Making a Clever Chatbot for World Domination

Rob Miles
Author and Raconteur
www.robmiles.com

Agenda

An introduction to chatbots

Making a simple chatbot

Making an FAQ chatbot

Chatbots and robots

Adding language understanding

What is a chatbot or “bot”?

A chatbot chats with users

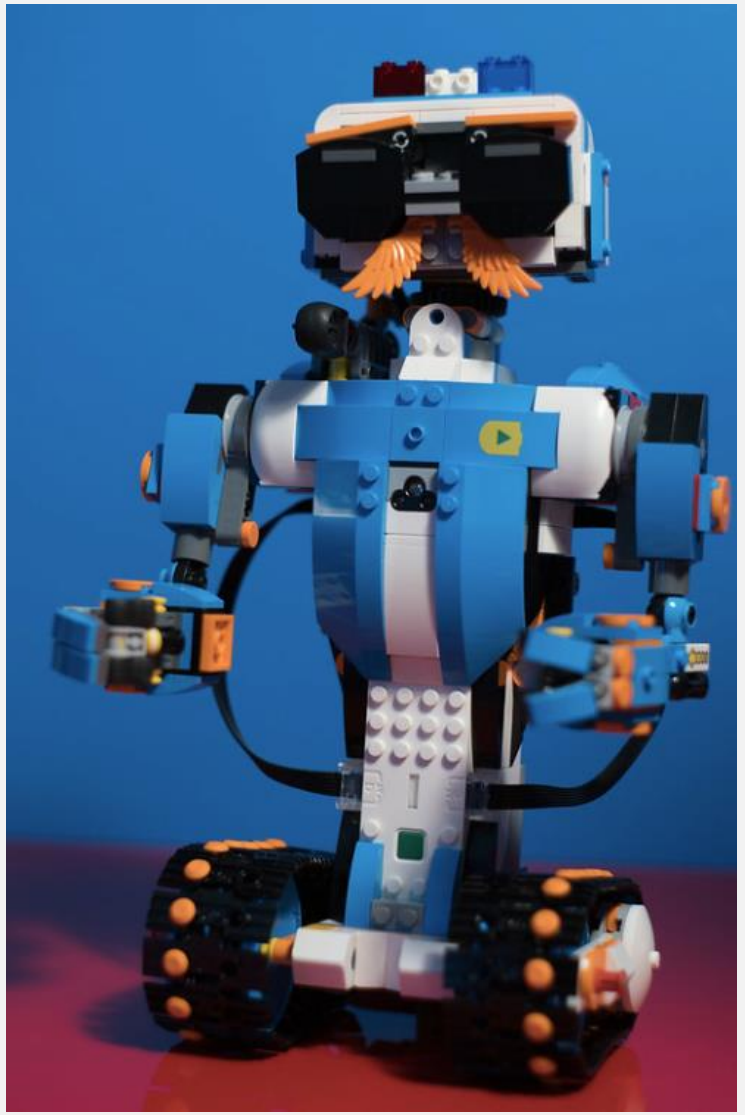
It can use a variety of different channels including voice or a web page

Preferable to installing an application – which users just don’t bother with

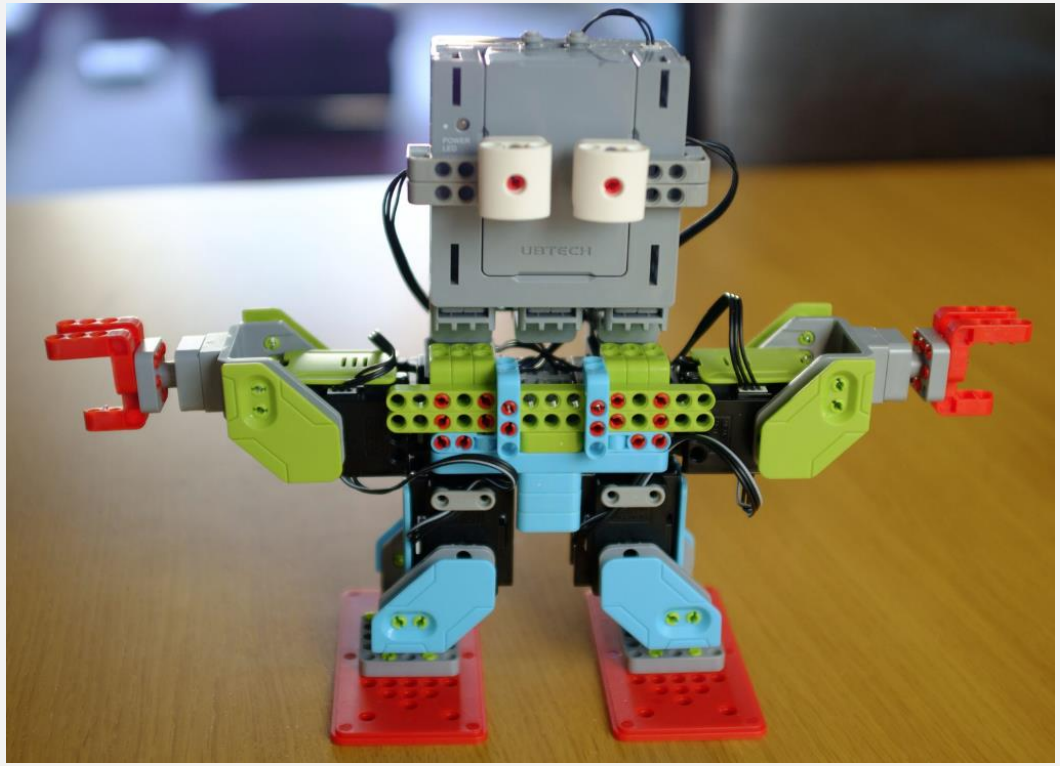
I’m going to talk about the Microsoft Bot Framework

Open Source, can be hosted anywhere

Grown to version 4.0



Bots are best as dumb devices



We don’t have humanoid robots helping us do the housework, but we do have washing machines and vacuum cleaners.

These are robots that have been specifically designed to perform a simple task

The same is true when we think about bots.

Don’t consider them as “your plastic pal who’s fun to be with”.

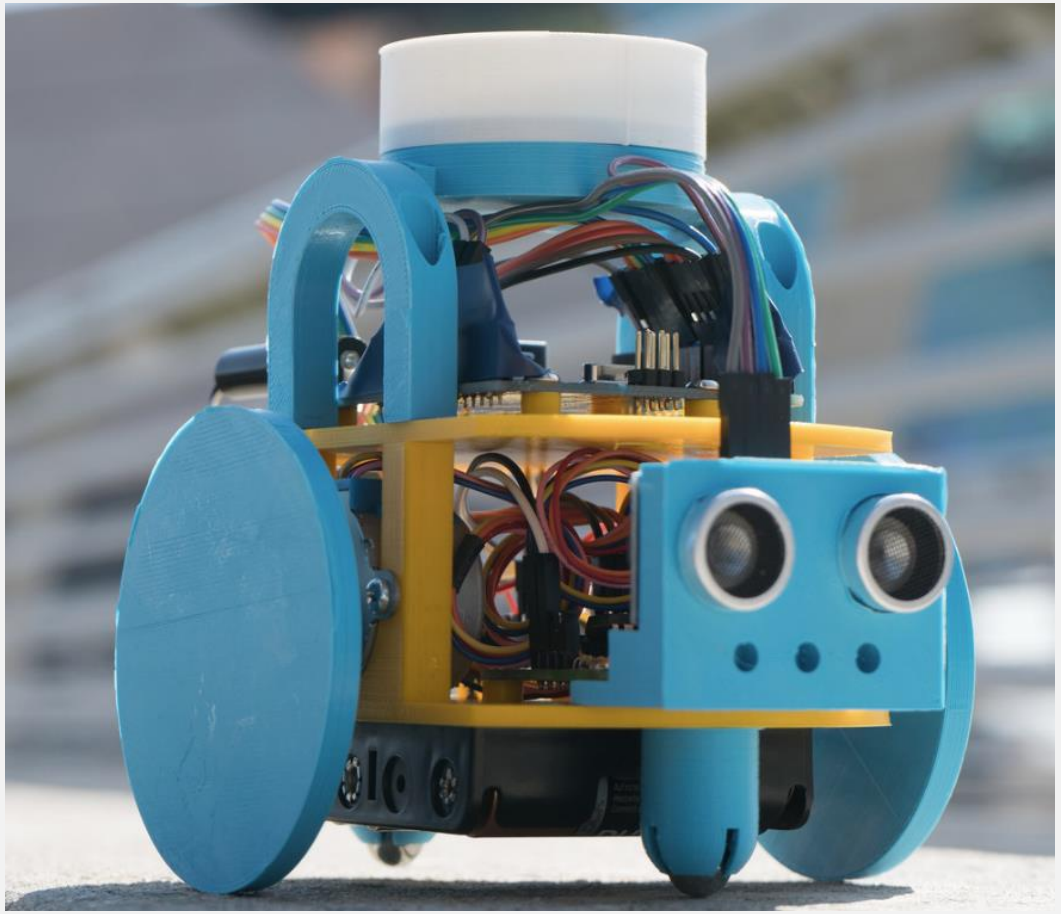
Think of them as something that you can use to book a flight, reserve a game or ask about the weather.

Bots must be helpful rather than human

A bit of humanity is a good idea, but it is much more useful if the bot can respond in a helpful way when it fails to understand what is going on

We will see how to give bots “personality”

You can also build in the ability to “escalate” if a bot user needs human attention



Making bots

A bot is a data processor

In its simplest form a bot is takes something in, does something with it, and then produces an output

This is a classic model of a computer. When we look at the bot framework in a minute we'll see something that lets us take something and push back a reply

Bots vs Wizards

A good way to regard a bot is as a wizard that guides the user through a process. When you are booking a haircut or changing your password you don't want conversation, just a particular outcome. Considering it this way and recognising the importance of context will make for great bots

You don't need to understand natural language

You can get a long way with regular expressions or just simple text analysis

Context is key

A crucial way to improve your bot is to have it maintain context during the conversation. So that when the user says "it" meaning "the book I'm buying" you can respond sensibly.

Clever Bots

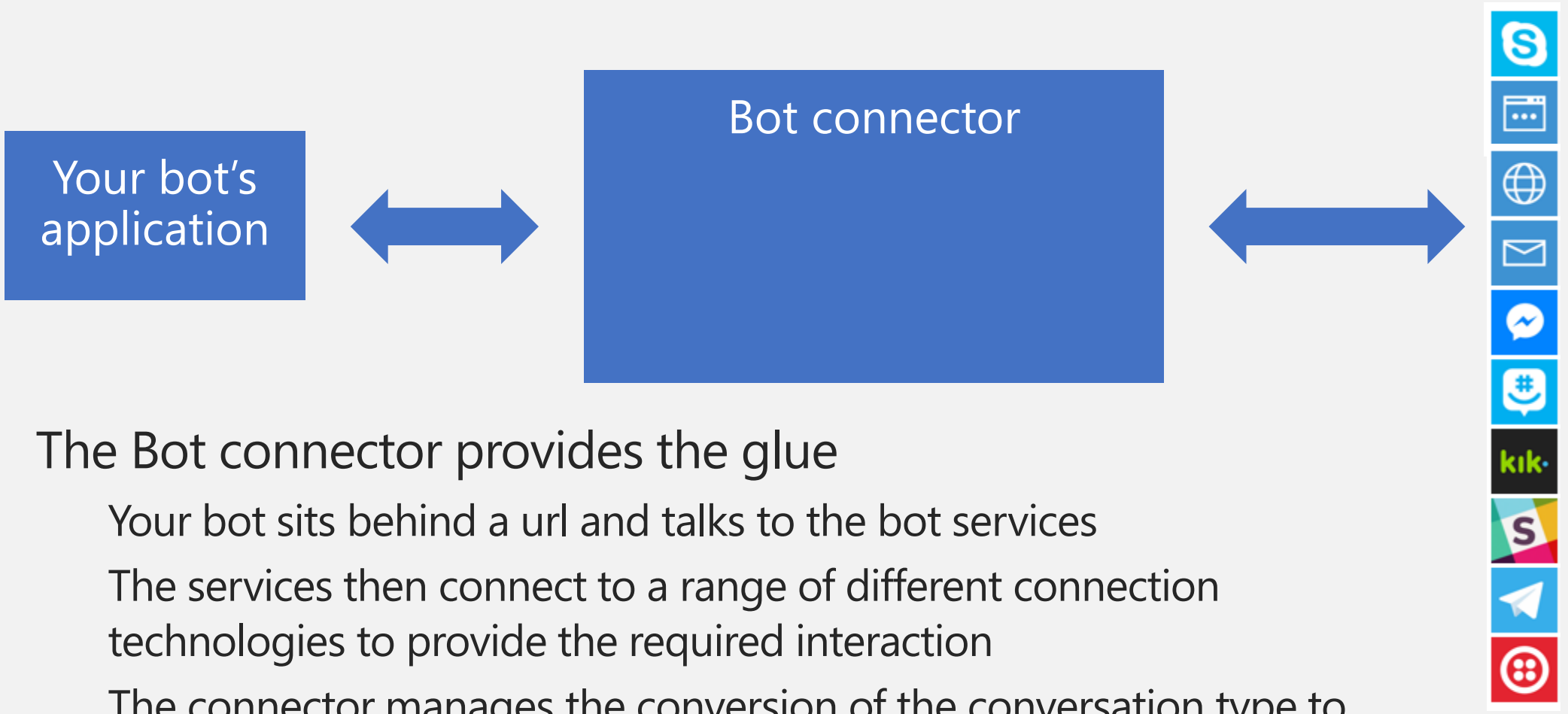
You can make clever bots with LUIS

- The LUIS (Language Understanding Intelligent Service) can be used to improve how well your bot understands natural language
- It adds voice control
- You can train LUIS to work in your particular context

You can make bots that have empathy with their users

- There are even tools that let your bot determine how happy your user is – so that you can automatically escalate a conversation
- We're not going to do anything like that though, we are just going to take in text, do something with it and then send back a response

Microsoft bot framework architecture



The Bot connector provides the glue

- Your bot sits behind a url and talks to the bot services
- The services then connect to a range of different connection technologies to provide the required interaction
- The connector manages the conversion of the conversation type to the channels

Bot Options

C# and .NET

- Write C# applications that contain bot instances

JavaScript and Node

- Embed in web page using JavaScript and Node

REST API

- Create restful calls from any device that you fancy

Getting Started

- https://dev.botframework.com/

Bot Options

C# and .NET

- Write C# applications that contain bot instances (this is what we are going to do)

JavaScript and Node

- Embed in web page using JavaScript and Node

REST API

- Create restful calls from any device that you fancy

Getting Started

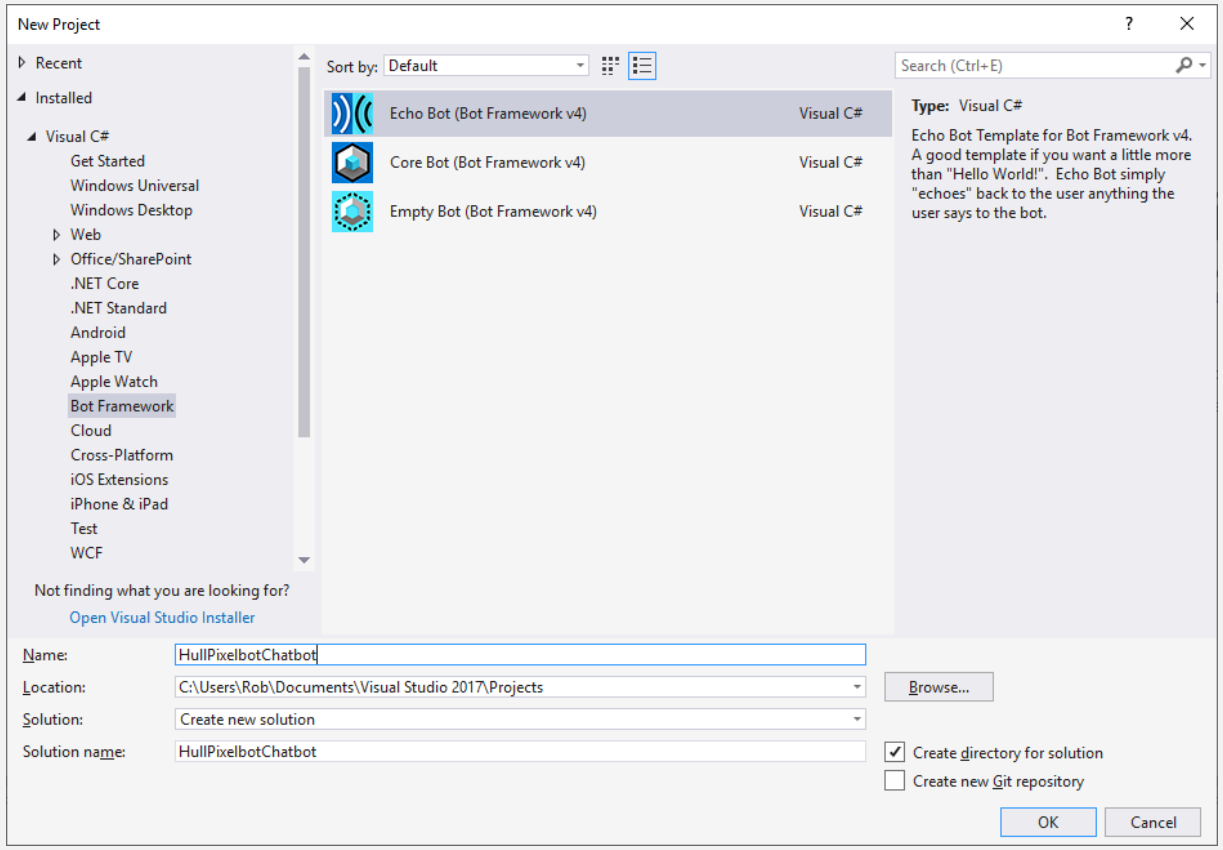
- https://dev.botframework.com/

Chatbot project templates

Download the template

Make a new C# project from the template

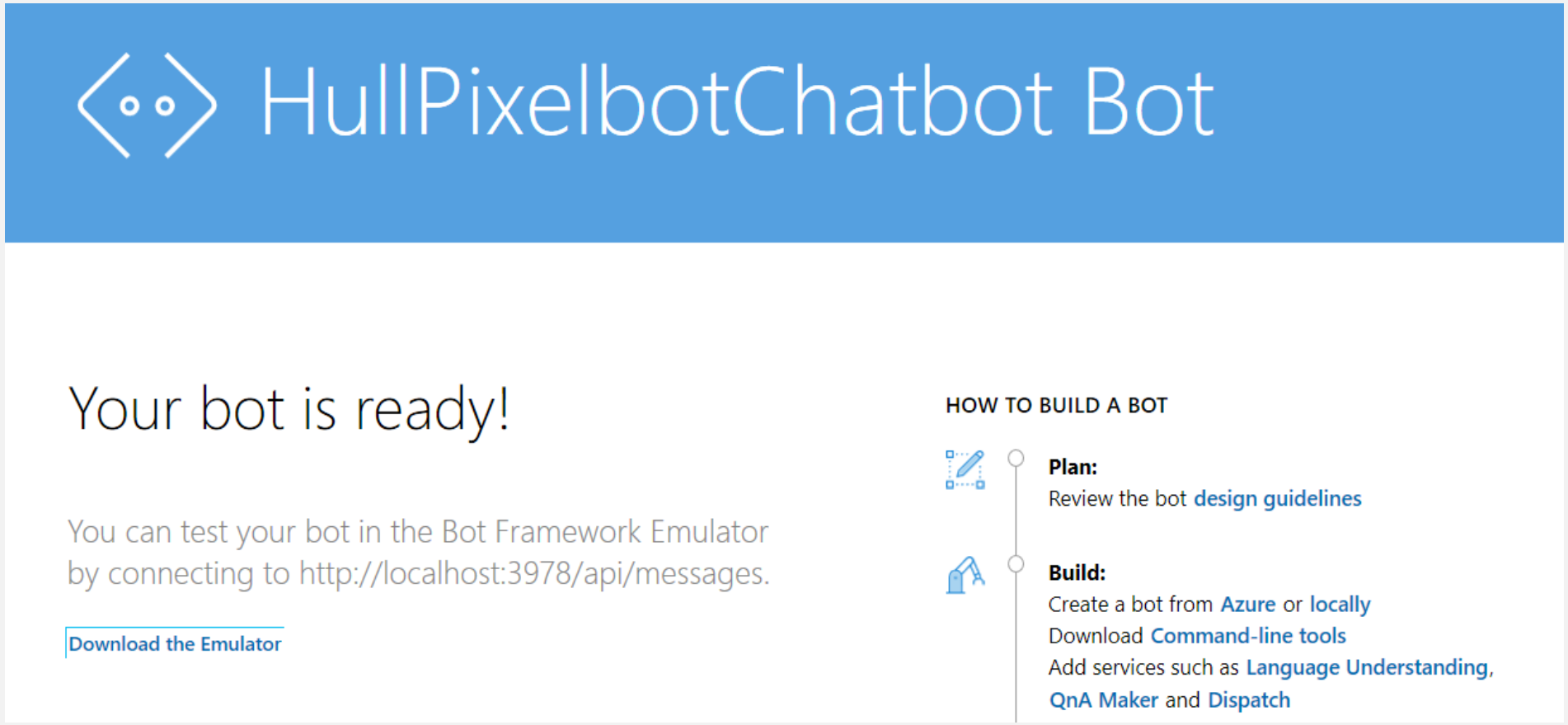
There are numerous versions, I'm starting with the echo one



Template available from visx:

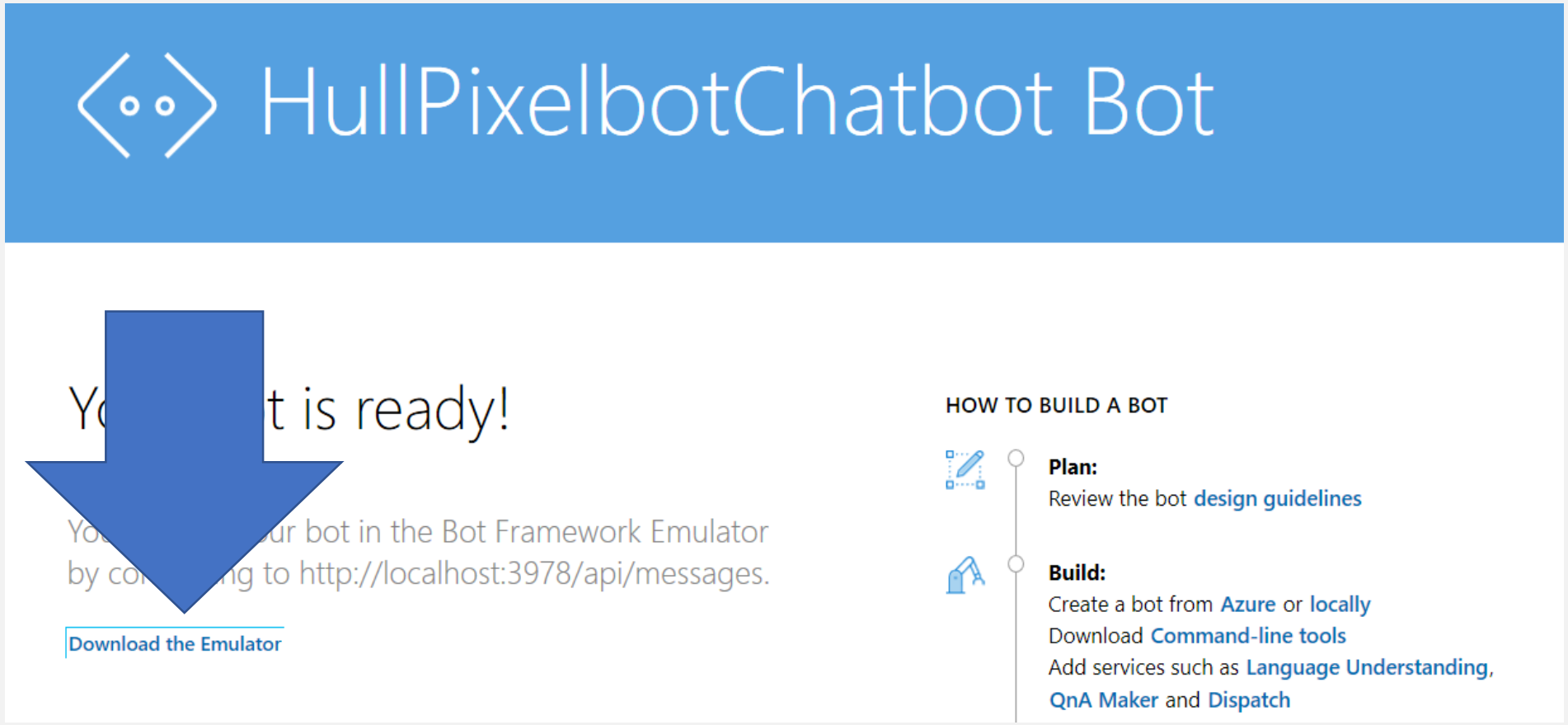
https://marketplace.visualstudio.com/items?itemName=BotBuilder.botbuilderv4

Talking to the chatbot



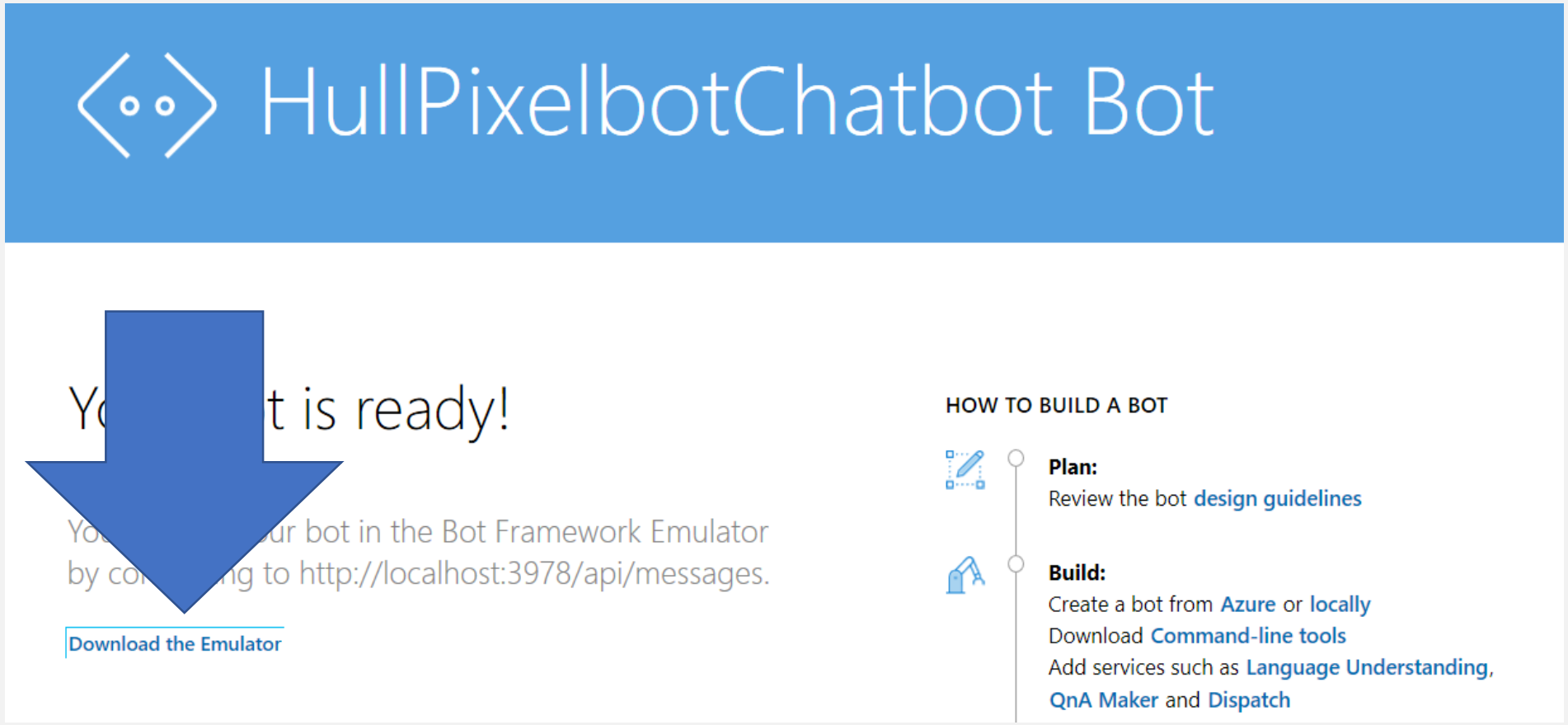
We can fire up our chatbot and talk to it

Talking to the chatbot



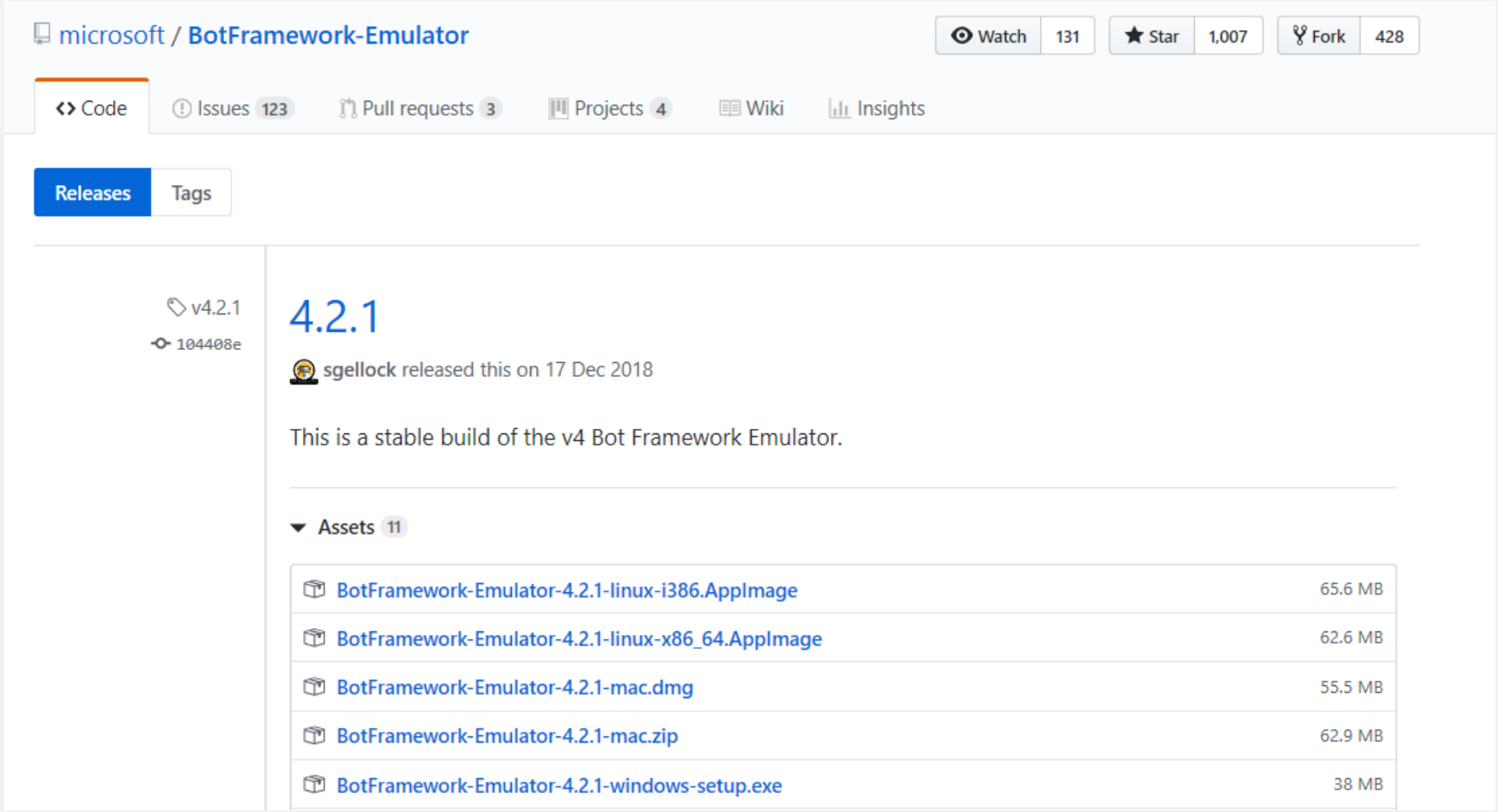
Use the emulator for this

Talking to the chatbot



The emulator can send and receive messages to the bot directly

Getting the emulator



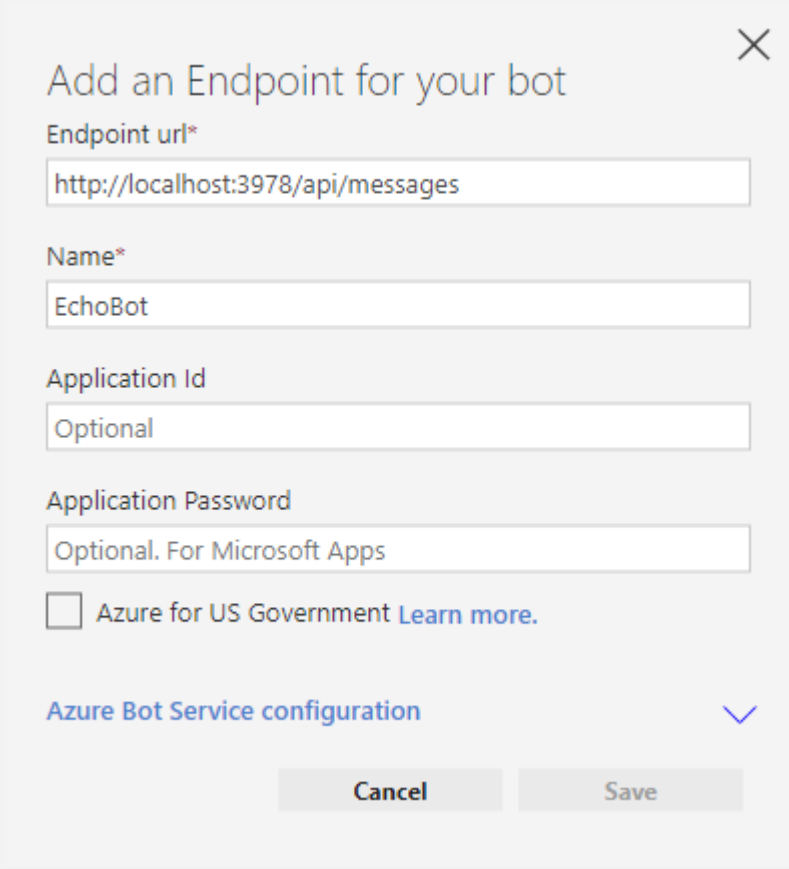
The emulator lives on GitHub

Configuring the emulator

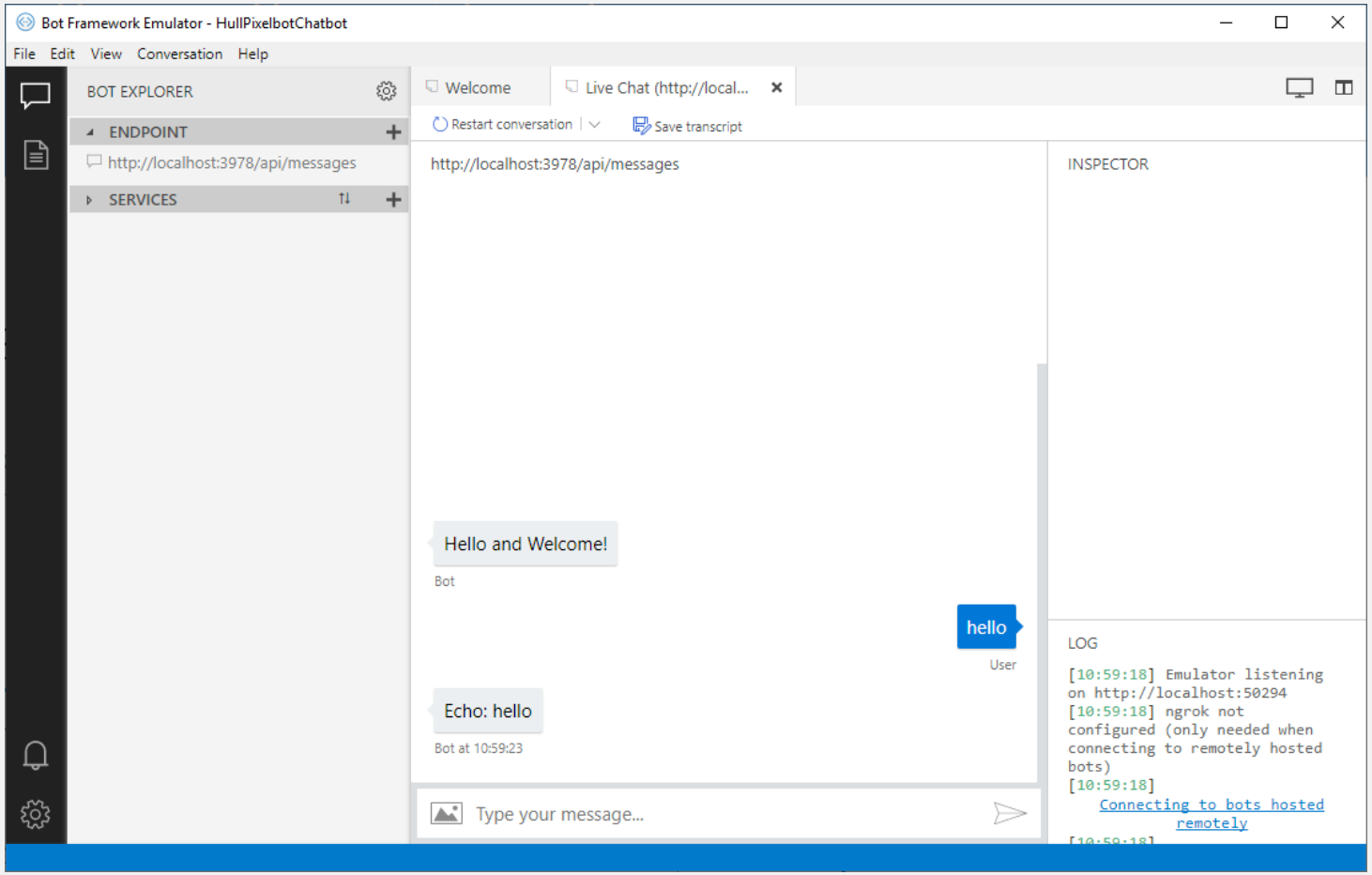
For testing we connect the emulator to the locally hosted bot

This means that we can debug the code behind the bot

We can also connect to a bot that is hosted remotely so that we can see the messages exchanged by the devices



Talking to the Echo bot



The echo bot just echoes back whatever was sent

How the echo bot works

```
public class EchoBot : ActivityHandler
{
    protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
                                                         CancellationToken cancellationToken)
    {
        await turnContext.SendActivityAsync(
              MessageFactory.Text($"Echo: {turnContext.Activity.Text}"), cancellationToken);
    }
}
```

The EchoBot class contains methods that are called when messages arrive for a bot

Your code can manage the incoming messages at this point

How the echo bot works

```
public class EchoBot : ActivityHandler
{
    protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
                                                         CancellationToken cancellationToken)
    {
        await turnContext.SendActivityAsync(
              MessageFactory.Text($"Echo: {turnContext.Activity.Text}"), cancellationToken);
    }
}
```

This is the method that sends the message back to the client

The message is just a string of text

Demo

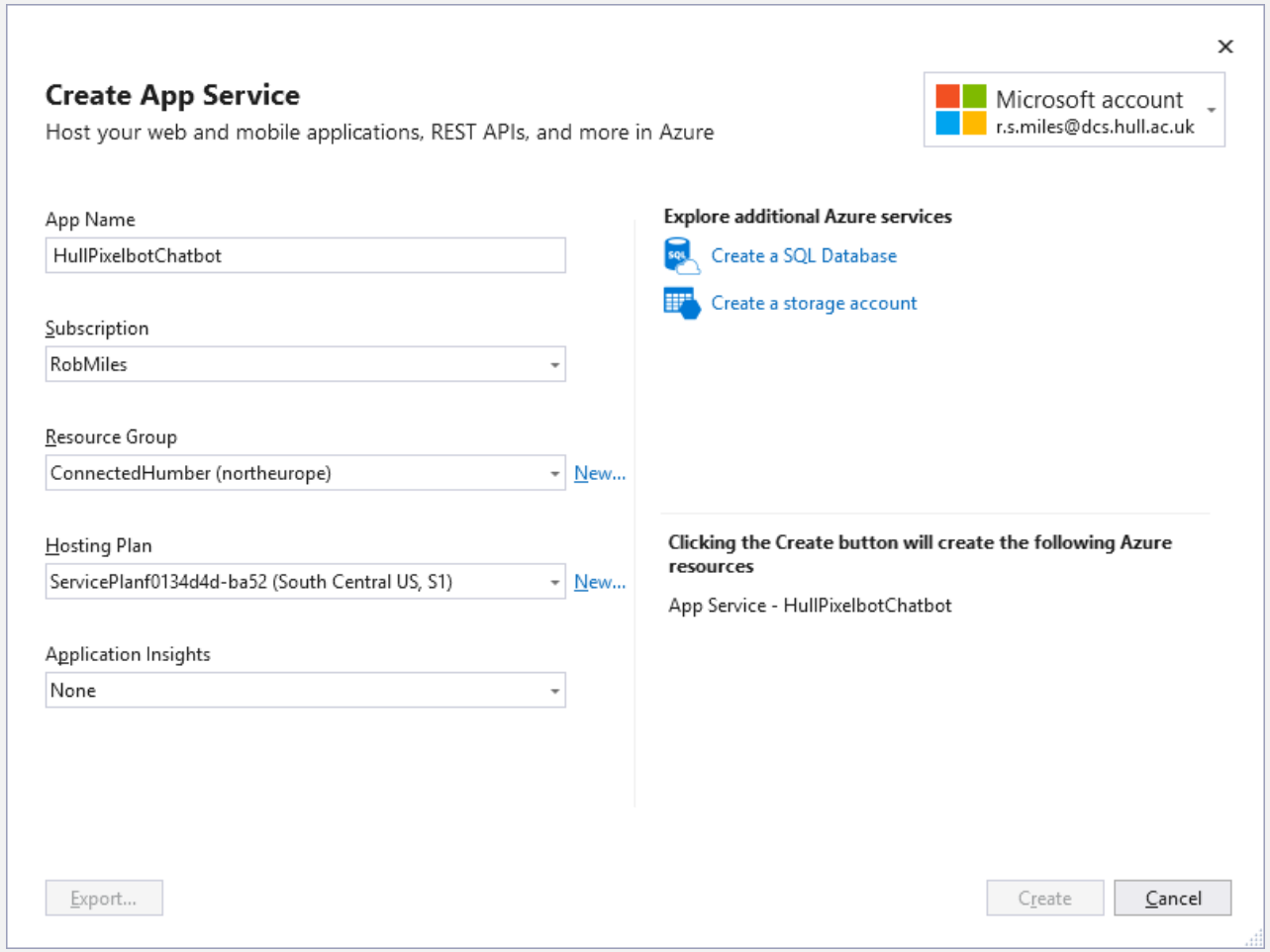
Build an echo bot and talk to it

Deploying the bot service to the cloud

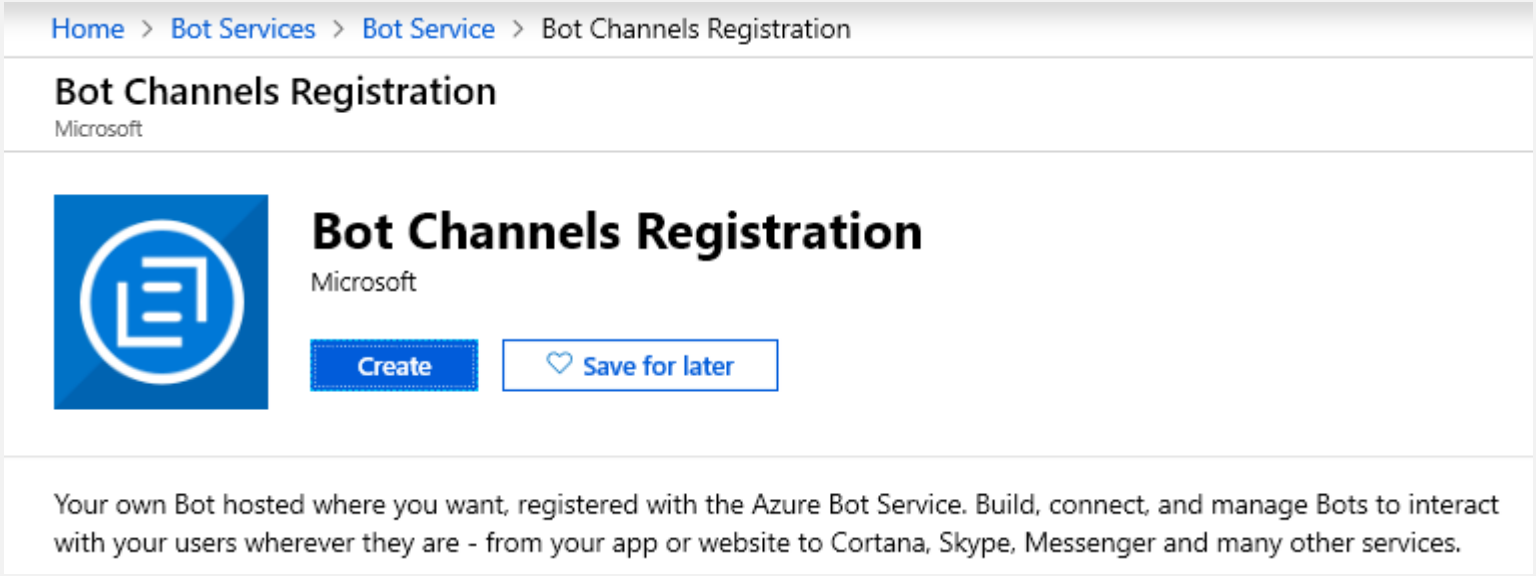
Once you have made your bot service and tested it you can deploy it to the cloud

This just provides the service behind the endpoint

We can set up a Channel registration service to provide the connection to users



Bot registration

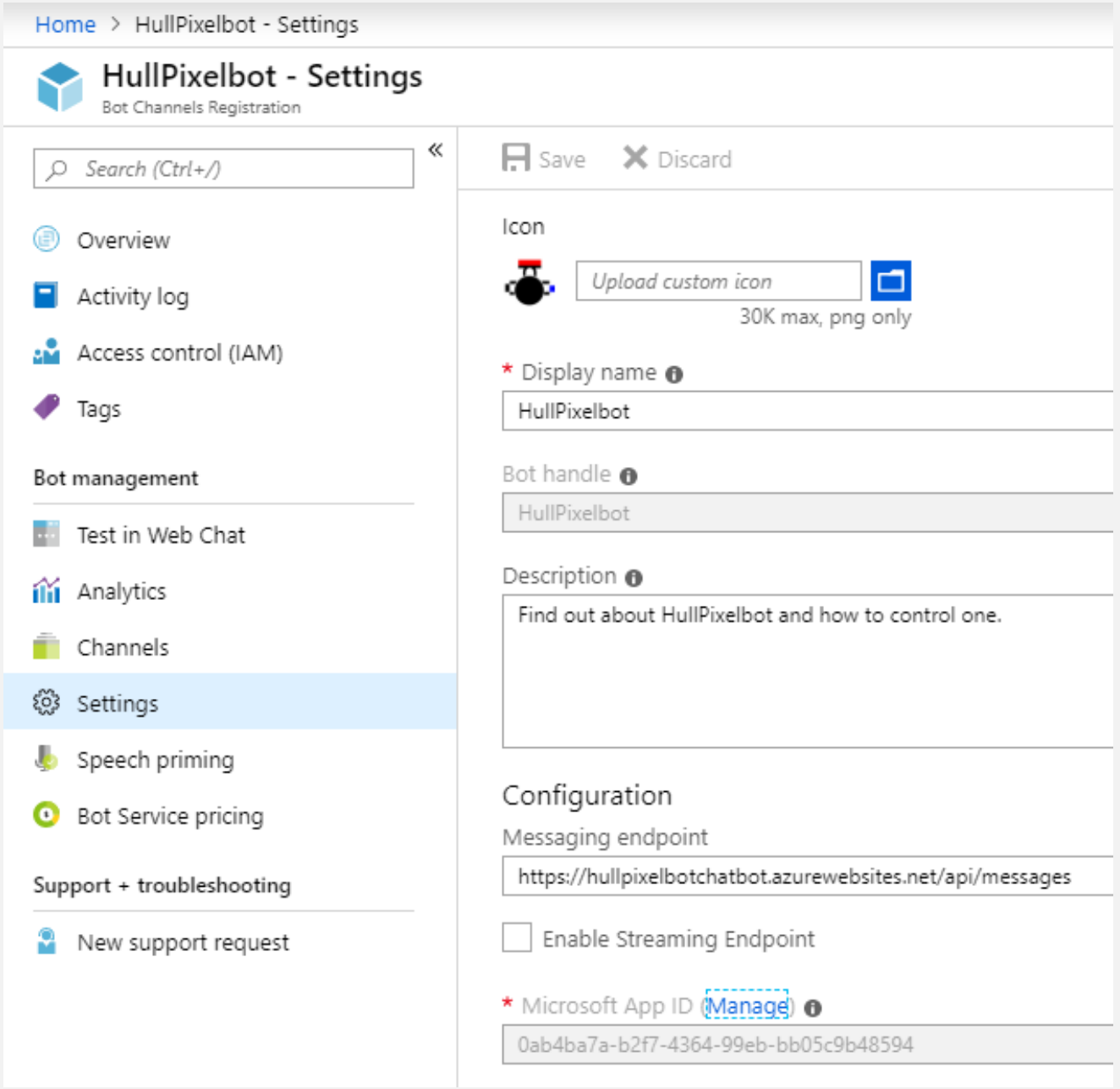


A separate service manages the channels for a bot

The channels and the service can be on completely different servers

Bot registration

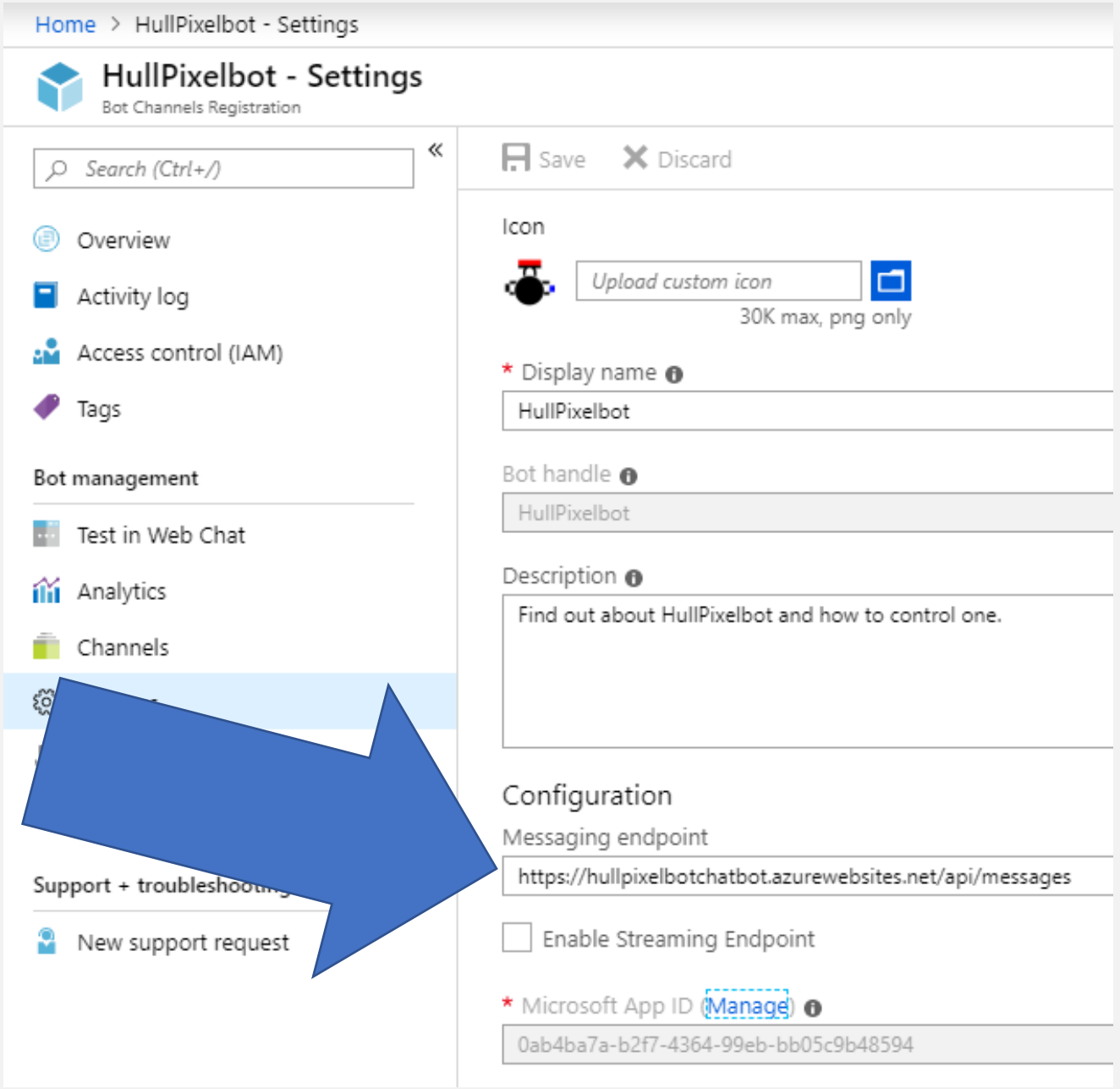
The Channel Registration service is managed as any other Azure service via the Azure portal



Bot registration

We configure the channel registration with the messaging endpoint

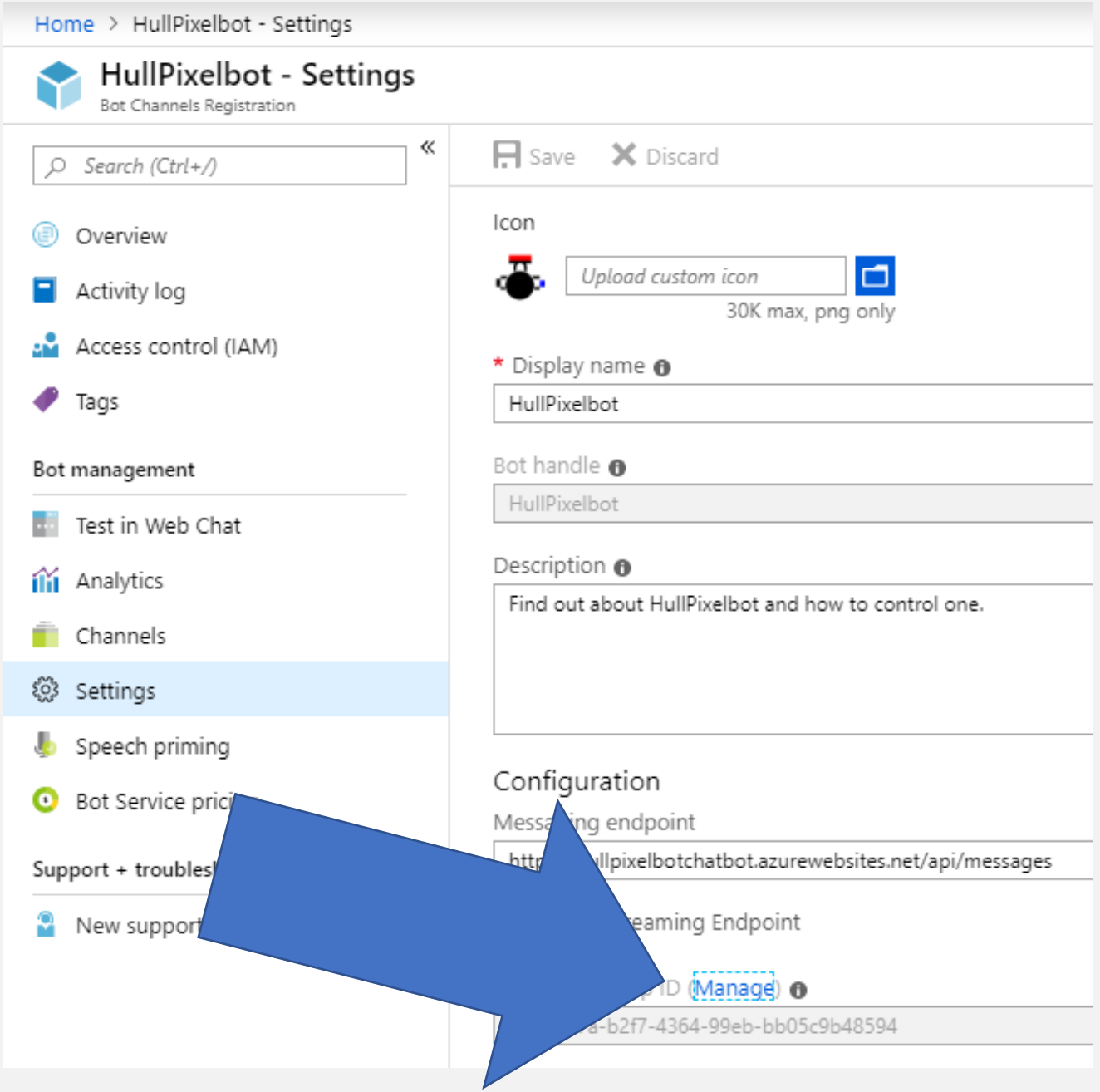
This is the connection between the channels and the bot service



Authentication

A client must authenticate to the bot when it connects

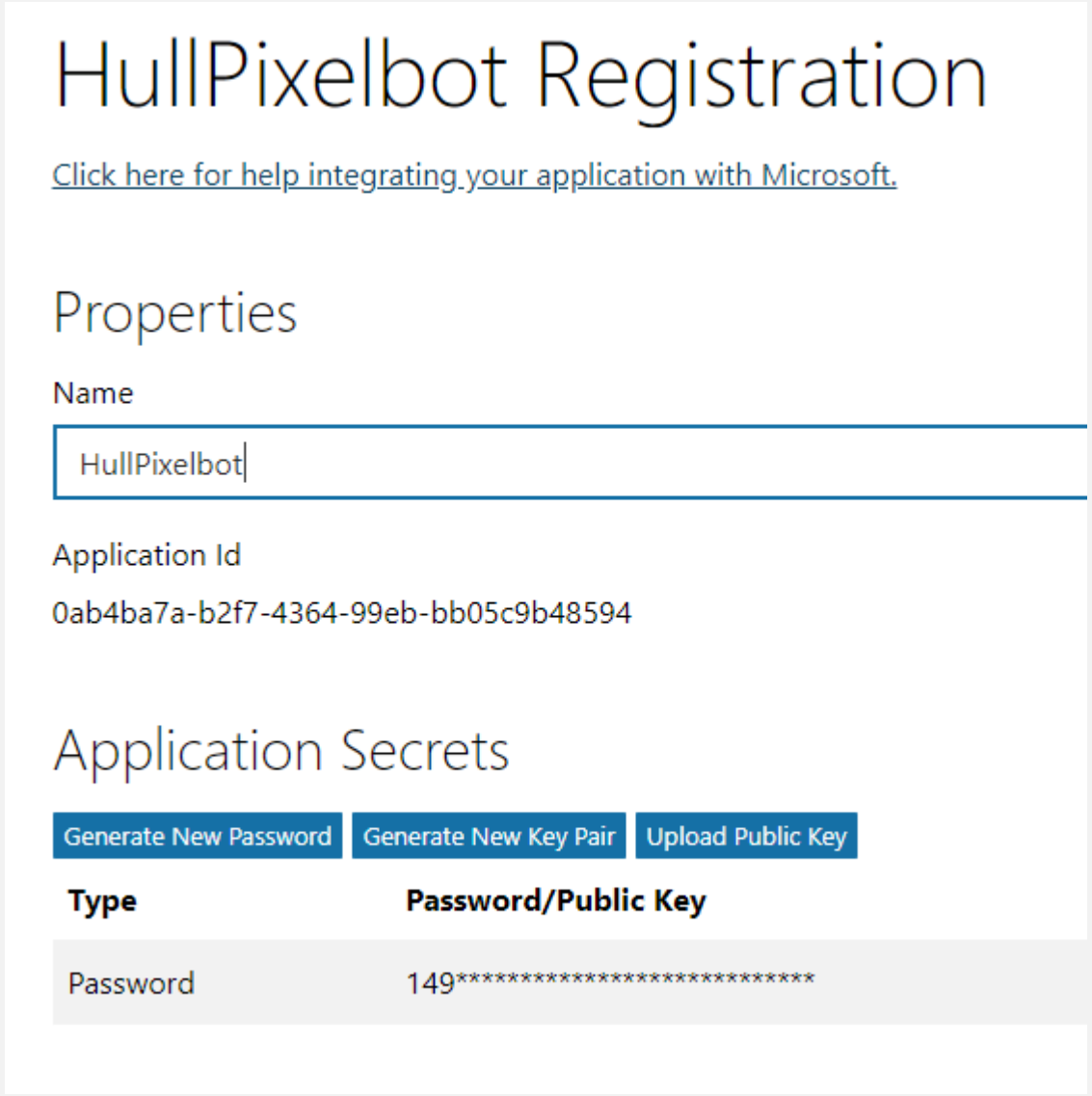
We need to set the app id and key in the client service



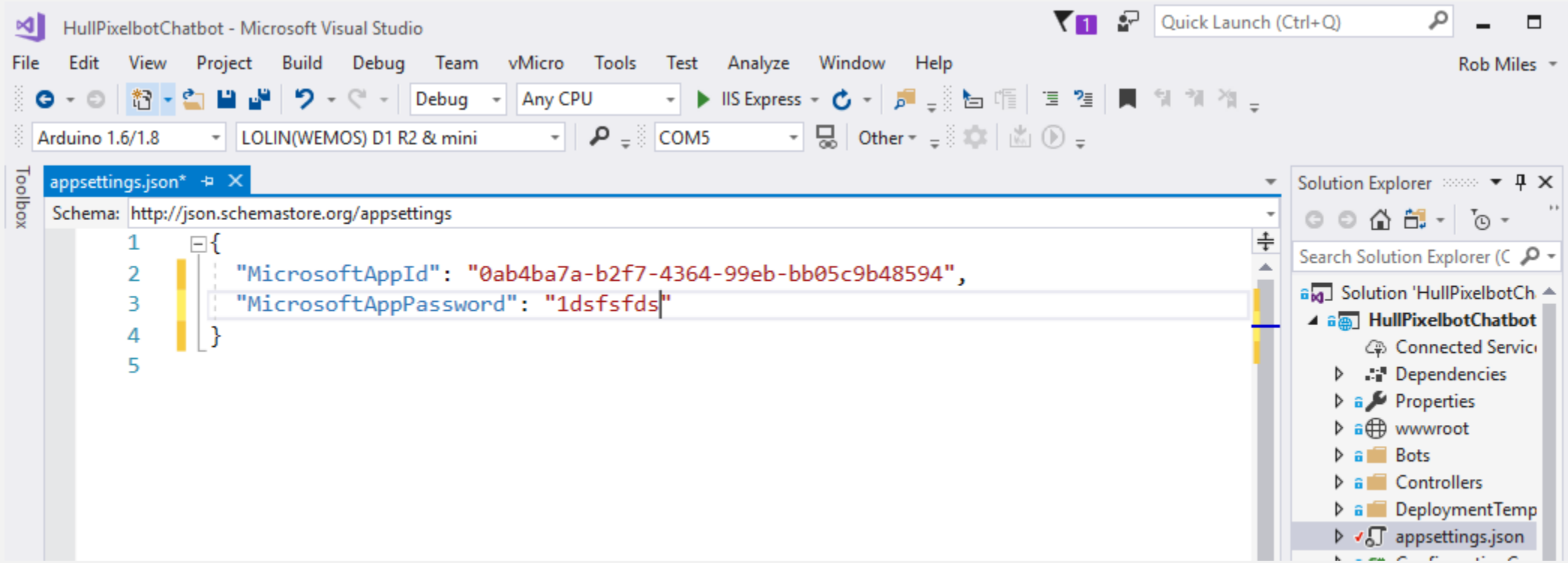
Getting the keys

I can create a password for inclusion in the application

Need to be careful with this, we only see the password once, when it is generated



Setting the keys



The bot service contains the app id and password in a config file

These must be set before we host the service in the cloud

To recap:

We create our service that implements a bot

- The service responds to text inputs and generates text outputs

We place the application in the cloud as a service

We create a channel connector that is mapped to the endpoint of our service

- We use an application ID and key to allow the connector to authenticate the service

To recap:

We create our service that implements a bot

The service responds to text inputs and generates text outputs

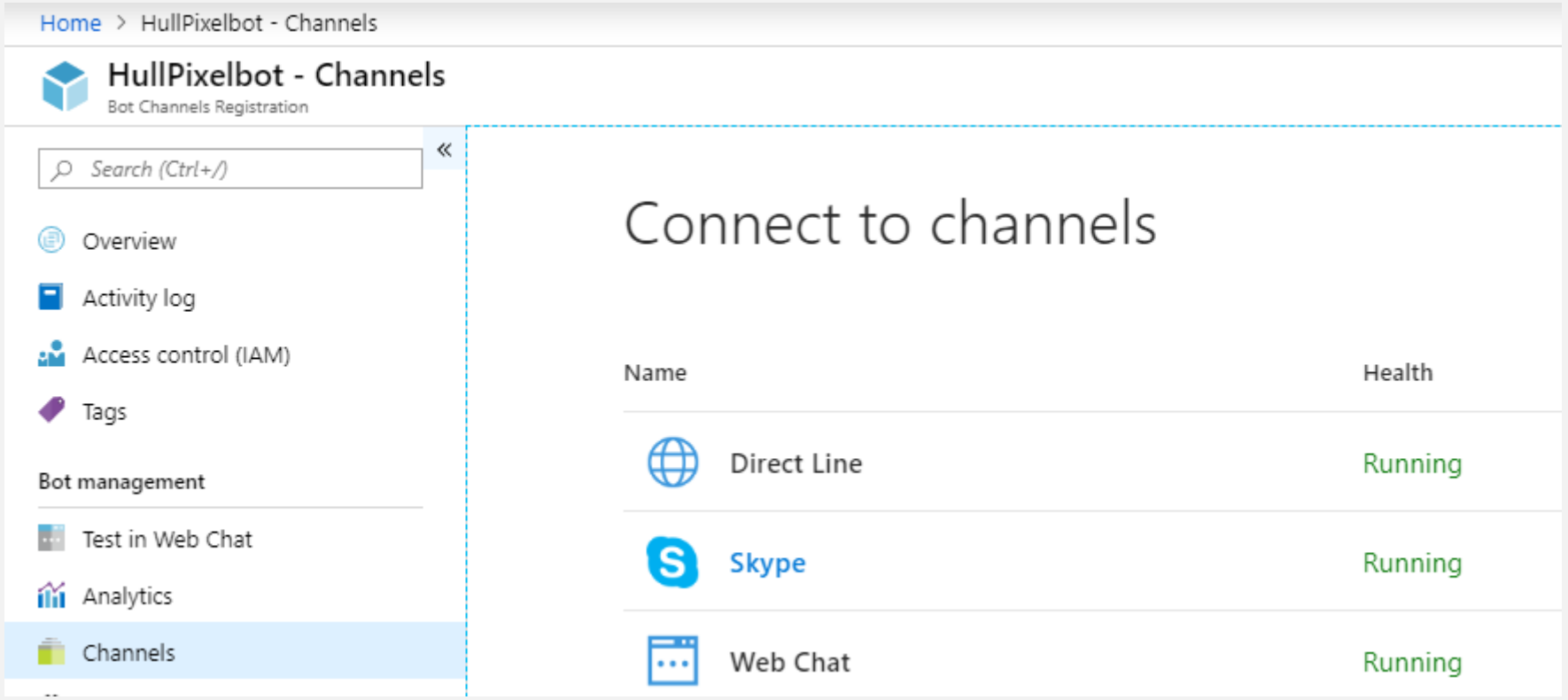
We place the application in the cloud as a service

We create a channel connector that is mapped to the endpoint of our service

We use an application ID and key to allow the connector to authenticate the service

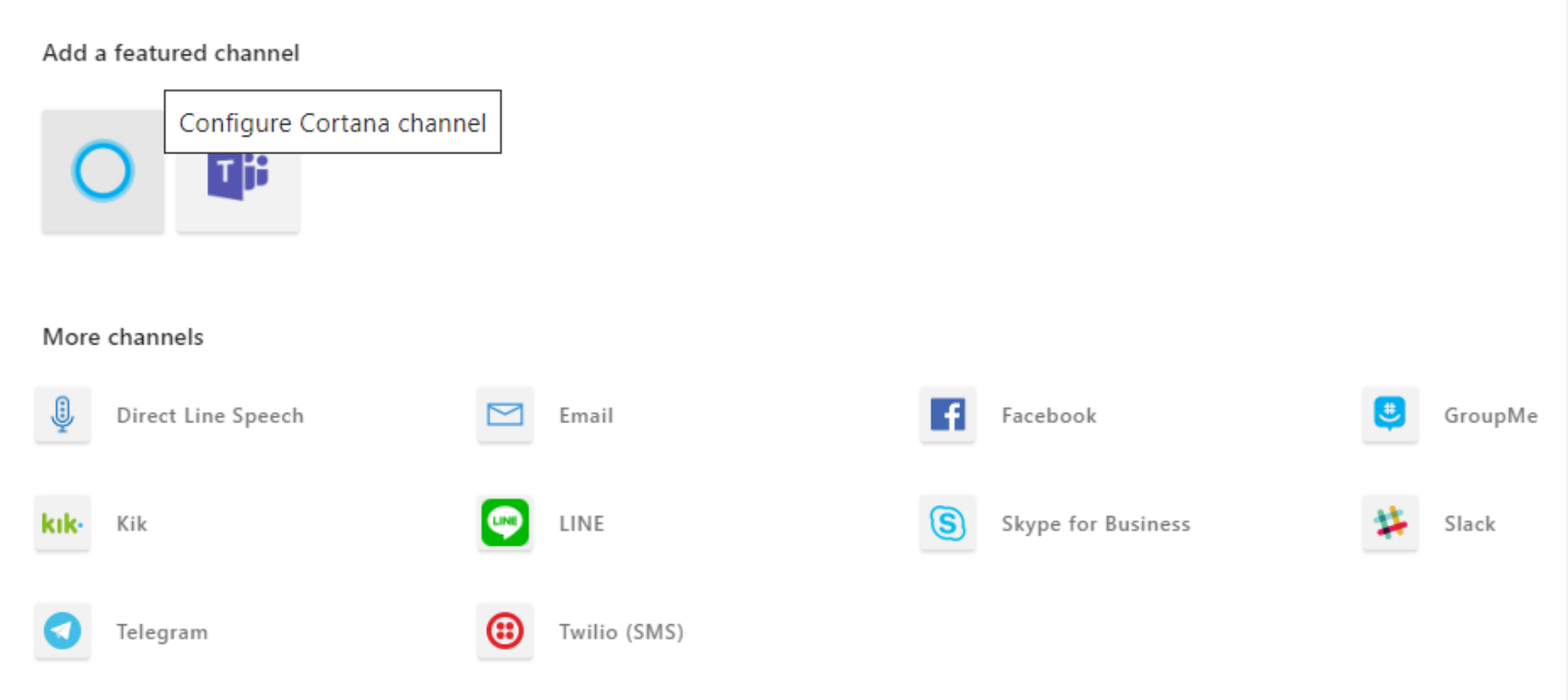
Now we tell the connector which channels we want to use

Connecting channels



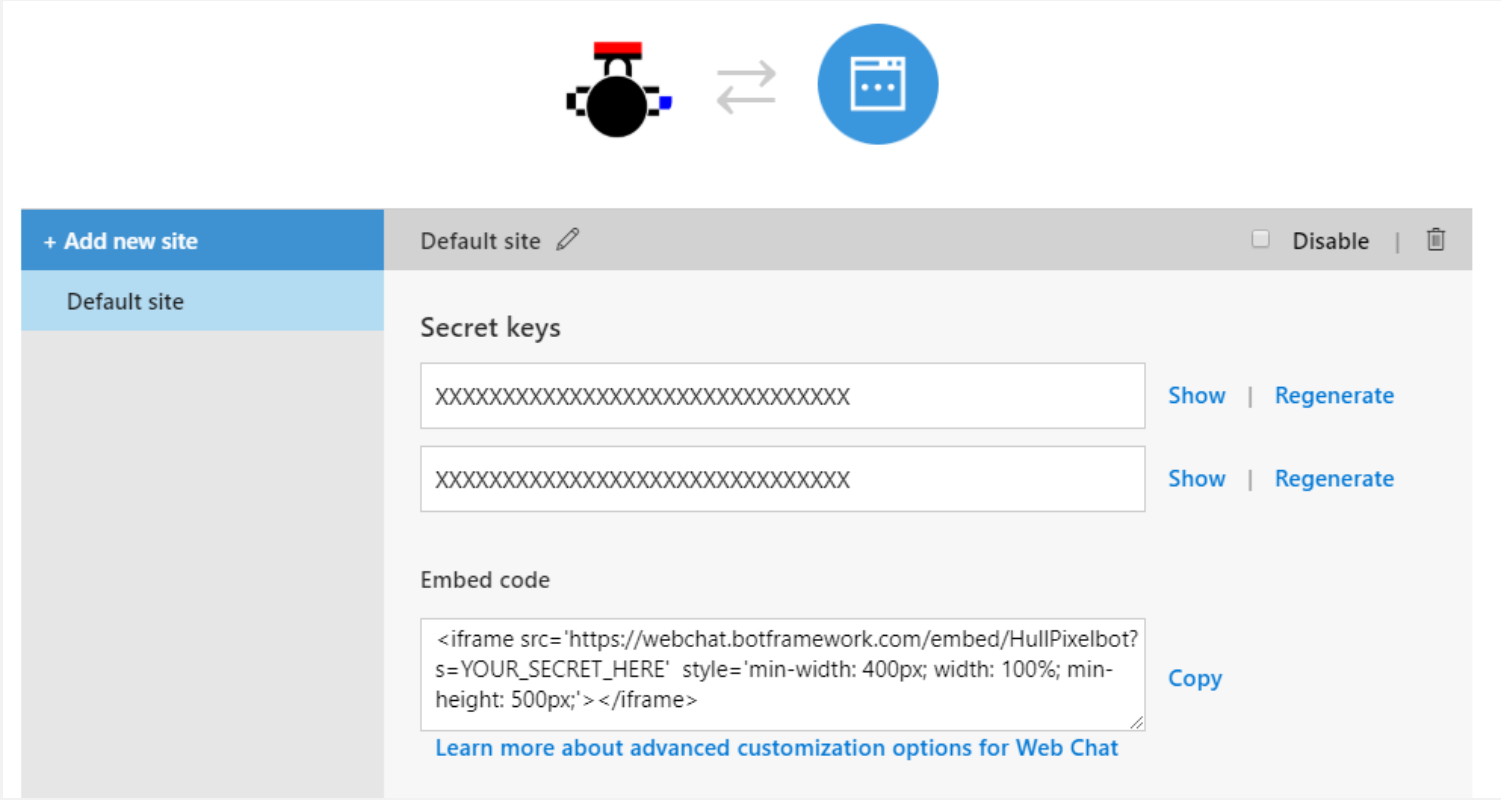
We can manage each channel connection individually

Adding channels



There are lots of channel connectors available including Cortana, Teams, Slack, email and Skype

Web embed codes

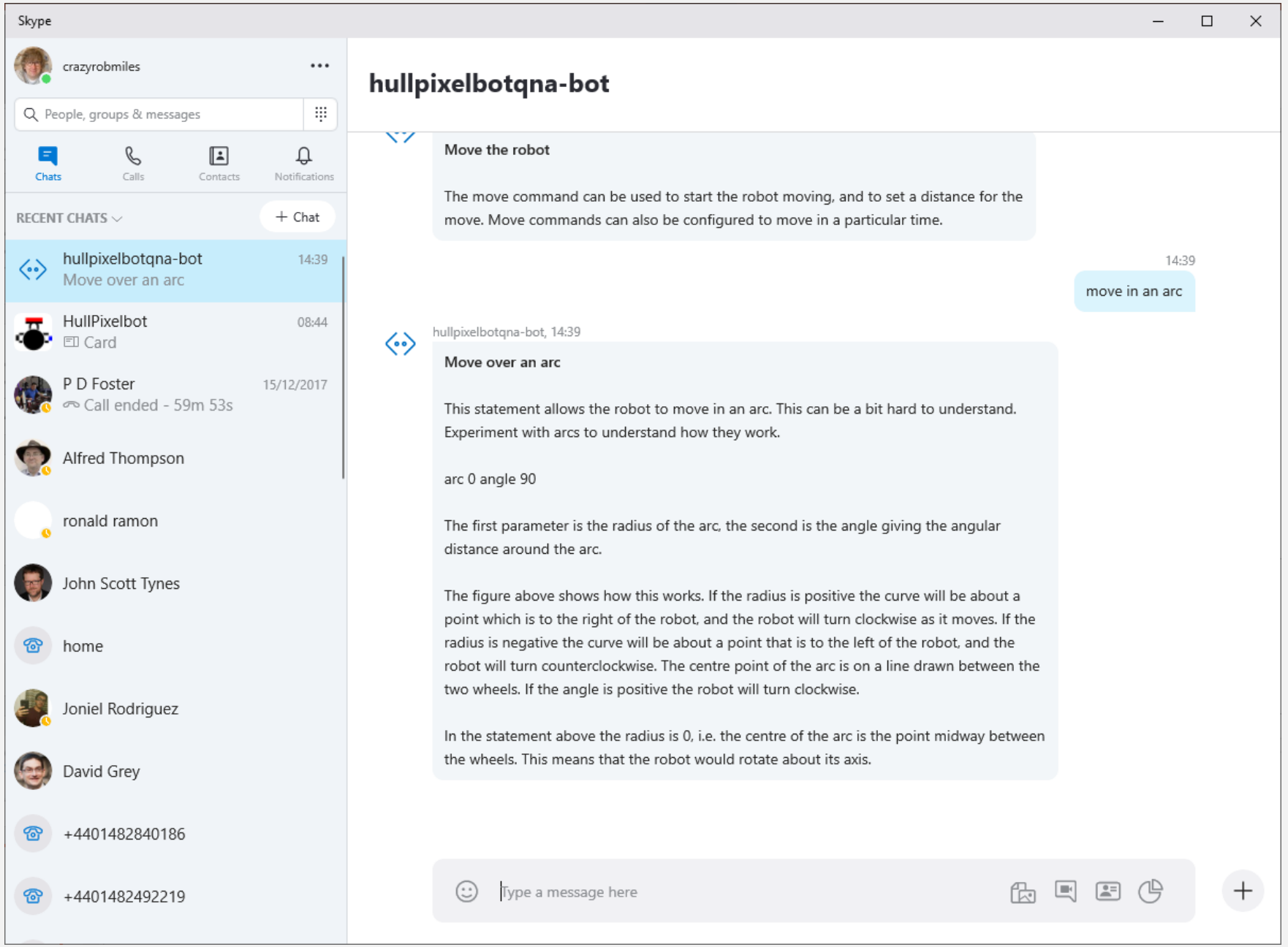


We can generate lumps of html to drop into web pages to embed the chatbot into web pages

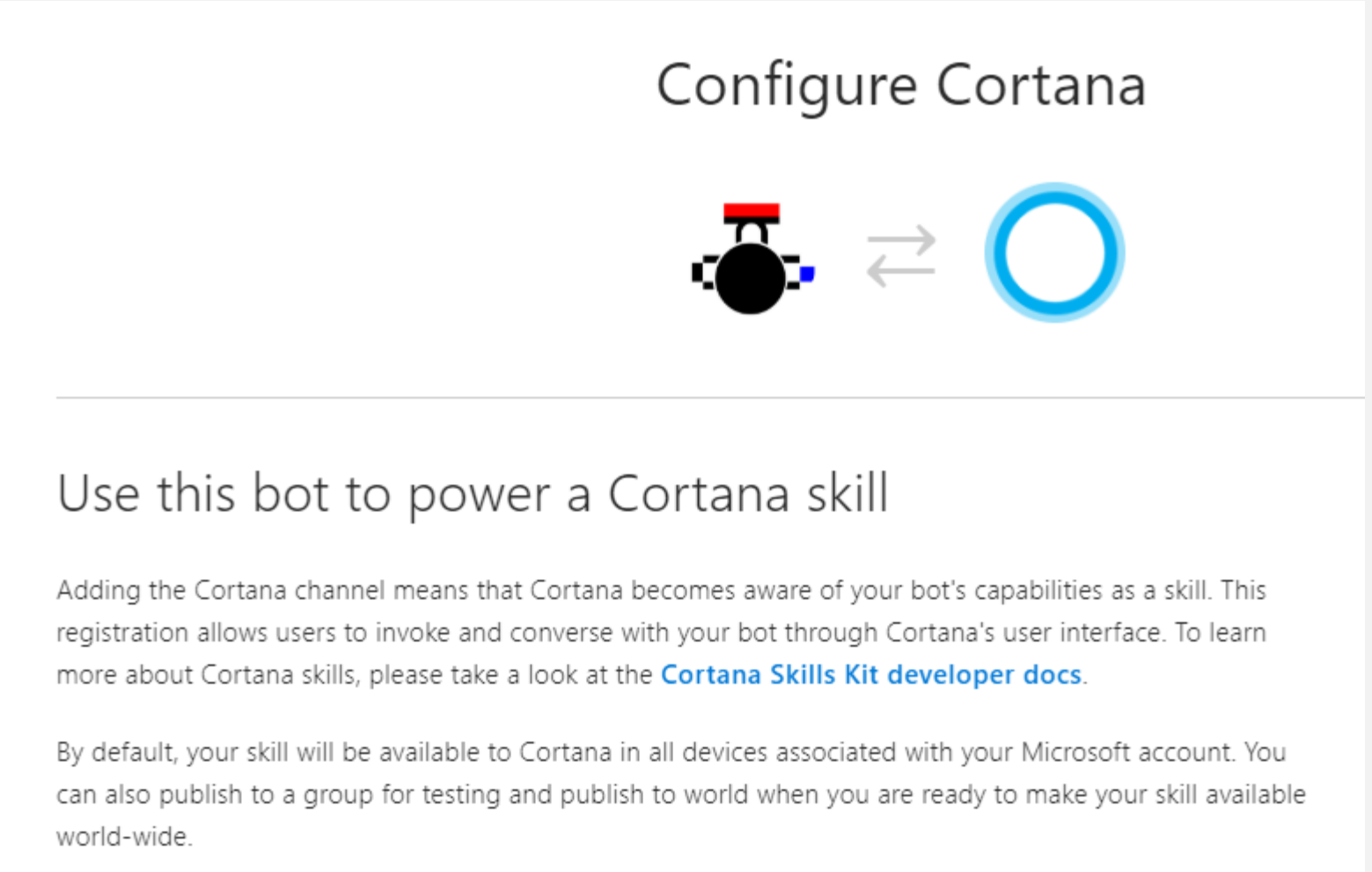
Skype

We can use skype to have a chat with the chatbot

You can ask this robot questions about how to program it



Cortana



You can also create a channel that links a chatbot to a Cortana skill

These can be published for the whole world to use

Adding richness to the interface

A bot sends and receives text

We can add richness to the interface by using adaptive cards

These are an open standard that we can use to display graphics and implement things like buttons for the user to press

How they work will depend on the channel used to deliver the interface

- At the moment we can’t use adaptive cards with the Skype channel which is rather sad ☹

Adaptive Cards

Adaptive cards are an exchange format for user interface elements

They are lumps of JSON that describes the user interface intent

Then the target framework can render them however they see fit

We can use them with our chatbot

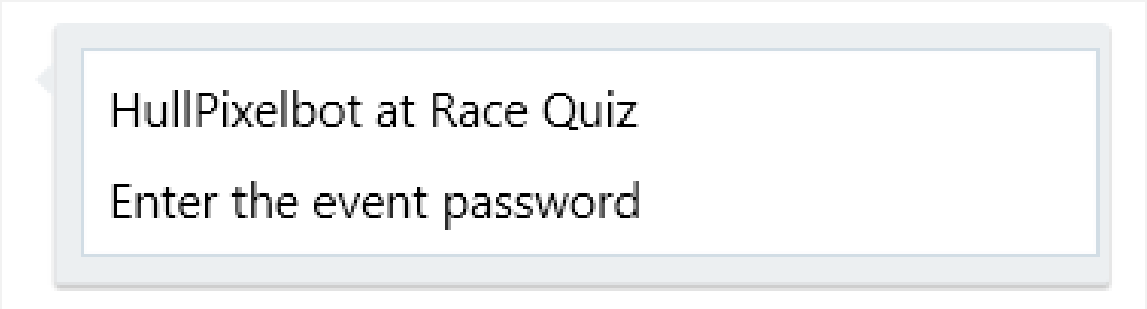
You can find out more about them here:

https://adaptivecards.io/

Filling in a card

```
AdaptiveCard result = new AdaptiveCard();

result.Body.Add(new AdaptiveTextBlock()
{
    Text = "HullPixelbot at " + eventName,
    Size = AdaptiveTextSize.Medium
});
result.Body.Add(new AdaptiveTextBlock()
{
    Text = "Enter the event password",
    Size = AdaptiveTextSize.Medium
});
```

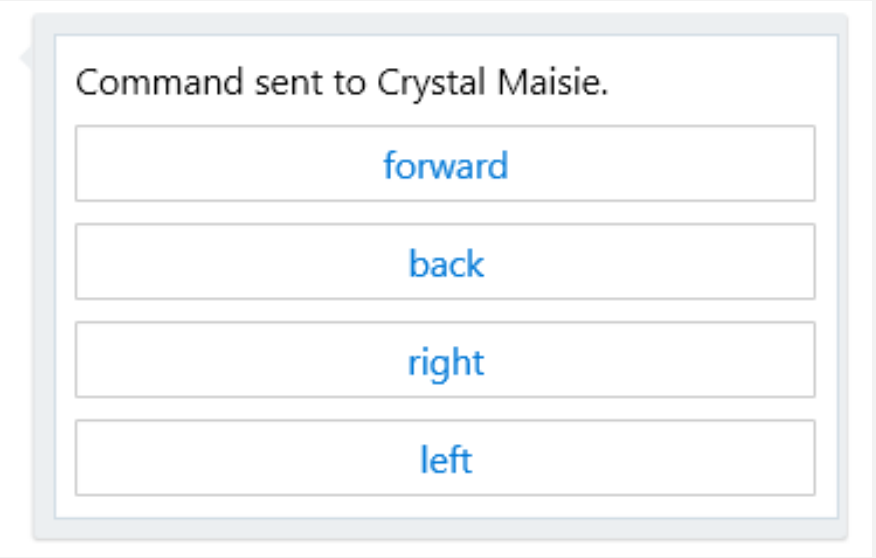


- An `AdaptiveCard` is a container object
- We can add elements and set the properties of the elements to inform the rendering process

Getting card options

```
AdaptiveCard result = new AdaptiveCard();

private void addOptionsToCard(AdaptiveCard card,
                    IEnumerable<string> options)
{
    foreach (string str in options)
    {
        card.Actions.Add(new AdaptiveSubmitAction()
           { Title = str, Data = str });
    }
}
```



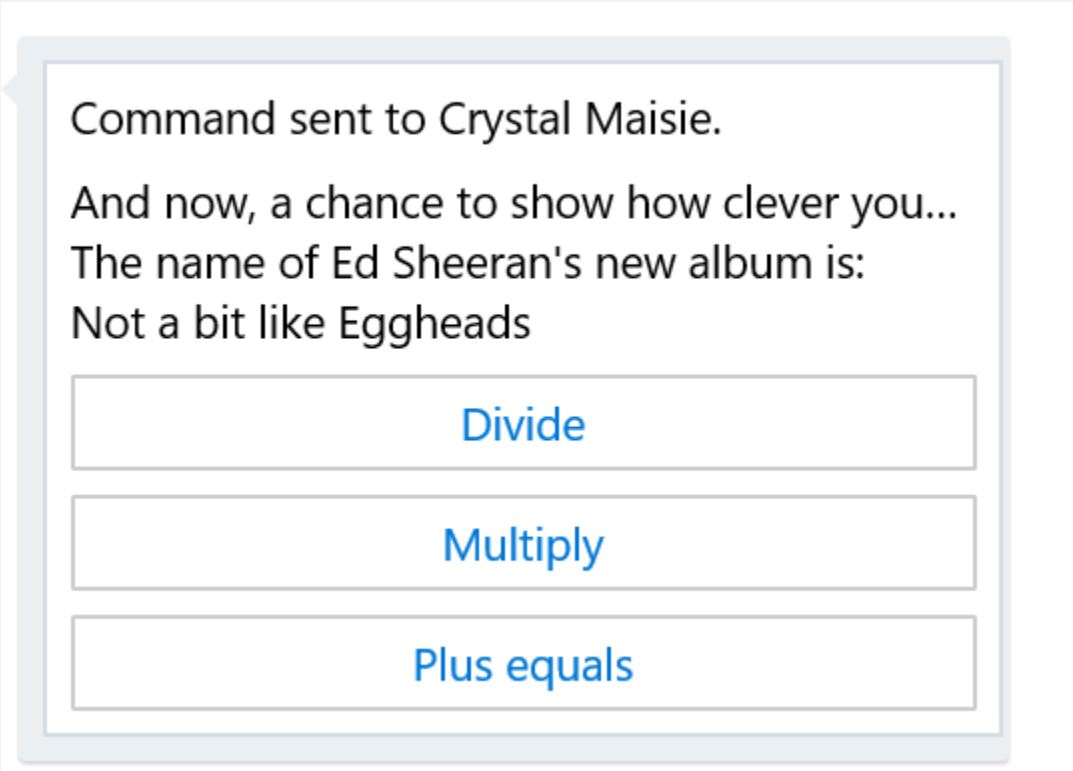
- We can use `AdaptiveSubmitAction` items on the cards to allow the user to input by clicking the option
- The data item of the item is submitted as if it had been entered at the keyboard

Adding a Quiz

I added a quiz to the chatbot

This asks questions in between letting you control the robot

The players can control the robot but must answer questions every now and then.



Linking a chatbot to a robot

```
protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    setupEvent();

    AdaptiveCard actionResult = await ActiveEvent.ActOnCommand(turnContext.Activity.Text,
                                    turnContext.Activity.Conversation.Id);

    Attachment actionAttachement = new Attachment(
          contentType: "application/vnd.microsoft.card.adaptive", content: actionResult);

    Activity response = CreateResponse(turnContext.Activity, actionAttachement);

    await turnContext.SendActivityAsync(response, cancellationToken);
}
```

This is the code that links the chatbots to my robot event

Linking a chatbot to a robot

```
protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    setupEvent();

    AdaptiveCard actionResult = await ActiveEvent.ActOnCommand(turnContext.Activity.Text,
                                    turnContext.Activity.Conversation.Id);

    Attachment actionAttachement = new Attachment(
          contentType: "application/vnd.microsoft.card.adaptive", content: actionResult);

    Activity response = CreateResponse(turnContext.Activity, actionAttachement);

    await turnContext.SendActivityAsync(response, cancellationToken);
}
```

Set the event up (only does something the first time it is called)

Linking a chatbot to a robot

```
protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    setupEvent();

    AdaptiveCard actionResult = await ActiveEvent.ActOnCommand(turnContext.Activity.Text,
                                    turnContext.Activity.Conversation.Id);

    Attachment actionAttachement = new Attachment(
          contentType: "application/vnd.microsoft.card.adaptive", content: actionResult);

    Activity response = CreateResponse(turnContext.Activity, actionAttachement);

    await turnContext.SendActivityAsync(response, cancellationToken);
}
```

This is the text entered by the user

Linking a chatbot to a robot

```
protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    setupEvent();

    AdaptiveCard actionResult = await ActiveEvent.ActOnCommand(turnContext.Activity.Text,
                                    turnContext.Activity.Conversation.Id);

    Attachment actionAttachement = new Attachment(
          contentType: "application/vnd.microsoft.card.adaptive", content: actionResult);

    Activity response = CreateResponse(turnContext.Activity, actionAttachement);

    await turnContext.SendActivityAsync(response, cancellationToken);
}
```

This is the Id for this conversation (this is how I maintain state)

Linking a chatbot to a robot

```
protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    setupEvent();

    AdaptiveCard actionResult = await ActiveEvent.ActOnCommand(turnContext.Activity.Text,
                                      turnContext.Activity.Conversation.Id);

    Attachment actionAttachement = new Attachment(
          contentType: "application/vnd.microsoft.card.adaptive", content: actionResult);

    Activity response = CreateResponse(turnContext.Activity, actionAttachement);

    await turnContext.SendActivityAsync(response, cancellationToken);
}
```

The `ActOnCommand` method returns an `AdaptiveCard`

Linking a chatbot to a robot

```
protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    setupEvent();

    AdaptiveCard actionResult = await ActiveEvent.ActOnCommand(turnContext.Activity.Text,
                                    turnContext.Activity.Conversation.Id);

    Attachment actionAttachement = new Attachment(
          contentType: "application/vnd.microsoft.card.adaptive", content: actionResult);

    Activity response = CreateResponse(turnContext.Activity, actionAttachement);

    await turnContext.SendActivityAsync(response, cancellationToken);
}
```

This builds a response and sends it back to the user

Demo

Controlling Robots with Chatbots

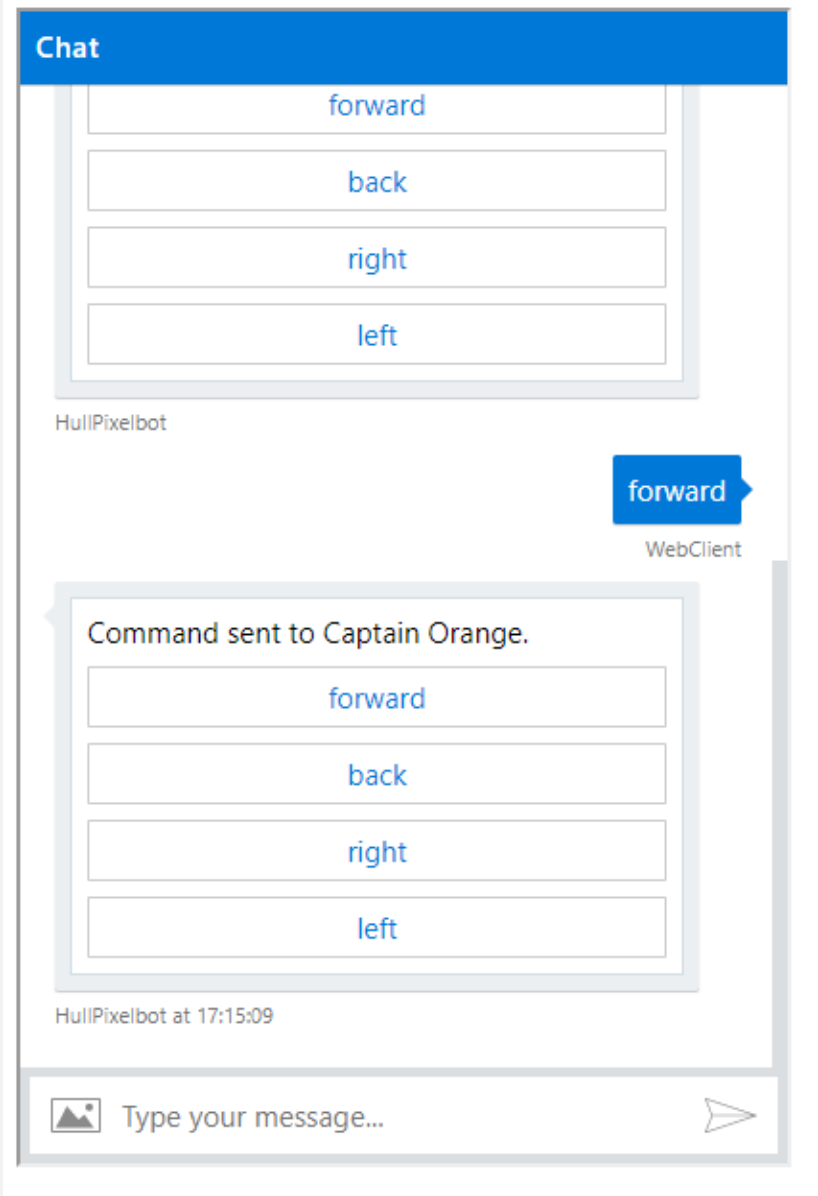
Connect to the robot quiz

Go to:

`hpb.host`

Select the quiz

The first three people will be allocated a robot and can control it via the chatbot



Adding Artificial Intelligence

Chatbots are all very well, but so far we've not seen any AI

There are two ways I'm going to use AI with our chatbots

The first one is to use a Microsoft QnA maker to create a bot that can handle simple queries

The second is to take a look at natural language understanding in bots

Microsoft QnA Generator

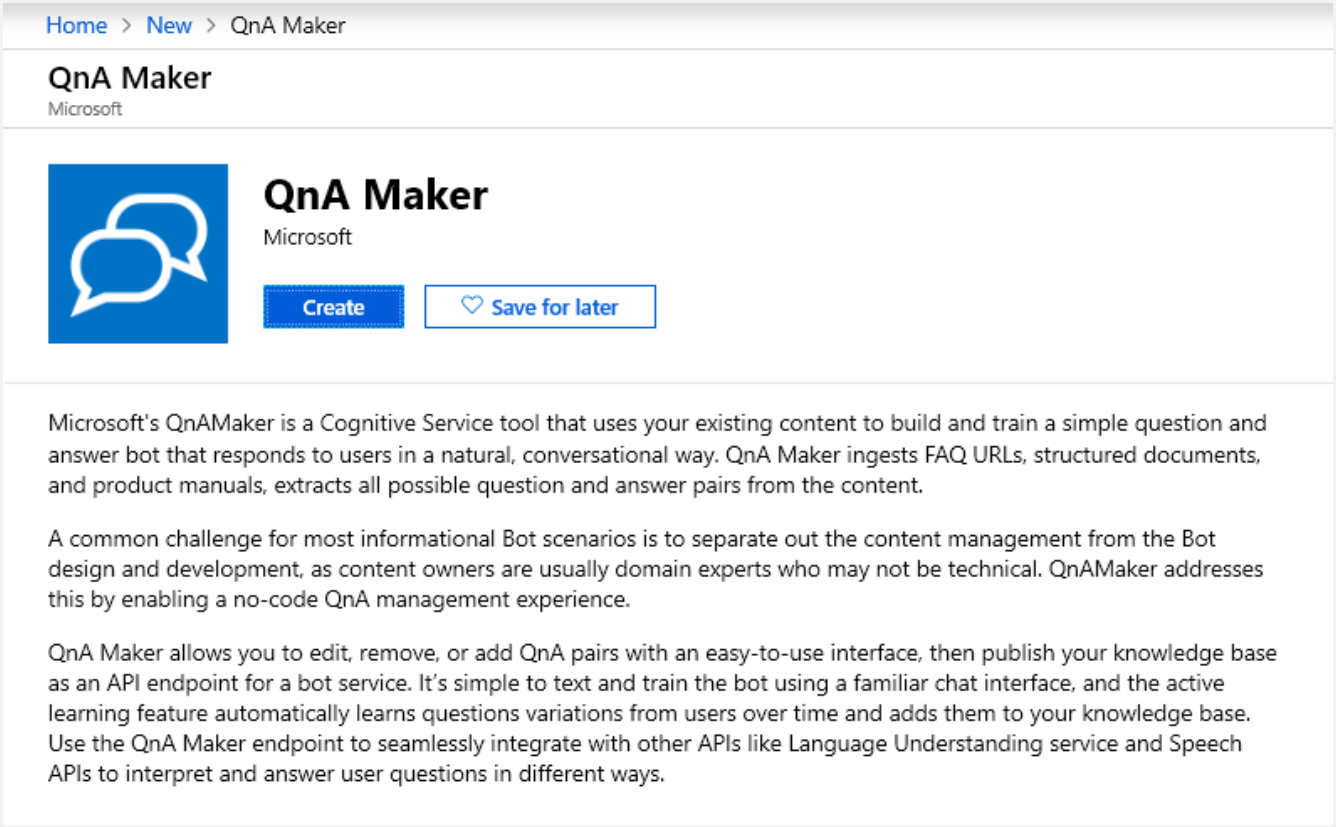
If you want to present your documents as an intelligent ‘bot’ you can do this using the Microsoft QnA generator

It will work on web sites, manuals, documents or brochures

Creating a QnA Maker

The QnA maker is a tool that runs in the cloud and processes the input to produce the data for the bot to use

You have to provision one of these on your Azure host

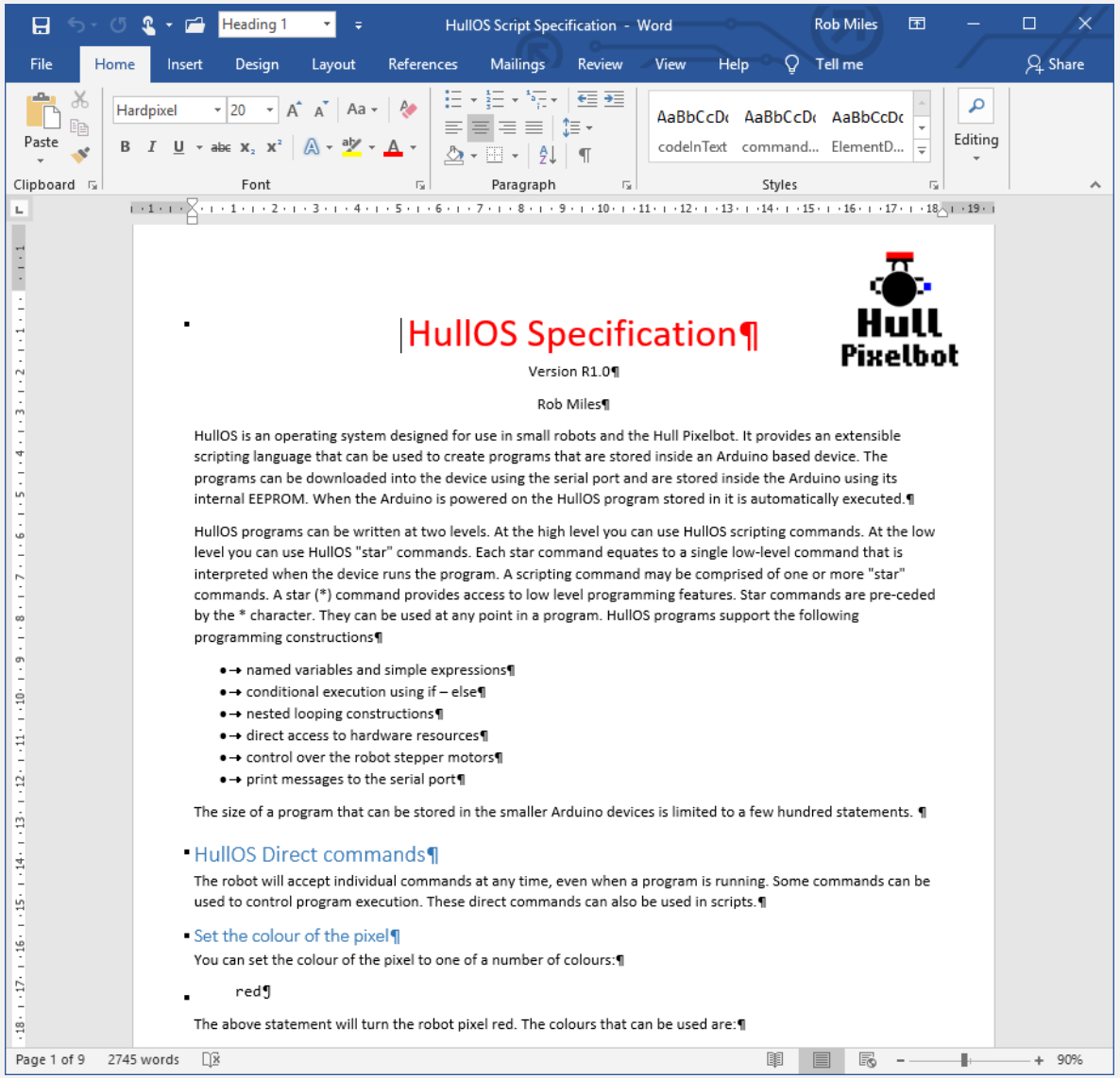


What I did

I took a word document that describes the scripting commands for my Hull Pixelbot robots

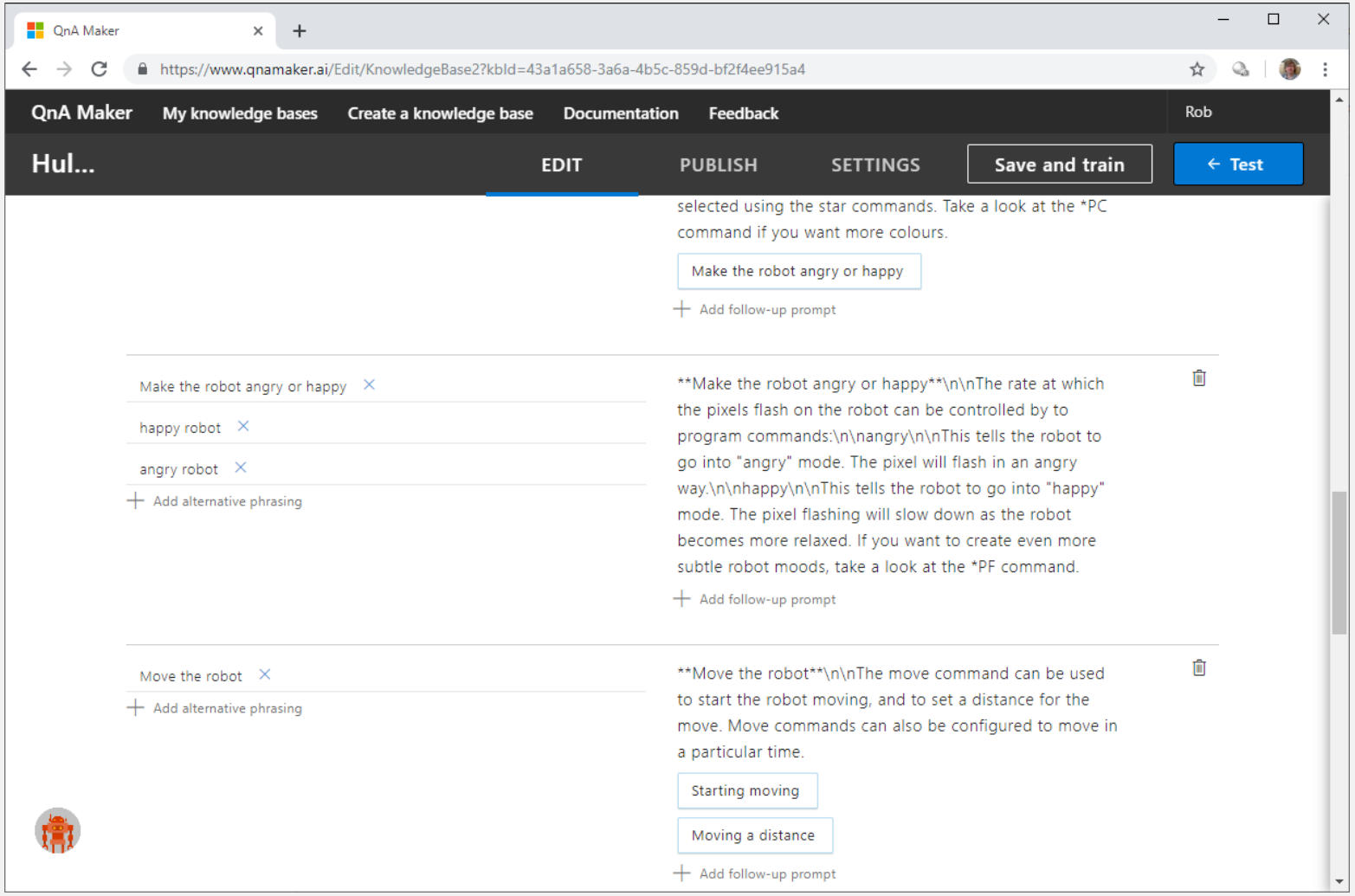
I fed this raw document into the QnA generator to see what it did

I used the QnA generator to automatically make a web chatbot that exposes the information in the document



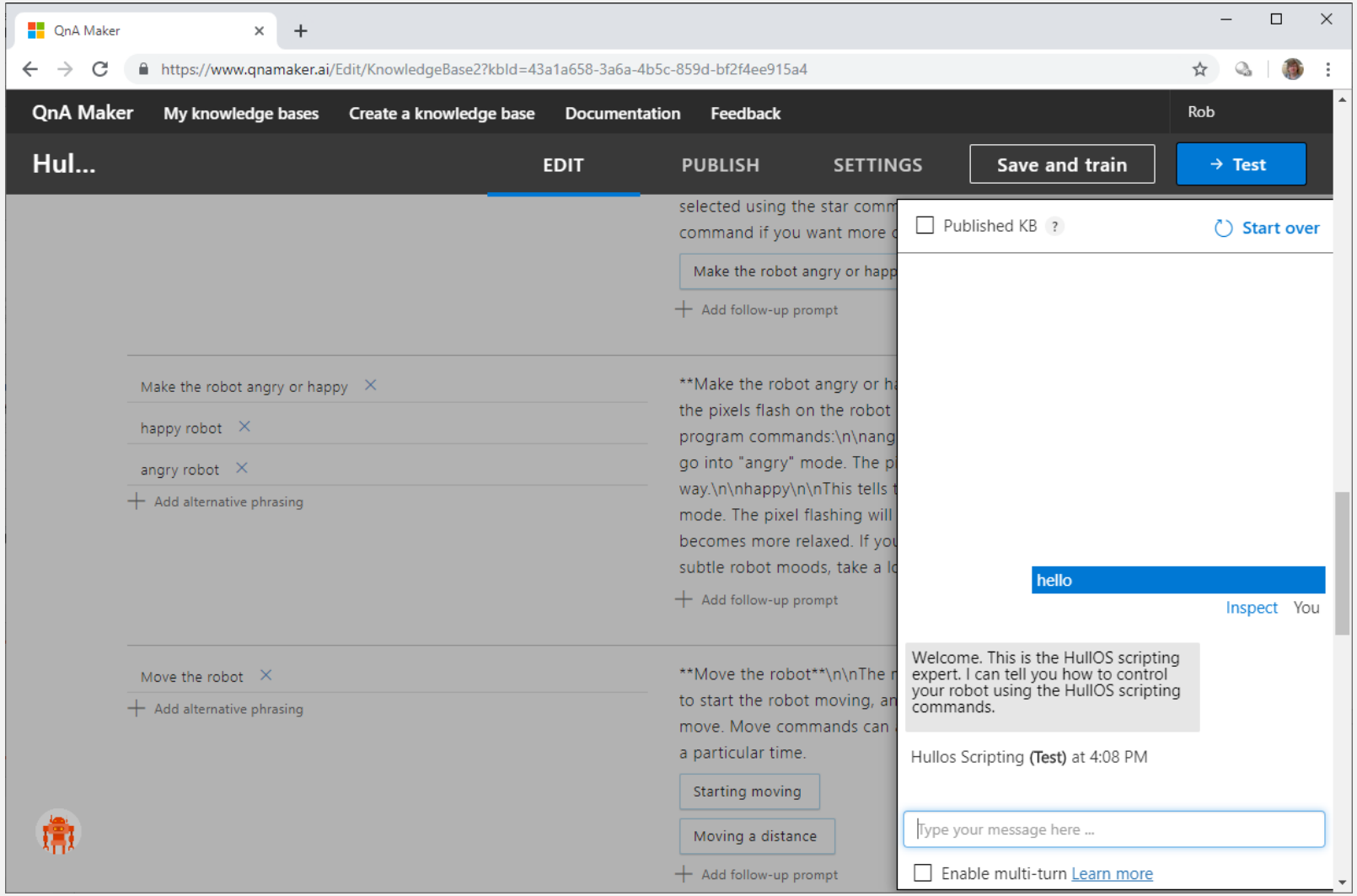
QnA maker

We can use the web editor to modify the questions and answers that have been extracted from the data



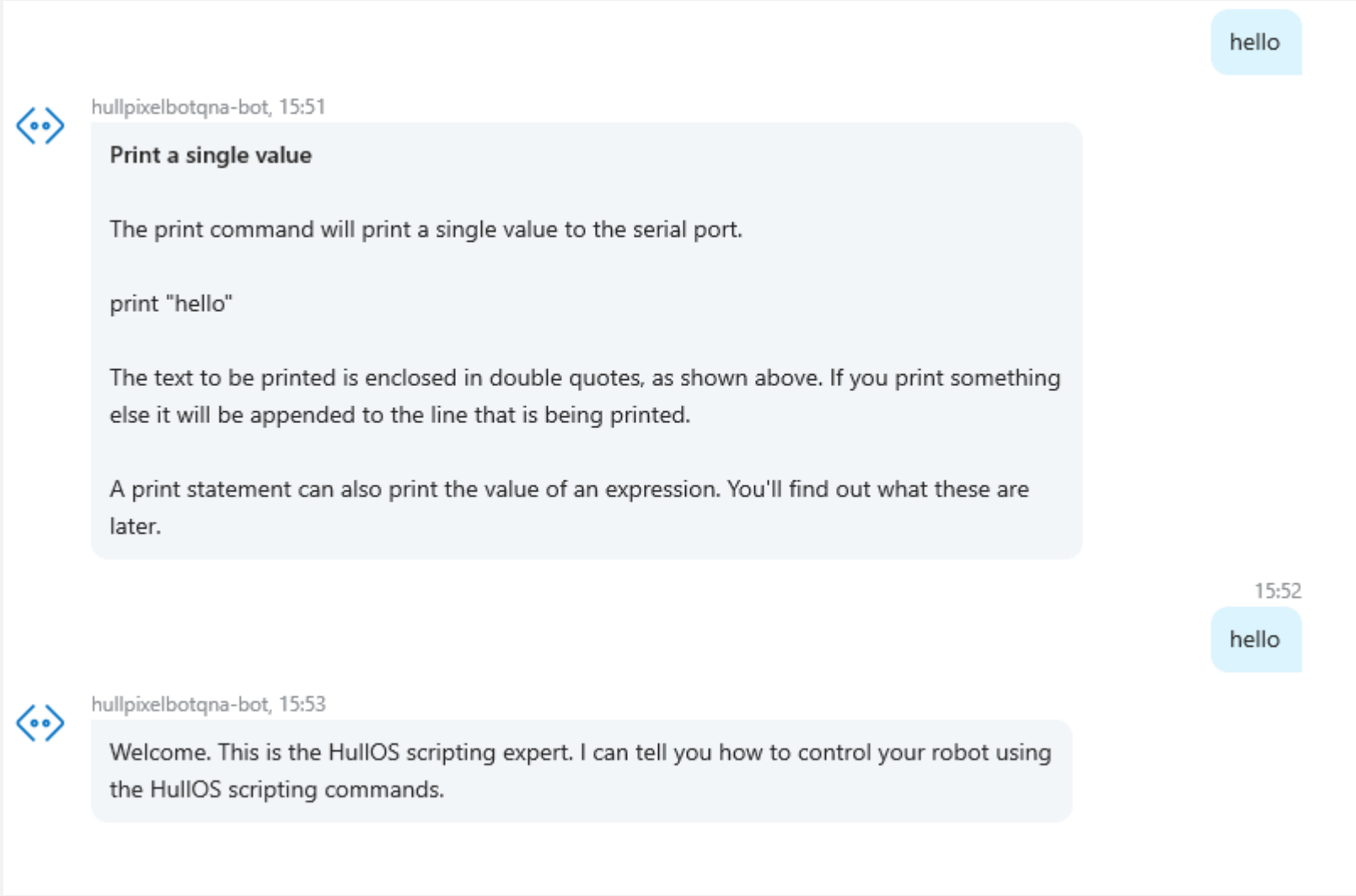
QnA Maker

We can also test the interactions with the data set



Editing

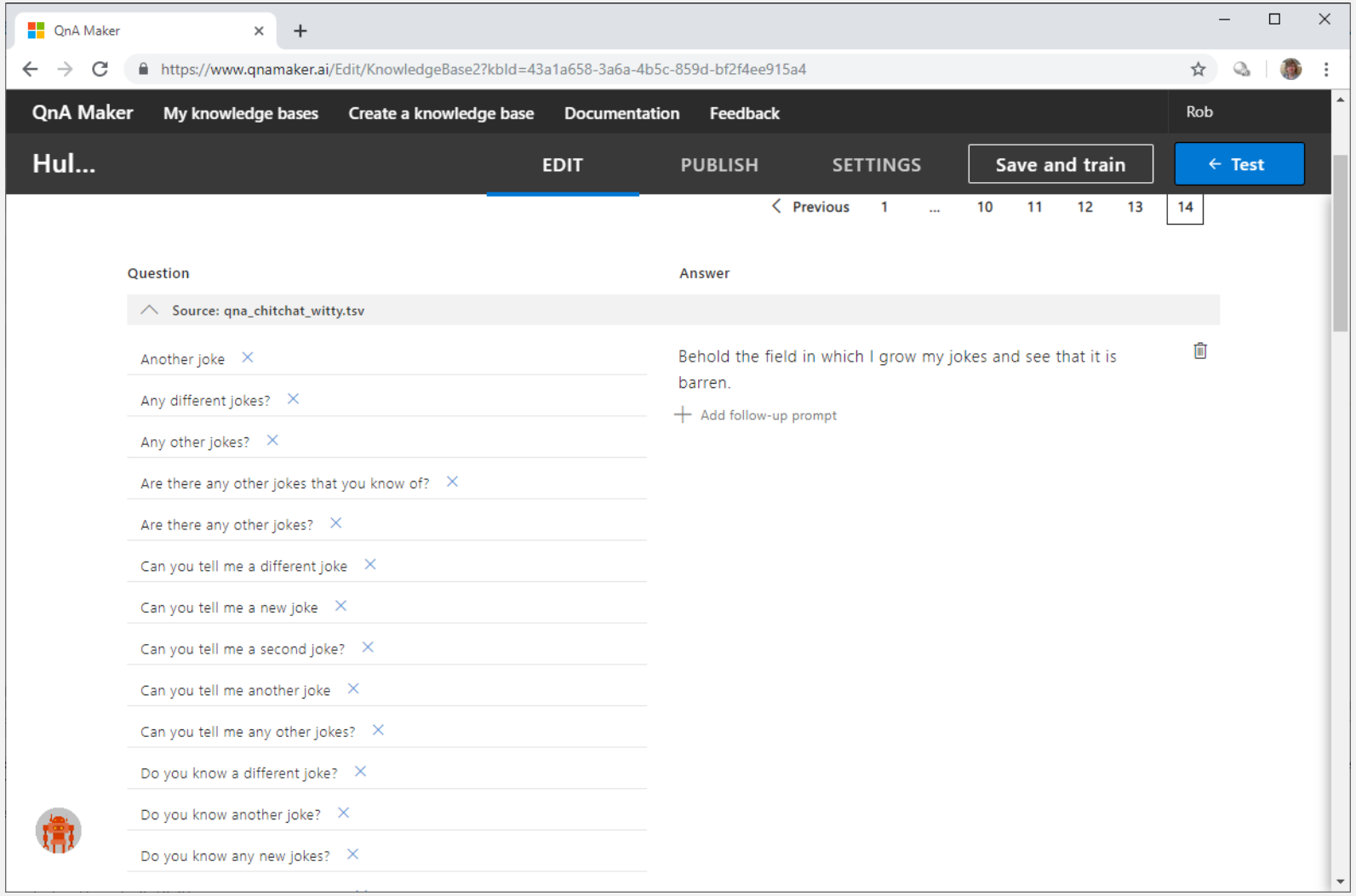
I've added some extra questions and answers to make some responses make more sense



Adding Personality

There are some “chit chat” personalities that you can add to your bot

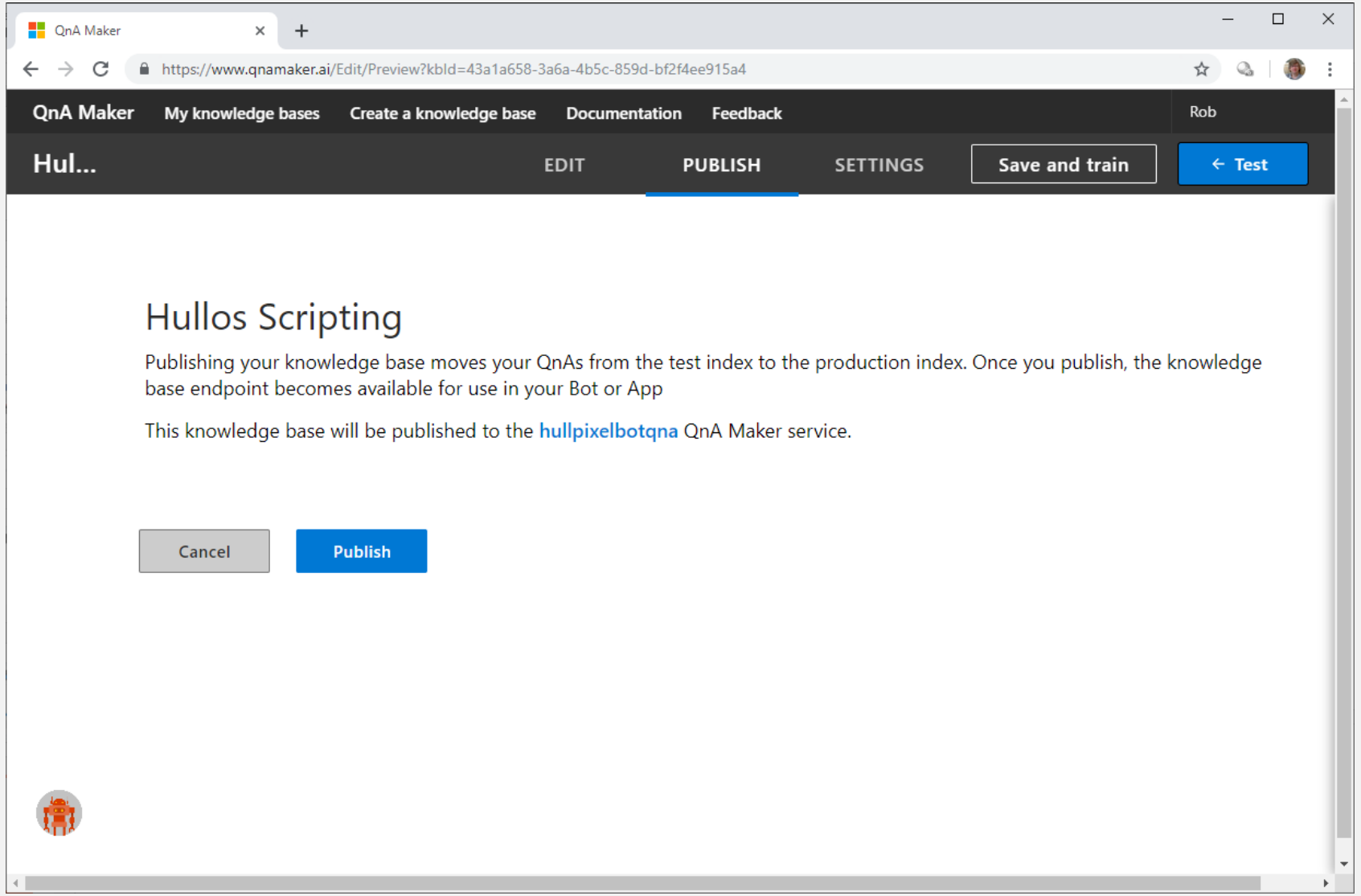
These are supplied as question sets that you can add



Publishing the data set

When we are happy with the content we can publish the data to create a service

We can then create a bot that serves out this knowledge



Demo

Talking to my document with Skype

Using language recognition in your apps

```
var response = await qnaMaker.GetAnswersAsync(turnContext);
if (response != null && response.Length > 0)
{
    await turnContext.SendActivityAsync(MessageFactory.Text(response[0].Answer + " " +
                                        response[0].Score), cancellationToken);
}
```

The pre-built bot displays the most likely response

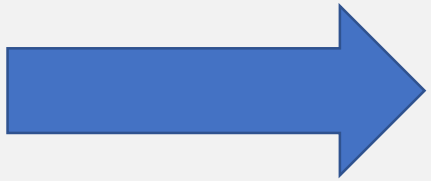
If we use the QnA service in our own application we can pull out a score value that will allow use to decide how much confidence the system has that it has a valid response

Demo

Talking to my document with C#

Adding more understanding

Book me a flight to Cairo



You can also use LUIS as a way of automating response to user queries

This requires a bit of training, mapping intent to input


```
{
   "query": "Book me a flight to Cairo",
   "topScoringIntent": {
       "intent": "BookFlight",
       "score": 0.9887482
   },
   "intents": [
       {
           "intent": "BookFlight",
           "score": 0.9887482
       }
],
   "entities": [
       {
           "entity": "cairo",
           "type": "Location“
           "score": 0.956781447
       }
   ]
}
```

Summary

A chatbot takes user input and generates responses

A response can take the form of an Adaptive Card that can contain buttons and images

The chatbot is deployed as a service that is resident in the cloud

A channel registration service connects the chatbot service with the channels it is to use

Channels can include web chat, Skype, email, Slack

The QnA service can convert input into questions and answers that can be deployed via a chatbot

The QnA service can be made “chatty” using “chit chat” resources

The QnA service can also be used in a regular chatbot

The LUIS framework allows user intent to be determined from input

www.robmiles.com