Azure Functionsでサーバーレスアーキテクチャが何かを理解する

AWS C# Azure lambda serverless

この記事は最終更新日から1年以上が経過しています。

サーバーレスアーキテクチャとAzure Functions

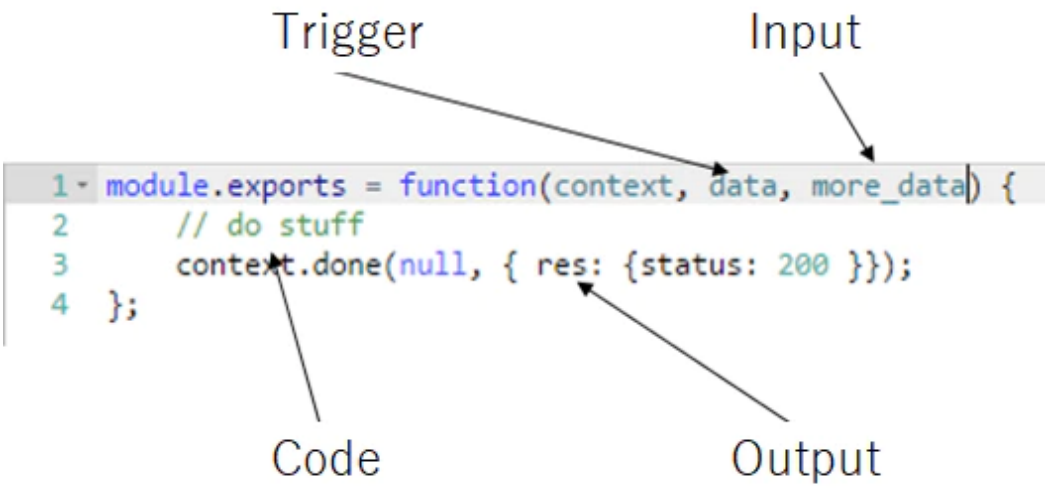
サーバレスアーキテクチャとは、マネージドサービスのみを利用してシステムを構築するアーキテクチャです。そして、Azure Functionは非常駐型のプロセスをイベントによってトリガーして実行するサービスです。

Azure Functionsの特徴はサーバーでプログラムを直接動作することができるということで、「プログラマが考えるのはコードだけ」というようにプログラマは必要なプログラミングだけに集中できます。そのため、その他のプログラムに関わらない多くのことが抽象化されています。
そうです、あなたがフォーカスするのはプログラミングだ

け。Azure Functionsを使えば今までインフラの制御などに奪われがちだった時間はあなたのものになり、より簡潔で楽しいプログラミングの時間を持つことが出来るのです。

Azure Functionsのプログラミング

Azure functionsのプログラミングコンセプトはとても簡単です。下の図を見てわかる通り、大変簡単な理解しやすい作りになっています。フローを確認すると、まずトリガーになるイベントを取得し、コードで読ませたいデータなどのリソースをインプットとして、出したい結果をアウトプットとして設定します。あとはコードを書くだけです。Functionsは複数のインプット、複数のアウトプット先を持てます。



Azure Functionsを利用するメリット

ここで、Azure Functionsを利用することで得られるメリットについてもう少し深く掘っていきましょう。
Azure Functionsを利用することによるメリットは「フォーカスできる」ということです。従来のようにプログラマーがサーバーを意識する必要性や、設計者がアーキテクチャの面でスケールアウトのことを深く考える必要性がなくなった結果、プログマーはコーディングに集中でき、設計者はビジネスのロジックに集中することができます。　それを可能にし

ている理由はAzure Functionsでは「バインディング」の機能によってデータやリソースの連携部分が抽象化されているからです。
OSのレイヤー等の低いレイヤーはもちろん、複雑な連携部分もAzure Functionsがマネージするので、ユーザーは「真に人間が考えるべきコード」だけを書けばよいのです。

また、もう一つのメリットはコストが低いということでしょう。コストでは料金的なコストと時間的なコストを減らすことが出来ます。Azure Functionsでは従来型のVMが起動している時間に対する課金形態以外に、実行したら実行した時間だけお金がかかる課金形態も選べるのでコストも抑えられ、経済的です。バインディングの機能によって開発の時間を大幅に短縮できることもメリットです。例えばメッセージングアプリケーションで簡易的なBOTの機能を作る際には、データベースの接続部分等のややこしい部分のプログラミングを忘れて、会話のコアとなるプログラミングをいきなり始めることが出来るのです。運用の際にもスケールアウト等はAzure Functionsがやってくれます。

サーバーレスアーキテクチャの利点まとめ

- やりたいことにフォーカスできる : データやリソースの連携部分の抽象化 / 自動的にスケールし、任意の処理が実行できる
- 料金的/時間的なコストの低さ：利用開始のコストは低く、簡単に体験できる

Azure Functions

ここで Azure functions の機能的な面を詳しく見ていきましょう。
実際に以下の Azure Functionsのサービスの機能を見ると、Azure Functionsの機能がいかにサーバー レスアーキテクチャやマイクロサービスアーキテクチャといったアーキテクチャのメリットを享受できるように作られたかがわかると思います。以下に挙げられるようにC#やnode.js/javascript、PHPなど、普段あなたが開発に使っている言語を利用することができ、Azure Functionsの関数を発火させるためのトリガーもHTTPSの提供される簡単なWEBAPIや、cronの形式で定義できるタイマーなど、利用しやすいものとなっています。

Azure Functionsの特徴

- Azure Functions = イベント駆動の「機能」実行プラットフォームで、柔軟で強力なサーバーレス スクリプト実行環境
- クラウドアプリケーションを驚くほど簡単に作成できる

- インフラを抽象化し設計者はビジネス、開発者はコードに注力できる
- 業務要件に応じたスケーリングが可能
- HTTP APIのエンドポイントとして公開可能
- Functionsは様々なプログラミング言語に対応
    - C#, Node.js/Javascript, F#, Python, PHP, Batch, Bash, PowerShell
- ファイル作成やPush通知を容易にする「バインディング」の機能をもつ
    - トリガー , スケジュール , HTTP (Webhook) , キュー

Azure Functionsのバインディングとトリガー

バインディングはAzure FunctionsをAzure Functionsたらしめる最も重要な機能と言えます。2016/08/10の時点で以下のト

リガー/バインディングをサポートしています。

タイプ	サービス	トリガー	インプット	アウトプット
スケジュール（タイマー）	Azure Functions	✔	-	-
HTTP (REST / WebHook)	Azure Functions	✔	-	✔
Blob ストレージ	Azure Storage	✔	✔	✔
キュー	Azure Storage	✔	-	✔
テーブル	Azure Storage	-	✔	✔
テーブル	Azure Mobile Apps Easy Tables	-	✔	✔
No-SQL DB	Azure DocumentDB	-	✔	✔
ストリーム	Azure Event Hubs	✔	-	✔
プッシュ通知	Azure Notification Hubs	-	-	✔
キュー / トピック	Azure Service Bus Queue / Topic	-	-	✔
SaaS ファイル	Box	✔	✔	✔
	Dropbox	✔	✔	✔
	OneDrive	✔	✔	✔
	OneDrive for Business	✔	✔	✔
	FTP	-	✔	✔
	SFTP	-	✔	✔
	Google Drive	-	✔	✔

このバインディングのメリットは接続が簡単になるだけではありません。例えば、SaaSのプロバイダーがAPIの形式を変更した場合にアプリケーション側ではそれらのメンテナンスをする必要がほとんどなくなるのです。
この機能によってAPI仕様の変化を特に気にせずにSaaSに接続できるようになるので、あなたのプログラミングの可能性はさらに広がるでしょう。

Azure Functionsの主要なトリガー

トリガーの中でも利用頻度が高そうなものを紹介します。これらのバインディングを使えば、HTTPのAPIやWebHook、Cronの使い心地のタイマーで楽に関数をトリガーすることができるでしょう。

HTTP

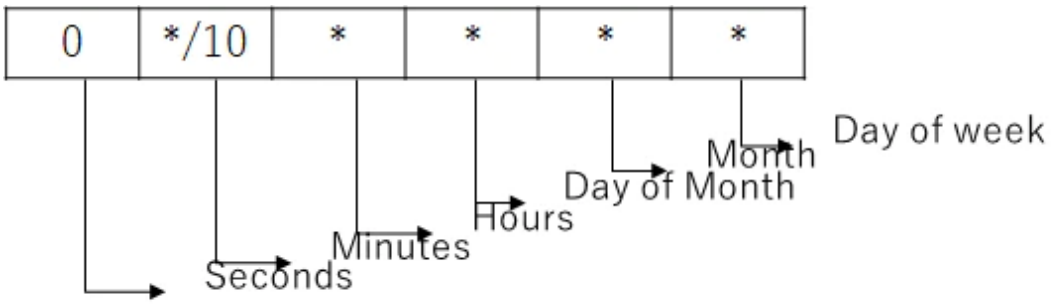
増えてゆくサービスへ接続するもっとも簡単な方法の一つ
簡単に作成でき、ワークフローを自動化できます。

WebHooks

Functionsは、WebHooks をリッスンするエンドポイントを簡単に作成可能
ASP.NET のWebHooks libraryを使えば、バラエティに富んだトークンベースの検証機能を利用可能

タイマー

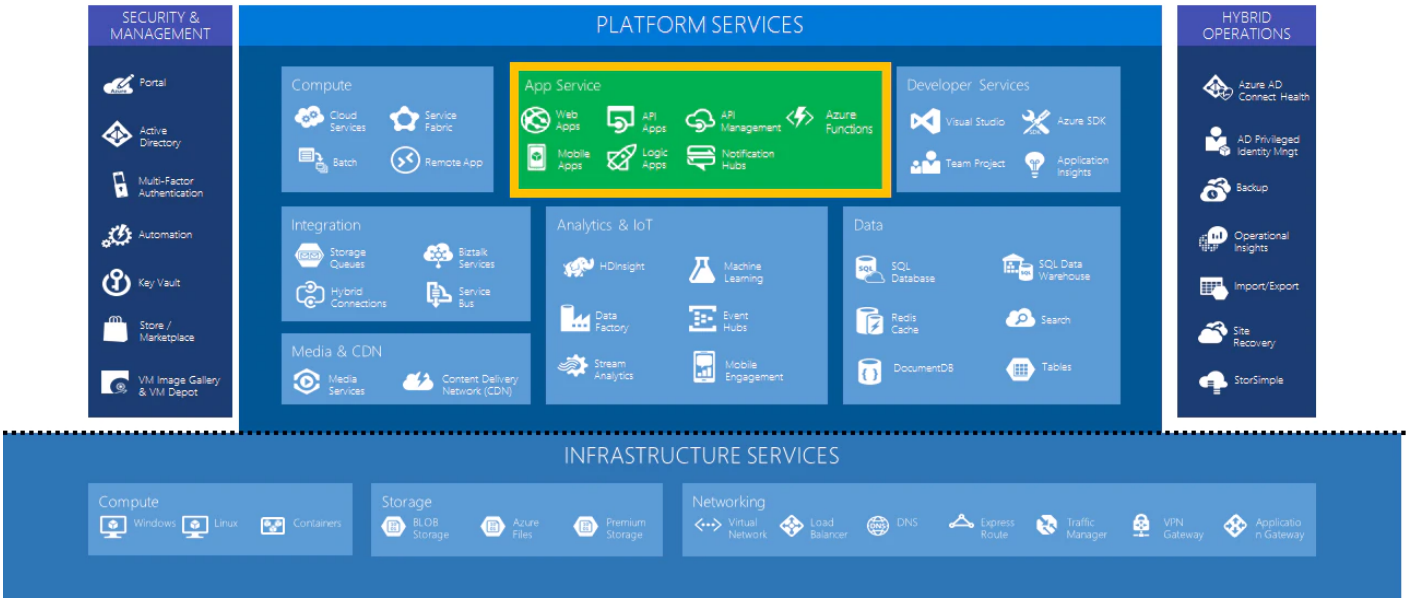
Cronの表現方法で記述可能です。



Azure Functionsの位置づけ

FucntionsはAzureの中でApp Serviceに属します。具体的にはPaaSの分野で、ユーザーがサーバーのことを考えずにシステ

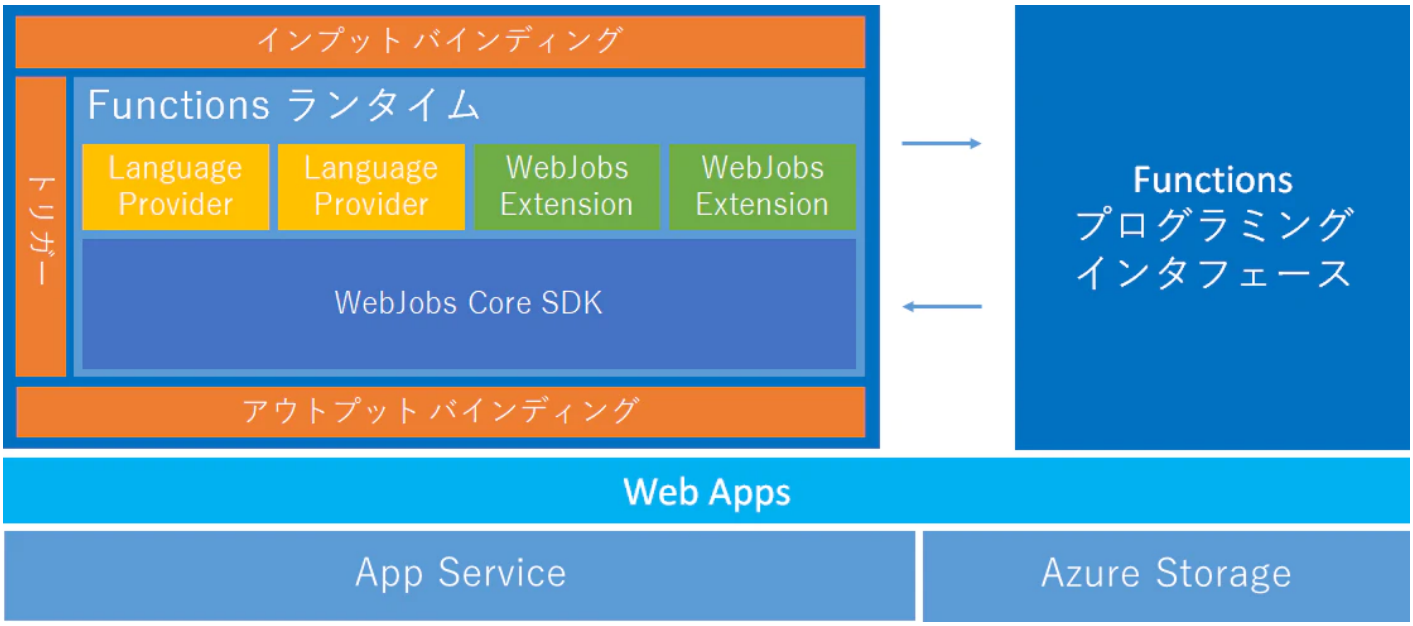
ムを提供できるようにするための仕組みです。



Azure FunctionsのアーキテクチャとWebJobsとの違い

Azure Functionsの中がどうなっているのかを見ていきましょう。この図を見てわかる通り、Azure Functionsの中身は

WebJobsであることがわかります。



ここで、言及すべきWebJobsとの大きな違いはHTTPのサポートです。WebJobsはHTTPをサポートしていないので、外部にHTTPのAPIとして提供することが出来ません。しかし、Azure FunctionsはHTTPをサポートしているので、外部に向けてHTTPのAPIとして公開することが出来ます。また、Azure FunctionsとWebJobsの違いについて、もう少し言及すると、以下になります。

- WebJobs　＝　自由、構築が多少面倒、App Service Plan（VMを動かした時間課金）
- Azure Functions　＝　ある程度の制約、構築がかなり楽、App Service Planに加えてDynamic Service Plan（関数が動

いている時間のみ課金）もサポート

具体的な違いは以下の表に載せてあります（2016/08/10時点）

	WebJobs	Functions
プログラミングモデル	C#	C# & Node.js +その他多種多様
	従来の開発環境(Visual Studio, NuGet, MSBuild)	より多様な開発環境(Web portal, VSCode, dynamically builds itself)
	HTTPをサポートしない	HTTPのサポート
	多くの関数が一つのクラスに存在	存在するのは一つの関数
ホスティングモデル	ホストの設定を自分でして、コンソールアプリケーションを作る	ホストの設定は限られるが、コードを書くだけ
	スケーリングは自分で設定	スケーリングは最適化される、
	Web/Mobile/API appのバックグラウンドで動く、	Function Appがwebフロントエンドも含みすべてをマネージする
	SDKベース以外のアプリも動く	Functions対応アプリのみ動く
共通の機能	機能指向プログラム	
	外部ライブラリ利用可能	
	Binding(Input Output Trigger)をサポート	
	WebJobs SDK Extensionをサポート	
	local開発・デバッグが可能	
	WebJobsのダッシュボードが使用可能	

実際の開発

ここでAzure Functionsのアプリケーションを作ってみましょう。今回Azure Functionsで作るのは、サムネイル画像のリサイズジングAPIです。
モノリシックなアプリケーションでは一つのアプリケーション上に多くのプログラムがのっています。開発が進んでいく

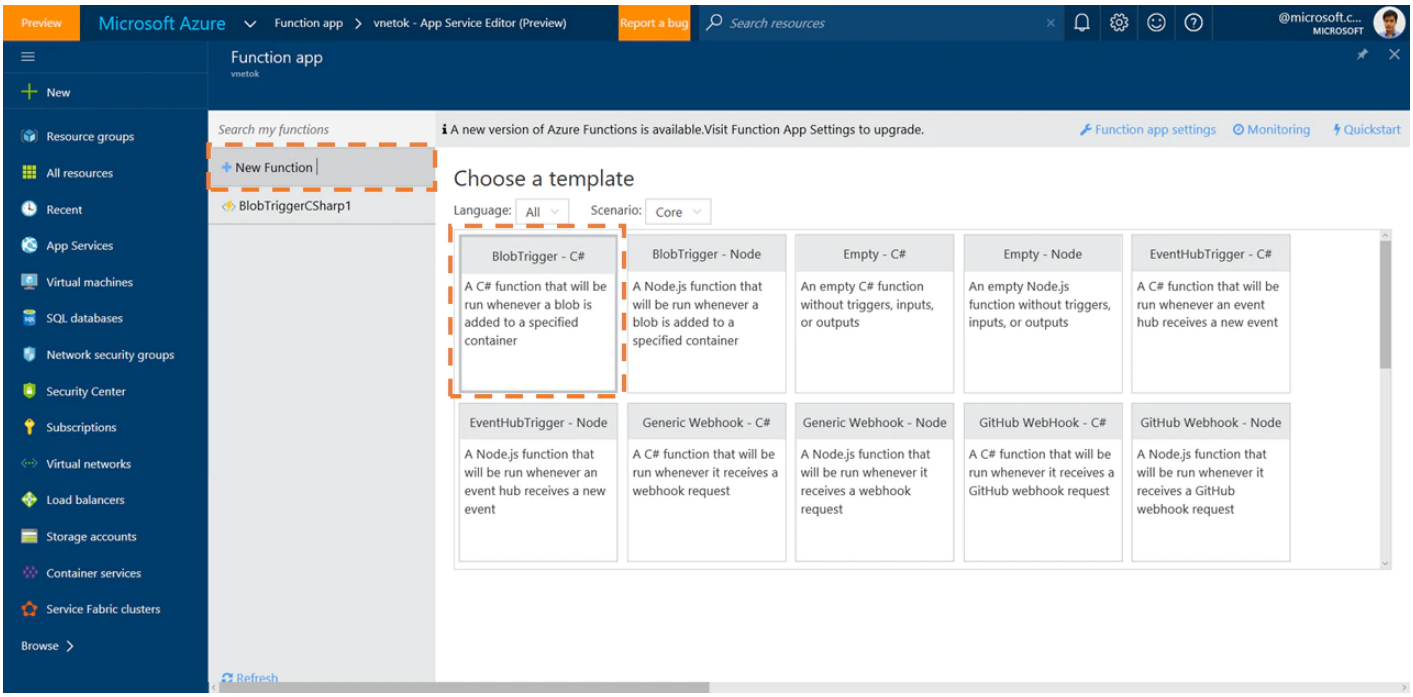
とアプリケーションの中には雑多な小さな機能が多く存在していることに気づくでしょう。　しかし、そこで振り返ってみたときに、例えば「画像のリサイズ」は本当にサーバーで行うべき処理でしょうか。機能が膨らんでアプリーションの管理が煩雑になっているなら、その一部の処理をアウトソースする選択肢も考えてみましょう。
そこで今回はサムネイル画像をリサイズする機能をAzure Functionsにアウトソースすることにしてみます。今回のプログラムではマイクロソフトが出しているCognitive APIの一機能をつかって画像のリサイジングを行います。

ここで一つの疑問がわきます。
「あれ、結局Cognitivce ServiceのAPIをたたくのなら、システムからそこにアクセスすればいいだけの話じゃない？」
しかし、ここで考えてほしいメリットは、Azure FunctionsがAPIとして機能する事だけではなく、今回の目的もCognitive ServiceのAPIをラッピングすることではありません。

今回体験していただきたいのは、「画像がBlobストレージの特定のフォルダにアップロードされたら、それをトリガーにBlobストレージの別フォルダにリサイジングした画像をアッ

プロードする」というトリガーの部分です。

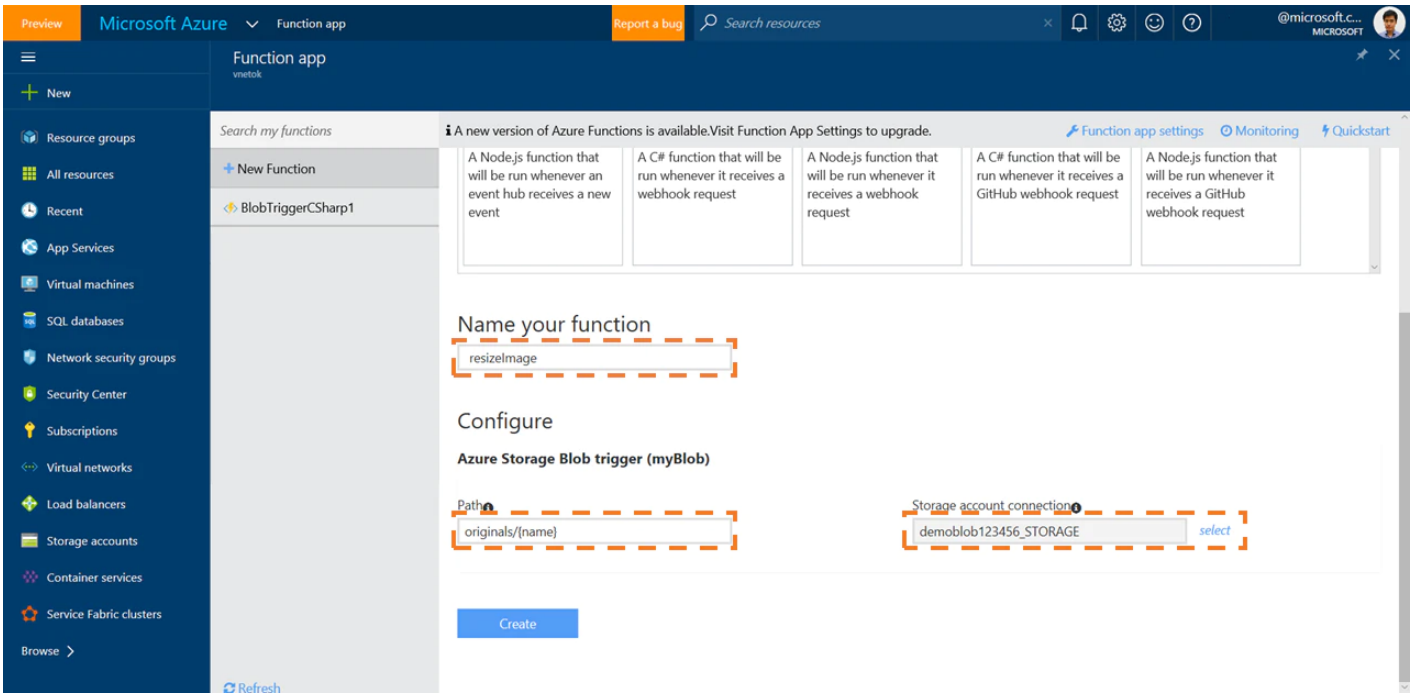
まず、新しいFunctionを作ってみましょう



今回はBlobにファイルがアップロードされると、イベントを発火させるFunction を作成してみます。例えば「BlobTrigger – C#」を選んでみましょう。　例えば、もしあなたがJavascript使いであれば、「BlobTrigger – Node」を選んで進

めてみましょう。このテンプレートを選ぶと、関数のベースとなるコードが追加されたファイルが作成されます。もしも1からプログラムを書きたい場合はEmptyのプログラムを選びましょう。

続いてFunctionの設定をしていきます。



最初にするのは、先ほど選んだ「BlobTrigger」の詳細設定です。
具体的にこのときに考えることは以下です。

- Functionsの名前
- どのBlobストレージアカウントに接続するか
- どのフォルダにファイルが追加されたら発火するか

もうこれで基本的な設定は終了です。

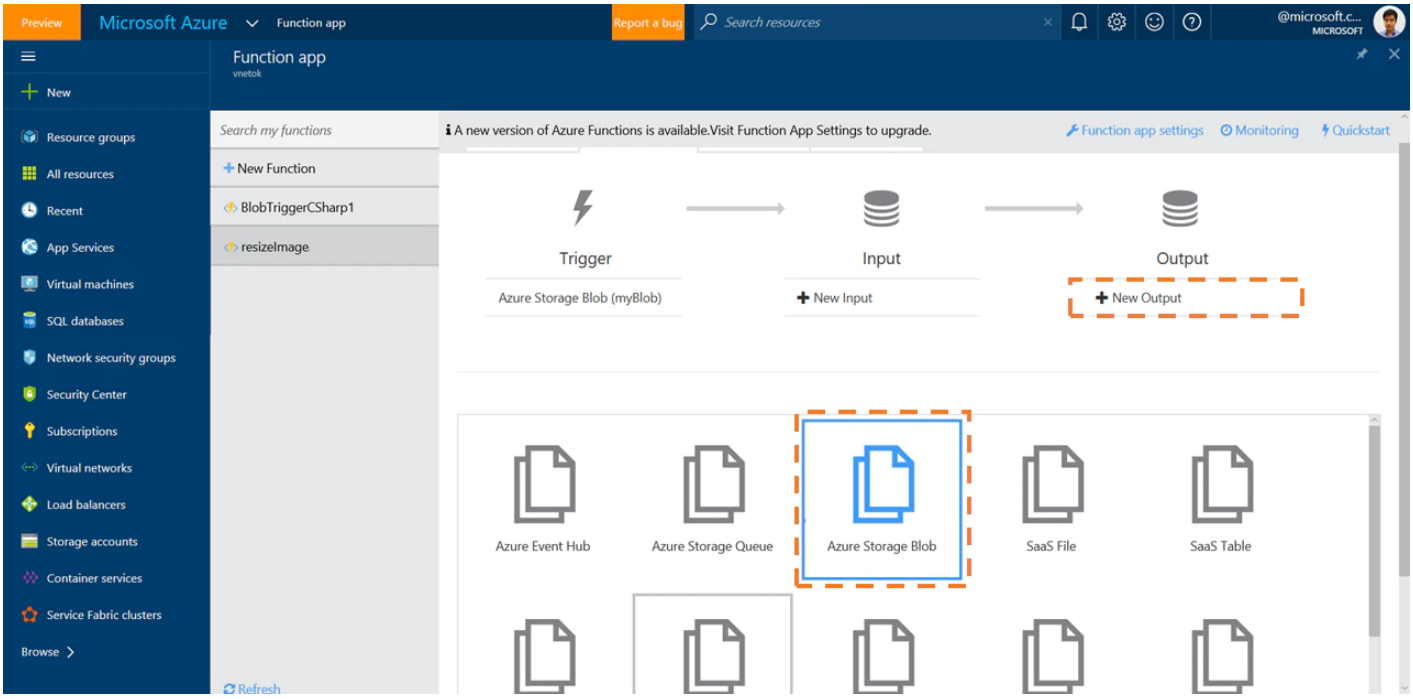
次にアウトプットの設定です。

続いて処理の結果をどのように返すのか決めていきましょう。Functionsで加工した後のデータをどこにアウトプットするのかを設定します。

なおFunctionsでは処理の返り値として一つの処理を返すというわけではなく、1つの処理に対して複数のインプットとアウトプットを持つことが出来ます。Azure Functionsではシリア

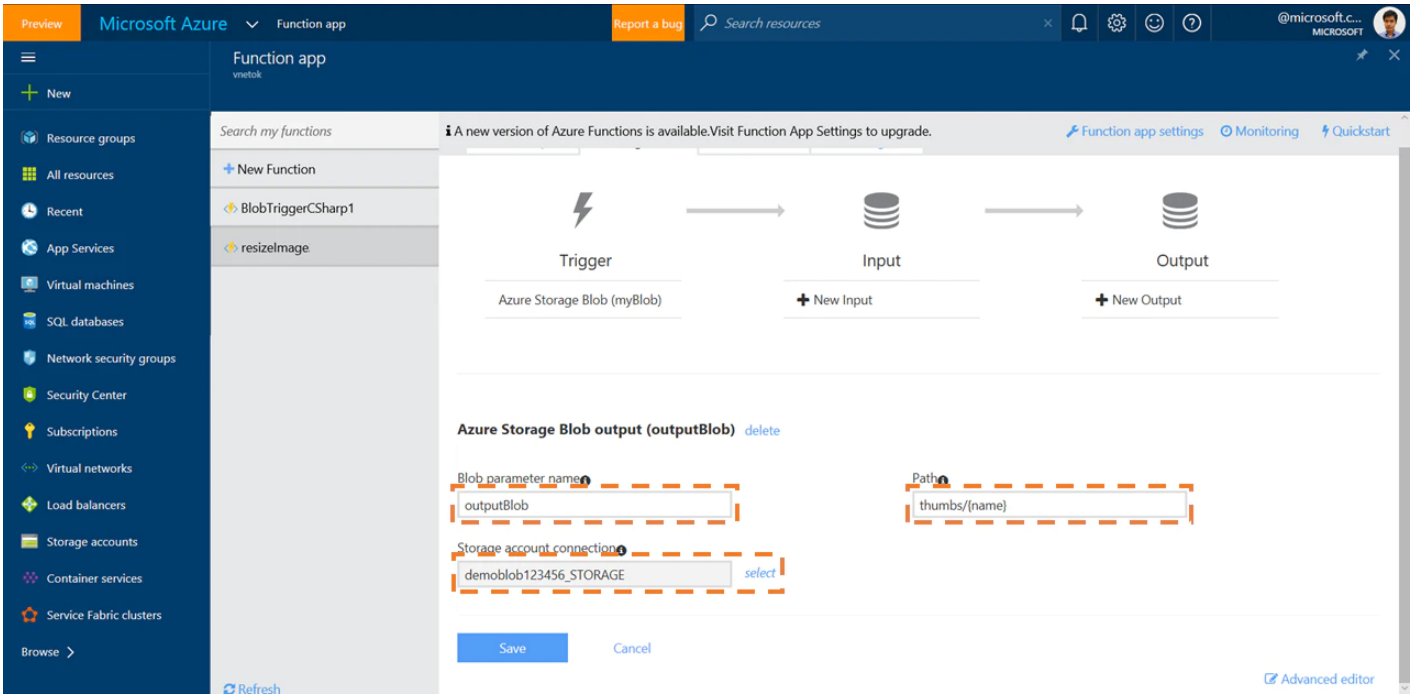
ル化と逆シリアル化が自動的にされる仕組みがあるため、明示的にFunction App の関数の最後の行にreturn BlobObjectと書くのではなく、例えばバインディングの設定で引数にOutputBlobというオブジェクトを渡した場合に、コードの途中にOutputBlob = blob2と書くだけで処理のアウトプットがされたということになります。

しかし、ここで注意したいのがAzure Functionsの設計理念です。主に1関数あたり1つの機能というのがAzure Functionsでの関数のベストプラクティスになっており、また関数には冪等性が求められます。関数を作る際には複数の機能を持つ「なんでも関数」を作らずに「1つの機能を持つ汎用的な関数」となるように心がけましょう。



続いてOutputの詳細な設定をしていきましょう。必要な設定は以下です。

- バインディングで接続し、オブジェクトを関数に渡す時に利用する引数名
- どのBlobストレージアカウントに接続するか
- どのフォルダにファイルを追加するか



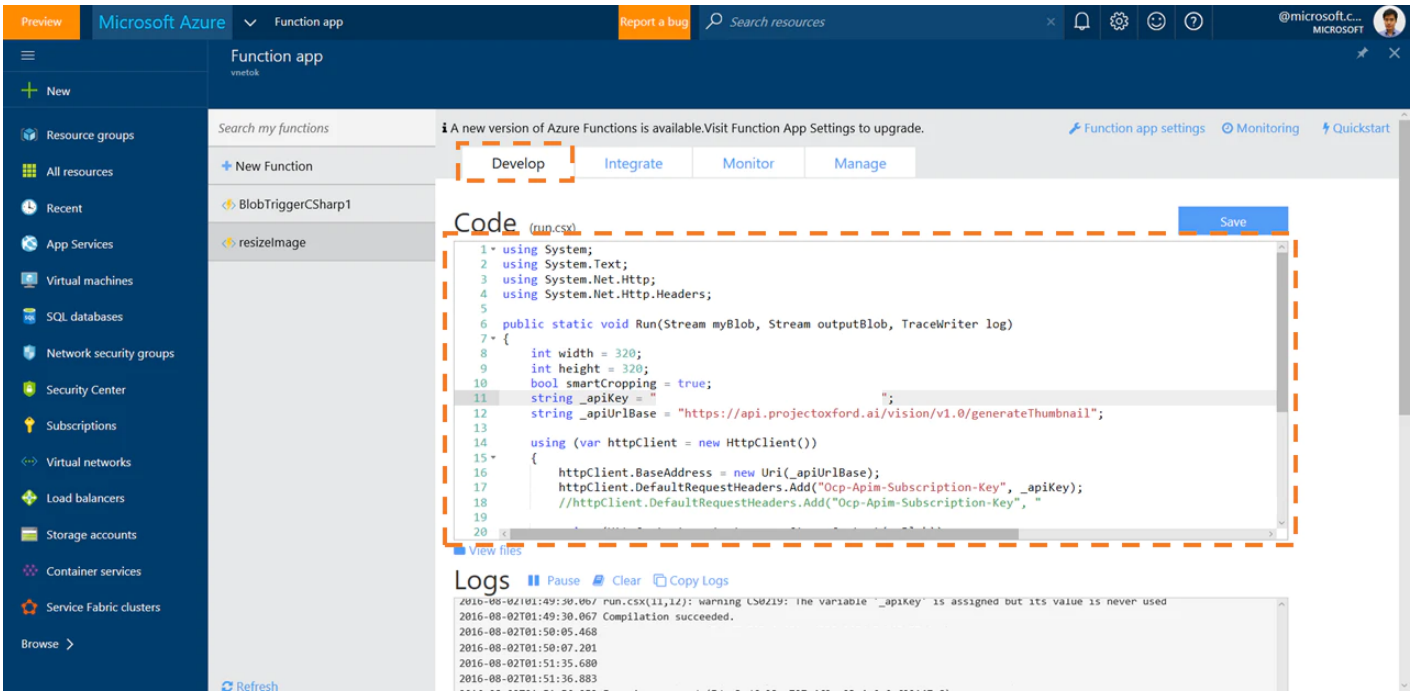
今回の流れを整理すると以下の様になります。

1. 特定のストレージにファイルが追加されたというイベントを検知してイベント（Function）を発火
2. 追加されたファイルはmyBlobという変数の中にStringの型で追加され、関数の引数で渡される。（シリアル化）　ファンクションの実行にあたってはoutputBlobという名前でアウトプットバインディングの変数も引数に渡される。
3. 関数の処理が実行される。
4. 処理された結果、変換された値をStringとしてoutputBlob変数に代入（逆シリアル化）

実際にコードを書いていくとさらに理解しやすいので、書いていきましょう。

最後に、コードを書いていきましょう。

もう準備は完了です。さっそくコードを書き始めましょう。



run.csx

```
using System;
using System.Text;
using System.Net.Http;
using System.Net.Http.Headers;

public static void Run(Stream myBlob, Stream outputBlob, TraceI
{
    int width = 320;
    int height = 320;
    bool smartCropping = true;
    string _apiKey = "YOUR_KEY";
    string _apiUrlBase = "https://api.projectoxford.ai/vision/
    string _subscriptionKey = "YOUR_KEY";

    using (var httpClient = new HttpClient())
    {
        httpClient.BaseAddress = new Uri(_apiUrlBase);
        httpClient.DefaultRequestHeaders.Add("Ocp-Apim-Subscri

        using (HttpContent content = new StreamContent(myBlob)
        {
            //get response
            content.Headers.ContentType = new MediaTypeWithQua
            var uri = $"{_apiUrlBase}?width={width}&height={he
            var response = httpClient.PostAsync(uri, content).
            var responseBytes = response.Content.ReadAsByteArr

            //write to output thumb
            outputBlob.Write(responseBytes, 0, responseBytes.L
```

```
            }
        }
    }
```

それでは試しにBlobにファイルをアップロードしてみましょう。
そうするとリサイズされた画像がthumb/outputに入るとわかるでしょう。

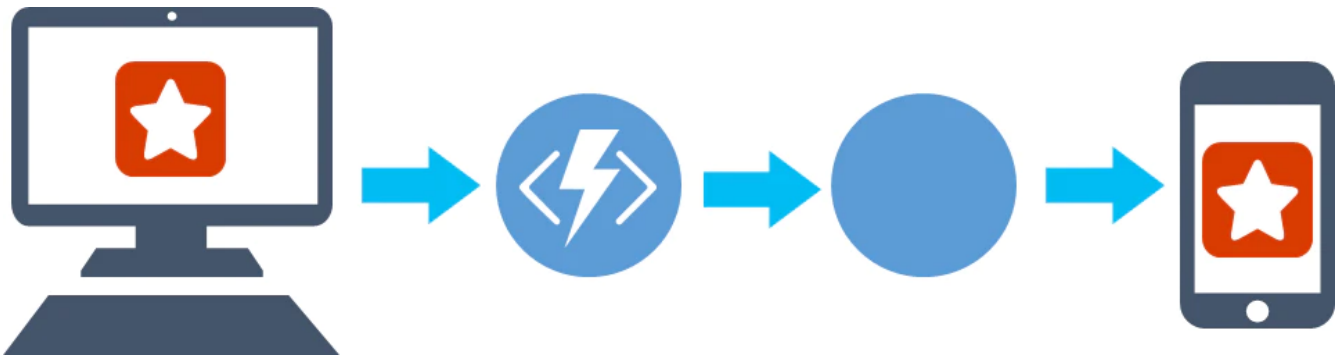
Azure Functions利用シーン

上ではAzure Functionsの利用方法のうち、画像リサイジングの機能を提供するサンプルを紹介しました。
ではどのようにAzure Functionsを始めればよいのでしょうか。その方法としてまずは、「小さく始める」ということがおすすめです。既存サービスの 1機能を置き換えてみましょう、そしてどちらかと言うとコアの機能ではない機能をアウトソーシングしてみましょう。

はじめかた=小さく始める

まずは、既存のサービスの１機能（APIやバックグラウンドプロセス）を置き換えてみる
既存の環境で、特定の用途にしか使わない機能をアウトソースしてみる
例としては以下の様利用例があります。

- 毎夜で日付の古い Blob を Cool Blob Storage に移動
- Logic Apps や Microsoft Flow 単体では難しい処理を実行
- ログの整形やスクレイピング
- Notification Hubと合わせたPush API



たとえば、Notification Hubの場合、プッシュ通知を出すときは基本的にオンデマンドの時が多いです。しかしその機能のためだけにサーバーをいつも起動させておくのはもったいな

いです。そうした時に functions サーバーを特定の時だけ起動させると、運用時のコストが下がる上に、その部分だけアウトソースにするので、コード本体の見栄えもよくなります。

利用法の注意

ここで注意したいのが、処理の内容です。　基本的にはこうした処理を大きな処理に対してやるのは Azure Functions の使い方としては良くありません。例えばバッチ処理で1回実行したら裏で3時間も処理を続けているという処理は Azure Functions には合いません。
上の例では「ログの整形やスクレイピング」がありますが、Azure Functionsは例えば1年分の大量のログデータを整形したりスクレイピングするのは不向きだといえます。どちらかというと、1日分の少しのログデータを対象に整形やスクレイピングするのが良いでしょう。下記のポイントをおさえてAzure Functionsを実装にすると、システム全体としても見通しが良くなるでしょう。

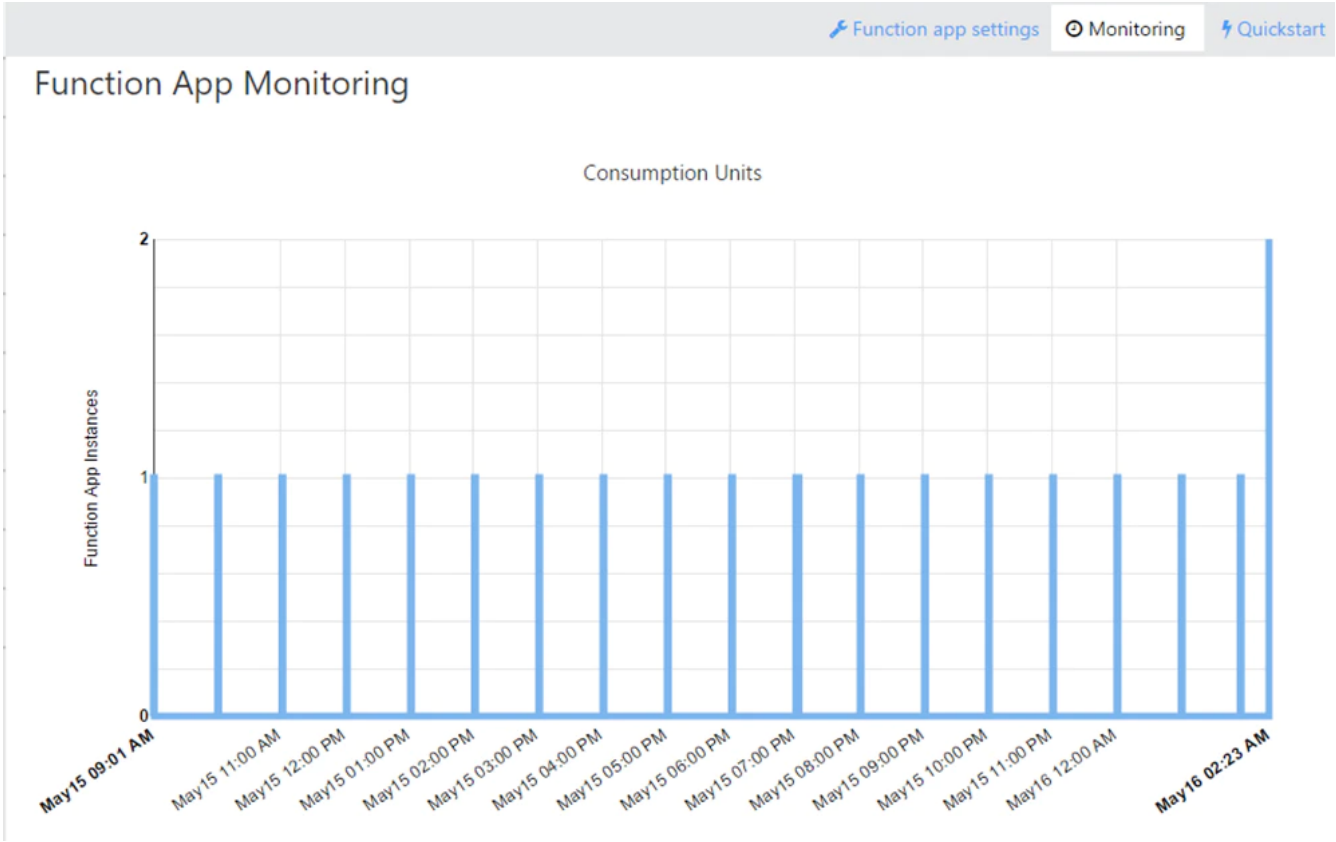
利用方法のポイント

- １関数につき、「１個の処理」を定義する
- 何回処理を実行しても同じ結果が返る
- 処理はできるだけ早く終わらせる

Azure Functionsの運用

運用が簡単ということはAzure Functionsの強みです。例えばスケールイン/アウトは GUIで上限値を設定するだけです。
モニタリングも簡単で、従来のApplication Insight、Audit Logsなども用いることが出来ます。
スケールの特徴
スケールの特徴は以下になります。スケールの形式はDynamic Service PlanとApp Service Planで異なるため、注意が必要です。

- スケール粒度は Function App 単位
- 0～10 （2017/08/10時点）でインスタンスが自動的に増減（Dynamic Service Plan 時のみ ）
- 構成されているトリガーに基づきトラフィック等から判断
- インスタンス内のFunction App 同時実行可能数はメモリ設定によって異なる



ログの特徴

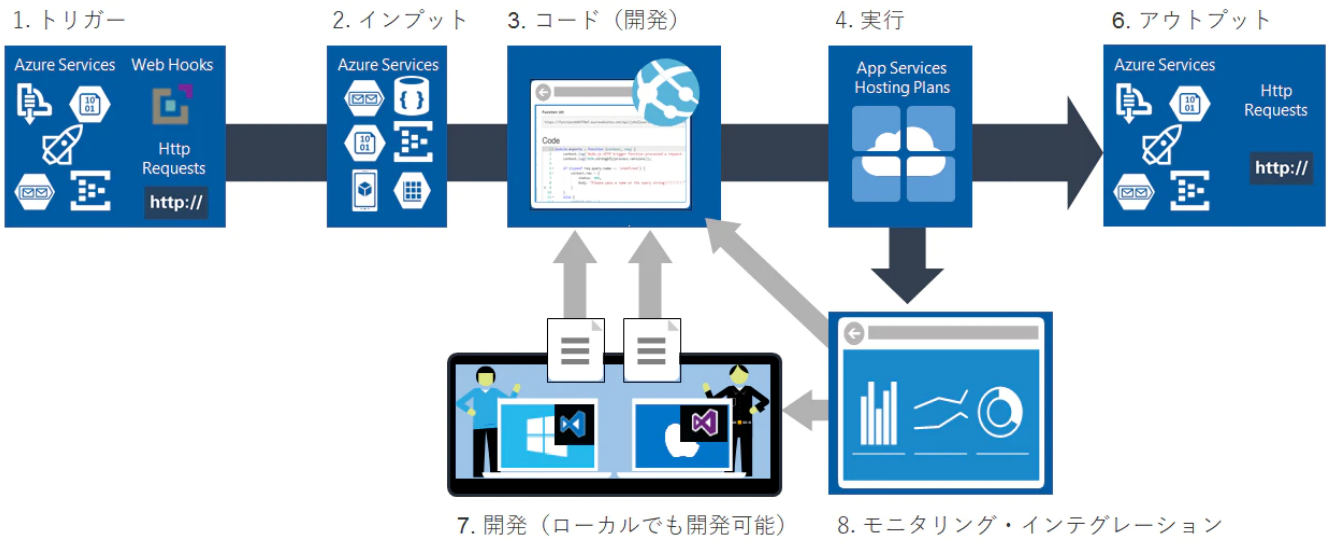
ログ監査はAzure Application Insight等で従来通り行うことが出来ます。また、ログはAzure Table上のAzureFunctionsLogTableコンテナに、バインドされている値の詳細や出力された値などが出力されます。

デプロイとインテグレーション

デプロイもとても簡単です。代表的なデプロイ方法が以下です。あなたがgithubユーザーだったら普段リポジトリにpushするように開発すればよいのです。

- SCM を使用したデプロイ/継続的インテグレーション
- GitHub や Bitbucket など Web Apps と同様に設定可能
- FTP経由や WebDeploy を使用したデプロイ
- Azure File Storage を SMB で直接マウントできる環境の場合、直接ファイルをコピーすることも可能
- Azure Resource Manager を使用したデプロイ

- Quick start templateも利用可能（https://github.com/Azure/azure-quickstart-templates/tree/master/101-function-app-create-dynamic）　これらのインテグレーションをまとめると以下の様になります。



料金

料金プランは2つあります。とくにDnamic Service PlanはAzure Functionsの実行環境に最も適したものになっています。

App Service Plan

- Basic / Standard / Premiumの階層
- VMの利用時間に合わせた料金
- スケールはVMの設定に基づく

Dynamic Service Plan

- 実行した回数に基づく課金
- 予約済みメモリ x 実行時間
- スケールはFunctionsプラットフォームが自動で行う

詳細情報

「Functions の価格」https://azure.microsoft.com/ja-jp/pricing/details/functions/

Azure Functions、小さく始めてみましょう

いかがでしょうか。コード以外の部分がすべてサービスとして提供されるAzure Functions、サーバーレスアーキテクチャやマイクロサービスアーキテクチャはこれからの技術でしたが、その素晴らしさの片りんを見ることができたのではないでしょうか。
まずは業務アプリの1機能、自分が毎回やっているルーティーン等、小さい部分からサーバーレス化を進めてみて、感触を確かめてみましょう。