Microsoft Cognitive Services のカスタム画像分析による自動返答ボットの構築

概要

今日、AI (artificial intelligence: 人工知能) という言葉を聞かない日はないほど注目を浴びており、人間、またはそれ以上の知能 (の一部) をコンピューター上で実現することが可能になりました。

Microsoft Cognitive Services は 画像、文章、言語、情報を処理する機能を API 経由で利用できる “人工知能パーツ” とも言えるサービスです。ビックデータを投入、統計解析手法を用いて、人工知能として機能する分析モデルを作成する、という作業をご自身で行うことなしに、データを投入するだけで分析結果を取得できるようになりました。

このラボでは、まず Custom Vision Service を利用して、画像から会議室の空き状況を判別するカスタム画像分析モデルを作成します。次に、Web を経由して定期的に取得される画像が Azure Blob ストレージに送られるという状況を踏まえ、画像がアップロードされるたびに Custom Vision Service の API により、自動で画像を分析して結果を Azure テーブルストレージに保存する仕組みを Azure Logic App を利用して構築します。そして、ユーザーインターフェースとして Microsoft Bot Framework を利用したボット (チャットボット(Chatbot)) により、会議室の空き状況を自動で返答するボットを構築します。

目的

このハンズオン ラボでは、以下の方法について学習します。

- Cognitive Services Custom Vision Service を利用してカスタム画像分析モデルを作成する。
- Azure Blob ストレージへの画像アップロードをトリガーとして、その画像を Custom Vision Service の Prediction API を用いて分析し、取得したタグと確度を Azure Table

ストレージに保存するワークフローを、Azure Logic App で構築する。

- Azure Table ストレージのデータを元に自動で返答するボットを Microsoft Bot Framework で構築する

前提条件

このハンズオン ラボを実行するには、以下が必要です。

- マイクロソフトアカウント
- Azure サブスクリプション
- Visual Studio 2017 Community エディションまたはこれ以降

目次

このラボの推定所要時間: 45 分

演習 1: Cognitive Services Custom Vision Service でカスタム画像分析モデルを作成する

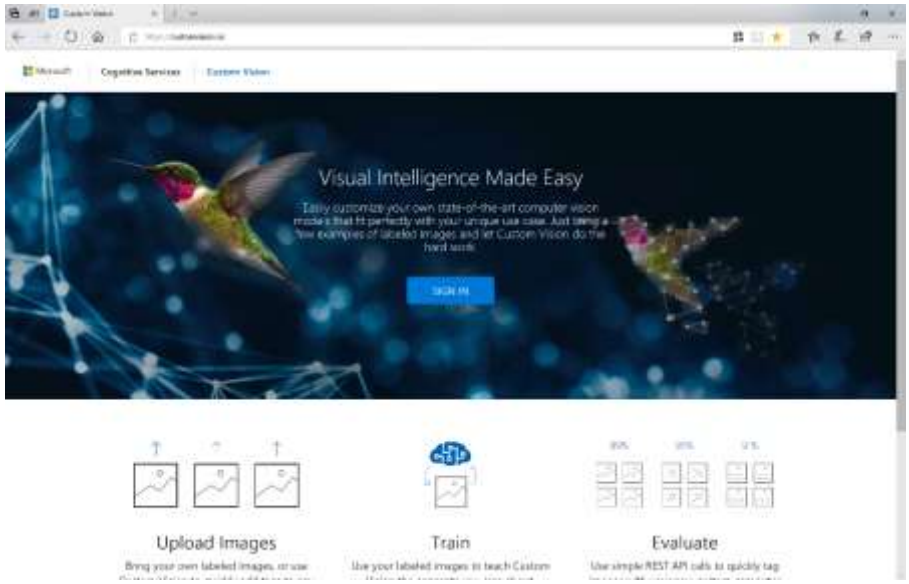
Cognitive Services の Custom Vision は、オリジナルの画像判定エンジンを作成して API で推定値を取得できるサービスです。機械学習などによりご自身で画像判定ロジックを構築することなく、お手持ちの画像から画像判定を行う分析モデルを作成できます。

この演習では Custom Vision Service に画像をアップロードしてタグ付けを行い、カスタム画像分析モデルを作成します。

Custom Vision Service ポータルにアクセスする

Custom Vision Service はポータルが用意されており、画像のアップロードや画像判定エンジンの学習と管理を行うことができます。

1. ブラウザーで Custom Vision ポータル (https://customvision.ai/) にアクセスします。画面中央の **[SIGN IN]** をクリックして、Microsoft アカウントでサインインします。



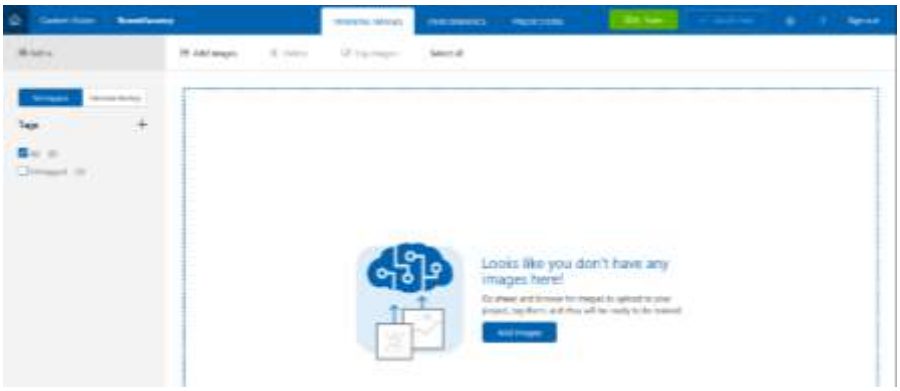
2. 初めてアクセスすると アプリアクセス許可 画面が表示されますので、**[はい]** をクリックします。Custom Vision の *Terms of Service* (サービス条件) にも ☑ をつけて **[I agree]** をクリックします。

新規プロジェクトの作成と学習データ画像のアップロード

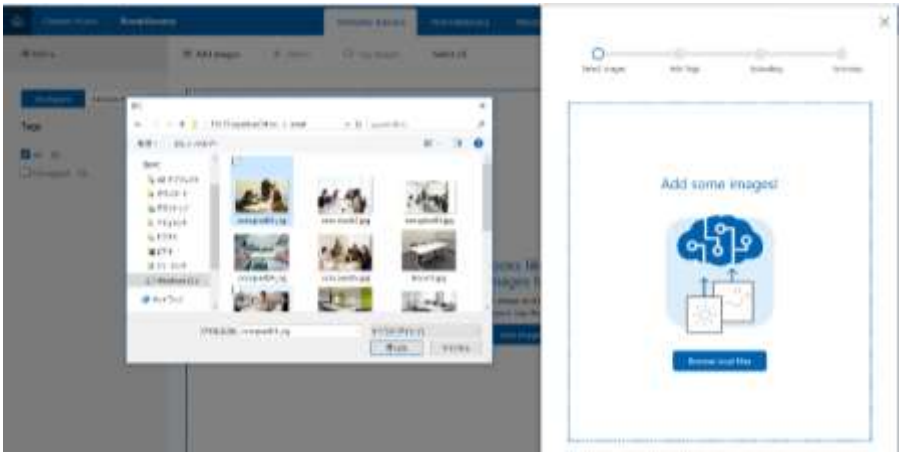
3. 画像判定エンジンは *Project* と呼ばれています。**[New Project]** をクリックして新規プロジェクトを作成します。

4. **Name**（プロジェクト名）と **Description**（プロジェクトの説明）を入力します。Domain (ドメイン、画像のカテゴリ) は **General** を選択します。**[Create Project]** をクリックしてプロジェクトを作成します。

5. 作成したプロジェクトの構成画面が開きます。こちらから画像をアップロードします。画面中央の **[Add images]** をクリックします。



6. assets フォルダー にある画像をすべてダウンロードします。Custom Vision のポータルに戻り、**[Browse local files]** をクリックして、ダウンロードした occupied01.jpg をアップロードします。



7. **[Add some tags...]** にタグを入力します。**occupied** と入力し、＋をクリックしてタグとして追加します。**[Upload 1 File]** をクリックしてアップロードします。

8. *Images updated successfully* と表示されたらタグ付け完了です。**[Done]** をクリックします。

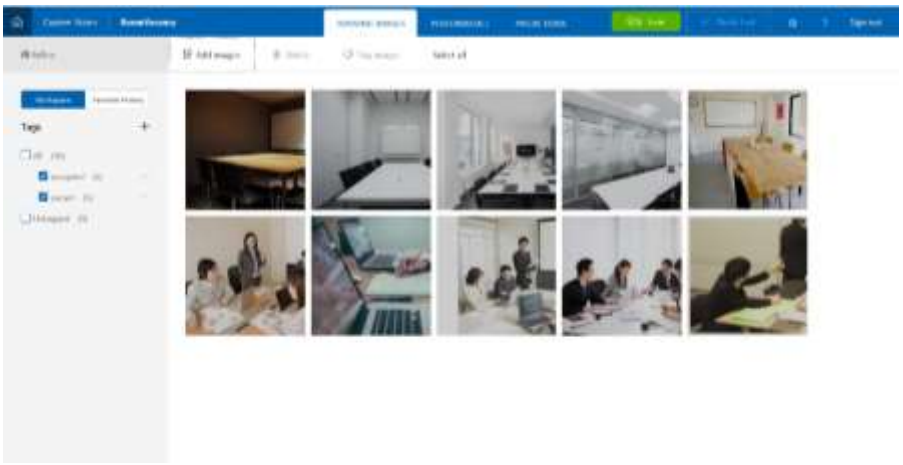
9. タグが左列に、画像が中央部分に表示されます。上部の **Add images** をクリックして、他の画像も追加します。

10. 同様の手順で、occupied02~05 には **occupied** というタグを、vacant01~05 には **vacant** というタグをつけてアップロードします。一度作成したタグは、他の画像のタグ入力時にも候補として表示されます。

同じタグをつけたい画像は、同時に複数アップロードすることが可能です。

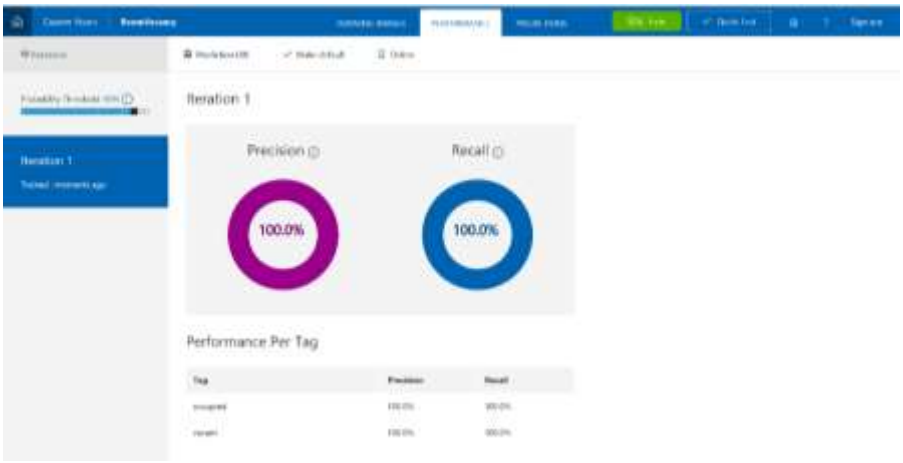
学習データを使って画像判定エンジンを構成する

11. 画面上の [Train] をクリックすると、画像判定エンジンを構築するための学習が始まります。

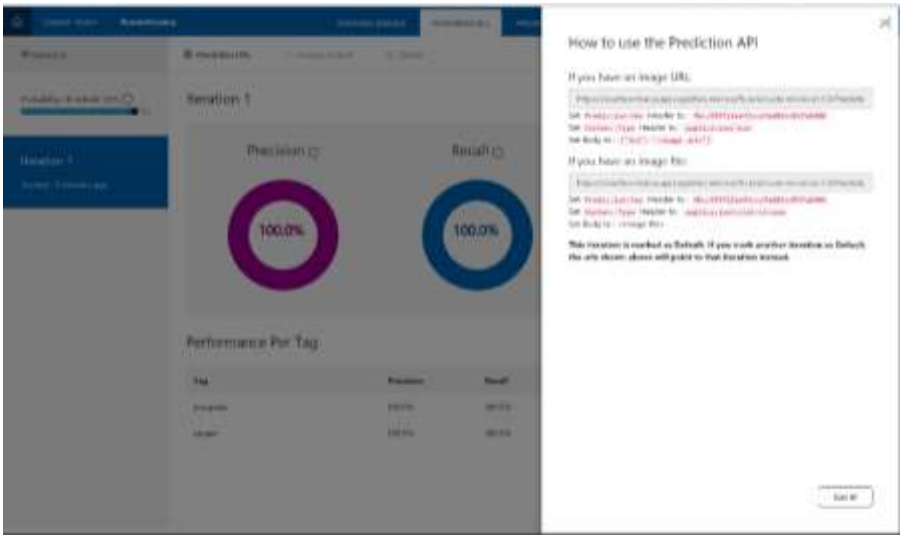


12. 終了すると、学習結果が表示されます。学習が完了すると、Prediction URL が表示されるようになります。✓ **Make default** をクリックして、デフォルトの判定エンジン

に指定します。



13. **Prediction URL** をクリックします。API でアクセスするための情報が表示されます。**If you have an image URL:** の欄にある URL、Predicition Key をコピーして保存しておきます。

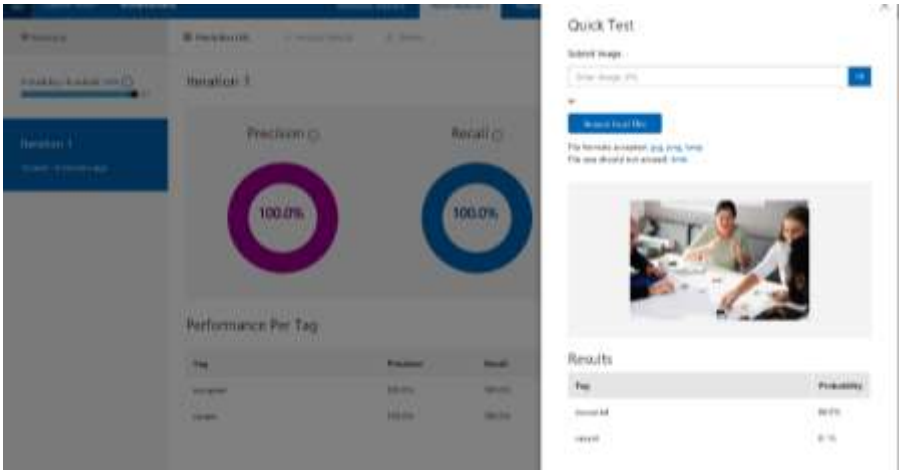


Content-type および Prediction-key を header にセット、Body に 画像を配置している URL (または画像のバイナリデータ) をバインドして、POST します。戻り値は JSON 形式で返されます。

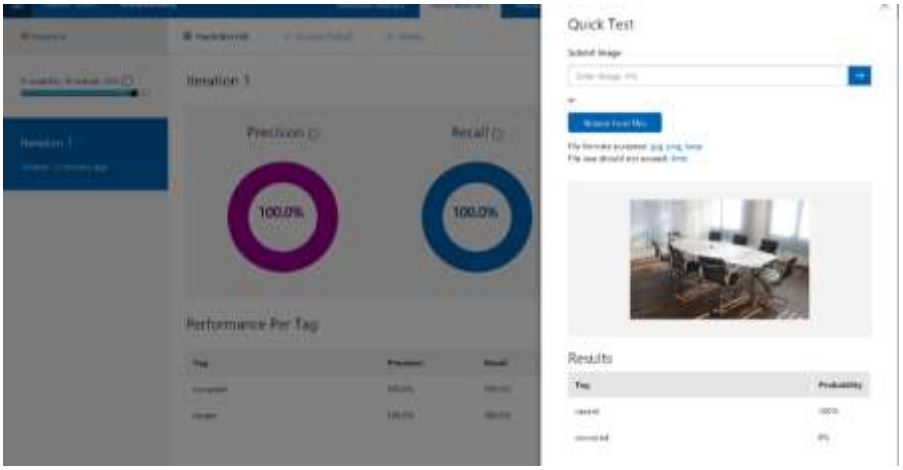
画像判定エンジンをテストする

14. **[Quick Test]** をクリックして、画像判定エンジンのテストを行います。

15. *Quick Test* の画面で **[Browse local file]** をクリックして、assets フォルダー からダウンロードした **occupied_test.jpg** をアップロードします。学習データとして使用していないテスト画像で、判定されたタグ（Tag）と信頼度（Probability）が正しく表示されるのを確認します。



16. 同様に、**vacant_test.jpg** を使用して、vacant タグが一番上位に判定されることを確認してください。画面右上の [×] をクリックしてテスト画面を閉じます。



演習 2: Azure Logic App で自動画像分析ワークフローを構築する

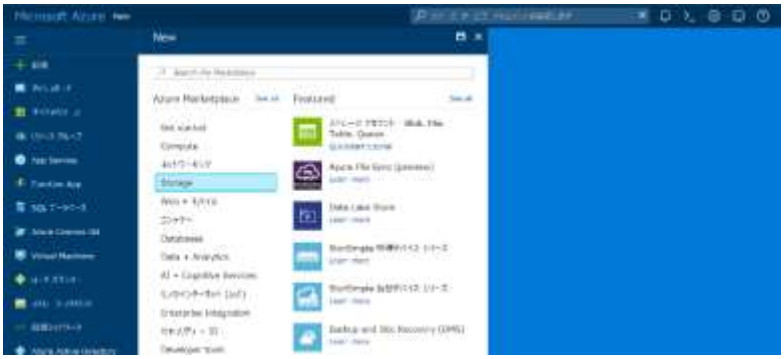
Azure Logic App はコードを書くことなく手軽に複数のサービスを組み合わせて実行するワークフロー機能を構築できます。さまざまなサービスを利用できるコネクターが予め準備されています。

この演習では、まず 画像を保管する Azure Blob ストレージ と分析結果を保存する Azure Table ストレージ を作成します。(会議室の状況を把握するための画像が Azure Blob ストレージに定期的にアップロードされると想定します。) その後、Blob ストレージ の更新をトリガーとして稼働するワークフローを Azure Logic App で作成します。Custom Vision Service の API を呼び出すチャンネルを用いて、Blob にアップロードされた画像を分析、取得したタグと信頼度を Table ストレージに保存します。

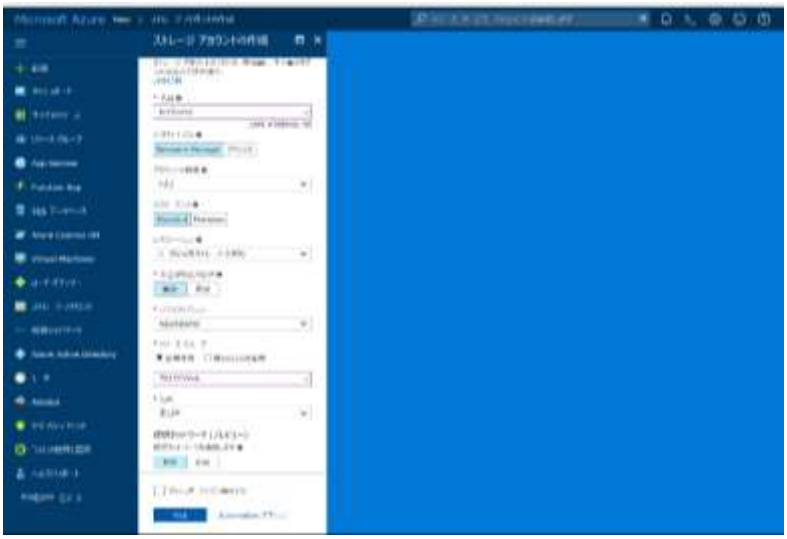
Azure ストレージを作成する

1. Azure ポータル (http://portal.azure.com) をブラウザーで開きます。利用可能なサブスクリプションが紐づけられているマイクロソフトアカウント (または組織アカウント) でサインインします。**+新規** をクリックします。

2. **Storage** タブをクリックして表示される ストレージアカウント をクリックします。



3. ストレージアカウントの作成 ペインで、**名前** を入力し、✓ が表示されるのを確認してください。**リソースグループ** は新規作成を選択し、同一サブスクリプション内で一意となる名前を入力します。場所 は 東日本 (またはお好みのデータセンター) を選択します。**[作成]** をクリックして作成します。

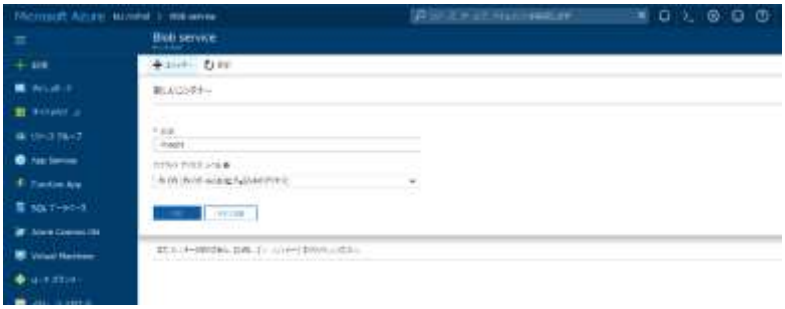


4. 作成完了のメッセージが表示されたら、作成したストレージアカウントを開きます。ストレージアカウントの設定に表示されている **[アクセスキー]** をクリックします。

5. 表示されている ストレージアカウント名、および Key1 の列にあるキーと接続文字列をコピーして保存しておきます。

6. Blob 内に画像を保存するコンテナーを作成します。ストレージアカウント ペインで **[概要]** をクリックし、サービスの欄に表示される **[BLOB]** をクリックします。

7. **Blob service** ペインで **+コンテナー** をクリックして、名前に **images** と入力し、パブリックアクセスレベルは **BLOB** を選択します。**[OK]** をクリックします。



8. コンテナーが作成されたら右上の × をクリックして、Blob Service ペインを閉じます。

9. ストレージアカウントの 概要 ペインで、サービス の欄に表示されている [テーブル] をクリックします。

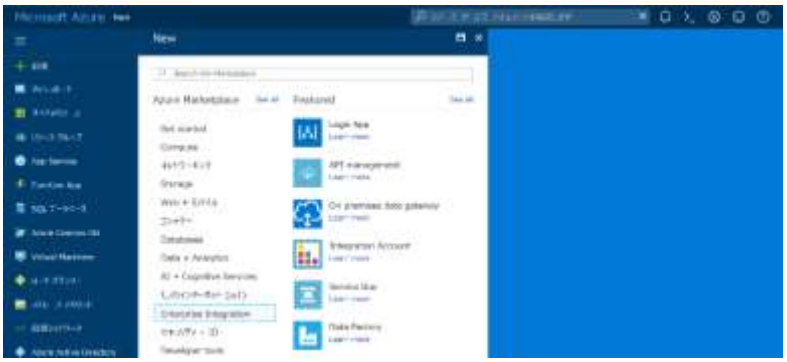
10. 画像を判定したタグなどを保存するテーブルを作成します。**+テーブル** をクリックして、テーブル名に **AnalyzedTable** と入力し、**[OK]** をクリックします。

今回は、同じ Azure ストレージアカウント の Blob および テーブル を作成、利用します。それぞれのエンドポイント (アクセス URL) は以下のようになります。

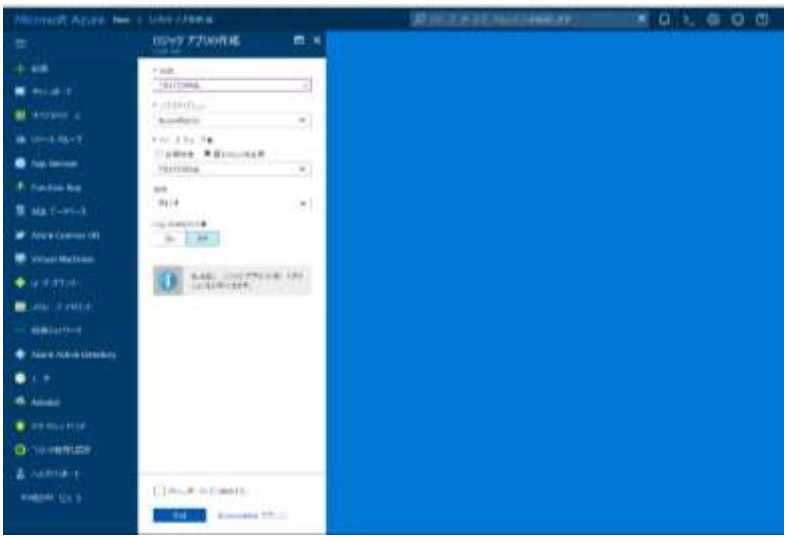
- Blob - https://[指定のストレージアカウント名].blob.core.windows.net/
- テーブル - https://[指定のストレージアカウント名].table.core.windows.net/

Logic App でワークフローを作成する

11. Azure ポータルの画面左端にある **＋新規** をクリックし、表示される **Enterprise Integration** タブをクリックします。**[Logic App]** をクリックして作成します。



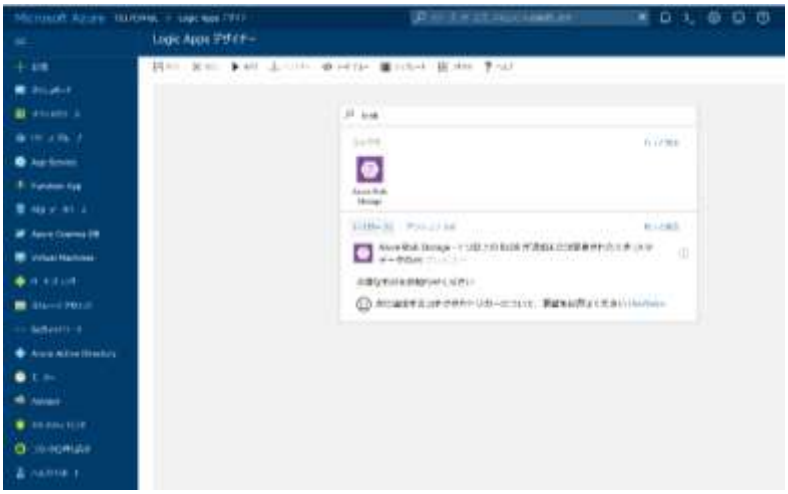
12. ロジックアプリの作成 ペインで、**名前** を入力し ✓ が表示されるのを確認してください。**リソースグループ** はストレージ作成のステップで作成したリソースグループを選択します。**場所** はストレージと同じデータセンター（東日本）を選択します。**[作成]** をクリックします。



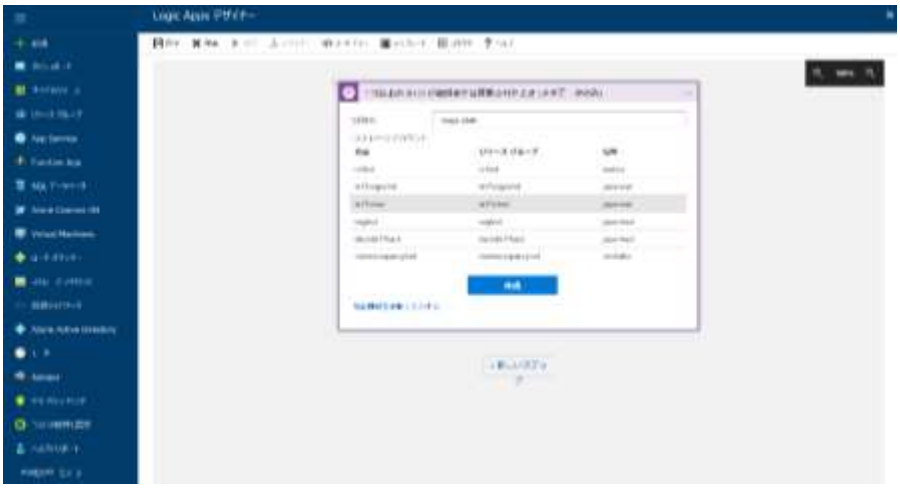
13. 作成完了のメッセージが表示されたら、作成した Logic App を開きます。

14. Logic App デザイナーが開きます。画面をスクロールして、テンプレートの欄にある **+ 空のロジックアプリ** をクリックして、ワークフローを作成します。

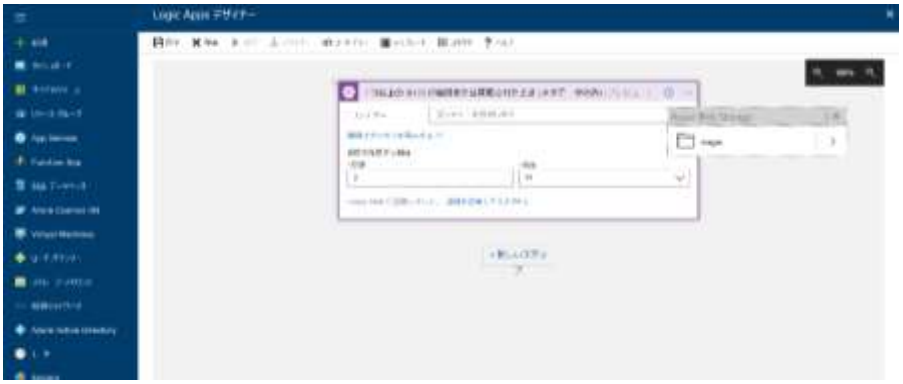
15. まず、このワークフローが起動するトリガーを設定します。検索覧に **Blob** と入力して、 **[Azure Blob Storage - 1 つ以上の BLOB が追加または変更されたとき (メタデータのみ)]** というトリガーをクリックして選択します。



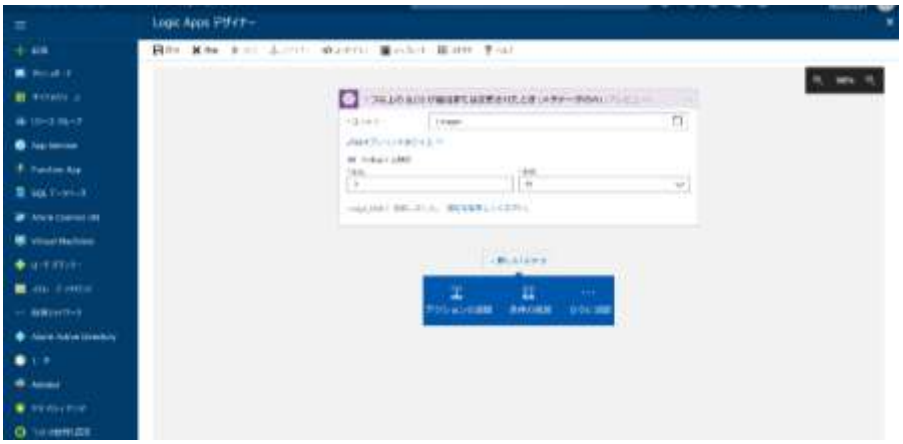
16. 接続名を入力し、上記で作成したストレージアカウントを選択して、**[作成]** をクリックします。



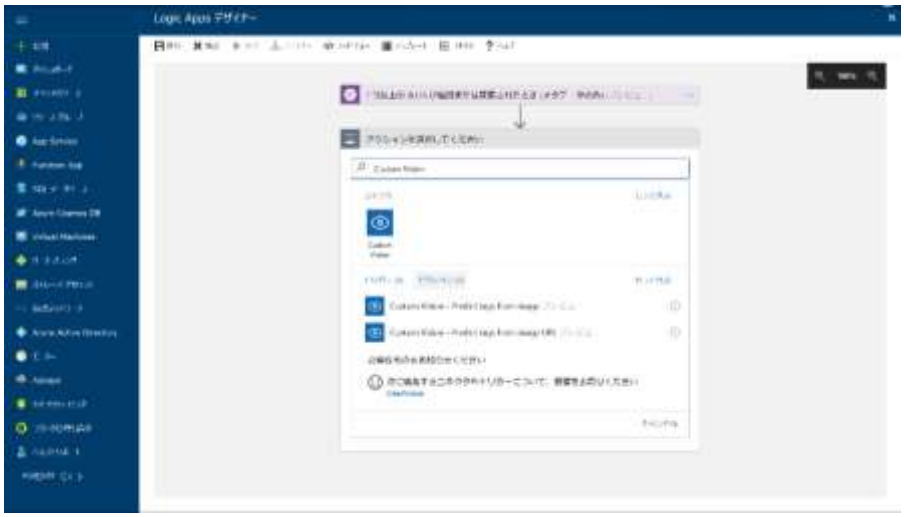
17. **コンテナー**にはこれまでのステップで作成した **image** コンテナーを選択します。



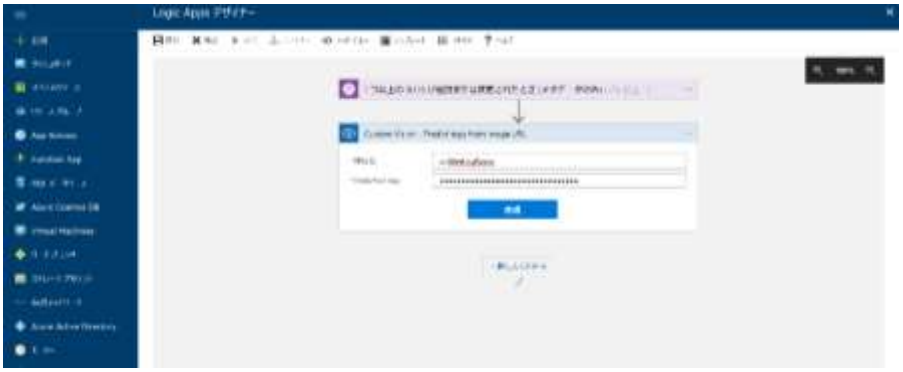
18. **[+次のステップ]** をクリックし、**[アクションの追加]** をクリックし、次のアクションを設定します。



19. 検索欄に **Custom Vision** と入力し、**Custom Vision - Predict tags from image URL** をクリックして選択します。



20. 接続名を入力し、Custom Vision ポータルからコピーしておいた Prediction Key をコピーし、**[作成]** をクリックします。

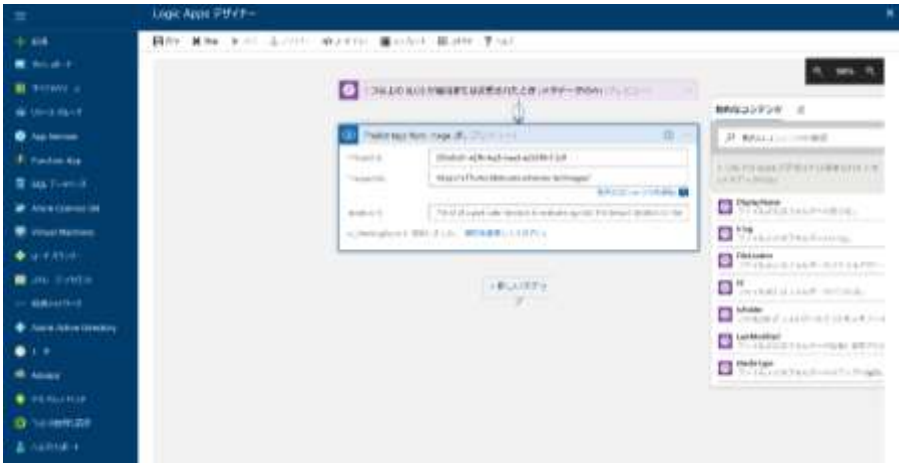


21. Custom Vision からコピーした Prediction URL に含まれる ProjectID をコピーします。Prediction URL は 下記 **YOUR_PRODUCTID** 部分です。

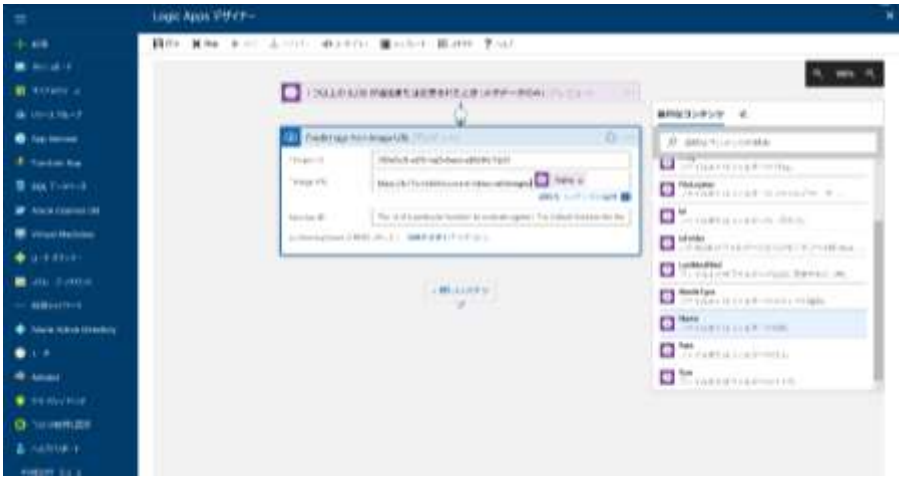
https://southcentralus.api.cognitive.microsoft.com/customvision/v1.0/Prediction/YOUR_PROJECTID/url

22. **Image URL** にはまず、作成した Blob ストレージコンテナーのエンドポイントを下記のフォーマットで入力します。

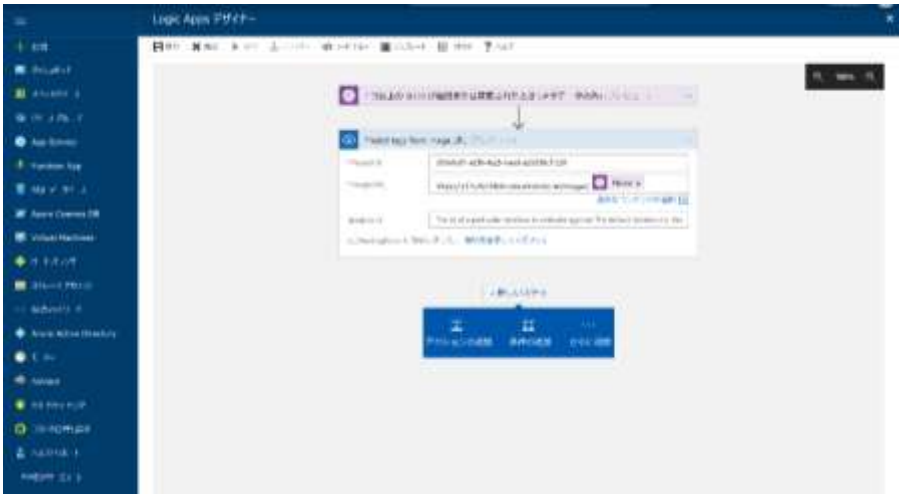
https://[指定のストレージアカウント名].blob.core.windows.net/images/



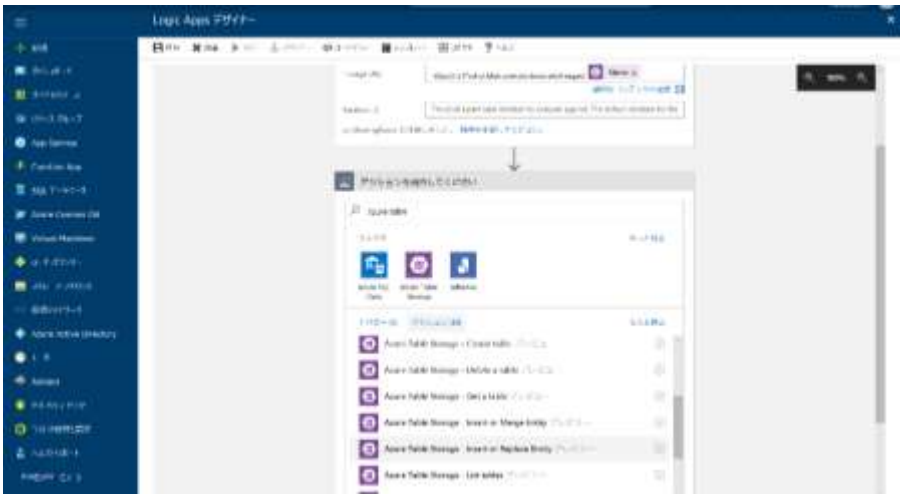
23. 上記の入力に続けて、[動的なコンテンツ] から **[Name]** をクリックして入力します。



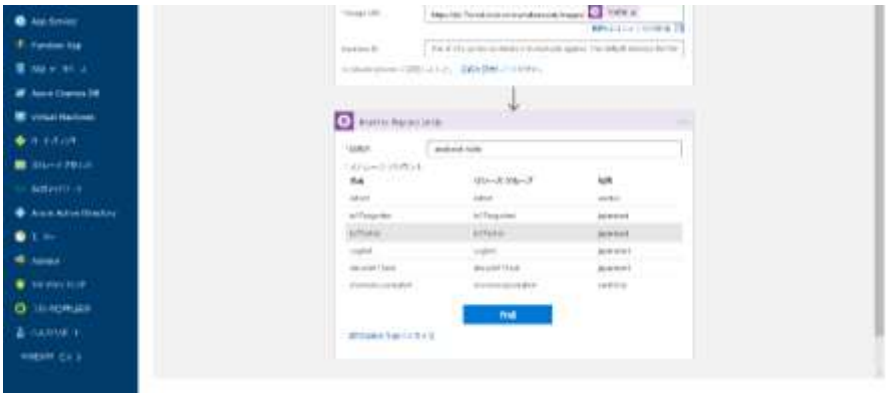
24. **[+次のステップ]** をクリックし、[アクションの追加] をクリックします。



25. **Azure Table** と入力して、下の欄に表示されるアクション一覧から **[Azure Table Storage - Insert or Merge Entity]** をクリックします。



26. Table Storage 設定画面で、これまでのステップで作成した Table ストレージ を選択し、**[作成]** をクリックします。

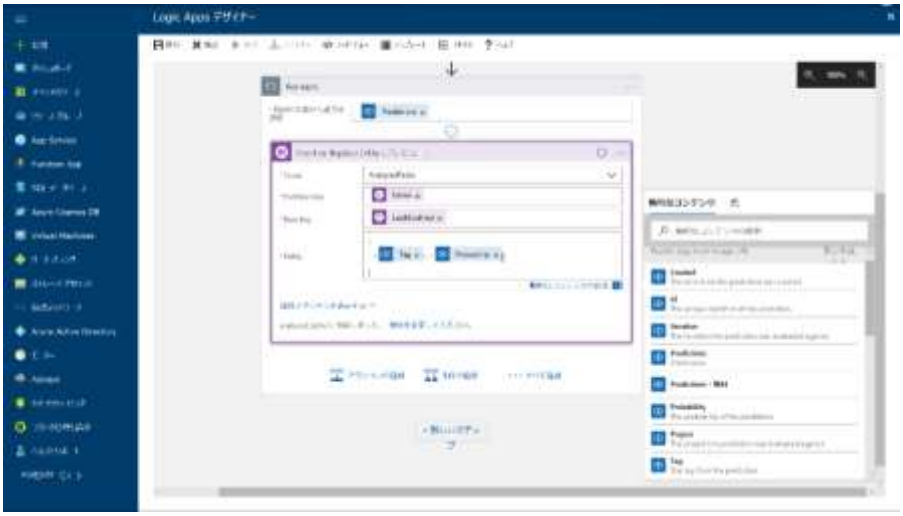


27. **Table** の欄をクリックし、**analyzedTable** を選択します。**Partition Key** には Blob ストレージから取得した **Name**（ファイル名）を選択します。Row Key には **LastModified** (ファイルが更新された日時) を選択します。

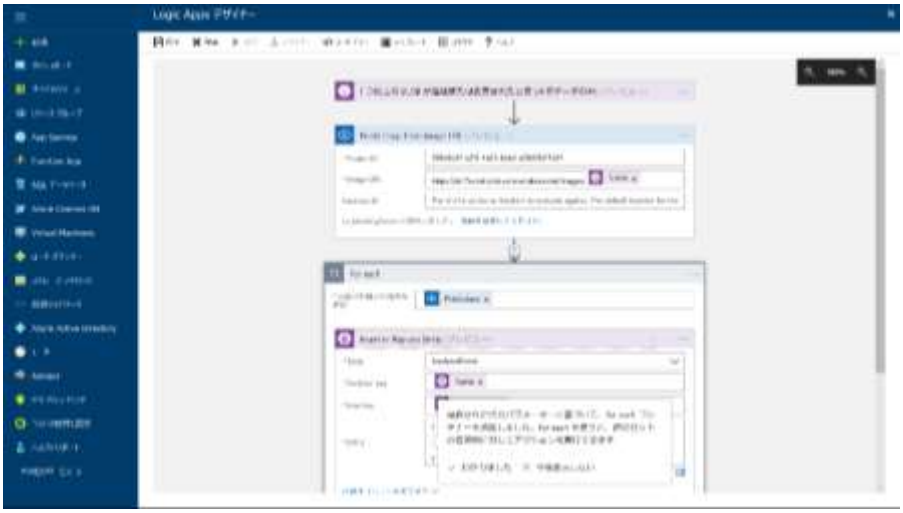
28. **Entity** の欄は、まず以下の構文をコピーします。

```
{
   "Tag": "Probability"
}
```

29. Tag、Probability の部分を [動的なコンテンツ] から選択できる Tag、Probability に差し替えます。



30. [For each を追加します] メッセージが表示されたら OK をクリックします。



Custom Vision の API 戻り値では、一つのレコードに対して複数の Tag と Probability のセットが含まれます。Logic Apps デザイナーではそれに自動で対応し、For each ループを追加します。

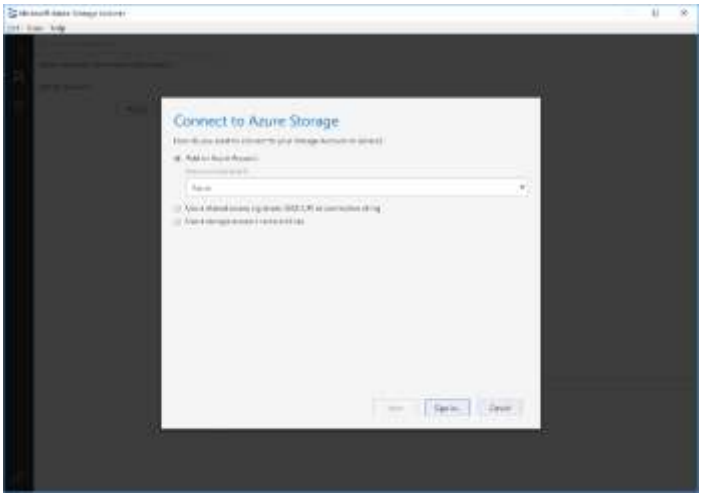
31. 上部バーから **[保存]** をクリックしてこれまでの作業を保存します。確認メッセージが表示されます。

Logic Apps ワークフローの動作を確認する

32. Azure Blob および テーブルストレージの操作を行うため、Azure Storage Explorer (https://azure.microsoft.com/ja-jp/features/storage-explorer/) をダウンロードしてインストールします。

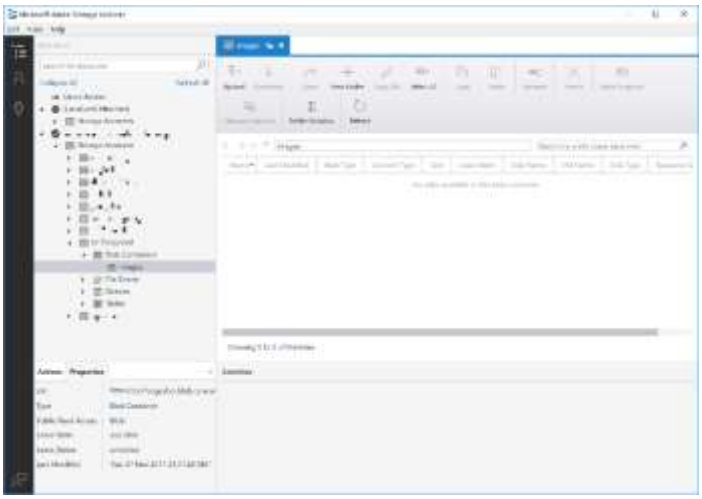
33. Azure Storage Explorer を起動します。Manage Account をクリックし、Add account をクリックして追加します。

34. Connect to Azure Storage 画面で **[Sign in]** をクリックし、Azure サブスクリプションが紐づいているアカウントを追加します。

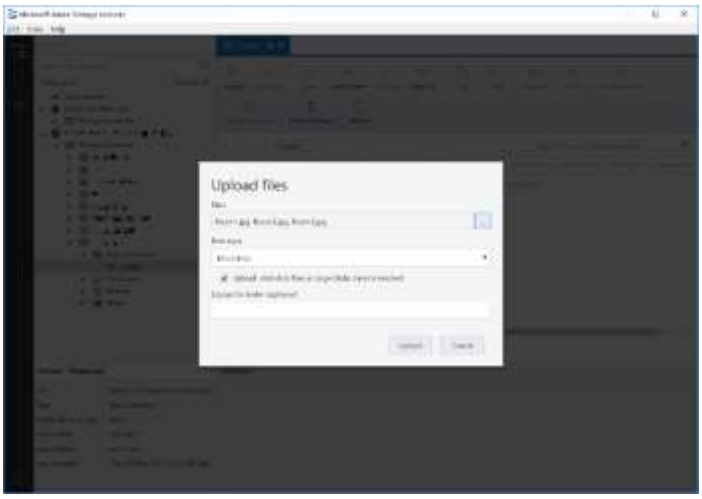


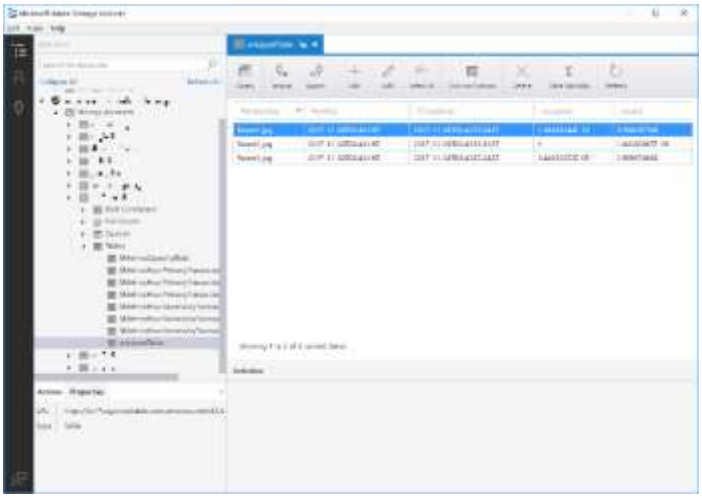
35. 表示されるサブスクリプションを確認して **[Apply]** をクリックします。

36. これまでの手順で作成した ストレージアカウントをクリックして表示します。Blob Container に表示される **Image** コンテナーをクリックします。



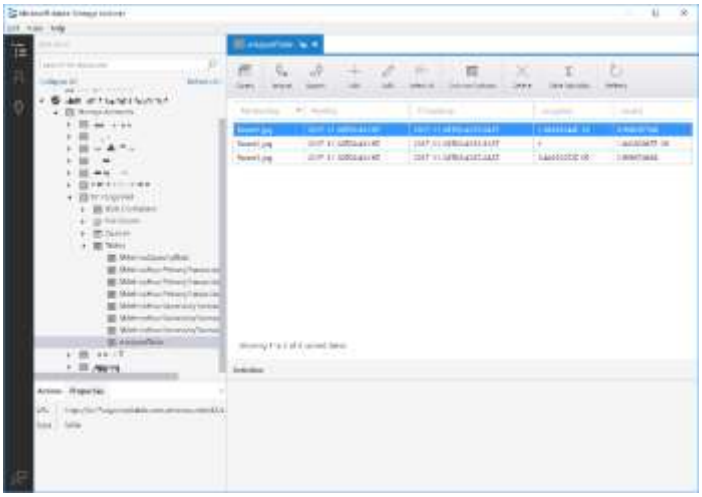
37. 上部バーから **[Upload] > [Upload Files]** をクリックし、assets フォルダーからダウンロードした Room1~3.jpg を選択してアップロードします。これらの画像はそれぞれの会議室の状況を示しています。





38. Azure Portal の Logic Apps デザイナーの画面に戻ります。[▶実行] をクリックして Logic Apps を稼働させます。

39. Azure Storage Explorer に戻ります。作成したストレージの Tables > AnalyzedTable をクリックします。画像の判定結果が挿入されていれば、Logic Apps ワークフローの完成です。



演習 3: Bot Framework で自動応答ボットを構築する

この演習では、演習 2 までで作成した Custom Vision のカスタム画像分析エンジンを利用した自動画像判定ワークフローの結果を自動返答するボットを構築します。

この演習では、Bot Builder SDK for .NET を使用して、演習 2 までで作成した画像判定結果を自動応答するボットを構築します。

Bot Builder SDK for .NET は、Visual Studio と Windows を使用してチャットボット (Chatbot) を開発するためのフレームワークです。SDK では C# を活用し、.NET 開発者にとってなじみのある方法で強力なボットを作成する手段を提供します。

> Bot Builder SDK for .NET は現在 C# 版となります。Visual Studio for Mac はサポートされていません。

Bot Application テンプレートを使用して新規ボットを作成する

1. Bot Application テンプレートをダウンロードし、zip ファイルを Visual Studio 2017 プロジェクト テンプレート ディレクトリに保存してテンプレートをインストールします。Visual Studio 2017 プロジェクト テンプレート ディレクトリは通常、以下の場所にあります。

   ```
   %USERPROFILE%¥Documents¥Visual Studio 2017¥ Templates¥
   ProjectTemplates¥Visual C#¥
   ```

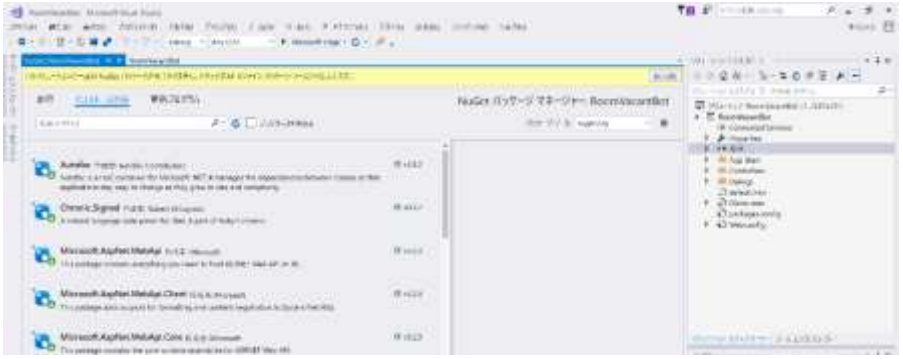
2. Visual Studio を開き、ツールバーの [ファイル] > [新規作成] > [プロジェクト] をクリックして、新規プロジェクトを作成します。

3. C# の Bot Application テンプレートを選択します。プロジェクト名には RoomVacantBot を使用します。

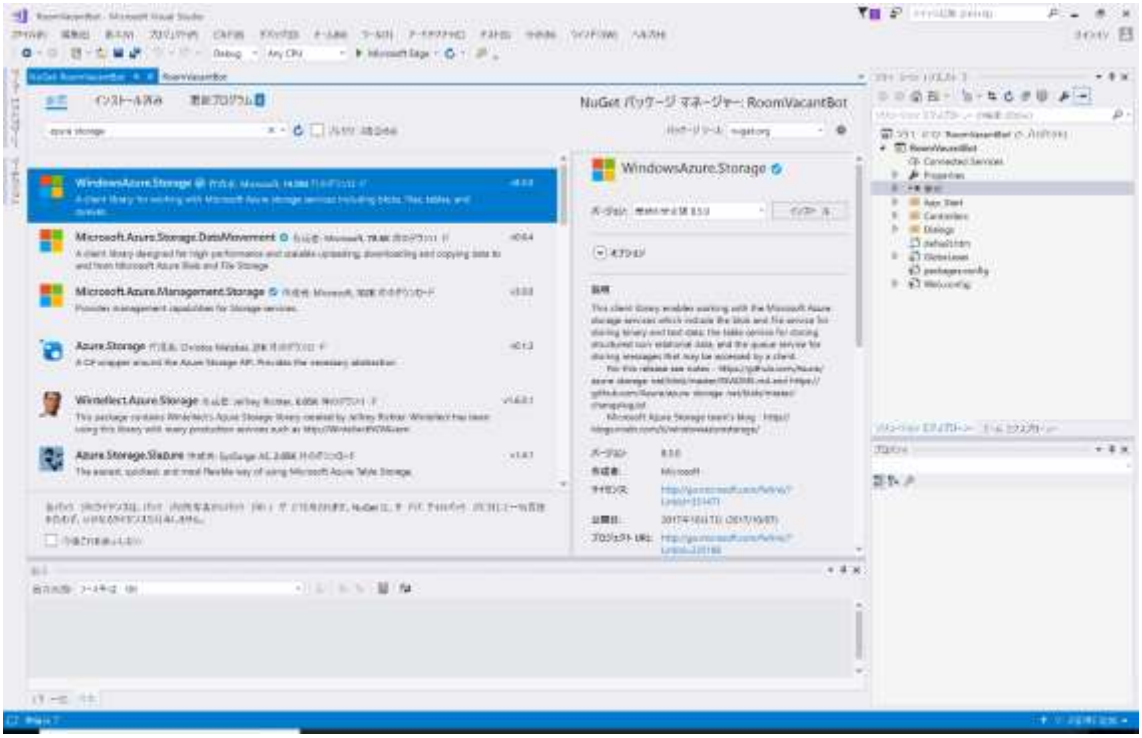
Bot Application テンプレートを使用することで、単純なボットの作成に必要なすべてのコンポーネントが既に含まれているプロジェクトを作成することになります。これには、Microsoft.Bot.Builder NuGet パッケージに含まれる Bot Builder SDK for .NET への参照も含まれます。

4. ソリューション エクスプローラーでプロジェクトの [参照] フォルダーを右クリックして、[NuGet パッケージの管理] をクリックします。

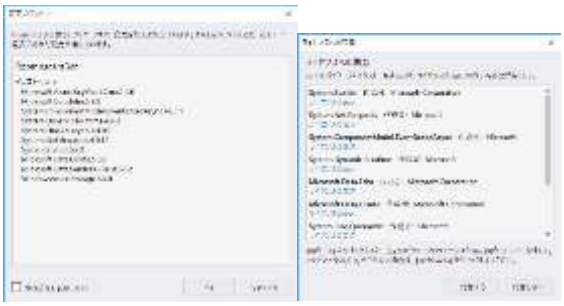
5. 「このソリューションに一部の NuGet パッケージが見つかりません...」と表示される場合は、その横にある [復元] をクリックして、パッケージを復元します。



6. [参照] タブの検索欄に azure storage と入力して WindowsAzure.Storage を探します。[インストール] をクリックして、最新の安定版をインストールします。



7. 表示に従って、[OK] をクリックして変更を受け入れ、パッケージを更新します。

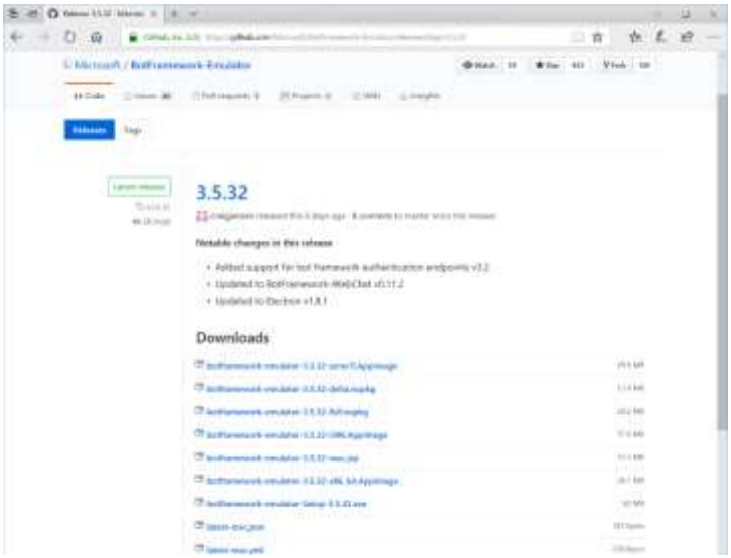


ボットをテストする

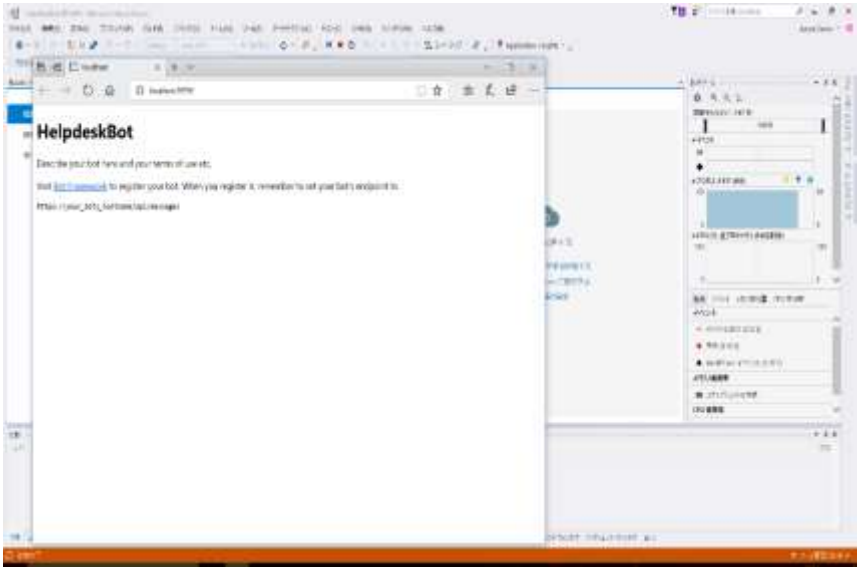
まずこの段階で、Bot Framework Emulator を使用してボットをテストし、動作の様子を見てみましょう。このエミュレーターは、localhost 上のボット、またはトンネルを通じてリモートで実行しているボットをテストおよびデバッグできる、デスクトップ アプリケーションです。エミュレーターは、Web チャットの UI に表示されるとおりにメッセージを表示し、JSON 要求をログに記録し、ユーザーがボットとメッセージのやり取りを行うの同

様に応答します。

8. まず、Bot Framework Emulator の Web サイトから botframwork-emurator-setup-xxx.exe というインストーラーをダウンロードし、ローカルで起動してインストールを行います。

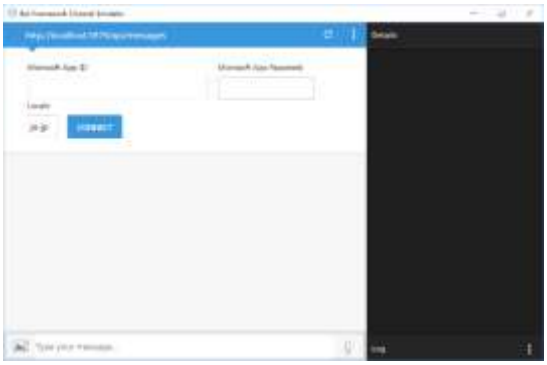


9. エミュレーターのインストール後、IIS Express をアプリケーション ホストとして使用して、Visual Studio でこれまでのタスクで作成したボットを起動します。Visual Studio の [実行] ボタンをクリックすると、Visual Studio でアプリケーションが構築されて localhost に展開、Web ブラウザーが起動してアプリケーションの default.htm ページを表示します。



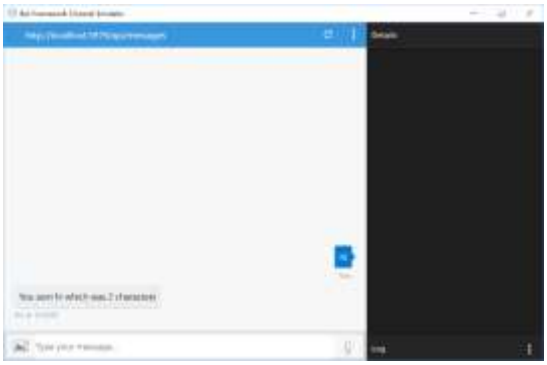
Windows ファイアウォールの警告が表示される場合は、[アクセスを許可] をクリックします。

10. 次に、エミュレーターを起動し、ボットに接続します。アドレス バーに http://localhost:3979/api/messages と入力します。これは、ボットがローカルにホストされたときにリッスンする既定のエンドポイントです。[ロケール] を ja-jp に設定し、**[Connect]** をクリックします。



ボットをローカルで実行しているので、[Microsoft App ID] と [Microsoft App Password] を指定する必要はありません。これらのフィールドは、今のところ空白のままにしてかまいません。この情報は、Bot Framework Portal にボットを登録する際に取得します。

11. 何かメッセージを送信します。送信したメッセージに対して、先頭に "You sent"、末尾に "which was ## characters" (## はユーザーのメッセージの文字数) というテキストを付けて、”おうむ返し” にボットが応答するのを確認できます。



12. ブラウザーを終了し、Visual Studio の [停止] ボタンをクリックして、アプリケーションを停止します。

会話実行コード（RootDialog.cs）の開発

上記のテストの通りに入力の “おうむ返し” を行うコードは、Bot Framework のテンプレートで生成される Dialog¥RootDialog.cs に含まれています。こちらにコードを追加して、Azure Table の結果を取得し、部屋の利用状況を返答する機能を実装します。

13. RootDialog.cs を開きます。ユーザーから会話が入力されると、RoodDialog の StartAsync から MessageReceivedAsync が呼び出され、実行されます。まずは冒頭に Azure Storage クライアントライブラリーや、他に利用するライブラリー群への参照を追加します。

```
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.Table;
using System.Linq;
using System.Collections.Generic;
```

14. StartAsync の前に Azure ストレージへの接続情報を追加します。

```
namespace RoomVacantBot.Dialogs
{
    [Serializable]
    public class RootDialog : IDialog<object>
    {
        readonly static string azureStorageAccount = "YourAzureStorageAccount";
        readonly static string azureStorageKey = "YourAzureStorageKey";
        readonly static string azureStorageConnStr
            = "DefaultEndpointsProtocol=https;AccountName="+
            azureStorageAccount + ";AccountKey=" + azureStorageKey +
            ";EndpointSuffix=core.windows.net";

        public Task StartAsync(IDialogContext context)
        {
            :
            (後略)
```

15. MessageReceivedAsync を下記の内容で書き換えます。ユーザーからの入力を受信したら、Azure Table の値を取得して、会議室の一覧を作成し、ユーザーに会議室の選択を尋ねるプロンプトを実装します。

```
private async Task MessageReceivedAsync(IDialogContext context,
        IAwaitable<object> result)
{
        var tableResultList = await GetTableResult();
        var roomList = await GetRoomList(tableResultList);
        PromptDialog.Choice(context, RoomStatusAsync, roomList,
                "空きを調べたい会議室を選択してください。");
}
```

16. Azure Table の値を取得する GetTableResult を追加します。Azure ストレージへ接続、AnalyzedTable に保存されている値を取得して、resultEntity クラスで定義した TableEntity に格納します。

```
public class resultEntity : TableEntity
{
        public resultEntity(string roomId, string modifiedTime)
        {
                PartitionKey = roomId;
                RowKey = modifiedTime;
        }
        public resultEntity() { }
        public string occupied { get; set; }
        public string vacant { get; set; }
}

private async Task<List<resultEntity>> GetTableResult()
{
        var storageAccount = CloudStorageAccount.Parse(azureStorageConnStr);
        var tableClient = storageAccount.CreateCloudTableClient();
        var table = tableClient.GetTableReference("AnalyzedTable");
        var query = new TableQuery<resultEntity>();
        var tableResultList = table.ExecuteQuery(query).ToList();

        return tableResultList;
}
```

17. 取得した resultEntity に含まれるファイル名から会議室名を取り出してリスト化する GetRoomList を追加します。

```
private async Task<List<string>> GetRoomList(List<resultEntity> tableResultList)
{
	var roomList = new List<string>();

	foreach (var item in tableResultList)
	{
		roomList.Add(item.PartitionKey.ToString().Replace(".jpg",""));
	}
	return roomList;
}
```

18. 選択された会議室を検索し、結果を取得して画像と共に表示する、RoomStatusAsync を下記のように追加します。

```
private async Task RoomStatusAsync(IDialogContext context, IAwaitable<string>
result)
{
	var roomId = await result;

	var storageAccount = CloudStorageAccount.Parse(azureStorageConnStr);
	var tableClient = storageAccount.CreateCloudTableClient();
	var table = tableClient.GetTableReference("analyzedTable");
	var query
		= new
		TableQuery<resultEntity>().Where(TableQuery.GenerateFilterCondition
		("PartitionKey", QueryComparisons.Equal, roomId+".jpg" ));
	var tableResultList = table.ExecuteQuery(query).ToList();

	var roomStatusMsg = $"{roomId} の利用状況を判定できませんでした。";
	foreach (var item in tableResultList)
	{
		if (item.PartitionKey == roomId + ".jpg")
		{
			var occupied = Double.Parse(item.occupied);
			var vacant = Double.Parse(item.vacant);

			if (item.occupied > 0.7 && item.occupied > item.vacant)
			{
				roomStatusMsg = $"{roomId} は使用中です。";
```

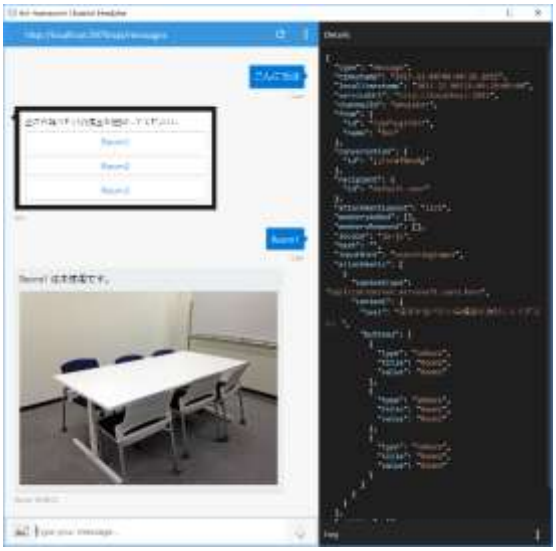
```
                }
                else if (item.vacant > 0.7 && item.vacant > item.occupied)
                {
                        roomStatusMsg = $"{roomId} は未使用です。";
                }
            }
        }

        var message = context.MakeMessage();
        message.Text = roomStatusMsg;
        message.Attachments.Add((new Attachment()
        {
            ContentUrl = "https://" + azureStorageAccount +
                                ".blob.core.windows.net/images/" + roomId +".jpg",
            ContentType = "image/jpg",
        }));

        await context.PostAsync(message);

        context.Done<object>(null);
}
```

今回は TableQuery.GenerateFilterCondition を使って、PartitionKey ＝RoomID と一致するアイテムを取得します。

また、Custom Vision の判定結果が 0.7 以下の場合は、判別不可のメッセージを表示しています。この “しきい値” は Custom Vision の判定結果によって調整してください。

ボットの動きを確認する

19. Visual Studio で、ボットアプリケーションを起動します。エミュレーターを起動し、ボットに接続します。アドレス バーに http://localhost:3979/api/messages と入力し、**[Connect]** をクリックして接続します。何か入力すると、会議室の一覧を表示します。会議室をひとつ選択すると、画像から判定した空き情報を画像と共に表示します。



おつかれさまでした。以上で、Custom Vision Service により、自動で画像を分析して、会議室の空き状況を自動で返答するボットは完成です。

まとめ

このハンズオン ラボでは、以下の方法について学習しました。

- Cognitive Services Custom Vision Service を利用してカスタム画像分析モデルを作成する
- Azure Blob ストレージへの画像アップロードをトリガーとして、その画像を Custom Vision Service の Prediction API を用いて分析し、取得したタグと確度を Azure Table ストレージに保存するワークフローを、Azure Logic App で構築する
- Azure Table ストレージのデータを元に自動で返答するボットを Microsoft Bot Framework で構築する
