

メシウス株式会社

Azure Functions と DioDocs で Excel や PDF ファイルを出力する

2024 年 7 月 29 日

※本資料は、[弊社ブログ](#)に投稿された記事「[Azure Functions と DioDocs で Excel や PDF ファイルを出力する \(1\) ~ \(3\)](#)」の連載記事をまとめて資料化した内容となります。下記が記事の原文となります。

[Azure Functions と DioDocs で Excel や PDF ファイルを出力する \(1\)](#)

[Azure Functions と DioDocs で Excel や PDF ファイルを出力する \(2\)](#)

[Azure Functions と DioDocs で Excel や PDF ファイルを出力する \(3\)](#)

目次

Azure Functions と DioDocs で Excel や PDF ファイルを出力する (1)	2
Azure Functions とは	2
実装する内容	2
アプリケーションを作成	2
NuGet パッケージの追加	4
DioDocs for Excel を使うコードを追加	4
DioDocs for PDF を使う関数を追加	5
DioDocs for PDF を使うコードを追加	7
デバッグ実行で確認	8
Azure ヘデプロイ	10
デプロイしたアプリケーションを確認	14
さいごに	15
Azure Functions と DioDocs で Excel や PDF ファイルを出力する (2)	16
実装する内容	16
アプリケーションを作成	16
NuGet パッケージの追加	18
Azure Blob Storage を使うコードを追加	19
DioDocs for Excel を使うコードを追加	20
DioDocs for PDF を使う関数を追加	21
Azure Blob Storage を使うコードを追加	22
DioDocs for PDF を使うコードを追加	23
デバッグ実行で確認	24
Azure ヘデプロイ	26
デプロイしたアプリケーションを確認	30
さいごに	32
Azure Functions と DioDocs で Excel や PDF ファイルを出力する (3)	33
セルに追加するテキストの日本語フォント (DioDocs for Excel)	33
ワークシートを PDF 出力する際の日本語フォント (DioDocs for Excel)	34
PDF ドキュメントを保存する際の日本語フォント (DioDocs for PDF)	38

Azure Functions と DioDocs で Excel や PDF ファイルを出力する (1)

本記事では、Azure Functions で「[DioDocs \(ディオドック\)](#)」を使用した C# (.NET 8) のクラスライブラリをベースにした関数を作成し、Excel や PDF ファイルを出力する方法について紹介します。

Azure Functions とは

[Azure Functions](#) は [Microsoft Azure](#) で提供されている、各種イベントをトリガーに処理を実行するサーバーレスなアプリケーションを作成できるクラウドサービスです。今回は [Visual Studio 2022](#) で Azure Functions アプリケーションを作成し、Azure へデプロイして確認します。

実装する内容

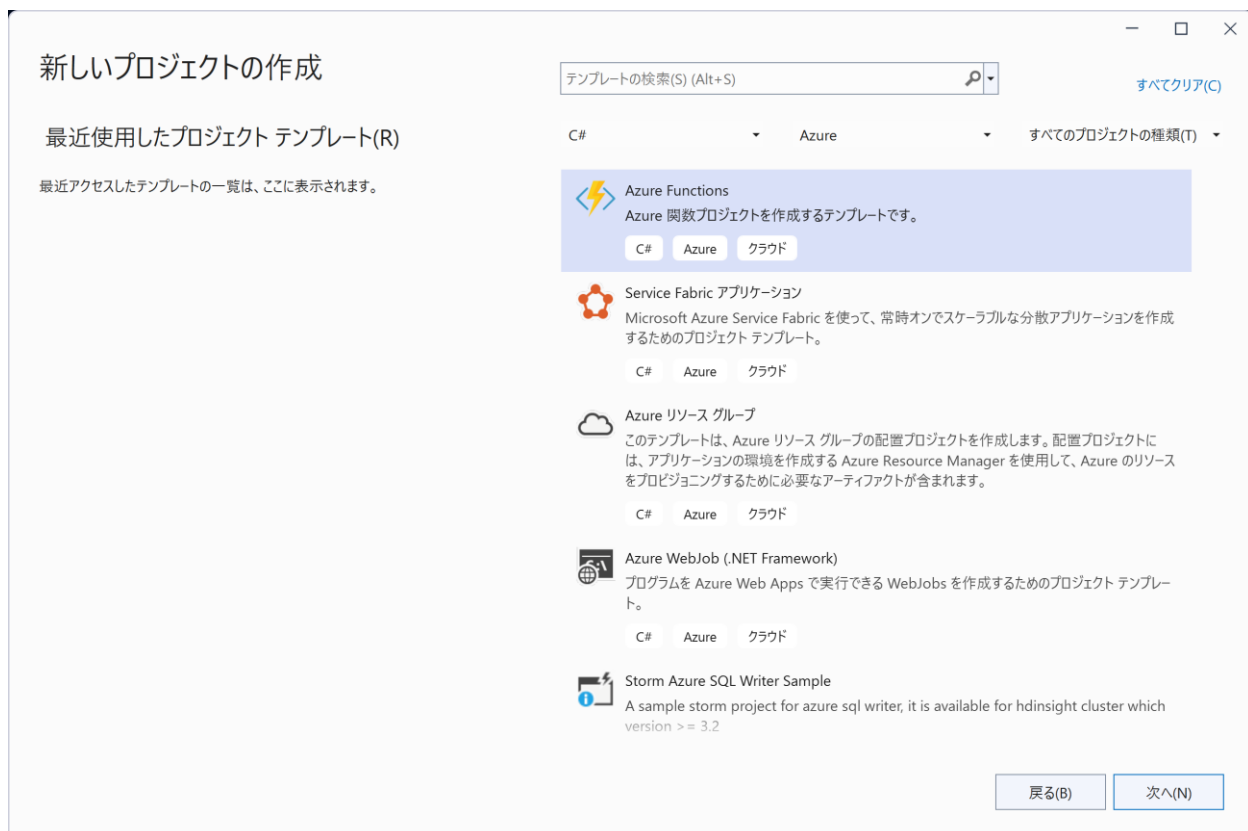
今回実装する内容は非常にシンプルです。Azure Functions アプリケーションで [HTTP トリガー](#) を使用する関数を作成します。関数の実行時に DioDocs を使用して Excel と PDF ファイルを作成し、クエリパラメータで受け取った文字列をそれぞれのファイルへ追加します。その後、作成した Excel と PDF ファイルを [FileContentResult](#) で直接ローカルへ出力する、といった内容です。

アプリケーションを作成

以下のドキュメントを参考に Azure Functions アプリケーションを作成していきます。

[クイック スタート: Visual Studio を使用して Azure で初めての C# 関数を作成する](#)

Visual Studio 2022 でプロジェクトテンプレート「Azure Functions」を選択して [次へ] をクリックします。



プロジェクト名 `DioDocsFileExportFunctionApp` を入力して [次へ] をクリックします。

新しいプロジェクトを構成します

Azure Functions C# Azure クラウド

プロジェクト名(I)

DioDocsFileExportFunctionApp

場所(L)

C:\Users\%kuni\Desktop

ソリューション名(M) ⓘ

DioDocsFileExportFunctionApp

ソリューションとプロジェクトを同じディレクトリに配置する(D)

プロジェクトは "C:\Users\%kuni\Desktop\DioDocsFileExportFunctionApp\DioDocsFileExportFunctionApp" で作成されます

戻る(B) 次へ(N)

Azure Functions で作成する関数のテンプレートを選択します。 `Http Trigger` を選択して [作成] をクリックします。

追加情報

Azure Functions C# Azure クラウド

Functions worker ⓘ

.NET 8.0 (長期的なサポート)

Function ⓘ

Http trigger

ランタイム ストレージ アカウントに Azurite を使用する (AzureWebJobsStorage) ⓘ

コンテナのサポートを有効にする ⓘ

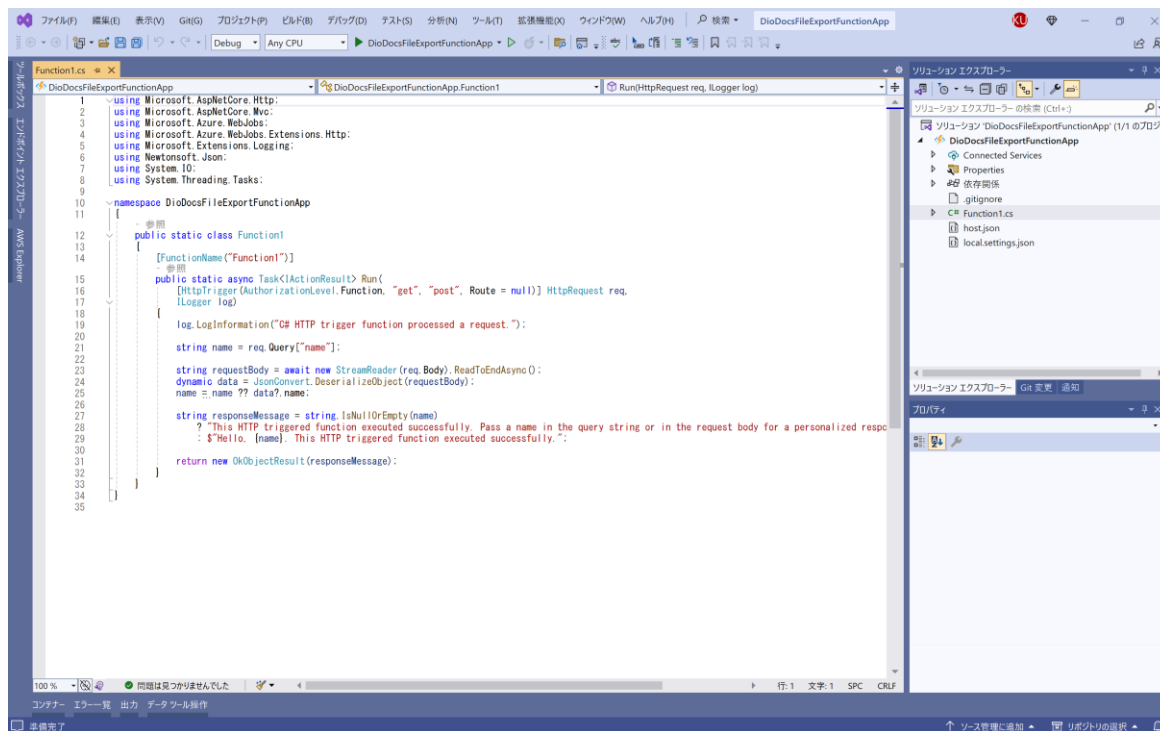
Authorization level ⓘ

Function

.NET Aspire オークストレーションへの参加 ⓘ

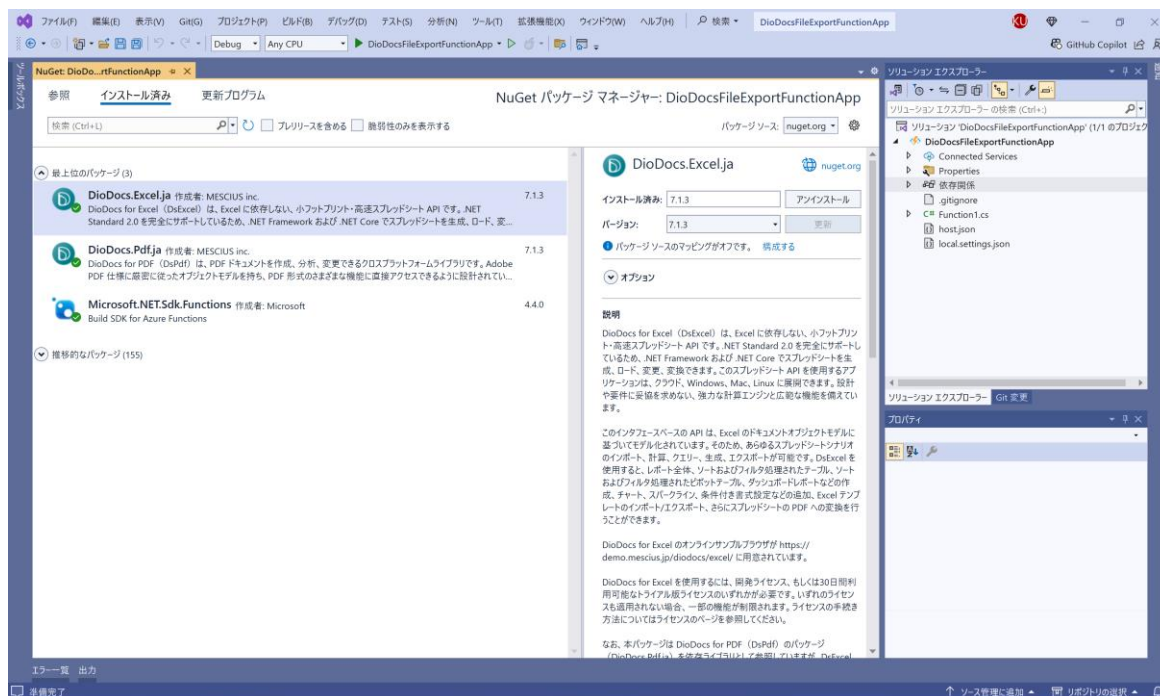
戻る(B) 作成(C)

DioDocsFileExportFunctionApp プロジェクトが作成されます。



NuGet パッケージの追加

Visual Studio の「NuGet パッケージ マネージャー」から DioDocs のパッケージ DioDocs.Excel.ja、DioDocs.Pdf.ja をインストールします。



DioDocs for Excel を使うコードを追加

DioDocs で Excel ファイルを作成するコードを追加して Function1 を以下のように更新します。

```

public static class Function1
{
    [FunctionName("Function1")]
    public static async Task<IActionResult> Run(
        [HttpTrigger(AuthorizationLevel.Function, "get", "post", Route = null)] HttpRequest req,
        ILogger log)
    {
        log.LogInformation("C# HTTP trigger function processed a request.");

        string name = req.Query["name"];

        string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
        dynamic data = JsonConvert.DeserializeObject(requestBody);
        name = name ?? data?.name;

        string Message = string.IsNullOrEmpty(name)
            ? "Hello, World!!"
            : $"Hello, {name}!!";

        Workbook workbook = new Workbook();
        workbook.Worksheets[0].Range["A1"].Value = Message;

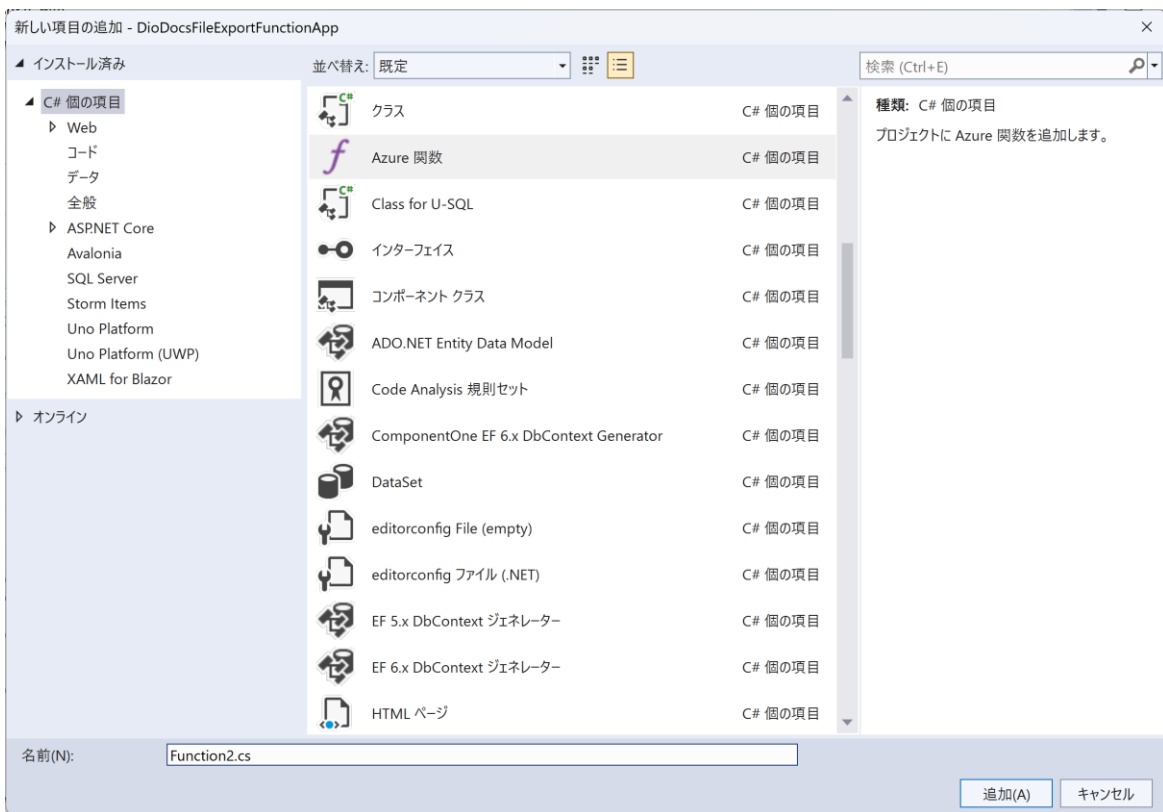
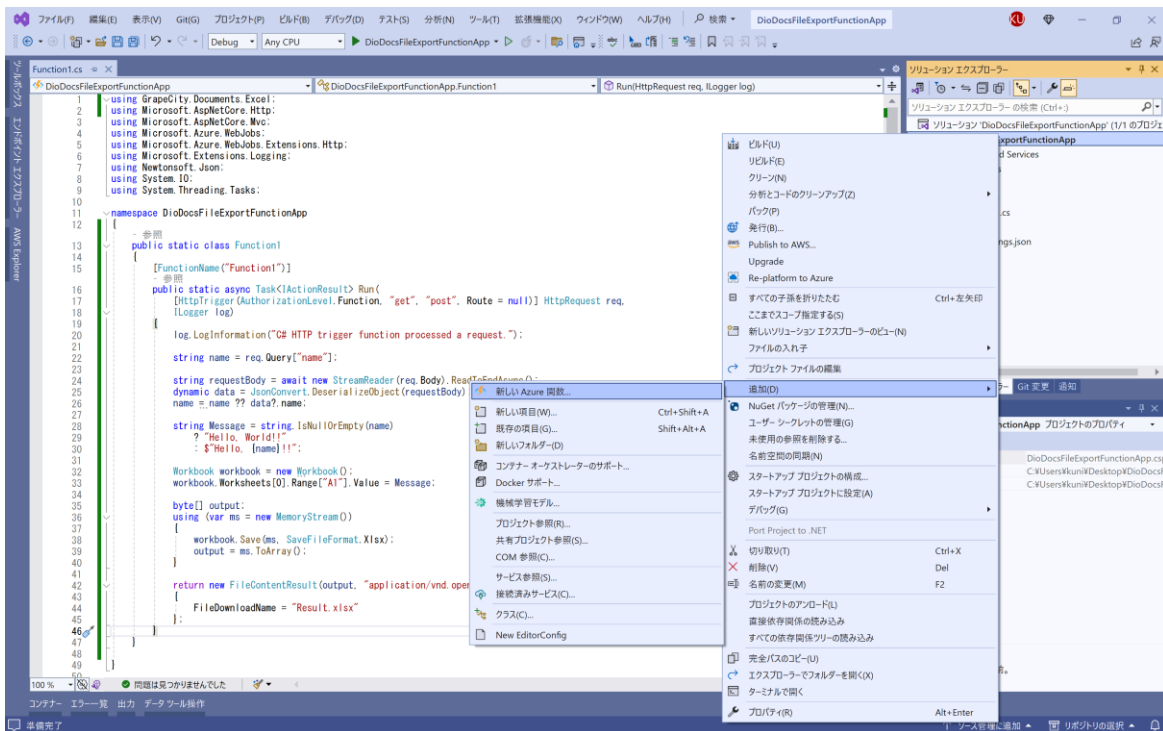
        byte[] output;
        using (var ms = new MemoryStream())
        {
            workbook.Save(ms, SaveFileFormat.Xlsx);
            output = ms.ToArray();
        }

        return new FileContentResult(output, "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet")
        {
            FileNameDownloadName = "Result.xlsx"
        };
    }
}

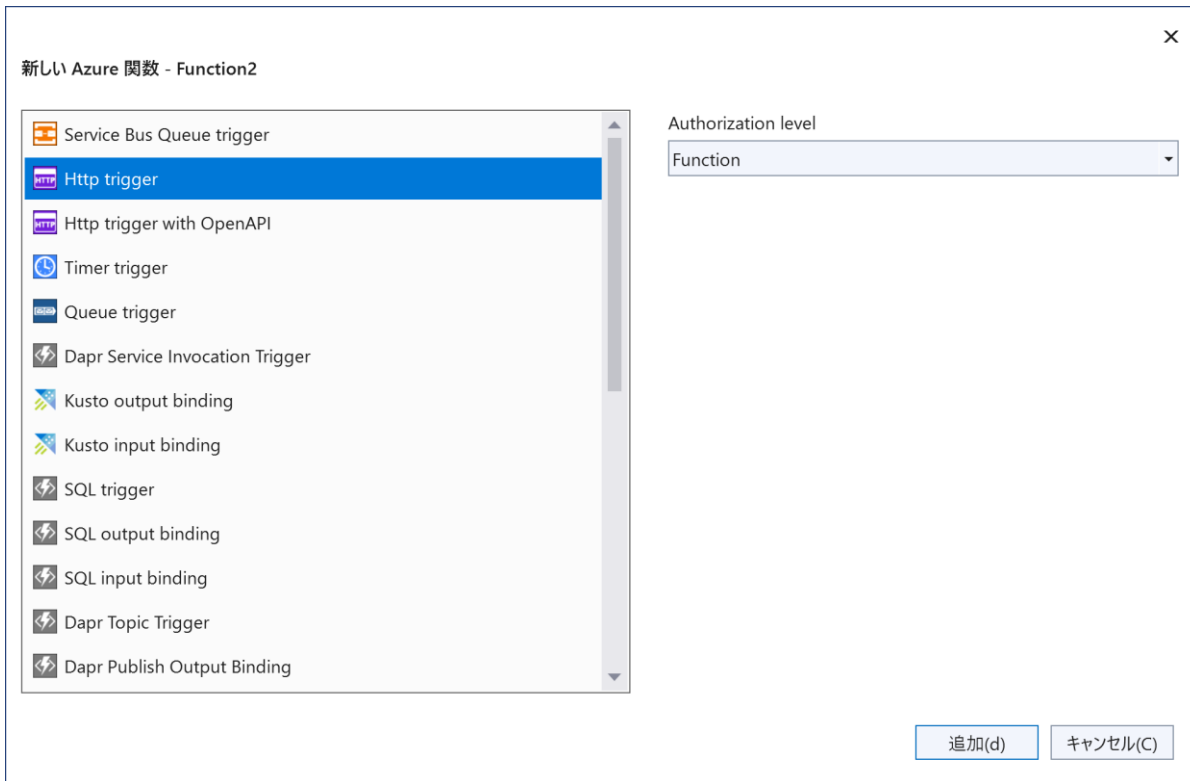
```

DioDocs for PDF を使う関数を追加

ソリューションエクスプローラーから `DioDocsFileExportFunctionApp` プロジェクトを右クリックして[追加]
 - [新しい Azure 関数] を選択して、DioDocs で PDF ファイルを作成する関数 `Function2` を追加します。



関数のテンプレートを選択します。Http Trigger を選択して [追加] をクリックします。



DioDocs for PDF を使うコードを追加

DioDocs で PDF ファイルを作成するコードを追加して Function2 を以下のように更新します。

```
public static class Function2
{
    [FunctionName("Function2")]
    public static async Task<IActionResult> Run(
        [HttpTrigger(AuthorizationLevel.Function, "get", "post", Route = null)] HttpRequest req,
        ILogger log)
    {
        log.LogInformation("C# HTTP trigger function processed a request.");

        string name = req.Query["name"];

        string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
        dynamic data = JsonConvert.DeserializeObject(requestBody);
        name = name ?? data?.name;

        string Message = string.IsNullOrEmpty(name)
            ? "Hello, World!!"
            : $"Hello, {name}!!";

        GcPdfDocument doc = new GcPdfDocument();
        GcPdfGraphics g = doc.NewPage().Graphics;
```



```
g.DrawString(Message,
    new TextFormat() { Font = StandardFonts.Helvetica, FontSize = 12 },
    new PointF(72, 72));

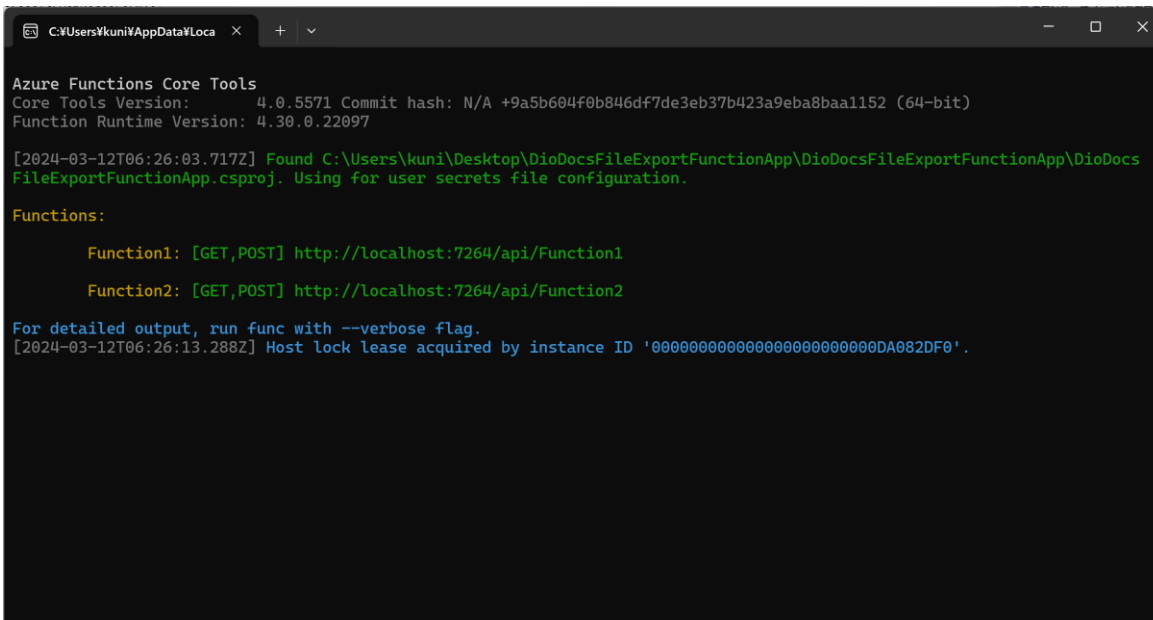
byte[] output;

using (var ms = new MemoryStream())
{
    doc.Save(ms, false);
    output = ms.ToArray();
}

return new FileContentResult(output, "application/pdf")
{
    FileNameDownloadName = "Result.pdf"
};
}
```

デバッグ実行で確認

作成した Azure Functions アプリケーションをローカルでデバッグ実行して確認します。Visual Studio からデバッグ実行すると以下のコンソールが表示されます。



```
C:\Users\kuni\AppData\Local...
Azure Functions Core Tools
Core Tools Version: 4.0.5571 Commit hash: N/A +9a5b604f0b846df7de3eb37b423a9eba8baa1152 (64-bit)
Function Runtime Version: 4.30.0.22097

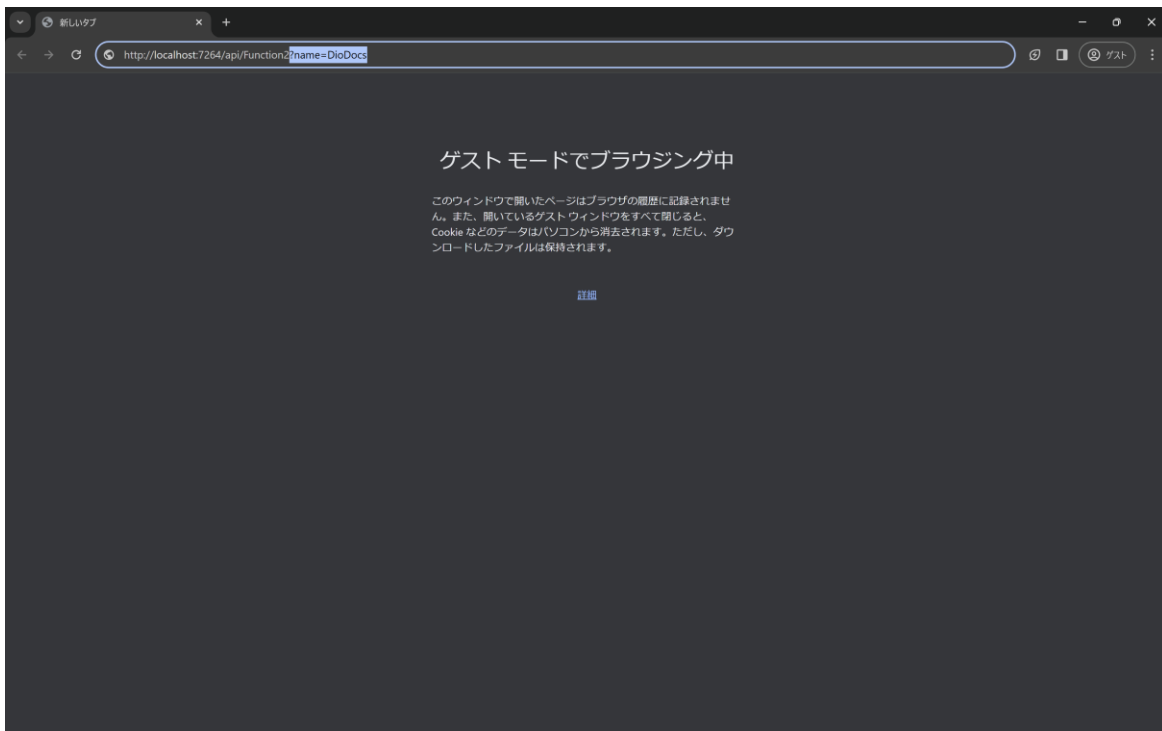
[2024-03-12T06:26:03.717Z] Found C:\Users\kuni\Desktop\DioDocsFileExportFunctionApp\DioDocsFileExportFunctionApp\DioDocsFileExportFunctionApp.csproj. Using for user secrets file configuration.

Functions:

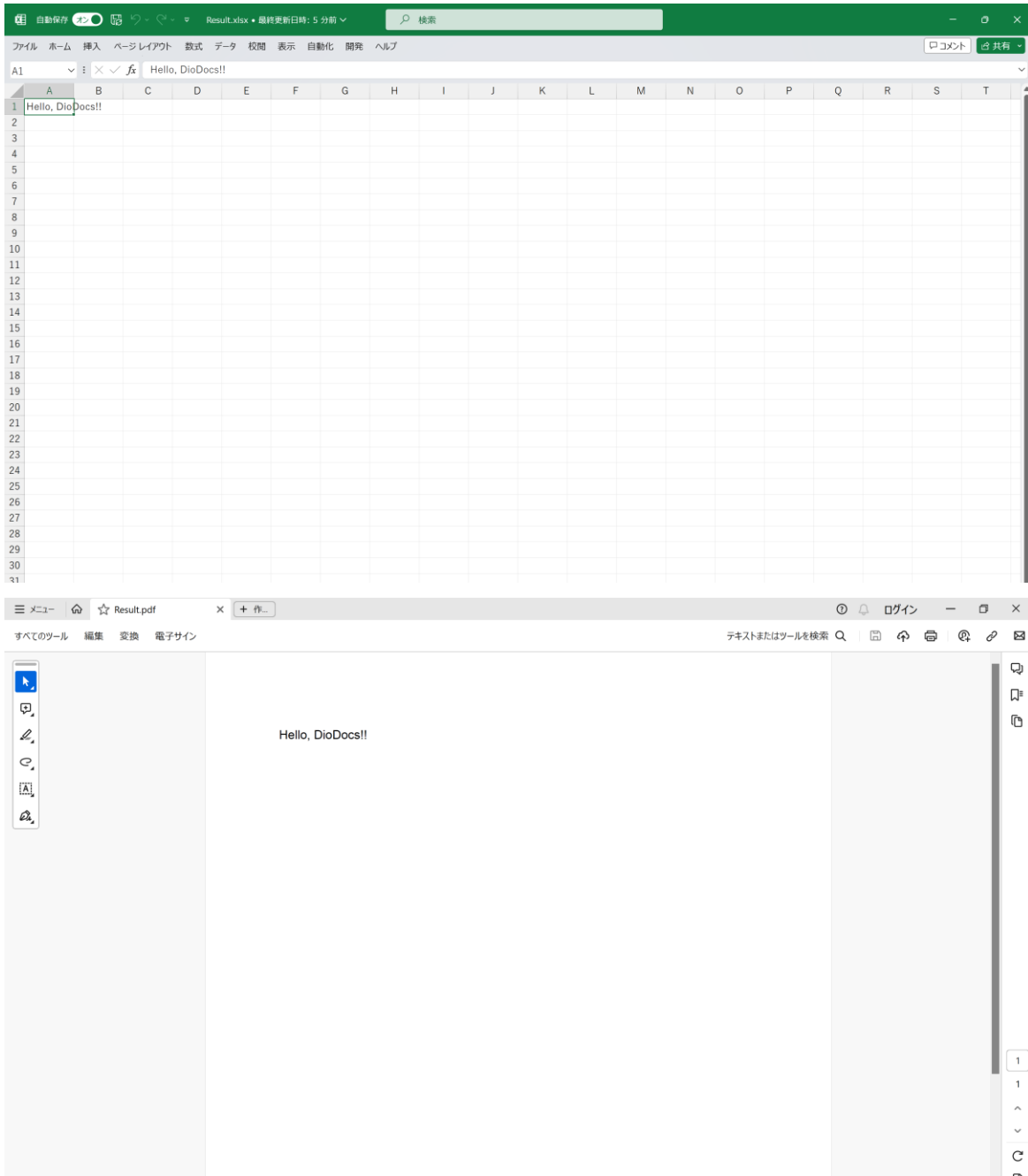
    Function1: [GET,POST] http://localhost:7264/api/Function1
    Function2: [GET,POST] http://localhost:7264/api/Function2

For detailed output, run func with --verbose flag.
[2024-03-12T06:26:13.288Z] Host lock lease acquired by instance ID '000000000000000000000000DA082DF0'.
```

アプリケーションに含まれる関数の URL は `http://localhost:7264/api/Function1`、`http://localhost:7264/api/Function2` となっています。この URL にクエリパラメータと文字列 `?name=DioDocs` を追加して、それぞれの関数をブラウザで実行します。

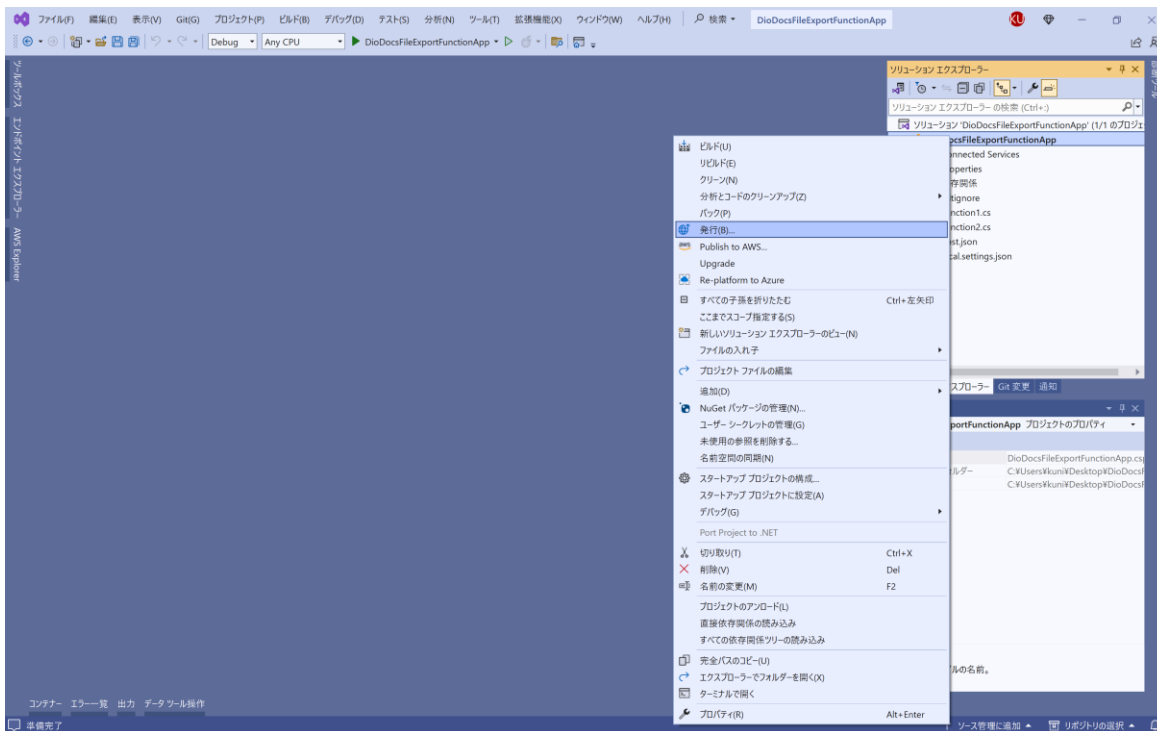


ローカルのフォルダに保存された `Result.xlsx`、`Result.pdf` を確認します。クエリパラメータで渡した文字列 `DioDocs` が表示されていれば成功です。

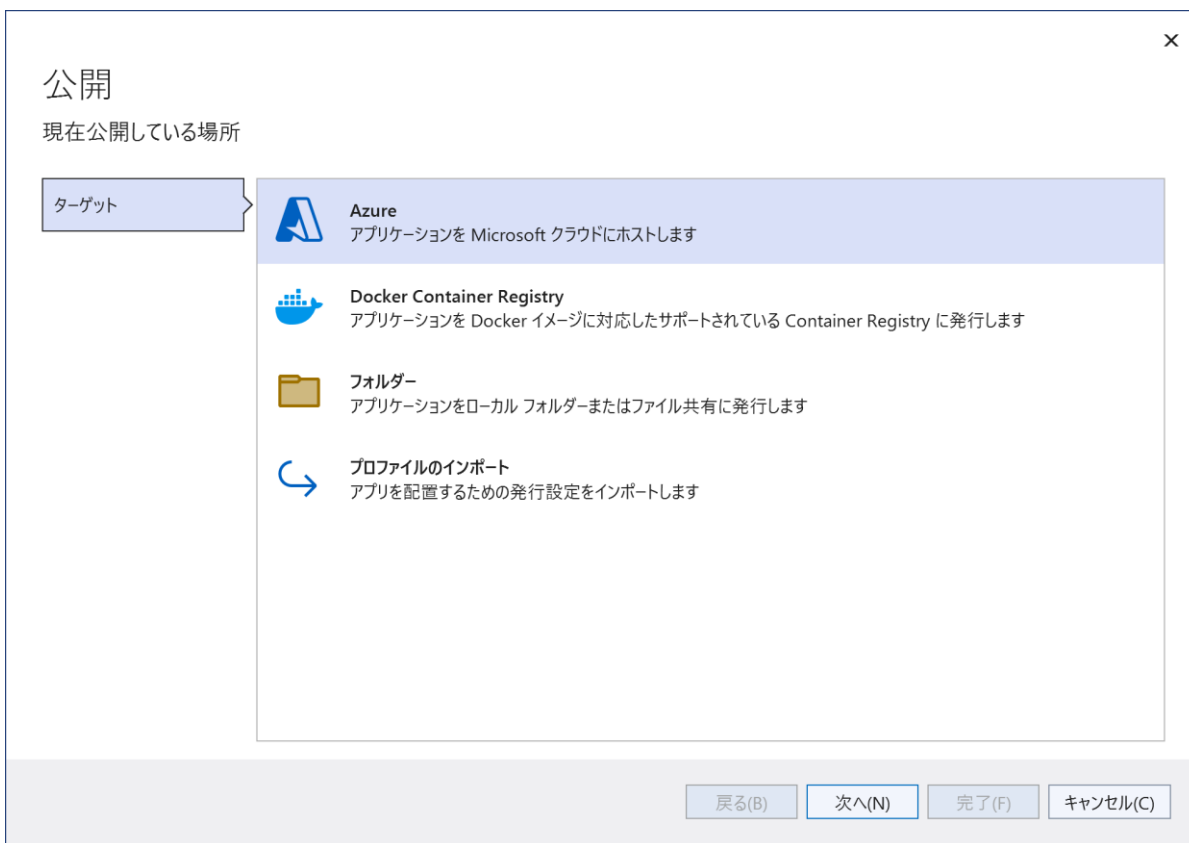


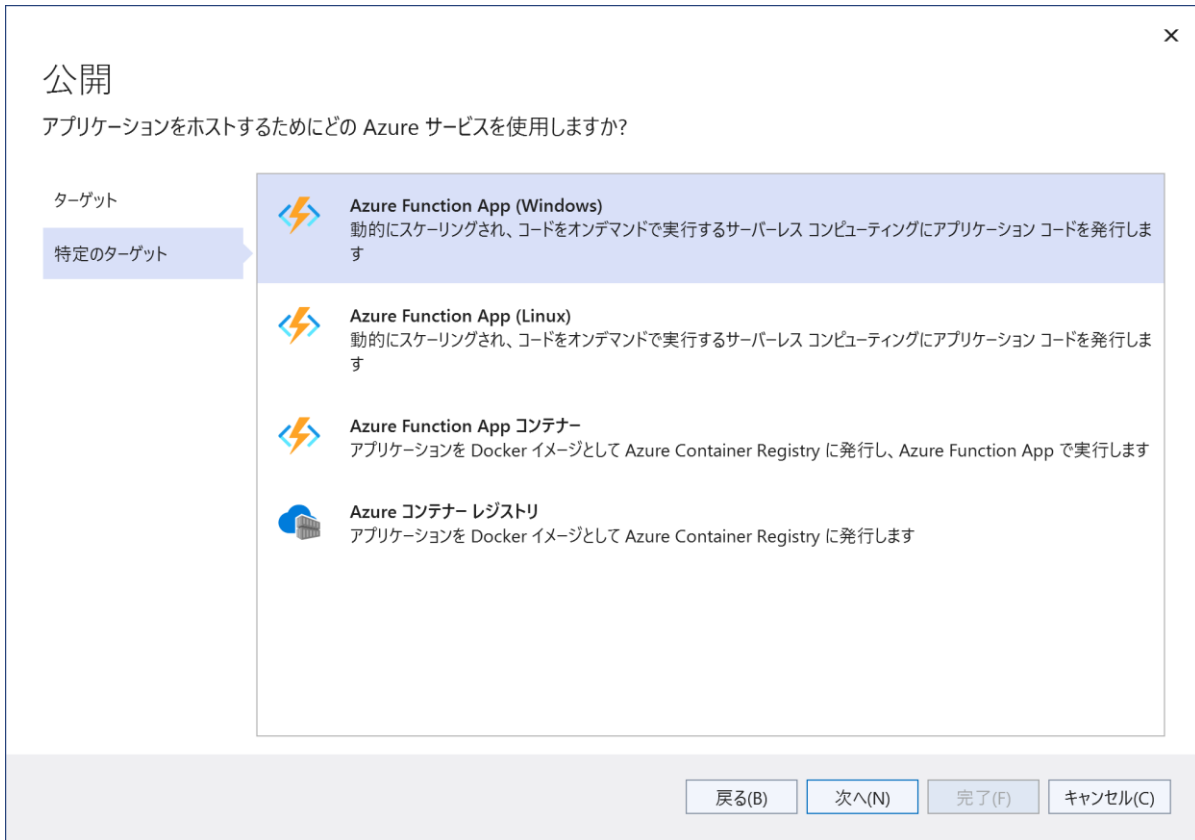
Azure ヘデプロイ

作成した Azure Functions アプリケーションを Azure ヘデプロイして確認します。ソリューションエクスプローラーから `DioDocsFileExportFunctionApp` プロジェクトを右クリックして [発行] を選択します。



公開するターゲットは「Azure」を選択します。特定のターゲットは「Azure Function App (Windows)」を選択します。

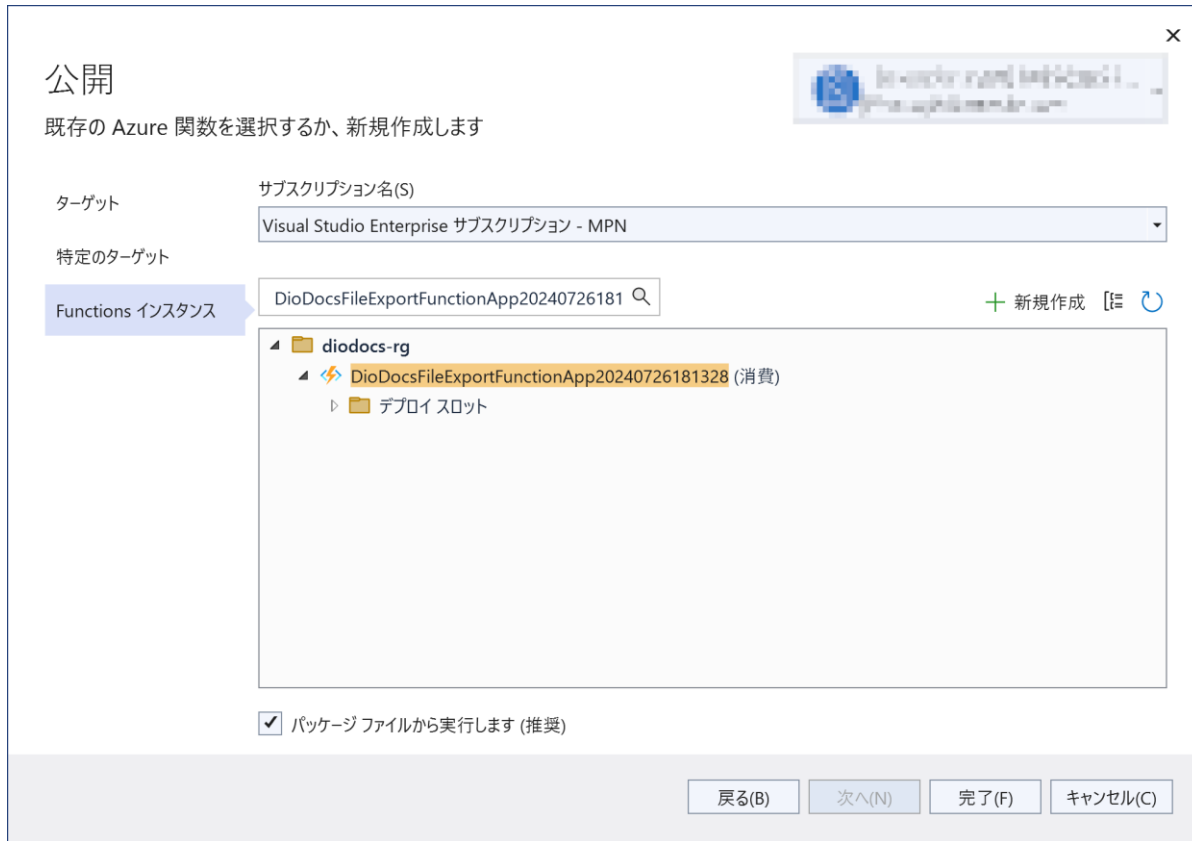




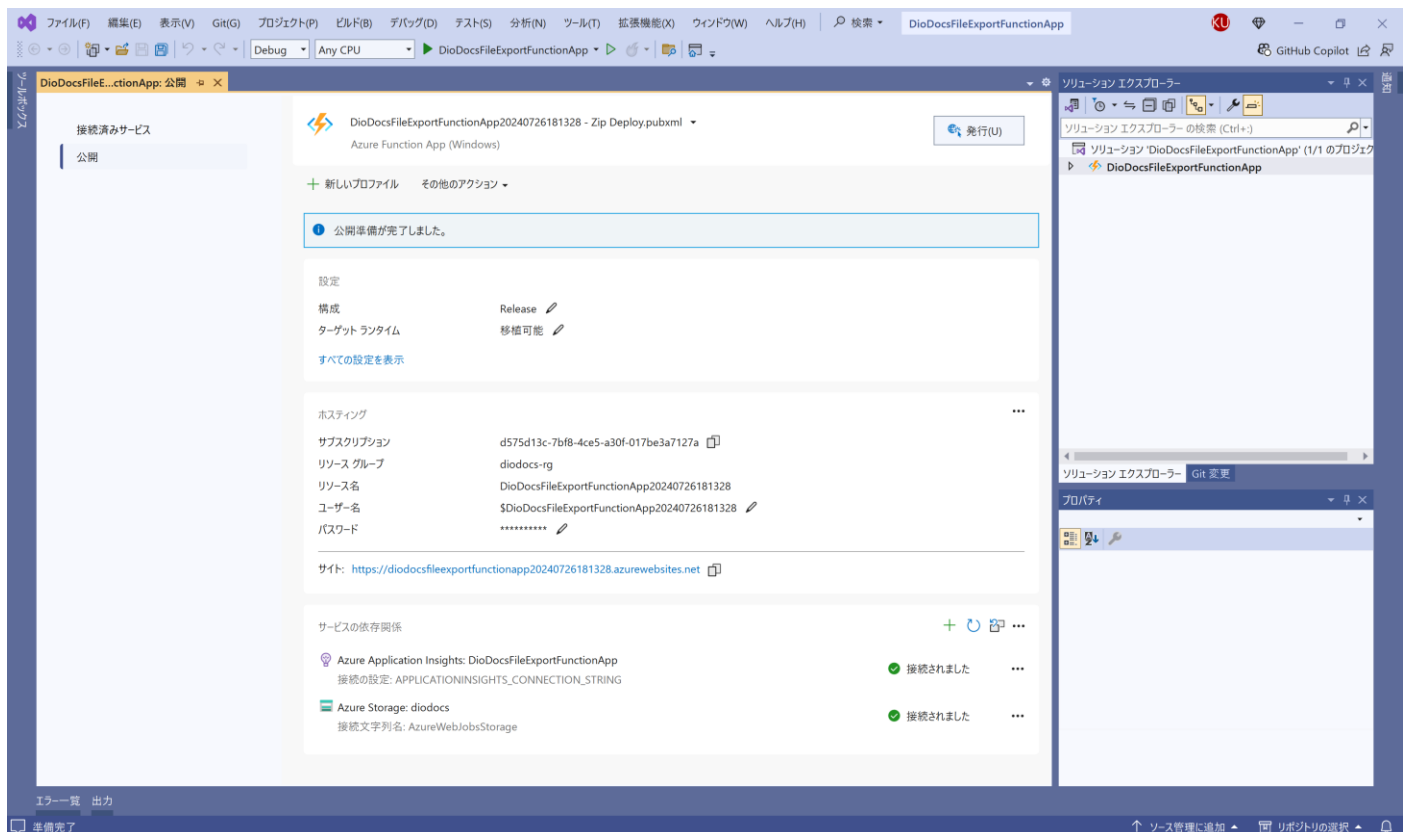
アプリケーションの名前やリソースグループなどを設定して [作成] をクリックします。



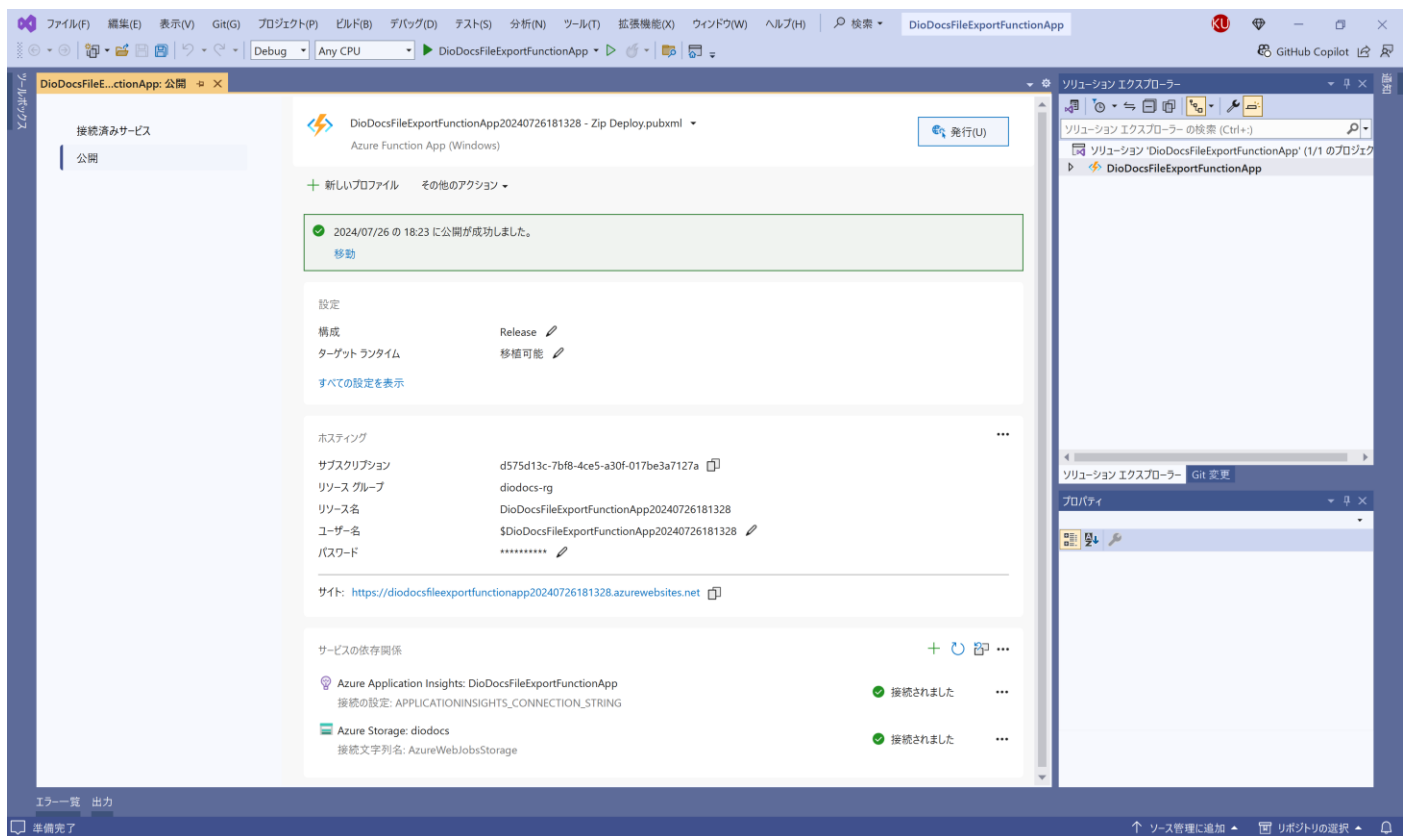
以下の画面に切り替わったら [完了] をクリックします。



これで公開の準備が完了しました。Visual Studio で [発行] をクリックして作成した Azure Function アプリケーションを Azure へデプロイします。

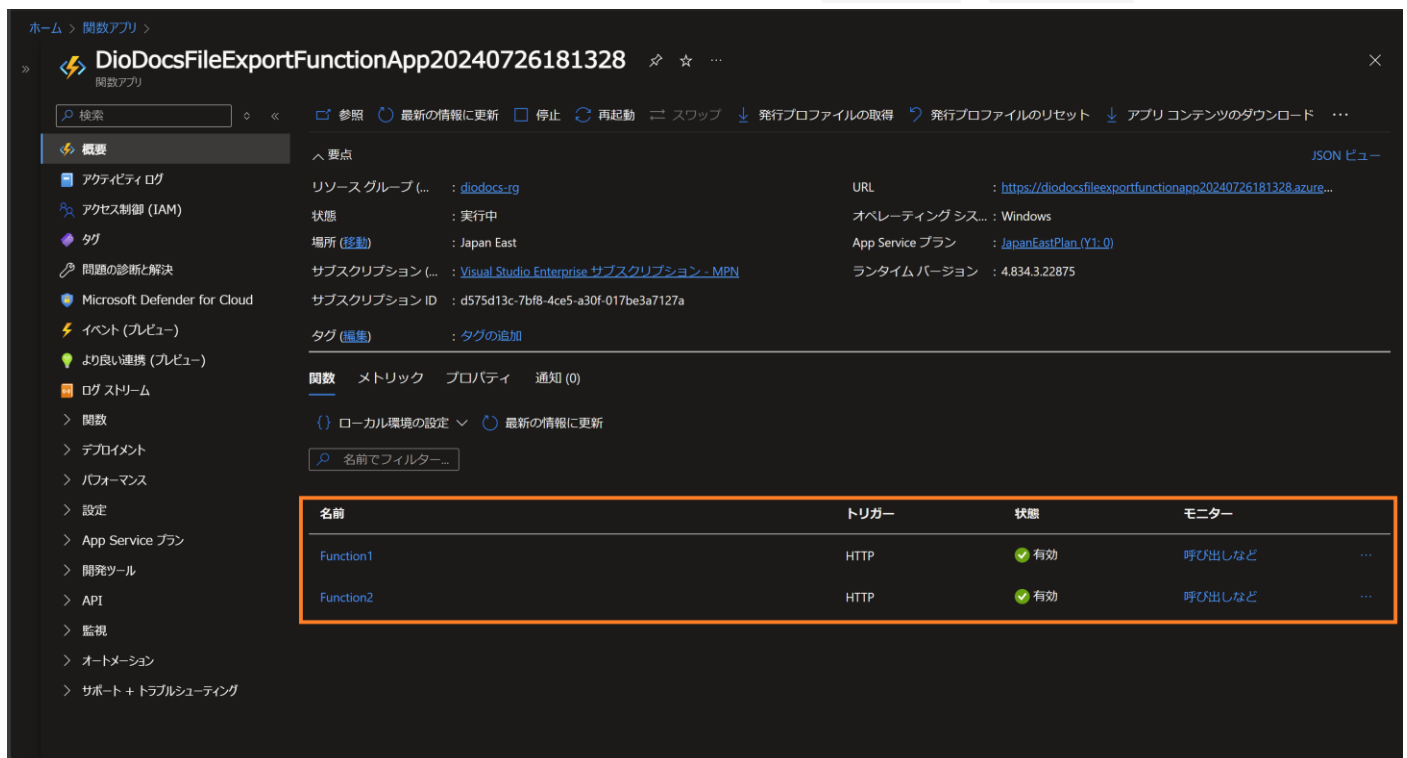


公開が完了するとメッセージが表示されます。

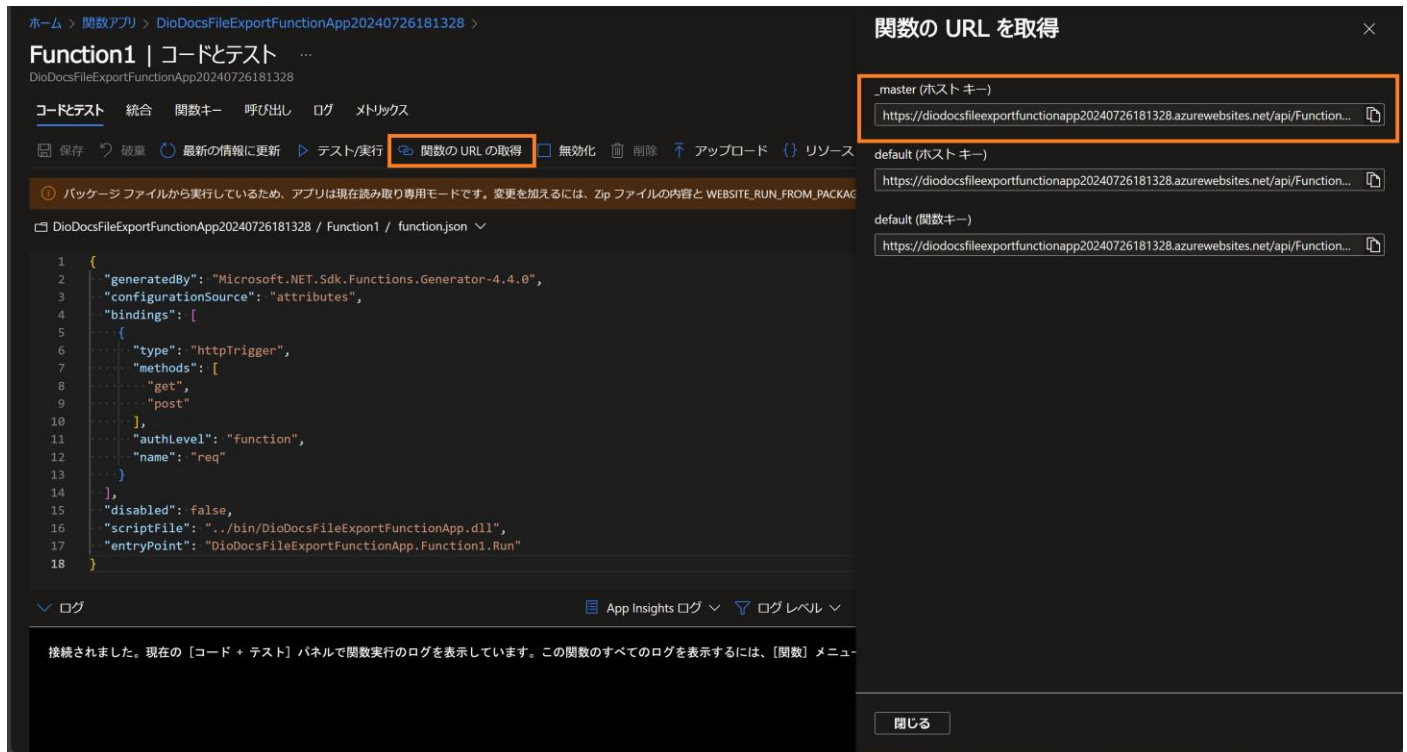


デプロイしたアプリケーションを確認

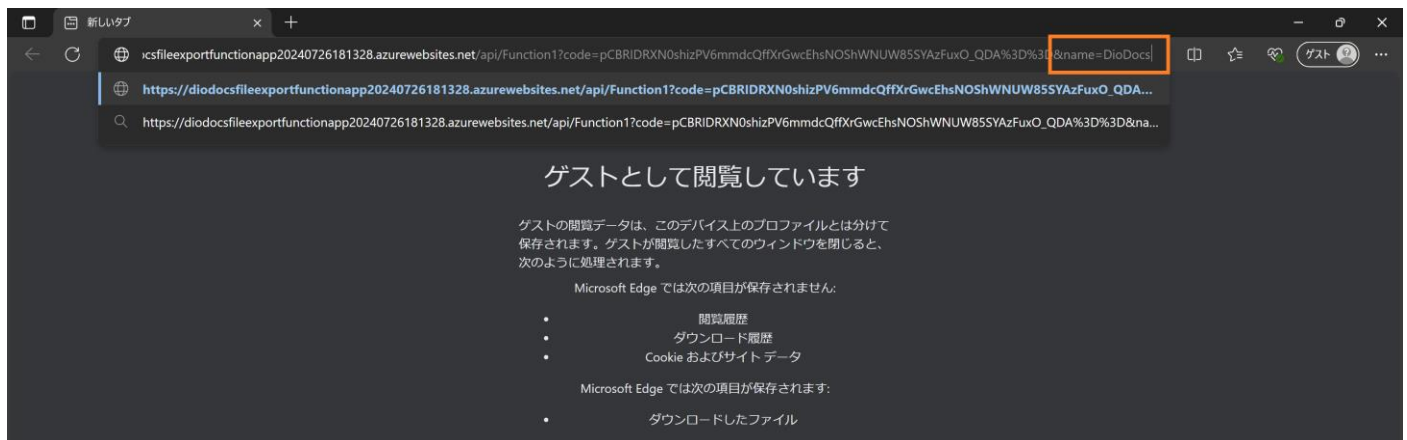
Azure ポータルでデプロイしたアプリケーションに含まれる関数 Function1、Function2 が表示されます。



関数 Function1 をクリックして「コードとテスト」から「テスト/実行」を選択します。「関数の URL を取得」が表示されるのでこちらをクリックします。この関数の URL をコピーします。



コピーした URL をブラウザに張り付けて、さらにクエリパラメータと文字列 `&name=DioDocs` を追加します。



関数を実行するとデバッグ実行時と同じように、クエリパラメータで渡した文字列が追加された Excel ファイルがローカルに出力されます。関数 Function2 も同じ手順で確認できます。

さいごに

動作を確認できる Azure Functions アプリケーションのサンプルはこちらです。

<https://github.com/MESCIUSJP/DioDocsFileExportFunctionApp>

Azure Functions と DioDocs で Excel や PDF ファイルを出力する (2)

[前回](#)に引き続き、本記事でも Azure Functions で「[DioDocs \(ディオドック\)](#)」を使用した C# (.NET 8) のクラスライブラリをベースにした関数を作成し、Excel や PDF ファイルを出力する方法について紹介します。

実装する内容

実装する内容は今回も非常にシンプルです。Azure Functions アプリケーションで [HTTP トリガー](#) を使用する関数を作成します。関数の実行時に DioDocs を使用して Excel と PDF ファイルを作成し、クエリパラメータで受け取った文字列をそれぞれのファイルへ追加します。

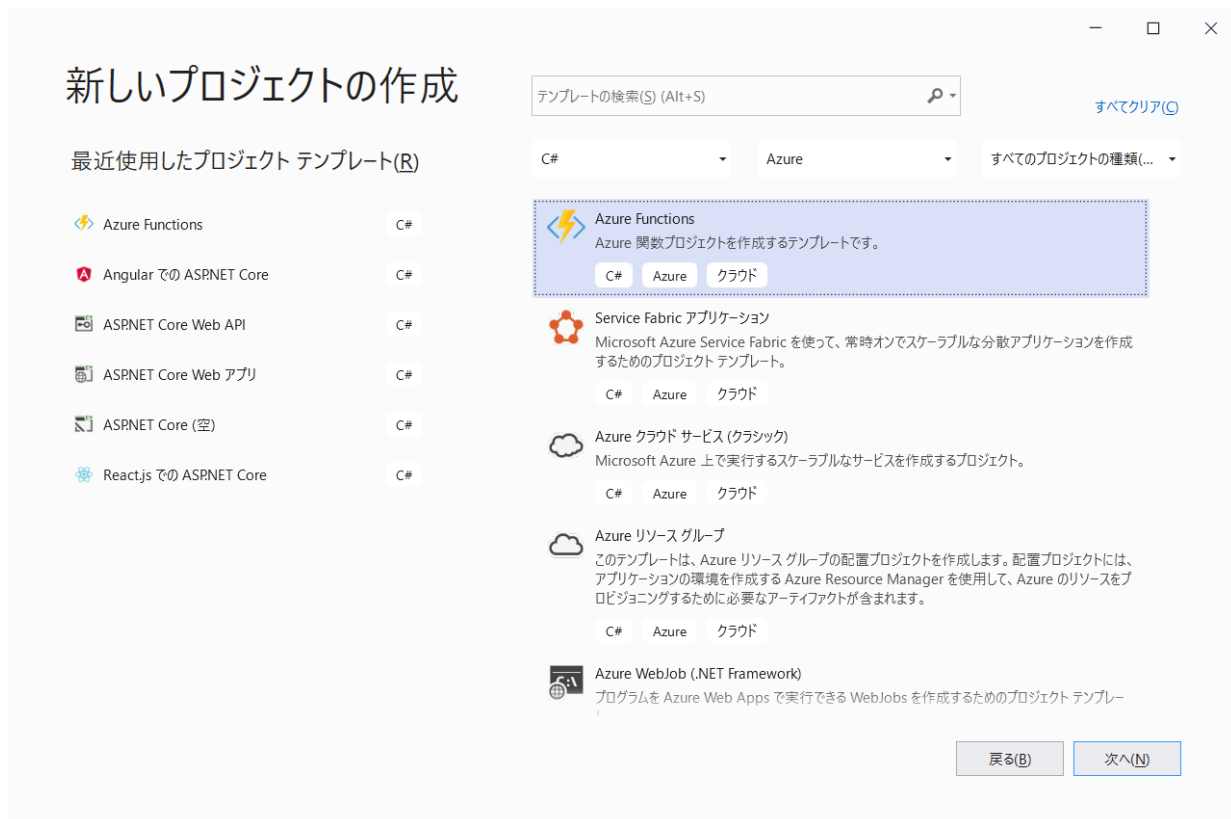
前回は作成した Excel と PDF ファイルを [FileContentResult](#) で直接ローカルへ出力していましたが、今回は作成した Excel と PDF ファイルを [Azure Blob Storage](#) へ出力します。

アプリケーションを作成

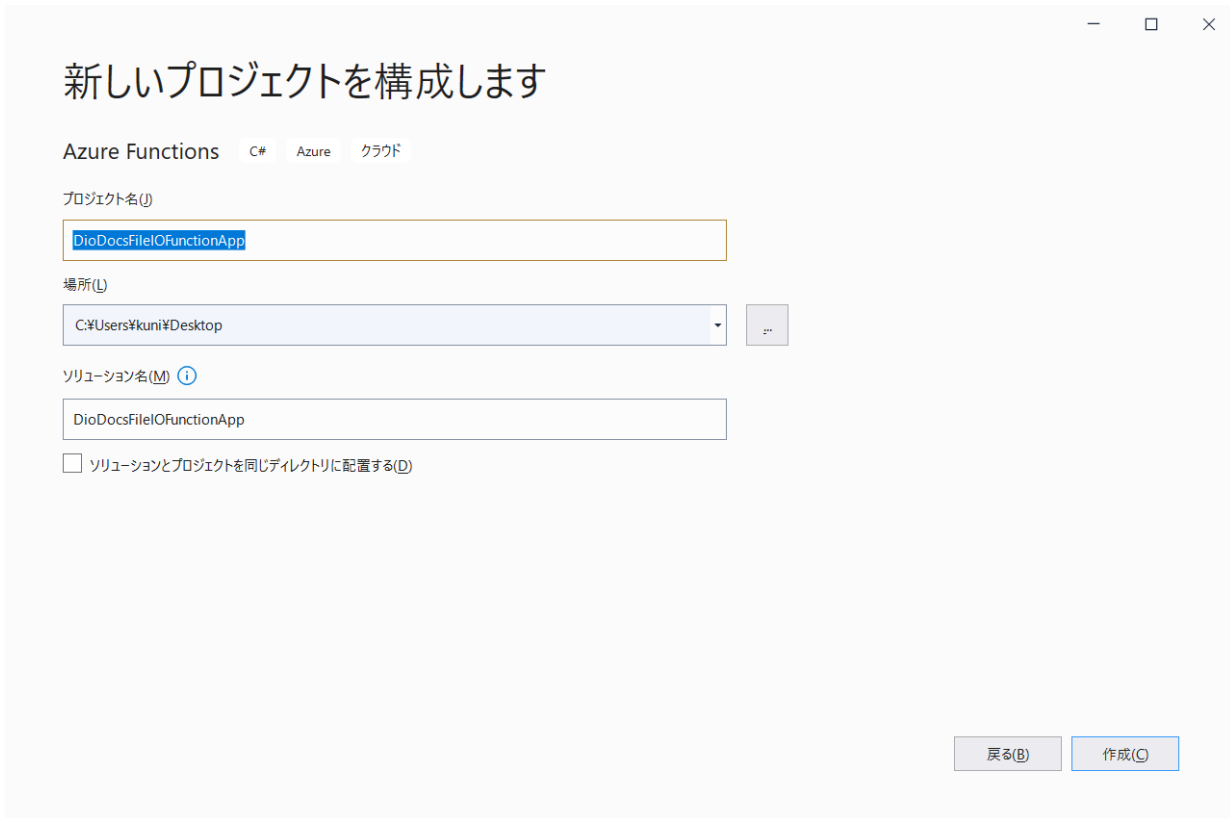
以下のドキュメントを参考に Azure Functions アプリケーションを作成していきます。

[クイック スタート:Visual Studio を使用して Azure で初めての関数を作成する](#)

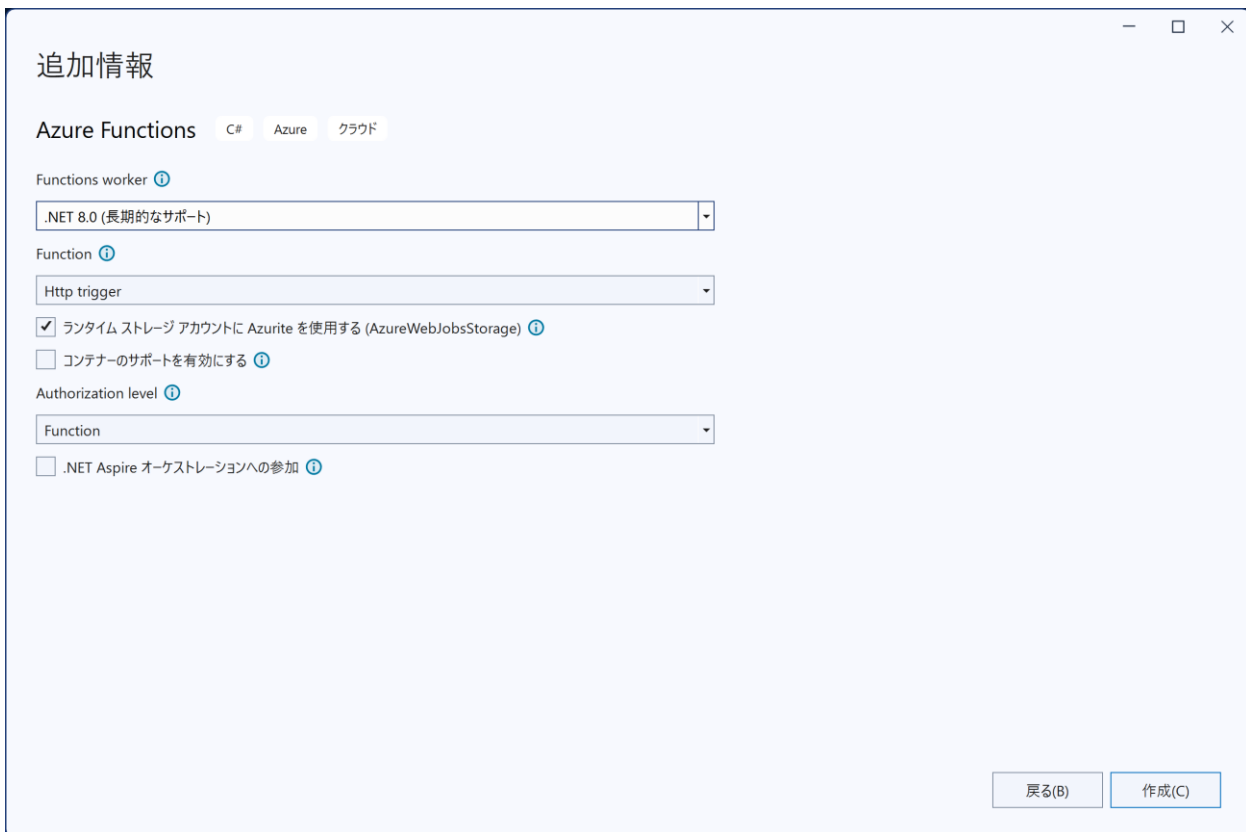
Visual Studio 2022 でプロジェクトテンプレート「Azure Functions」を選択して [次へ] をクリックします。



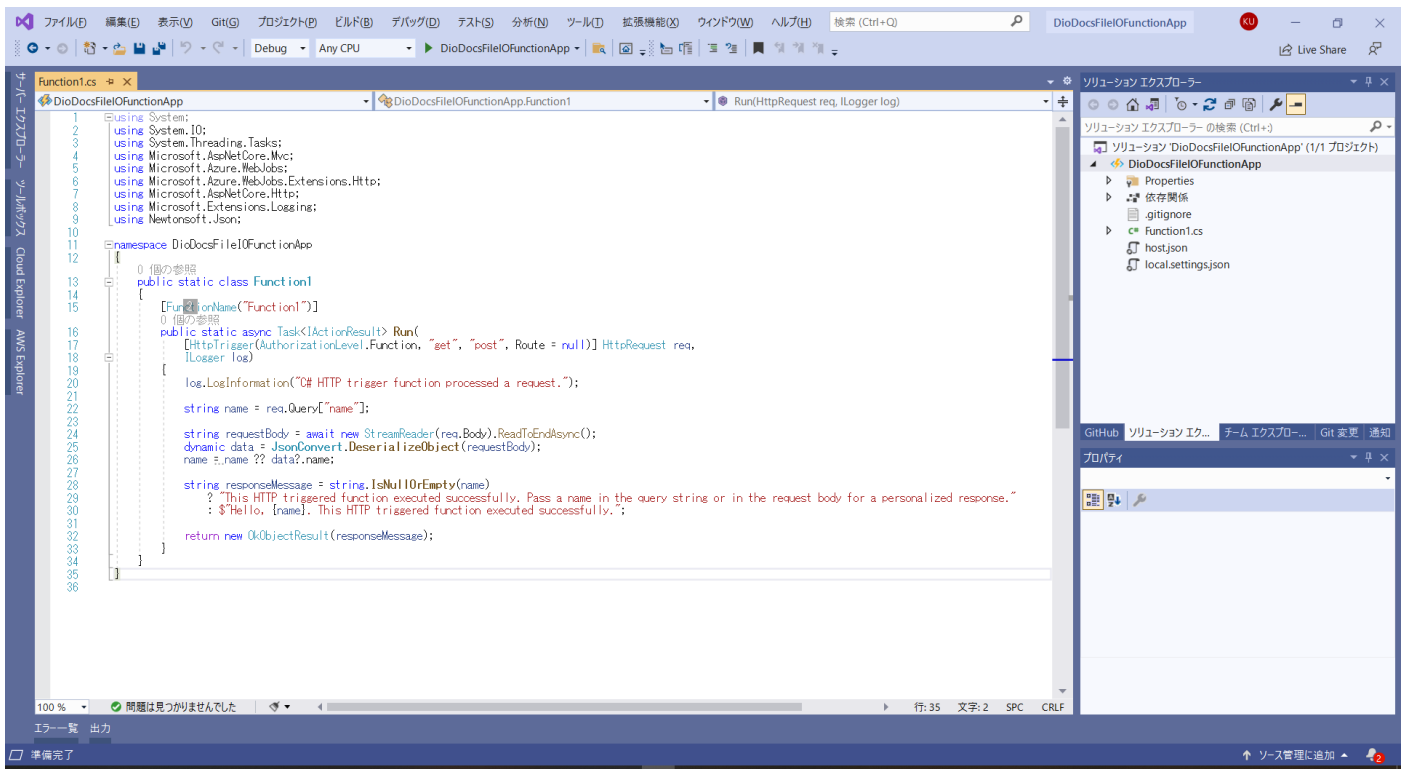
プロジェクト名 `DioDocsFileIOFunctionApp` を入力して [作成] をクリックします。



Azure Functions で作成する関数のテンプレートを選択します。Http Trigger を選択して [作成] をクリックします。

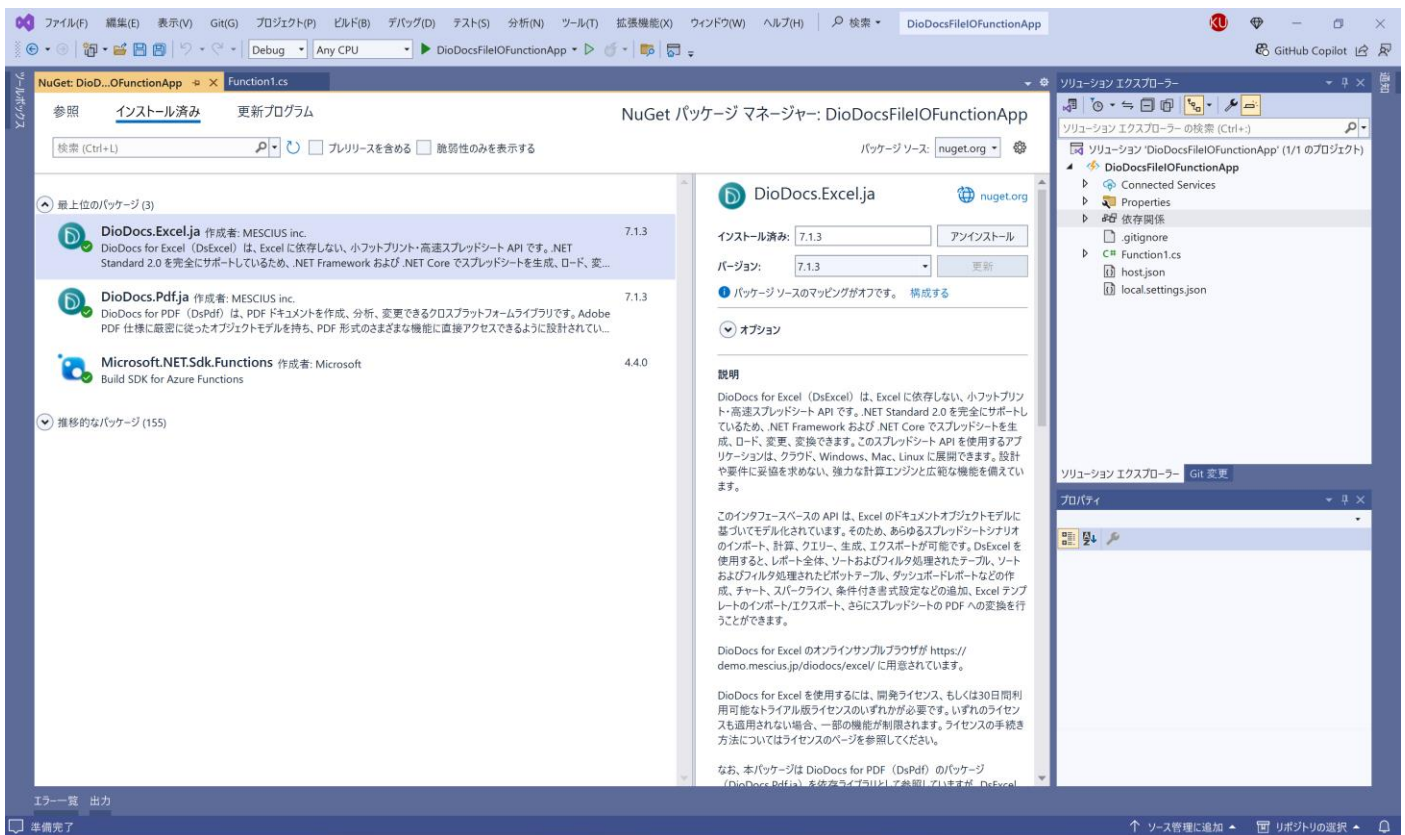


DioDocsFileIOFunctionApp プロジェクトが作成されます。



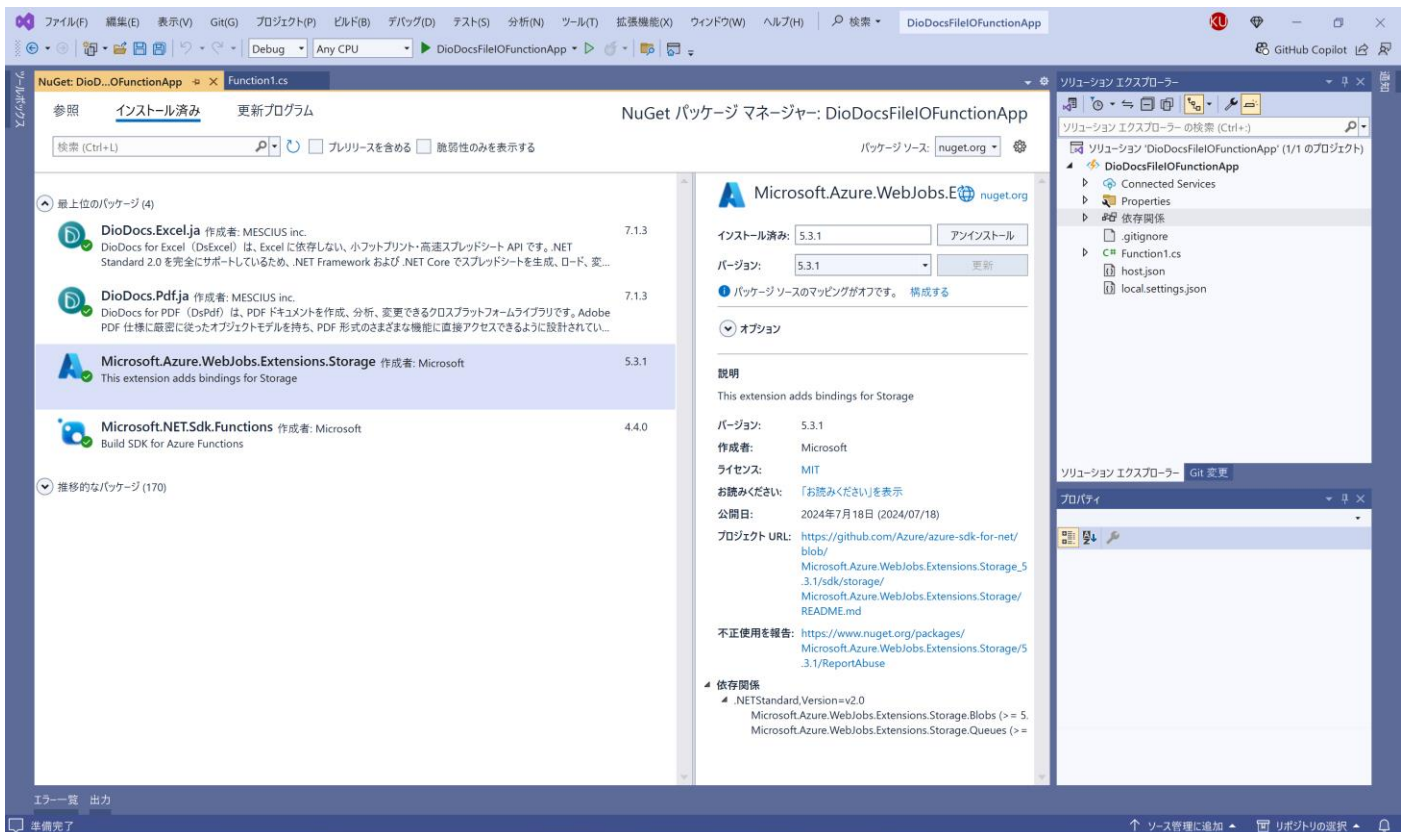
NuGet パッケージの追加

Visual Studio の「NuGet パッケージ マネージャー」から DioDocs のパッケージ `DioDocs.Excel.ja`、`DioDocs.Pdf.ja` をインストールします。



また、作成した Excel と PDF ファイルを Azure Blob Storage へ出力するために、[出力バインド](#)で [Azure Blob Storage](#) を使用する

ルします。



Azure Blob Storage を使うコードを追加

Azure Blob Storage の出力バインドを設定するコードを追加して `Function1` を以下のように更新します。

```
public static class Function1
{
    [FunctionName("Function1")]
    public static async Task<ActionResult> Run(
        [HttpTrigger(AuthorizationLevel.Function, "get", "post", Route = null)] HttpRequest req,
        [Blob("output/result.xlsx", FileAccess.Write)] Stream outputfile,
        ILogger log)
    {
        log.LogInformation("C# HTTP trigger function processed a request.");

        string name = req.Query["name"];

        string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
        dynamic data = JsonConvert.DeserializeObject(requestBody);
        name = name ?? data?.name;

        string responseMessage = string.IsNullOrEmpty(name)
            ? "This HTTP triggered function executed successfully. Pass a name in the query string or in the request body for a personalized response."
    }
}
```

```

        : $"Hello, {name}. This HTTP triggered function executed successfully.";

        return new OkObjectResult(responseMessage);
    }
}

```

追加した6行目のコードですが、出力バインドの属性コンストラクタ[Blob("output/result.xlsx", FileAccess.Write)]では、出力先となる Azure Blob Storage のコンテナとファイル名 output/result.xlsx と、Azure Blob Storage への書き込みアクセス FileAccess.Write を設定しています。

DioDocs for Excel を使うコードを追加

DioDocs で Excel ファイルを作成するコードを追加して Function1 を以下のように更新します。

```

public static class Function1
{
    [FunctionName("Function1")]
    public static async Task<IActionResult> Run(
        [HttpTrigger(AuthorizationLevel.Function, "get", "post", Route = null)] HttpRequest req,
        [Blob("output/result.xlsx", FileAccess.Write)] Stream outputfile,
        ILogger log)
    {
        log.LogInformation("C# HTTP trigger function processed a request.");

        string name = req.Query["name"];

        string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
        dynamic data = JsonConvert.DeserializeObject(requestBody);
        name = name ?? data?.name;

        string Message = string.IsNullOrEmpty(name)
            ? "Hello, World!!"
            : $"Hello, {name}!!";

        Workbook workbook = new Workbook();

        workbook.Worksheets[0].Range["B2"].Value = Message;

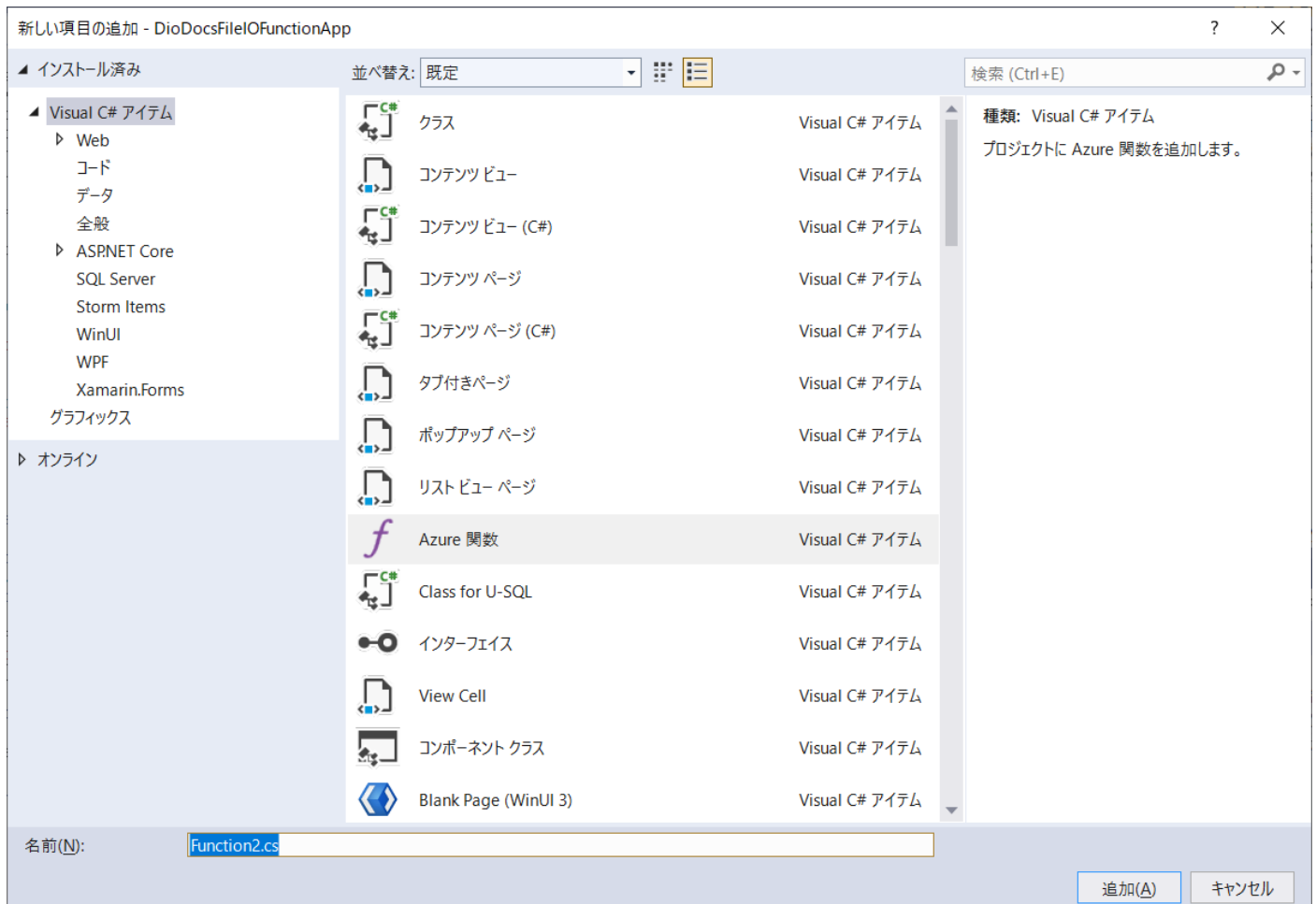
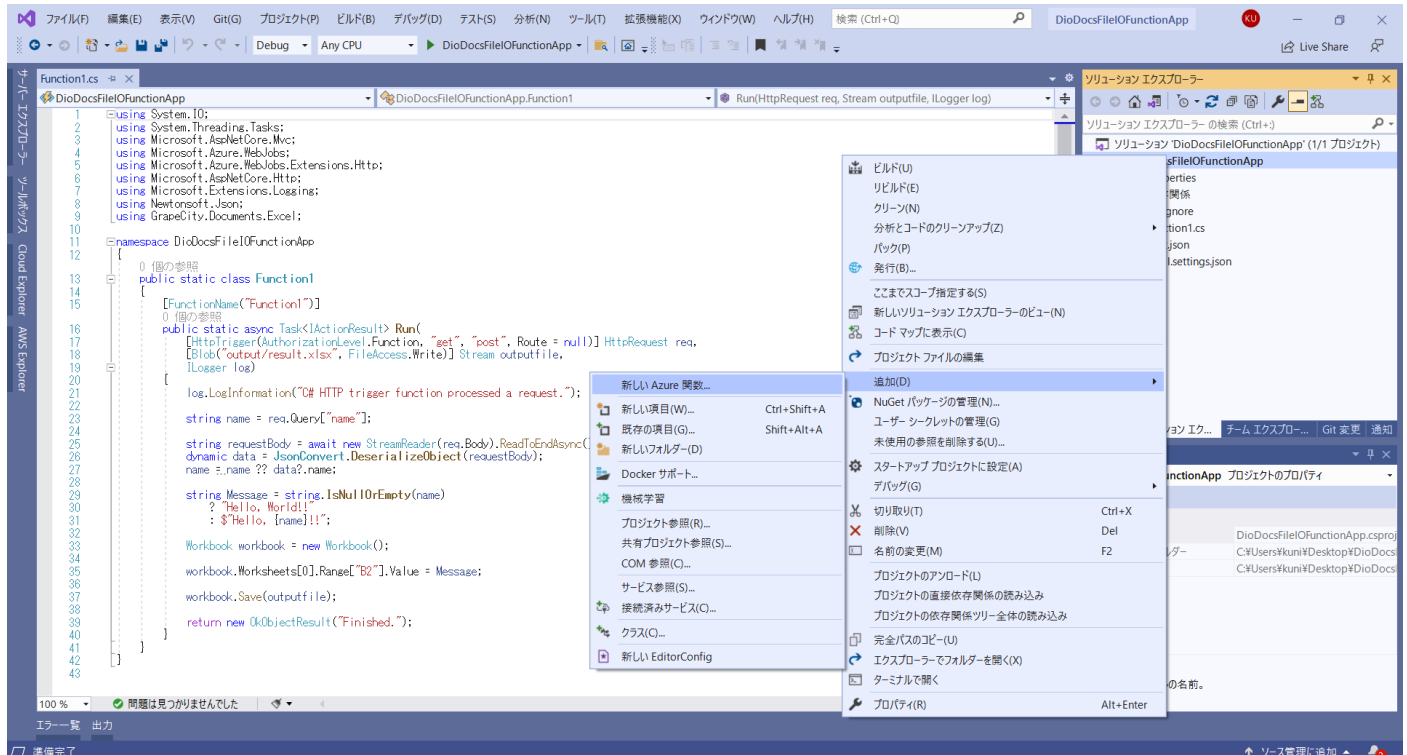
        workbook.Save(outputfile);

        return new OkObjectResult("Finished.");
    }
}

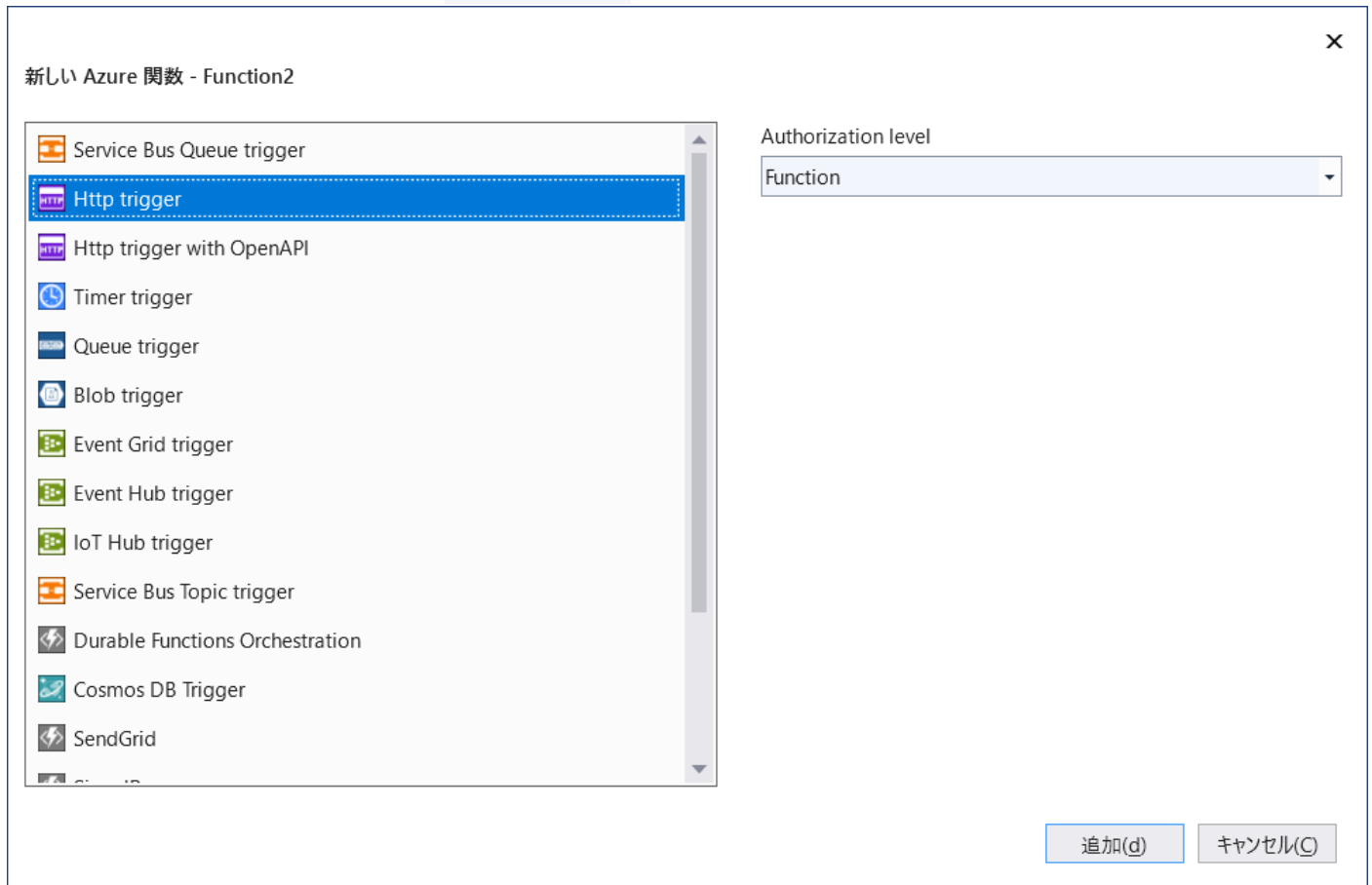
```

DioDocs for PDF を使う関数を追加

ソリューションエクスプローラーから DioDocsFileIOFunctionApp プロジェクトを右クリックして [追加] - [新しい Azure 関数] を選択して、DioDocs で PDF ファイルを作成する関数 Function2 を追加します。



関数のテンプレートを選択します。Http Trigger を選択して [追加] をクリックします。



Azure Blob Storage を使うコードを追加

Azure Blob Storage の出力バインドを設定するコードを追加して Function2 を以下のように更新します。

```
public static class Function2
{
    [FunctionName("Function2")]
    public static async Task<IActionResult> Run(
        [HttpTrigger(AuthorizationLevel.Function, "get", "post", Route = null)] HttpRequest req,
        [Blob("output/result.pdf", FileAccess.Write)] Stream outputfile,
        ILogger log)
    {
        log.LogInformation("C# HTTP trigger function processed a request.");

        string name = req.Query["name"];

        string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
        dynamic data = JsonConvert.DeserializeObject(requestBody);
        name = name ?? data?.name;

        string responseMessage = string.IsNullOrEmpty(name)
```

```

        ? "This HTTP triggered function executed successfully. Pass a name in the query string or in
the request body for a personalized response."
        : $"Hello, {name}. This HTTP triggered function executed successfully.";

    return new OkObjectResult(responseMessage);
}
}

```

先程と同じように、追加した6行目のコードでは出力バインドの属性コンストラクタ

[Blob("output/result.pdf", FileAccess.Write)]では、出力先となる Azure Blob Storage のコンテナとファイル名 output/result.pdf と、Azure Blob Storage への書き込みアクセス FileAccess.Write を設定しています。

DioDocs for PDF を使うコードを追加

DioDocs で PDF ファイルを作成するコードを追加して Function2 を以下のように更新します。

```

public static class Function2
{
    [FunctionName("Function2")]
    public static async Task<IActionResult> Run(
        [HttpTrigger(AuthorizationLevel.Function, "get", "post", Route = null)] HttpRequest req,
        [Blob("output/result.pdf", FileAccess.Write)] Stream outputfile,
        ILogger log)
    {
        log.LogInformation("C# HTTP trigger function processed a request.");

        string name = req.Query["name"];

        string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
        dynamic data = JsonConvert.DeserializeObject(requestBody);
        name = name ?? data?.name;

        string Message = string.IsNullOrEmpty(name)
            ? "Hello, World!!"
            : $"Hello, {name}!!";

        GcPdfDocument doc = new GcPdfDocument();
        GcPdfGraphics g = doc.NewPage().Graphics;

        g.DrawString(Message,
            new TextFormat() { Font = StandardFonts.Helvetica, FontSize = 12 },
            new PointF(72, 72));
    }
}

```



```
doc.Save(outputfile, false);

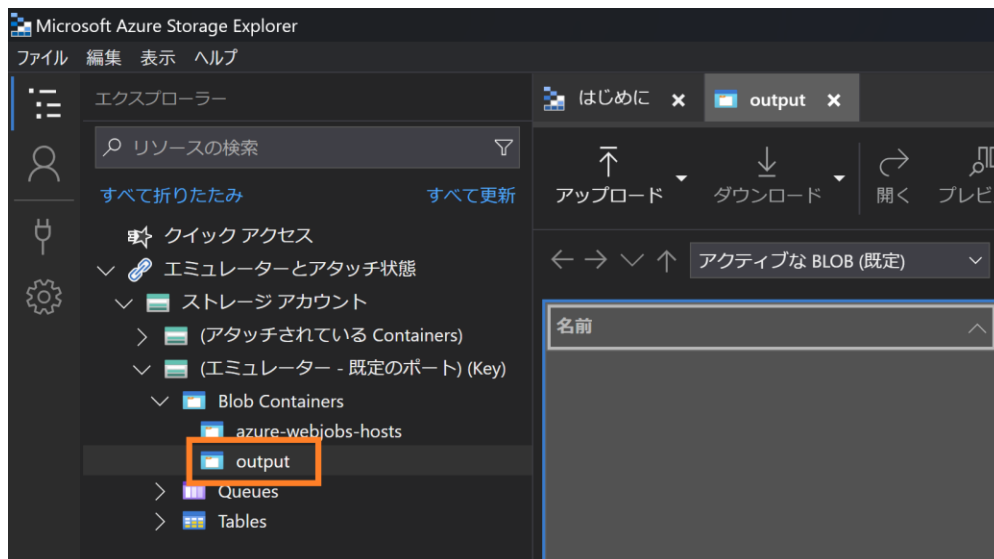
return new OkObjectResult("Finished.");
}
}
```

デバッグ実行で確認

今回は Azure Function アプリケーションの関数で作成する Excel と PDF ファイルの出力先として、関数の出力バインドで設定している Azure Blob Storage を使用しています。ローカルでの確認には [Azurite エミュレーター](#) を使用します。[Microsoft Azure Storage Explorer](#) で [Blob Containers] を右クリックして表示されるメニューから [BLOB コンテナの作成] を選択します。



output というコンテナを作成します。



作成した Azure Functions アプリケーションをローカルでデバッグ実行して確認します。Visual Studio からデバッグ実行すると以下のコンソールが表示されます。

```
C:\Users\kuni\AppData\Local\AzureFunctionsTools\Releases\3.23.5\cli_x64\func.exe
Azure Functions Core Tools
Core Tools Version:      3.0.3442 Commit hash: 6bfab24b2743f8421475d996402c398d2fe4a9e0 (64-bit)
Function Runtime Version: 3.0.15417.0

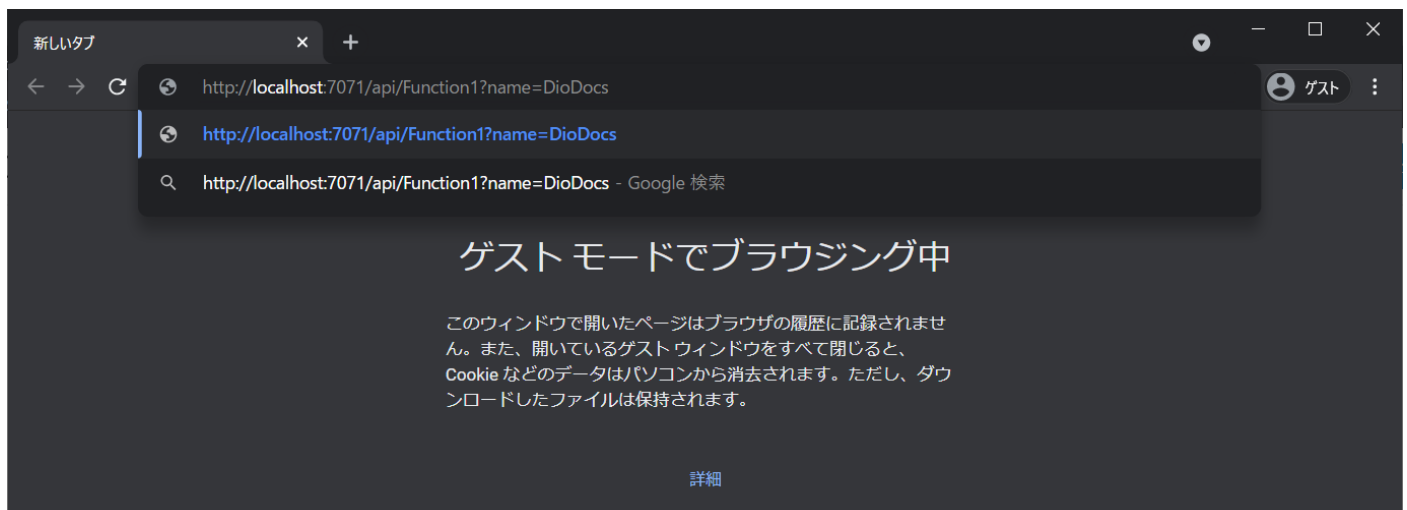
[2021-06-29T06:53:48.493Z] Found C:\Users\kuni\Desktop\DioDocsFile10FunctionApp\DioDocsFile10FunctionApp\DioDocsFile10FunctionApp.csproj. Using for user secrets file configuration.

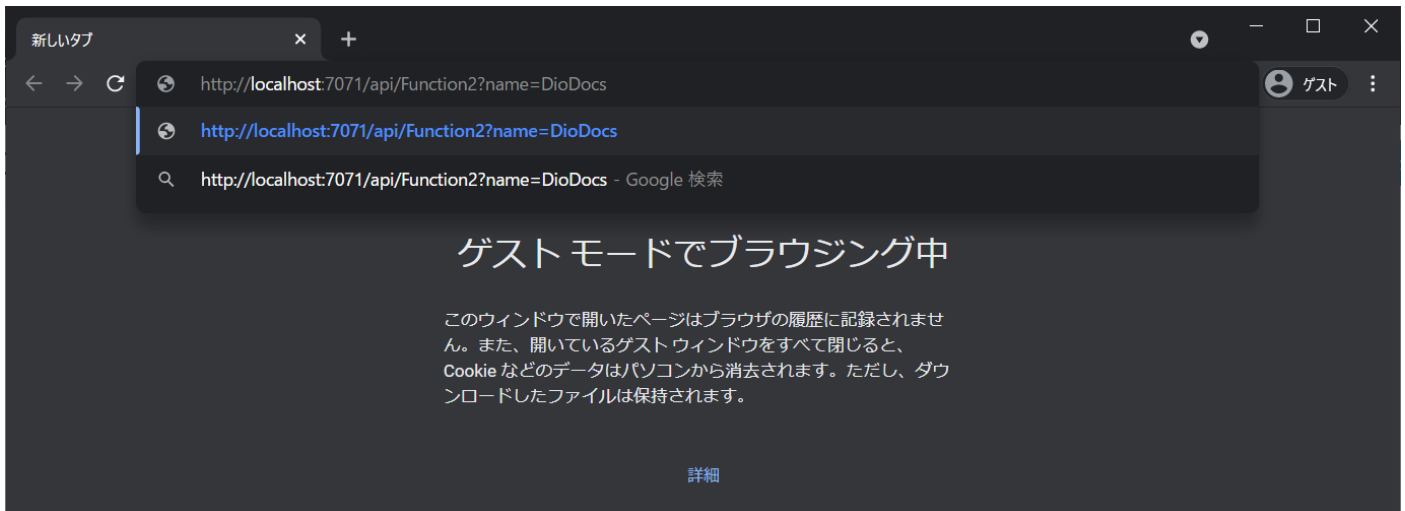
Functions:

    Function1: [GET,POST] http://localhost:7071/api/Function1
    Function2: [GET,POST] http://localhost:7071/api/Function2

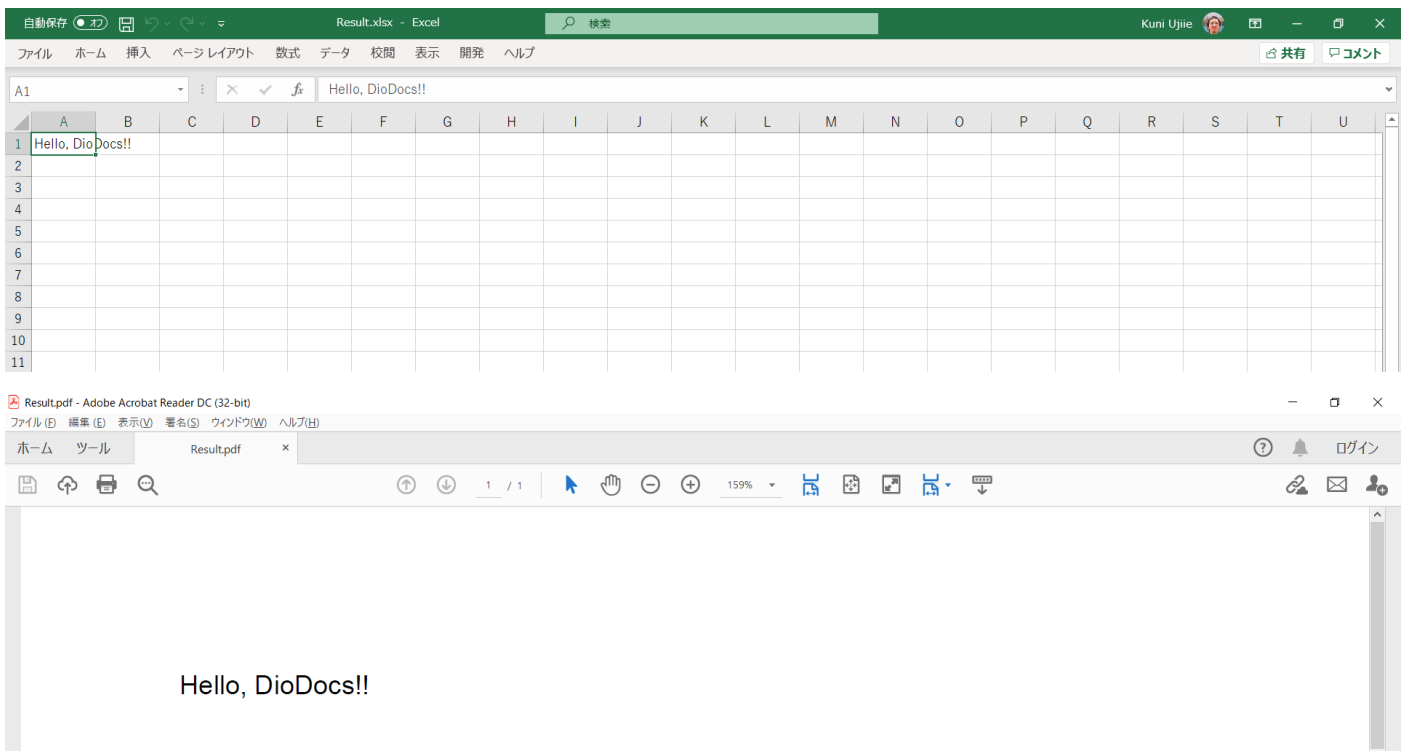
For detailed output, run func with --verbose flag.
```

アプリケーションに含まれる関数の URL は `http://localhost:7071/api/Function1`、`http://localhost:7071/api/Function2` となっています。この URL にクエリパラメータと文字列 `?name=DioDocs` を追加して、それぞれの関数をブラウザで実行します。



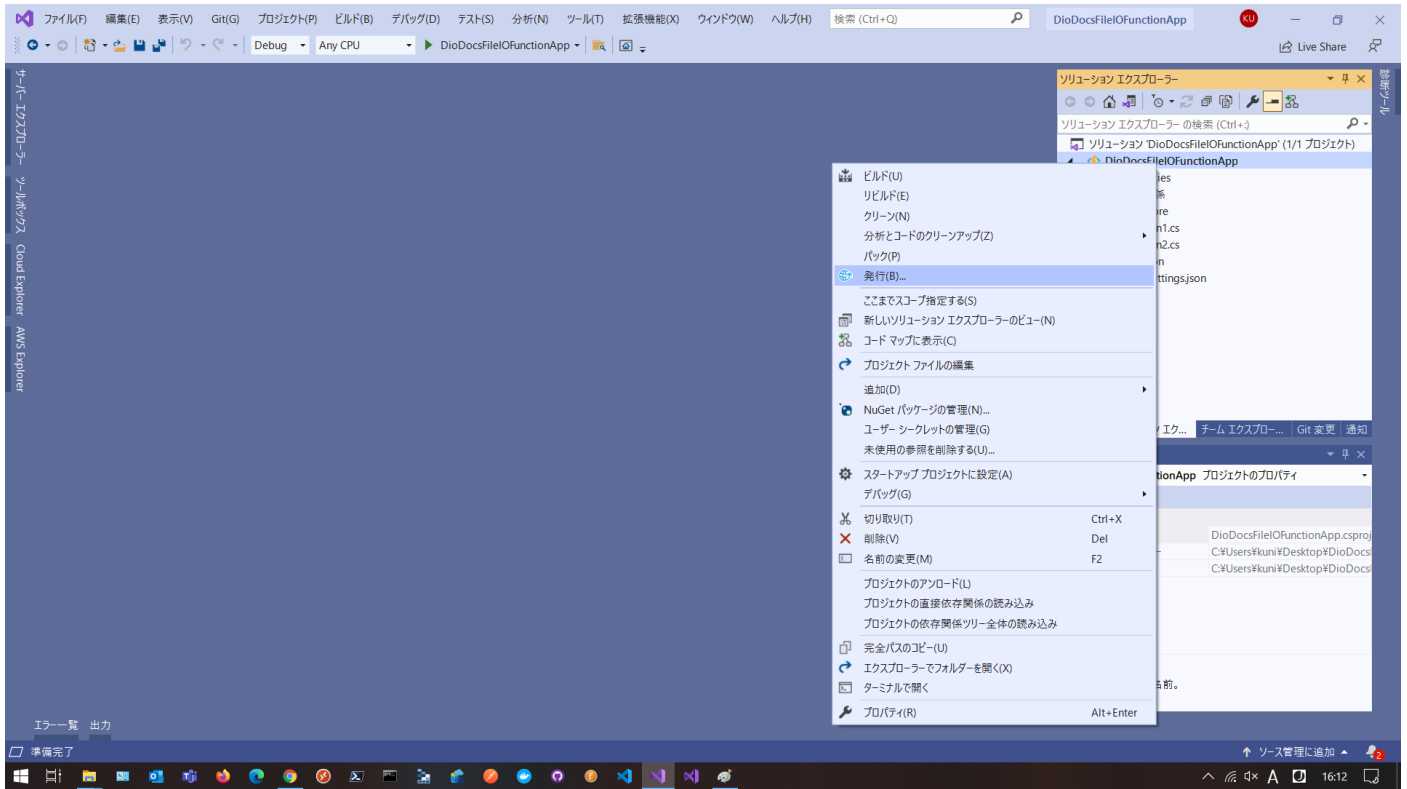


Microsoft Azure Storage Emulator のコンテナ output に保存された Result.xlsx、Result.pdf を確認します。クエリパラメータで渡した文字列 DioDocs が表示されていれば成功です。

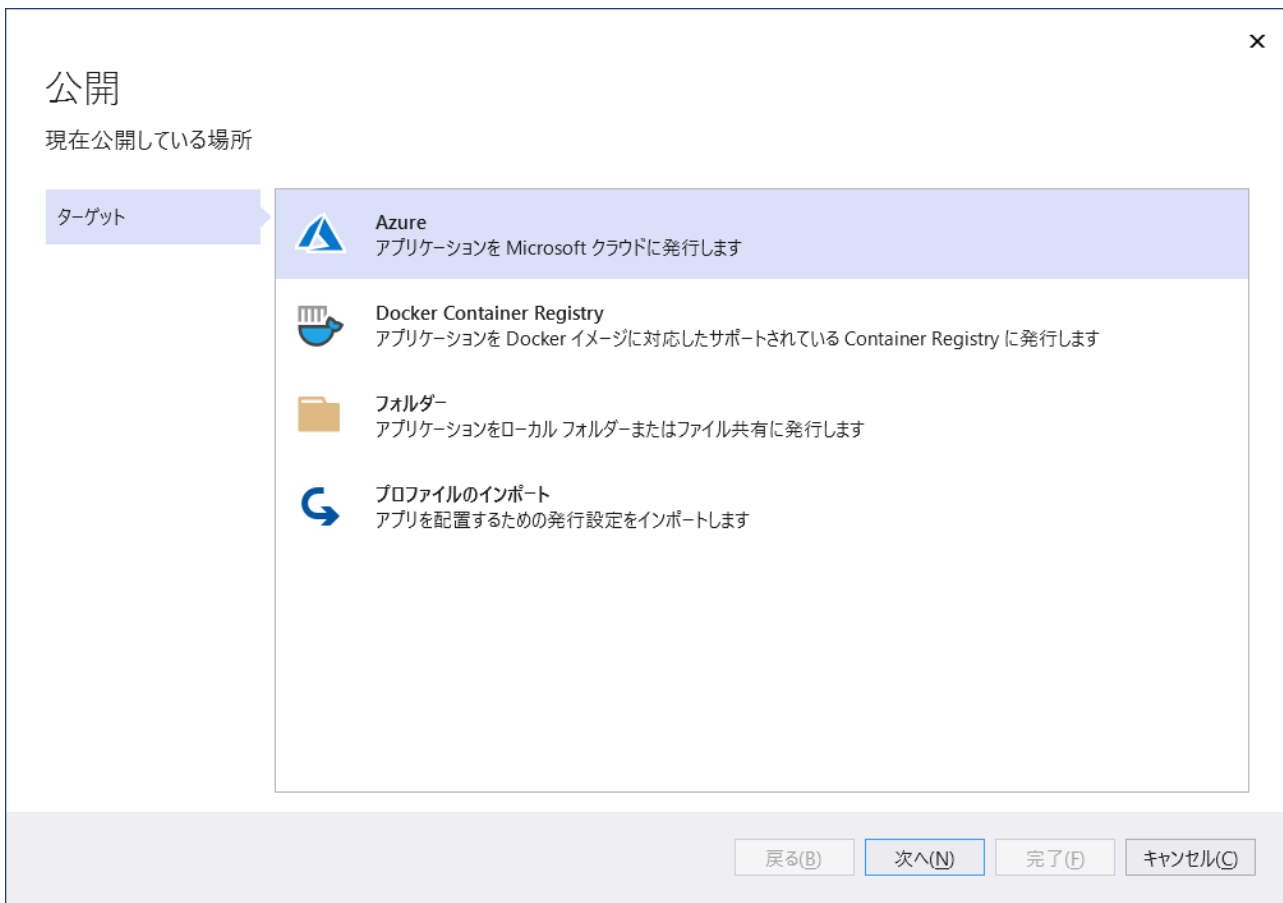


Azure へデプロイ

作成した Azure Functions アプリケーションを Azure へデプロイして確認します。ソリューションエクスプローラーから DioDocsFileIOFunctionApp プロジェクトを右クリックして [発行] を選択します。



公開するターゲットは「Azure」を選択します。特定のターゲットは「Azure Function App (Windows)」を選択します。





公開

アプリケーションをホストするためにどの Azure サービスを使用しますか？

ターゲット

特定のターゲット



Azure Function App (Windows)

動的にスケーリングされ、コードをオンデマンドで実行するサーバーレス コンピューティングにアプリケーション コードを発行します



Azure Function App (Linux)

動的にスケーリングされ、コードをオンデマンドで実行するサーバーレス コンピューティングにアプリケーション コードを発行します



Azure Function App コンテナ

アプリケーションを Docker イメージとして Azure Container Registry に発行し、Azure Function App で実行します



Azure コンテナ レジストリ

アプリケーションを Docker イメージとして Azure Container Registry に発行します

戻る(B)

次へ(N)

完了(F)

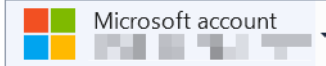
キャンセル(C)

アプリケーションの名前やリソースグループ、使用する Azure Storage などを設定して [作成] をクリックします。



Function App (Windows)

新規作成



名前(N)

DioDocsFileIOFunctionApp20210629161753

サブスクリプション名(S)

Visual Studio Enterprise サブスクリプション - MPN

リソースグループ(G)

diodocs-rg (Japan East)

新規(N)...

プランの種類(P)

消費

場所(L)

Japan East

Azure Storage

diodocs (Japan East)

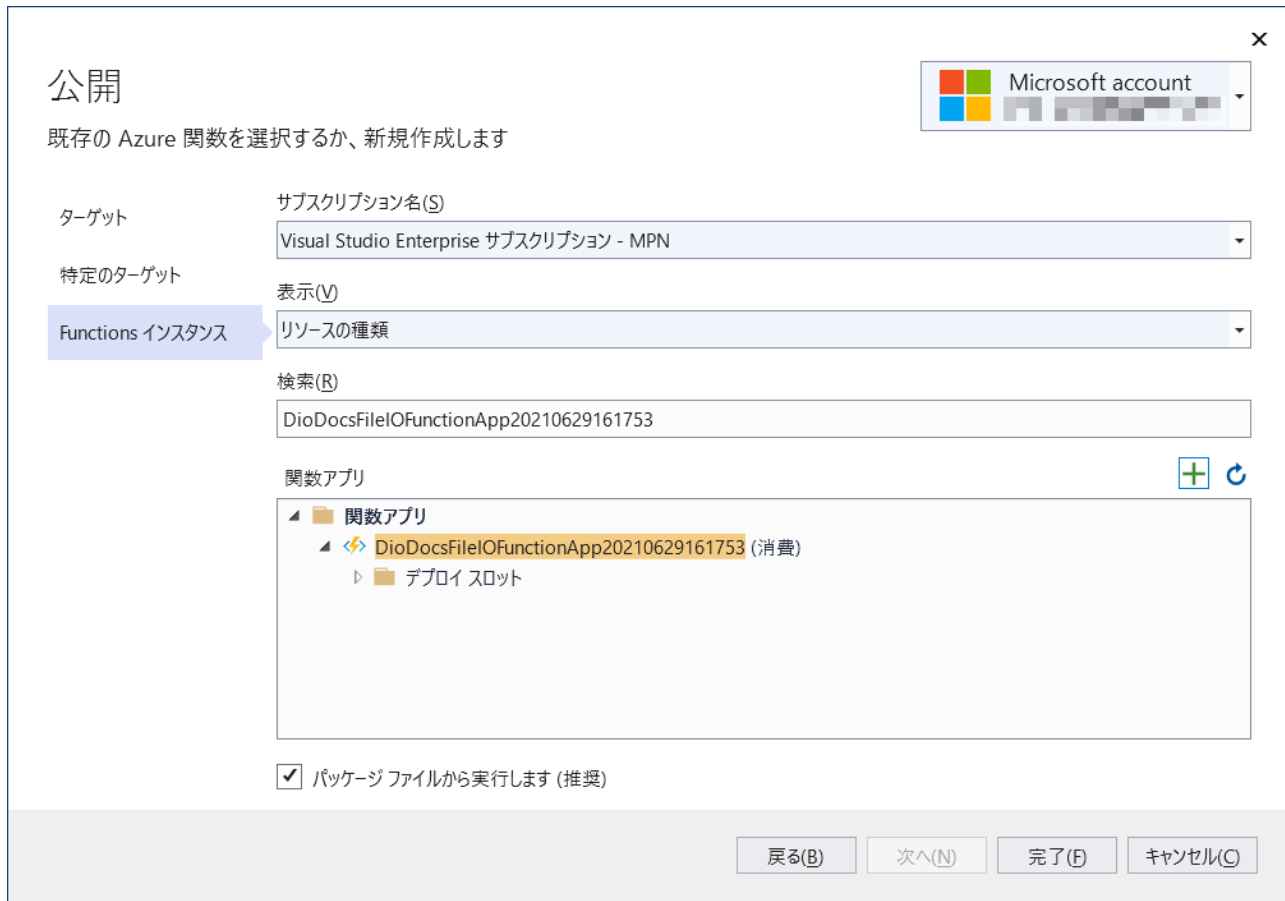
新規(N)...

エクスポート(E)...

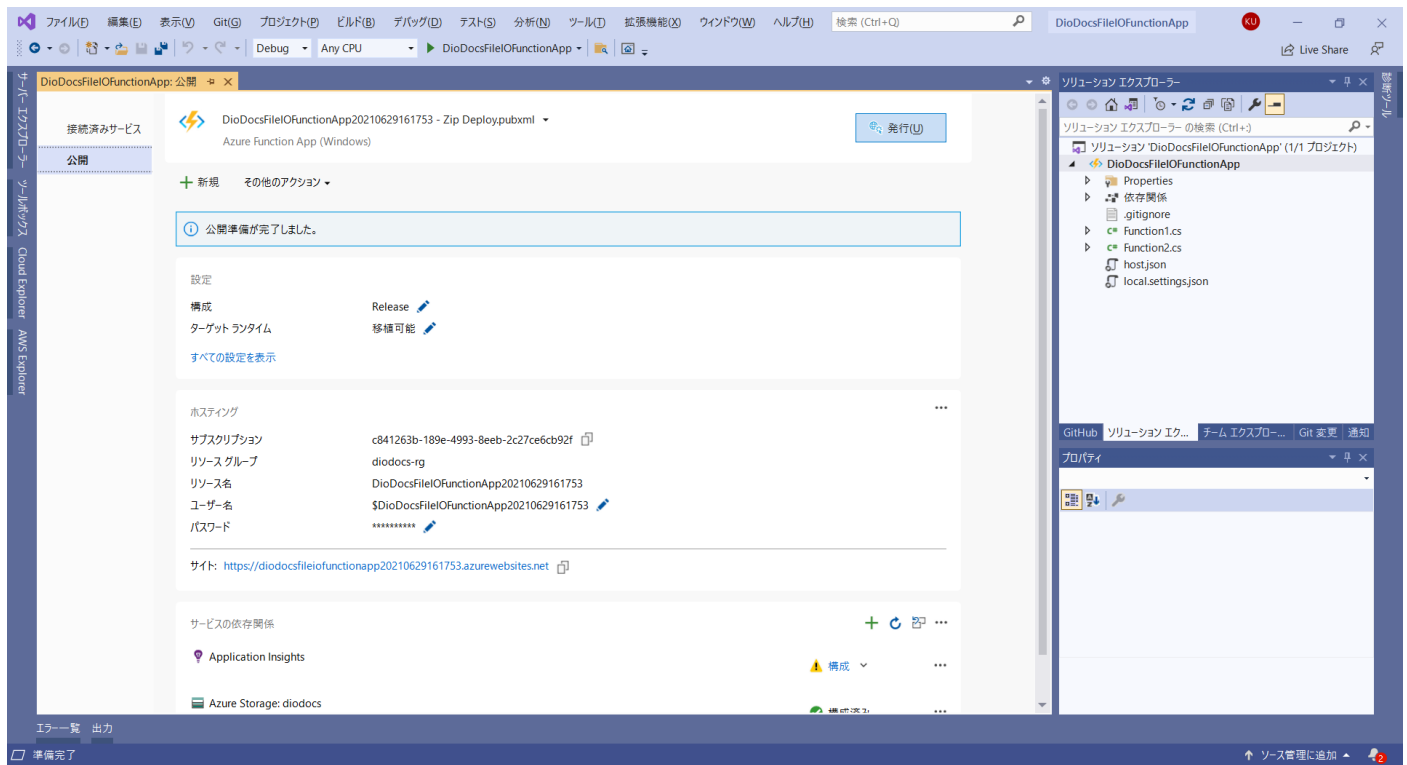
作成(B)

キャンセル(C)

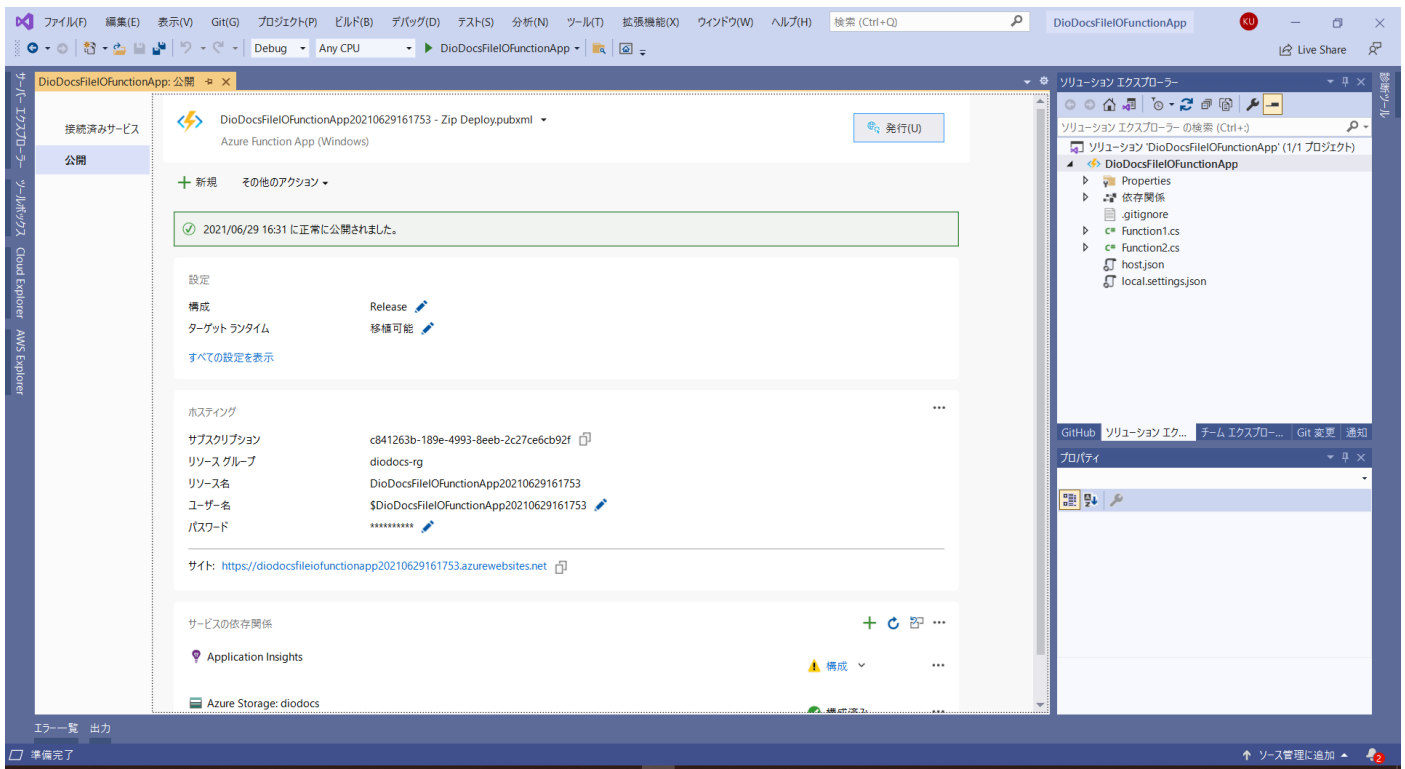
以下の画面に切り替わったら [完了] をクリックします。



これで公開の準備が完了しました。Visual Studio で [発行] をクリックして作成した Azure Function アプリケーションを Azure へデプロイします。

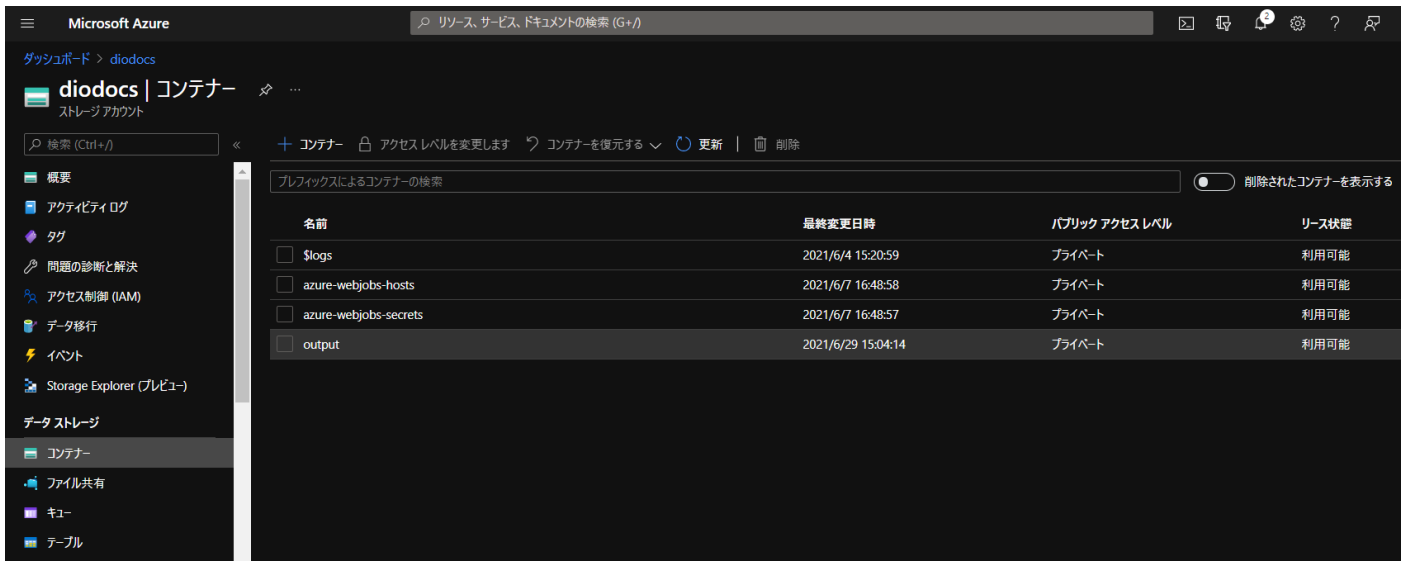


公開が完了するとメッセージが表示されます。



デプロイしたアプリケーションを確認

Azure Functions アプリケーションを実行する前に、ローカルでの確認と同じく関数の出力バインドで設定している Azure Blob Storage のコンテナー `output` を作成しておきます。



Azure ポータルでデプロイしたアプリケーションに含まれる関数 `Function1`、`Function2` が表示されます。

名前	トリガー	状態	モニター
Function1	HTTP	有効	呼び出しなど
Function2	HTTP	有効	呼び出しなど

関数 **Function1** をクリックして「コードとテスト」を選択します。「関数の URL を取得」が表示されるのでこちらをクリックします。

関数の URL を取得

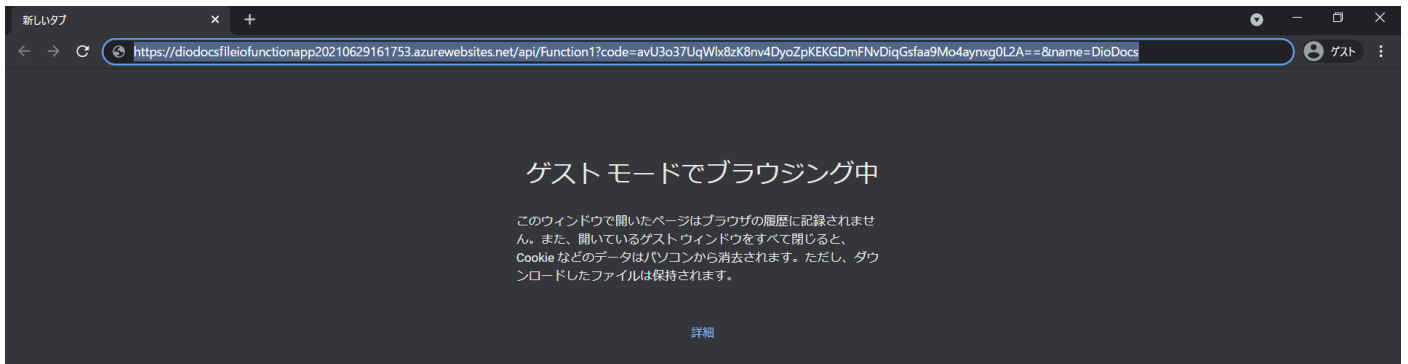
.master (ホスト キー)
https://diiodocsfileiofunctionapp20240729155904.azurewebsites.net/api/Function1?co...

blobs_extension (システム キー)
https://diiodocsfileiofunctionapp20240729155904.azurewebsites.net/api/Function1?co...

default (ホスト キー)
https://diiodocsfileiofunctionapp20240729155904.azurewebsites.net/api/Function1?co...

default (関数 キー)
https://diiodocsfileiofunctionapp20240729155904.azurewebsites.net/api/Function1?co...

コピーした URL をブラウザに張り付けて、さらにクエリパラメータと文字列 **&name=DioDocs** を追加します。



関数 `Function1` を実行するとデバッグ実行時と同じように、クエリパラメータで渡した文字列が追加された Excel ファイル `Result.xlsx` が Azure Blob Storage のコンテナ `output` に出力されます。関数 `Function2` も同じ手順で確認できます。



さいごに

動作を確認できる Azure Functions アプリケーションのサンプルはこちらです。

<https://github.com/MESCIUSJP/DioDocsFileIOFunctionApp>

Azure Functions と DioDocs で Excel や PDF ファイルを出力する (3)

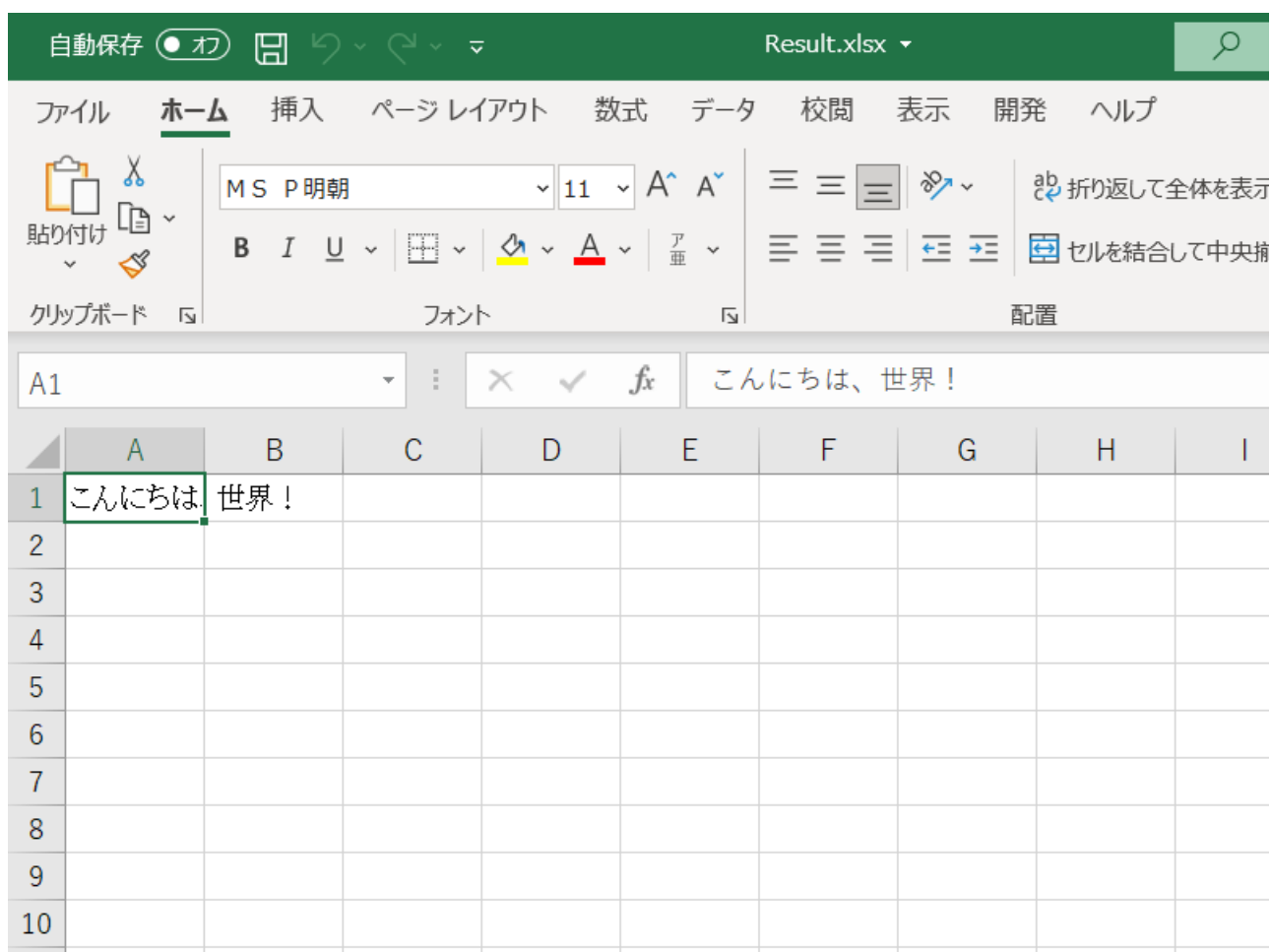
[前回](#)と[前々回](#)の記事では Azure Functions で「[DioDocs \(ディオドック\)](#)」を使用した C# (.NET 8) のクラスライブラリをベースにした関数を作成し、Excel や PDF ファイルを出力する方法について紹介しました。

今回は Azure Functions で DioDocs を利用する際に、日本語フォントを使用する Tips を紹介します。

セルに追加するテキストの日本語フォント (DioDocs for Excel)

セルに追加するテキストの日本語フォントを設定したい場合は、Font プロパティを使用します。

```
Workbook workbook = new Workbook();  
workbook.Worksheets[0].Range["A1"].Font.Name = "MS P明朝";
```



セルではなくシート全体のフォントを設定したい場合はこちらのナレッジベースを参考にしてください。

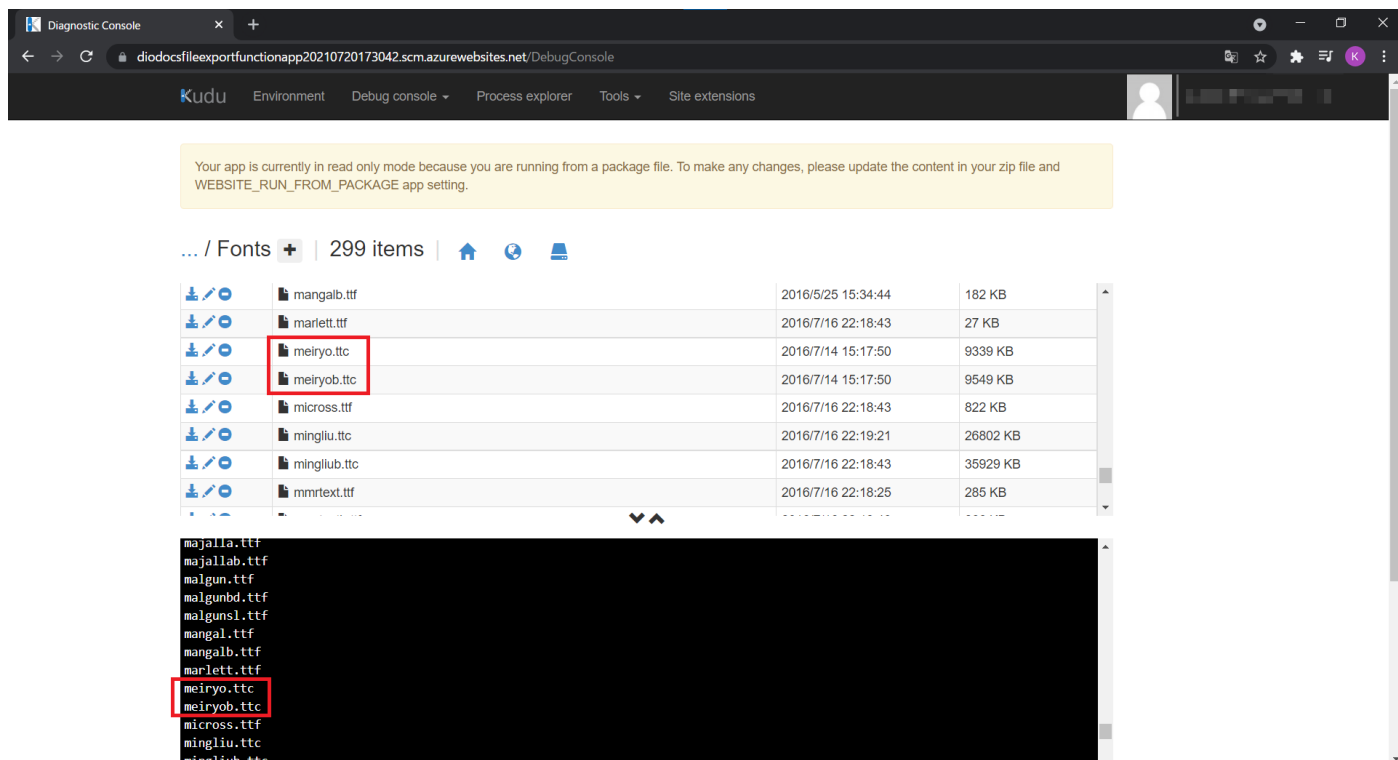
[「シート全体のフォントを設定する方法」を見る](#)

ワークシートを PDF 出力する際の日本語フォント (DioDocs for Excel)

DioDocs for Excel では `FontsFolderPath` プロパティで参照するフォルダを設定していない場合、Azure Functions アプリケーションの実行環境 (Windows) の `C:\Windows\Fonts` にインストールされている日本語フォントであれば `Font` プロパティで設定するだけです。

[「PDF エクスポート時に使用されるフォントについて」を見る](#)

例えば、`Font` プロパティで「メイリオ」を設定してワークシートを PDF 出力した場合、Azure Functions の実行環境の `C:\Windows\Fonts` には `meiryo.ttc` と `meiryob.ttc` が含まれていますので、追加で設定する必要は無くそのまま利用することができます。



```
public static class Function4
{
    [FunctionName("Function4")]
    public static async Task<IActionResult> Run(
        [HttpTrigger(AuthorizationLevel.Function, "get", "post", Route = null)] HttpRequest req,
        ILogger log)
    {
        log.LogInformation("C# HTTP trigger function processed a request.");

        string name = req.Query["name"];

        string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
        dynamic data = JsonConvert.DeserializeObject(requestBody);
        name = name ?? data?.name;
    }
}
```

```
string Message = string.IsNullOrEmpty(name)
    ? "こんにちは、世界！"
    : $"こんにちは、{name}！";

Workbook workbook = new Workbook();

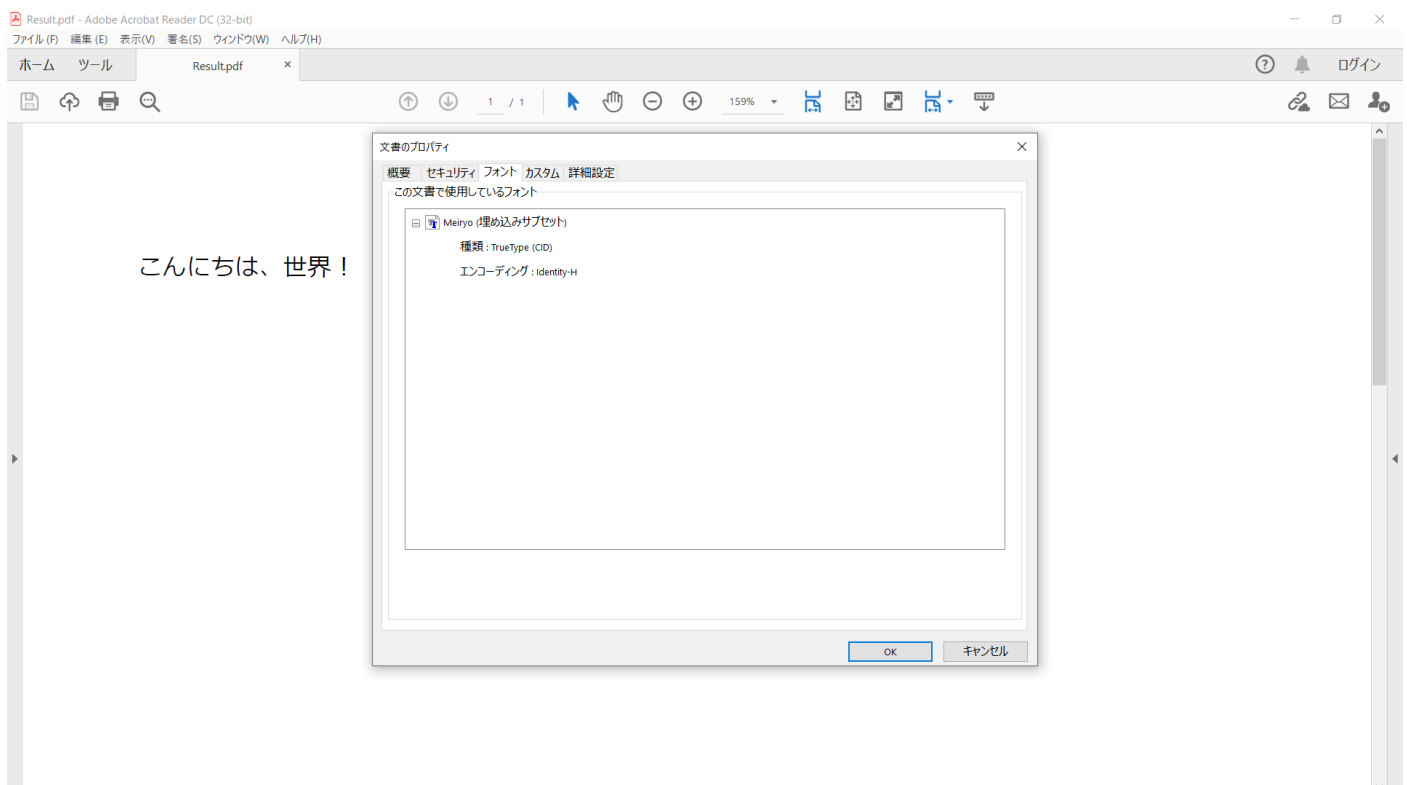
workbook.Worksheets[0].Range["A1"].Font.Name = "メイリオ";

workbook.Worksheets[0].Range["A1"].Value = Message;

byte[] output;

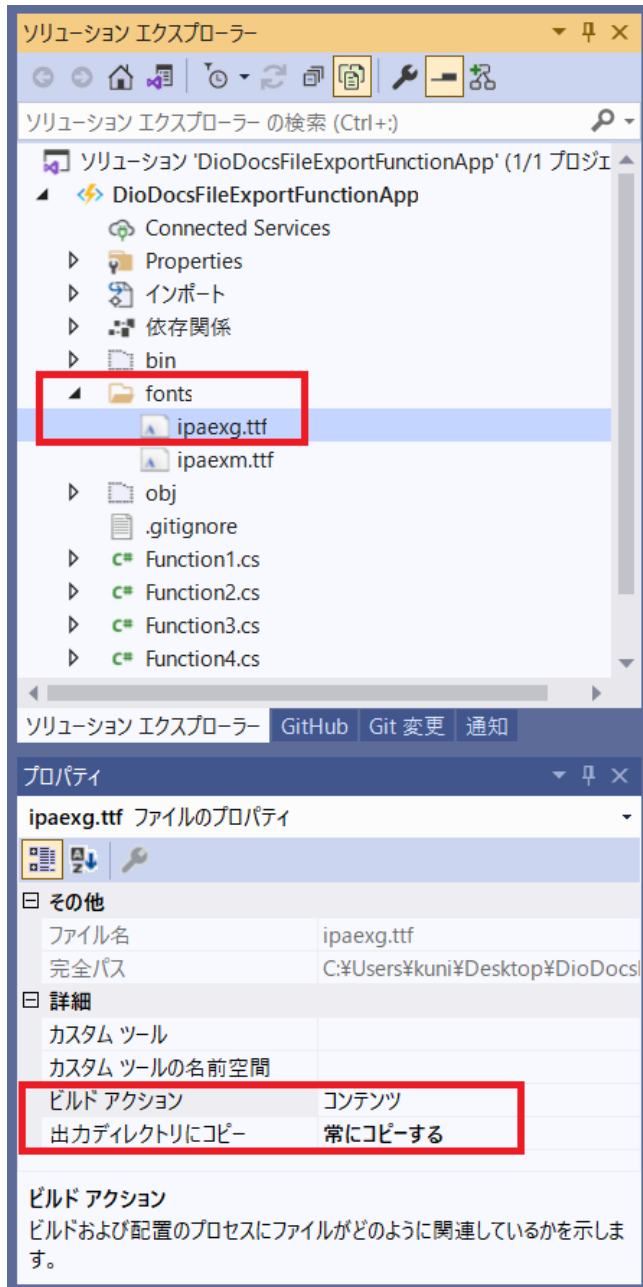
using (var ms = new MemoryStream())
{
    workbook.Save(ms, SaveFileFormat.Pdf);
    output = ms.ToArray();
}

return new FileContentResult(output, "application/pdf")
{
    FileNameDownload = "Result.pdf"
};
}
```



「IPAex フォント」のようなアプリケーションの実行環境 (Windows) の `C:\Windows\Fonts` に含まれていない日本語フォントを利用する場合は、日本語フォントをアプリケーションのコンテンツとして配置し、`FontsFolderPath` プロパティで参照する方法があります。

まず、以下のように「IPAex ゴシック」のフォント `ipaexg.ttf` をアプリケーションのプロジェクトに追加します。



このフォントが含まれる `fonts` フォルダへのパスを `FontsFolderPath` プロパティに設定します。この設定により `Font` プロパティで設定した「IPAex ゴシック」が利用できます。

```
public static class Function4
{
    [FunctionName("Function4")]
    public static async Task<IActionResult> Run(
```

```

    [HttpTrigger(AuthorizationLevel.Function, "get", "post", Route = null)] HttpRequest req,
Microsoft.Azure.WebJobs.ExecutionContext context,
    ILogger log)
{
    log.LogInformation("C# HTTP trigger function processed a request.");

    string name = req.Query["name"];

    string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
    dynamic data = JsonConvert.DeserializeObject(requestBody);
    name = name ?? data?.name;

    string Message = string.IsNullOrEmpty(name)
        ? "こんにちは、世界！"
        : $"こんにちは、{name}！";

    Workbook workbook = new Workbook();

    Workbook.FontsFolderPath = Path.Combine(context.FunctionAppDirectory, "fonts");

    workbook.Worksheets[0].Range["A1"].Font.Name = "IPAex ゴシック";

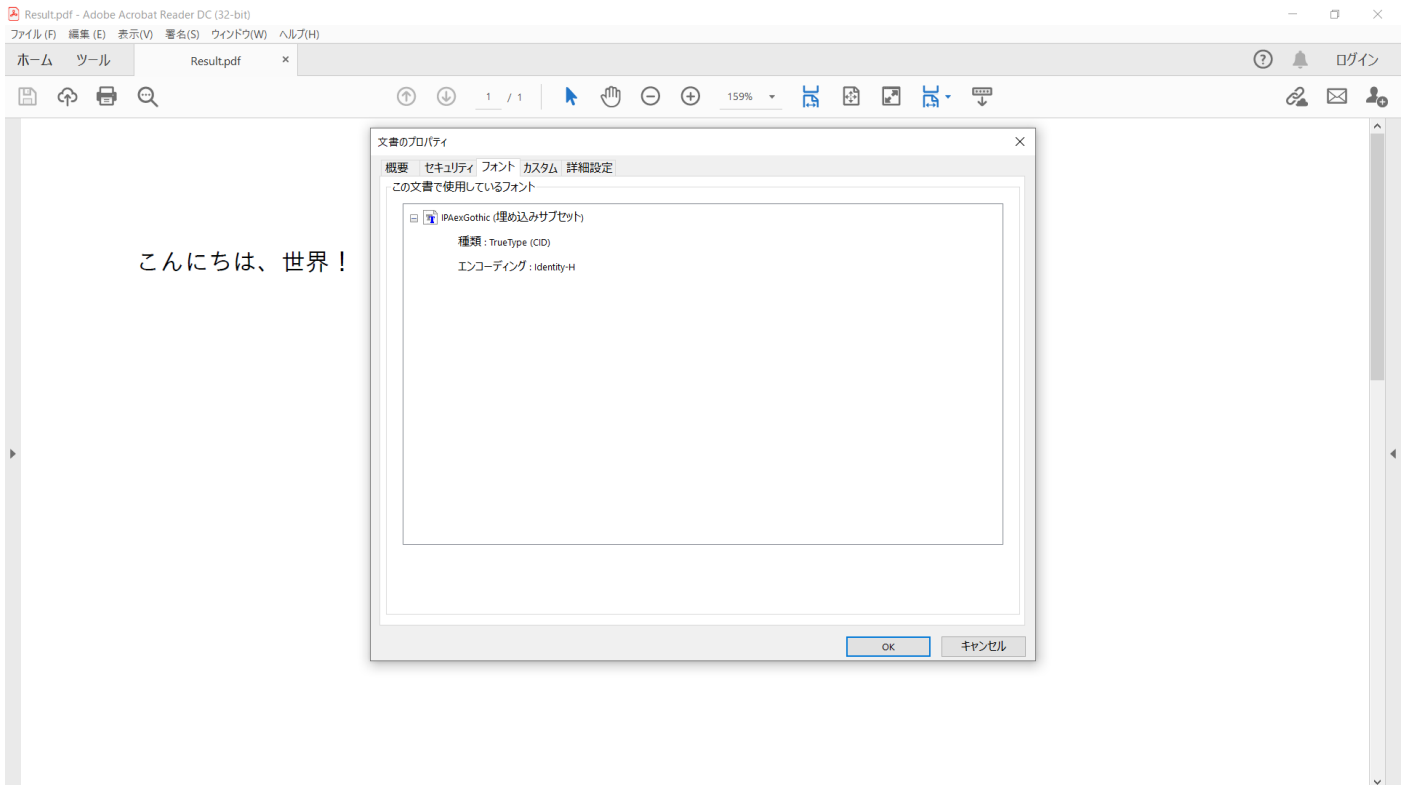
    workbook.Worksheets[0].Range["A1"].Value = Message;

    byte[] output;

    using (var ms = new MemoryStream())
    {
        workbook.Save(ms, SaveFileFormat.Pdf);
        output = ms.ToArray();
    }

    return new FileContentResult(output, "application/pdf")
    {
        FileNameDownload = "Result.pdf"
    };
}
}

```



PDF ドキュメントを保存する際の日本語フォント（DioDocs for PDF）

DioDocs for PDF で作成した PDF ドキュメントで日本語フォントを利用する場合も「作成したワークシートを PDF 出力する際の日本語フォントを設定する（DioDocs for Excel）」と同じような動作になります。

Azure Functions アプリケーションの実行環境（Windows）の `C:\Windows\Fonts` にインストールされている日本語フォントであれば `FontName` プロパティで設定するだけです。

例えば、`FontName` プロパティで「メイリオ」を設定して PDF ドキュメントを作成した場合、Azure Functions の実行環境の `C:\Windows\Fonts` には `meiry0.ttc` と `meiryob.ttc` が含まれていますので、追加で設定する必要は無くそのまま利用することができます。

```
public static class Function3
{
    [FunctionName("Function3")]
    public static async Task<IActionResult> Run(
        [HttpTrigger(AuthorizationLevel.Function, "get", "post", Route = null)] HttpRequest req,
        ILogger log)
    {
        log.LogInformation("C# HTTP trigger function processed a request.");

        string name = req.Query["name"];

        string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
        dynamic data = JsonConvert.DeserializeObject(requestBody);
```

```
name = name ?? data?.name;

string Message = string.IsNullOrEmpty(name)
    ? "こんにちは、世界！"
    : $"こんにちは、{name}！";

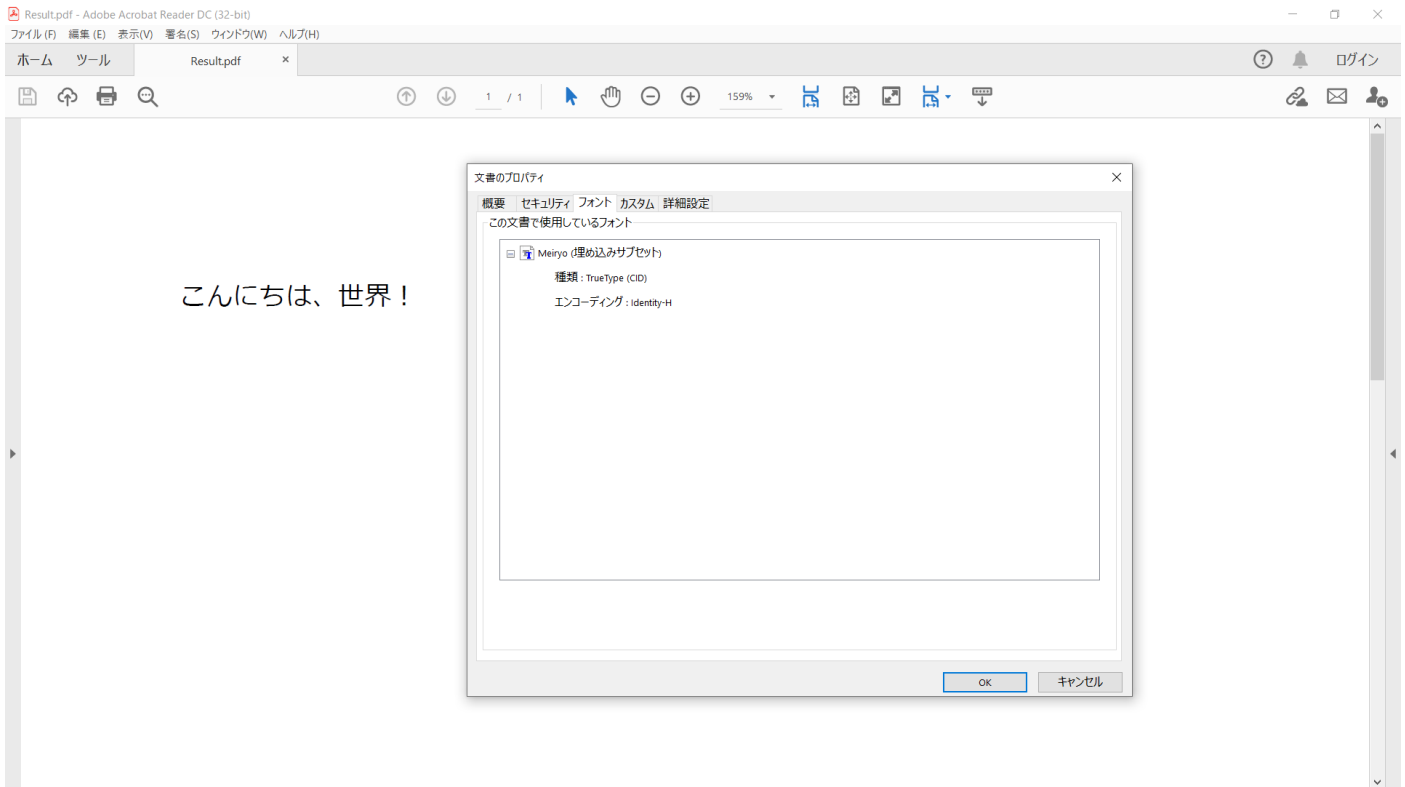
GcPdfDocument doc = new GcPdfDocument();
GcPdfGraphics g = doc.NewPage().Graphics;

g.DrawString(Message,
    new TextFormat() { FontName = "メイリオ", FontSize = 12 },
    new PointF(72, 72));

byte[] output;

using (var ms = new MemoryStream())
{
    doc.Save(ms, false);
    output = ms.ToArray();
}

return new FileContentResult(output, "application/pdf")
{
    FileNameDownload = "Result.pdf"
};
}
}
```

「IPAex フォント」のようなアプリケーションの実行環境 (Windows) の `C:\Windows\Fonts` に含まれていない日本語フォントを利用する場合は、日本語フォントをアプリケーションのコンテンツとして配置し、`Font` プロパティで参照する方法があります。

```
public static class Function3
{
    [FunctionName("Function3")]
    public static async Task<IActionResult> Run(
        [HttpTrigger(AuthorizationLevel.Function, "get", "post", Route = null)] HttpRequest req,
        Microsoft.Azure.WebJobs.ExecutionContext context,
        ILogger log)
    {
        log.LogInformation("C# HTTP trigger function processed a request.");

        string name = req.Query["name"];

        string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
        dynamic data = JsonConvert.DeserializeObject(requestBody);
        name = name ?? data?.name;

        string Message = string.IsNullOrEmpty(name)
            ? "こんにちは、世界！"
            : $"こんにちは、{name}！";

        GcPdfDocument doc = new GcPdfDocument();
```

```

GcPdfGraphics g = doc.NewPage().Graphics;

Font font = Font.FromFile(Path.Combine(context.FunctionAppDirectory, "fonts", "ipaexg.ttf"));

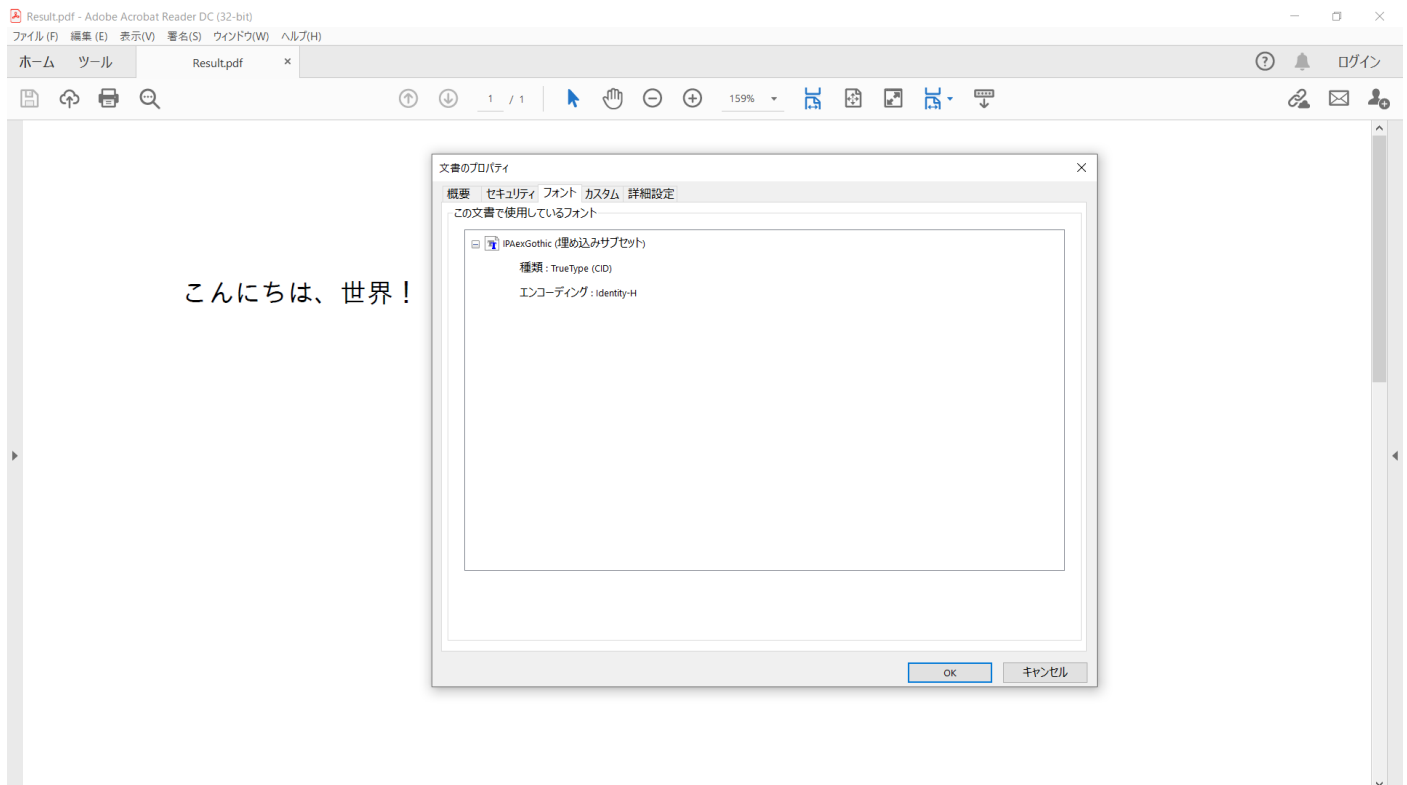
g.DrawString(Message,
    new TextFormat() { Font = font, FontSize = 12 },
    new PointF(72, 72));

byte[] output;

using (var ms = new MemoryStream())
{
    doc.Save(ms, false);
    output = ms.ToArray();
}

return new FileContentResult(output, "application/pdf")
{
    FileDownloadName = "Result.pdf"
};
}
}

```



日本語フォントの配置方法は「作成したワークシートを PDF 出力する際の日本語フォントを設定する (DioDocs

for Excel)」と同じです。

本記事では Azure Functions で DioDocs を利用する際に、日本語フォントを使用する Tips を紹介しましたが、Azure Functions の実行環境として Windows を選択した場合の Tips になります。Azure Functions の実行環境として Linux を選択した場合については今後の記事で紹介したいと思います。

弊社 Web サイトでは、製品の機能を気軽に試せるデモアプリケーションやトライアル版も公開していますので、こちらをご確認いただければと思います。

- [デモアプリケーション \(DioDocs for Excel\) を試す](#)
- [デモアプリケーション \(DioDocs for PDF\) を試す](#)
- [トライアル版をダウンロードして試す](#)

また、ご導入前の製品に関するご相談やご導入後の各種サービスに関するご質問など、お気軽にお問合せください。

- [問合せ先を確認する](#)
- [個別相談会 \(Web 会議\) について確認する](#)