



---

# ALEXA JAMES FRANCIS

---

email: [james.francis@mckesson.com](mailto:james.francis@mckesson.com)



FEBRUARY 10, 2018

AZURE DEEP AZURE  
Dr. Zoran B. Djordjević

## Table of Contents

Problem Statement .....	3
Data Description .....	3
Hardware Description .....	4
Software Description .....	4
Installation and Configuration .....	5
PharmAid Configuration .....	5
PharmApi Configuration .....	7
Alexa Skills Configuration .....	9
PharmAid Code .....	13
Web.config .....	13
AlexaConstants.cs .....	15
AlexaSpeechletAsync.cs .....	16
PharmStatus.cs .....	19
AlexaUtils.cs .....	22
WebApiConfig.cs .....	25
AlexaImagesController.cs .....	26
Main Website Page (HTML) .....	27
default.htm .....	27
privacy.htm .....	28
terms.htm .....	28
PharmAid Packages Used .....	29
packages.config .....	29
PharmApi Code .....	30
PAFunc1.cs .....	30
Compile and Run Code .....	32
Compile/Publish PharmAid .....	32
Compile/Publish PharmApi .....	34
Run PharmAid and PharmApi Programs .....	36
PharmAid Results .....	36
PharmApi Results .....	37
Show PharmAid Alexa Skill Results .....	38
Summary .....	39
Lessons Learned .....	39
What Next? .....	40
References .....	41
Reference URLs .....	41

YouTube URLs, GitHub URLs .....	41
Separate GitHub URLs for cloning .....	41

# PharmAid: Alexa Assisted Pharmacy

## Problem Statement

Consumers are making a tectonic shift away from “Brick-and-Mortar” stores to online shopping using various technologies: Intelligent Personal Assistants, Web Apps, Phone Apps, etc. At the same time, these consumers are expecting quicker and easier ordering with faster delivery of their products; even expecting same-day delivery in some cases. Pharmacies have been lagging in this new frontier and are beginning to understand that it is no longer a question of if their industry will be transformed but it is a question of who will lead this transformation and when it will happen. Pharmacies are on the verge of being overhauled due to pharmaceutical demand, pricing pressure, consumer habits, and the rise of disruptive technologies such as A.I., Drones, Autonomous Vehicles, and Robotics and must adapt or become extinct.

In this project, we will leverage Alexa as an interactive assistant to a Pharmacy Management System hosted in Azure to automate the process of filling prescriptions for consumers.

## Data Description

AlexaIntentSchema: a JSON Object which defines the intents Alexa can use when “Ask PharmAid” is invoked

```
{
  "intents": [
    {
      "intent": "FillRxIntent"
    },
    {
      "intent": "FindRxIntent"
    },
    {
      "intent": "WhenRxIntent"
    },
    {
      "intent": "CallDoctorIntent"
    }
  ]
}
```

AlexaUtterances.txt: this is what you can say to Alexa when invoke PharmAid with a question

FillRxIntent to fill my prescription  
WhereRxIntent to find my prescription  
RxReadyIntent when will my prescription arrive  
CallDoctorIntent to call my doctor

## Hardware Description

### Dell Laptop

#### Manufacturer:

RSO Image Build: 1603A (03-01-2016) (MPS-WIN7PX64-MBI)

Deployment Completed: 3/31/2017 3:50:18PM

Deployment Time: 01:32:24 || Deployment Source: PXE

Model: Precision 7710

Rating: 7.2

Processor: Intel Core i7-6820HW CPU@2.70GHz 2.70GHz

Memory: 16.0 GB (15.7 GB Usable)

System type: 64-bit Operating System

Pen and Touch: No Pen or Touch Input is available for this Display

### App Service Plan

Location: East US

Pricing Tier: Standard

Instance Size: Small

Instance Count: 1

## Software Description

PharmAid is a Web Api .NET Application running on Azure as an App Service. It handles the responses sent from Alexa to Azure, interprets the intents, and sends the request to PharmApi for a status.

PharmApi is an Azure .NET Function which is just an api simulator for me. Ideally, this would be replaced with a real Pharmacy Application Api.

Here is the list of software that I used to create the PharmAid and PharmAPI applications.

Development IDE: Visual Studio 2017, C# and .NET Framework

#### Important PharmAid Packages installed via NuGet

- AlexaSkillsKit: .NET library to write Alexa skills (compatible with Amazon's AlexaSkillsKit for Java)
- BouncyCastleCrypto: for certificates and encryption (used by AlexaSkillsKit)
- Newtonsoft.json: .NET JSON Framework (used for creating, serializing, and parsing JSON objects)

#### PharmApi Packages installed via NuGet

- Newtonsoft.json: .NET JSON Framework (used for creating, serializing, and parsing JSON objects)

To build the Alexa Skill, I used the Alexa Skills Kit available on the the Alexa Developer Console.

<https://developer.amazon.com/edw/home.html#/skills>

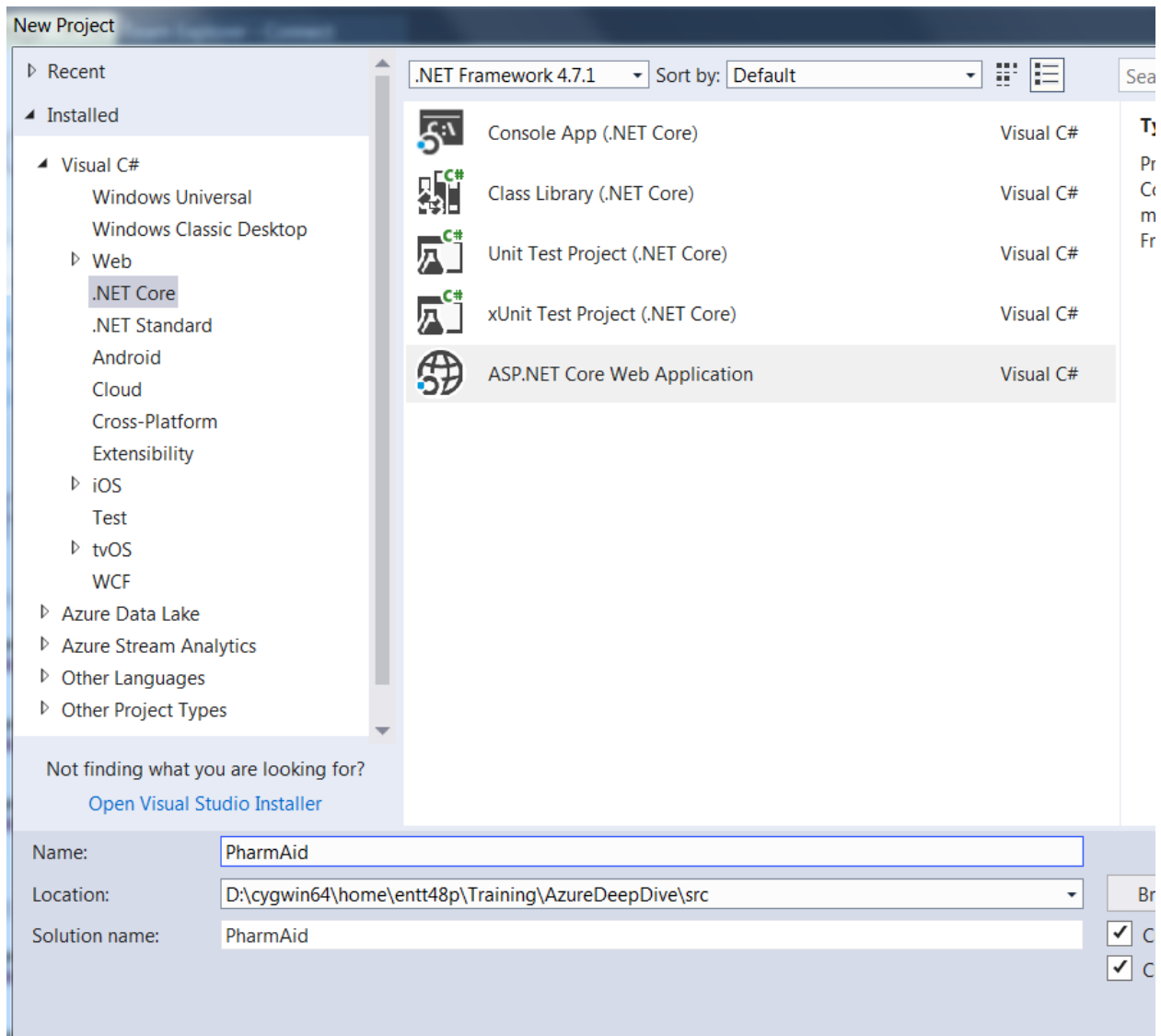
The following sample code was invaluable in helping me understand how Alexa works as well as how to use the outstanding AlexaSkillsKit library.

- <https://github.com/AreYouFreeBusy/AlexaSkillsKit.NET>
- <https://github.com/tamhinsf/Azure4Alexa>

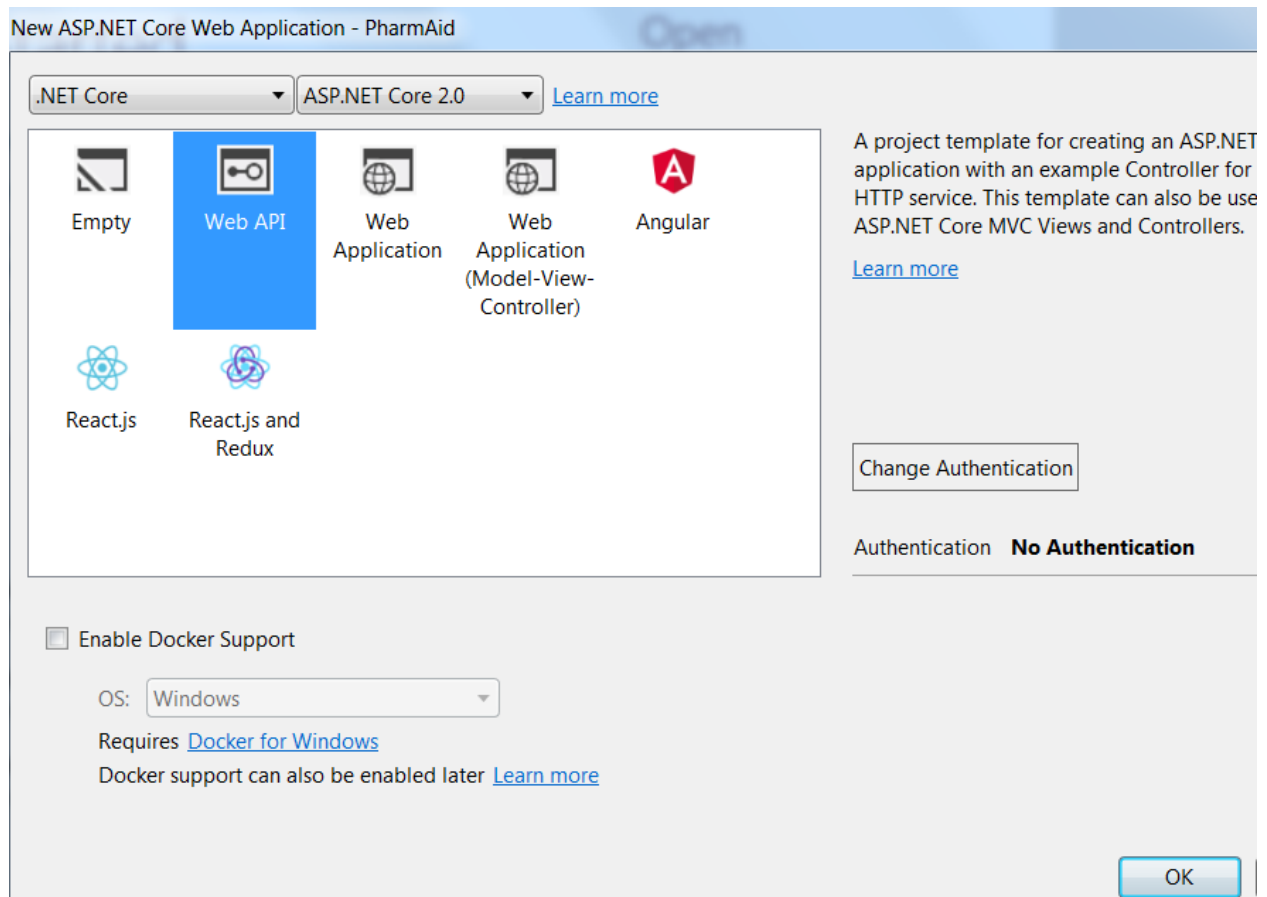
## Installation and Configuration

### PharmAid Configuration

Start by creating an ASP.NET Core Web Application project in Visual Studio called PharmAid.

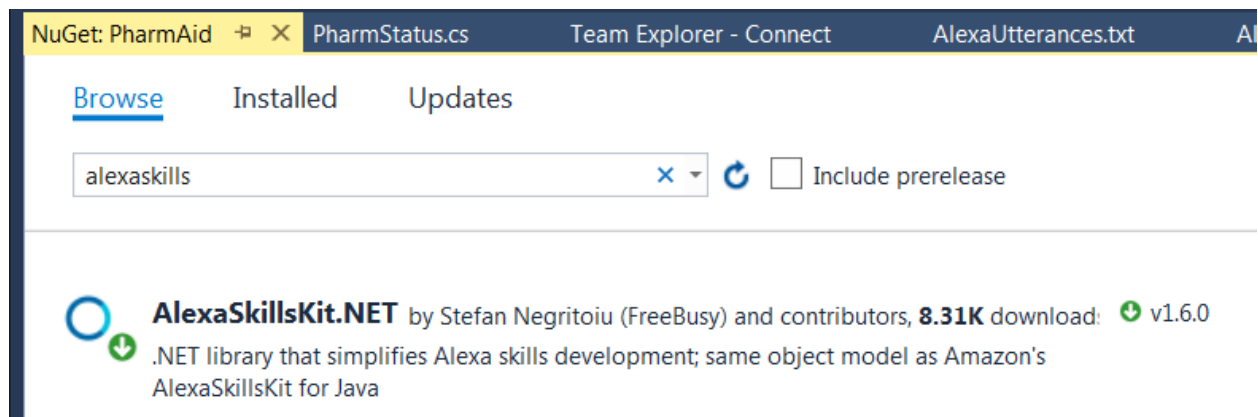


Select the Web API template which uses MVC and will be functioning as the API for our Alexa Skill.



There are a few packages you will need to install:

- The AlexaSkillsKit.NET packages that simplifies Alexa skills development.
- Newtonsoft.JSON to parse JSON elements.



NuGet: PharmAid X PharmStatus.cs Team Explorer - Connect AlexaUtterances.txt

[Browse](#)

Installed

Updates

newtonsoft.json



Include prerelease

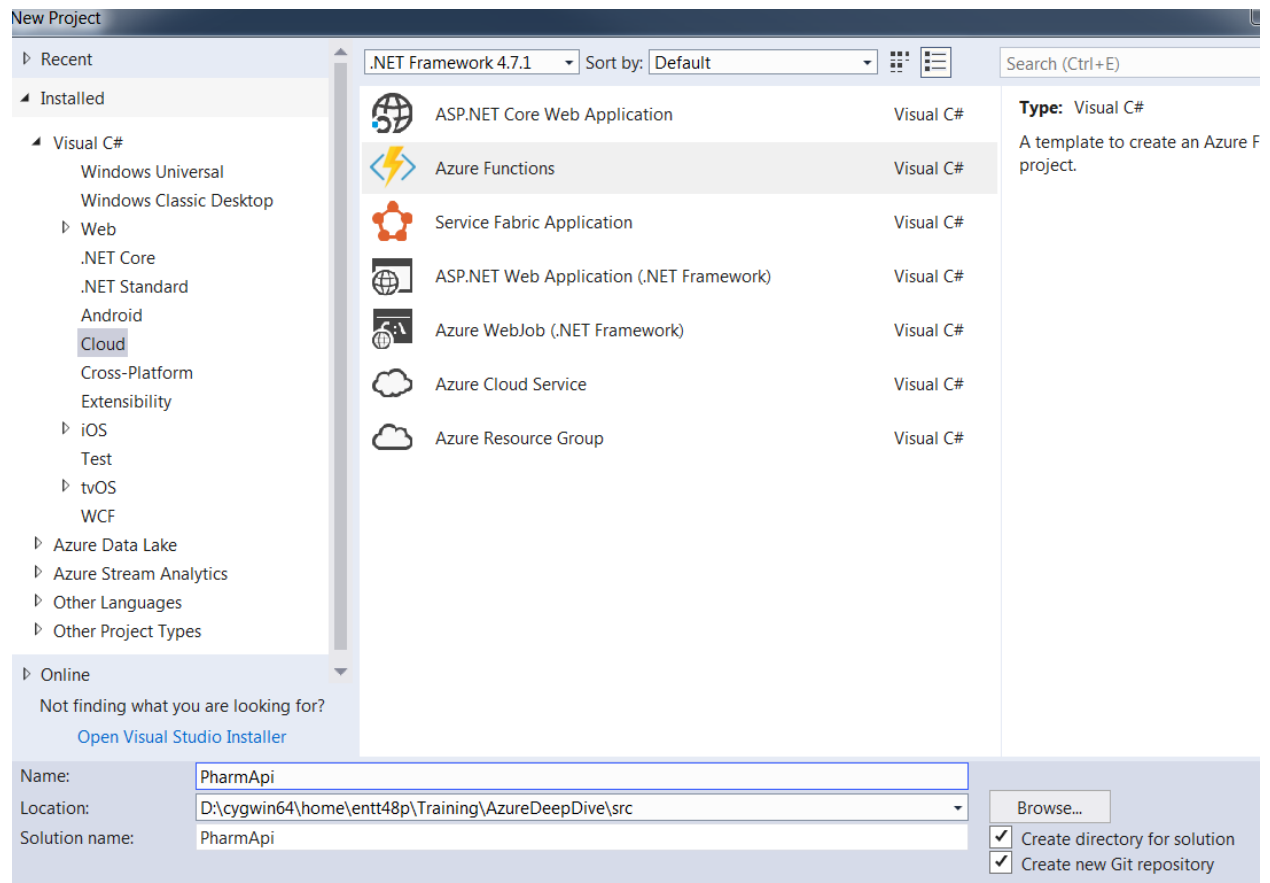
**Newtonsoft.Json** by James Newton-King, **102M** downloads

v10.0.3

Json.NET is a popular high-performance JSON framework for .NET

## PharmApi Configuration

Create a simple Azure Function in Visual Studio to respond to my PharmAid application. This is going to be used for testing but would be replaced with a fully functional Pharmacy System API.








Use the Http trigger template so PharmApi responds to Get requests from PharmAid.


### New Template - PharmApi

Azure Functions v1 (.NET Framework)

 Empty


 Http trigger

 Queue trigger

 Timer trigger

Storage Account (AzureWebJobsStorage)

Storage Emulator

 Some capabilities may require an Azure storage account.

Access rights

Anonymous

Creates an Azure function project with an Http trigger. Additional triggers can be added during development

[Get started with Azure Functions](#)

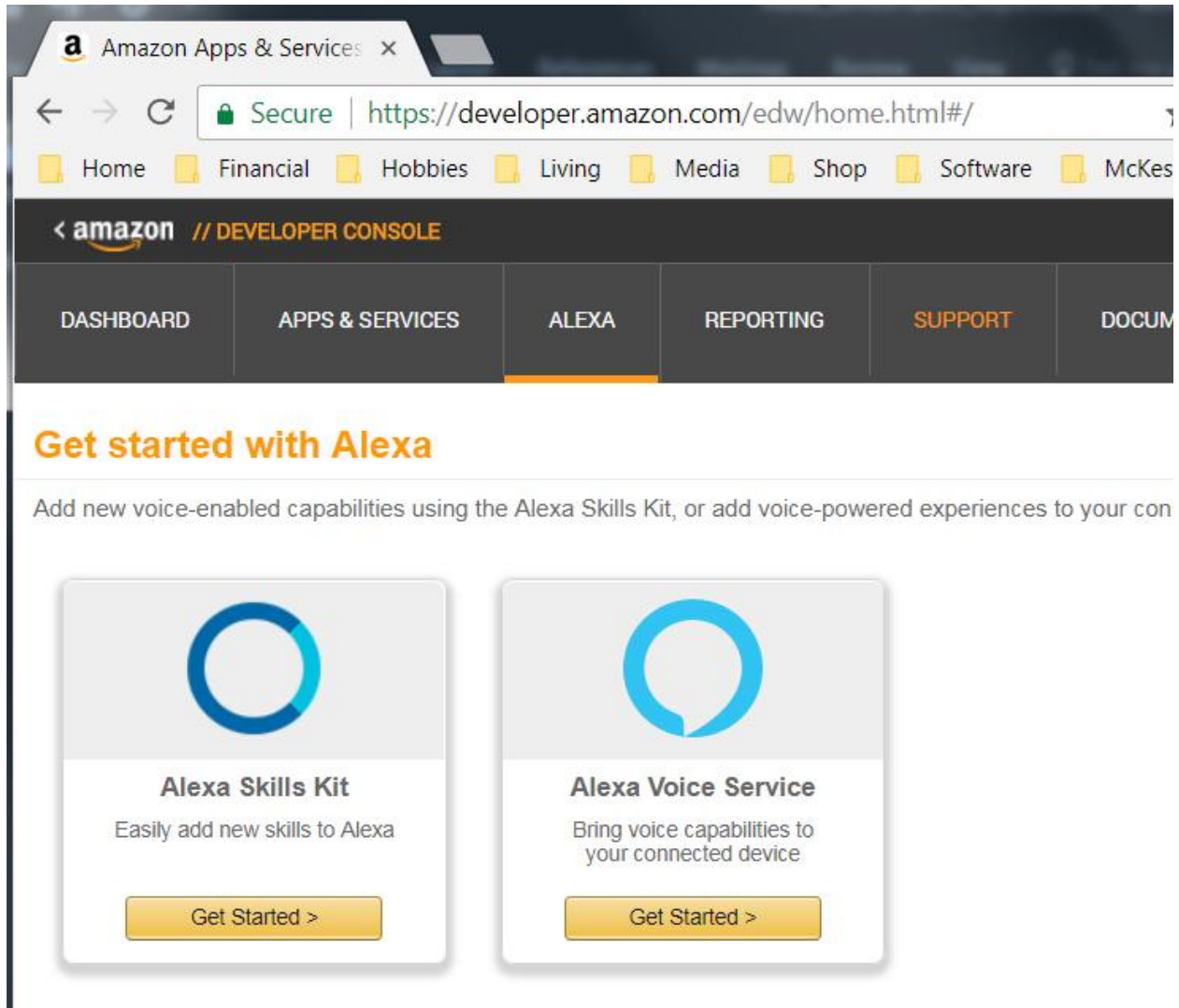
OK

Cancel

I did not need to install any additional packages for the PharmApi Function.

## Alexa Skills Configuration

To create an Alexa Skill, you must have an Amazon Developer Account which is free. After registering, you can login to the Alex Skills Developer Portal to create your skill. Click on Alexa Skills Kit to get started.



Enter your skill information. My skill will be invoked by "Ask PharmAid...".

English (U.S.) ✓	Add a New Language
<b>Skill Type</b> Define a custom interaction model or use one of the predefined skill APIs. <a href="#">Learn more</a> Custom	
<b>Language</b> Language of your skill	English (U.S.)
<b>Application Id</b> The ID for this skill	amzn1.ask.skill.6dfc7b73-5a7c-4755-9d54-598ad83c83b9
<b>Name</b> Name of the skill that is displayed to customers in the Alexa app. Must be between 2-50 characters.	PharmAid
<b>Invocation Name</b> The name customers use to activate the skill. For example, "Alexa ask Tide Pooler...".	pharmaid

Since we have our invocation name defined, create an interaction model which is a combination of an Intent Schema in JSON format and Utterances. Utterances are phrases you say after invoking your application. Example: Ask pharmaid to fill my prescription.

Slots are optional but simplify your utterances and how many utterances you have to define. I did not implement them for pharmaid as it is a prototype, but I would use them to define a list of drugs that could be filled at the remote central fill pharmacy.

### Intent Schema

The schema of user intents in JSON format. For more information, see [Intent Schema](#). Also see [built-in slots](#) and [built-in intents](#).

```
1 {  
2   "intents": [  
3     {  
4       "intent": "FillRxIntent"  
5     },  
6     {  
7       "intent": "FindRxIntent"  
8     },  
9     {  
10      "intent": "WhenRxIntent"  
11    }  
  ]  
}
```

### Sample Utterances

These are what people say to interact with your skill. Type or paste in all the way:

Up to 3 of these will be used as Example Phrases, which are hints to users.

1	FillRxIntent to fill my prescription
2	FindRxIntent to find my prescription
3	WhenRxIntent when will my prescription arrive
4	CallDoctorIntent to call my doctor

The next step is critical to connect to Azure. Select an HTTPS endpoint which is my Azure PharmAid Web API application, not AWS Lambda, and enter the Azure endpoint.

### Endpoint

Service Endpoint Type:

☐ AWS Lambda ARN (Amazon Resource Name) ⓘ ☒ HTTPS

*Recommended*

AWS Lambda is a server-less compute service that runs your code in response to events and automatically manages the underlying compute resources for you.

[More info about AWS Lambda](#)

[How to integrate AWS Lambda with Alexa](#)

Default

`https://pharmaid.azurewebsites.net/api/alex`

We move on to the SSL Certificate piece of configuring your skill. For Azure, pharmaid is a subdomain of azurewebsites.net which uses a wildcard certificate.

## Global Fields

These fields apply to all languages supported by the skill.

To protect your security and the security of end users, we require that you use a certificate while developing an Alexa skill. For more information, see [Registering and Managing Alexa Skills - About SSL Options](#).

### Certificate for DEFAULT Endpoint:

Please select one of the three methods below for the web service:

- ☐ My development endpoint has a certificate from a trusted certificate authority
- ☒ My development endpoint is a sub-domain of a domain that has a wildcard certificate from a certificate authority
- ☐ I will upload a self-signed certificate in X.509 format. [Learn how to create a self signed certificate](#).

We are done with the configuration for the Alexa skill.

## PharmAid Code

### Web.config

This is where you define you Alexa Skill you build in the Alexa Developer Console. When you build a skill, Amazon assigns a unique Skill ID which is placed here. You also should also update the AppName as it is displayed in the Alexa Companion App. This is used by

```
<?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>
    <add key="AppId" value="amzn1.ask.skill.6dfc7b73-5a7c-4755-9d54-598ad83c83b9" />
    <add key="AppName" value="PharmAid" />
  </appSettings>
  <system.web>
    <compilation debug="true" targetFramework="4.6.1" />
    <httpRuntime targetFramework="4.6.1" />
  </system.web>
  <system.webServer>
    <handlers>
      <remove name="ExtensionlessUrlHandler-Integrated-4.0" />
      <remove name="OPTIONSVerbHandler" />
      <remove name="TRACEVerbHandler" />
      <add name="ExtensionlessUrlHandler-Integrated-4.0" path="*" verb="*"
type="System.Web.Handlers.TransferRequestHandler" preCondition="integratedMode,runtimeVersionv4.0" />
    </handlers>
  </system.webServer>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="System.Web.Helpers" publicKeyToken="31bf3856ad364e35" />
        <bindingRedirect oldVersion="1.0.0.0-3.0.0.0" newVersion="3.0.0.0" />
      </dependentAssembly>
      <dependentAssembly>
        <assemblyIdentity name="System.Web.Mvc" publicKeyToken="31bf3856ad364e35" />
        <bindingRedirect oldVersion="1.0.0.0-5.2.3.0" newVersion="5.2.3.0" />
      </dependentAssembly>
      <dependentAssembly>
        <assemblyIdentity name="System.Web.WebPages" publicKeyToken="31bf3856ad364e35" />
        <bindingRedirect oldVersion="1.0.0.0-3.0.0.0" newVersion="3.0.0.0" />
      </dependentAssembly>
      <dependentAssembly>
        <assemblyIdentity name="Newtonsoft.Json" publicKeyToken="30ad4fe6b2a6aeed" culture="neutral" />
        <bindingRedirect oldVersion="0.0.0.0-10.0.0.0" newVersion="10.0.0.0" />
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
</configuration>
```

```
<system.codedom>
  <compilers>
    <compiler language="c#;cs;csharp" extension=".cs"
type="Microsoft.CodeDom.Providers.DotNetCompilerPlatform.CSharpCodeProvider,
Microsoft.CodeDom.Providers.DotNetCompilerPlatform, Version=1.0.8.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" warningLevel="4" compilerOptions="/langversion:default
/nowarn:1659;1699;1701" />
    <compiler language="vb;vbs;visualbasic;vbscript" extension=".vb"
type="Microsoft.CodeDom.Providers.DotNetCompilerPlatform.VBCodeProvider,
Microsoft.CodeDom.Providers.DotNetCompilerPlatform, Version=1.0.8.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" warningLevel="4" compilerOptions="/langversion:default /nowarn:41008
/define:_MYTYPE=\&quot;Web\&quot; /optionInfer+" />
  </compilers>
</system.codedom>
</configuration>
```

## AlexaConstants.cs

This is used by the web configuration manager and retrieve the values from Web.config.

```
using System.Web.Configuration;
```

```
namespace PharmAid.Alexa
```

```
{
```

```
    public class AlexaConstants
```

```
    {
```

```
        // Inbound requests from Amazon include the Voice Skills AppId, assigned to you when registering your skill
```

```
        // Validate the value against what you registered, to ensure that someone else isn't calling your service.
```

```
        public static string AppId = WebConfigurationManager.AppSettings["AppId"];
```

```
        // The value of AppName has no correspondence to what you have registered in Amazon
```

```
        // we just store it here because it's useful. It does appear on the card that is shown to the user
```

```
        // in the Alexa Companion App
```

```
        public static string AppName = WebConfigurationManager.AppSettings["AppName"];
```

```
        // standard error message
```

```
        public static string AppErrorMessage = "Sorry, something went wrong. Please try again.";
```

```
    }
```

```
}
```



## AlexaSpeechletAsync.cs

This is one of the most important classes. This processes the requests from Alexa which is passed into the PharmAid Application. I implemented the interface supplied and specified by intents (the Alexa utterances) which I have highlighted below.

```
using System;
using AlexaSkillsKit.Speechlet;
using AlexaSkillsKit.Slu;
using AlexaSkillsKit.Authentication;
using AlexaSkillsKit.Json;
using System.Threading.Tasks;
using System.Net.Http;

namespace PharmAid.Alexa
{
    // Follow the AlexaSkillsKit documentation and override the base class and children with our own
    // implementation
    // the functions below map to the Alexa requests described at this URL
    // https://developer.amazon.com/public/solutions/alexa/alexa-skills-kit/docs/handling-requests-sent-by-alex
    public class AlexaSpeechletAsync : SpeechletAsync
    {
        // Alexa provides a security wrapper around requests sent to your service, which the
        // AlexaSkillsKit nuget package validates by default. However, you might not want this wrapper enabled while
        // you do local development and testing - in DEBUG mode.

        // Note: the default Azure publishing option in Visual Studio is Release (not Debug), so by default the
        // security wrapper will be enabled when you publish to Azure.

        // Amazon requires that your skill validate requests sent to it for certification, so you shouldn't
        // deploy to production with validation disabled

        // #if DEBUG

        public override bool OnRequestValidation(SpeechletRequestValidationResult result, DateTime
        referenceTimeUtc, SpeechletRequestEnvelope requestEnvelope)
        {
            return true;
        }

        // #endif

        // Invoked when the user begins their session
        // according to the docs we can load user data here
        public override Task OnSessionStartedAsync(SessionStartedRequest sessionStartedRequest, Session session)
        {
            // this function is invoked when a user begins a session with your skill
            // this is a chance to load user data at the start of a session

            // Respond with an error message if an invalid Alexa Application Id is passed
        }
    }
}
```

Alexa

```

    if (AlexaUtils.IsRequestInvalid(session))
    {
        return Task.FromResult<SpeechletResponse>(InvalidApplicationId(session));
    }

    // return some sort of Task per function definition
    return Task.Delay(0);
}

// Invoked when a user ends their session
// according to the docs we can save user data here
public override Task OnSessionEndedAsync(SessionEndedRequest sessionEndedRequest, Session session)
{
    // Respond with an error message is an invalid Alexa Application Id is passed
    if (AlexaUtils.IsRequestInvalid(session))
    {
        return Task.FromResult<SpeechletResponse>(InvalidApplicationId(session));
    }

    // return some sort of Task per function definition
    return Task.Delay(0);
}

// No intent was passed from Alexa
public override async Task<SpeechletResponse> OnLaunchAsync(LaunchRequest launchRequest, Session
session)
{
    // Respond with an error message is an invalid Alexa Application Id is passed
    if (AlexaUtils.IsRequestInvalid(session))
    {
        return await Task.FromResult<SpeechletResponse>(InvalidApplicationId(session));
    }

    // No intent was passed
    return await Task.FromResult<SpeechletResponse>(GetOnLaunchAsyncResult(session));
}

// We have a valid intent passed to us from Alexa
public override async Task<SpeechletResponse> OnIntentAsync(IntentRequest intentRequest, Session session)
{
    // Respond with an error message is an invalid Alexa Application Id is passed
    if (AlexaUtils.IsRequestInvalid(session))
    {
        return await Task.FromResult<SpeechletResponse>(InvalidApplicationId(session));
    }

    // intentRequest.Intent.Name contains the name of the intent
    // intentRequest.Intent.Slots.* contains slot values if you're using them
    // session.User.AccessToken contains the OAuth 2.0 access token if the user has linked to your auth system

    // Get intent from the request object
    Intent intent = intentRequest.Intent;

```

```

string intentName = (intent != null) ? intent.Name : null;

// Create an http client which can be reused across requests
var httpClient = new HttpClient();

// Since we have a valid intent, we can process the request in our PharmStatus class
switch (intentName)
{
    case ("FillRxIntent"):
        return await RX.PharmStatus.FillRxIntent(session, httpClient);
    case ("FindRxIntent"):
        return await RX.PharmStatus.FindRxIntent(session, httpClient);
    case ("WhenRxIntent"):
        return await RX.PharmStatus.WhenRxIntent(session, httpClient);
    case ("CallDoctorIntent"):
        return await RX.PharmStatus.CallDoctorIntent(session, httpClient);
    default:
        return await Task.FromResult<SpeechletResponse>(GetOnLaunchAsyncResult(session));
}

}

// Called by OnLaunchAsync - when the skill is invoked without an intent.
// Called by OnIntentAsync - when an intent is not mapped to action.
private SpeechletResponse GetOnLaunchAsyncResult(Session session)
{
    return AlexaUtils.BuildSpeechletResponse(new AlexaUtils.SimpleIntentResponse() { cardText = "Not sure what you are asking" }, true);
}

// Called when the request does not include an Alexa Skills Appld.
private SpeechletResponse InvalidApplicationId(Session session)
{
    return AlexaUtils.BuildSpeechletResponse(new AlexaUtils.SimpleIntentResponse()
    {
        cardText = "An invalid Application ID was received from Alexa."
    }, true);
}
}
}

```

## PharmStatus.cs

PharmStatus is another very important class. I specify my Azure Function URL(PharmAPI) which is simulating the Pharmacy Management System API. Inside this class I also implemented my intents. The intents are passed from my Speechlet and a call is made to the PharmAPI function. PharmApi returns a JSON object which is processed and the response is sent back to Alexa via the Speechlet.

```
using AlexaSkillsKit.Speechlet;
using PharmAid.Alexa;
using System;
using System.Net.Http;
using System.Threading.Tasks;
using Newtonsoft.Json.Linq;

namespace PharmAid.RX
{
    public class PharmStatus
    {
        // This is Azure Function which would simulate a restful call to a pharmacy system API.
        public static string RxUrl = "http://pharmapi.azurewebsites.net/api/PAFunc1";

        // This method is invoked when the user want to get his prescription filled.
        public static async Task<SpeechletResponse> FillRxIntent(Session session, HttpClient httpClient)
        {
            string httpResultString = "";

            httpClient.DefaultRequestHeaders.Clear();
            var httpResponseMessage = await httpClient.GetAsync(RxUrl+"?action=pharmFill");
            if (httpResponseMessage.IsSuccessStatusCode)
            {
                httpResultString = await httpResponseMessage.Content.ReadAsStringAsync();
            }
            else
            {
                httpResponseMessage.Dispose();
                return AlexaUtils.BuildSpeechletResponse(new AlexaUtils.SimpleIntentResponse() { cardText =
                AlexaConstants.AppErrorMessage }, true);
            }

            // prescription is hard-coded to Crestor
            var simpleIntentResponse = ParseResults(httpResultString, "Prescription for Crestor");
            httpResponseMessage.Dispose();
            return AlexaUtils.BuildSpeechletResponse(simpleIntentResponse, true);
        }

        // This method is invoked when the user wants to know where his prescription is.
        public static async Task<SpeechletResponse> FindRxIntent(Session session, HttpClient httpClient)
        {
            string httpResultString = "";
```

Alexa

```

httpClient.DefaultRequestHeaders.Clear();
var httpResponseMessage = await httpClient.GetAsync(RxUrl + "?action=pharmFind");
if (httpResponseMessage.IsSuccessStatusCode)
{
    httpResultString = await httpResponseMessage.Content.ReadAsStringAsync();
}
else
{
    httpResponseMessage.Dispose();
    return AlexaUtils.BuildSpeechletResponse(new AlexaUtils.SimpleIntentResponse() { cardText =
AlexaConstants.AppErrorMessage }, true);
}

// prescription is hard-coded to Crestor
var simpleIntentResponse = ParseResults(httpResultString, "Prescription for Crestor");
httpResponseMessage.Dispose();
return AlexaUtils.BuildSpeechletResponse(simpleIntentResponse, true);
}

// This method is invoked when the user wants to know where his prescription is.
public static async Task<SpeechletResponse> WhenRxIntent(Session session, HttpClient httpClient)
{
    string httpResultString = "";

    httpClient.DefaultRequestHeaders.Clear();
    var httpResponseMessage = await httpClient.GetAsync(RxUrl + "?action=pharmWhen");
    if (httpResponseMessage.IsSuccessStatusCode)
    {
        httpResultString = await httpResponseMessage.Content.ReadAsStringAsync();
    }
    else
    {
        httpResponseMessage.Dispose();
        return AlexaUtils.BuildSpeechletResponse(new AlexaUtils.SimpleIntentResponse() { cardText =
AlexaConstants.AppErrorMessage }, true);
    }

    // prescription is hard-coded to Crestor
    var simpleIntentResponse = ParseResults(httpResultString, "Prescription for Crestor");
    httpResponseMessage.Dispose();
    return AlexaUtils.BuildSpeechletResponse(simpleIntentResponse, true);
}

// This method is invoked when the user wants to know where his prescription is.
public static async Task<SpeechletResponse> CallDoctorIntent(Session session, HttpClient httpClient)
{
    string httpResultString = "";

    httpClient.DefaultRequestHeaders.Clear();
    var httpResponseMessage = await httpClient.GetAsync(RxUrl + "?action=pharmCall");
    if (httpResponseMessage.IsSuccessStatusCode)

```

Alexa

```

    {
        httpResultString = await httpResponseMessage.Content.ReadAsStringAsync();
    }
    else
    {
        httpResponseMessage.Dispose();
        return AlexaUtils.BuildSpeechletResponse(new AlexaUtils.SimpleIntentResponse() { cardText =
AlexaConstants.AppErrorMessage }, true);
    }

    var simpleIntentResponse = ParseResults(httpResultString, "Doctor");
    httpResponseMessage.Dispose();
    return AlexaUtils.BuildSpeechletResponse(simpleIntentResponse, true);
}

// This method parses the json object returned from the Azure function.
private static AlexaUtils.SimpleIntentResponse ParseResults(string resultString, string rx)
{
    string stringToRead = String.Empty;
    string stringForCard = String.Empty;

    // you'll need to use JToken instead of JObject with results
    dynamic resultObject = JToken.Parse(resultString);

    // if you're into structured data objects, use JArray
    // JArray resultObject2 = JArray.Parse(resultString);
    stringToRead = resultObject.message;

    // Build the response
    if (stringForCard == String.Empty && stringToRead == String.Empty)
    {
        string noRx = "Unable to find a valid prescription";
        stringToRead += Alexa.AlexaUtils.AddSpeakTagsAndClean(noRx);
        stringForCard = noRx;
    }
    else
    {
        stringForCard = rx + " " + stringToRead;
        stringToRead = Alexa.AlexaUtils.AddSpeakTagsAndClean(rx + " " + stringToRead);
    }

    return new AlexaUtils.SimpleIntentResponse() { cardText = stringForCard, ssmlString = stringToRead };
}
}
}

```

## AlexaUtils.cs

This is a utility class which adds speech tags to the responses. It also cleans up any text which needs to be displayed on the Alexa Companion App.

```
using AlexaSkillsKit.Speechlet;
using AlexaSkillsKit.UI;

namespace PharmAid.Alexa
{
    public class AlexaUtils
    {
        // In a debug environment, you might find inbound request validation to be a nuisance, especially if you're
        // manually generating requests using cURL, Postman or another utility.
        // The #if #endif directives disable validation in DEBUG builds.

        public static bool IsRequestInvalid(Session session)
        {
            /*
            #if DEBUG
                return false;
            #endif
            in production, you'd probably want to do the check as follows:
            if (session.Application.Id != AlexaConstants.AppId)
            In development, you might have a single instance of this service and multiple different Alexa skills pointing
            at it. Structuring the AppId check in this manner lets you have a single string (AlexaConstants.AppId)
            containing all your
            valid AppIds. Lazy, but it works.
            */

            if (!AlexaConstants.AppId.Contains(session.Application.Id))
            {
                return true;
            }
            return false;
        }

        // a convenience class to pass the multiple components used to build
        // an Alexa response in one object

        public class SimpleIntentResponse
        {
            public string cardText { get; set; } = "";
            public string ssmlString { get; set; } = "";
            public string smallImage { get; set; } = "";
            public string largeImage { get; set; } = "";
        };

        public static string AddSpeakTagsAndClean(string spokenText)
        {
            // remove characters that will cause SSML to break.
            // probably a whole lot of other characters to remove or sanitize. This is just a lazy start.
        }
    }
}
```

```

        return "<speak> " + spokenText.Replace("&", "and") + " </speak>";
    }

    public static SpeechletResponse BuildSpeechletResponse(SimpleIntentResponse simpleIntentResponse, bool
shouldEndSession)
    {
        SpeechletResponse response = new SpeechletResponse();
        response.ShouldEndSession = shouldEndSession;

        // Create the speechlet response from SimpleIntentResponse.
        // If there's an ssmlString use that as the spoken reply
        // If ssmlString is empty, speak cardText

        if (simpleIntentResponse.ssmlString != "")
        {
            SsmlOutputSpeech speech = new SsmlOutputSpeech();
            speech.Ssml = simpleIntentResponse.ssmlString;
            response.OutputSpeech = speech;
        }
        else
        {
            PlainTextOutputSpeech speech = new PlainTextOutputSpeech();
            speech.Text = simpleIntentResponse.cardText;
            response.OutputSpeech = speech;
        }

        // if images are passed, then assume a standard card is wanted
        // images should be stored in the ~/Images/ folder and follow these requirements

        // JPEG or PNG supported, no larger than 2MB
        // 720x480 - small size recommendation
        // 1200x800 - large size recommendation

        if (simpleIntentResponse.smallImage != "" && simpleIntentResponse.largeImage != "")
        {
            StandardCard card = new StandardCard();
            card.Title = AlexaConstants.AppName;
            card.Text = simpleIntentResponse.cardText;

            // The trailing slash after the image name is required because we're serving off the image through a Web
            // API controller and
            // don't want to change the default web project settings

            card.Image = new Image()
            {
                LargeImageUrl = "https://" + System.Web.HttpContext.Current.Request.Url.Host + "/api/alexaimages/"
+ simpleIntentResponse.largeImage + "/",
                SmallImageUrl = "https://" + System.Web.HttpContext.Current.Request.Url.Host + "/api/alexaimages/"
+ simpleIntentResponse.smallImage + "/",
            };
        }
    }

```



```
        response.Card = card;

    }
    else
    {
        SimpleCard card = new SimpleCard();
        card.Title = AlexaConstants.AppName;
        card.Content = simpleIntentResponse.cardText;
        response.Card = card;
    }

    return response;
}
}
```

## WebApiConfig.cs

Configures the Web Api Routes for the controller.

```
using System.Web.Http;

namespace PharmAid
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            // Enable CORS

            config.EnableCors();

            // Web API configuration and services

            // Web API routes
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}
```

## AlexaController.cs

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

namespace PharmAid.Controllers
{
    public class AlexaController : ApiController
    {
        // you can set an explicit route if you want ...
        // [Route("alex/alex-session")]
        [HttpPost]
        public async Task<HttpResponseMessage> AlexaSession()
        {
            var alexaSpeechletAsync = new Alexa.AlexaSpeechletAsync();
            return await alexaSpeechletAsync.GetResponseAsync(Request);
        }
    }
}
```

## AlexaImagesController.cs

Displays images which are used on cards which are primarily used on the Alexa Companion apps and Echo Show devices.

```
using System;
using System.IO;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Web.Hosting;
using System.Web.Http;
using System.Web.Http.Cors;

namespace PharmAid.Controllers
{
    // Per Alexa requirements, all images need to be served off of a CORS-enabled server
    // and the only image types supported are PNG and JPG

    // This controller restricts CORS to the Alexa server and serves local images
    // stored in the ~/Images/ folder. Then, the Alexa/AlexaUtils.cs BuildSpeechletResponse class constructs
    // the full URL to the image for Alexa's consumption

    [EnableCors(origins: "http://ask-ifr-download.s3.amazonaws.com", headers: "*", methods: "get")]
    public class AlexaImagesController : ApiController
    {
        [HttpGet]
        public HttpResponseMessage AlexaImagesSession(string id)
        {
            var imageType = String.Empty;

            if (id.EndsWith(".png"))
            {
                imageType = "image/png";
            }
            else if (id.EndsWith(".jpg") || id.EndsWith(".jpeg"))
            {
                imageType = "image/jpg";
            }
            else
            {
                return null;
            }

            // probably don't need to try catch here, but do it here just
            // in case a missing file causes a problem
            try
            {
                var imageData = File.ReadAllBytes(HostingEnvironment.MapPath("~/Images/") + id);
                var result = new HttpResponseMessage(HttpStatusCode.OK);
                result.Content = new ByteArrayContent(imageData);
            }
        }
    }
}
```

```

        result.Content.Headers.ContentType = new MediaTypeHeaderValue(imageType);
        return result;
    }
    catch
    {
        return null;
    }
}
}
}

```

## Main Website Page (HTML)

These HTML pages can be used to verify the application is running. This page is also used if the application is invoked without going through Alexa. Normally, you won't see them.

### default.htm

```

<!DOCTYPE html>
<html>
<head>
    <title>PharmAid</title>
    <meta charset="utf-8" />
</head>
<body>
    <div style="font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif; margin-left:5%; margin-right:5%; margin-top:2%;">
        <h2>PharmAid</h2>

        <div>
            <br />
            <text>This application demonstrates how to connect Alexa to Azure to invoke a Pharmacy Application API</text>
            <br/>
            <br />
            <text>PharmAid uses the <a href="https://github.com/AreYouFreeBusy/AlexaSkillsKit.NET">AlexaSkillsKit.NET</a> package</text>
            <br />
        </div>
        <div>
            <br />
            <small><a href="terms.htm">Terms of Service</a></small>
            <br />
            <small><a href="privacy.htm">Privacy Policy</a></small>
            <br />
        </div>

        <hr />
        <footer>
            <p>Footer Placeholder</p>
        </footer>
    </div>
</body>

```

Alexa

&lt;/html&gt;

## privacy.htm

```
<!DOCTYPE html>
<html>
<head>
  <title>Privacy Policy</title>
  <meta charset="utf-8" />
</head>
<body>
  <div style="font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif; margin-left:5%; margin-right:5%; margin-top:2%;">
    <text>Alexa Skills Privacy Policy Placeholder</text>
  </div>
</body>
</html>
```

## terms.htm

```
<!DOCTYPE html>
<html>
<head>
  <title>Terms of Service</title>
  <meta charset="utf-8" />
</head>
<body>
  <div style="font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif; margin-left:5%; margin-right:5%; margin-top:2%;">
    <text>Alexa Skills Terms of Service Placeholder</text>
  </div>
</body>
</html>
```

## PharmAid Packages Used

Here is the list of packages used. The most important package is AlexaSkillsKit.NET without that being installed the application will not run.

### packages.config

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="AlexaSkillsKit.NET" version="1.6.0" targetFramework="net461" />
  <package id="BouncyCastle" version="1.8.1" targetFramework="net461" />
  <package id="Microsoft.AspNet.Cors" version="5.2.3" targetFramework="net461" />
  <package id="Microsoft.AspNet.WebApi" version="5.2.3" targetFramework="net461" />
  <package id="Microsoft.AspNet.WebApi.Client" version="5.2.3" targetFramework="net461" />
  <package id="Microsoft.AspNet.WebApi.Core" version="5.2.3" targetFramework="net461" />
  <package id="Microsoft.AspNet.WebApi.Cors" version="5.2.3" targetFramework="net461" />
  <package id="Microsoft.AspNet.WebApi.WebHost" version="5.2.3" targetFramework="net461" />
  <package id="Microsoft.CodeDom.Providers.DotNetCompilerPlatform" version="1.0.8"
targetFramework="net461" />
  <package id="Microsoft.Net.Compilers" version="2.6.1" targetFramework="net461"
developmentDependency="true" />
  <package id="Newtonsoft.Json" version="10.0.3" targetFramework="net461" />
</packages>
```

## PharmApi Code

This is the PharmApi Function. This function is really simple. It is just an HTTP trigger that takes a get request. This is just a sample API which would be replaced with a functioning Pharmacy System API such as EnterpriseRx. PhamAid calls it with the intents and the API returns a JSON object with the results.

### PAFunc1.cs

```
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Threading.Tasks;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Extensions.Http;
using Microsoft.Azure.WebJobs.Host;
using Newtonsoft.Json;
using System.Text;

namespace PharmApi
{
    public static class PAFunc1
    {
        [FunctionName("PAFunc1")]
        public static async Task<HttpResponseMessage> Run([HttpTrigger(AuthorizationLevel.Anonymous, "get",
"post", Route = null)]HttpRequestMessage req, TraceWriter log)
        {
            log.Info("C# HTTP trigger function processed a request.");

            // parse query parameter
            string action = req.GetQueryNameValuePairs()
                .FirstOrDefault(q => string.Compare(q.Key, "action", true) == 0)
                .Value;

            if (action == null)
            {
                // Get request body
                dynamic data = await req.Content.ReadAsAsync<object>();
                action = data?.action;
            }

            // all responses are canned due to lacking a real pharmacy api to talk to.
            switch (action)
            {
                // determines if a prescription can be filled.
                case "pharmFill":
                    return new HttpResponseMessage(HttpStatusCode.OK)
                    {
                        Content = new StringContent(JsonConvert.SerializeObject(new { message = "will be filled" }),
Encoding.UTF8, "application/json")
                    };
                // determines where a prescription is.
                case "pharmFind":
```

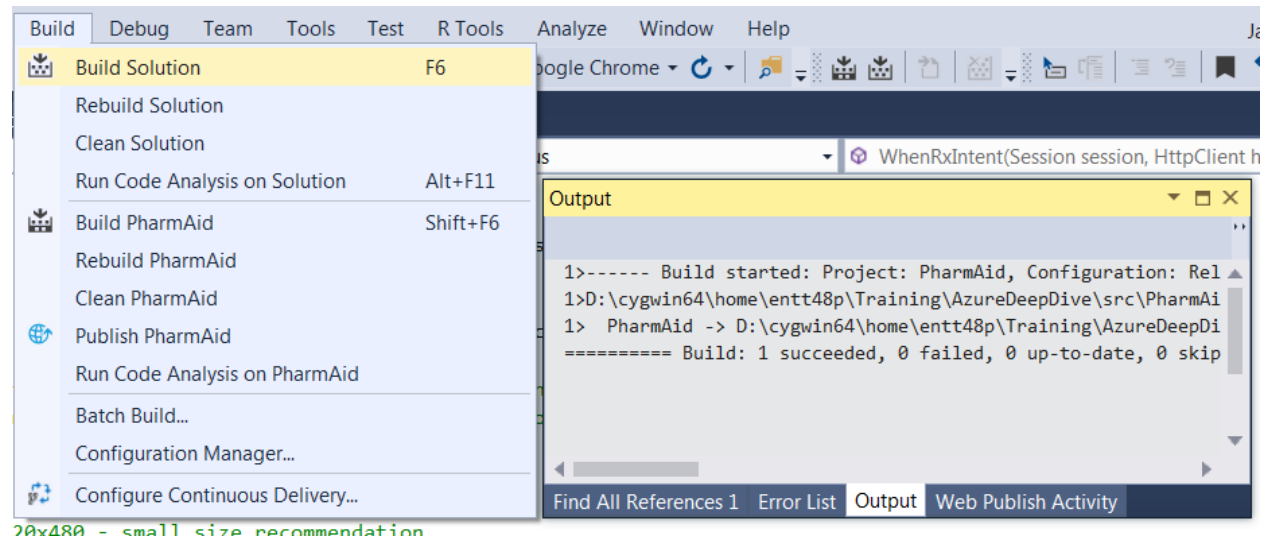
```
        return new HttpResponseMessage(HttpStatusCode.OK)
        {
            Content = new StringContent(JsonConvert.SerializeObject(new { message = "is delayed" })),
            Encoding.UTF8, "application/json")
        };
        // determines when a prescription will be delivered.
        case "pharmWhen":
            return new HttpResponseMessage(HttpStatusCode.OK)
            {
                Content = new StringContent(JsonConvert.SerializeObject(new { message = "by 4pm" })),
                Encoding.UTF8, "application/json")
            };
        // calls a doctor.
        case "pharmCall":
            return new HttpResponseMessage(HttpStatusCode.OK)
            {
                Content = new StringContent(JsonConvert.SerializeObject(new { message = "is called" })),
                Encoding.UTF8, "application/json")
            };
        default:
            return new HttpResponseMessage(HttpStatusCode.OK)
            {
                Content = new StringContent(JsonConvert.SerializeObject(new { message = "invalid call to pharmaapi"
            })), Encoding.UTF8, "application/json")
            };
        }
    }
}
```



## Compile and Run Code

For the Alexa Skill, there is no separate compilation step. The skill compiles as you build it through the portal. Earlier in this report I showed how to build the Alexa skill. Compiling and Publishing PharmAid and PharmApi is easy since they are Visual Studio C# .NET projects.

### Compile/Publish PharmAid



20x480 - small size recommendation

## Publish


Publish your app to Azure or another host. [Learn more](#)

 PharmAid - Web Deploy ▼

Publish

[Create new profile](#)

## Summary

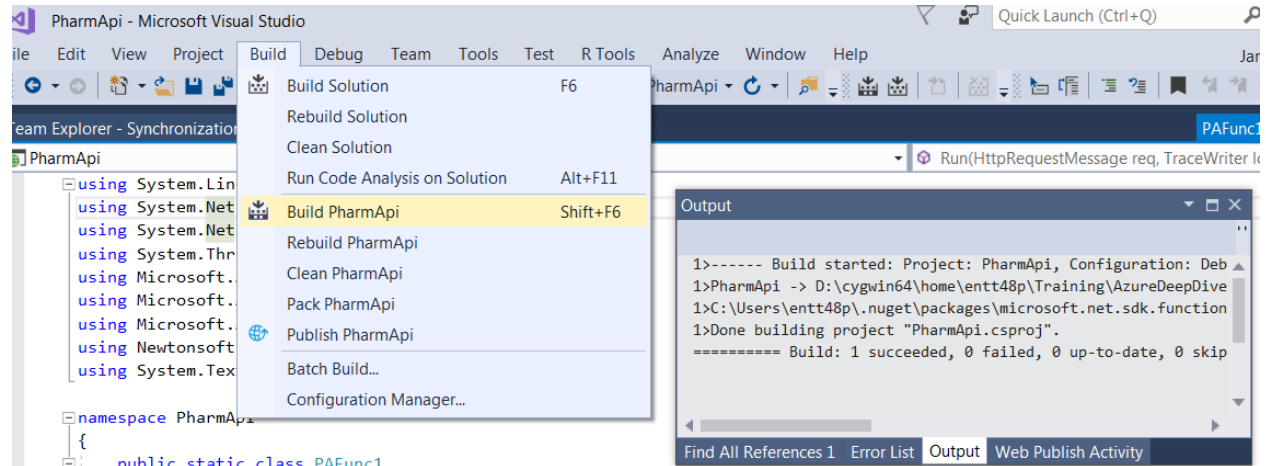
Site URL	<a href="http://pharmaid.azurewebsites.net">http://pharmaid.azurewebsites.net</a> 	<a href="#">Settings...</a>
Resource Group	rg-jfrancis	<a href="#">Preview...</a>
Configuration	Release	<a href="#">Rename profile...</a>
Username	\$PharmAid	<a href="#">Delete profile</a>
Password	*****	

Output

```
1>Adding ACLs for path (PharmAid)
1>Adding ACLs for path (PharmAid)
1>Publish Succeeded.
1>Web App was published successfully http://pharmaid.azureweb
===== Build: 0 succeeded, 0 failed, 1 up-to-date, 0 skip
===== Publish: 1 succeeded, 0 failed, 0 skipped =====
```

Find All References 1 | Error List ... | Output | Web Publish Activity

## Compile/Publish PharmApi



## Publish


Publish your app to Azure or another host. [Learn more](#)

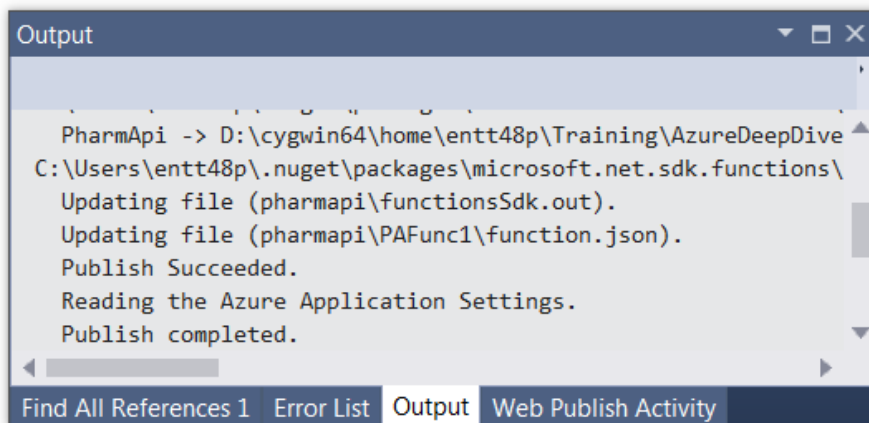
 pharmapi - Web Deploy ▼

Publish

[Create new profile](#)

## Summary

Site URL	<a href="http://pharmapi.azurewebsites.net">http://pharmapi.azurewebsites.net</a> 	<a href="#">Manage Application Settings...</a>
Configuration	Release	<a href="#">Manage Profile Settings...</a>
Delete existing files	False	<a href="#">Rename profile...</a>
Username	\$pharmapi	<a href="#">Delete profile</a>
Password	*****	

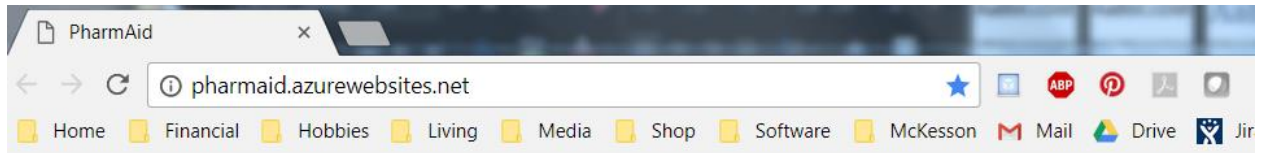


```
PharmApi -> D:\cygwin64\home\entt48p\Training\AzureDeepDive
C:\Users\entt48p\.nuget\packages\microsoft.net.sdk.functions\
Updating file (pharmapi\functionsSdk.out).
Updating file (pharmapi\PAFunc1\function.json).
Publish Succeeded.
Reading the Azure Application Settings.
Publish completed.
```

Find All References 1 Error List Output Web Publish Activity

## Run PharmAid and PharmApi Programs

### PharmAid Results



## PharmAid

This application demonstrates how to connect Alexa to Azure to invoke a Pharmacy Application API

PharmAid uses the [AlexaSkillsKit.NET](#) package

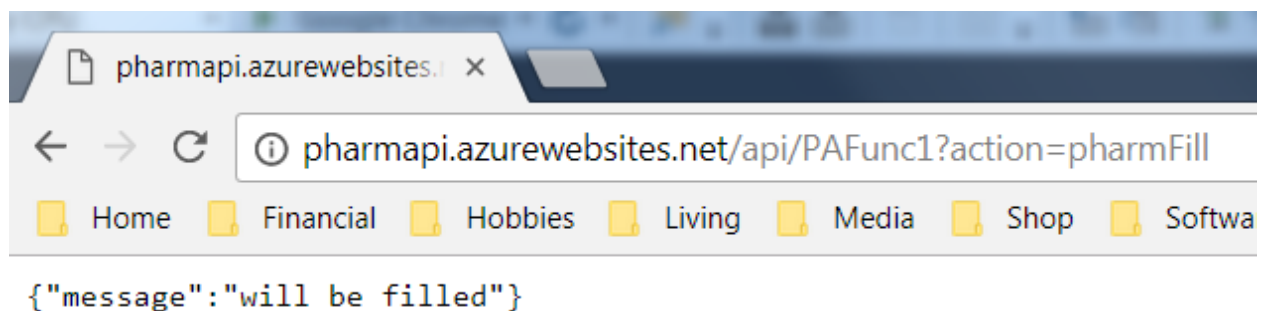
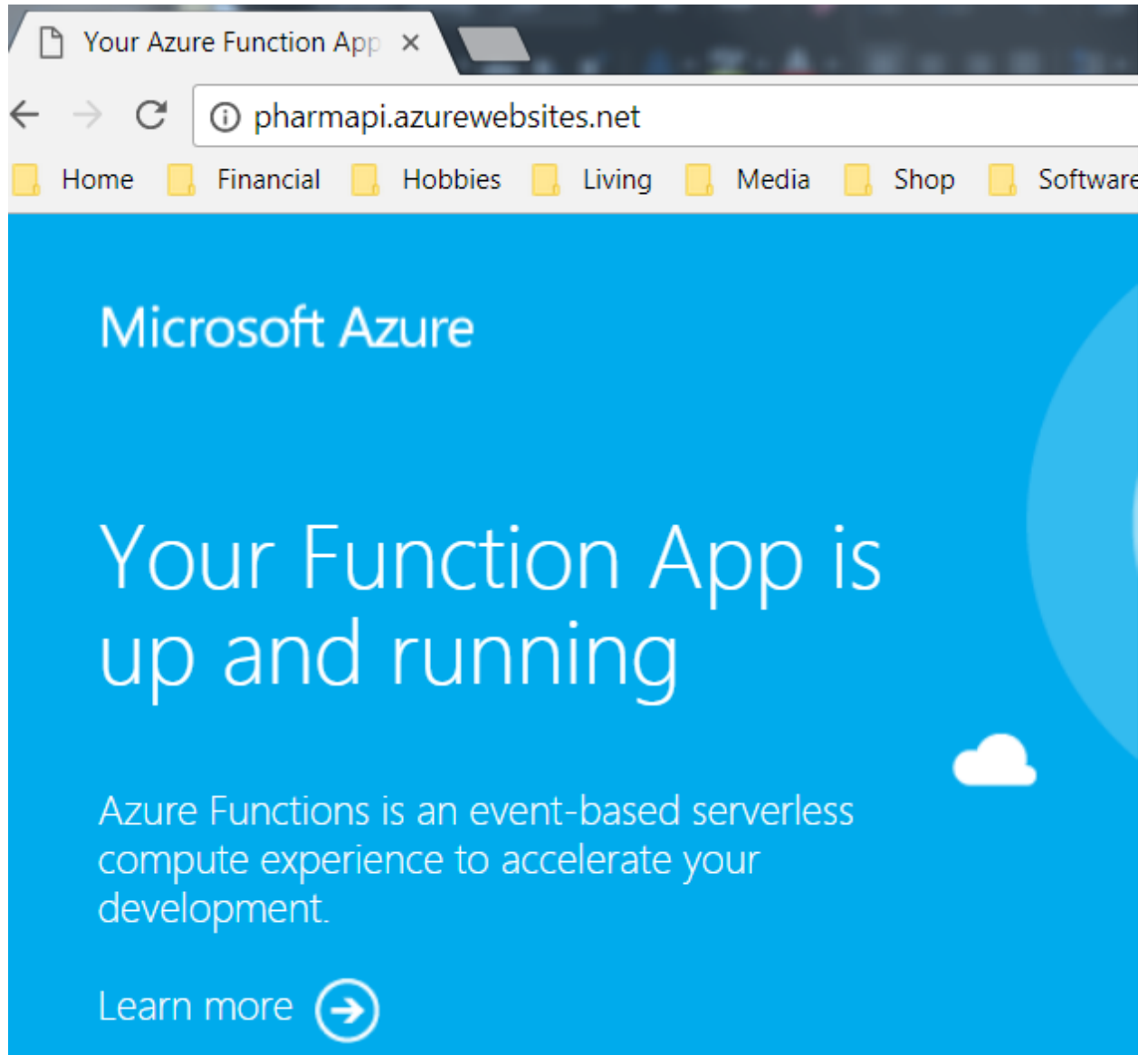
[Terms of Service](#)

[Privacy Policy](#)

---

Footer Placeholder

## PharmApi Results



## Show PharmAid Alexa Skill Results

The Alexa Skill Developer Portal gives you several ways to test your skill end-to-end.

The first way is to enter the Beta Test Simulator. Hold down the microphone and speak your invocation word and your utterance. The dark grey is my voice input and the Alexa response is spoken as well as displayed.

The screenshot shows the Alexa Skill Developer Portal Beta Test Simulator interface. At the top, there are three tabs: "Alexa Simulator", "Manual JSON", and "Voice & Tone". The "Alexa Simulator" tab is selected. Below the tabs, there is a language selector set to "English (US)" and a microphone icon with the text "Type or click and hold the mic". To the right, there are three checkboxes: "Skill I/O" (checked), "Echo Show Display" (checked), and "Device Log" (unchecked). The main area displays a conversation flow. The first utterance is "alexa ask pharmaid to fill my prescription", and the response is "Prescription for Crestor will be filled". The second utterance is "alexa ask pharmaid to find my prescription", and the response is "Prescription for Crestor is delayed". To the right of the conversation flow, there is a "Skill I/O" section showing the JSON input and output. The JSON input is a standard Alexa skill invocation request, and the JSON output is the skill's response.

The second way to test your skill is to use the Developer Console. Enter your invocation and utterance to display the response.

## Service Simulator

Use Service Simulator to test your HTTPS endpoint: <https://pharmaid.azurewebsites.net/api/alexa>

**Note:** Service Simulator does not currently support testing audio player directives, dialog model, customer permissions and customer account. Text mode does not support launch intents and single interaction phrases.

The screenshot shows the Service Simulator interface. At the top, there are two tabs: "Text" (selected) and "JSON". Below the tabs, there is a text input field labeled "Enter Utterance" with the text "call a doctor". Below the input field, there are two buttons: "Ask PharmAid" and "Reset". Below the buttons, there are two sections: "Service Request" and "Service Response". The "Service Request" section shows a JSON object with the following structure: 

```
{  "session": {    "new": true,    "sessionId": "SessionId.5ea8f08d-327c-4e11-bd...",    "application": {      "applicationId": "amzn1.ask.skill.6dfc7b73-..."    }  }}
```

 The "Service Response" section shows a JSON object with the following structure: 

```
{  "version": "1.0",  "response": {    "outputSpeech": {      "ssml": "<speak> Doctor is called </...",      "type": "SSML"    }  }}
```

## Summary

### Lessons Learned

#### Likes

- The AlexaSkillsKit.NET package was excellent. It made implementing the Alexa calls very easy.
- Visual Studio is simply great to program in when building Azure applications due to its tight integration. It also works seamlessly with github. It is simply a matter of a few mouse clicks to compile, test, and publish the application.
- The Alexa Skill portal was very nice. After becoming an Alexa developer, you login and create your skill in a matter of minutes. You give it an invocation word, a JSON schema, and your utterances. From there, you pass and https endpoint to Azure and you are done. The portal makes it easy to test your application. You can test it via verbal commands and listening to the response or by typing in your commands and seeing the response.
- Alexa has a concept of slots where you can define preset values. For example, you could use slots to hold the 10 most common medications to make it easier to order them.

#### Dislikes

- The verbal commands are tricky. Alexa is pretty good at understanding speech but it doesn't always get it right which can be frustrating.
- It would be nice if Azure was more tightly integrated with Amazon Alexa. The AlexaSkillsKit is nice but it feels like it should be easier to integrate your Alexa skill.

#### Benefits/Pros

- The entire application is serverless using Azure App Services so there is no infrastructure to maintain. The application should also scale rather easily.
- Alexa is already prevalent in a lot of households and is leading in market share so it would be useful for individuals that are comfortable with the technology.
- It shouldn't be too hard to extend Alexa to work with an existing Pharmacy Application Management System that has a decent API using this structure. You could easily implement as many intents as you wanted and retrieve valid information for the consumer.

#### Issues/Cons

- Security/PHI would be a big concern. We would have to be very careful about what data is being submitted to Alexa.
- Privacy is another concern. Users might not want to be shouting out prescriptions they are taking to an active listening device.
- I'm not sure how easy this would be to port to other voice assistants. I just don't know enough about them.
- There are a lot of medications and I don't think Alexa would get them right most of the time.



## What Next?

Alexa and other digital assistants are here to stay. The devices will become more and more prevalent. I also am pretty confident that the pharmacy industry will look dramatically different a few years from now than it does today.

In a few years, the millennial generation, whom are very comfortable with technology, will demand that their prescriptions are delivered to them. It is no longer a question of when this transformation will happen but it is simply a matter of when and whom will lead it.

We are at a cross-roads with technology. ...

- The cloud has proven an effective strategy to reduce and eliminate infrastructure costs.
- The cloud is evolving rapidly with technology such as machine learning, analytics, server-less functions, etc. which hosted applications could ever afford to manage or support.
- IOT (Internet of Things) has arrived and will connect everything. Medications will have RFID devices that track their distribution and ensure they are being taken.
- A.I. (Artificial Intelligence) has arrived. A.I. will replace a lot of pharmacy functions.
- Autonomous Vehicles/Drones are coming.

I could envision a world where I can simply ask Alexa to fill my prescription and a few hours later the medication is delivered via an autonomous vehicle or drone that I walk up to which verifies my identity via facial recognition and then opens a door to allow me to retrieve it.

Tomorrows pharmacy will look dramatically different than the pharmacy of today.

PharmAid was written as a proof-of-concept to see if I could integrate Alexa with Azure. The idea could be extended for pharmacy-to-doctor communications, pharmacy-to-pharmacy, pharmacy-to-distributor, and pharmacy to consumer.

I would like to see what would happen if Alexa was integrated with a real Pharmacy Management System. This would be interesting and could help lead pharmacies into the future.

## References

### Reference URLs

- <http://www.manchesterdeveloper.com/blog/post/Creating-Alexa-Skills-with-Web-API-and-hosting-on-Microsoft-Azure>
- <https://freebusy.io/blog/getting-started-with-alexa-app-development-for-amazon-echo-using-dot-net>
- <https://developer.amazon.com/docs/custom-skills/handle-requests-sent-by-alexa.html>
- <https://developer.amazon.com/docs/custom-skills/understanding-custom-skills.html>

### YouTube URLs, GitHub URLs

- 2 minute (short): <https://youtu.be/QwHZInw3xYE>
- 15 minutes (long): <https://youtu.be/KFFh9k4tmng>
- GitHub Repository with all artifacts:  
<https://github.com/james-francis/DeepAzure-FinalProject>

### Separate GitHub URLs for cloning

- PharmAid (includes Alexa JSON): <https://github.com/james-francis/PharmAid>
- PharmApi: <https://github.com/james-francis/PharmApi>