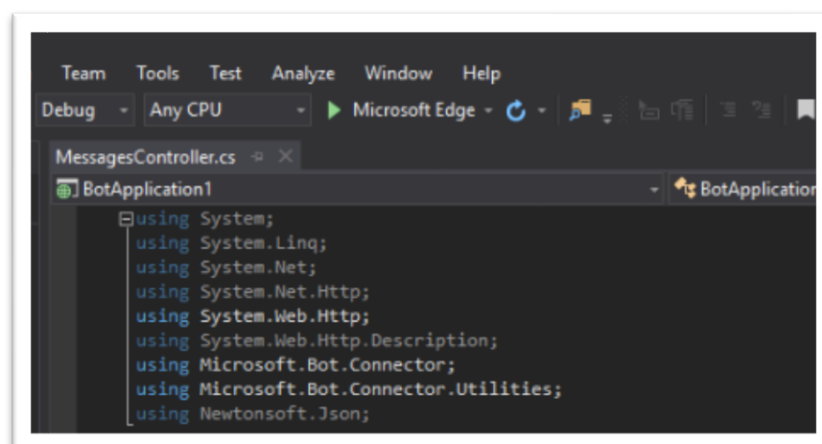


## Testing with Emulator

Use the Bot Framework Emulator to test your Bot application

The Bot Framework provides a channel emulator that lets you test calls to your Bot as if it were being called by the Bot Framework cloud service. To install the Bot Framework Emulator, download it from [here](#).

One installed, you're ready to test. First, start your Bot in Visual Studio using a browser as the application host. The image below uses Microsoft Edge.

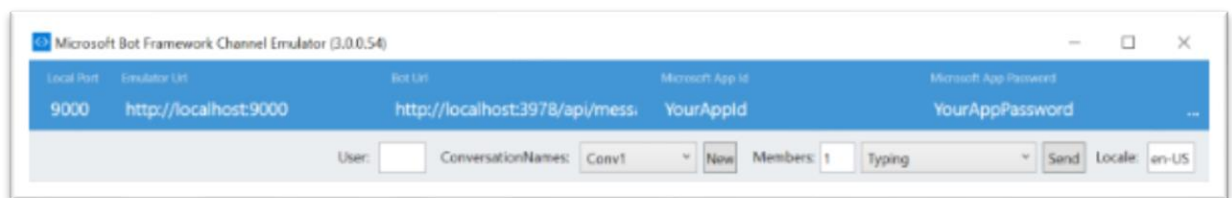


When using the emulator to test your Bot application, make note of the port that the application is running on, which in this example is port 3978. You will need this information to run the Bot Framework Emulator.

Now open the Bot Framework Emulator. There are a few items that you will need to configure in the tool before you can interact with your Bot Application.

When working with the emulator with a bot **running locally**, you need:

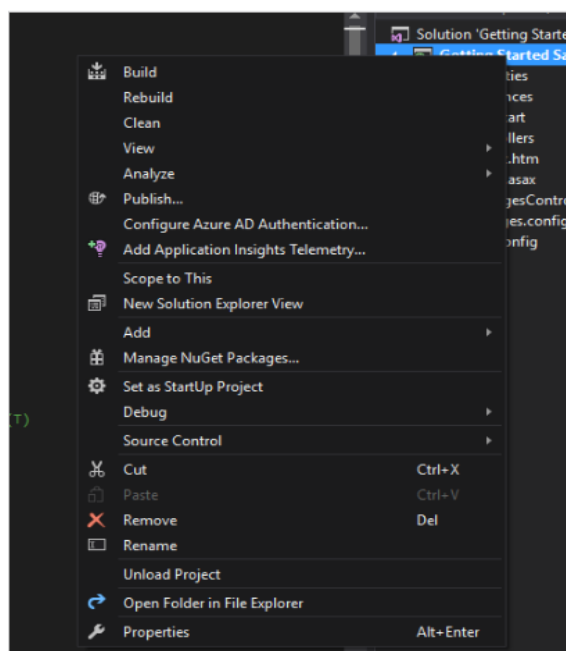
- The **Url** for your bot set the localhost:<port> pulled from the last step. > Note: will need to add the path "/api/messages" to your URL when using the Bot Application template.
- Empty out the **MicrosoftAppId** field
- Empty out the **MicrosoftAppPassword** field



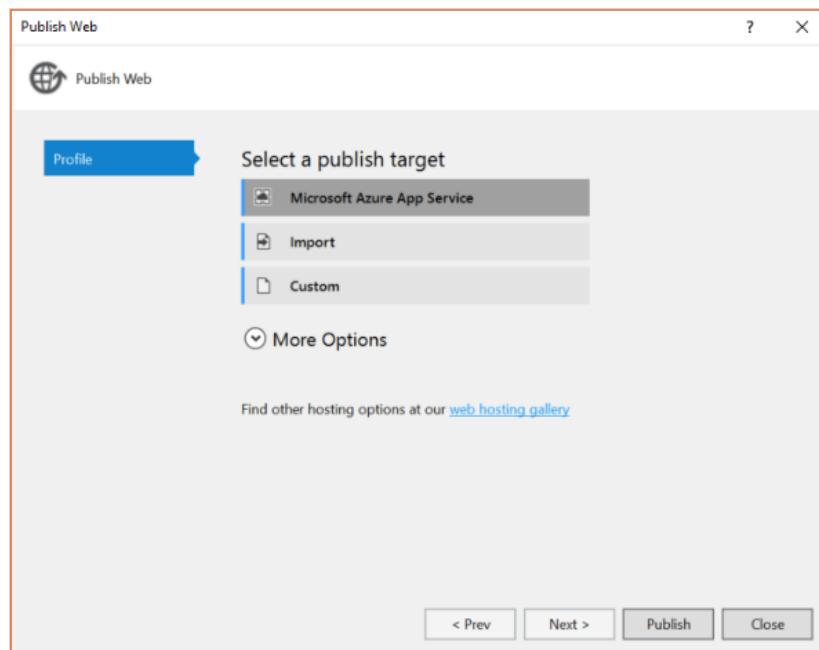
## 2.2.2 Publishing your Bot Application to Microsoft Azure

In this tutorial, we use Microsoft Azure to host the Bot application. To publish your Bot Application, you will need a Microsoft Azure subscription. You can get a 30-day free trial from here: [azure.microsoft.com/en-us/](https://azure.microsoft.com/en-us/)

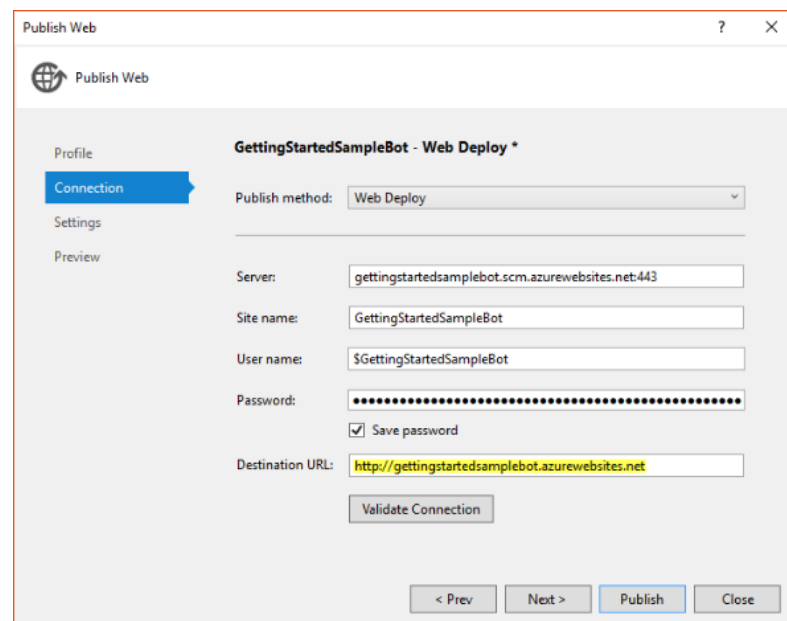
In Visual Studio, right clicking on the project in Solution Explorer and select "Publish" - or alternately selecting "Build | Publish" displays the following dialog:



Publish dialog will appear. Please select Microsoft Azure App Services

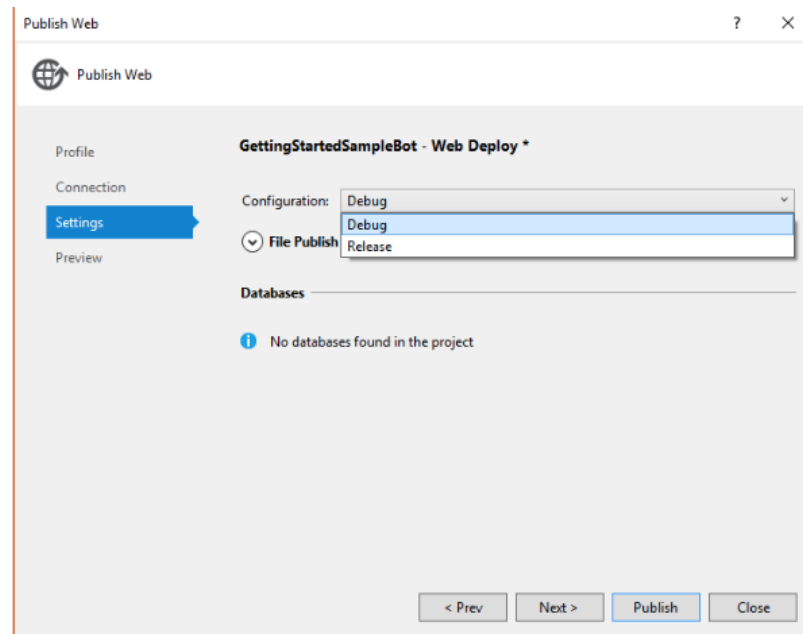


Then select next and you will see following:

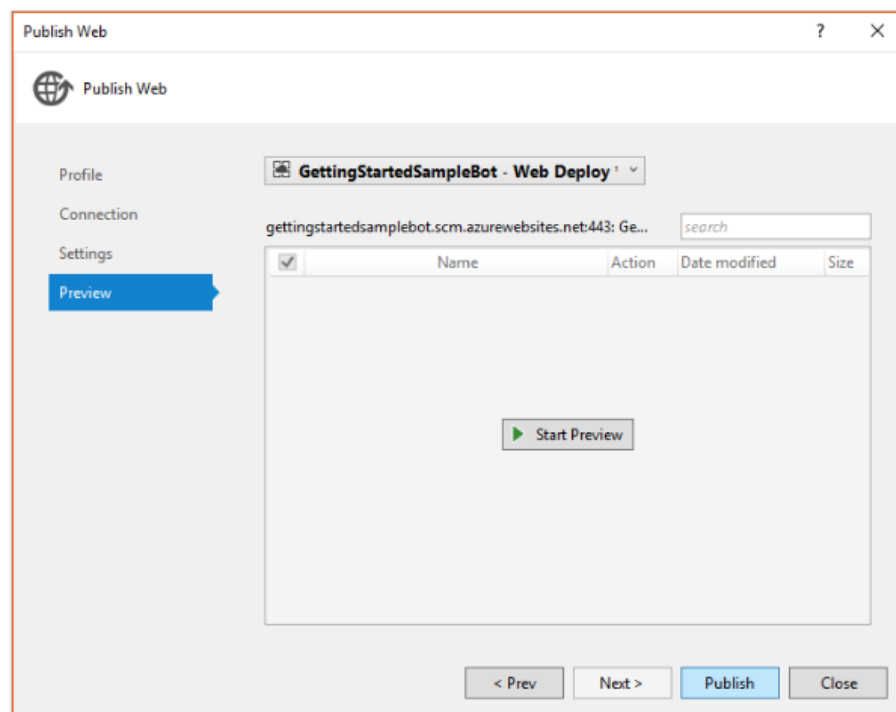


Click on validate connection to check everything is fine. Destination URL is important, this the location from where you can access the bot. Copy it to the clipboard. Click next and select debug or release.

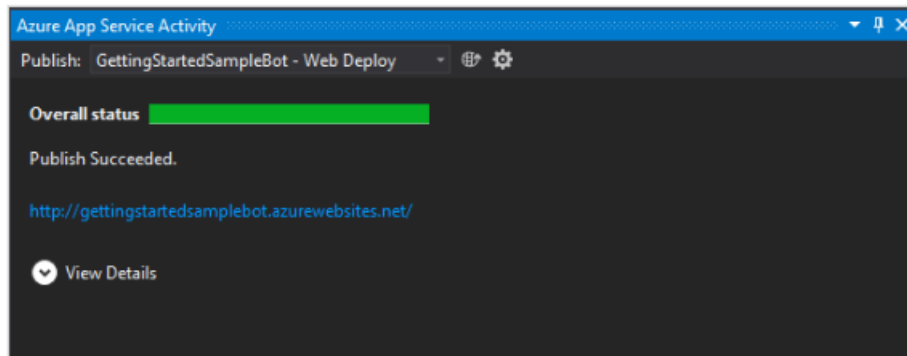




By default, your Bot will be published in a Release configuration. If you want to debug your Bot, change Configuration to Debug. Regardless, from here you'll hit "Publish" and your Bot will be published to Azure.



You will see a number of messages displayed in the Visual Studio 2015 "Output" window. Once publishing is complete you will also see the web page for your Bot Application displayed in your browser (the browser will launch, and render your Bot Application HTML page), see below.



## 2.3 Registering your Bot with the Microsoft Bot Framework

Registering your Bot tells the Connector how to call your Bot's web service. Note that the **MicrosoftAppId** and **MicrosoftAppPassword** are generated when your Bot is registered with the Microsoft Bot Framework Connector, the MicrosoftAppId and MicrosoftAppPassword are used to authenticate the conversation, and allows the developer to configure their Bot with the Channels they'd like to be visible on. The BotId, which you specify, is used for the URL in the directory and developer portal.

1. Go to the Microsoft Bot Framework portal at **<https://dev.botframework.com>** and sign in with your Microsoft Account.
2. Click the "Register a Bot" button and fill out the form. Many of the fields on this form can be changed later. Use the endpoint generated from your Azure deployment, and don't forget that when using the Bot Application template, you'll need to extend the URL you pasted in with the path to the endpoint at / API / Messages. You should also prefix your URL with HTTPS instead of HTTP; Azure will take care of providing HTTPS support on your bot. Save your changes by hitting "Create" at the bottom of the form.

1. Once your registration is created, Microsoft Bot Framework will take you through generating your **MicrosoftAppId** and **MicrosoftAppPassword**. These are used to authenticate your Bot with the Microsoft Bot Framework. **NOTE:** When you generate your MicrosoftAppPassword, be sure to record it somewhere as you won't be able to see it again.

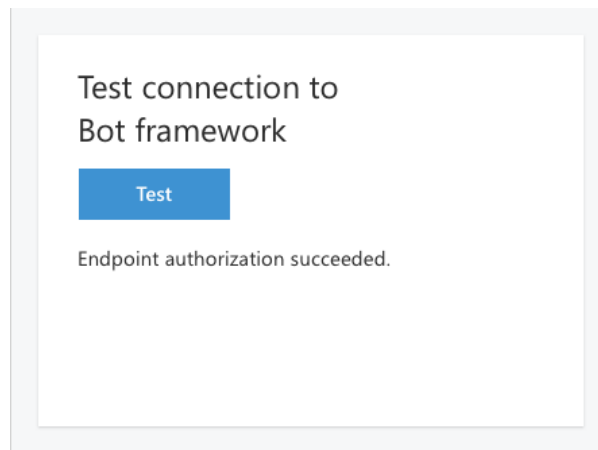
Now that the Bot is registered, you need to update the keys in the web.config file in your Visual Studio project. Change the following keys in the web.config file to match the ones generated when you saved your registration, and you're ready to build. Clicking the "show" link will show the value, along with exposing the regenerate link if you ever need to change your AppPassword. Update your web.config, and re-publish your bot to Azure.

```
<? xml version = "1.0" encoding = "utf-8" ?>
<!--
  For more information on how to configure your ASP.NET application, please visit
  http://go.microsoft.com/fwlink/?LinkId=301879
-->
< configuration >
  < appSettings >
    <!--update these with your appid and one of your appsecret keys-->
    < add key = "MicrosoftAppId" value = "[GUID]" />
    < add key = "MicrosoftAppPassword" value = "[PASSWORD]" />
  </ appSettings >
```

## Testing the connection to your bot

Back in the developer dashboard for your Bot there's a test chat window that you can use to interact with your Bot without further configuration, and verify that the Bot Framework can communicate with your Bot's web service.

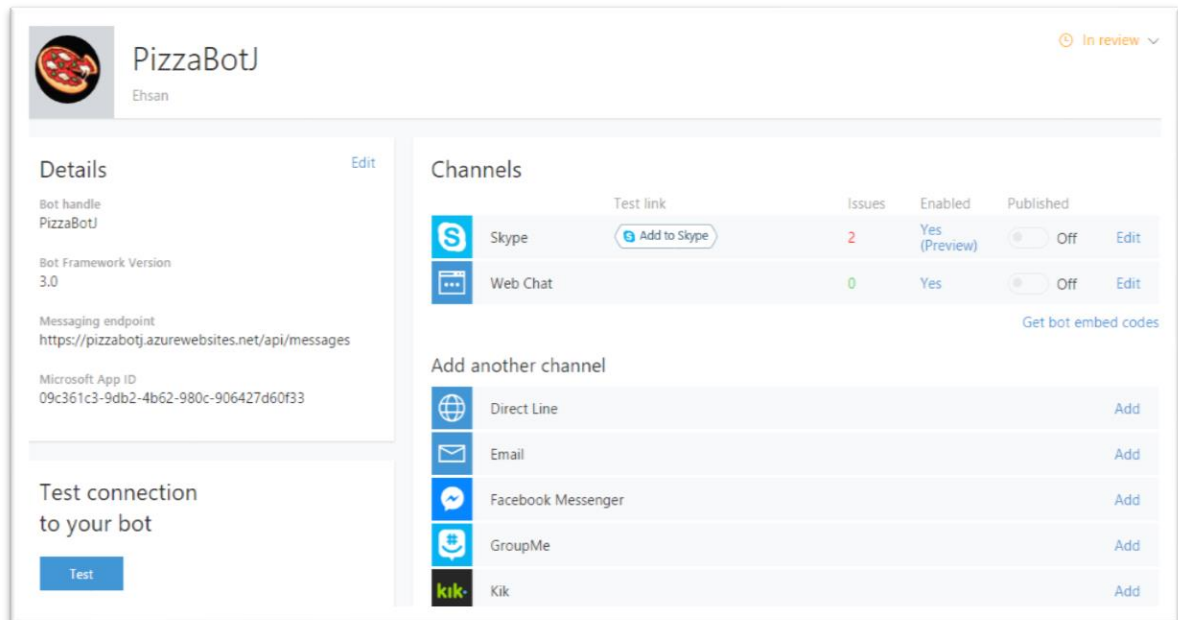
Note that the first request after your Bot starts up can take 10 - 15 s as Azure starts up the web service for the first time. Subsequent requests will be quick. This simple viewer will let you see the JSON object returned by your Bot.



## Configuring Channels

Now that you have a Bot up and running, you'll want to configure it for one or more channels your users are using. Configuring channels is a combination of Microsoft Bot Framework workflow and conversation service workflow, and is unique for each channel you wish to configure.

1. To configure a channel, go back to the Bot Framework portal at <https://www.botframework.com>. Sign in, select your Bot, and go to the channels panel.



## 2.4 Adding language understanding capability using LUIS.ai

Language Understanding Intelligent Service (LUIS) offers a fast and effective way of adding language understanding to applications. With LUIS, you can use pre-existing, world-class, pre-built models from Bing and Cortana whenever they suit your purposes -- and when you need specialized models, LUIS guides you through the process of quickly building them.

LUIS draws on technology for interactive machine learning and language understanding from **Microsoft Research** and Bing, including Microsoft Research's **Platform for Interactive Concept Learning (PICL)**. LUIS is a part of a project of **Microsoft Cognitive Services**. [8][9]

One of the key problems in human-computer interactions is the ability of the computer to understand what a person wants, and to find the pieces of information that are relevant to their intent. Language Understanding Intelligent Service is designed to provide you with an easy way to create models, which allow your applications to understand user commands. Following diagram shows a model for Pizza Ordering bot with multiple intents and entities. For further information on how create intents and entities in LUIS visit <https://www.luis.ai/> .

LUIS

[My Applications](#)
[About](#)
[Help Docs](#)
[Support](#)

PizzaBot

App Settings

Publish

Intents

None
OrderPizza
DeliveryTime
UseCoupon
StoreHours
DeliveryAddress

Entities

BVO.Crust

BVO.Toppings

GourmetDelite

Address

BVO.Sauce

Kind

Available

Signature

Size

Stuffed

Pre-built Entities

No pre-built entities added

Regex Features

No patterns added

New utterances

Search

Suggest

Review labels

Show all labeled utterances

Select text in an utterance to label an entity, or click to clear.

Model prediction

what are the store hours today ?

StoreHours (1)

what are the store hours today ?

StoreHours(1)

Model prediction

send a large pepperoni pizza to my house

OrderPizza (1)

send a large pepperoni pizza to my house

OrderPizza(1)

Model prediction

whats the latest you deliver ?

StoreHours (1)

whats the latest you deliver ?

StoreHours(1)

Model prediction

i ' d like to order a pizza

OrderPizza (0.99)

i ' d like to order a pizza

OrderPizza(0.99)

Model prediction

what are the store hours ?

StoreHours (1)

what are the store hours ?

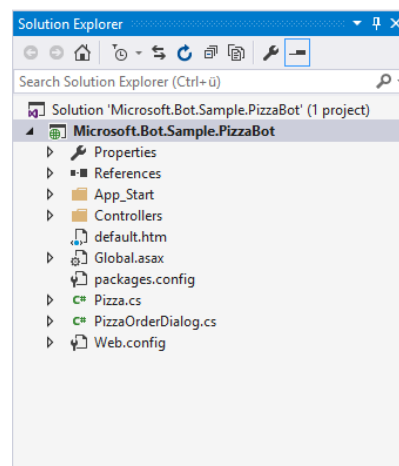
StoreHours(1)

Train

## Part 3: Project Code

### 3.1 Folders Structure

Project contains following list of folders and files. App\_Start folder contains configuration file for web API and defines path for communication with the Bot. Controllers folder contains the MessagesController.cs which the main file for handling all the message communication between user and the Bot. Pizza.cs and PizzaOrderDialog.cs handles basically the form flow and LUIS api dialogs. Web.config is one of the most important file which stores Bot-ID and Password for secure communication over some channel. Solution explorer view looks as following



### 3.2 MessageControler.cs

This is the main starting point of the application. It contains the following code:

```
using System.Web.Http;
using System.Threading.Tasks;

using Microsoft.Bot.Connector;
using Microsoft.Bot.Builder.FormFlow;
using Microsoft.Bot.Builder.Dialogs;
using System.Web.Http.Description;
using System.Net.Http;
using System.Diagnostics;

namespace Microsoft.Bot.Sample.PizzaBot
{
    [BotAuthentication]
    public class MessagesController : ApiController
    {
        {
            private static IForm<PizzaOrder> BuildForm()
            {
                var builder = new FormBuilder<PizzaOrder>();

                ActiveDelegate<PizzaOrder> isBYO = (pizza) => pizza.Kind ==
                PizzaOptions.BYOPIzza;
            }
        }
    }
}
```

```

        ActiveDelegate<PizzaOrder> isSignature = (pizza) => pizza.Kind ==
        PizzaOptions.SignaturePizza;
        ActiveDelegate<PizzaOrder> isGourmet = (pizza) => pizza.Kind ==
        PizzaOptions.GourmetDelitePizza;
        ActiveDelegate<PizzaOrder> isStuffed = (pizza) => pizza.Kind ==
        PizzaOptions.StuffedPizza;

        return builder
            .Field(nameof(PizzaOrder.Choice))
            .Field(nameof(PizzaOrder.Size))
            .Field(nameof(PizzaOrder.Kind))
            .Field("BYO.Crust", isBYO)
            .Field("BYO.Sauce", isBYO)
            .Field("BYO.Toppings", isBYO)
            .Field(nameof(PizzaOrder.GourmetDelite), isGourmet)
            .Field(nameof(PizzaOrder.Signature), isSignature)
            .Field(nameof(PizzaOrder.Stuffed), isStuffed)
            .AddRemainingFields()
            .Confirm("Would you like a {Size}, {BYO.Crust} crust, {BYO.Sauce},
            {BYO.Toppings} pizza?", isBYO)
            .Confirm("Would you like a {Size}, {&Signature} {Signature} pizza?",
            isSignature, dependencies: new string[] { "Size", "Kind", "Signature" })
            .Confirm("Would you like a {Size}, {&GourmetDelite} {GourmetDelite}
            pizza?", isGourmet)
            .Confirm("Would you like a {Size}, {&Stuffed} {Stuffed} pizza?",
            isStuffed)
            .Build()
            ;
    }

    internal static IDialog<PizzaOrder> MakeRoot()
    {
        return Chain.From(() => new PizzaOrderDialog(BuildForm));
    }

    /// <summary>
    /// POST: api/Messages
    /// receive a message from a user and send replies
    /// </summary>
    /// <param name="activity"></param>
    [ResponseType(typeof(void))]
    public virtual async Task<HttpResponseMessage> Post([FromBody] Activity
activity)
    {
        if (activity != null)
        {
            // one of these will have an interface and process it
            switch (activity.GetActivityType())
            {
                case ActivityTypes.Message:
                    await Conversation.SendAsync(activity, MakeRoot);
                    break;

                case ActivityTypes.ConversationUpdate:
                case ActivityTypes.ContactRelationUpdate:
                case ActivityTypes.Typing:
                case ActivityTypes.DeleteUserData:
                default:
                    Trace.TraceError($"Unknown activity type ignored:
                    {activity.GetActivityType()}");
                    break;
            }
        }
    }

```



```

    }
    return new HttpResponseMessage(System.Net.HttpStatusCode.Accepted);
}
}
}

```

### 3.3 Pizza.cs

It contains all the enumeration for Pizza options and selection of choices. It contains the following code:

```

using Microsoft.Bot.Builder.FormFlow;
using System;
using System.Collections.Generic;
using System.Text;
#pragma warning disable 649

namespace Microsoft.Bot.Sample.PizzaBot
{
    public enum SizeOptions
    {
        // 0 value in enums is reserved for unknown values. Either you can supply an
        // explicit one or start enumeration at 1.
        Unknown,
        [Terms(new string[] { "med", "medium" })]
        Medium,
        Large,

        [Terms(new string[] { "family", "extra large" })]
        Family
    };

    public enum PizzaOptions
    {
        Unknown, SignaturePizza, GourmetDelitePizza, StuffedPizza,

        [Terms(new string[] { "byo", "build your own" })]
        [Describe("Build your own")]
        BYOPizza
    };

    public enum SignatureOptions { Hawaiian = 1, Pepperoni, MurphysCombo,
        ChickenGarlic, TheCowboy };
    public enum GourmetDeliteOptions { SpicyFennelSausage = 1,
        AngusSteakAndRoastedGarlic, GourmetVegetarian, ChickenBaconArtichoke,
        HerbChickenMediterranean };
    public enum StuffedOptions { ChickenBaconStuffed = 1, ChicagoStyleStuffed,
        FiveMeatStuffed };

    // Fresh Pan is large pizza only
    public enum CrustOptions
    {
        Original = 1, Thin, Stuffed, FreshPan, GlutenFree
    };

    public enum SauceOptions
    {
        [Terms(new string[] { "traditional", "tomatoe?" })]
        Traditional = 1,

```

```

        CreamyGarlic, OliveOil
    };

    public enum ToppingOptions
    {
        Beef = 1,
        BlackOlives,
        CanadianBacon,
        CrispyBacon,
        Garlic,
        GreenPeppers,
        GrilledChicken,

        [Terms(new string[] { "herb & cheese", "herb and cheese", "herb and cheese
blend", "herb" })]
        HerbAndCheeseBlend,
        ItalianSausage,
        ArtichokeHearts,
        MixedOnions,
        MozzarellaCheese,
        Mushroom,
        Onions,
        ParmesanCheese,
        Pepperoni,
        Pineapple,
        RomaTomatoes,
        Salami,
        Spinach,
        SunDriedTomatoes,
        Zucchini,
        ExtraCheese
    };

    public enum CouponOptions { Large20Percent = 1, Pepperoni20Percent };

    [Serializable]
    class BYOPizza
    {
        public CrustOptions Crust;
        public SauceOptions Sauce;
        public List<ToppingOptions> Toppings = new List<ToppingOptions>();
    };

    [Serializable]
    class PizzaOrder
    {
        public SizeOptions Size;
        [Prompt("What kind of pizza do you want? {||}")]
        [Template(TemplateUsage.NotUnderstood, "What does \"{0}\" mean???")]
        [Describe("Kind of pizza")]
        public PizzaOptions Kind;
        public SignatureOptions Signature;
        public GourmetDeliteOptions GourmetDelite;
        public StuffedOptions Stuffed;
        public BYOPizza BYO;
        public string Address;
        [Optional]
        public CouponOptions Coupon;

        public override string ToString()
        {
            var builder = new StringBuilder();

```

```

        builder.AppendFormat("PizzaOrder({0}, ", Size);
        switch (Kind)
        {
            case PizzaOptions.BYOPIzza:
                builder.AppendFormat("{0}, {1}, {2}, [", Kind, BYO.Crust,
BYO.Sauce);
                foreach (var topping in BYO.Toppings)
                {
                    builder.AppendFormat("{0} ", topping);
                }
                builder.AppendFormat("]");
                break;
            case PizzaOptions.GourmetDelitePizza:
                builder.AppendFormat("{0}, {1}", Kind, GourmetDelite);
                break;
            case PizzaOptions.SignaturePizza:
                builder.AppendFormat("{0}, {1}", Kind, Signature);
                break;
            case PizzaOptions.StuffedPizza:
                builder.AppendFormat("{0}, {1}", Kind, Stuffed);
                break;
        }
        builder.AppendFormat(", {0}, {1}", Address, Coupon);
        return builder.ToString();
    }
};
}

```

### 3.3 PizzaOrderDialog.cs

It handles and creates the user dialog with the help of LUIS.ai (Microsoft's API for Natural language understanding). It contains the following code:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Builder.FormFlow;
using Microsoft.Bot.Builder.Luis;
using Newtonsoft.Json;
using Microsoft.Bot.Builder.Luis.Models;

namespace Microsoft.Bot.Sample.PizzaBot
{
    [LuisModel("4311ccf1-5ed1-44fe-9f10-a6adbad05c14",
"6d0966209c6e4f6b835ce34492f3e6d9")]
    [Serializable]
    class PizzaOrderDialog : LuisDialog<PizzaOrder>
    {
        private readonly BuildFormDelegate<PizzaOrder> MakePizzaForm;

        internal PizzaOrderDialog(BuildFormDelegate<PizzaOrder> makePizzaForm)
        {
            this.MakePizzaForm = makePizzaForm;
        }
    }
}

```

```

[LuisIntent("")]
public async Task None(IDialogContext context, LuisResult result)
{
    await context.PostAsync("I'm sorry. I didn't understand you.");
    context.Wait(MessageReceived);
}

[LuisIntent("OrderPizza")]
[LuisIntent("UseCoupon")]
public async Task ProcessPizzaForm(IDialogContext context, LuisResult result)
{
    var entities = new List<EntityRecommendation>(result.Entities);
    if (!entities.Any((entity) => entity.Type == "Kind"))
    {
        // Infer kind
        foreach (var entity in result.Entities)
        {
            string kind = null;
            switch (entity.Type)
            {
                case "Signature": kind = "Signature"; break;
                case "GourmetDelite": kind = "Gourmet delite"; break;
                case "Stuffed": kind = "stuffed"; break;
                default:
                    if (entity.Type.StartsWith("BYO")) kind = "byo";
                    break;
            }
            if (kind != null)
            {
                entities.Add(new EntityRecommendation(type: "Kind") { Entity =
kind });
                break;
            }
        }
    }

    var pizzaForm = new FormDialog<PizzaOrder>(new PizzaOrder(),
this.MakePizzaForm, FormOptions.PromptInStart, entities);
    context.Call<PizzaOrder>(pizzaForm, PizzaFormComplete);
}

private async Task PizzaFormComplete(IDialogContext context,
IAwaitable<PizzaOrder> result)
{
    PizzaOrder order = null;
    try
    {
        order = await result;
    }
    catch (OperationCanceledException)
    {
        await context.PostAsync("You canceled the form!");
        return;
    }

    if (order != null)
    {
        await context.PostAsync("Your Pizza Order: " + order.ToString());
    }
    else
    {
        await context.PostAsync("Form returned empty response!");
    }
}

```

```

    }

    context.Wait(MessageReceived);
}

enum Days { Saturday, Sunday, Monday, Tuesday, Wednesday, Thursday, Friday };

[LuisIntent("StoreHours")]
public async Task ProcessStoreHours(IDialogContext context, LuisResult result)
{
    var days = (IEnumerable<Days>)Enum.GetValues(typeof(Days));

    PromptDialog.Choice(context, StoreHoursResult, days, "Which day of the
week?");
}

private async Task StoreHoursResult(IDialogContext context, IAwaitable<Days>
day)
{
    var hours = string.Empty;
    switch (await day)
    {
        case Days.Saturday:
        case Days.Sunday:
            hours = "5pm to 11pm";
            break;
        default:
            hours = "11am to 10pm";
            break;
    }

    var text = $"Store hours are {hours}";
    await context.PostAsync(text);

    context.Wait(MessageReceived);
}
}
}

```

## Part 3 Famous Bot Examples

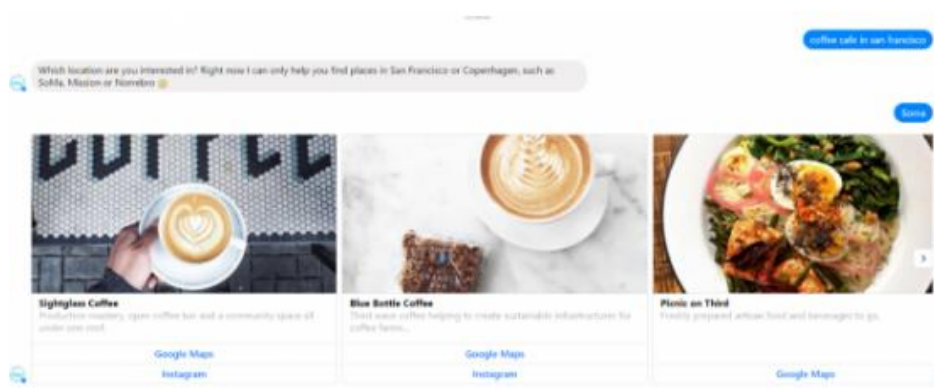
### 3.1 Wall street Journal Bot

The Wall Street Journal Bot provides users with the latest breaking news and stock charts and market data. It also enables users to follow stories or companies and will ping the user once updated.



### 3.2 Besure Messenger

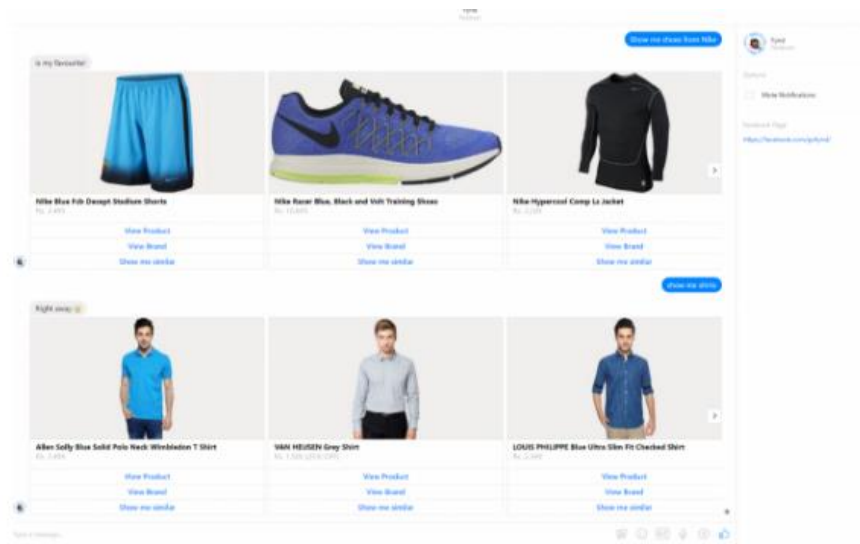
You can ask Besure to find you any type of restaurant imaginable and it will find it for you based on your location. But the only locations it serves, are Copenhagen and San Francisco and nothing in-between (so far). Two great cities for cuisine, but the populations are small and so the coverage is light. You can be sure they expand in the near future.



### 3.3 Fydy's Fify

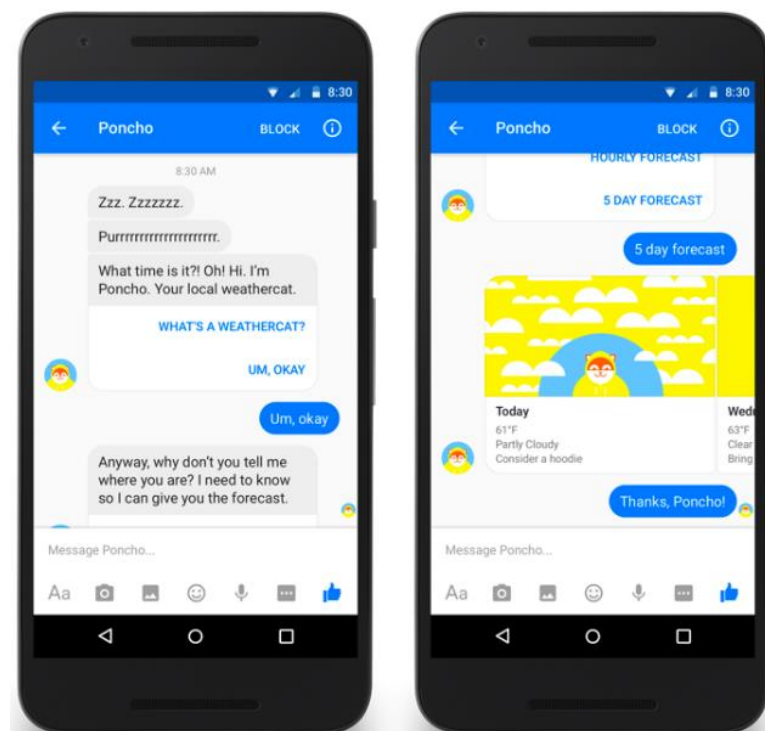
Fify claims to be an intelligent fashion discovery and transaction bot. "Fify will have memory and personality to behave just as humans. She will behave differently with different people. She will remember one's taste and preferences. She will be context aware of what is happening in your world.

Fifty will first talk only about Fynd Fashion and eventually do all sorts of thing related to fashion—discuss trends, alert you about new arrivals, and even gossip about the latest fad of a movie star,” their blog claims.



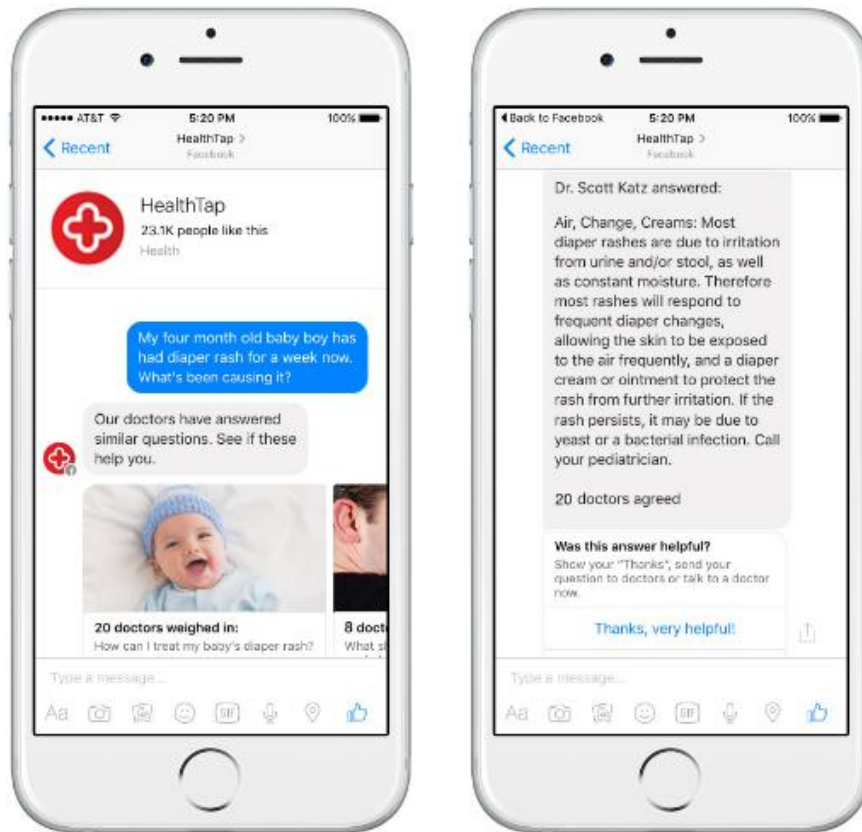
### 3.4 Poncho

Poncho is using bots on Messenger to send people real time local weather reports, including a new on-boarding experience and scheduled weather pushes based on your preference during on-boarding.



### 3.5 HealthTap

HealthTap is now offering the expertise of its network of top U.S. doctors instantly via Messenger. Now people using Messenger around the world have a new, convenient and simple way to access health information. Anyone can type a question into Messenger and receive free and trusted doctor answers wherever they are and whenever they need them.





## References

- [1] <https://chatbotsmagazine.com/>
- [2] <https://docs.botframework.com/en-us/csharp/builder/sdkreference/index.html>
- [3] <https://azure.microsoft.com/en-us/get-started/>
- [4] <https://www.engadget.com/2016/04/13/here-are-all-the-facebook-messenger-bots-we-know-about-so-far/>
- [5] <https://botlist.co/bots/filter?platform=13>
- [6] <https://docs.botframework.com/en-us/>
- [7] <http://aihelpwebsite.com/Blog>
- [8] <https://www.microsoft.com/cognitive-services/en-us/luis-api/documentation/home>
- [9] <https://dev.projectoxford.ai/docs/services/56d95961e597ed0f04b76e58/operations/56f8a55119845511c81de488>
- [10] <https://blog.botframework.com/>