

Azure Serverless Computing

Cookbook

Second Edition

Recipes for building and monitoring event-driven applications using
Azure Functions



Packt

www.packt.com

Praveen Kumar Sreeram

Azure Serverless Computing Cookbook

Second Edition

Recipes for building and monitoring event-driven applications using Azure Functions

Praveen Kumar Sreeram

Packt>

Azure Serverless Computing Cookbook

Second Edition

Commissioning Editor:
Acquisition Editor:
Content Development Editors:
Technical Editor:
Copy Editor: Safis Editing
Project Coordinator:
Proofreader: Safis Editing
Indexers:
Graphics:
Production Coordinator:

It would have not been possible to complete the book without the support of my best half, my wife, Haritha, and my cute little angel, Rithwika Sreeram



mapt.io

Why subscribe?

PacktPub.com

with PDF and ePub files available? You can upgrade to the eBook version at
www.PacktPub.com

service@packtpub.com

www.PacktPub.com

Contributors

About the author

Praveen Kumar Sreeram

He has over 14 years of experience in the field of development, analysis, design

Kumar Sreeram

Praveen

@PrawinSreeram

First of all, my thanks go to the Packt Publishing team, including Shrilekha Inani, Nithin George Varghese and Komal Karne.

I would like to thank my grandma, Neelavatamma, dad, Kamalakar and mom, Seetha; for being in my life and giving me courage all the time.

I would like to express my deepest gratitude to Bhagyamma (my grandmother), Kamala Kumar (my maternal uncle), his brothers and rest of the family, who have been supporting me all the time.

About the reviewers

Kasam Shaikh

with one of the leading IT companies in Mumbai, India. He is a certified Azure

First and foremost, I would like to thank the Almighty Allah, my family and especially my better half, for motivating me throughout this process. I am highly grateful to Packt Publishing for believing in me and for considering me for this opportunity.

Michael Sync (Soe Htike)

Packt is searching for authors like you

authors.packtpub.com

just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author

Table of Contents

| | |
|--|-------------|
| Preface | xiii |
| Chapter 1: Developing Cloud Applications Using Function Triggers and Bindings | 1 |
| Introduction | 1 |
| Building a backend Web API using HTTP triggers | 2 |
| Getting ready | 3 |
| How to do it... | 3 |
| How it works... | 8 |
| See also | 8 |
| Persisting employee details using Azure Storage table output bindings | 8 |
| Getting ready | 9 |
| How to do it... | 9 |
| How it works... | 13 |
| Understanding storage connection | 14 |
| What is the Azure Table storage service? | 15 |
| Partition key and row key | 15 |
| There's more... | 15 |
| Saving the profile images to Queues using Queue output bindings | 15 |
| Getting ready | 16 |
| How to do it... | 16 |
| How it works... | 18 |
| Storing the image in Azure Blob Storage | 18 |
| Getting ready | 18 |
| How to do it... | 18 |
| How it works... | 20 |
| There's more... | 21 |

| | |
|---|-----------|
| Chapter 2: Working with Notifications Using the SendGrid and Twilio Services | 23 |
| Introduction | 23 |
| Sending an email notification to the administrator of a website using the SendGrid service | 24 |
| Getting ready | 24 |
| Creating a SendGrid account | 24 |
| Generating an API key from the SendGrid portal | 27 |
| Configuring the SendGrid API key with the Azure Function app | 28 |
| How to do it... | 28 |
| Create Storage Queue binding to the HTTP Trigger | 29 |
| Create Queue Trigger to process the message of the HTTP Trigger | 30 |
| Create SendGrid output binding to the Queue Trigger | 31 |
| How it works... | 33 |
| There's more | 34 |
| Sending an email notification dynamically to the end user | 34 |
| Getting ready | 34 |
| How to do it... | 35 |
| Accept the new email Parameter in the RegisterUser function | 35 |
| Retrieve the UserProfile information in the SendNotifications trigger | 36 |
| How it works... | 37 |
| There's more... | 38 |
| Implementing email logging in Azure Blob Storage | 39 |
| How to do it... | 39 |
| How it works... | 41 |
| Modifying the email content to include an attachment | 41 |
| Getting ready | 42 |
| How to do it... | 42 |
| Customizing the log file name using IBinder interface | 42 |
| Adding an attachment to the email | 43 |
| Sending an SMS notification to the end user using the Twilio service | 44 |
| Getting ready | 45 |
| How to do it... | 47 |
| How it works... | 49 |
| Chapter 3: Seamless Integration of Azure Functions with Azure Services | 51 |
| Introduction | 51 |
| Using Cognitive Services to locate faces in images | 52 |
| Getting ready | 52 |
| Creating a new Computer Vision API account | 52 |
| Configuring application settings | 53 |

| | |
|--|-----------|
| How to do it... | 53 |
| How it works... | 60 |
| There's more... | 60 |
| Azure SQL Database interactions using Azure Functions | 61 |
| Getting ready | 61 |
| How to do it... | 63 |
| How it works... | 66 |
| Monitoring tweets using Logic Apps and notifying users when a popular user tweets | 66 |
| Getting ready | 67 |
| How to do it... | 67 |
| Creating a new Logic App | 67 |
| Designing the Logic App with Twitter and Gmail connectors | 69 |
| Testing the Logic App functionality | 73 |
| How it works... | 74 |
| Integrating Logic Apps with serverless functions | 74 |
| How to do it... | 75 |
| There's more... | 79 |
| See also | 80 |
| Auditing Cosmos DB data using change feed triggers | 80 |
| Getting ready | 80 |
| Creating a new Cosmos DB account | 81 |
| Creating a new Cosmos DB collection | 81 |
| How to do it... | 82 |
| How it works... | 86 |
| There's more... | 86 |
| Chapter 4: Understanding the Integrated Developer Experience of Visual Studio Tools | 87 |
| Introduction | 87 |
| Creating a function app using Visual Studio 2017 | 88 |
| Getting ready | 88 |
| How to do it... | 90 |
| How it works... | 92 |
| There's more... | 92 |
| Debugging C# Azure Functions on a local staged environment using Visual Studio 2017 | 92 |
| Getting ready | 93 |
| How to do it... | 93 |
| How it works... | 97 |
| There's more... | 97 |

| | |
|---|------------|
| Connecting to the Azure Storage cloud from the local Visual Studio environment | 98 |
| Getting ready | 98 |
| How to do it... | 98 |
| How it works... | 102 |
| There's more... | 102 |
| Deploying the Azure Function app to Azure Cloud using Visual Studio | 103 |
| How to do it... | 103 |
| There's more... | 107 |
| Debugging a live C# Azure Function, hosted on the Microsoft Azure Cloud environment, using Visual Studio | 107 |
| Getting ready | 108 |
| How to do it... | 108 |
| Deploying Azure Functions in a container | 111 |
| Getting ready | 112 |
| Creating an ACR | 113 |
| How to do it... | 114 |
| Creating a Docker image for the function app | 115 |
| Pushing the Docker image to the ACR | 116 |
| Creating a new function app with Docker | 118 |
| How it works... | 119 |
| Chapter 5: Exploring Tests Tools for the Validation of Azure Functions | 121 |
| Introduction | 121 |
| Testing Azure Functions | 122 |
| Getting ready | 122 |
| How to do it... | 123 |
| Testing HTTP triggers using Postman | 123 |
| Testing a Blob trigger using Microsoft Storage Explorer | 125 |
| Testing the Queue trigger using the Azure Management portal | 128 |
| There's more... | 131 |
| Testing an Azure Function on a staged environment using deployment slots | 131 |
| How to do it... | 132 |
| There's more... | 139 |
| Load testing Azure Functions using Azure DevOps | 139 |
| Getting ready | 140 |
| How to do it... | 140 |
| There's more... | 143 |
| See also | 144 |

| | |
|--|------------|
| Creating and testing Azure Functions locally using Azure CLI tools | 144 |
| Getting ready | 144 |
| How to do it... | 144 |
| Testing and validating Azure Function responsiveness using Application Insights | 147 |
| Getting ready | 148 |
| How to do it... | 149 |
| How it works... | 152 |
| There's more... | 152 |
| Developing unit tests for Azure Functions with HTTP triggers | 152 |
| Getting ready | 153 |
| How to do it... | 154 |
| Chapter 6: Monitoring and Troubleshooting Azure Serverless Services | 157 |
| Introduction | 157 |
| Troubleshooting your Azure Functions | 158 |
| How to do it... | 158 |
| Viewing real-time application logs | 158 |
| Diagnosing the entire function app | 160 |
| There's more... | 161 |
| Integrating Azure Functions with Application Insights | 163 |
| Getting ready | 163 |
| How to do it... | 164 |
| How it works... | 166 |
| There's more... | 166 |
| Monitoring your Azure Functions | 166 |
| How to do it... | 166 |
| How it works... | 168 |
| Pushing custom telemetry details to Application Insights Analytics | 168 |
| Getting ready | 170 |
| How to do it... | 170 |
| Creating an Application Insights function | 171 |
| Configuring access keys | 172 |
| Integrating and testing an Application Insights query | 174 |
| Configuring the custom derived metric report | 176 |
| How it works... | 178 |
| Sending application telemetry details via email | 179 |
| Getting ready | 179 |
| How to do it... | 180 |
| How it works... | 182 |

| | |
|---|------------|
| There's more... | 182 |
| See also | 182 |
| Integrating real-time Application Insights monitoring data with Power BI using Azure Functions | 182 |
| Getting ready | 183 |
| How to do it... | 184 |
| Configuring Power BI with a dashboard, a dataset and the push URI | 184 |
| Creating an Azure Application Insights real-time Power BI – C# function | 190 |
| How it works... | 193 |
| There's more... | 193 |
| Chapter 7: Developing Reliable Serverless Applications Using Durable Functions | 195 |
| Introduction | 195 |
| Configuring Durable Functions in the Azure Management portal | 196 |
| Getting ready | 196 |
| How to do it... | 197 |
| There's more... | 198 |
| Creating a Durable Function hello world app | 199 |
| Getting ready | 199 |
| How to do it... | 199 |
| Creating an HttpStart function in the Orchestrator client | 200 |
| Creating the Orchestrator function | 202 |
| Creating an activity function | 204 |
| How it works... | 205 |
| There's more... | 205 |
| Testing and troubleshooting Durable Functions | 205 |
| Getting ready | 206 |
| How to do it... | 206 |
| Implementing multithreaded reliable applications using Durable Functions | 208 |
| Getting ready | 208 |
| How to do it... | 209 |
| Creating the Orchestrator function | 209 |
| Creating a GetAllCustomers activity function | 210 |
| Creating a CreateBARCodeImagesPerCustomer activity function | 211 |
| How it works... | 213 |
| There's more... | 213 |
| Chapter 8: Bulk Import of Data Using Azure Durable Functions and Cosmos DB | 215 |
| Introduction | 215 |
| Business problem | 216 |

| | |
|---|------------|
| Durable serverless way of implementing an Excel import | 217 |
| Uploading employee data into Blob Storage | 217 |
| How to do it... | 218 |
| How it works... | 221 |
| There's more... | 222 |
| Creating a Blob trigger | 222 |
| Getting ready | 222 |
| How to do it... | 226 |
| There's more... | 226 |
| Creating the Durable Orchestrator and triggering it for each Excel import | 227 |
| How to do it... | 227 |
| How it works... | 230 |
| There's more... | 230 |
| Reading Excel data using activity functions | 231 |
| Getting ready | 231 |
| How to do it... | 232 |
| Read data from Blob Storage | 232 |
| Read Excel data from the stream | 233 |
| Create the activity function | 234 |
| There's more... | 236 |
| Auto-scaling Cosmos DB throughput | 237 |
| Getting ready | 237 |
| How to do it... | 239 |
| There's more... | 241 |
| Bulk inserting data into Cosmos DB | 241 |
| How to do it... | 241 |
| There's more... | 242 |
| Chapter 9: Implementing Best Practices for Azure Functions | 243 |
| Adding multiple messages to a queue using the IAsyncCollector function | 244 |
| Getting ready | 244 |
| How to do it... | 245 |
| How it works... | 247 |
| There's more... | 247 |
| Implementing defensive applications using Azure Functions and queue triggers | 247 |
| Getting ready | 248 |
| How to do it... | 248 |
| CreateQueueMessage – C# console application | 248 |
| Developing the Azure Function – queue trigger | 249 |
| Running tests using the console application | 250 |

| | |
|--|------------|
| How it works... | 251 |
| There's more... | 251 |
| Handling massive ingress using Event Hubs for IoT and other similar scenarios | 252 |
| Getting ready | 252 |
| How to do it... | 252 |
| Creating an Azure Function event hub trigger | 252 |
| Developing a console application that simulates IoT data | 253 |
| Avoiding cold starts by warming the app at regular intervals | 256 |
| Getting ready | 256 |
| How to do it... | 257 |
| Creating an HTTP trigger | 257 |
| Creating a timer trigger | 257 |
| There's more... | 258 |
| See also | 258 |
| Enabling authorisation for function apps | 258 |
| Getting ready | 258 |
| How to do it... | 259 |
| How it works... | 260 |
| There's more... | 260 |
| Controlling access to Azure Functions using function keys | 260 |
| How to do it... | 261 |
| Configuring the function key for each application | 261 |
| Configuring one host key for all the functions in a single function app | 262 |
| There's more... | 264 |
| Securing Azure Functions using Azure Active Directory | 264 |
| Getting ready | 265 |
| How to do it... | 265 |
| Configuring Azure AD to the function app | 265 |
| Registering the client app in Azure AD | 266 |
| Granting the client app access to the backend app | 269 |
| Testing the authentication functionality using a JWT token | 269 |
| Configuring throttling of Azure Functions using API Management | 271 |
| Getting ready | 272 |
| How to do it... | 273 |
| Integrating Azure Functions with API Management | 273 |
| Configuring request throttling using inbound policies | 276 |
| Testing the rate limit inbound policy configuration | 278 |
| How it works... | 279 |
| Securely accessing SQL Database from Azure Functions using Managed Service Identity | 280 |
| Getting ready | 280 |

| | |
|---|------------|
| How to do it... | 281 |
| Creating a function app using Visual Studio 2017 with V1 runtime | 281 |
| Creating a Logical SQL Server and a SQL Database | 284 |
| Enabling the managed service identity | 284 |
| There's more... | 287 |
| See also | 287 |
| Shared code across Azure Functions using class libraries | 287 |
| How to do it... | 288 |
| How it works... | 290 |
| There's more... | 291 |
| Using strongly typed classes in Azure Functions | 291 |
| Getting ready | 291 |
| How to do it... | 292 |
| How it works... | 294 |
| There's more... | 294 |
| Chapter 10: Configuring of Serverless Applications in the Production Environment | 295 |
| Introduction | 295 |
| Deploying Azure Functions using the Run From Package | 296 |
| Getting ready | 297 |
| How to do it... | 298 |
| How it works... | 299 |
| There's more... | 299 |
| Deploying Azure Function using ARM templates | 299 |
| Getting ready | 299 |
| How to do it... | 300 |
| There's more... | 303 |
| Configuring custom domain to Azure Functions | 303 |
| Getting ready | 304 |
| How to do it... | 304 |
| Configuring function app with an existing domain | 306 |
| Techniques to access Application Settings | 308 |
| Getting ready | 308 |
| How to do it... | 308 |
| Accessing Application Settings and connection strings in the Azure Function code | 308 |
| Application setting – binding expressions | 311 |
| Creating and generating open API specifications using Swagger | 311 |
| Getting ready | 312 |
| How to do it... | 312 |

| | |
|---|------------|
| Breaking down large APIs into small subsets of APIs using proxies | 316 |
| Getting ready | 316 |
| How to do it... | 317 |
| Creating microservices | 318 |
| Creating the gateway proxies | 318 |
| Testing the proxy URLs | 321 |
| There's more... | 321 |
| See also | 322 |
| Moving configuration items from one environment to another using resources | 322 |
| Getting ready | 323 |
| How to do it... | 324 |
| Chapter 11: Implementing and Deploying Continuous Integration Using Azure DevOps | 329 |
| Introduction | 329 |
| Prerequisites | 330 |
| Continuous integration – creating a build definition | 331 |
| Getting ready | 332 |
| How to do it... | 333 |
| How it works... | 337 |
| There's more... | 338 |
| Continuous integration – queuing a build and triggering it manually | 338 |
| Getting ready | 338 |
| How to do it... | 339 |
| Configuring and triggering an automated build | 341 |
| How to do it... | 342 |
| How it works... | 344 |
| There's more... | 344 |
| Continuous integration – executing unit test cases in the pipeline | 345 |
| How to do it... | 346 |
| There's more... | 348 |
| Creating a release definition | 348 |
| Getting ready | 349 |
| How to do it... | 350 |
| How it works... | 358 |
| There's more... | 358 |
| See also | 359 |

| | |
|---|------------|
| Triggering the release automatically | 360 |
| Getting ready | 360 |
| How to do it... | 360 |
| How it works... | 362 |
| There's more... | 362 |
| Other Books You May Enjoy | 363 |

Preface

The benefits of serverless computing span throughout the whole organisation

patterns worth mentioning here. While each of them shares the benefits stated above, they have also specific characteristics and additional benefits that make them unique and a better fit for certain scenarios. Here are the serverless

Low-code workflows for an easy and fast orchestration of combined tasks

integrations, nor learning any new APIs or specifications, with Logic Apps.

some use cases on the benefits and key features of Azure Functions. Then, we'll deep

real-world serverless use cases to guide you through configuring and setting up

Who this book is for

Azure Serverless Computing Cookbook

What this book covers

Chapter 1 Developing Cloud Applications Using Function Triggers and Bindings,

Chapter 2 Working with Notifications Using the SendGrid and Twilio Services

Chapter 3 Seamless Integration of Azure Functions with Azure Services

Chapter 4 Understanding the Integrated Developer Experience of Visual Studio Tools for Azure Functions

Chapter 5, Exploring Testing Tools for the Validation of Azure Functions

Finally, you will also learn how to configure alerts that notify you when your apps

Chapter 6 Monitoring and Troubleshooting Azure Serverless Services

Chapter 7 Developing Reliable Serverless Applications Using Durable Functions

Chapter 8 Bulk Import of Data Using Azure Durable Functions and Cosmos DB

Chapter 9 Implementing Best Practices for Azure Functions

Chapter 10 Configuring of Serverless Applications in the Production Environment demonstrates how to deploy a function app in an efficient way and copy/move the configurations in a smarter way so as to avoid human error. You will also learn how to configure a custom domain that you could share with your customers or end users

Chapter 11 Implementing and Deploying Continuous Integration Using Azure DevOps

To get the most out of this book

Download the example code files

You can download the example code files for this book from your account at <http://www.packtpub.com>
<http://www.packtpub.com/support> and register to have the files emailed directly

You can download the code files by following these steps:

<http://www.packtpub.com>

SUPPORT

Code Downloads & Errata

Search

Once the file is downloaded, please make sure that you unzip or extract the folder

WinRAR/7-Zip for Windows

Zipeg/iZip/UnRarX for Mac

7-Zip/PeaZip for Linux

The code bundle for the book is also hosted on GitHub at <https://github.com/PacktPublishing/Azure-Serverless-Computing-Cookbook-Second-Edition>

<https://github.com/PacktPublishing/>

Download the EPUB/mobi and example code files

An EPUB and mobi version of this book is available free of charge on GitHub.

Download the colour images

We also provide a PDF file that has colour images of the screenshots/diagrams

https://www.packtpub.com/sites/default/files/downloads/9781789615265_ColorImages.pdf

Conventions used

CodeInText

filenames, file extensions, pathnames, dummy URLs, user input and Twitter handles. Here is an example: “Mount the downloaded `WebStorm-10*.dmg` disk image file as another disk in your system.”

```
using System.Net;  
using Microsoft.AspNetCore.Mvc;  
using Microsoft.Extensions.Primitives;  
using Newtonsoft.Json;
```

```
using System.Net;  
using Microsoft.AspNetCore.Mvc;  
using Microsoft.Extensions.Primitives;  
using Newtonsoft.Json;
```

```
docker tag functionsindocker cookbookregistry.azurecr.io/  
functionsindocker:v1
```

Bold:

for example, in menus or dialogue boxes, also appear in the text like this. Here is an example: “During the installation, choose **Azure development Workloads** section.”



Sections

In this book, you will find several headings that appear frequently (*Getting ready*
How to do it... *How it works...* *There's more...* *See also*)

Getting ready

How to do it...

How it works...

There's more...

See also

Get in touch

General feedback feedback@packtpub.com

questions@packtpub.com

Errata

<http://www.packtpub.com/submit-errata>

Piracy

copyright@packtpub.com

If you are interested in becoming an author

<http://authors.packtpub.com>

Reviews

packtpub.com

1

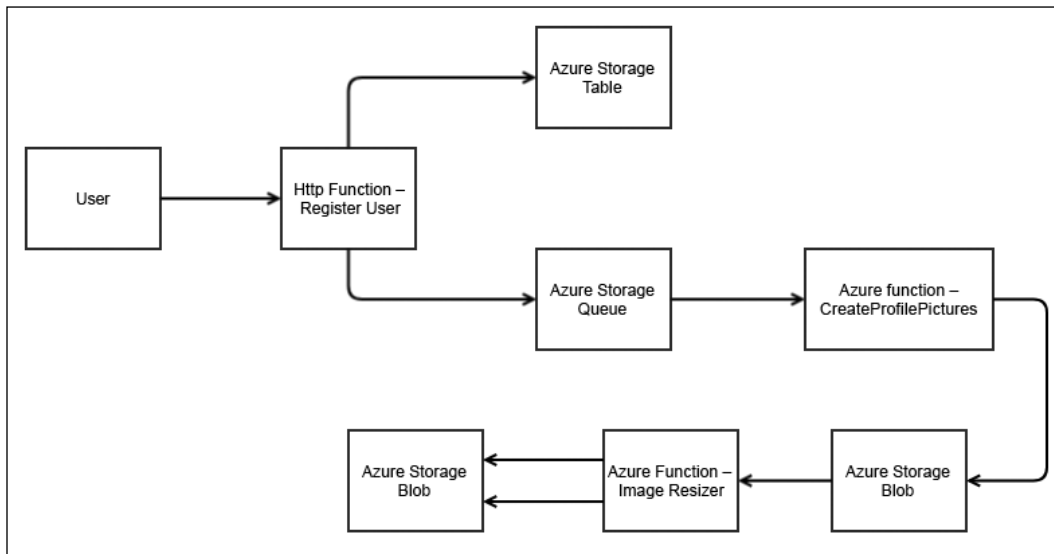
Developing Cloud Applications Using Function Triggers and Bindings

Introduction

as databases and filesystems. Each of these backend components could be developed using different technologies. Azure serverless technology also allows us to develop these backend APIs using Azure Functions.

problems, such as connecting to storage, building Web APIs and cropping images. In this chapter, we will learn how to use these built-in templates. Along with

for any organisation to manage the internal employee information.



Building a backend Web API using HTTP triggers

a Web API using HTTP triggers.

of making HTTP calls.


Getting ready

https://azure.microsoft.com/free/?wt.mc_id=AID607363_SEM_8y6Q27AS Account.

<https://docs.microsoft.com/azure/azure-functions/functions-create-function-app-portal>

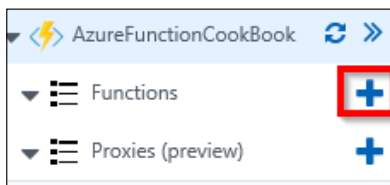
<https://docs.microsoft.com/azure/azure-functions/functions-create-first-azure-function>
create a function. While creating a function, a Storage Account is also created for storing all the files. Remember be used later in the other chapters.

work. I highly recommend you to go through the <https://docs.microsoft.com/azure/azure-functions/functions-triggers-bindings> article before you proceed.

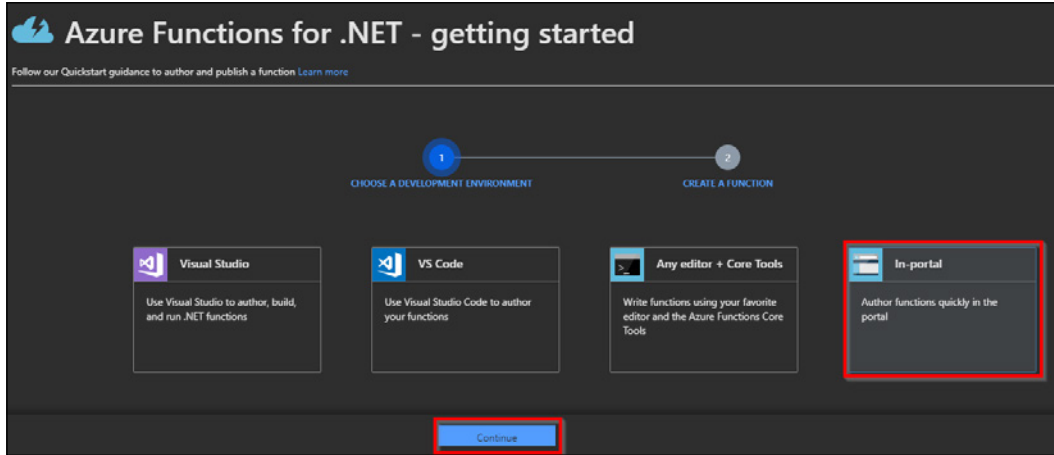

 book. Most of the functions are developed using Azure Functions V2 run-time. However, there are a few recipes which are not yet recipe. Hopefully, by the time you are read this book, Microsoft will have made those features available for V2 run-time as well.

How to do it...

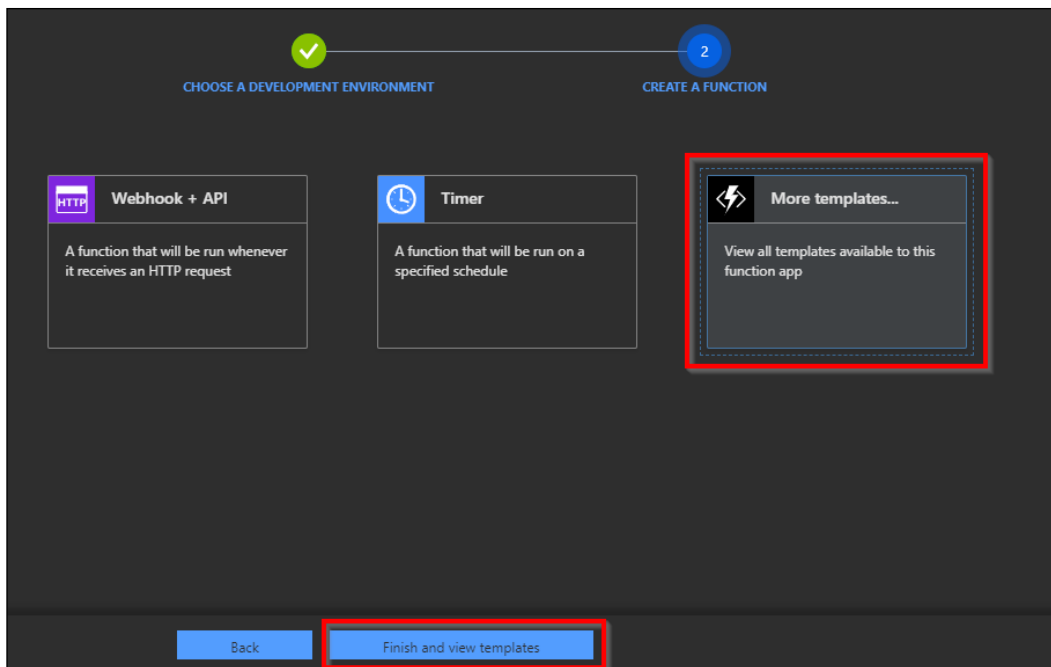
1. **Function App** **Function**
Apps menu, which is available on the left hand side.
- 2.



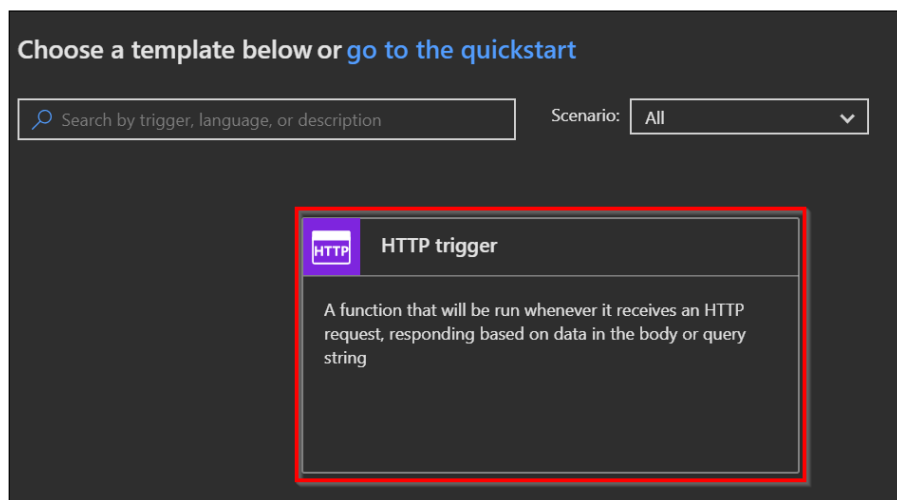
3. **Azure Functions for .NET – getting started**
you will be prompted to choose the type of tools you would like to use. You can choose the one you are interested in. For the initial few chapters, we will **In-portal** from the portal without any tools. Later, in the other chapters, we will use Visual Studio and Azure Functions Core Tools to create the Functions.:



4. **More templates** **Finish and view**
templates



5. Choose a template below or go to the quick-start HTTP trigger



6. meaningful name. For this RegisterUser
as the name of the Azure Function.

7. **Authorisation level** Anonymous option.
Chapter 9

Implementing Best Practices for Azure Functions



8. **Create** button to create the HTTP trigger function.

9. As soon as you create the function, all the required code and configuration files will be created automatically and the `run.csx` file will be opened for you to edit the code. Remove the default code and replace it with the following code. I have added two parameters (`firstname` `lastname`

```
#r "Newtonsoft.Json"

using System.Net;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Primitives;
using Newtonsoft.Json;

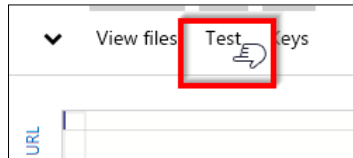
public static async Task<IActionResult> Run(
    HttpRequest req,
    ILogger log)
{
    log.LogInformation("C# HTTP trigger function processed
a request.");
    string firstname=null,lastname = null;
    string requestBody = await new StreamReader(req.Body).
ReadToEndAsync();

    dynamic inputJson = JsonConvert.DeserializeObject(requestBody);
    firstname = firstname ?? inputJson?.firstname;
    lastname = inputJson?.lastname;

    return (lastname + firstname) != null
        ? (ActionResult)new OkObjectResult($"Hello, {firstname + " " +
lastname}")
        : new BadRequestObjectResult("Please pass a name on the query" +
"string or in the request body");
}
```

10. **Save**
editor.

11. **Test** RegisterUser **Test** **Test** console. Click on

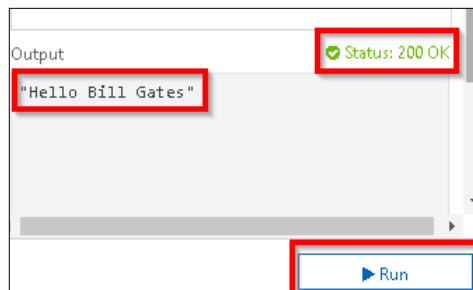


12. **Test** **Request body**



Make sure you select **POST** **HTTP method** drop-down.

13. **Test** **Run**



14.

Status 200 OK

Output

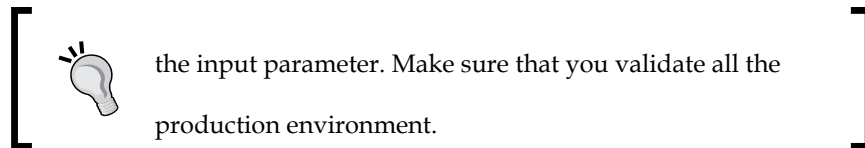
window will be as shown in the preceding screenshot.

How it works...

We have created the first basic Azure Function using HTTP triggers and made a few modifications to the default code. The code just accepts the `firstname` and `lastname`

```
Hello {firstname}
{lastname}
```

as a response. We also learned how to test the HTTP trigger function right from the Azure Management portal.



See also

*Enabling authorisation for function apps
Practices for Azure Functions*

Chapter 9 Implementing Best

Persisting employee details using Azure Storage table output bindings

input parameters. Let's now work on something interesting, that is, where you store the input data into a persistent medium. Azure Functions supports us to store data in many ways. For this example, we will store the data in Azure Table storage.

Getting ready

Azure Table storage using output bindings. The Azure HTTP trigger function receives the data from multiple sources and stores the user profile data in a `tblUserProfile`. We will follow the pre-requisites listed below:

previous recipe.

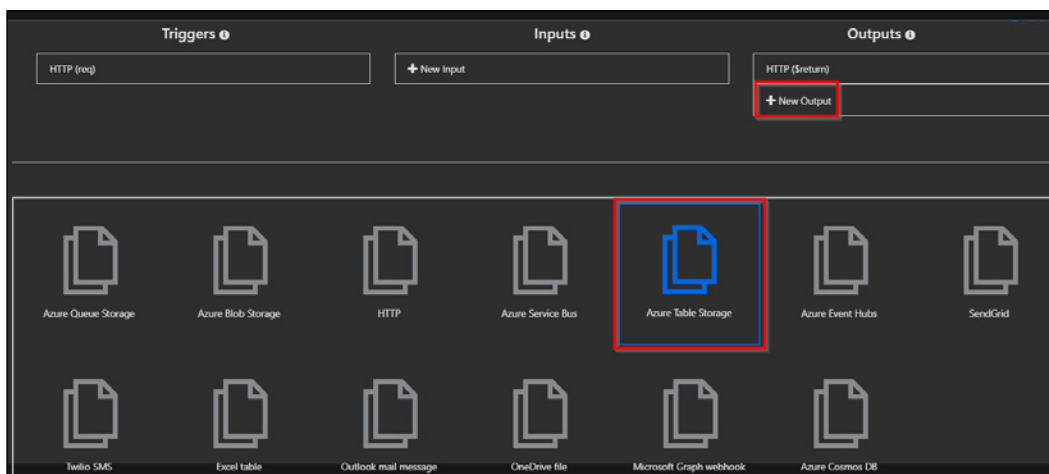
Azure Storage Explorer

in the Azure Storage account. You can download <http://storageexplorer.com/>.

<https://docs.microsoft.com/azure/vs-azure-tools-storage-manage-with-storage-explorer>.

How to do it...

1. **Integrate** `RegisterUser` HTTP trigger function.
2. **New Output** **Azure Table Storage**
Select



3. take a few minutes. Once the bindings are installed, choose the following
 - **Table parameter name**
Run method of the Azure Function. For this objUserProfileTable as the value.
 - **Table name**
to persist the data. If the table doesn't exist already, Azure will tblUserProfile as the table name.
 - **Storage account connection** **Storage account connection** **new** (as shown in the storage account.
 -

Azure Table Storage output [delete](#)

Table parameter name ⓘ
objUserProfileTable

Table name ⓘ
tblUserProfile

Use function return value

Storage account connection ⓘ [show value](#)
azurefunctionscookbooks_STORAGE ▼ *new*

4. **Save** to save the changes.
5. following code. The following code accepts the input passed by the end user

```
#r "Newtonsoft.Json"
#r "Microsoft.WindowsAzure.Storage"

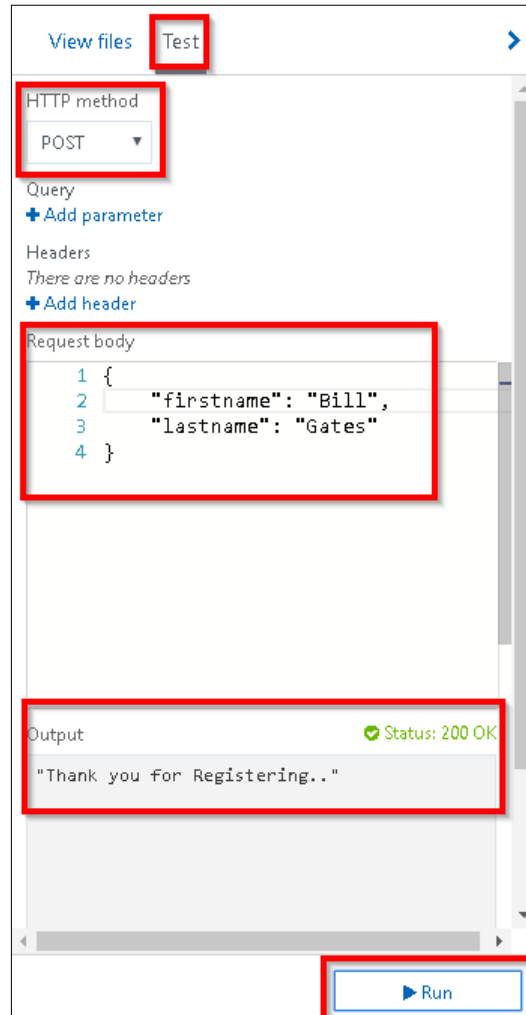
using System.Net;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Primitives;
using Newtonsoft.Json;
using Microsoft.WindowsAzure.Storage.Table;
```

```
public static async Task<IActionResult> Run(
    HttpRequest req,
    CloudTable objUserProfileTable,
    ILogger log)
{
    log.LogInformation("C# HTTP trigger function processed a
request.");
    string firstname=null,lastname = null;
    string requestBody = await new StreamReader(req.Body).
ReadToEndAsync();
    dynamic inputJson = JsonConvert.DeserializeObject(requestBody);
    firstname = firstname ?? inputJson?.firstname;
    lastname = inputJson?.lastname;
    UserProfile objUserProfile = new UserProfile(firstname,
lastname);
    TableOperation objTblOperationInsert = TableOperation.
Insert(objUserProfile);
    await objUserProfileTable.ExecuteAsync(objTblOperationInsert);
    return (lastname + firstname) != null
? (ActionResult)new OkObjectResult($"Hello, {firstname + " " +
lastname}")
: new BadRequestObjectResult("Please pass a name on the query" +
"string or in the request body");
}

class UserProfile : TableEntity
{
    public UserProfile(string firstName,string lastName)
    {
        this.PartitionKey = "p1";
        this.RowKey = Guid.NewGuid().ToString();
        this.FirstName = firstName;
        this.LastName = lastName;
    }
    UserProfile() { }
    public string FirstName { get; set; }
    public string LastName { get; set; }
}
```

6.

Run Test
firstname lastname Request body



7.

Status 200 OK

Output box as shown in the preceding screenshot. Let's navigate to Azure

tblUserProfile


| PartitionKey | RowKey | Timestamp | FirstName | LastName |
|--------------|---------------------------------------|---------------------------|-----------|----------|
| p1 | 411e5c64-0129-4393-b74a-b4e3395e37c88 | L2018-01-08T16:54:38.0347 | L.Bill | L.Gate< |

How it works...


adding an output binding to the trigger. For this example, we have integrated the HTTP trigger with the Azure Storage table binding and also configured the Azure

received by the HTTP trigger.

`objUserProfileTable` `CloudTable` `Run` method. We can perform `objUserProfileTable`.

[ The input parameters are not validated in the code sample.]
 kind of persisting medium.

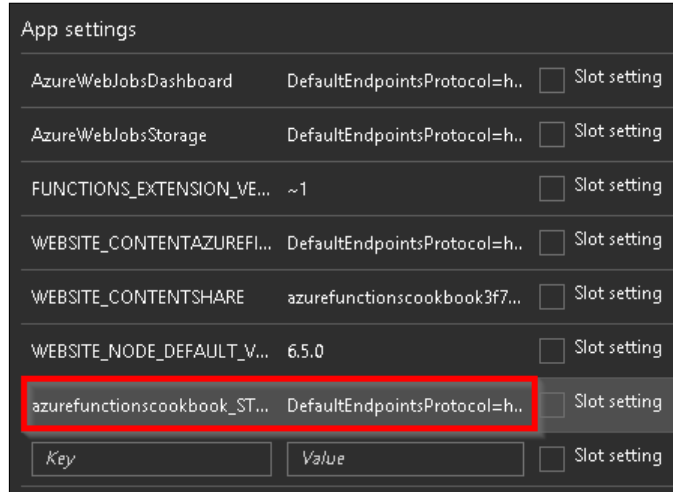
`UserProfile` object and filled it with the values received in the request object, and then passed it to a table operation.

[ <https://docs.microsoft.com/en-us/azure/storage/storage-dotnet-how-to-use-tables>.]

Understanding storage connection

storage connection (refer to the step 3 of the How to do it...

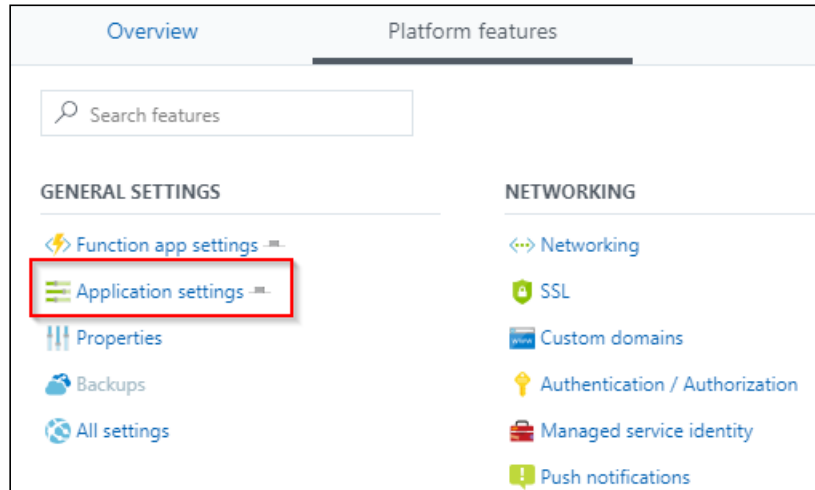
App settings



| Key | Value | Slot setting |
|------------------------------|------------------------------|--------------------------|
| AzureWebJobsDashboard | DefaultEndpointsProtocol=h.. | <input type="checkbox"/> |
| AzureWebJobsStorage | DefaultEndpointsProtocol=h.. | <input type="checkbox"/> |
| FUNCTIONS_EXTENSION_VE... | ~1 | <input type="checkbox"/> |
| WEBSITE_CONTENTAZUREFI... | DefaultEndpointsProtocol=h.. | <input type="checkbox"/> |
| WEBSITE_CONTENTSHARE | azurefunctionscookbook3f7... | <input type="checkbox"/> |
| WEBSITE_NODE_DEFAULT_V... | 6.5.0 | <input type="checkbox"/> |
| azurefunctionscookbook_ST... | DefaultEndpointsProtocol=h.. | <input type="checkbox"/> |
| Key | Value | <input type="checkbox"/> |

App settings GENERAL SETTINGS

Application settings Platform features



Overview | Platform features

Search features

GENERAL SETTINGS


- Function app settings
- Application settings**
- Properties
- Backups
- All settings

NETWORKING

- Networking
- SSL
- Custom domains
- Authentication / Authorization
- Managed service identity
- Push notifications

What is the Azure Table storage service?

semi-structured data.

[ <https://azure.microsoft.com/services/storage/tables/>]

Partition key and row key

Partition key

into partitions. Each record located in key (p1 in our example).

Row key value should be assigned to each of the rows.

There's more...

is the very first lines of the code in this recipe:

```
#r "Newtonsoft.Json"
#r "Microsoft.WindowsAzure.Storage"
```

to the specified library to the current context.

Saving the profile images to Queues using Queue output bindings

firstname lastname Request body ProfilePicUrl
storage. In this profile picture of the user that is publicly accessible via the internet. In this recipe,

container of an Azure Storage account.

ProfilePicUrl

Persisting

employee details using Azure Storage table output bindings. We didn't do it because the size of the profile pictures might be huge with the modern technology and so the processing of images on the fly in the HTTP requests might hinder the performance of the overall application. For that reason, we will just profile picture and store it in Queue, and later we can process the image and store it in the Blob.

Getting ready

RegisterUser

previous recipes.

How to do it...

1. **Integrate** RegisterUser HTTP trigger function.
2. **New Output Select** button. **Azure Queue Storage**
3. **Azure Queue Storage output**
 - **Message parameter name**
objUserProfileQueueItem Run
 - **Queue name**
userprofileimagesqueue
 - **Storage account connection:** Make sure that you select the right **Storage account connection**
4. **Save to the create the new output binding.**
5. (RegisterUser run.csx file and make the changes

```
public static async Task<IActionResult> Run(  
    HttpRequest req,  
    CloudTable objUserProfileTable,  
    IAsyncCollector<string> objUserProfileQueueItem,  
    ILogger log)  
{  
    ....  
}
```

```

string firstname= inputJson.firstname;
string profilePicUrl = inputJson.ProfilePicUrl;
await objUserProfileQueueItem.AddAsync(profilePicUrl);

....
objUserProfileTable.Execute(objTblOperationInsert);
}

```

6.

IAsyncCollector Run
 AddAsync
 ProfilePicUrl **Queue. Now, Click on Save**
 run.csx file.

7.

 ProfilePicUrl
Request body **Run** **Test**
 Function code editor window. The image used in the following JSON might not exist when you are reading this book. So, make sure that you provide a

```

{
  "firstname": "Bill",
  "lastname": "Gates",
  "ProfilePicUrl": "https://upload.wikimedia.org/wikipedia/commons/1/19/Bill_Gates_June_2015.jpg"
}

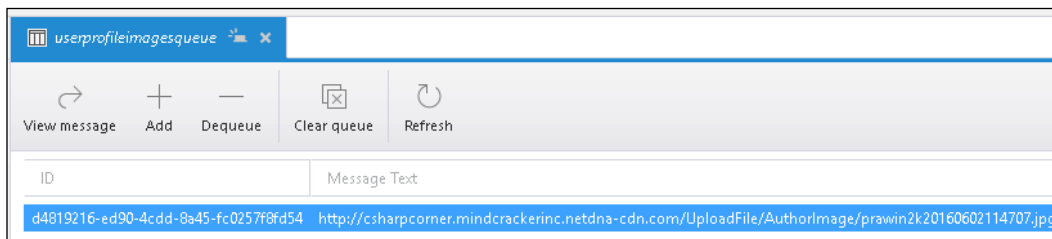
```

8. If everything goes fine you will see the **Status: 200 OK**

Request body
 will be created as a Queue message in the Azure Storage Queue service.

userprofileimagesqueue

step 3. The following is a screenshot of the Queue message that was created:



How it works...

```
out string objUserProfileQueueItem
```

```
        AddAsync                IAsyncCollector        Run  
which saves the profile URL to the Queue as a Queue message.
```

Storing the image in Azure Blob Storage

In the previous recipe, we stored the image URL in the queue message. Let's learn how to trigger an Azure Function (Queue Trigger) when a new queue item is added to the Azure Storage Queue service. Each profile picture of a user, which will be processed by the Azure Functions and will be as a Blob in the Azure Storage Blob service.

Getting ready

In the previous recipe, we learned how to create Queue output bindings. In this to a Blob.

This recipe is a continuation of the previous recipes. Make sure that you have implemented them.

How to do it...

1. **Azure Queue Storage Trigger**
from the templates.
2.
 - **Name your function**
CreateProfilePictures.

- **Queue name** `userprofileimagesqueue`. This will be monitored by the Azure Function. Our previous recipe trigger (named `RegisterUser` `userprofileimagesqueue` Queue. For each new entry of a queue message `CreateProfilePictures` trigger will be executed automatically.
 - **Storage account connection** where the Queues are located.
3. **Create** to create the new function.
 4. **Integrate** **New Output** **Azure Blob Storage** **Select** button.
 5. **Azure Blob Storage output**
 - **Blob parameter name** `outputBlob`
 - **Path** `userprofileimagecontainer/{rand-guid}`
 - **Storage account connection**
- Save**

Azure Blob Storage output (outputBlob) [delete](#)

Blob parameter name ⓘ Path ⓘ

Use function return value

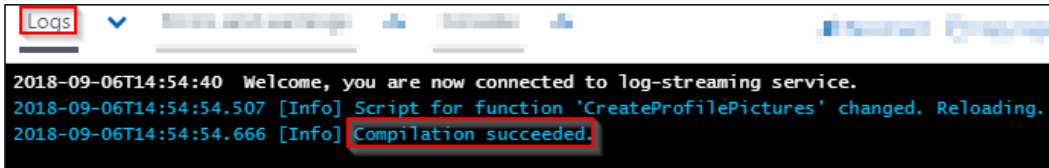
Storage account connection ⓘ [new](#)

6. **Save** button to save all the changes.
7. `run.csx` file of the `CreateProfilePictures` function with the following code. The following code grabs the URL from the

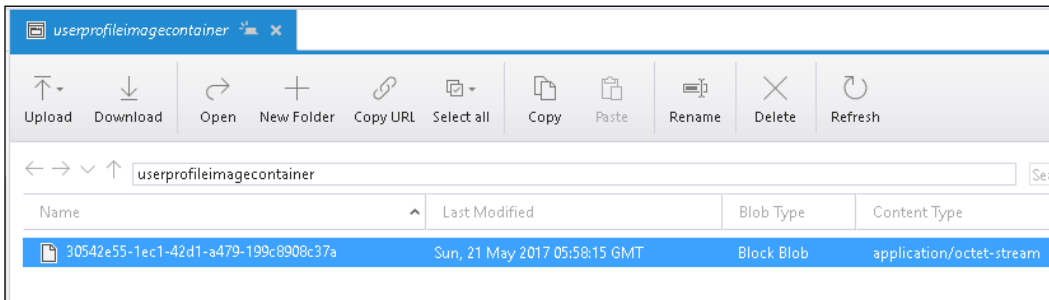
```
using System;
public static void Run(Stream outputBlob, string
myQueueItem,
    TraceWriter log)
{
    byte[] imageData = null;
    using (var wc = new System.Net.WebClient())
```

```
{
  imageData = wc.DownloadData(myQueueItem);
}
outputBlob.WriteAsync(imageData, 0, imageData.Length);
}
```

8. **Save** button to save the changes. Make sure that there are no **Logs**



9. `RegisterUser`
firstname lastname ProfilePicUrl fields as we did in the *Saving the profile images to Queues using Queue output bindings* recipe.
10. `userprofileimagecontainer` Blob container. You will find a new Blob:



11. You can view the image in any tool (such as MS Paint or Internet Explorer).

How it works...

arrives in the Queue. Once it finds a new
as we know the message is a URL of a profile picture. The function makes a web

data into the Blob which is configured as an output Blob.

There's more...

`rand-guid`
gets created each time the trigger is fired.



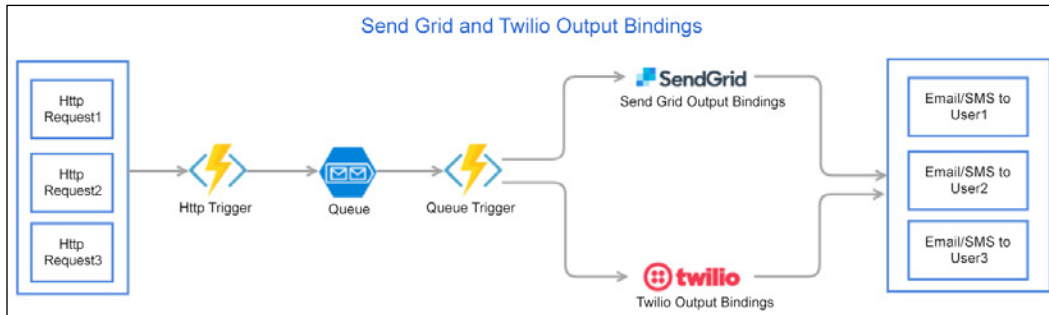
`Path`
of the Blob storage output binding while configuring the Blob storage output. Azure Functions creates one automatically if it doesn't exist.

that are up to 64 KB. If you would like to store messages greater than 64 KB, you need to use the Azure Service Bus.

2

Working with Notifications Using the SendGrid and Twilio Services

Introduction



Sending an email notification to the administrator of a website using the SendGrid service

SendGrid

email notification, containing static

To address field of the SendGrid output (message)

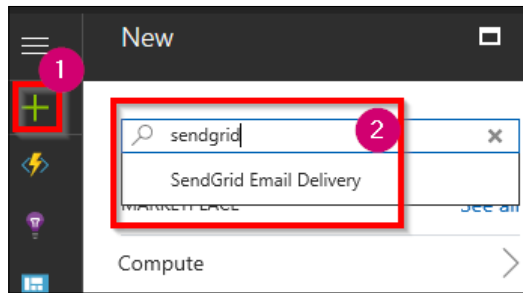
Getting ready

Configuring the SendGrid API key with the Azure Function app

Creating a SendGrid account

SendGrid Email

Delivery



SendGrid Email Delivery
Create a New SendGrid Account **free** **Create Pricing tier**
Create

Create a New SendGrid Account
CREATE

* Name
azurecookbook ✓

* Password ⓘ
..... ✓

* Confirm Password
..... ✓

* Subscription
Developer Program Benefit ▾

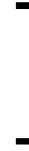
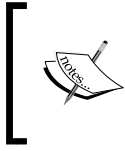
* Resource group ⓘ
 Create new Use existing
AzureFunctionCookBook ▾

* Pricing tier
free >

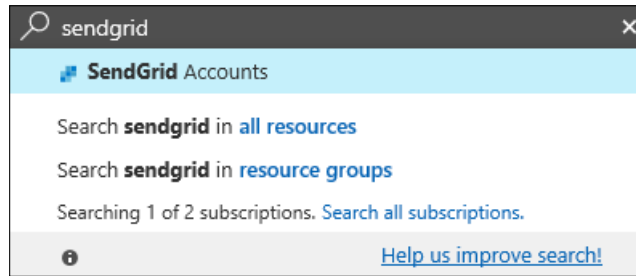
Promotion Code ⓘ
.....

Pin to dashboard

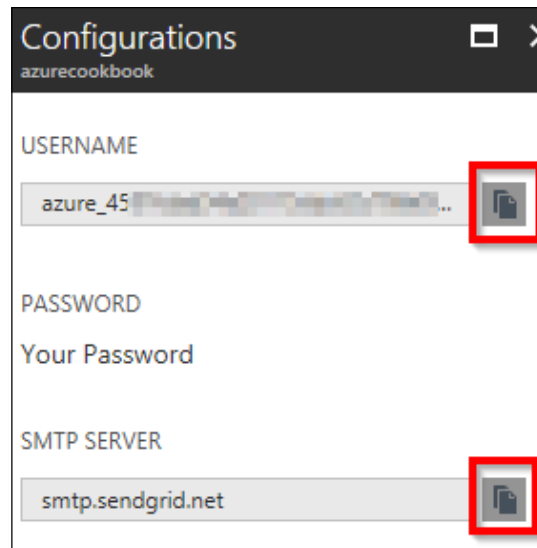
Create Automation options



SendGrid Accounts

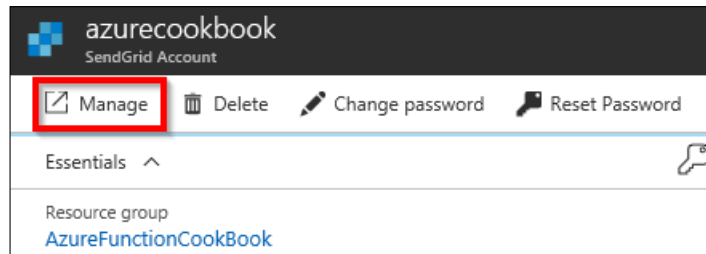


Settings **Configurations** **USERNAME**
SMTP SERVER **Configurations**

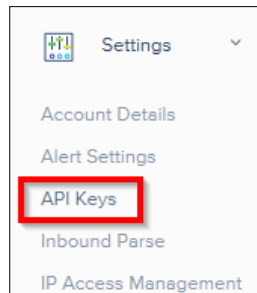


Generating an API key from the SendGrid portal

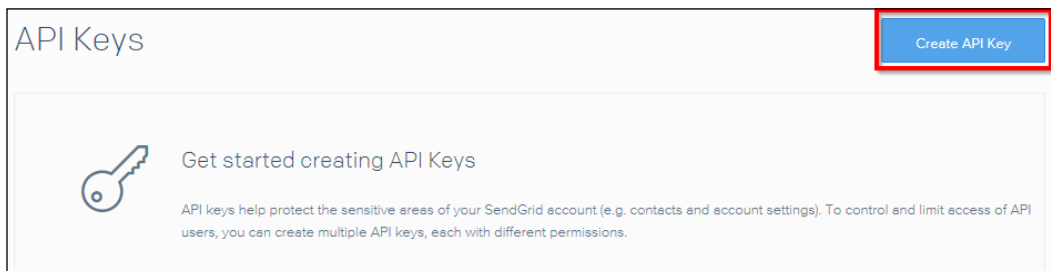
SendGrid Account Manage Essentials



API Keys Settings



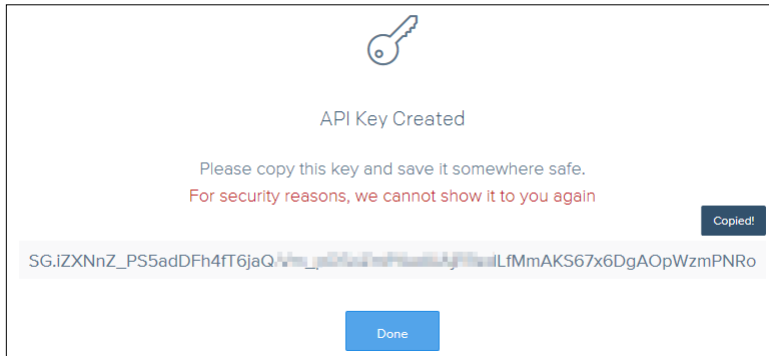
API Keys Create API Key



Create API Key
Permissions

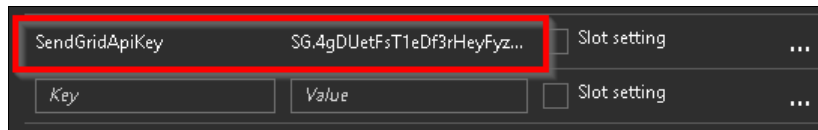
Create & View

API Key



Configuring the SendGrid API key with the Azure Function app

App settings configuration in the Azure Function app by
Application settings **Platform features**



Save

App settings

How to do it...

Create Storage Queue binding to the HTTP Trigger

Integrate RegisterUser
 New Output
 Azure Queue Storage Select
 Save notificationqueue

Run RegisterUser

```
NotificationQueueItem.AddAsync("");

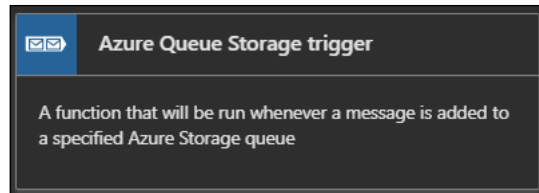
public static async Task<IActionResult> Run(
    HttpRequest req,
    CloudTable objUserProfileTable,
    CloudTable objUserProfileQueueItem,
    IAsyncCollector<string> NotificationQueueItem,
    ILogger log)
{
    log.LogInformation("C# HTTP trigger function processed a request.");
    string firstname=null,lastname = null;
    ...
    ...
    await NotificationQueueItem.AddAsync("");
    return (lastname + firstname) != null
        ? (ActionResult)new OkObjectResult($"Hello, {firstname + " " + lastname}")
        : new BadRequestObjectResult("Please pass a name on the
```



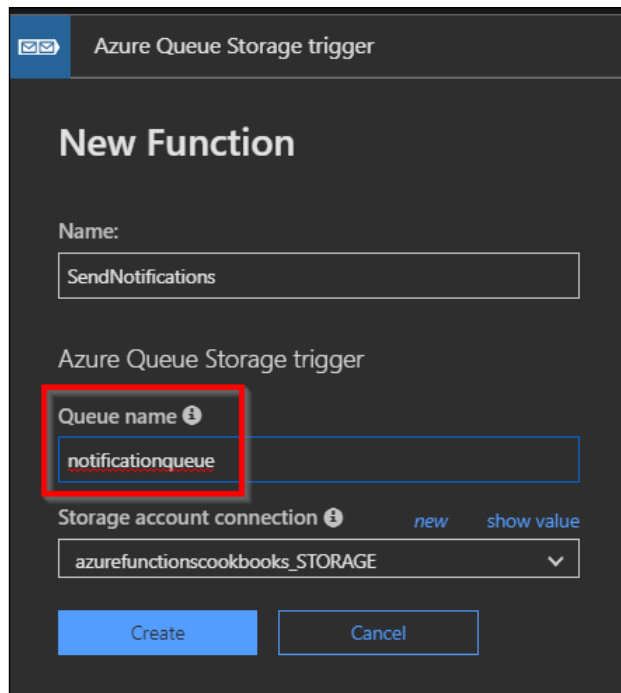
```
query" +  
    "string or in the request body");  
}
```

Create Queue Trigger to process the message of the HTTP Trigger

Azure Queue Storage



name of the queue which needs to be monitored for sending the notifications.

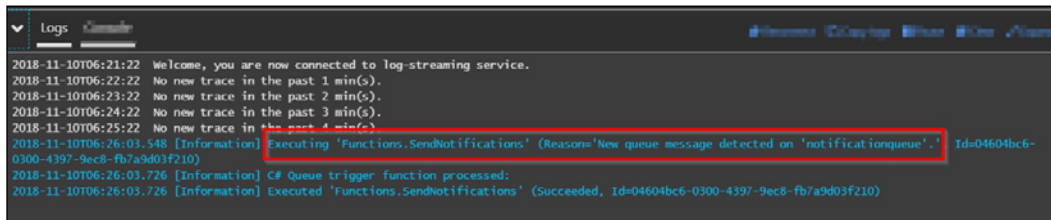


RegisterUser

RegisterUser

Run

SendNotifications



SendGrid

Create SendGrid output binding to the Queue Trigger

Integrate tab of the SendNotifications function and click **New Output**

SendGrid

Select

SendGrid

Install

SendGrid output (message)

- **Message parameter name** message
- **SendGrid API Key** Run
- **To address** **App settings**
- **From address** donotreply@example.com

- **Message subject**
- **Message Text**

SendGrid output (message)
the fields:

SendGrid output

Message parameter name ⓘ
message

Use function return value

To address ⓘ
prawin2k@gmail.com

Message subject ⓘ
New User got Registered Successfully

SendGrid API Key App Setting ⓘ show value
SendGrid-APKey new

From address ⓘ
donotreply@example.com

Message Text ⓘ
Hi Admin, A new user got registered successfully, Than

Save Cancel

Save

Run

SendNotifications

- SendGrid.
- Helpers.Mail
- SendGridMessage
- SendGridMessage

Run

```
#r "SendGrid"
using System;
using SendGrid.Helpers.Mail;

public static void Run(string myQueueItem,out SendGridMessage
message, ILogger log)
{
    log.LogInformation($"C# Queue trigger function processed:
```

```
{myQueueItem}");  
    message = new SendGridMessage();  
}
```

RegisterUser

```
{  
    "firstname": "Bill",  
    "lastname": "Gates",  
    "ProfilePicUrl": "https://upload.wikimedia.org/  
wikipedia/commons/thumb/1/19/  
Bill_Gates_June_2015.jpg/220px-  
Bill_Gates_June_2015.jpg"  
}
```

How it works...

send a notification via email to the administrator,

SendGrid

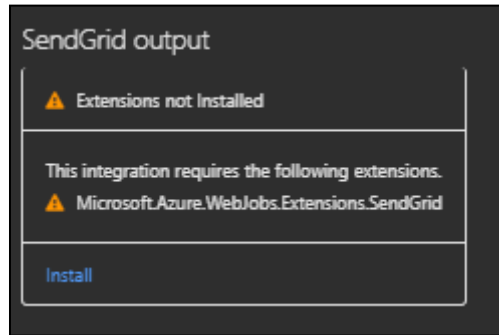
Simple Mail Transfer Protocol SMTP

SendGrid output (message)

SendGrid output (message)
App settings

SendGrid

There's more



<https://docs.microsoft.com/azure/azure-functions/install-update-binding-extensions-manual>

Sending an email notification dynamically to the end user

Thank you for registration

Getting ready

Make sure that the following are configured properly:

App settings

Application settings

App settings

SendGrid output (message)

How to do it...

run.csx file of the following

```
RegisterUser
SendNotifications
```

Accept the new email Parameter in the RegisterUser function

```
RegisterUser                                     run.csx file, add a new string
                                                email           request
                                                UserProfile

string firstname=null,lastname = null, email = null;
...
...
string email = inputJson.email;
...
...
UserProfile objUserProfile = new UserProfile(firstname,
lastname,email);
...
...
await
NotificationQueueItem.AddAsync(JsonConvert.SerializeObject(objUser
Profile));
```

UserProfile

Save

```
public class UserProfile : TableEntity
{
    public UserProfile(string firstname,string lastname,
string          profilePicUrl,string email)
    {
        ....
        ....
        this.ProfilePicUrl = profilePicUrl;
        this.Email = email;
    }
    ....
    ....
    public string ProfilePicUrl {get; set;}
```

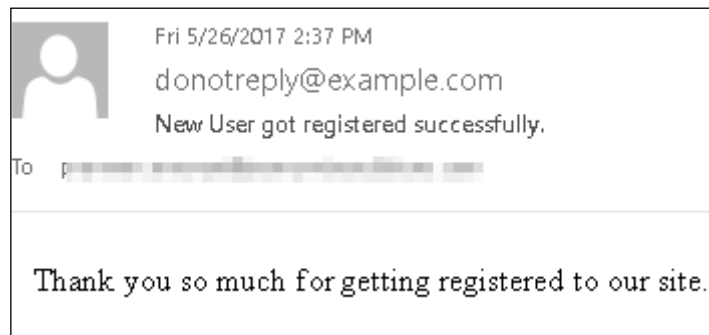
```
        public string Email { get; set; }  
    }  
}
```

Retrieve the UserProfile information in the SendNotifications trigger

```
        SendNotifications                                run.csx file, add  
Newtonsoft.Json  
  
        JsonConvert.DeserializeObject  
  
        SendGridMessage  
  
#r "SendGrid"  
#r "Newtonsoft.Json"  
using System;  
using SendGrid.Helpers.Mail;  
using Newtonsoft.Json;  
  
public static void Run(string myQueueItem,  
                        out SendGridMessage message,  
                        ILogger log)  
{  
    log.LogInformation($"C# Queue trigger function processed:  
    {myQueueItem}");  
    dynamic inputJson = JsonConvert.DeserializeObject(myQueueItem);  
    string FirstName=null, LastName=null, Email = null;  
    FirstName=inputJson.FirstName;  
    LastName=inputJson.LastName;  
    Email=inputJson.Email;  
    log.LogInformation($"Email{inputJson.Email}, {inputJson.FirstName  
+ " " + inputJson.LastName}");  
    message = new SendGridMessage();  
    message.SetSubject("New User got registered successfully.");  
    message.SetFrom("donotreply@example.com");  
    message.AddTo(Email,FirstName + " " + LastName);  
    message.AddContent("text/html", "Thank you " + FirstName + " " +  
    LastName +" so much for getting registered to our site.");  
}
```

Let's run a test by adding a new input field email to the test request payload,

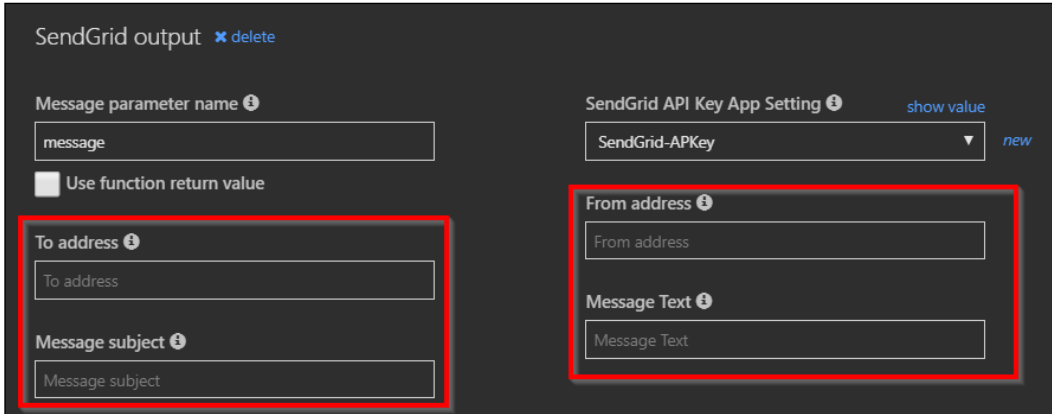
```
{
  "firstname": "Praveen",
  "lastname": "Sreeram",
  "email": "example@gmail.com",
  "ProfilePicUrl": "A Valid url here"
}
```





How it works...

```
RegisterUser
email
email
the SendGrid API. We have also configured all the other parameters, such as the
From Subject
```


We can also clear the fields in the **SendGrid output**

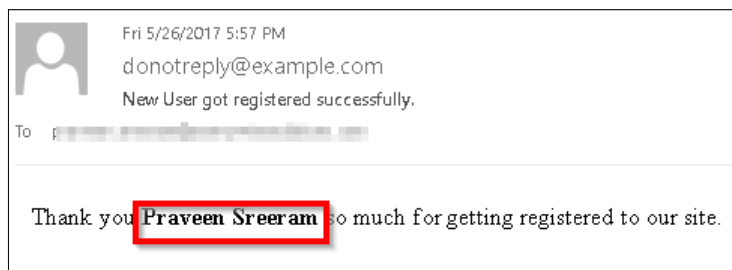


 The values specified in the code will take precedence over the values specified in the preceding step. 

There's more...

(

```
message.AddContent("text/html", "Thank you <b>" + FirstName + "</b><b>" + LastName + " </b>so much for getting registered to our site.");
```



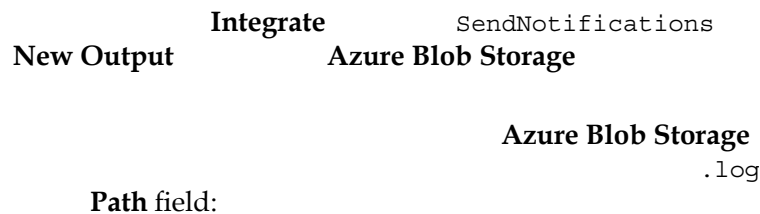
Implementing email logging in Azure Blob Storage

emails containing notifications, alerts and so on, to the end user. At times, users error in the application while sending such notification alerts.

of the email service providers has different spam filters that might block the emails

In this recipe, you will learn how to create a new email log file with the `.log` extension for each new registration. This log file can be used as a redundancy for files as a Blob in a storage container, alongside the data entered by the end user

How to do it...



Blob parameter name `outputBlob`

Path `userregistrationemaillogs/{rand-guid}.log`

Use function return value

Storage account connection `azurefunctionscookbooks_STORAGE` show value new

Save **Cancel**

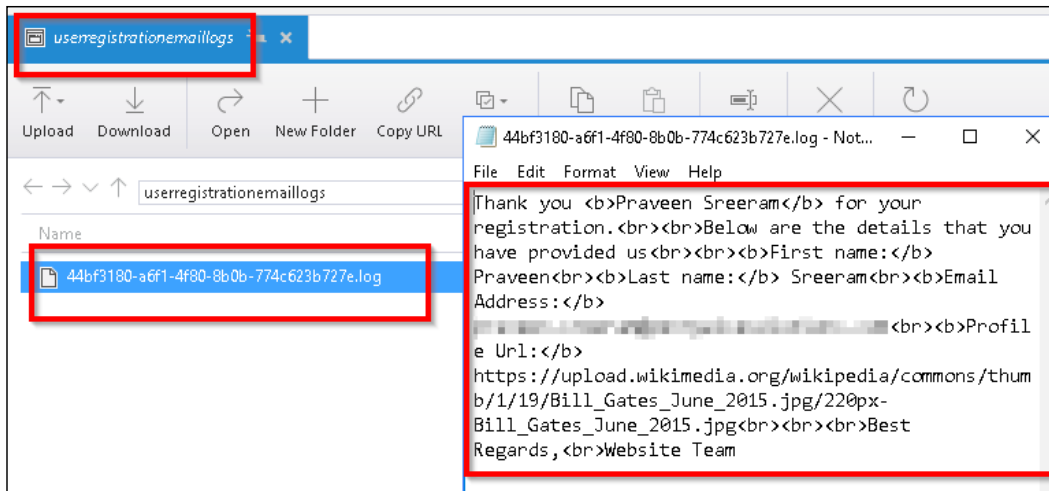
run.csx file of the SendNotifications

```
outputBlob    TextWriter    Run
              emailContent
```

Request body

```
public static void Run(string myQueueItem,
                       out SendGridMessage message,
                       TextWriter outputBlob,
                       ILogger log)
{
    ....
    ....
    string FirstName=null, LastName=null, Email = null;
    string emailContent;
    ....
    ....
    emailContent = "Thank you <b>" + FirstName + " " +
                   LastName + "</b> for your registration.<br><br>" +
                   "Below are the details that you have provided
    us<br> <br>" + "<b>First name:</b> " +
                   FirstName + "<br>" + "<b>Last name:</b> " +
                   LastName + "<br>" + "<b>Email Address:</b> " +
                   inputJson.Email + "<br><br> <br>" + "Best Regards, " +
    "<br>" + "Website Team";
    message.AddContent (new
        Content ("text/html", emailContent));
    outputBlob.WriteLine (emailContent);
}
```

After running the test, the log file will be created in the container named `userregistrationemaillogs`



How it works...

.log file (note that you can use any other extension as well) that is stored as a Blob in the container specified **Path** field of the output bindings.

Modifying the email content to include an attachment

In this recipe, you will learn how to send a file user. In our previous recipe, we created a log file of the email content. We will send the same file as an attachment to the email. However, in real-world applications, you might not intend to send log files to the end user. For the sake of simplicity, we will send the log file as an attachment.



Getting ready

This is a continuation of the previous recipe. If you are reading this first, make sure

How to do it...

Make the changes to the code to create a log file with the RowKey of the
IBinder
Send this file as an attachment to the email.

Customizing the log file name using IBinder interface

```
run.csx file of the SendNotifications
    StreamWriter
IBinder                                     Run

#r "SendGrid"
#r "Newtonsoft.Json"
#r "Microsoft.Azure.WebJobs.Extensions.Storage"
using System;
using SendGrid.Helpers.Mail;
using Newtonsoft.Json;
using Microsoft.Azure.WebJobs.Extensions.Storage;
public static void Run(string myQueueItem,
                       out SendGridMessage message,
                       IBinder binder,
                       ILogger log)

    StreamWriter          outputBlob.
WriteLine(emailContent);

    using (var emailLogBloboutput = binder.
Bind<StreamWriter>(new
    BlobAttribute($"userregistrationemaillogs/
{objInsertedUser.RowKey}.log")))
    {
        emailLogBloboutput.WriteLine(emailContent);
    }
}
```

You can see the email log file that is created using the RowKey of the

| PartitionKey | RowKey | Timestamp |
|--------------|--------------------------------------|--------------------------|
| p1 | 782601c1-8863-4f9f-9911-f681ed33674d | 2017-06-03T10:34:05.641Z |

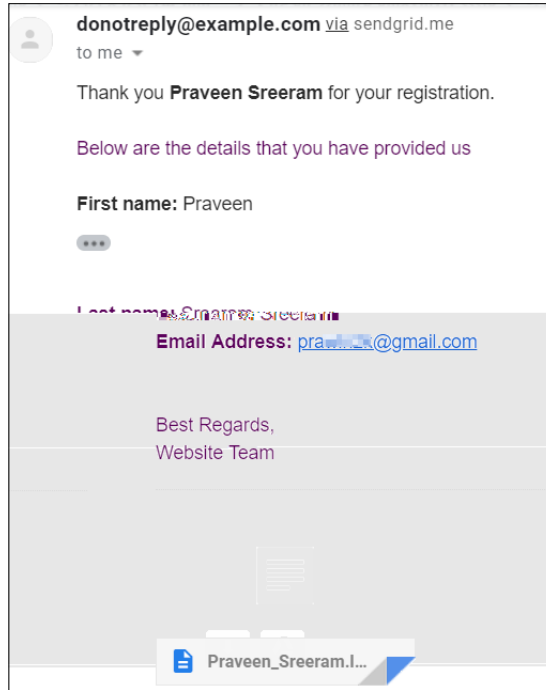
| Name | Last Modified |
|--|-------------------------------|
| 782601c1-8863-4f9f-9911-f681ed33674d.log | Sat, 03 Jun 2017 10:34:05 GMT |


Adding an attachment to the email

```

Run          SendNotifications
            Save
message.AddAttachment(FirstName + "_" + LastName + ".log",
    System.Convert.
ToBase64String(System.Text.Encoding.UTF8.GetBytes(emailContent)),
    "text/plain",
    "attachment",
    "Logs"
);


```



[ https://sendgrid.com/docs/API_Reference/api_v3.html]

Sending an SMS notification to the end user using the Twilio service

send notifications via SMS, using one of the leading cloud communication platforms,

[ <https://www.twilio.com/>]

Getting ready

Twilio SMS output (objsmsmessage)

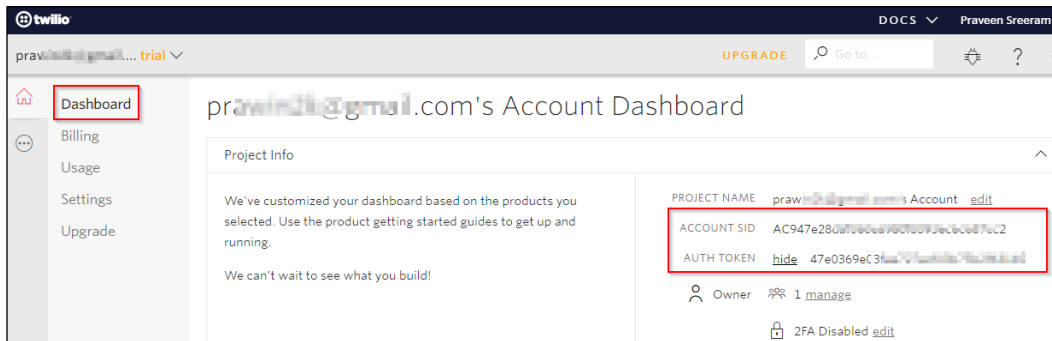
<https://www.twilio.com/try-twilio>

AUTH TOKEN

**Dashboard
App settings**

ACCOUNT SID

Application settings

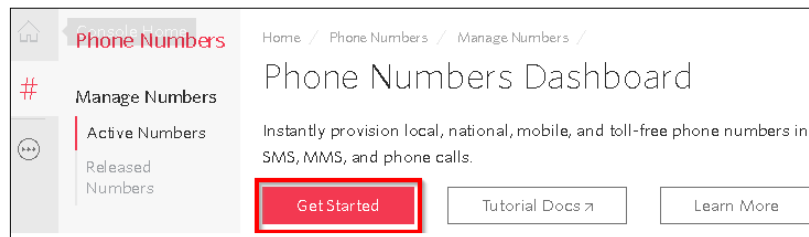


From number

Phone

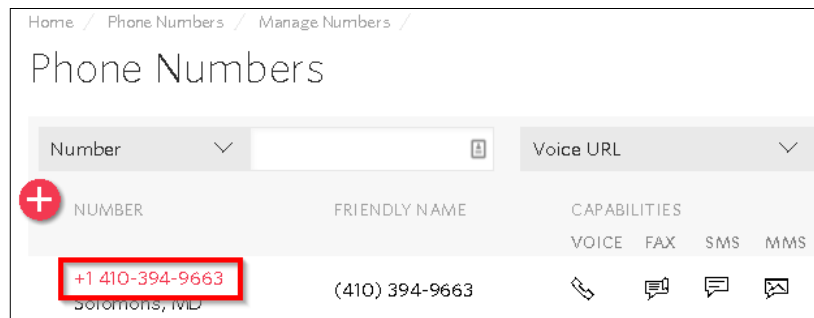
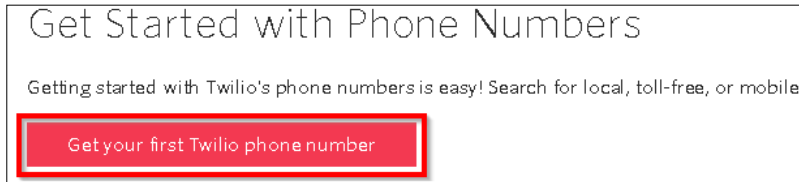
Numbers Dashboard
[phone-numbers/incoming](https://www.twilio.com/console/phone-numbers/incoming)

[https://www.twilio.com/console/](https://www.twilio.com/console/phone-numbers/incoming)
Get Started



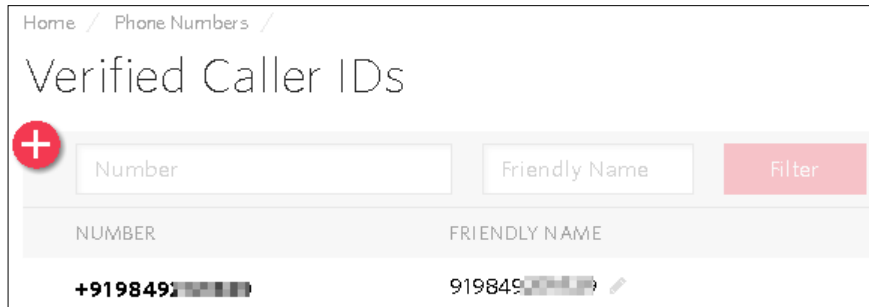
Get Started with Phone Numbers Twilio phone number

Get your first



The final step is to verify a number to which you would like to send an

number on Twilio's Verified page, available at <https://www.twilio.com/console/phone-numbers/verified>
of verified numbers:



How to do it...

Application settings TwilioAccountSID TwilioAuthToken

| | |
|------------------|-----------------------|
| TwilioAccountSID | AC947e28[REDACTED]e.. |
| TwilioAuthToken | 47e0369e0[REDACTED] |

Integrate tab of the SendNotifications function, click on **New Output Twilio SMS**

Select

**Twilio SMS output
From number**

Getting ready

Twilio SMS output [✕ delete](#)

Message parameter name ⓘ

Use function return value

Auth Token setting ⓘ

Message text ⓘ

Account SID setting ⓘ

From number ⓘ

To number

```

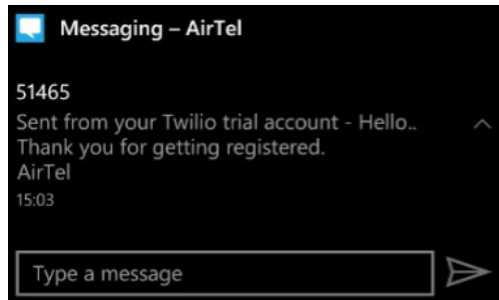
...
...
#r "Twilio"
#r "Microsoft.Azure.WebJobs.Extensions.Twilio"
...
...
using Microsoft.Azure.WebJobs.Extensions.Twilio;
using Twilio.Rest.Api.V2010.Account;
using Twilio.Types;

```

```
public static void Run(string myQueueItem,
    out SendGridMessage message,
    IBinder binder,
    out CreateMessageOptions objsmsmessage,
    ILogger log)
...
...
...
message.AddAttachment(FirstName + "_" + LastName + ".log",
    System.Convert.ToBase64String(System.Text.Encoding.UTF8.
GetBytes(emailContent)),
    "text/plain",
    "attachment",
    "Logs"
    );

    objsmsmessage = new CreateMessageOptions(new
PhoneNumber("+91 98492*****"));
    objsmsmessage.Body = "Hello.. Thank you for getting
registered.";
}
```

RegisterUser



How it works...

App settings

<https://www.youtube.com/watch?v=ndxQXnoDIj8>

3

Seamless Integration of Azure Functions with Azure Services

Introduction

applications from scratch for each of your business needs. You would first need to research the existing systems and see whether they fit your business requirements.

Using Cognitive Services to locate faces in images

Getting ready

To get started, we need to create a Computer Vision API and configure its API keys

Make sure that you have Azure Storage Explorer installed and have also configured

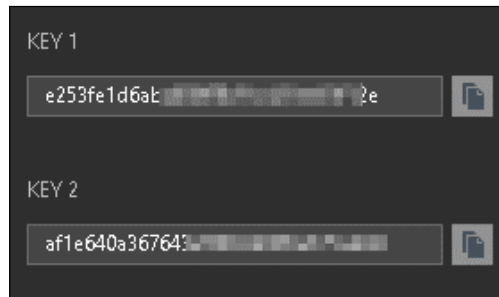
Creating a new Computer Vision API account

`computer vision` and click on **Create**

F0

Configuring application settings

Keys blade and grab any of the following keys:



Navigate to your Azure functions app, configure **Application settings**
`Vision_API_Subscription_Key`
keys as the value. This key will be used by the Azure Functions Runtime

Make a note

West Europe

of the API starts with the location name. It would be something like this:
`https://westeurope.api.cognitive.microsoft.com/vision/v1.0/analyze?visualFeatures=Faces&language=en`

How to do it...

Storage Trigger

Azure Blob

Path **Storage account connection**
Blob Storage trigger (image)

Path **Azure**

The screenshot shows a dark-themed 'New Function' dialog box for an 'Azure Blob Storage trigger'. At the top, there is a header with the Azure logo and the text 'Azure Blob Storage trigger'. Below this, the title 'New Function' is displayed. The form contains the following fields and controls:

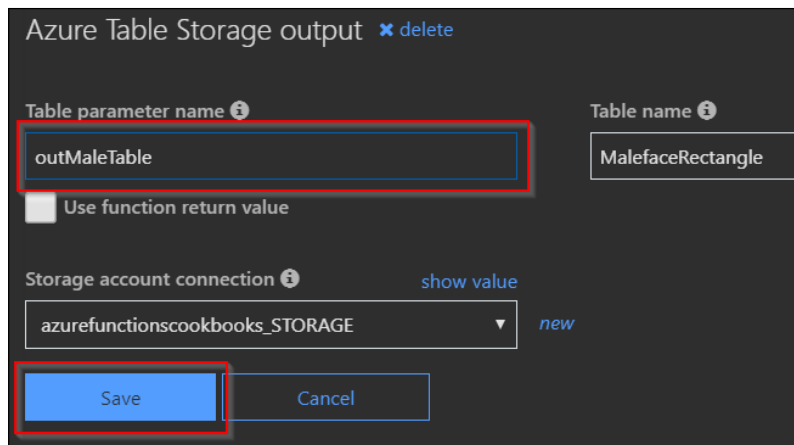
- Name:** A text input field containing 'LocateMaleFemaleFaces'.
- Azure Blob Storage trigger:** A label indicating the trigger type.
- Path:** A text input field containing 'images/(name)'. To the left of the field is a small information icon.
- Storage account connection:** A dropdown menu with the selected value 'azurefunctionscookbooks_STORAGE'. To the left of the dropdown is an information icon, and to the right are the words 'new' and 'show value'.
- Buttons:** Two buttons at the bottom: a blue 'Create' button and a 'Cancel' button.

Table output

Blob Storage
Table name

you have reviewed all the details, click on the **Create**

Output **Integrate** tab, click on **New**
Azure Table Storage, then click on the **Select**
Provide the parameter values, and then click on the **Save**



Azure Table Storage output ✕ delete

Table parameter name ⓘ
outMaleTable

Table name ⓘ
MalefaceRectangle

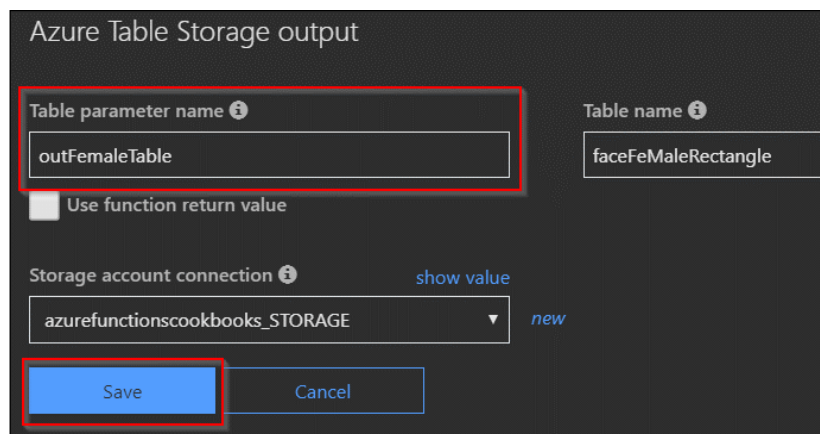
Use function return value

Storage account connection ⓘ show value
azurefunctionscookbooks_STORAGE new

Save Cancel

Azure Table Storage output

the information for women by clicking on the **New Output Integrate** **Azure Table Storage** and clicking on **Select** button. This is how it looks after providing the input values:



Azure Table Storage output

Table parameter name ⓘ
outFemaleTable

Table name ⓘ
faceFeMaleRectangle

Use function return value

Storage account connection ⓘ show value
azurefunctionscookbooks_STORAGE new

Save Cancel

reviewed all the details, click on the **Save Azure Table Storage output**

```

                                Run
LocateMaleFemaleFaces          outMaleTable
outFemaleTable

#r "Newtonsoft.Json"
#r "Microsoft.WindowsAzure.Storage"

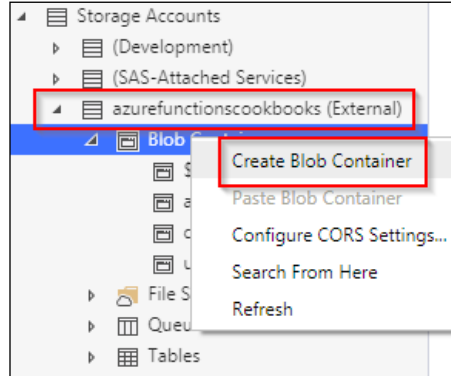
using Newtonsoft.Json;
using Microsoft.WindowsAzure.Storage.Table;
using System.IO;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
public static async Task Run(Stream myBlob,
                             string name,
                             IAsyncCollector<FaceRectangle>
outMaleTable,
                             IAsyncCollector<FaceRectangle>
outFemaleTable,
                             ILogger log)
{
    log.LogInformation($"C# Blob trigger function Processed blob\n
Name:{name} \n Size: {myBlob.Length} Bytes");
    string result = await CallVisionAPI(myBlob);
    log.LogInformation(result);
    if (String.IsNullOrEmpty(result))
    {
        return;
    }
    ImageData imageData = JsonConvert.DeserializeObject<ImageData>
(result);
    foreach (Face face in imageData.Faces)
    {
        var faceRectangle = face.FaceRectangle;
        faceRectangle.RowKey = Guid.NewGuid().ToString();
        faceRectangle.PartitionKey = "Functions";
        faceRectangle.ImageFile = name + ".jpg";
        if (face.Gender=="Female")
        {
            await outFemaleTable.AddAsync(faceRectangle);

```

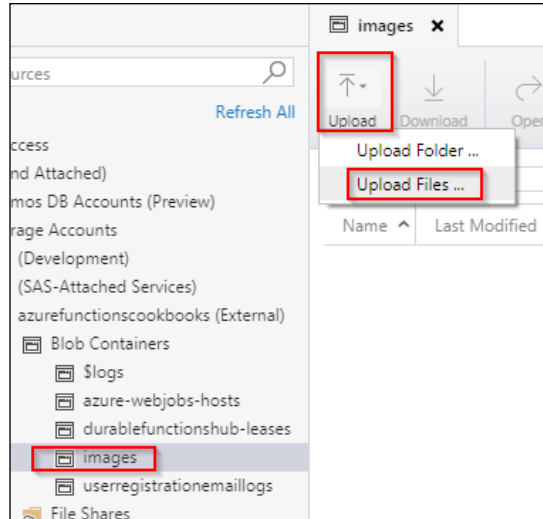
```
        }
        Else
        {
            await outMaleTable.AddAsync(faceRectangle);
        }
    }
}
static async Task<string> CallVisionAPI(Stream image)
{
    using (var client = new HttpClient())
    {
        var content = new StreamContent(image);
        var url = "https://westeurope.api.cognitive.microsoft.com/
vision/v1.0/analyze?visualFeatures=Faces&language=en";
        client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-
Key", Environment.GetEnvironmentVariable("Vision_API_Subscription_
Key"));
        content.Headers.ContentType = new MediaTypeHeaderValue("ap
plication/octet-stream");
        var httpResponse = await client.PostAsync(url, content);

        if (httpResponse.StatusCode == HttpStatusCode.OK)
        {
            return await httpResponse.Content.ReadAsStringAsync();
        }
    }
    return null;
}
public class ImageData
{
    public List<Face> Faces { get; set; }
}
public class Face
{
    public int Age { get; set; }
    public string Gender { get; set; }
    public FaceRectangle FaceRectangle { get; set; }
}
public class FaceRectangle : TableEntity
{
    public string ImageFile { get; set; }
    public int Left { get; set; }
    public int Top { get; set; }
    public int Width { get; set; }
    public int Height { get; set; }}
}
```

bold Step 10
to check the gender and, based on the gender, store the information in the
images using Azure Storage Explorer,

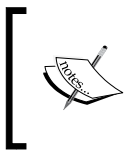


images using Azure Storage Explorer, as shown here:



Logs

```
{
  "requestId": "483566bc-7d4d-45c1-87e2-6f894aaa4c29",
  "metadata": { },
  "faces": [
    {
      "age": 31,
      "gender": "Female",
      "faceRectangle": {
        "left": 535,
        "top": 182,
        "width": 165,
        "height": 165
      }
    },
    {
      "age": 33,
      "gender": "Male",
      "faceRectangle": {
        "left": 373,
        "top": 182,
        "width": 161,
        "height": 161
      }
    }
  ]
}
```



| Female | | | | | Male | | | | | | | |
|---------------------|--------|--------|-------|--------|-------------------|-----------|--------------|--------|--------|-------|--------|--------|
| faceFeMaleRectangle | | | | | MalefaceRectangle | | | | | | | |
| Query | Import | Export | Add | Edit | Select all | Column Op | Query | Import | Export | Add | Edit | Select |
| PartitionKey | Left | Top | Width | Height | | | PartitionKey | Left | Top | Width | Height | R |
| Functions | 535 | 182 | 165 | 165 | | | Functions | 373 | 182 | 161 | 161 | 09 |

How it works...


We first created a Table Storage output binding for storing details about all the men

store all the face coordinates in a single table, we just made a small change to check

[ So, in your production environments, you should have a fallback]

There's more...


that the template provides invokes the Computer Vision API by
templates invoke the API call by passing the `visualFeatures=Faces`

[ `https://docs.microsoft.com/azure/cognitive-services/computer-vision/home`]

```
Environment.GetEnvironmentVariable("KeyName")
```

```
CallVisionAPI
```

method uses the function to retrieve the key, which is essential for making a request

[ It's a best practice to store all keys and other sensitive information in `ICollection` and `IAsyncCollector` are used for bulk insertion of data.]

Azure SQL Database interactions using Azure Functions

In this recipe, you will learn how to utilise the ADO.NET API to connect to `EmployeeInfo`

Getting ready

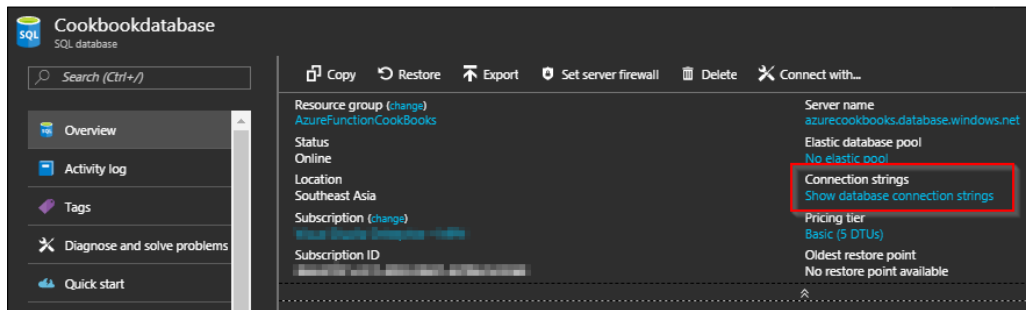
Blank database Select source Cookbookdatabase
SQL Database

Create a firewall rule for your IP address by clicking on the **Set Firewall rule**

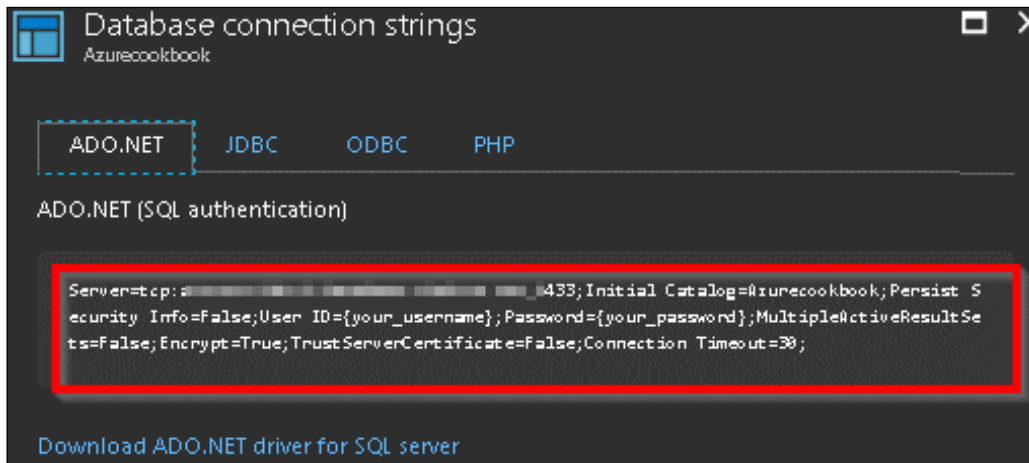
Overview SQL Server Management Studio SSMS

<https://docs.microsoft.com/sql/ssms/download-sql-server-management-studio-ssms>

Click on the **Show database connection strings** link in the **Essentials**



Copy the connection string from the following blade. Make sure that you
your_username your_password



EmployeeInfo

```
CREATE TABLE [dbo].[EmployeeInfo] (
  [PKEmployeeId] [bigint] IDENTITY(1,1) NOT NULL,
  [firstname] [varchar](50) NOT NULL,
  [lastname] [varchar](50) NULL,
  [email] [varchar](50) NOT NULL,
  [devicelist] [varchar](max) NULL,
  CONSTRAINT [PK_EmployeeInfo] PRIMARY KEY CLUSTERED
  (
    [PKEmployeeId] ASC
  )
)
```

How to do it...

HttpTrigger-CSharp

Authorisation level Anonymous

run.csx

SaveJSONToAzureSQLDatabase

```
#r "Newtonsoft.Json"
#r "System.Data"

using System.Net;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Primitives;
using Newtonsoft.Json;
using System.Data.SqlClient;
using System.Data;

public static async Task<IActionResult> Run(HttpContext
req, ILogger log)
{
```

```
log.LogInformation("C# HTTP trigger function processed a request.");

string firstname, lastname, email, devicelist;

string requestBody = await new StreamReader(req.Body).
ReadToEndAsync();
dynamic data = JsonConvert.DeserializeObject(requestBody);
firstname = data.firstname;
lastname = data.lastname;
email = data.email;
devicelist = data.devicelist;

SqlConnection con = null;
    Try
    {
        string query = "INSERT INTO EmployeeInfo
(firstname, lastname, email, devicelist) " + "VALUES (@firstname, @
lastname, @email, @devicelist) ";

        con = new
            SqlConnection("Server=tcp:azurecookbooks.database.
windows.net,1433;Initial Catalog=Cookbookdatabase;Persist Security
Info=False;User ID=username;Password=password;MultipleActiveResu
ltSets=False;Encrypt=True;TrustServerCertificate=False;Connection
Timeout=30;");
        SqlCommand cmd = new SqlCommand(query, con);
        cmd.Parameters.Add("@firstname", SqlDbType.VarChar,
50).Value = firstname;
        cmd.Parameters.Add("@lastname", SqlDbType.VarChar,
50)
            .Value = lastname;
        cmd.Parameters.Add("@email", SqlDbType.VarChar, 50)
            .Value = email;
        cmd.Parameters.Add("@devicelist", SqlDbType.VarChar)
            .Value = devicelist;
        con.Open();
        cmd.ExecuteNonQuery();
    }
    catch (Exception ex)
    {
        log.LogInformation(ex.Message);
    }
    Finally
    {
```

```

        if (con != null)
        {
            con.Close();
        }
    }

    return (ActionResult) new OkObjectResult($"Successfully
inserted the data.");
}

```

need to validate each and every input parameter. For the sake of simplicity, the code that validates the input parameters is not included. Make sure that you validate

Application settings

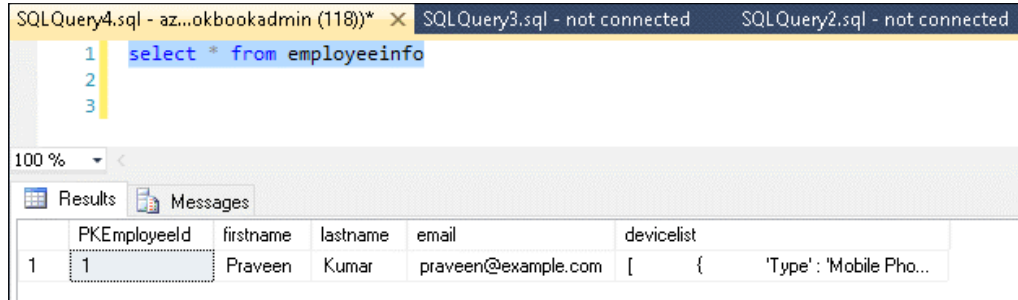
Test

```

{
    "firstname": "Praveen",
    "lastname": "Kumar",
    "email": "praveen@example.com",
    "devicelist":
        "[
            {
                'Type' : 'Mobile Phone',
                'Company': 'Microsoft'
            },
            {
                'Type' : 'Laptop',
                'Company': 'Lenovo'
            }
        ]"
}

```

need to validate each and every input parameter. For the sake of simplicity, the code that validates the input parameters is not included. Make sure that you validate



How it works...

known as **database as a service DBaaS**

created firewall rules that allow


workstation using SSMS. We have also created a table named `EmployeeInfo`

We have developed a simple program using the ADO.NET API that connects to the
`EmployeeInfo`

Monitoring tweets using Logic Apps and notifying users when a popular user tweets

works for a social grievance management project, is

such as Facebook, Twitter and so on. He was facing the problem of monitoring the tweets posted on his customer's Twitter handle with specific hashtags. His main job was to respond quickly to the tweets by users with a huge follower count, say, users with more than 50,000 followers. So, he was looking for a solution that kept monitoring a particular hashtag and alerted him whenever an user with more than 50,000 followers tweets so that he can quickly have his team respond

 Note that for the sake of simplicity, we will have the condition to check

Before I knew about Azure Logic Apps, I thought it would take a few weeks to learn about, develop, test and deploy such a solution. Obviously, it would take a good

hardly takes 10 minutes to design a solution for the problem that my friend had.

Getting ready

We need to have the following to work with this recipe:

When working with the recipe, we will need to authorise Azure Logic Apps to access

How to do it...

App by tweeting the tweets with the specific hashtag

Creating a new Logic App

Logic App

logic apps

Create logic app
group Subscription Location, click on the Create

Name Resource

Create logic app

Logic App

* Name
NotifyWhenTweetedByPopularUser ✓

* Subscription
[Subscription]

* Resource group ⓘ
 Create new Use existing
AzureFunctionCookBooks

Location
South Central US

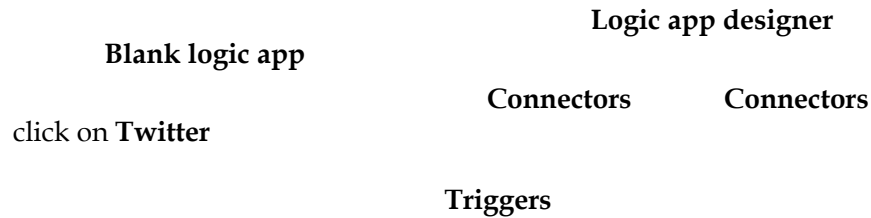
Log Analytics ⓘ

i You can add triggers and actions to your Logic App after creation.

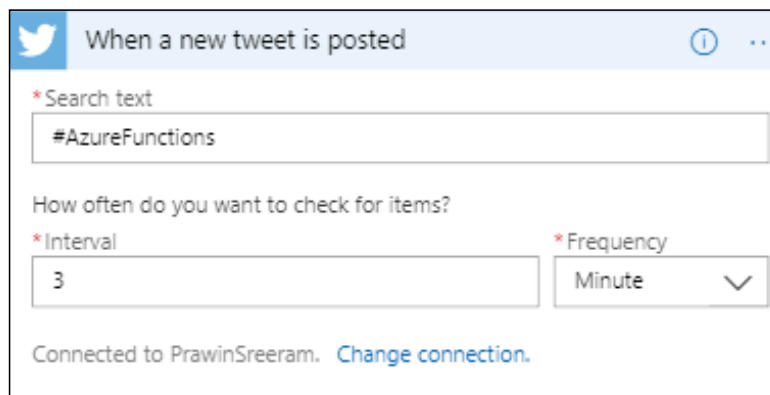
Pin to dashboard

[Automation options](#)

Designing the Logic App with Twitter and Gmail connectors

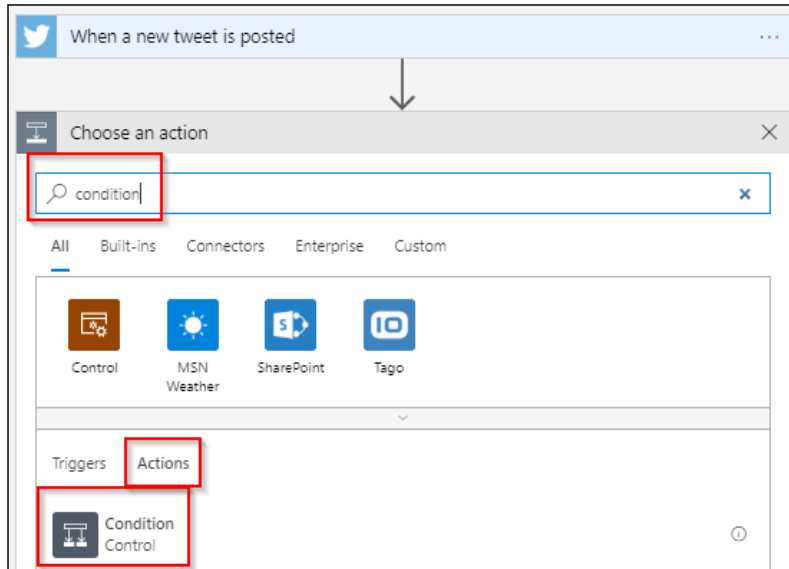


Once you have clicked on the **Twitter** **Search text** (for example, hashtags and keywords) and the **Frequency** at which you would like the Logic App to poll the tweets. This is how it looks after you provide the details:

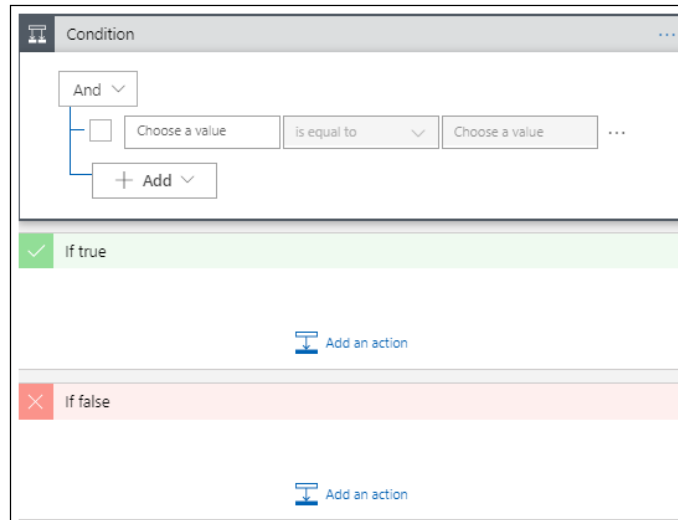


Let's add a new condition by clicking on **Next Step Condition**

condition

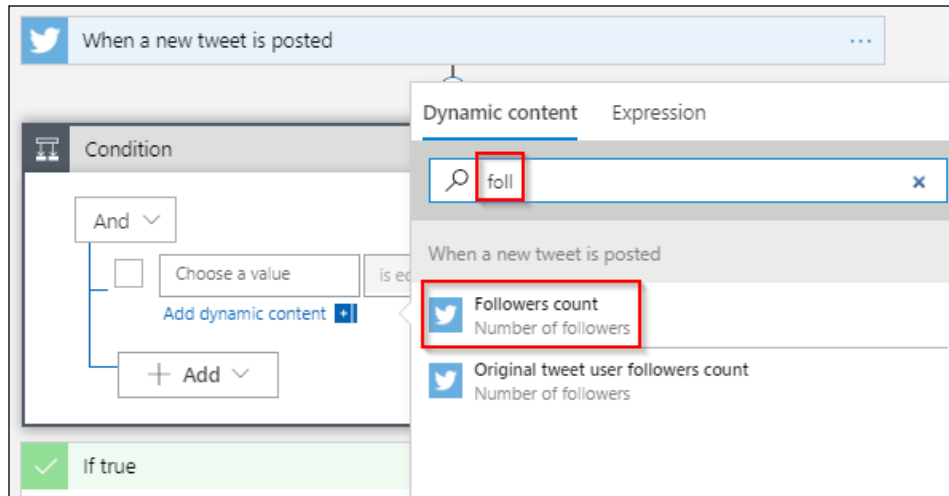


values for the condition and choose what you would like
true false



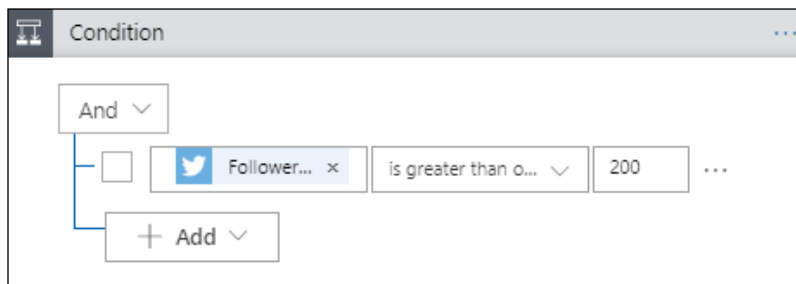
When you click on the **Choose a value** input field, you will get all the

Followers count

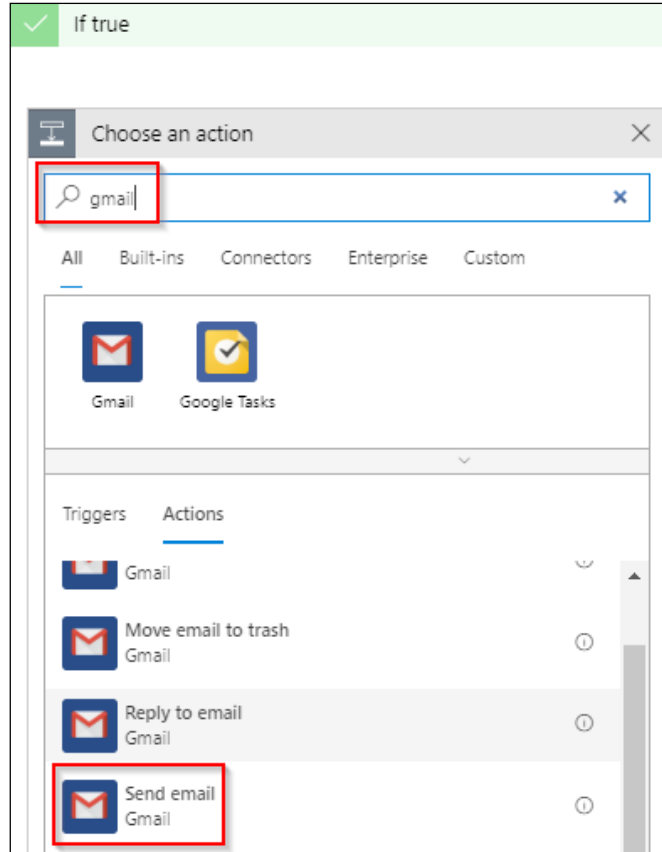


Followers Count

Followers count is greater than or equal to 200




If true **Condition**
Gmail **Send email**



It will ask you to

content

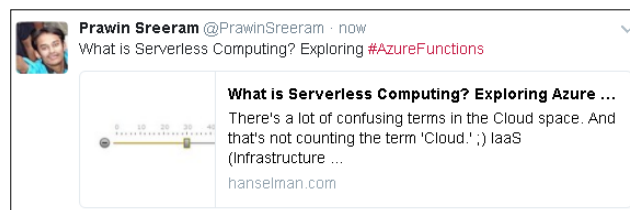
Add dynamic


Followers count
 screen, click the **See more** link.

are done, click on the **Save**

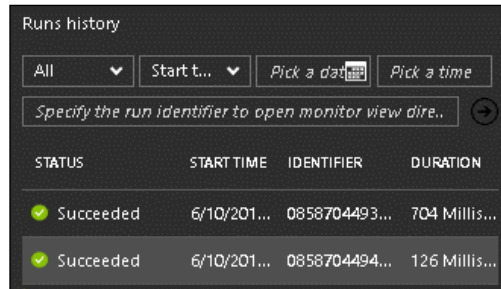
Testing the Logic App functionality

#AzureFunctions



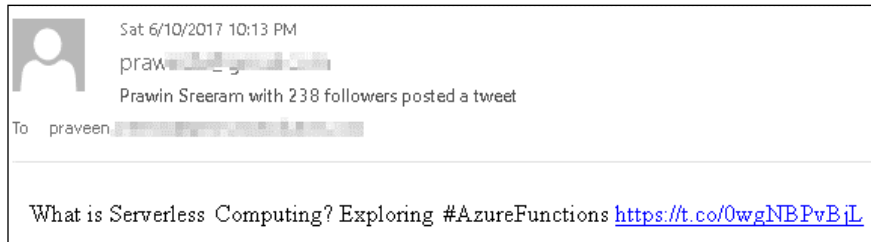
Overview

Runs history



The screenshot shows the 'Runs history' interface for an Azure Function. It includes a filter dropdown set to 'All', a 'Start t...' dropdown, and buttons for 'Pick a date' and 'Pick a time'. Below these is a search bar with the placeholder text 'Specify the run identifier to open monitor view dire..' and a search icon. A table below displays the run history with columns for STATUS, START TIME, IDENTIFIER, and DURATION. Two rows are visible, both showing 'Succeeded' status.

| STATUS | START TIME | IDENTIFIER | DURATION |
|-------------|-------------|---------------|---------------|
| ✓ Succeeded | 6/10/201... | 0858704493... | 704 Millis... |
| ✓ Succeeded | 6/10/201... | 0858704494... | 126 Millis... |



How it works...

#AzureFunctions
are any tweets with that hashtag, it checks whether the follower count is greater than

Integrating Logic Apps with serverless functions

in the previous recipe by just moving the conditional logic that checks the followers

Getting ready

key and create a new key in the **Application settings**

<https://www.getpostman.com/>

How to do it...

ValidateTwitterFollowerCount

Integrate
by clicking on the **New Output**

SendGrid

SendGrid output

Message parameter name ⓘ
message

Use function return value

To address ⓘ
To address

Message subject ⓘ
Message subject

SendGrid API Key App Setting ⓘ show value
SendGrid-APKey new

From address ⓘ
From address

Message Text ⓘ
Message Text

Save Cancel

Replace the default code with the following and click on **Save**
code just checks the

```
#r "Newtonsoft.Json"
#r "SendGrid"
using System.Net;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Primitives;
using Newtonsoft.Json;
```

```
using SendGrid.Helpers.Mail;
public static async Task<IActionResult> Run(HttpRequest req, IAsync
cCollector<SendGridMessage> messages, ILogger log)
{
    log.LogInformation("C# HTTP trigger function processed a
request.");

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

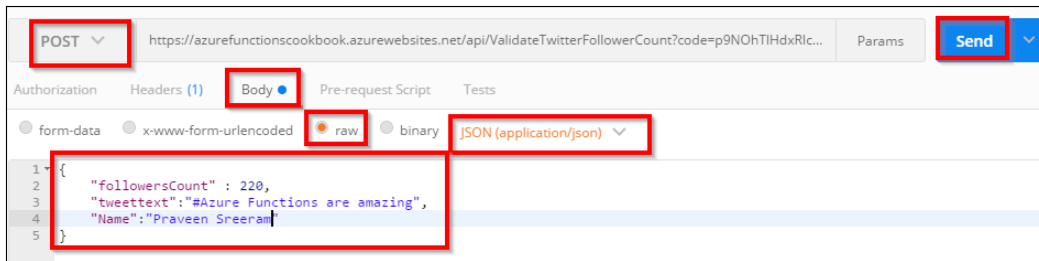
    string requestBody = await new StreamReader(req.Body).
ReadToEndAsync();
    dynamic data = JsonConvert.DeserializeObject(requestBody);

    string strTweet = "";
    SendGridMessage message = new SendGridMessage();
    if(data.followersCount >= 200)
    {
        strTweet = "Tweet Content" + data.tweettext;

        message.SetSubject($"{data.Name} with {data.
followersCount} followers has posted a tweet");
        message.SetFrom("donotreply@example.com");
        message.AddTo("prawin2k@gmail.com");
        message.AddContent("text/html", strTweet);

    }
    Else
    {
        message = null;
    }
    await messages.AddAsync(message);
    return (ActionResult)new OkObjectResult($"Hello");
}
```

```
ValidateTwitterFollowerCount
    followersCount tweettext    Name
```



NotifywhenTweetedbyPopularUserUsingFunctions

Blank logic app

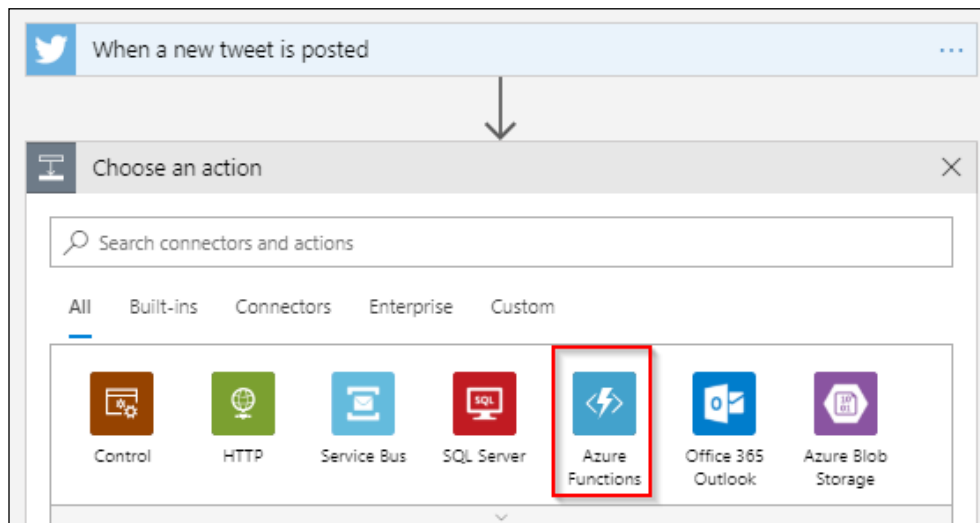
Twitter

configure **Search text** **Frequency** **Interval**

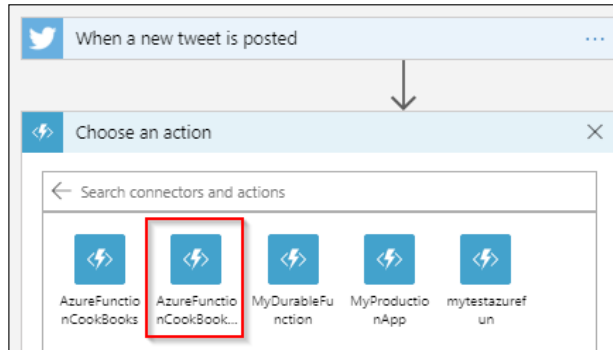
Click on **New step**

Choose an action

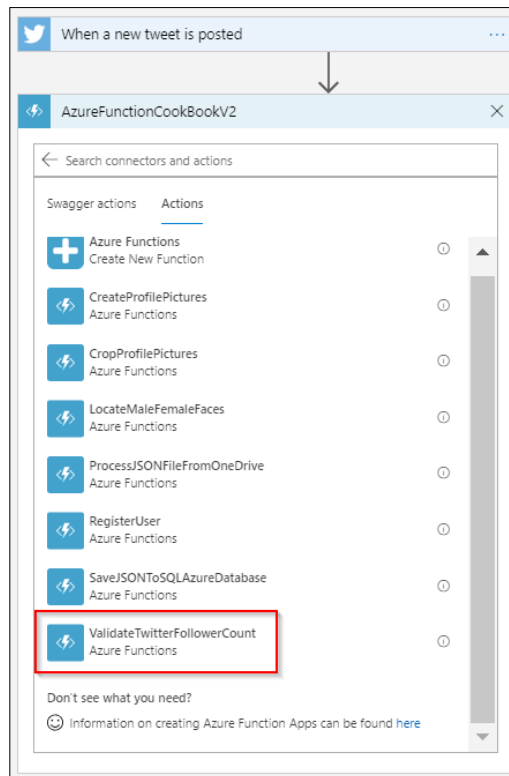
Azure Functions



Clicking on **Azure Functions**

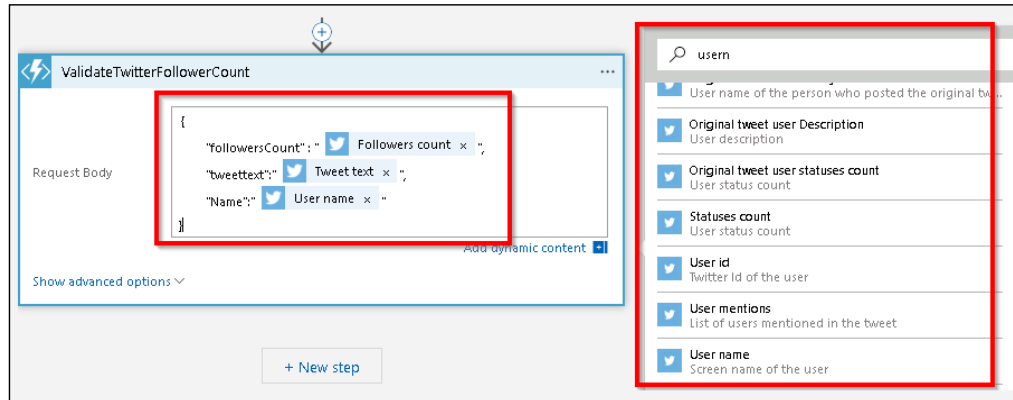


Click on the
ValidateTwitterFollowerCount
ValidateTwitterFollowerCount



ValidateTwitterFollowerCount

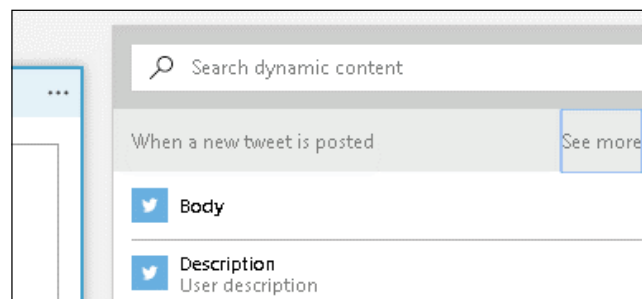
followersCount Name tweettext




`ValidateTwitterFollowerCount` function expects, click on the **Save** button to have configured in the **Search text** input field.

There's more...

dynamic parameter, click on the **See more**



 `ValidateTwitterFollowerCount`
200
store these values as configurable items by storing them in **Application settings**

See also

Sending an email notification to the end user dynamically
Chapter 2 Working with Notifications Using the SendGrid and Twilio Services

Auditing Cosmos DB data using change feed triggers

<https://docs.microsoft.com/azure/cosmos-db/introduction>

It might often be necessary to keep change logs of fields, attributes, documents
have seen developers using triggers or stored procedures to implement this kind of



Getting ready

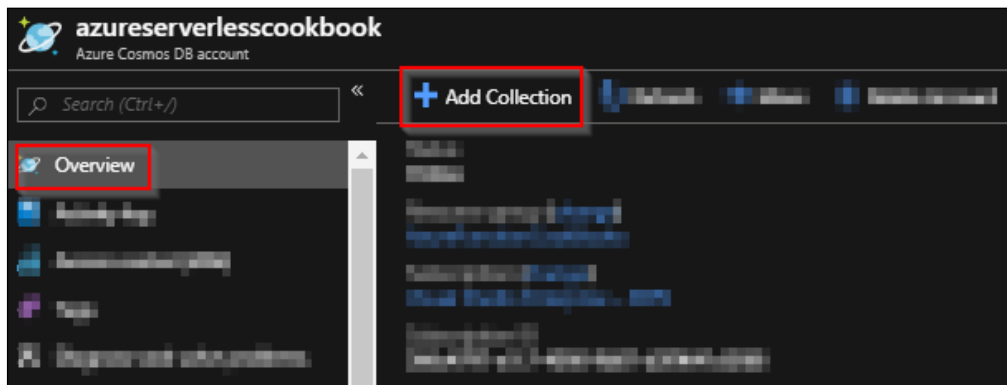
In order to get started, we need to first do the following:

Creating a new Cosmos DB account

document.azure.com <<accountname>>.

Creating a new Cosmos DB collection

Overview tab and click on the Add Collection

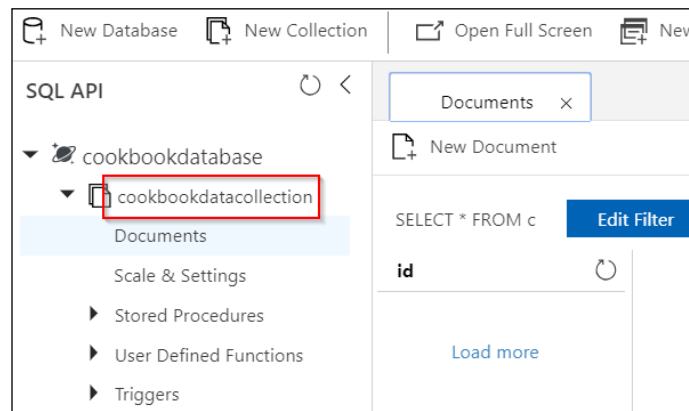


Data Explorer

| | | |
|--|------------------------|--|
| | | |
| | cookbookdatabase | |
| | cookbookdatacollection | |

| | | |
|-------|---------------|--------------------------------------|
| | Fixed (10 GB) | workloads, you might have to go with |
| RU/s) | 400 | |

Next, click on the **OK**
you will see something like the following in the **Data Explorer**




How to do it...

Navigate to the Cosmos DB Account and click on the **Add Azure Function**
All settings

taken to the **Add Azure Function**
choose the Azure function app in which you would like to create a new
collection happens. Here is how it looks:

Add Azure Function

 Create an Azure Function with an Azure Cosmos DB trigger that listens to the change feed of a collection. Click here to read more about Azure Functions and Azure Cosmos DB integration. [↗](#)

- ### 1 Select collection

Select the collection to monitor for changes. Your Azure Function will receive batches of changed items to be processed.
- ### 2 Create Azure Function

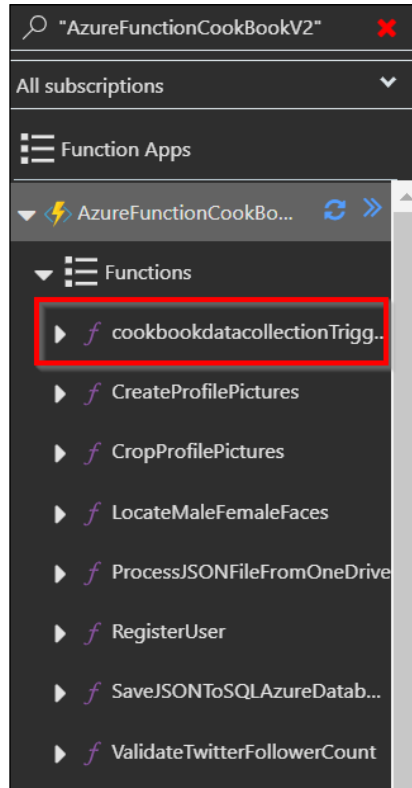
Select an Azure Function app

* Name your Azure Function

Function language

reviewed the details, click on the **Save**

for every change that is made in the collection. Let's quickly navigate to the
AzureFunctionCookBookV2
cookbookdatacollectionTrigger



Replace the

id of the first document in the **Logs**

```
#r "Microsoft.Azure.DocumentDB.Core"
using System;
using System.Collections.Generic;
using Microsoft.Azure.Documents;

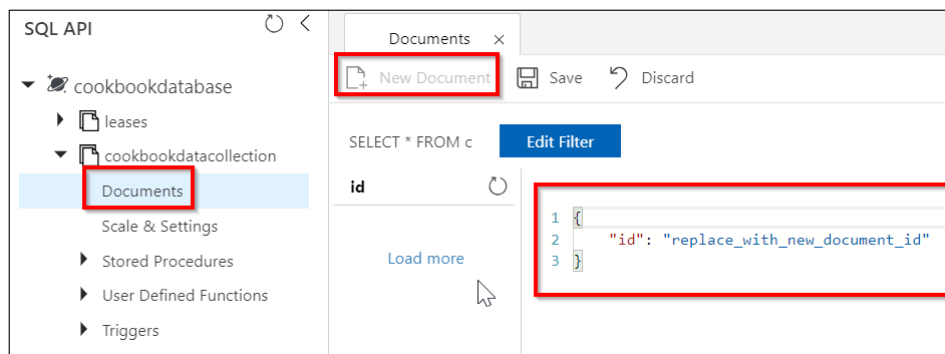
public static void Run(IReadOnlyList<Document> input, ILogger log)
{
```

```

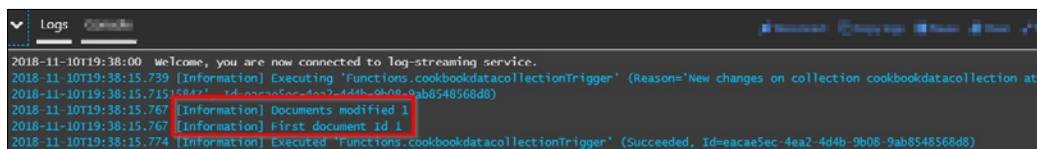
if (input != null && input.Count > 0)
{
    log.LogInformation("Documents modified " + input.Count);
    log.LogInformation("First document Id " + input[0].Id);
}
}

```

and see how the trigger gets fired in action. Open a new tab (leaving the `cookbookdatacollectionTrigger` the collection and create a new document by clicking on the **New Document**



with the JSON that has the required attributes, click on the **Save** button to save the changes and quickly navigate to the other browser tab, where you can view the logs. The following is how my logs look, as I just added a value to the `id` attribute of the document. It might look different for you, depending



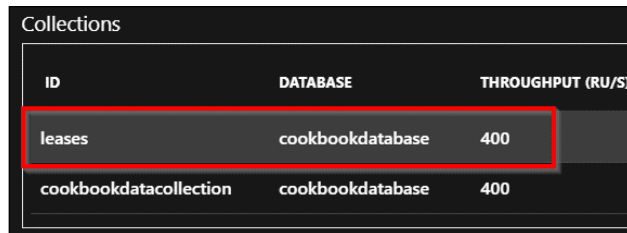
How it works...

Azure Functions with Cosmos DB, we first created a Cosmos DB collection, and then verified that the function was triggered automatically for all the changes (all reads and writes, but not deletes) that we make on the collection.

was created, we integrated it from within the Azure portal by clicking on the **Add Azure Function**

There's more...

track Cosmos DB changes, it will **leases** **request** **units RUs**



| ID | DATABASE | THROUGHPUT (RU/S) |
|------------------------|------------------|-------------------|
| leases | cookbookdatabase | 400 |
| cookbookdatacollection | cookbookdatabase | 400 |

to documents in a collection. If it is important for you to track deletes, then you need **soft deletes** **isDeleted** **isDeleted**

<https://docs.microsoft.com/azure/cosmos-db/change-feed>

if you think you won't use them anymore, because the collections are charged based on the RUs allocated even if you are not actively using them.

4

Understanding the Integrated Developer Experience of Visual Studio Tools

Introduction

In spite of all the advantages mentioned, developers might not find it comfortable

Integrated Development

Environments IDEs

Visual Studio Team Services VSTS

Implementing and Deploying

Continuous Integration Using Azure DevOps

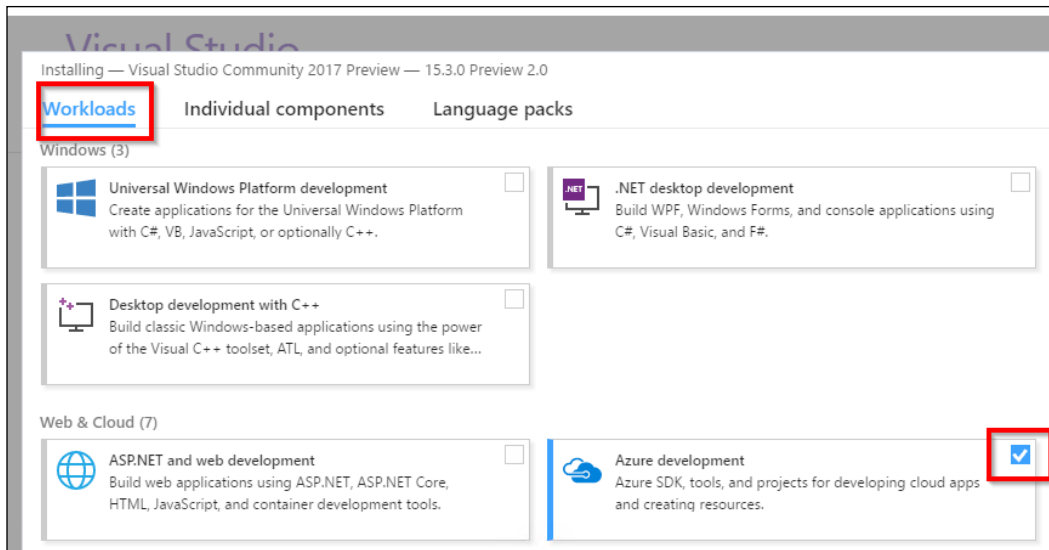
Creating a function app using Visual Studio 2017

Getting ready

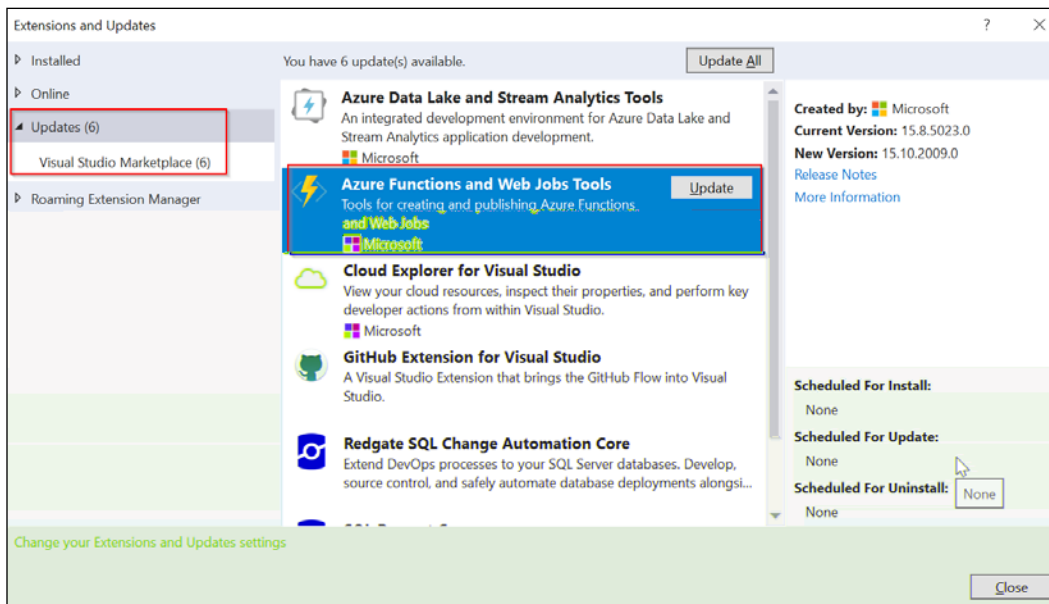
`https://visualstudio.microsoft.com/downloads/`

Azure development in the Workloads

Install



Tools Extensions and Updates



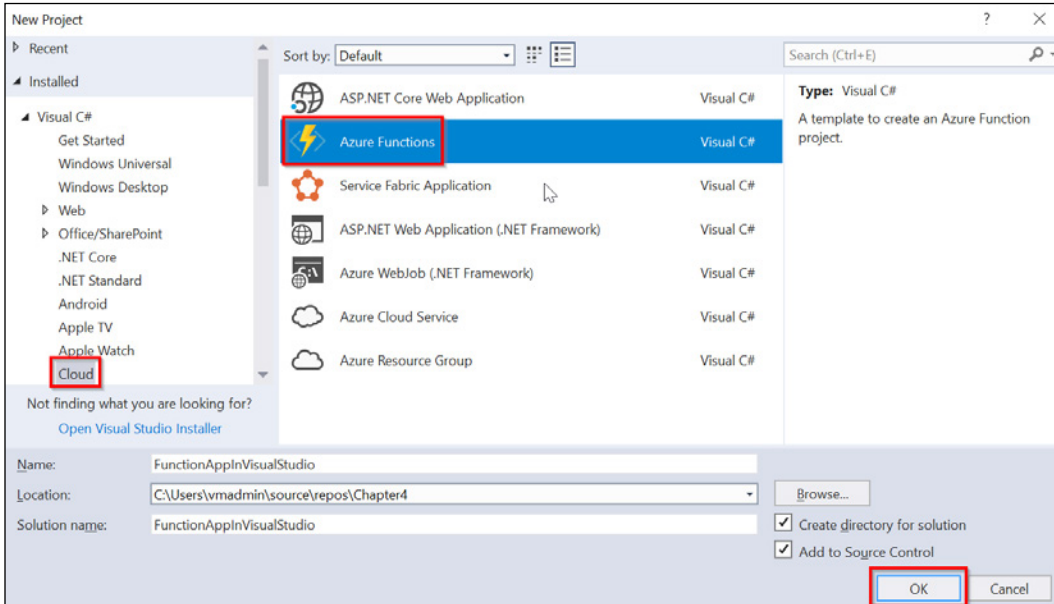
How to do it...

**Project
Cloud**

**File
Installed
Azure Functions**

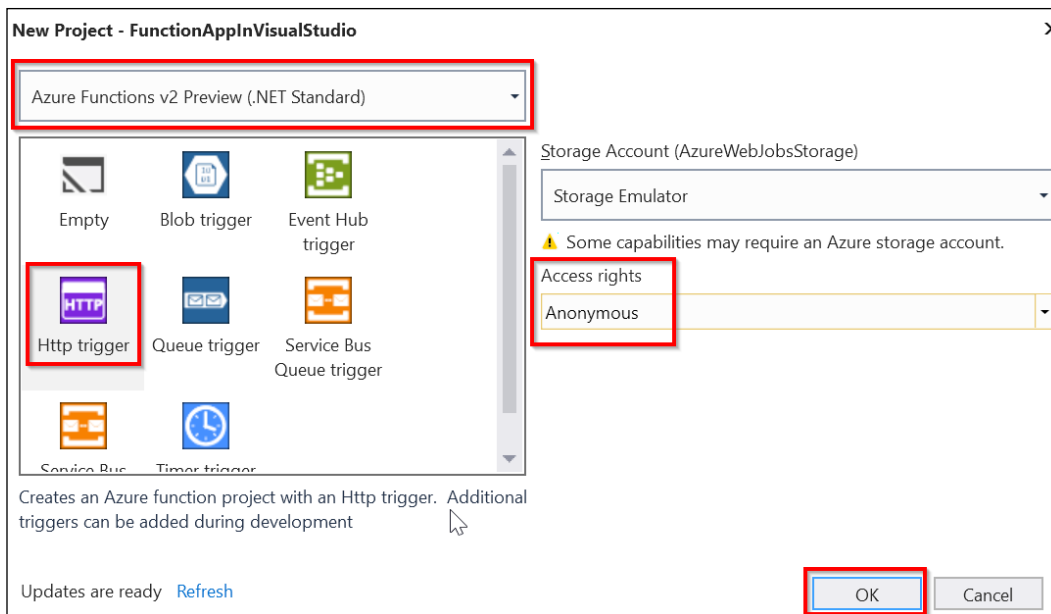
**New Project
Visual C#**

New

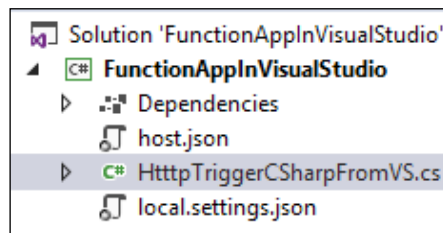


**(.NET Core)
OK**

**OK
Azure Functions v2
HTTP trigger**



Function1



How it works...

There's more...

*everything works fine on my local machine,
but not on the production environment*



Debugging C# Azure Functions on a local staged environment using Visual Studio 2017

issues. They need tools to help them identify the root cause of the problem and fix

In this recipe, you will learn how to configure and debug an Azure Function

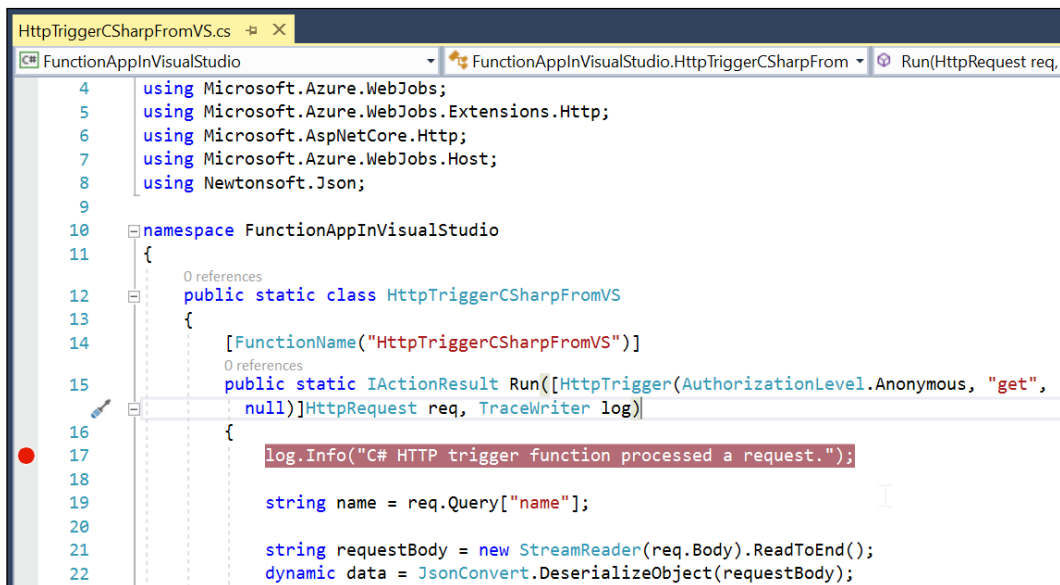
Getting ready

How to do it...

HTTPTrigger
Build

Build Solution

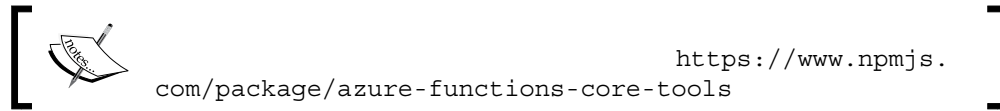
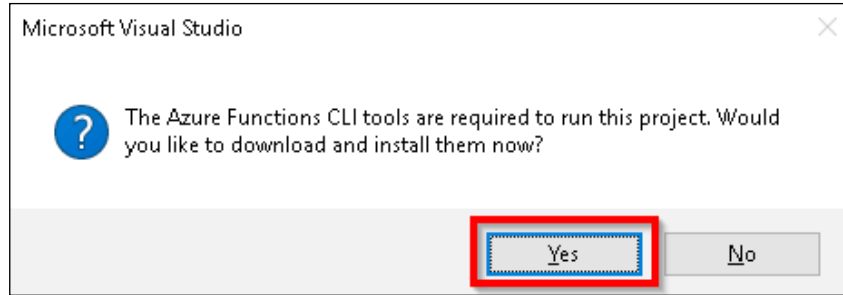
HTTPTriggerCSharpFromVS.cs file and create a breakpoint
F9



```
HttpTriggerCSharpFromVS.cs
FunctionAppInVisualStudio
FunctionAppInVisualStudio.HttpTriggerCSharpFrom
Run(HttpRequest req,

4  using Microsoft.Azure.WebJobs;
5  using Microsoft.Azure.WebJobs.Extensions.Http;
6  using Microsoft.AspNetCore.Http;
7  using Microsoft.Azure.WebJobs.Host;
8  using Newtonsoft.Json;
9
10 namespace FunctionAppInVisualStudio
11 {
12     0 references
13     public static class HttpTriggerCSharpFromVS
14     {
15         [FunctionName("HttpTriggerCSharpFromVS")]
16         0 references
17         public static IActionResult Run([HttpTrigger(AuthorizationLevel.Anonymous, "get",
18             null)]HttpRequest req, TraceWriter log)
19         {
20             log.Info("C# HTTP trigger function processed a request.");
21             string name = req.Query["name"];
22             string requestBody = new StreamReader(req.Body).ReadToEnd();
23             dynamic data = JsonConvert.DeserializeObject(requestBody);
```


F5 F5
first time, Visual Studio prompts you to download Visual Studio CLI tools



be created and started. It starts monitoring requests on a specific port for all

```

C:\Users\vmadmin\AppData\Local\AzureFunctionsTools\Releases\2.8.1\cli\func.exe
info: Host.Startup[0]
      Reading host configuration file 'C:\Users\vmadmin\source\repos\Chapter4\FunctionAppInVisualStudio\bin\Debug\netstandard2.0\host.json'
info: Host.Startup[0]
      Host configuration file read:
      {}
[9/23/2018 11:35:17 AM] Initializing Host.
[9/23/2018 11:35:17 AM] Host initialization: ConsecutiveErrors=0, StartupCount=1
[9/23/2018 11:35:17 AM] Starting JobHost
[9/23/2018 11:35:17 AM] Starting Host (HostId=vm2017-1799987705, InstanceId=2e6439d4-91e0-44a1-b7fd-a05n=2.0.12115.0, ProcessId=9760, AppDomainId=1, Debug=False, FunctionsExtensionVersion=)
[9/23/2018 11:35:17 AM] Generating 1 job function(s)
[9/23/2018 11:35:17 AM] Found the following functions:
[9/23/2018 11:35:17 AM] FunctionAppInVisualStudio.HttpTriggerCSharpFromVS.Run
[9/23/2018 11:35:17 AM]
[9/23/2018 11:35:17 AM] Host initialized (608ms)
[9/23/2018 11:35:17 AM] Host started (624ms)
[9/23/2018 11:35:17 AM] Job host started
Hosting environment: Production
Content root path: C:\Users\vmadmin\source\repos\Chapter4\FunctionAppInVisualStudio\FunctionAppInVisualStudio\netstandard2.0
Now listening on: http://0.0.0.0:7071
Application started. Press Ctrl+C to shut down.
Listening on http://0.0.0.0:7071/
Hit CTRL-C to exit...

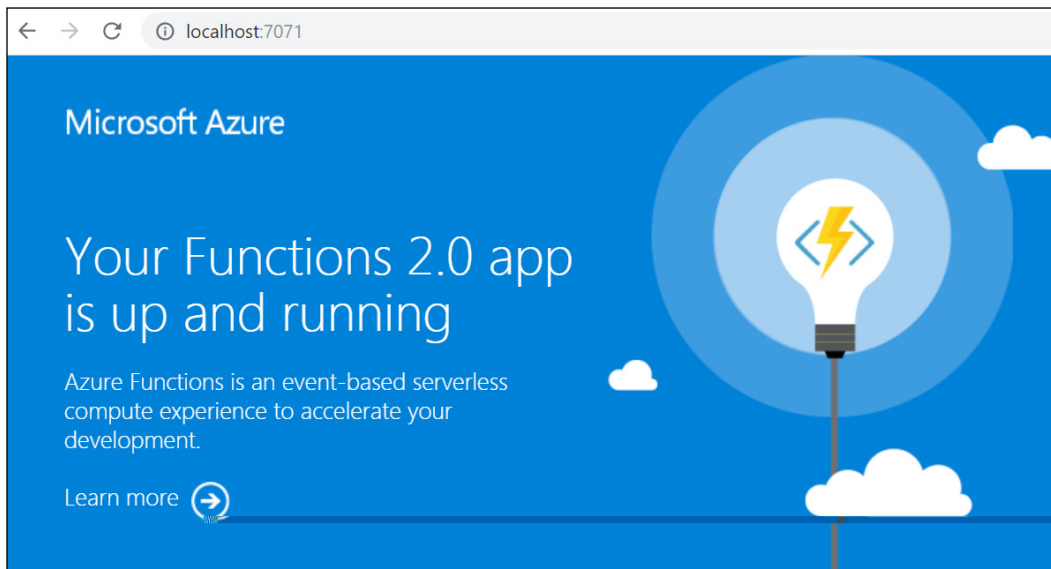
Http Functions:

HttpTriggerCSharpFromVS: http://localhost:7071/api/HttpTriggerCSharpFromVS

```

http://

localhost:7071



http://localhost:7071/api/HttpTriggerCSharpFromVS?name=Praveen Sreeram

```
10 namespace FunctionAppInVisualStudio
11 {
12     0 references
13     public static class HttpTriggerCSharpFromVS
14     {
15         [FunctionName("HttpTriggerCSharpFromVS")]
16         0 references
17         public static IActionResult Run([HttpTrigger(AuthorizationLevel.Anonym
18             null)]HttpRequest req, TraceWriter log)
19         {
20             log.Info("C# HTTP trigger function processed a request.");
21
22             string name = req.Query["name"];
23
24             string requestBody = new StreamReader(req.Body).ReadToEnd();
25             dynamic data = JsonConvert.DeserializeObject(requestBody);
26             name = name ?? data?.name;
```

```
string name = req.Query["name"];
name "Praveen Sreeram"
string requestBody = new StreamReader(req.Body).ReadToEnd();
dynamic data = JsonConvert.DeserializeObject(requestBody);
```

F5

```
localhost:7071/api/HttpTriggerCSharpFromVS?name=Praveen%20Sreeram
Hello, Praveen Sreeram
```

```
Authorization was successful.
9/23/2018 11:45:45 AM] Executing 'HttpTriggerCSharpFromVS' (Reason='This function was programmatically called via the
st APIs.', Id=940a2c99-aabf-40b7-b16f-f3b5e74fbf8b)
9/23/2018 11:45:45 AM] Executed 'HttpTriggerCSharpFromVS' (Succeeded, Id=940a2c99-aabf-40b7-b16f-f3b5e74fbf8b)
nfo: Microsoft.AspNetCore.Mvc.Infrastructure.ObjectResultExecutor[1]
    Executing ObjectResult, writing value of type 'System.String'.
nfo: Microsoft.AspNetCore.Hosting.Internal.WebHost[2]
    Request finished in 51.2225ms 200 text/plain; charset=utf-8
nfo: Microsoft.AspNetCore.Hosting.Internal.WebHost[1]
    Request starting HTTP/1.1 GET http://localhost:7071/api/HttpTriggerCSharpFromVS?name=Praveen%20Sreeram
nfo: Microsoft.AspNetCore.Authorization.DefaultAuthorizationService[1]
    Authorization was successful.
9/23/2018 11:45:53 AM] Executing 'HttpTriggerCSharpFromVS' (Reason='This function was programmatically called via the
st APIs.', Id=2d33812b-bacc-4d08-aed4-86e1a9c8d67d)
9/23/2018 11:47:00 AM] Executed 'HttpTriggerCSharpFromVS' (Succeeded, Id=2d33812b-bacc-4d08-aed4-86e1a9c8d67d)
nfo: Microsoft.AspNetCore.Mvc.Infrastructure.ObjectResultExecutor[1]
    Executing ObjectResult, writing value of type 'System.String'.
nfo: Microsoft.AspNetCore.Hosting.Internal.WebHost[2]
    Request finished in 66926.2838ms 200 text/plain; charset=utf-8
```

How it works...

The job host works as a server that listens to a specific port. If there are any requests

There's more...

.dll file that

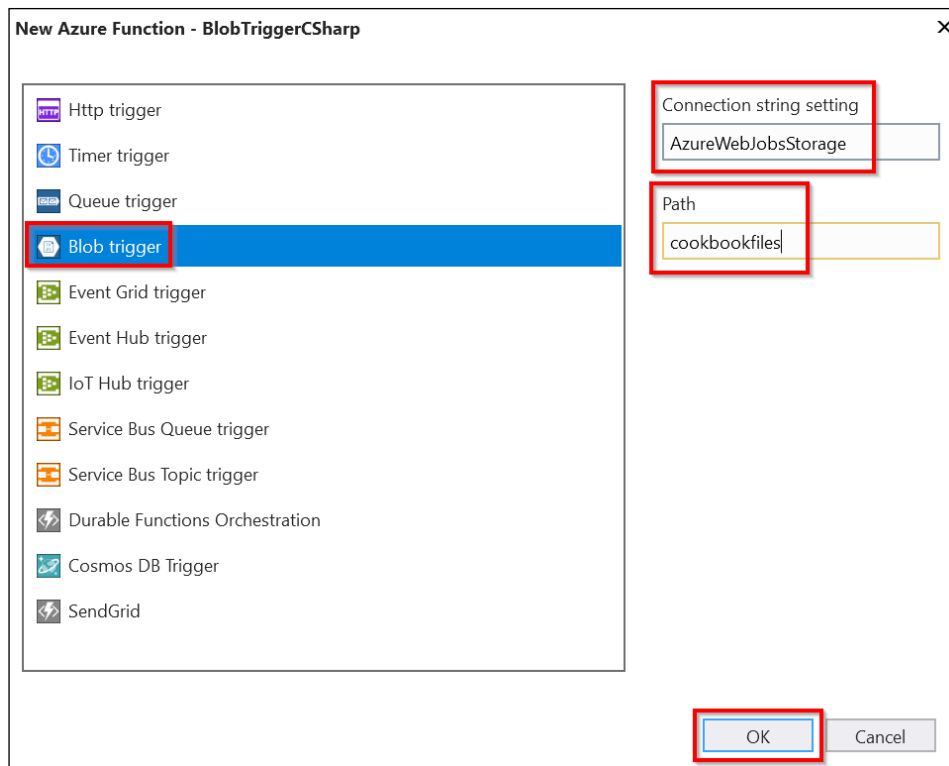
Connecting to the Azure Storage cloud from the local Visual Studio environment

Getting ready

```
com/                                     cookbookfiles
                                         http://storageexplorer.
```

How to do it...

```
FunctionAppInVisualStudio
FunctionAppInVisualStudio               Add New Azure Function
which opens a pop-up. Here, for the name field, enter BlobTriggerCSharp
Add
```

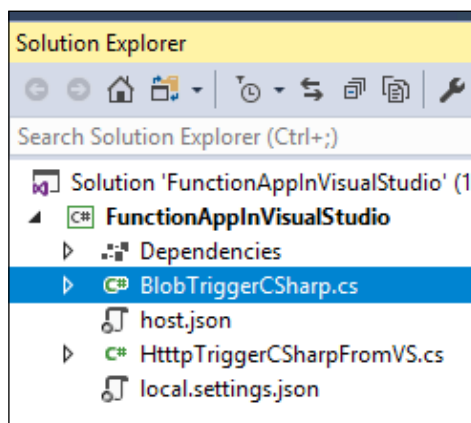


AzureWebJobsStorage

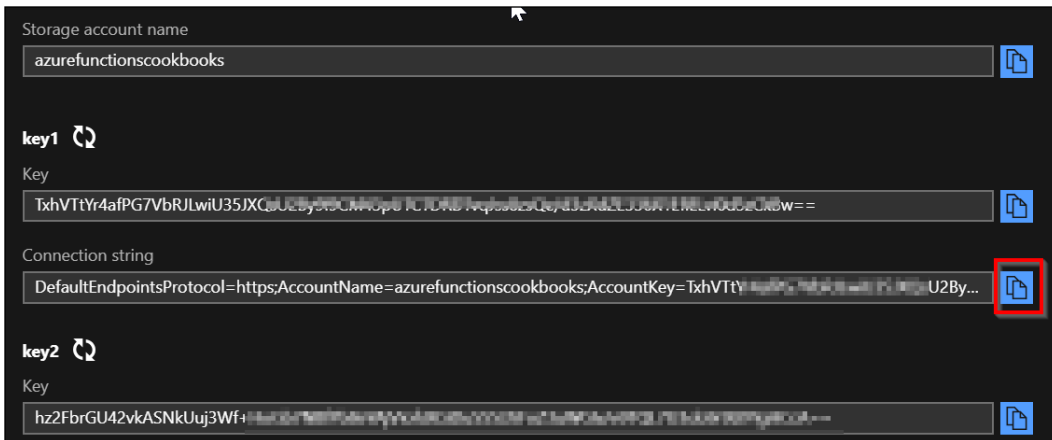
cookbookfiles

Path input field, then click

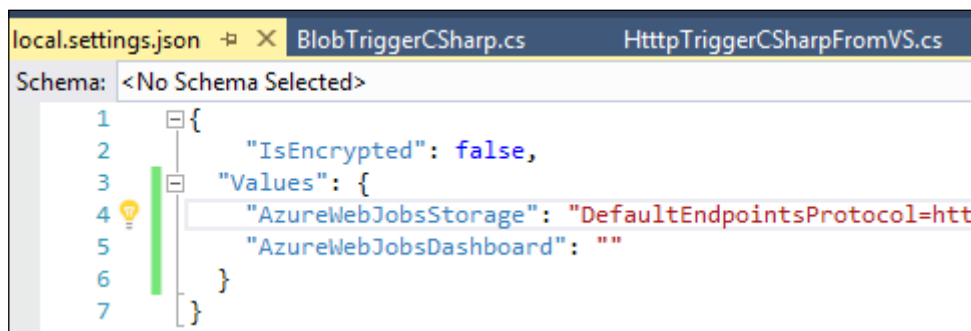
OK



Developing Cloud Applications Using Function Triggers
and Bindings



local.settings.json file, which
root folder of the project. This file is created when you create
AzureWebJobsStorage local.settings.json file should look like



BlobTriggerCSharp.cs file and

```

1  using System.IO;
2  using Microsoft.Azure.WebJobs;
3  using Microsoft.Azure.WebJobs.Host;
4  using Microsoft.Extensions.Logging;
5
6  namespace FunctionAppInVisualStudio
7  {
8      public static class BlobTriggerCSharp
9      {
10         [FunctionName("BlobTriggerCSharp")]
11         public static void Run([BlobTrigger("cookbookfiles/{name}", Connection = "AzureWebJobsStorage")]Stream myBlob, string name, ILogger log)
12         {
13             log.LogInformation($"C# Blob trigger function Processed blob\n Name:{name} \n Size: {myBlob.Length} Bytes");
14         }
15     }
16 }
17

```

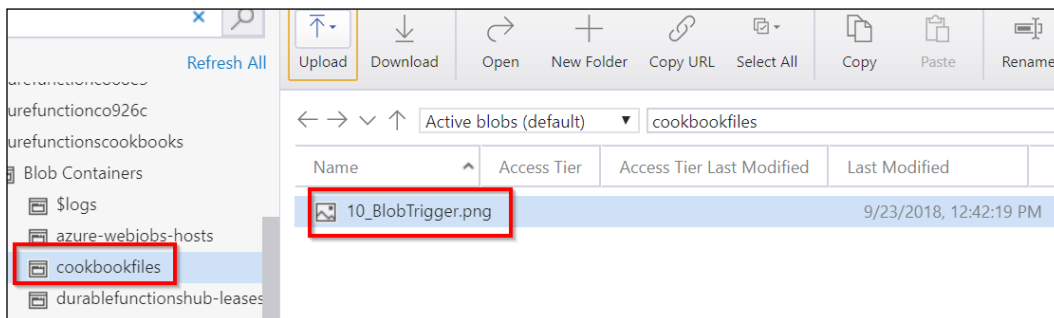
F5

```

[9/23/2018 12:39:21 PM] Initializing Host.
[9/23/2018 12:39:21 PM] Host initialization: ConsecutiveErrors=0, StartupCount=1
[9/23/2018 12:39:21 PM] Starting JobHost
[9/23/2018 12:39:21 PM] Starting Host (HostId=vm2017-1799987705, InstanceId=35c7497e-6bb8-4454-n=2.0.12115.0, ProcessId=13672, AppDomainId=1, Debug=False, FunctionsExtensionVersion=)
[9/23/2018 12:39:22 PM] Generating 2 job function(s)
[9/23/2018 12:39:22 PM] Found the following functions:
[9/23/2018 12:39:22 PM] FunctionAppInVisualStudio.BlobTriggerCSharp.Run
[9/23/2018 12:39:22 PM] FunctionAppInVisualStudio.HttpTriggerCSharpFromVS.Run
[9/23/2018 12:39:22 PM]
[9/23/2018 12:39:22 PM] Host initialized (502ms)
[9/23/2018 12:39:25 PM] Host started (3363ms)
[9/23/2018 12:39:25 PM] Job host started
Listening on http://0.0.0.0:7071/
Hit CTRL-C to exit...
Hosting environment: Production

```

new Blob file using



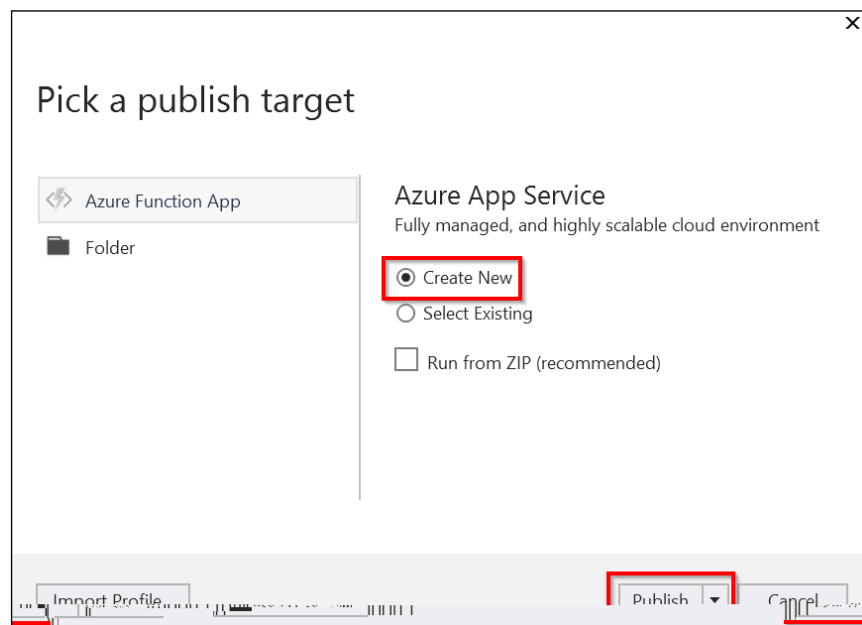
Deploying the Azure Function app to Azure Cloud using Visual Studio

How to do it...

Publish

Publish
Publish

Create New



Create App Service

New

Resource Group

Storage Account

Create App Service Microsoft account

Host your web and mobile applications, REST APIs, and more in Azure

App Name:

Subscription:

Resource Group: [New...](#)

Hosting Plan: [New...](#)

Storage Account: [New...](#)

Explore additional Azure services

- [Create a SQL Database](#)
- [Create a storage account](#)

Clicking the Create button will create the following Azure resources

- App Service - FunctionAppInVisualStudioV2

Consumption

Consumption

New

Consumption

Size

OK

Configure App Service Plan ✕

An App Service plan is the container for your app. The App Service plan settings will determine the location, features, cost and compute...

App Service Plan

Location

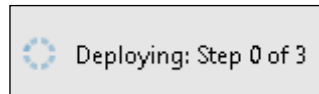
Size
 ⚠

⚠ Consumption App Service Plans are only available for Function Apps

App Service

Create

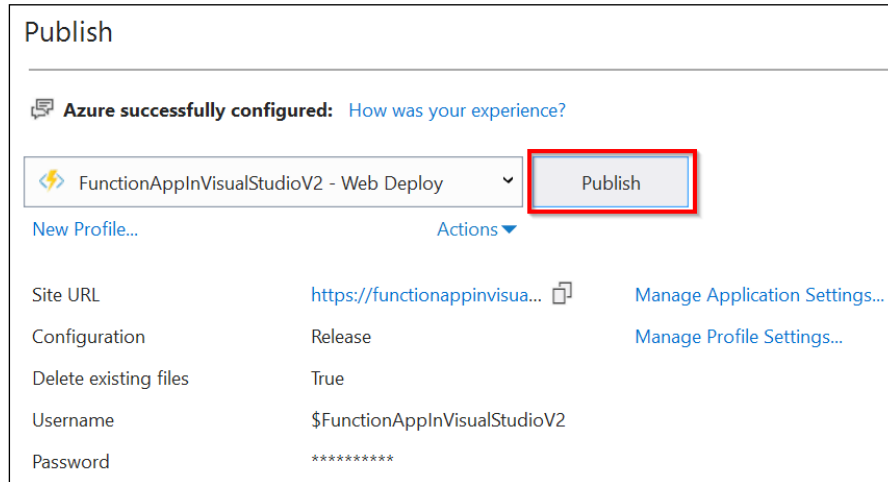
Create



If everything goes fine, you can view the newly created function app

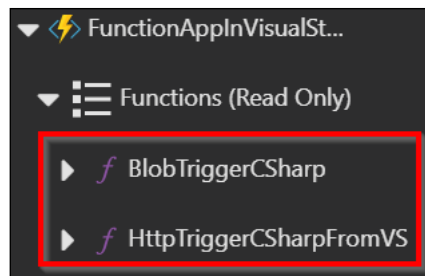
| Function Apps | | | |
|----------------------------------|---------------------------|-----------------------|----------------|
| NAME ▼ | SUBSCRIPTION ID ▼ | RESOURCE GROUP | LOCATION |
| AzureFunctionCookBook | Developer Program Benefit | AzureFunctionCookBook | southcentralus |
| cookbookPOC | Developer Program Benefit | cookbookPOC | southindia |
| FunctionAppinVisualStudio | Developer Program Benefit | cookbookPOC | southcentralus |

Publish



Output





There's more...

.dll files from Visual Studio 2017 to Azure. Therefore, you can make changes to the configurations, such as changing the Azure Storage

Debugging a live C# Azure Function, hosted on the Microsoft Azure Cloud environment, using Visual Studio

Connecting to the Azure Storage cloud from the local

Visual Studio environment

BlobTriggerCSharp

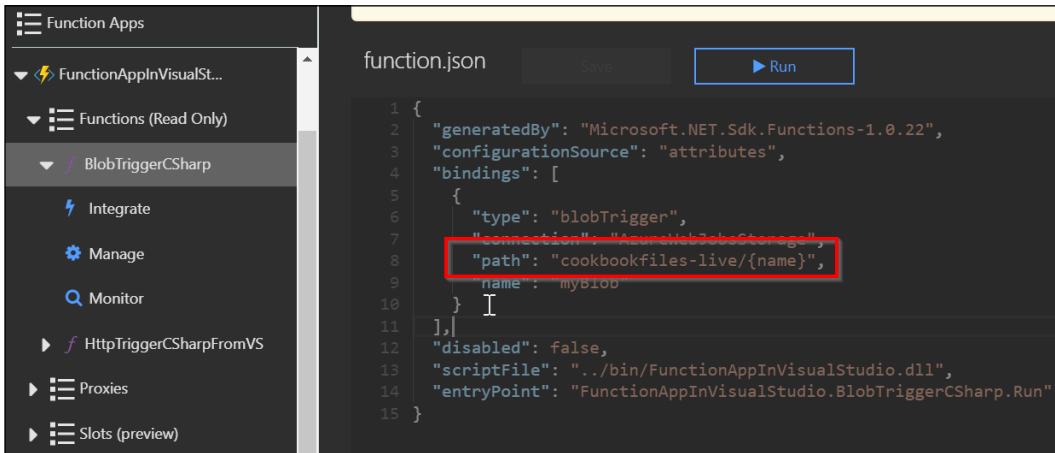
FunctionAppInVisualStudio

Getting ready

cookbookfiles

How to do it...

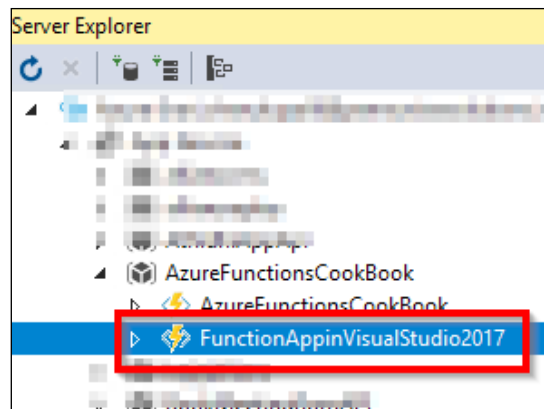
```
BlobTriggerCSharp
    path
cookbookfiles-live
```



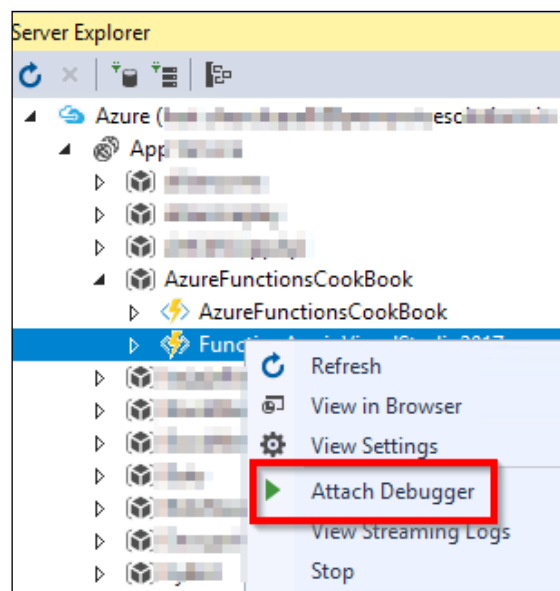
Explorer

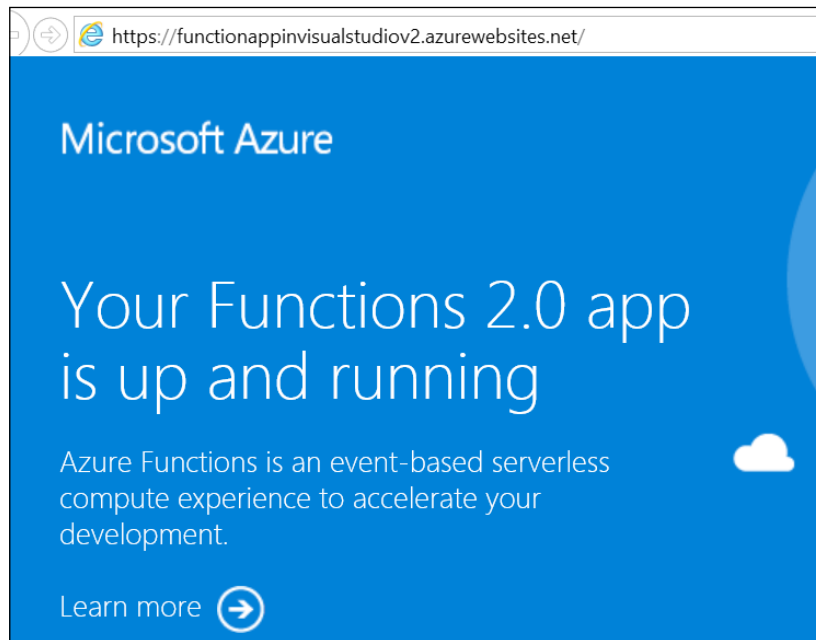
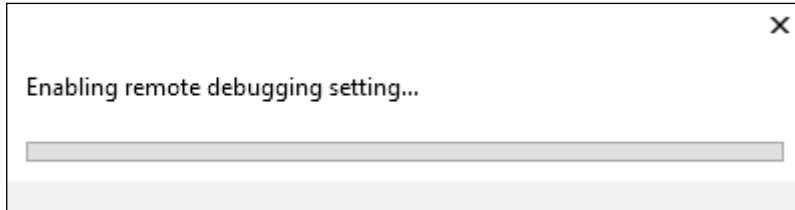
FunctionAppInVisualStudio2017

Server

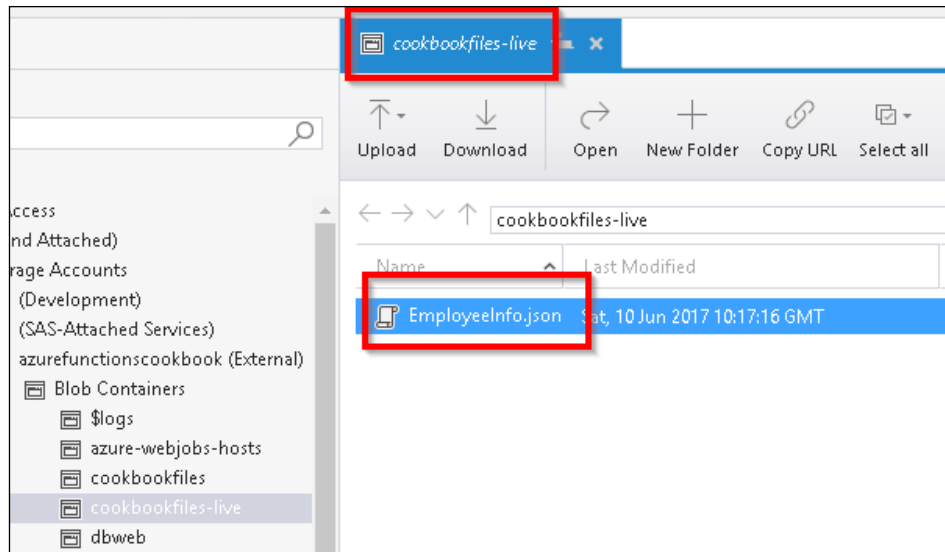


Attach Debugger

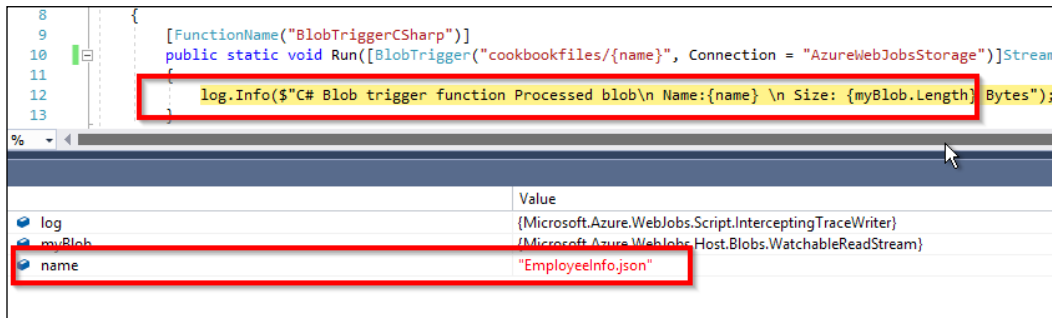




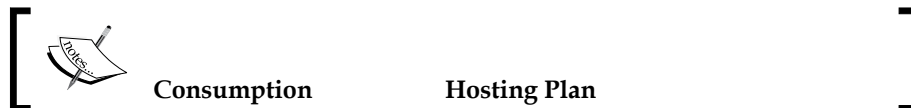
Storage Explorer and upload a new file (in this case, I uploaded
EmployeeInfo.json cookbookfiles-live



can view the filename that has been uploaded:



Deploying Azure Functions in a container



as you would lose all the serverless benefits of Azure Functions when you deploy

Getting ready

`https://docs.microsoft.com/cli/azure/
install-azure-cli?view=azure-cli-latest`

`https://store.docker.com/editions/
community/docker-ce-desktop-windows`

Azure Container Registry ACR

Creating an ACR

Create container registry [] [X]

* Registry name
cookbookregistry ✓
.azurecr.io

* Subscription
Visual Studio Enterprise – MPN

* Resource group
AzureFunctionCookBooks
[Create new](#)

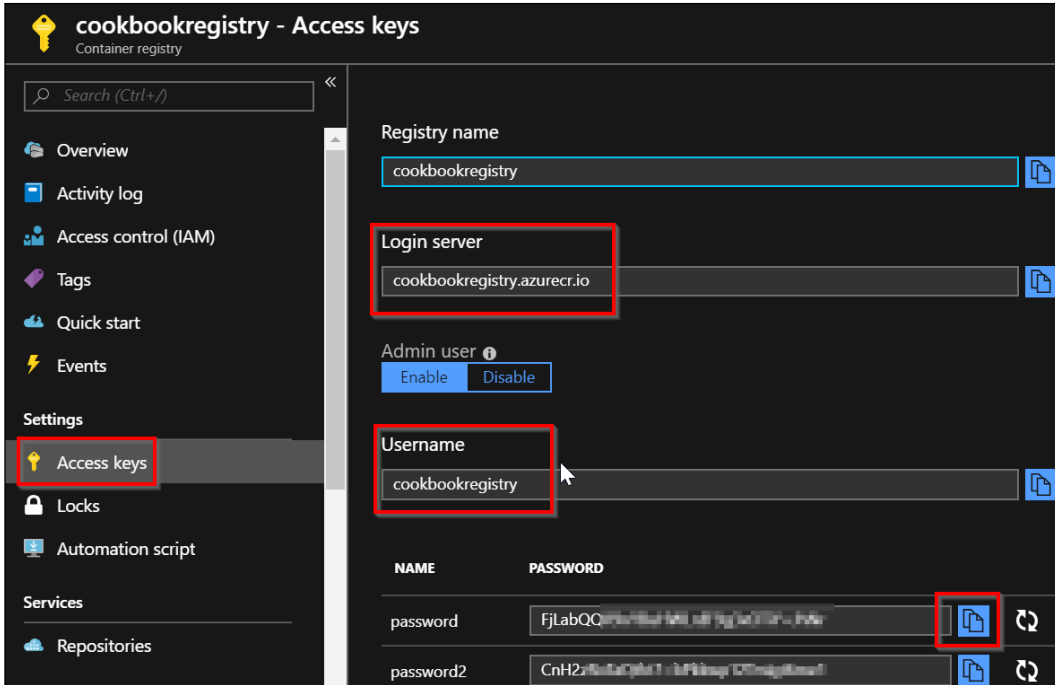
* Location
Central US

* Admin user ⓘ
Enable Disable

* SKU ⓘ
Basic

Create Automation options

Login server Username Access Keys
password



How to do it...

In the first three chapters, we

HTTPTrigger

From Docker

```

[FunctionName("HttpTriggerCSharpFromVS")]
0 references
public static IActionResult Run([HttpTrigger(AuthorizationLevel.Anonymous, "get", "post", Route = null)]HttpRequest req)
{
    //log.Info("C# HTTP trigger function processed a request.");

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

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

    return name != null
        ? (ActionResult)new OkObjectResult($"Hello, {name} - From Docker")
        : new BadRequestObjectResult("Please pass a name on the query string or in the request body");
}

```

Creating a Docker image for the function app

first step in creating a Docker image is to create a Dockerfile in our
.Dockerfile

```

FROM microsoft/azure-functions-dotnet-core2.0:2.0
COPY ./bin/Release/netstandard2.0 /home/site/wwwroot

```

```
docker build -t functionsindocker
```

```
docker build
```

```

C:\Users\vmadmin\source\repos\Chapter4\FunctionAppInVisualStudio\FunctionAppInVisualStudio>docker build -t functionsindocker .
Sending build context to Docker daemon 36.51MB
Step 1/2 : FROM microsoft/azure-functions-dotnet-core2.0:2.0
--> 35c818e033e2
Step 2/2 : COPY . .
--> b81847dc3c70
Successfully built b81847dc3c70
Successfully tagged functionsindocker:latest

```

image on a specific port. Run the command to execute it. You should see

```

C:\Users\vmadmin\source\repos\Chapter4\FunctionAppInVisualStudio\FunctionAppInVisualStudio>docker run -p 2380:80 functionsindocker
Hosting environment: Production
Content root path: /
Now listening on: http://[::]:80
Application started. Press Ctrl+C to shut down.

```



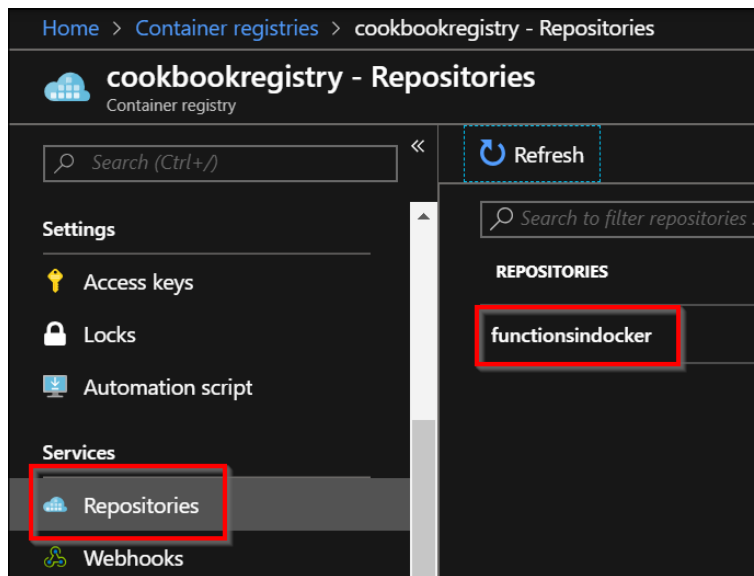
```
az acr login
```

```
--name cookbookregistry  
cookbookregistry
```

```
C:\Users\vmadmin\source\repos\Chapter4\FunctionAppInVisualStudio\FunctionAppInVisualStudio>az acr login --name cookbookregistry  
Login Succeeded  
C:\Users\vmadmin\source\repos\Chapter4\FunctionAppInVisualStudio\FunctionAppInVisualStudio>
```

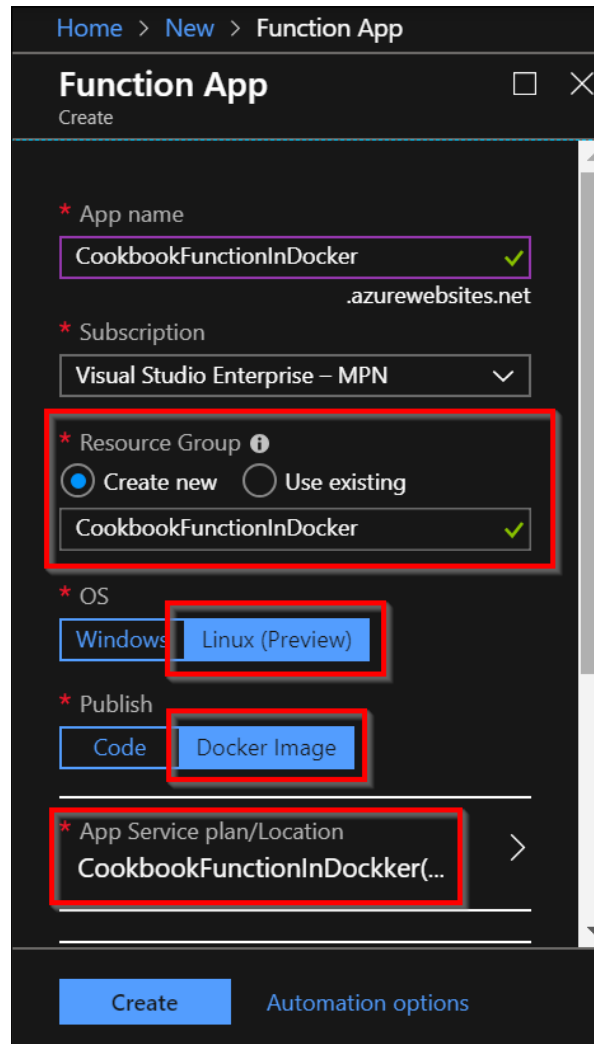
```
docker push cookbookregistry.azurecr.io/  
functionsindocker:v1
```

```
C:\Users\vmadmin\source\repos\Chapter4\FunctionAppInVisualStudio\FunctionAppInVisualStudio>docker push cookbookregistry.azurecr.io/functionsindocker:v1  
The push refers to repository [cookbookregistry.azurecr.io/functionsindocker]  
3f58e334a394: Pushed  
6f9d355b1699: Pushed  
e954f34d5c20: Pushed  
c30f8864b2d9: Pushed  
60add06a0c0d: Pushed  
8b15606a9e3e: Pushed  
v1: digest: sha256:2beca04c3ffaf2ded6df71eff718f0841840101ea5f5deac9f4dcec1c6ab0d9c size: 1588  
C:\Users\vmadmin\source\repos\Chapter4\FunctionAppInVisualStudio\FunctionAppInVisualStudio>
```



Creating a new function app with Docker

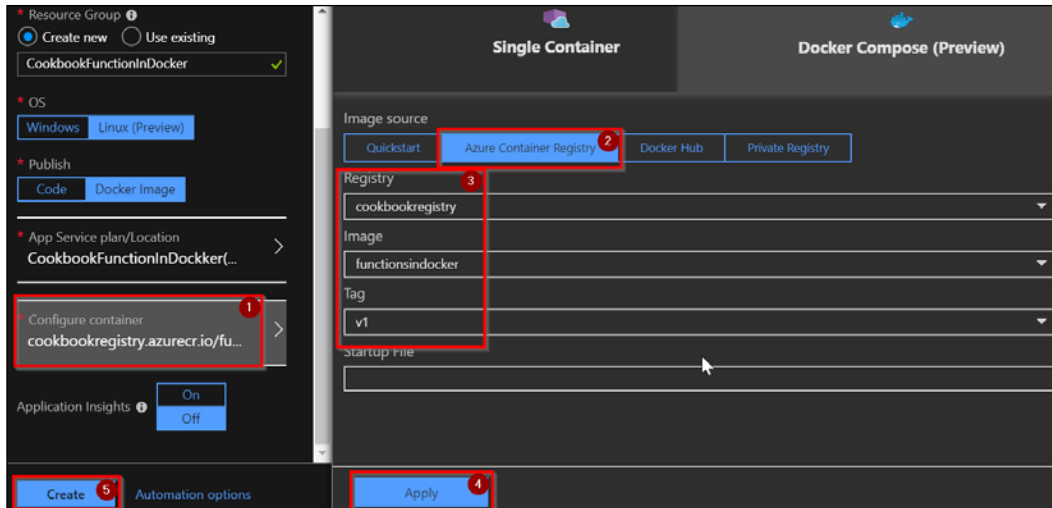
New Function App



Linux (Preview) OS field and choose **Docker Image** **Publish** field, and then click on the **App Service Plan/Location**

Basic

Configure container Azure Container Registry

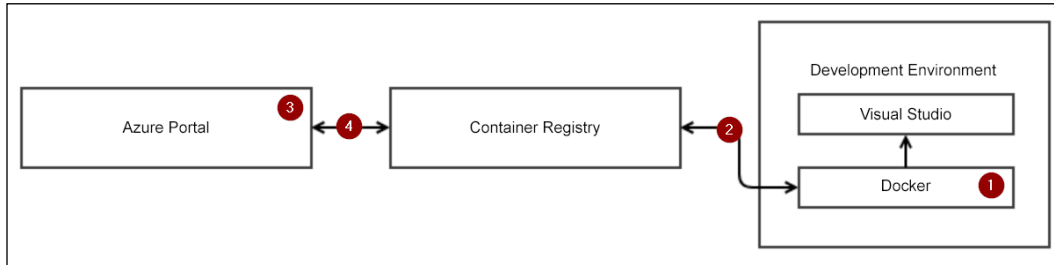


HttpTrigger



How it works...

In the first three chapters, we created both the function app and the functions right



5

Exploring Tests Tools for the Validation of Azure Functions

-
-
-

Introduction

by continuously pinging the application endpoints on a predefined frequency from

Testing Azure Functions

Getting ready

Postman

<https://www.getpostman.com/>

Microsoft Azure Storage Explorer
storageexplorer.com/

<http://>

How to do it...

Testing HTTP triggers using Postman

| | Firstname | Lastname |
|----------------------------|-----------|------------------|
| Authorisation Level | | Anonymous |

```
#r "Newtonsoft.Json"

using System.Net;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Primitives;
using Newtonsoft.Json;

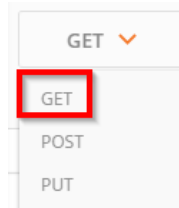
public static async Task<IActionResult> Run(HttpRequest req,
ILogger log)
{
    log.LogInformation("C# HTTP trigger function processed
a request.");

    string firstname=req.Query["firstname"];
    string lastname=req.Query["lastname"];

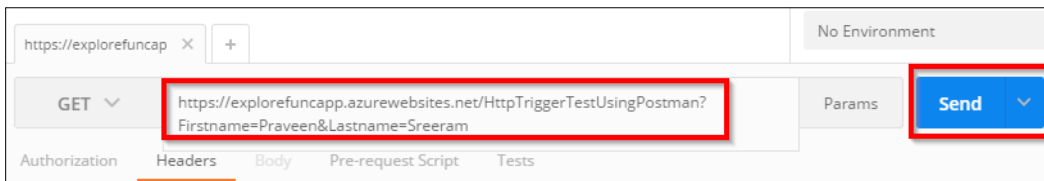
    string requestBody = await new StreamReader(req.Body).
ReadToEndAsync();
    dynamic data = JsonConvert.DeserializeObject(requestBody);
    firstname = firstname ?? data?.firstname;
    lastname = lastname ?? data?.lastname;

    return (ActionResult)new OkObjectResult($"Hello,
{firstname + " " + lastname}");
}
```

would like to make the HTTP request. As our function accepts most **GET**

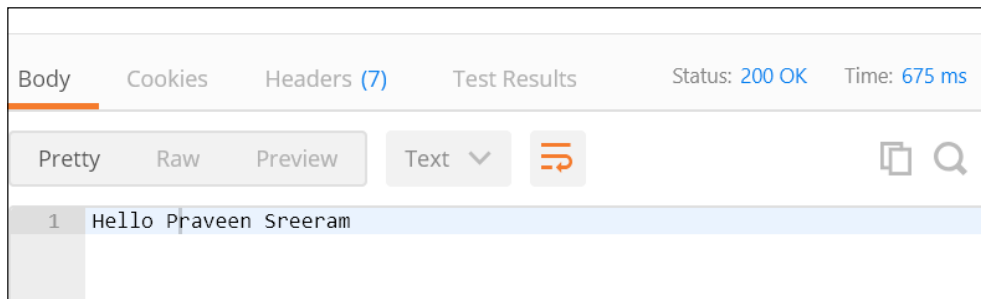


<HttpTriggerTestUsingPostman>



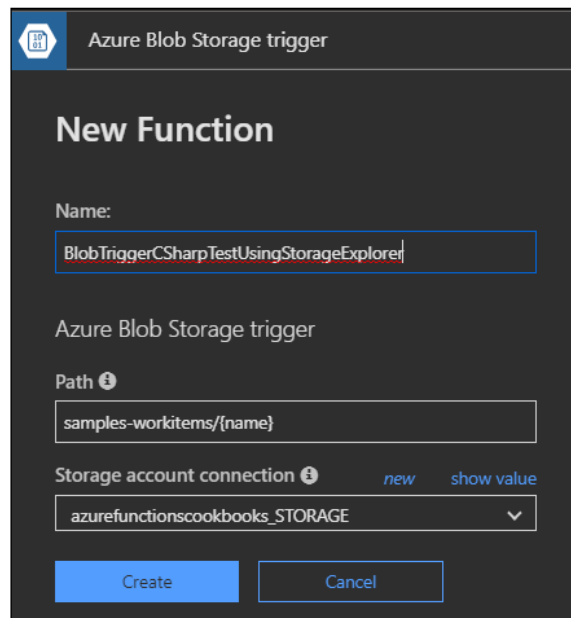
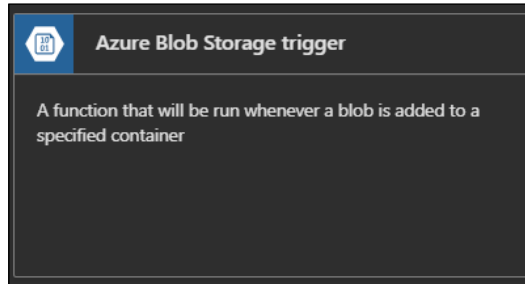
Send

request. If you have provided all the **Status: 200 OK**

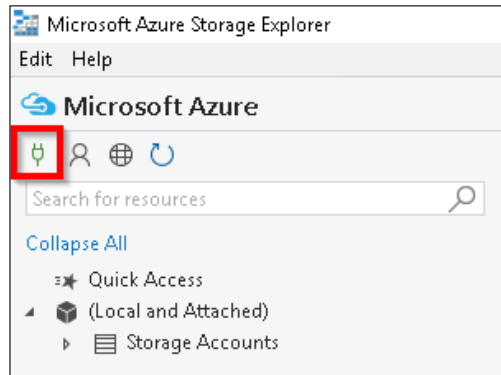


Testing a Blob trigger using Microsoft Storage Explorer

Azure Blob Storage trigger



Microsoft Azure Storage Explorer

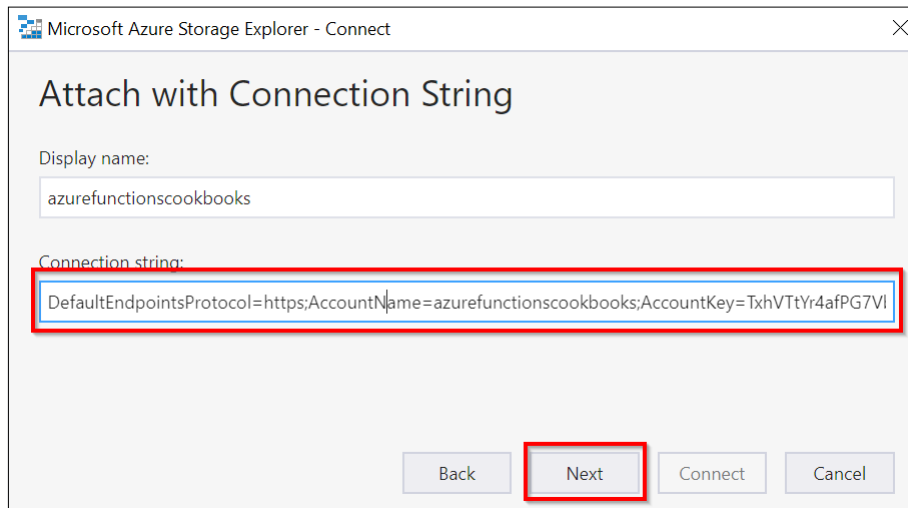


shared access signature SAS

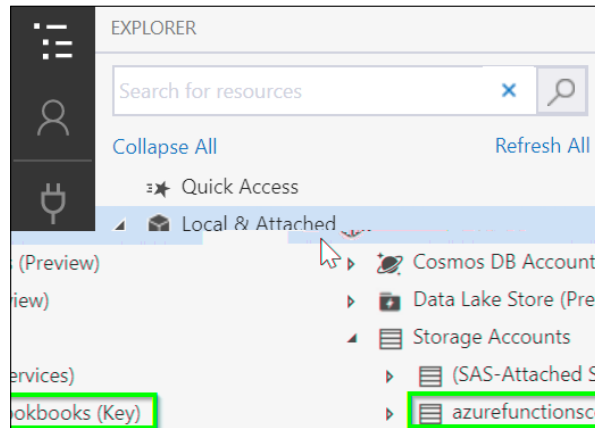
account key

Access Keys

Microsoft Azure Storage Explorer - Connect



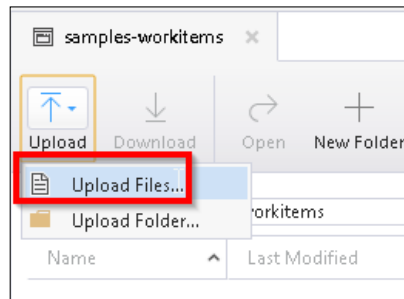
Next
Connection Summary
related details for confirmation. Click on the Connect button to connect to the



Blob Containers
files

samples-workitems
Create Blob Container

samples-workitems
Upload



Upload Files window, choose a file that you would like to upload,
Upload

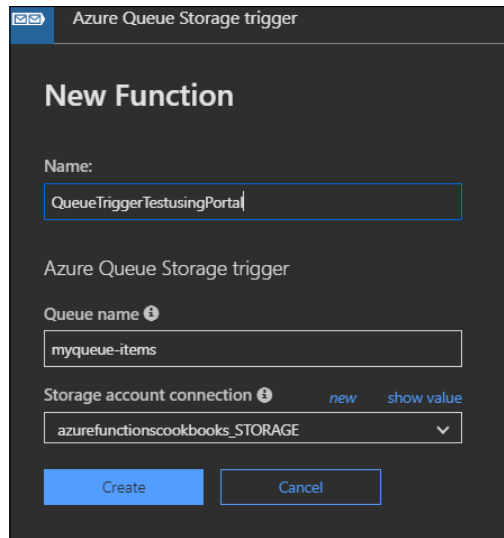
```
2018-09-27T02:52:35 Welcome, you are now connected to log-streaming service.
2018-09-27T02:53:35 No new trace in the past 1 min(s).
2018-09-27T02:54:35 No new trace in the past 2 min(s).
2018-09-27T02:55:35 No new trace in the past 3 min(s).
2018-09-27T02:56:35 No new trace in the past 4 min(s).
2018-09-27T02:57:35 No new trace in the past 5 min(s).
2018-09-27T02:58:35 No new trace in the past 6 min(s).
2018-09-27T02:59:15.577 [Info] Function started (Id=a07da295-103d-4bad-94b1-0113389bffffa)
2018-09-27T02:59:15.638 [Info] C# Blob trigger function Processed blob
  Name: 1_BlobTemplate.png
  Size: 12254 Bytes
2018-09-27T02:59:15.638 [Info] Function completed (Success, Id=a07da295-103d-4bad-94b1-0113389bffffa, Duration=72ms)
```

Testing the Queue trigger using the Azure Management portal

Azure Storage Queue trigger

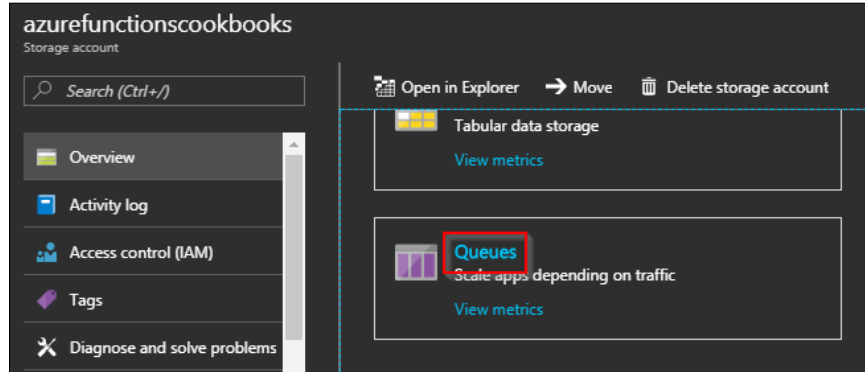
QueueTriggerTestusingPortal

myqueue-items



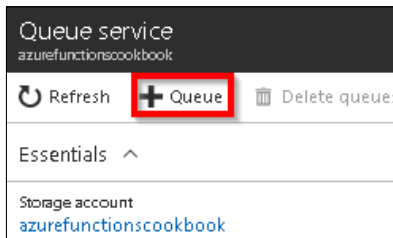
Overview

Queues



Queue service

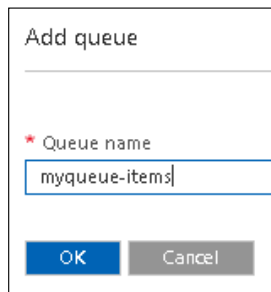
Queue



Provide myqueue-items as the **Queue name**

Add queue

OK



Messages myqueue-items Add message

OK

Refresh + Add message Dequeue message Clear queue

Add message to queue

* Message text

This is a queue message created for testing queues ✓

* Expires in:

7 Days

Encode the message body in Base64 ⓘ

OK Cancel

QueueTriggerTestusingPortal
Logs blade. Here, you can find out how the Queue function

```
2018-09-27T03:01:11 welcome, you are now connected to log-streaming service.
2018-09-27T03:02:11 No new trace in the past 1 min(s).
2018-09-27T03:03:11 No new trace in the past 2 min(s).
2018-09-27T03:03:37.216 [Info] Function started (Id=935287b1-e661-4249-85ef-6010ab20459a)
2018-09-27T03:03:37.247 [Info] C# Queue trigger function processed: This is a queue message created for testing queues
2018-09-27T03:03:37.247 [Info] Function completed (Success, Id=935287b1-e661-4249-85ef-6010ab20459a, Duration=28ms)
```

There's more...

POST POST Selected HTTP methods Selected methods

HTTP trigger (req) [delete](#)

Allowed HTTP methods ⓘ
Selected methods

Mode ⓘ
Standard

Request parameter name ⓘ
req

Route template ⓘ
Route template

Authorization level ⓘ
Anonymous

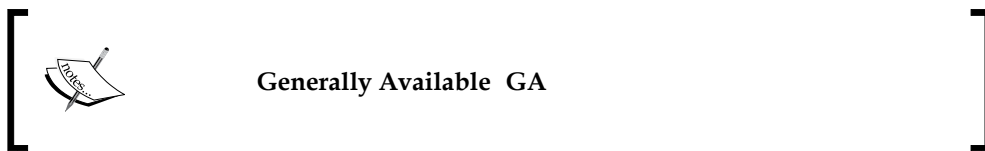
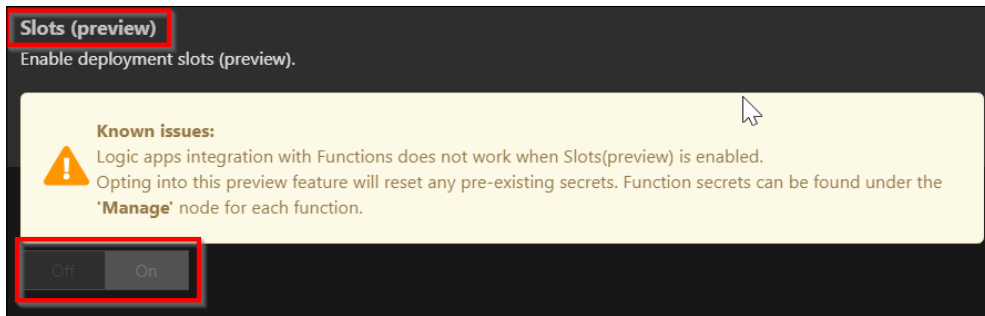
Selected HTTP methods ⓘ

| | | |
|----------------------------------|--|---------------------------------|
| <input type="checkbox"/> GET | <input checked="" type="checkbox"/> POST | <input type="checkbox"/> DELETE |
| <input type="checkbox"/> HEAD | <input type="checkbox"/> PATCH | <input type="checkbox"/> PUT |
| <input type="checkbox"/> OPTIONS | <input type="checkbox"/> TRACE | |

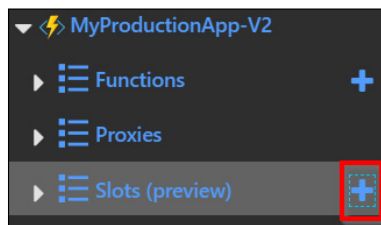
[Save](#) [Cancel](#)

Testing an Azure Function on a staged environment using deployment slots

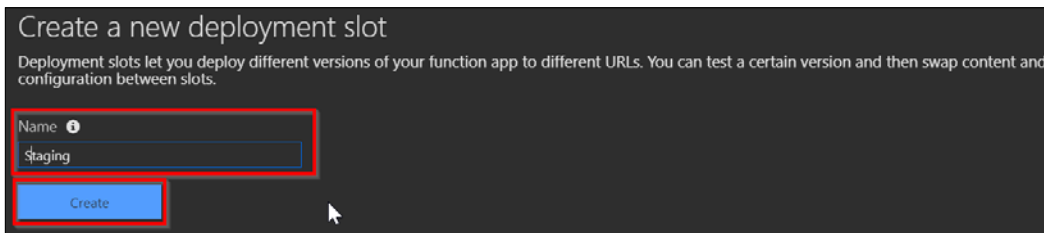
validate the application's functionality against business requirements, there are some



MyProductionApp

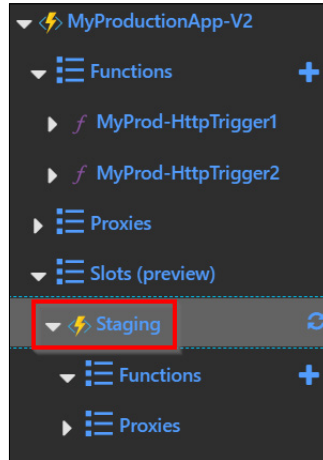



Staging



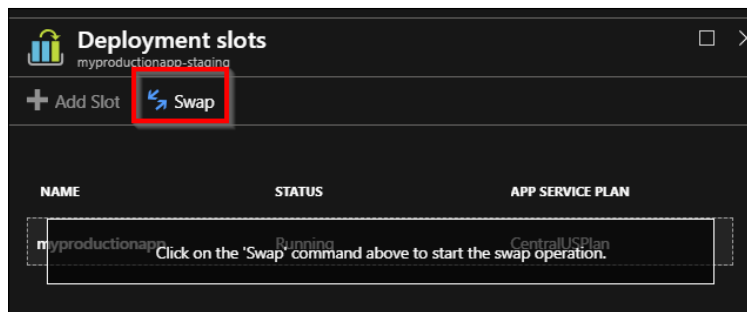
Create

Function App Settings



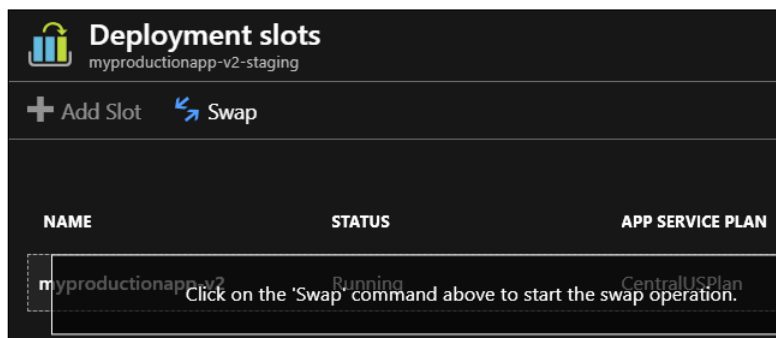
[ `https://<<functionappname>>-<<Slotname>>.azurewebsites.net>>`]

| | | | |
|---------------------|-----------------|---------------------|---------|
| MyProductionApp | | | |
| | | MyProd-HttpTrigger1 | |
| MyProd-HttpTrigger2 | MyProductionApp | | Staging |
| | production | | staging |



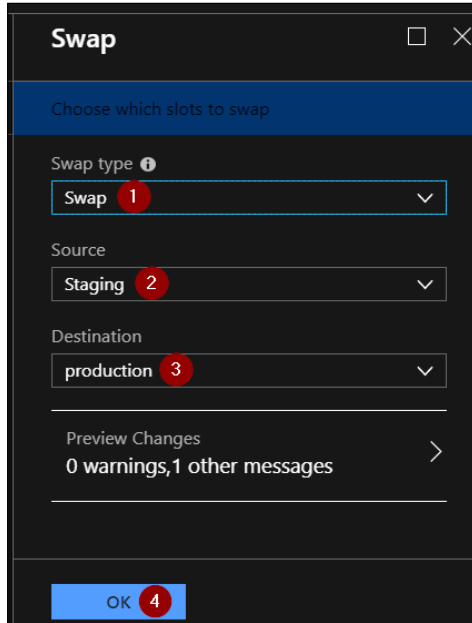
Swap

Deployment slots

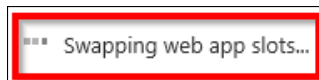


- Swap Type Swap
- Source Staging

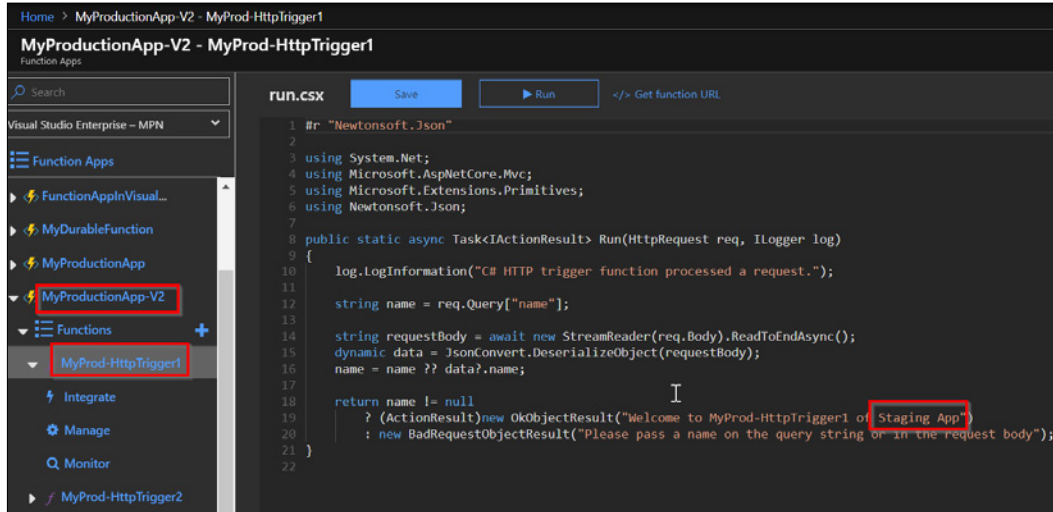
- **Destination** **production**



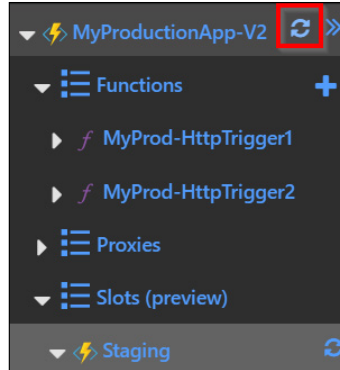
OK



run .csx cript files of the production:

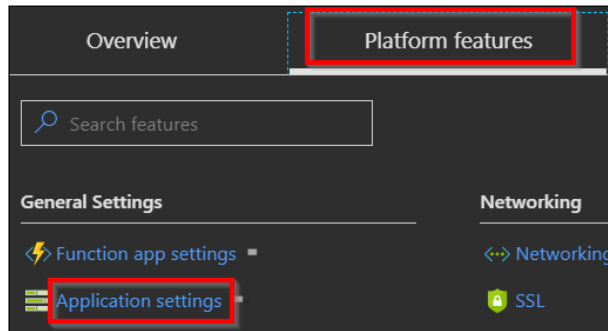


```
run.csx Save Run </> Get function URL
1 #r "Newtonsoft.Json"
2
3 using System.Net;
4 using Microsoft.AspNetCore.Mvc;
5 using Microsoft.Extensions.Primitives;
6 using Newtonsoft.Json;
7
8 public static async Task<ActionResult> Run(HttpRequest req, ILogger log)
9 {
10     log.LogInformation("C# HTTP trigger function processed a request.");
11
12     string name = req.Query["name"];
13
14     string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
15     dynamic data = JsonConvert.DeserializeObject(requestBody);
16     name = name ?? data?.name;
17
18     return name != null
19         ? (ActionResult)new OkObjectResult("Welcome to MyProd-HttpTrigger1 of Staging App")
20         : new BadRequestObjectResult("Please pass a name on the query string or in the request body");
21 }
22
```



Application settings **Database Connection Strings**
Slot Setting (slot-specific). Otherwise, **Application settings**
Database Connection Strings

Platform features

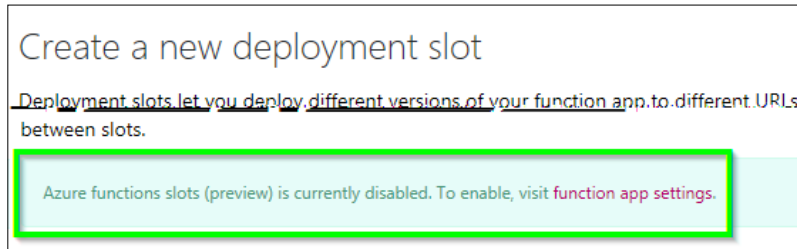


Application settings
SLOT SETTING

| APP SETTING NAME | VALUE | SLOT SETTING |
|--------------------------------|--|--------------------------|
| AzureWebJobsSecretStorageType | Blob | <input type="checkbox"/> |
| AzureWebJobsStorage | DefaultEndpointsProtocol=https;AccountName=myproductionappb23f;Acco... | <input type="checkbox"/> |
| FUNCTION_APP_EDIT_MODE | readwrite | <input type="checkbox"/> |
| FUNCTIONS_EXTENSION_VERSION | ~2 | <input type="checkbox"/> |
| FUNCTIONS_WORKER_RUNTIME | dotnet | <input type="checkbox"/> |
| WEBSITE_CONTENTAZUREFILECON... | DefaultEndpointsProtocol=https;AccountName=myproductionappb23f;Acco... | <input type="checkbox"/> |
| WEBSITE_CONTENTSHARE | myproductionapp-v2-493c497c | <input type="checkbox"/> |
| WEBSITE_NODE_DEFAULT_VERSION | 8.11.1 | <input type="checkbox"/> |



There's more...



Consumption

Load testing Azure Functions using Azure DevOps

Azure DevOps

Load Test

instances that are responsible for serving the requests.

Getting ready

<https://visualstudio.microsoft.com/>

Load Test

How to do it...

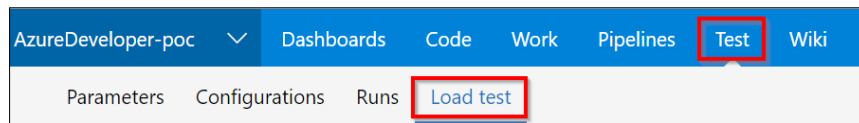
```
LoadTestHttpRequest
Authorisation Level Anonymous
run.csx
using System.Net;
using Microsoft.AspNetCore.Mvc;
public static async Task<IActionResult> Run(HttpRequest req,
ILogger log)
{
    System.Threading.Thread.Sleep(2000);
    return (ActionResult)new OkObjectResult($"Hello");
}
```

```
System.Threading.Thread.Sleep(2000);
```

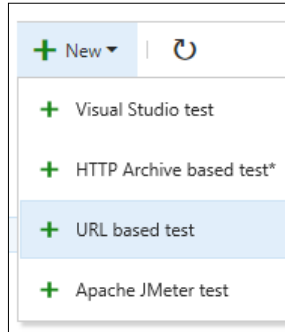
</> Get function URL

```
run.csx
```

Load test
can find it under the **Test**



New URL based test

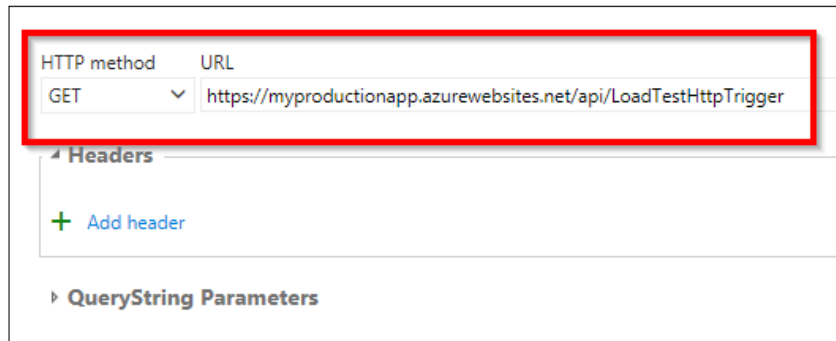


Web Scenarios

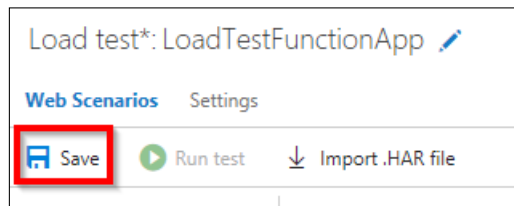


URL

field, as shown in the following screenshot:



Save



Settings
depending on your requirements:

The screenshot shows the 'Settings' tab for a load test named 'Load test*: LoadTestFunctionApp'. The 'Settings' tab is highlighted with a red box. Below the tabs, there are three buttons: 'Save' (highlighted with a red box), 'Run test', and 'Import .HAR file'. The settings are as follows:

| Setting | Value |
|------------------------------|------------------------|
| Run duration (minutes) | 20 |
| Load pattern | Step |
| Max v-users | 1000 |
| Start user count | 10 |
| Step duration (seconds) | 10 |
| Step user count (users/step) | 10 |
| Warmup duration (seconds) | 0 |
| Browser mix | IE - 60%, Chrome - 40% |

Run test

Save

The screenshot shows the top part of the Load Test interface for 'Load test*: LoadTestFunctionApp'. The 'Web Scenarios' tab is selected, and the 'Save' button is highlighted with a red box. The 'Run test' and 'Import .HAR file' buttons are also visible.

See also

Chapter 6

Monitoring and Troubleshooting Azure Serverless Services

Creating and testing Azure Functions locally using Azure CLI tools

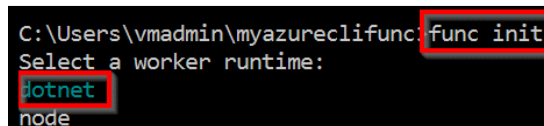
Integrated Development Environment IDE

Getting ready

<https://nodejs.org/en/download/>
[https://docs.microsoft.com/
cli/azure/install-azure-cli?view=azure-cli-latest](https://docs.microsoft.com/cli/azure/install-azure-cli?view=azure-cli-latest)

How to do it...

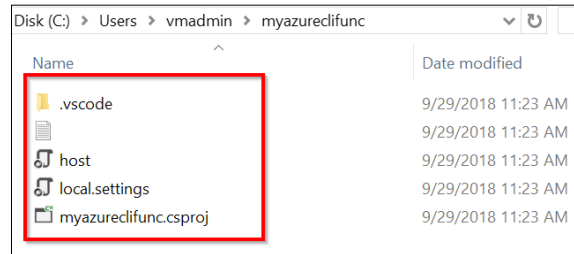
```
func init
```



```
C:\Users\vmadmin\myazureclifunc>func init  
Select a worker runtime:  
dotnet  
node
```

dotnet

will create the required files, as shown in the following screenshot:



func new

```
C:\Users\vmadmin\myazureclifunc> func new
Select a template:
QueueTrigger
HttpTrigger
BlobTrigger
TimerTrigger
DurableFunctionsOrchestration
SendGrid
EventHubTrigger
ServiceBusQueueTrigger
ServiceBusTopicTrigger
EventGridTrigger
CosmosDBTrigger
IoTHubTrigger
```

HttpTrigger

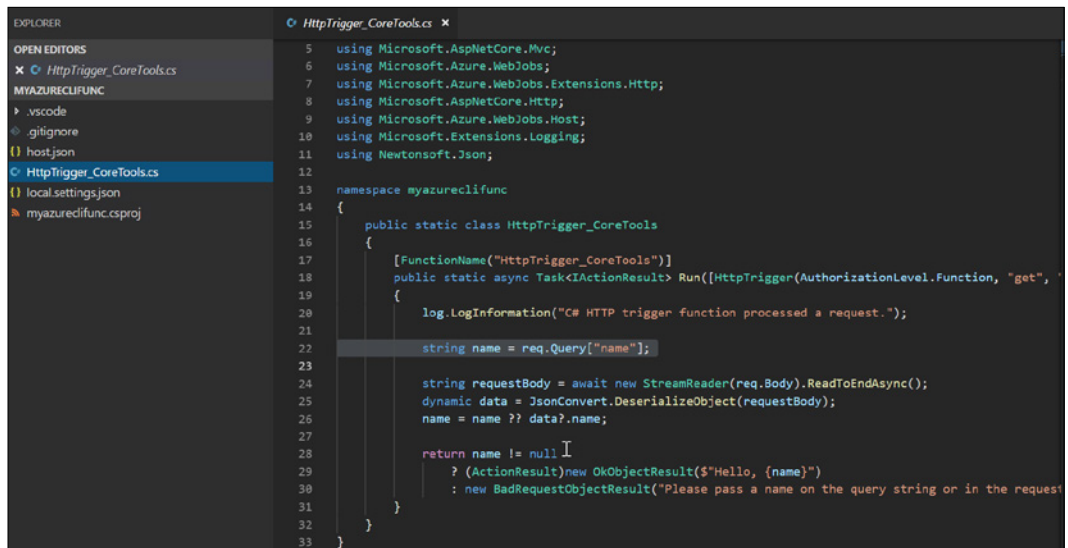
HttpTrigger type based on your requirements. You can navigate between the options

Enter

```
C:\Users\vmadmin\myazureclifunc>func new
Select a template: Function name: HttpTrigger-CoreTools
HttpTrigger-CoreTools

The function "HttpTrigger-CoreTools" was created successfully from the "HttpTrigger" template.
C:\Users\vmadmin\myazureclifunc>_
```

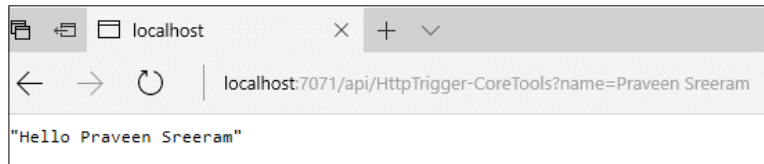
HttpTrigger



```
EXPLORER
OPEN EDITORS
  x HttpTrigger_CoreTools.cs
  x HttpTrigger_CoreTools.cs
MYAZURECLIFUNC
  .vscode
  .gitignore
  host.json
  HttpTrigger_CoreTools.cs
  local.settings.json
  myazureclifunc.csproj
HttpTrigger_CoreTools.cs x
5 using Microsoft.AspNetCore.Mvc;
6 using Microsoft.Azure.WebJobs;
7 using Microsoft.Azure.WebJobs.Extensions.Http;
8 using Microsoft.AspNetCore.Http;
9 using Microsoft.Azure.WebJobs.Host;
10 using Microsoft.Extensions.Logging;
11 using Newtonsoft.Json;
12
13 namespace myazureclifunc
14 {
15     public static class HttpTrigger_CoreTools
16     {
17         [FunctionName("HttpTrigger_CoreTools")]
18         public static async Task<IActionResult> Run([HttpTrigger(AuthorizationLevel.Function, "get",
19
20             log.LogInformation("C# HTTP trigger function processed a request.");
21
22             string name = req.Query["name"];
23
24             string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
25             dynamic data = JsonConvert.DeserializeObject(requestBody);
26             name = name ?? data?.name;
27
28             return name != null
29                 ? (ActionResult)new OkObjectResult($"Hello, {name}")
30                 : new BadRequestObjectResult("Please pass a name on the query string or in the request body");
31     }
32 }
33 }
```

Func host start --build

browser, along with a query string parameter name, as shown in



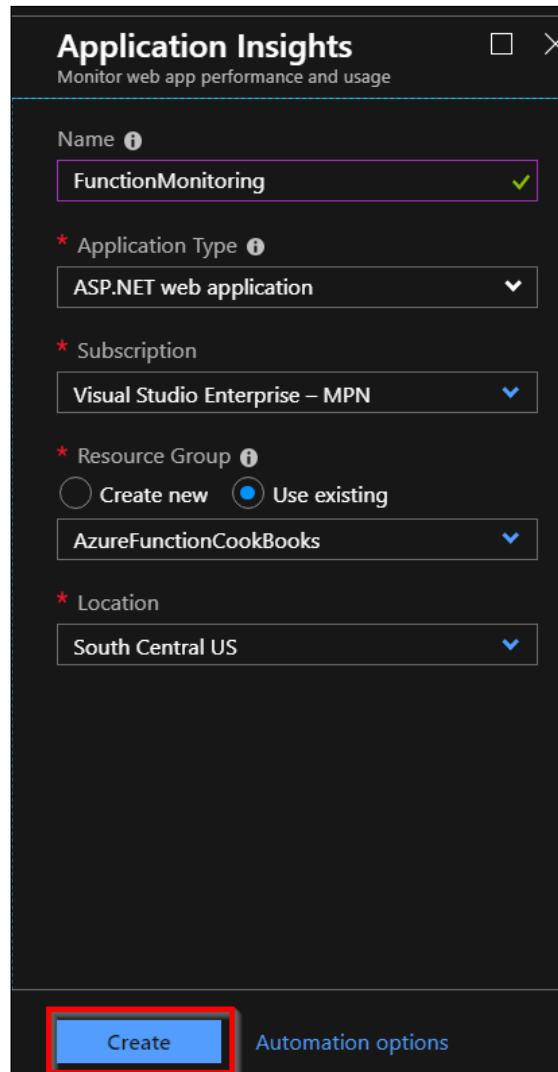
Testing and validating Azure Function responsiveness using Application Insights

It would be really helpful to get a notification when our site is not available or not responding to user requests. Azure provides a few tools to help by alerting us if the learn how to configure Application Insights to ping our Azure Function app every

Getting ready

Application Insights

Create button, and provide all the required details, as



The screenshot shows the 'Application Insights' configuration form. The title is 'Application Insights' with the subtitle 'Monitor web app performance and usage'. The form contains the following fields:

- Name:** 'FunctionMonitoring' (with a green checkmark icon).
- * Application Type:** 'ASP.NET web application' (dropdown menu).
- * Subscription:** 'Visual Studio Enterprise – MPN' (dropdown menu).
- * Resource Group:** Radio buttons for 'Create new' (unselected) and 'Use existing' (selected). Below it is a dropdown menu with 'AzureFunctionCookBooks'.
- * Location:** 'South Central US' (dropdown menu).

At the bottom left, there is a blue 'Create' button highlighted with a red rectangle. To its right is the text 'Automation options'.

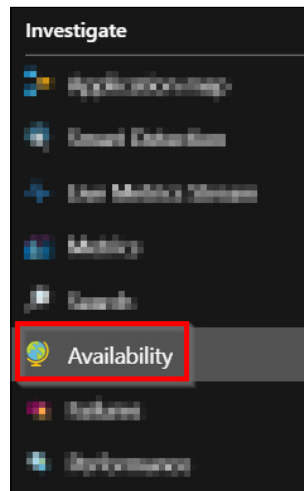
Overview

```
URL  
https://azurefunctioncookbookv2.azurewebsites.net
```

How to do it...

Availability

Add

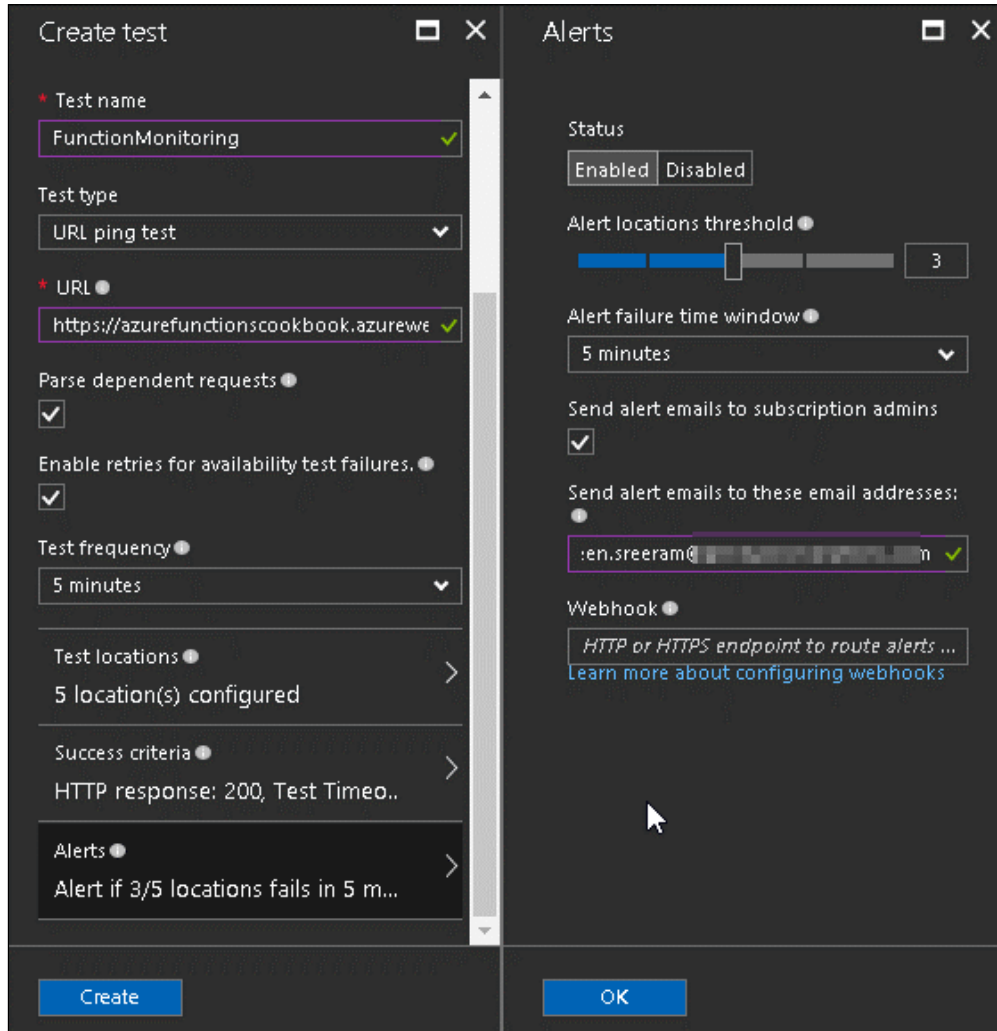


Create test blade, enter a meaningful name for your requirement

URL field of the **Create test**

Send alert emails

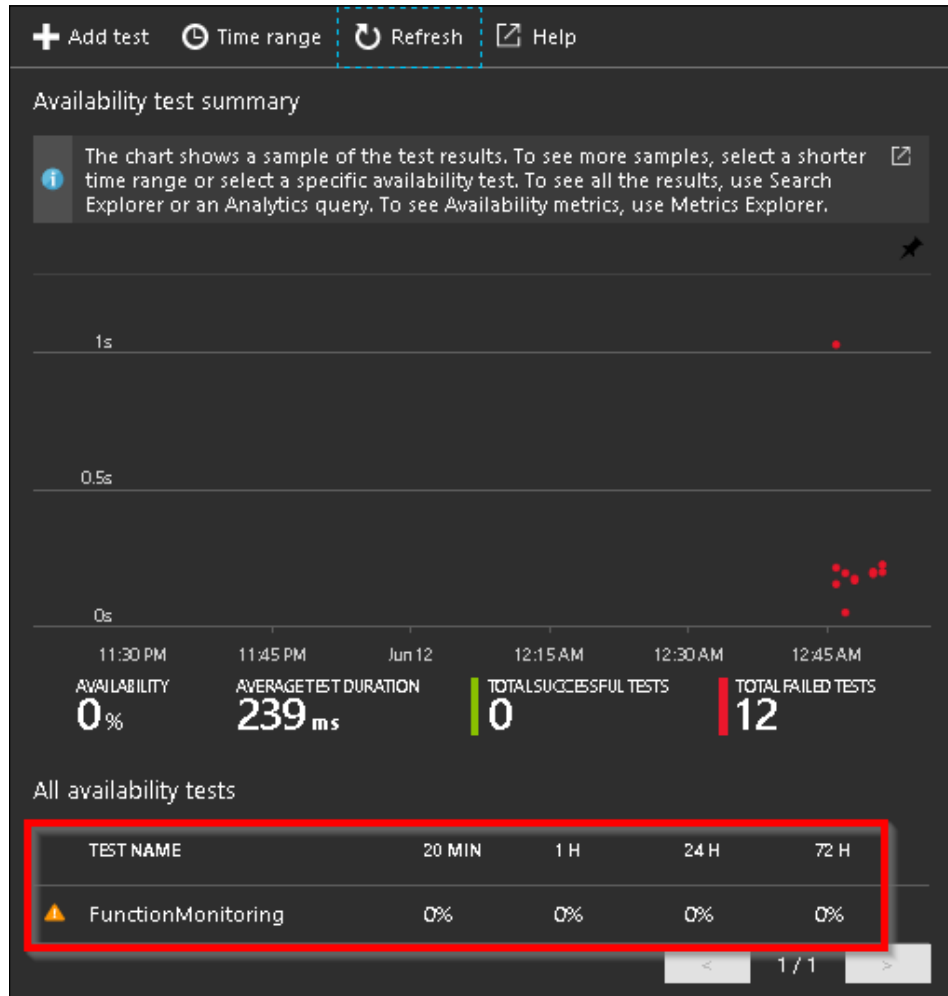
to these email addresses: field, to which an alert should be sent if the



OK Alerts Create

Create test

All availability tests

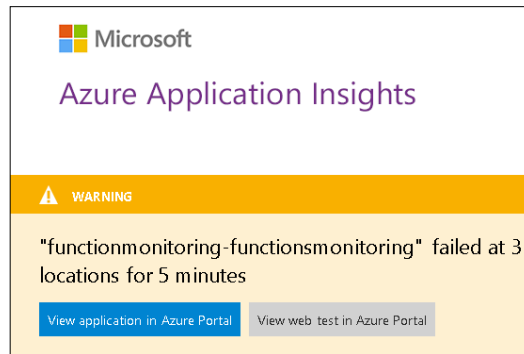


Stop

Overview

200

the app was stopped, which means the test failed and a notification should have been sent to the configured email, as shown in the following screenshot:



How it works...

Availability every five minutes from a maximum of five different locations across the world. You can configure them in the **Test Location** **Create test**

200 200
to the configurable email address.

There's more...

Test Type **Create test**
blade) if you would like to test a page or functionality that requires navigating

Developing unit tests for Azure Functions with HTTP triggers

to write automated unit test cases for this using Visual Studio Test Explorer and Moq

Getting ready

using the Moq mocking

Having a basic working knowledge of Moq is a requirement for this recipe. If you can learn more about Moq at <https://github.com/moq/moq4/wiki>

HTTPTriggerCSharpFromVS

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

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

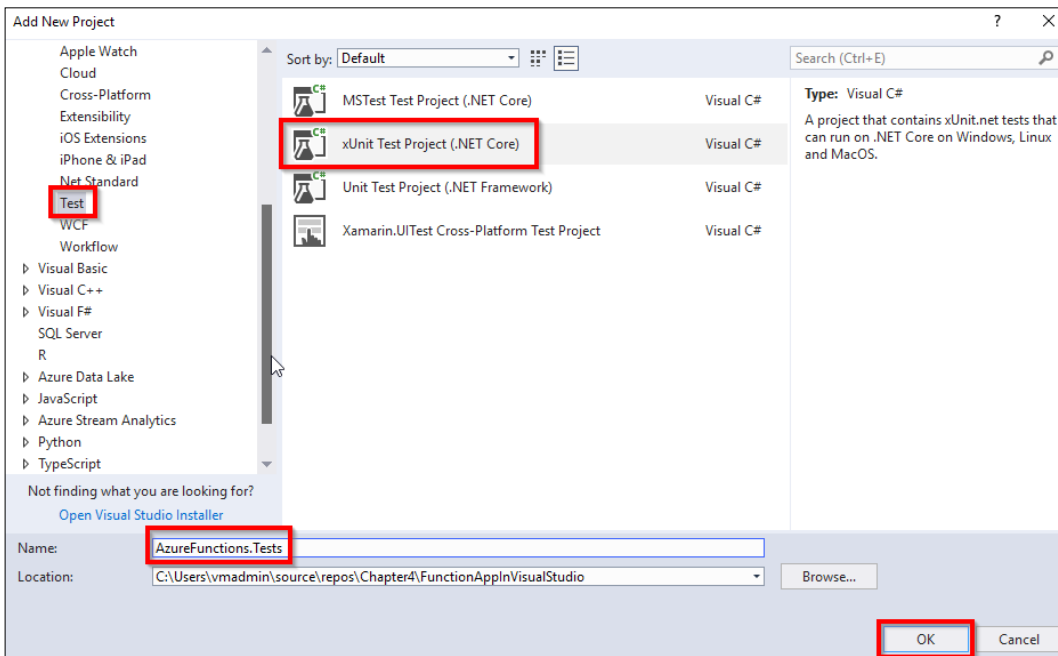
    //string requestBody = await new StreamReader(req.Body).
ReadToEndAsync();
    //dynamic data = JsonConvert.
DeserializeObject(requestBody);
    //name = name ?? data?.name;

    return name != null
        ? (ActionResult)new OkObjectResult($"Hello, {name}")
        : new BadRequestObjectResult("Please pass a name on
the query string or in the request body");
}
```

How to do it...

Add New Project
Project Type

xUnit Test Project(.NET Core) **Test**



Package Manager

- Install the Moq NuGet package using the Install-Package Moq

- `Install-Package Microsoft.AspNetCore`

FunctionAppInVisualStudio

Run

Add all the required namespaces to the Unit Test

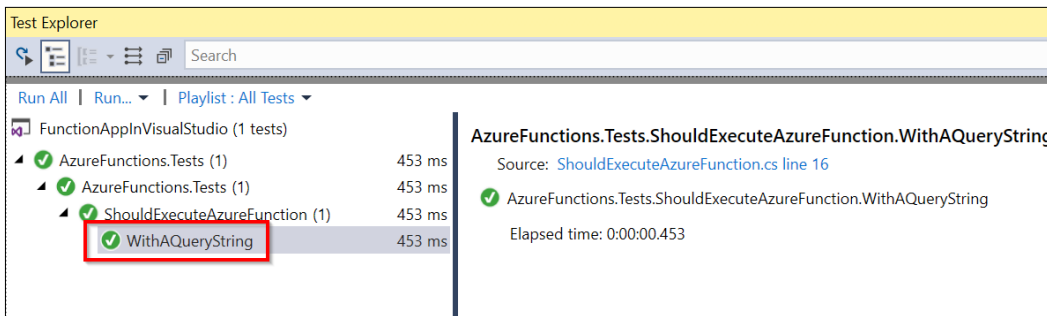
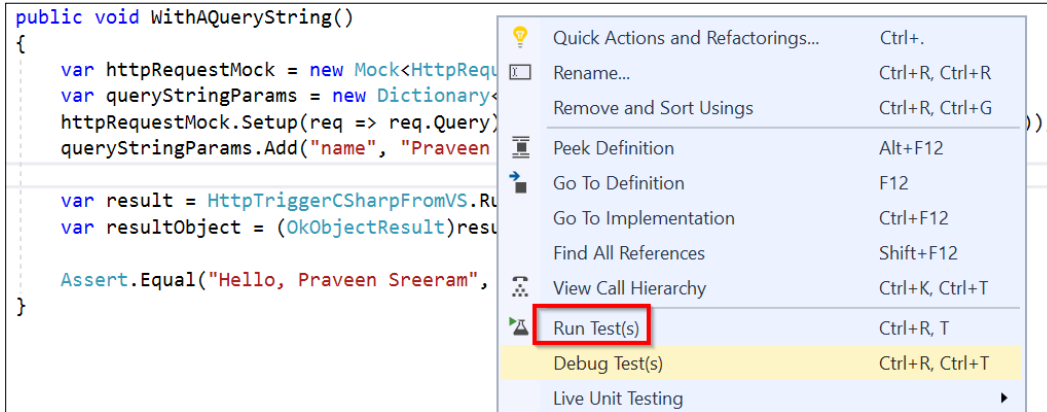
the requests, creates a Query string collection with a key named name
Praveen Sreeram

```
using FunctionAppInVisualStudio;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Http.Internal;
using Microsoft.Extensions.Primitives;
using Moq;
using System;
using System.Collections.Generic;
using Xunit;
using Microsoft.Extensions.Logging;
using System.Threading.Tasks;

namespace AzureFunctions.Tests
{
    public class ShouldExecuteAzureFunctions
    {
        [Fact]
        public async Task WithAQueryString()
        {
            var httpRequestMock = new Mock<HttpRequest>();
            var LogMock = new Mock<ILogger>();
            var queryStringParams = new Dictionary<String,
StringValues>();
            httpRequestMock.Setup(req => req.Query).Returns(new Qu
eryCollection(queryStringParams));
            queryStringParams.Add("name", "Praveen Sreeram");

            var result = await HTTPTriggerCSharpFromVS.
Run(httpRequestMock.Object, LogMock.Object);
            var resultObject = (OkObjectResult)result;
            Assert.Equal("Hello, Praveen Sreeram", resultObject.
Value);
        }
    }
}
```

Run test(s)



6

Monitoring and Troubleshooting Azure Serverless Services

Introduction


```

1 using Newtonsoft.Json;
2
3 public static async Task<IActionResult> Run(HttpRequest req, ILogger log)
4 {
5     log.LogInformation("C# HTTP trigger function processed a request.");
6
7     string firstname=req.Query["firstname"];
8     string lastname=req.Query["lastname"];
9
10    throw new Exception("Object reference not set to an instance of an object.");
11
12    string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
13    dynamic data = JsonConvert.DeserializeObject(requestBody);
14    firstname = firstname ?? data?.firstname;
15    lastname = lastname ?? data?.lastname;
16
17    return (ActionResult)new OkObjectResult($"Hello, {firstname + " " + lastname}");
18 }

```

Save

Run

Application logs

Overview Platform features Log streaming

Reconnect Copy logs Start Clear

Application logs Web Server logs

Connecting...

```

2018-10-10T07:11:38 Welcome, you are now connected to log-streaming service.
2018-10-10T07:11:41.972 [Info] Executing HTTP request: {"requestId": "8de7179f-1855-4be4-a3c9-2a9a2e8a991e", "method": "GET", "uri": "/" }
2018-10-10T07:11:41.972 [Info] Executed HTTP request: {"requestId": "8de7179f-1855-4be4-a3c9-2a9a2e8a991e", "method": "GET", "uri": "/" , "authorizationLevel": "Anonymous", "status": "NoContent"}
2018-10-10T07:11:48.162 [Info] Executing HTTP request: {"requestId": "24d12c5e-0d95-488a-8ed9-0ed92e0e3e4d", "method": "POST", "uri": "/api/HttpTriggerTestUsingPostman"}
2018-10-10T07:11:48.162 [Info,HttpTriggerTestUsingPostman] Function started (Id=63da7b8b-cac4-444f-95b3-264a089220f0)
2018-10-10T07:11:48.178 [Info,HttpTriggerTestUsingPostman] Executing 'Functions.HttpTriggerTestUsingPostman' (Reason='This function was programmatically called via the host APIs.', Id=63da7b8b-cac4-444f-95b3-264a089220f0)
2018-10-10T07:11:48.178 [Info,HttpTriggerTestUsingPostman] C# HTTP trigger function processed a request.
2018-10-10T07:11:48.178 [Error] A ScriptHost error has occurred.
2018-10-10T07:11:48.178 [Error] Object reference not set to an instance of an object.
2018-10-10T07:11:48.256 [Error,HttpTriggerTestUsingPostman] Exception while executing function: Functions.HttpTriggerTestUsingPostman. mscorlib: Exception has been thrown by the target of an invocation. f-HttpTriggerTestUsingPostman__-942327911: Object reference not set to an instance of an object.

```



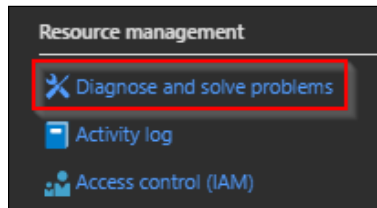
Diagnosing the entire function app

which will be helpful to quickly identify and fix any we come across. However, it is

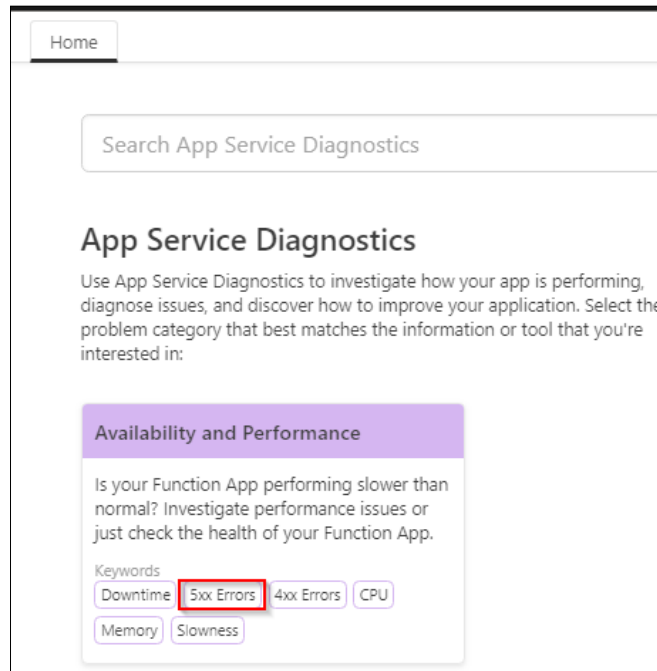
Diagnose
and solve problems

Platform features

Diagnose and solve problems

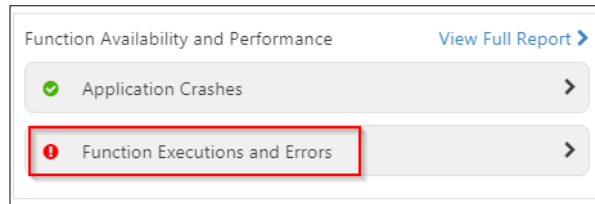


5xx Errors



Crashes Function Availability and Performance Application

Function Availability and Performance



Function Executions and Errors

Detected function(s) having execution failure rate more than 1%.

| Description | Function (by failure rate) | Total Executions | Failure Rate(%) | Top Exception |
|-------------|--|------------------|-----------------|--|
| | <code>HttpTriggerTestUsingPostman</code> | 15 | 20% | Type : System.NullReferenceException Total Count : 2 Message : Object reference not set to an instance of an object. |

Recommended Action Please review your functions code/config to see which part is causing the error and apply the fixes appropriately.

Monitor [Monitor Azure Functions Using Application Insights](#)

There's more...

AzureWebJobsHostLogs<Year><Month>

first find the `Id` field in the **Invocation details**

```
run.csx Save Run </> Get function URL
▼ Logs errors and warnings Console
2018-10-10T07:10:41.536 [Info] Function started (Id=1f833837-850c-410e-839f-c9bdecda3fc1)
2018-10-10T07:10:41.695 [Info] C# HTTP trigger function processed a request.
2018-10-10T07:10:41.695 [Info] The configuration value is 0
2018-10-10T07:10:41.695 [Info] Function completed (Success, Id=1f833837-850c-410e-839f-c9bdecda3fc1,
Duration=163ms)
2018-10-10T07:10:49.273 [Info] Function started (Id=b4f2f58e-78eb-4eea-92fe-55f76d11917f)
2018-10-10T07:10:49.273 [Info] C# HTTP trigger function processed a request.
2018-10-10T07:10:49.273 [Info] The configuration value is 0
2018-10-10T07:10:49.273 [Info] Function completed (Success, Id=b4f2f58e-78eb-4eea-92fe-55f76d11917f,
Duration=1ms)
2018-10-10T07:11:24.458 [Info] Script for function 'HttpTriggerTestUsingPostman' changed. Reloading.
2018-10-10T07:11:24.555 [Info] Compilation succeeded
2018-10-10T07:11:25.744 [Info] Function started (Id=3f18353b-ccd8-4da0-b085-7fcf4d26341a)
2018-10-10T07:11:25.869 [Info] C# HTTP trigger function processed a request.
2018-10-10T07:11:25.994 [Error] Exception while executing function: Functions.HttpTriggerTestUsingPostman.
mscorlib: Exception has been thrown by the target of an invocation. f-HttpTriggerTestUsingPostman__-942327911:
Object reference not set to an instance of an object.
2018-10-10T07:11:26.056 [Error] Function completed (Failure, Id=3f18353b-ccd8-4da0-b085-7fcf4d26341a,
Duration=297ms)
```

RowKey

AzureWebJobsHostLogs<year><month>

nth>

Search for resources

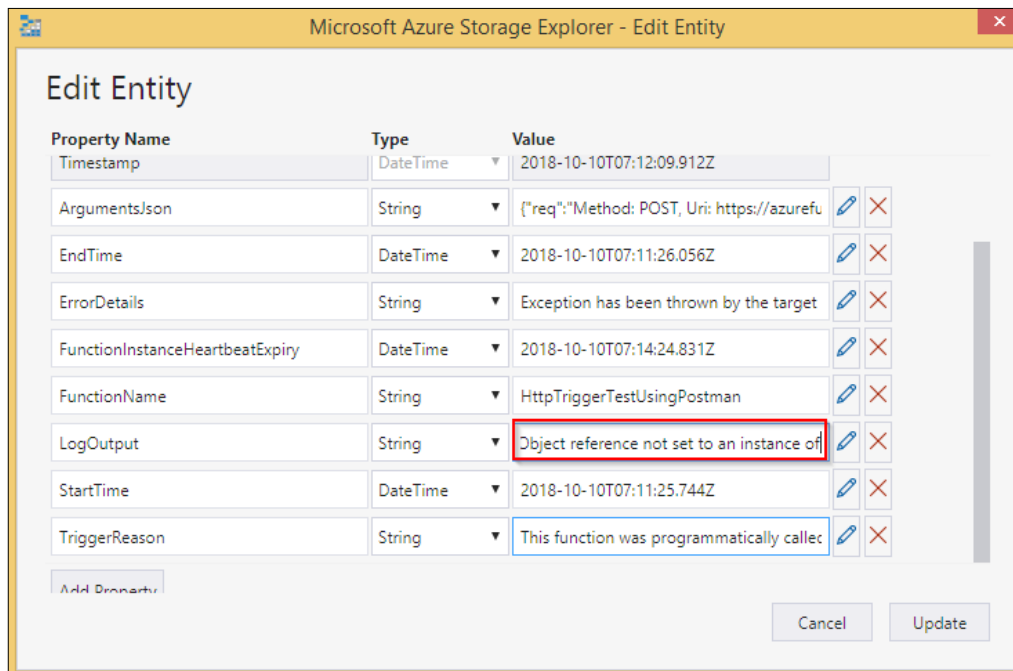
Close Query Import Export Add Edit Select all Column Options Delete Table Statistics Refresh

And/Or Field Type Operator Value

RowKey String = 3f18353b-ccd8-4da0-b085-7fcf4d26341a

Advanced Options

| PartitionKey | RowKey | Timestamp | EndTime | StartTime |
|--------------|--------------------------------------|--------------------------|--------------------------|------------|
| | 3f18353b-ccd8-4da0-b085-7fcf4d26341a | 2018-10-10T07:12:09.912Z | 2018-10-10T07:11:26.056Z | 2018-10-10 |



Integrating Azure Functions with Application Insights

Getting ready

*Functions responsiveness using Application Insights
Tools for the Validation of Azure Functions*

*Testing and validating Azure
Chapter 5 Exploring Testing*

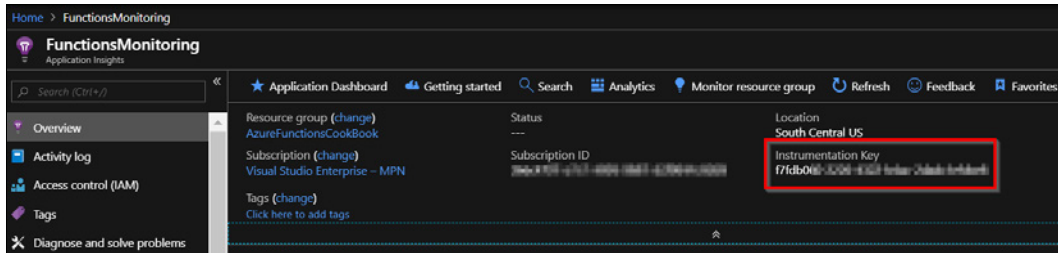
**Management Tools
Application Insights**

Create a resource

How to do it...

Instrumentation Key

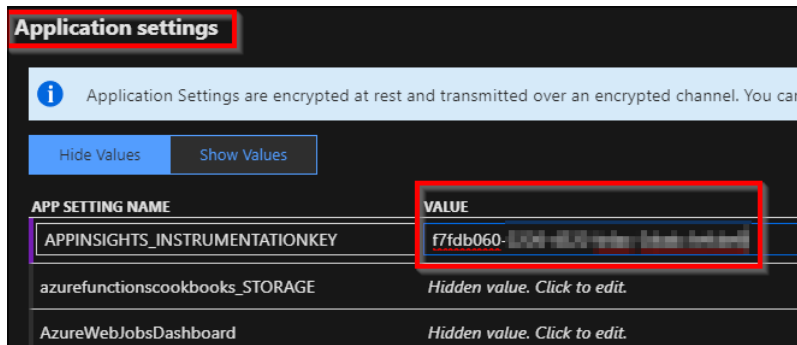
Overview



Function apps Application settings

APPINSIGHTS_INSTRUMENTATIONKEY

Save



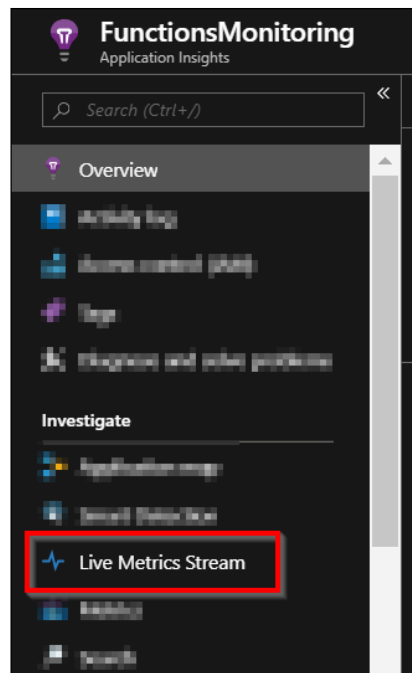
Insights Live Metrics Stream

o

Application Insights

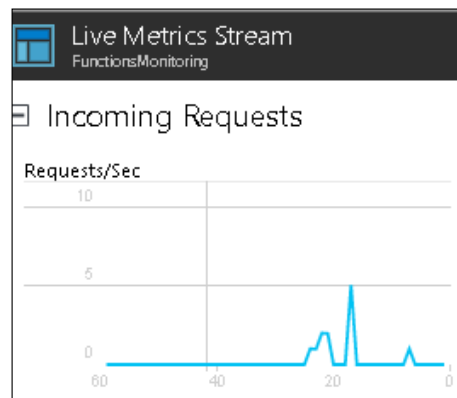
Application

Live Metrics Stream



o

Insights. You should see the live traffic going to your function app,



How it works...

Instrumentation Key

There's more...

Live Metrics Stream

Monitoring your Azure Functions

How to do it...

Logs

```
name
public static async Task<IActionResult> Run(HttpRequest req,
ILogger log)
{
    string name = req.Query["name"];
    string requestBody = await new
StreamReader(req.Body).ReadToEndAsync();
    dynamic data = JsonConvert.
DeserializeObject(requestBody);
    name = name ?? data?.name;
    log.LogInformation($"C# HTTP trigger function processed
a request with the input value {name}");
    return name != null
        ? (ActionResult)new OkObjectResult($"Hello, {name}")
        : new BadRequestObjectResult("Please pass a name on
the query string or in the request body");
}
```

name
Azure Test Run 1 Azure Test Run 2 Azure
Test Run 3

```
run.csx Save Run </> Get function URL
Log 200 events
2018-10-02T09:18:33 Welcome, you are now connected to log-streaming service.
2018-10-02T09:18:41.058 [Information] Executing 'Functions.HttpTriggerCSharp1' (Reason='This function was programmatically called via the host APIs.', Id=4ea36c7a-2ca9-443d-9fb9-951bf5f25126)
2018-10-02T09:18:41.059 [Information] CF HTTP trigger function processed a request with the input value Azure Test Run 1
2018-10-02T09:18:41.060 [Information] Executed 'Functions.HttpTriggerCSharp1' (Succeeded, Id=4ea36c7a-2ca9-443d-9fb9-951bf5f25126)
2018-10-02T09:18:53.725 [Information] Executing 'Functions.HttpTriggerCSharp1' (Reason='This function was programmatically called via the host APIs.', Id=bd62e53d-93f1-4740-a7ba-c3d06349d192)
2018-10-02T09:18:53.725 [Information] CF HTTP trigger function processed a request with the input value Azure Test Run 2
2018-10-02T09:18:53.726 [Information] Executed 'Functions.HttpTriggerCSharp1' (Succeeded, Id=bd62e53d-93f1-4740-a7ba-c3d06349d192)
2018-10-02T09:19:06.991 [Information] Executing 'Functions.HttpTriggerCSharp1' (Reason='This function was programmatically called via the host APIs.', Id=16bcd6da-4650-43e5-92a6-91258360daa1)
2018-10-02T09:19:06.991 [Information] CF HTTP trigger function processed a request with the input value Azure Test Run 3
2018-10-02T09:19:06.992 [Information] Executed 'Functions.HttpTriggerCSharp1' (Succeeded, Id=16bcd6da-4650-43e5-92a6-91258360daa1)
```

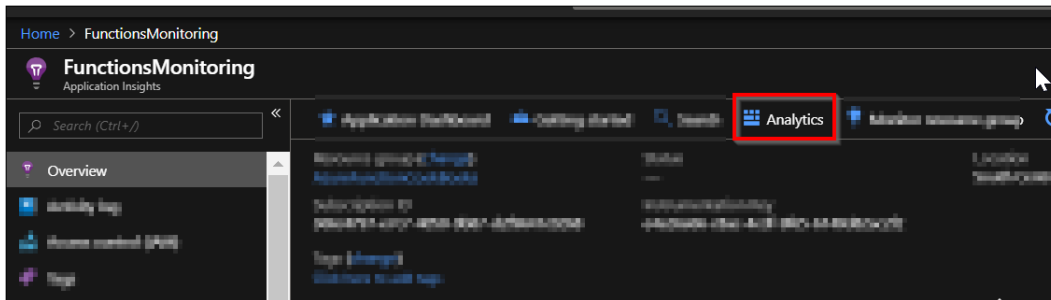
Logs

Logs

offline. That's where Application Insights comes in handy. Navigate to

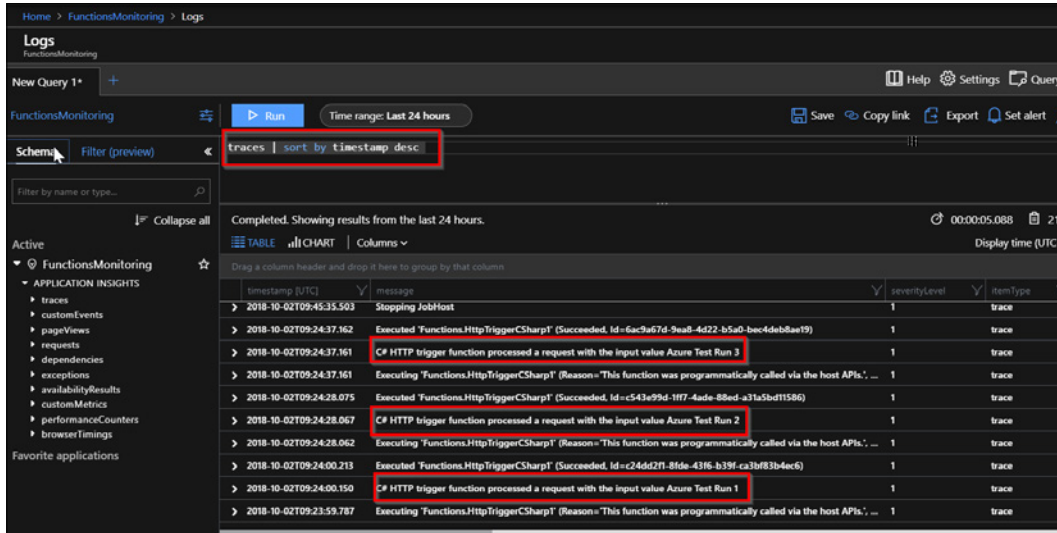
Analytics

Overview



traces | sort by timestamp

desc



How it works...

log

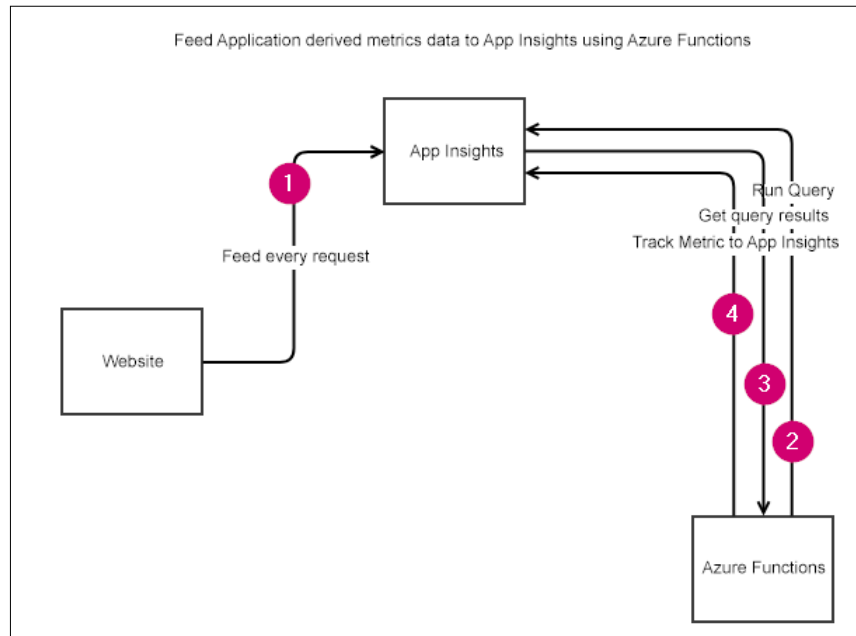
name

Analytics

Pushing custom telemetry details to Application Insights Analytics


requests per hour

requests per hour



request per hour

requests per hour for my campaign

[ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-analytics-reference>]

Getting ready

<https://docs.microsoft.com/en-us/azure/application-insights/app-insights-asp-net>



Application Insights Scheduled Analytics

FUNCTIONS_EXTENSIONS_VERSION
Application

~1
settings

Application settings

i Application Settings are encrypted at rest and transmitted over an encrypted channel.

Hide Values Show Values

| APP SETTING NAME | VALUE |
|--|------------------------------|
| APPINSIGHTS_INSTRUMENTATIONKEY | Hidden value. Click to edit. |
| AzureWebJobsStorage | Hidden value. Click to edit. |
| FUNCTIONS_EXTENSION_VERSION | ~1 |
| FUNCTIONS_WORKER_RUNTIME | Hidden value. Click to edit. |
| WEBSITE_CONTENTAZUREFILECONNECTIONSTRING | Hidden value. Click to edit. |
| WEBSITE_CONTENTSHARE | Hidden value. Click to edit. |
| WEBSITE_NODE_DEFAULT_VERSION | Hidden value. Click to edit. |

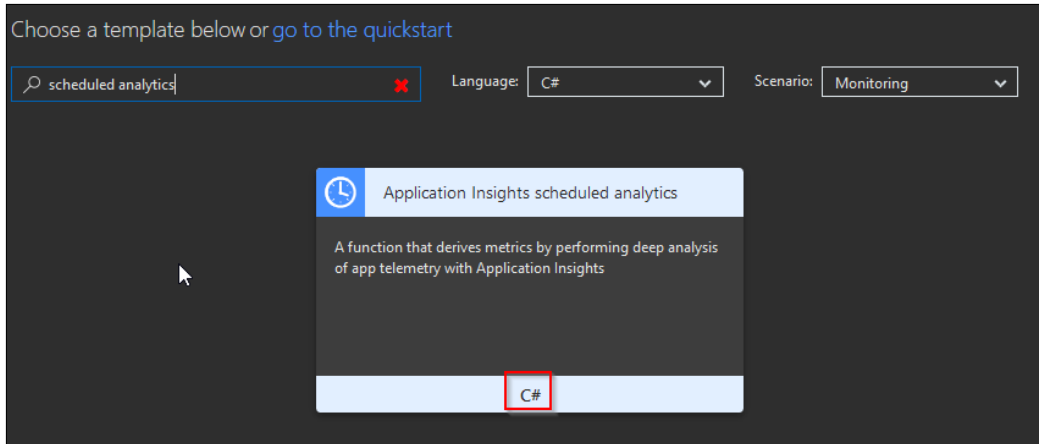
+ Add new setting

How to do it...

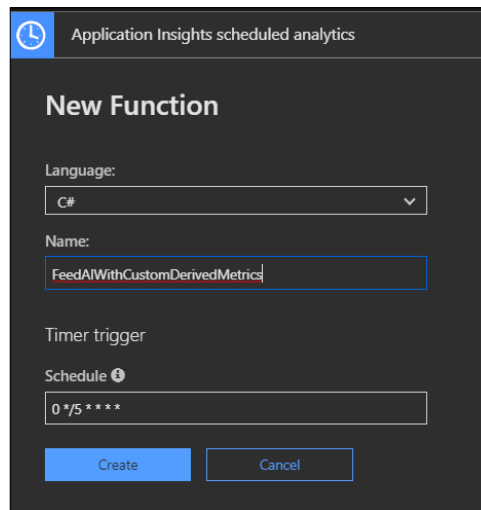
Creating an Application Insights function

Monitoring Scenario

scheduled analytics to easily find the template:



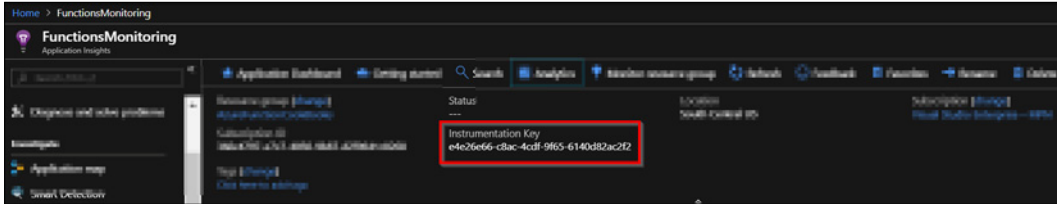
C#



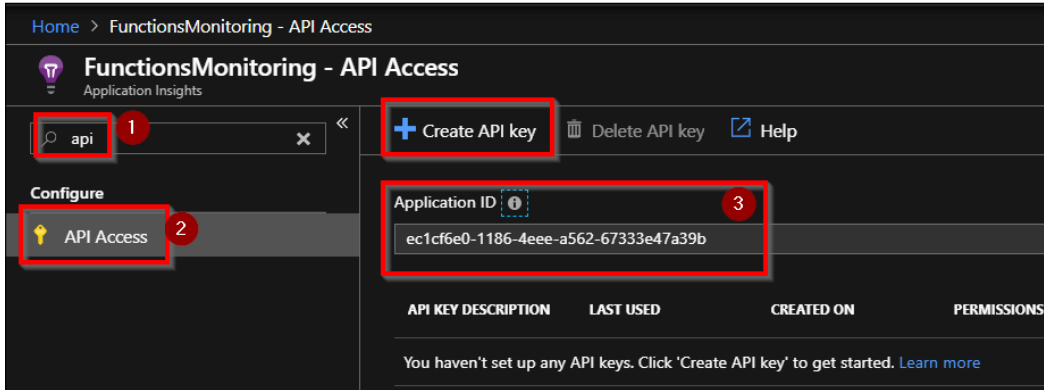
Create

Configuring access keys

Instrumentation Key Overview Instrumentation Key
AI_IKEY



API Access Application ID
Application ID
AI_APP_ID



Create API key Read telemetry
Generate key

Create API key

Create an API key to read Application Insights data.

API keys are used by applications outside the browser to access this resource.
Your API keys should be managed like passwords. Keep them secret.

Provide a description to help you identify this API key in the future. ⓘ

DerivedMetrics

Choose what this API key will allow apps to do:

- Read telemetry ⓘ
- Write annotations ⓘ
- Authenticate SDK control channel ⓘ

Generate key

AI_APP_KEY

Create API key

Create an API key to read Application Insights data.

API keys are used by applications outside the browser to access this resource.
Your API keys should be managed like passwords. Keep them secret.

Key:

ssipwfn3cmkkoq...

Make sure you copy this key now. We don't store it, and after you close this blade you won't be able to see it again.

FeedAIwithCustomDerivedMetric

| APP SETTING NAME | VALUE |
|------------------|------------------------------|
| AI_APP_ID | Hidden value. Click to edit. |
| AI_APP_KEY | Hidden value. Click to edit. |
| AI_IKEY | Hidden value. Click to edit. |

Integrating and testing an Application Insights query

per hour
Overview

Analytics
Analytics

requests

```
requests  
| where timestamp > now(-1h)  
| summarize count()
```

Run

The screenshot shows the Azure Application Insights query editor interface. At the top, there is a blue 'Run' button and a 'Time range: Set in query' dropdown. Below this, the query editor contains the following SQL query:

```
requests  
| where timestamp > now(-48h)  
| summarize count()
```

The query has been executed, and the results are displayed in a table. The table has a single column labeled 'count_'. The value '1,227' is shown in the first row of the table. The 'Run' button, the query text, and the '1,227' result are all highlighted with red boxes.

FeedAIwithCustomDerivedMetrics

Requests per hour

Save

```

29 public static async Task Run(TimerInfo myTimer, TraceWriter log)
30 {
31     if (myTimer.IsPastDue)
32     {
33         log.Warning($"[Warning]: Timer is running late! Last ran
34     }
35
36     // [CONFIGURATION_REQUIRED] update the query accordingly for
37     // be sure to run it against Application Insights Analytics
38     // output should be a number if sending derived metrics
39     // [Application Insights Analytics] https://docs.microsoft.com
40     aw = ScheduleAnalytics.Run(
41         name: "Requests per hour",
42         query: @"
43 requests
44 | where timestamp > now(-1h)
45 | summarize count()
46 ",
47         log: log
48     );
49 }

```

configured all three app
Integrate

Timer trigger (myTimer) [delete](#)

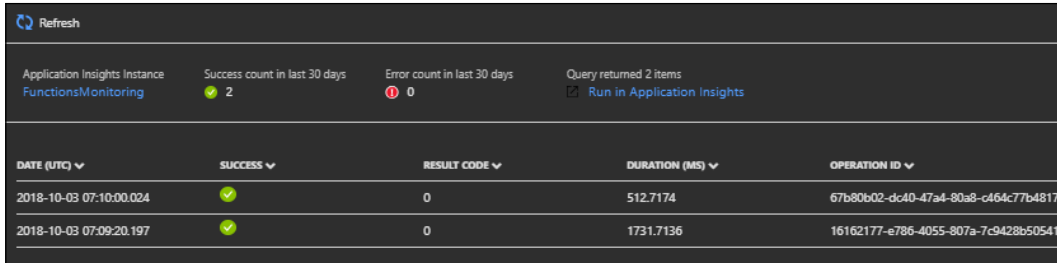
Timestamp parameter name ⓘ

myTimer

Schedule ⓘ

0*/1****

Monitor
working fine. If there are any problems, you will see an **X**
Status **Logs** **Monitor**
Invocation log

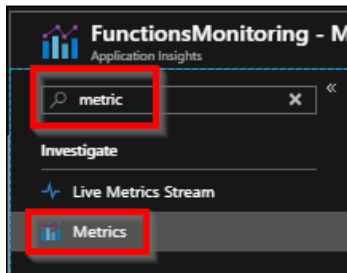


| DATE (UTC) ▾ | SUCCESS ▾ | RESULT CODE ▾ | DURATION (MS) ▾ | OPERATION ID ▾ |
|-------------------------|-----------|---------------|-----------------|--------------------------------------|
| 2018-10-03 07:10:00.024 | ✔ | 0 | 512.7174 | 67b80b02-dc40-47a4-80a8-ca64c77b4817 |
| 2018-10-03 07:09:20.197 | ✔ | 0 | 1731.7136 | 16162177-e786-4055-807a-7c9428b50541 |

Schedule

Configuring the custom derived metric report

Metrics **Overview**



Metrics Explorer is where you will find all of your analytics regarding
Metrics Explorer **Edit**
to configure your custom metric, as shown in the following screenshot
Add chart

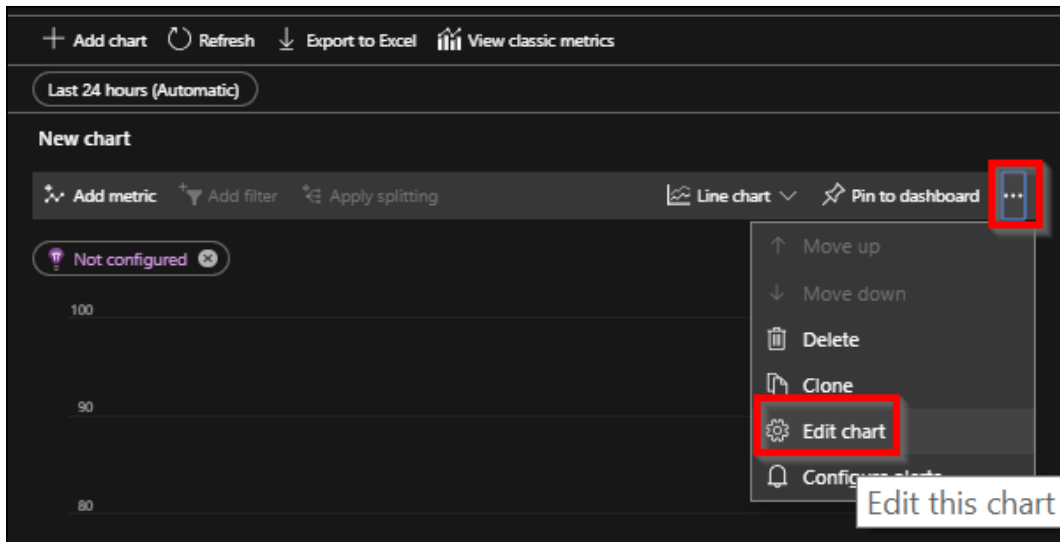
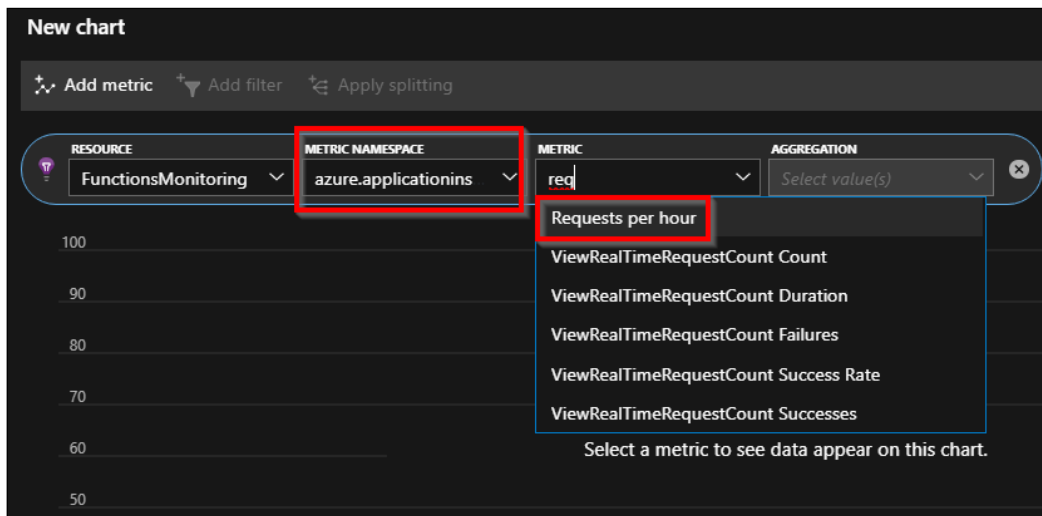
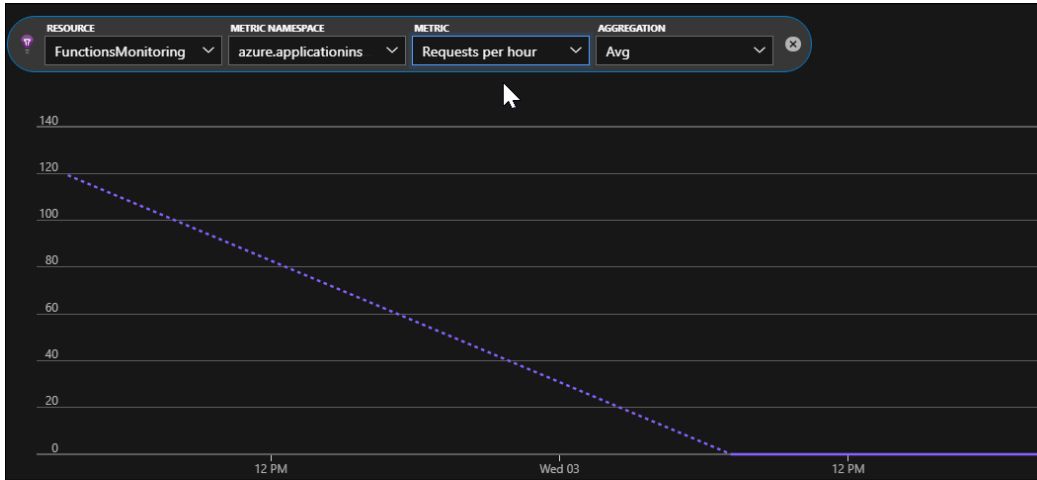


Chart details

configure your custom metric and all other details related to the chart. In
METRIC NAMESPACE **azure.applicationinsights**
Request per hour





How it works...

Scheduled Analytics

-
-
-

Application Insights

Application settings

Instrumentation Key

Metrics

Sending application telemetry details via email

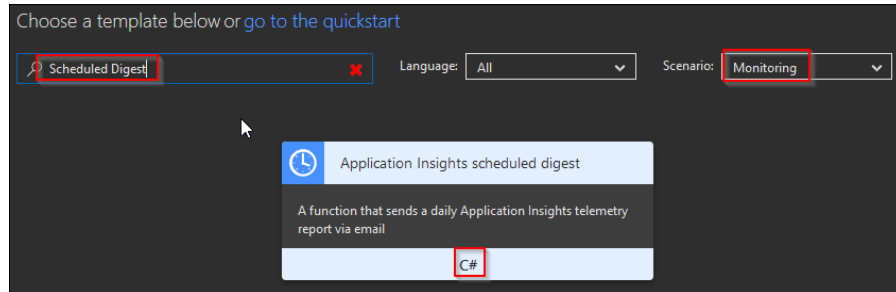
be to receive a notification

Getting ready

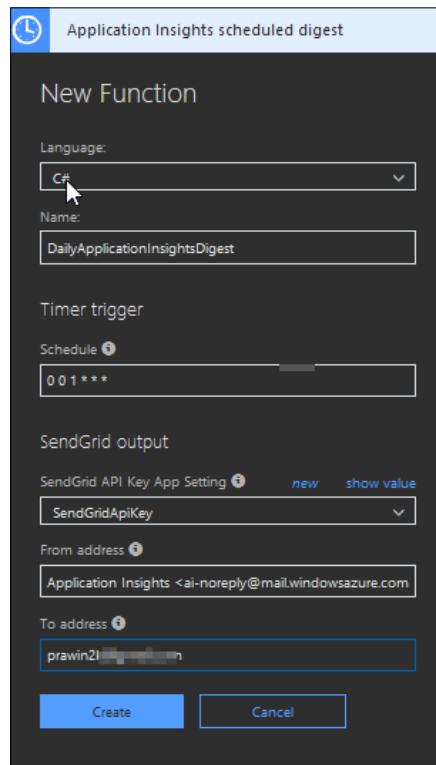
[ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-asp-net>]

How to do it...

Monitoring Scenario Application Insights scheduled digest – C#



SendGrid API Key



Create

To address



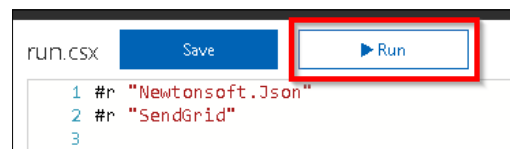
Make sure that you follow the steps in the Configuring access keys Insights recipe to configure these access keys: Application Insights **Instrumentation Key**

run.csx

```
// [CONFIGURATION_REQUIRED] configure {appName} accordingly for your app/email
string appName = "Azure Function Serverless Cookbook";
```

If you've configured all the settings properly, you will start receiving emails

Run



Run

| Azure Function Serverless Cookbook daily telemetry report 6/25/2017 | |
|--|----------------|
| The following data shows insights based on telemetry from last 24 hours. | |
| Total requests | 470,256 |
| Failed requests | 1,263 |
| Average response time | 77.78 ms |
| Total dependencies | 0 |
| Failed dependencies | 0 |
| Average response time | ----- ms |
| Total views | 0 |
| Total exceptions | 180 |
| Overall Availability | 100.0 % |
| Average response time | 457.8 ms |

How it works...

all the details and invokes the SendGrid API to send an email to the configured

There's more...

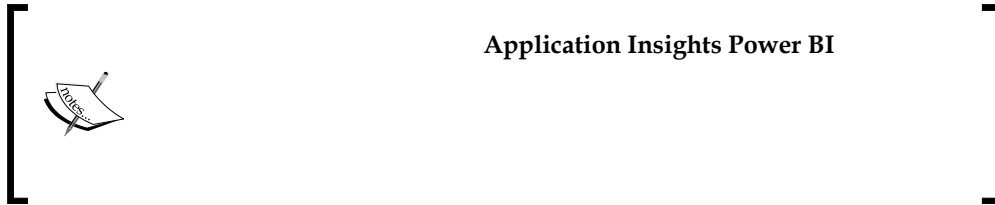
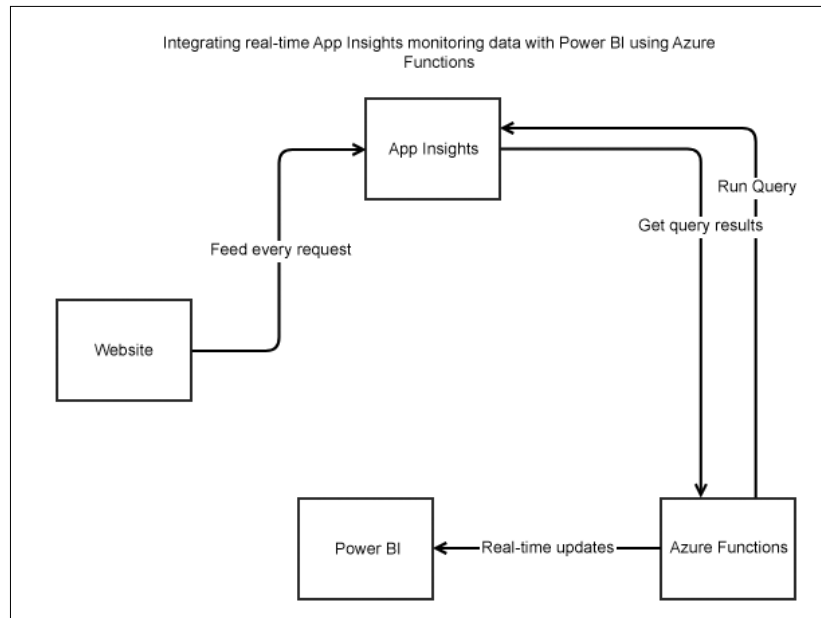
useful for monitoring application health. If you have any specific requirements for getting notification alerts, go ahead and add new queries to the `GetQueryString`

`DigestResult` `GetHtmlContentValue`

See also

Sending an email notification to the administrator of the website using the SendGrid service *Chapter 2 Working with Notifications Using the SendGrid and Twilio Services*

Integrating real-time Application Insights monitoring data with Power BI using Azure Functions



Application Insights Power BI

Getting ready

<https://powerbi.microsoft.com>

<https://docs.microsoft.com/en-us/azure/application-insights/app-insights-asp-net>

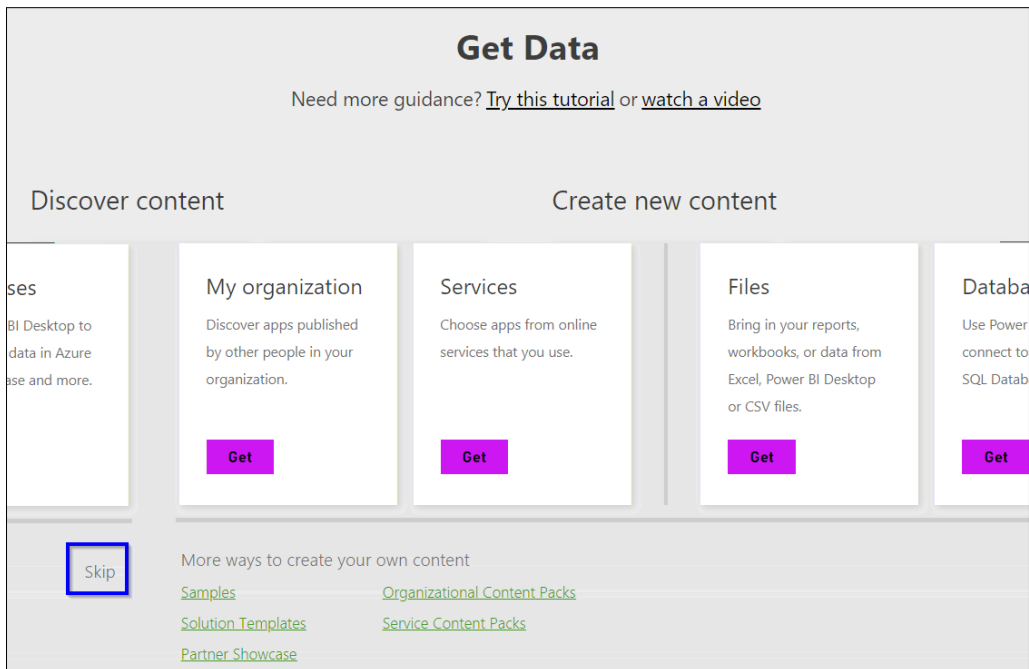


Make sure that you follow the steps in the Configuring access keys
Insights recipe to configure these access keys: Application Insights

How to do it...

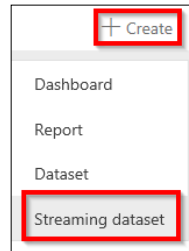
Configuring Power BI with a dashboard, a dataset and the push URI

If you are using the Power BI portal for the first time, you might have to click
Skip



Create

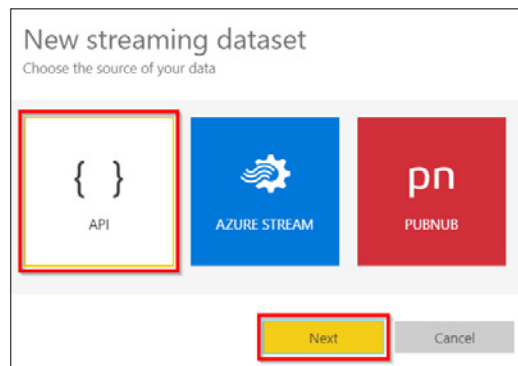
Streaming dataset



New streaming dataset

API

Next



In the next step, you need to create the fields for the streaming dataset.

just one field, named `RequestsPerSecond` **Number**
Create

Create a streaming dataset and integrate our API into your device or application to send data. [Learn more about the API.](#)

Dataset name *
Requests

Values from stream *
RequestsPerSecond Number

Enter a new value name Text

```
[  
  {  
    "RequestsPerSecond" : 98.6  
  }  
]
```

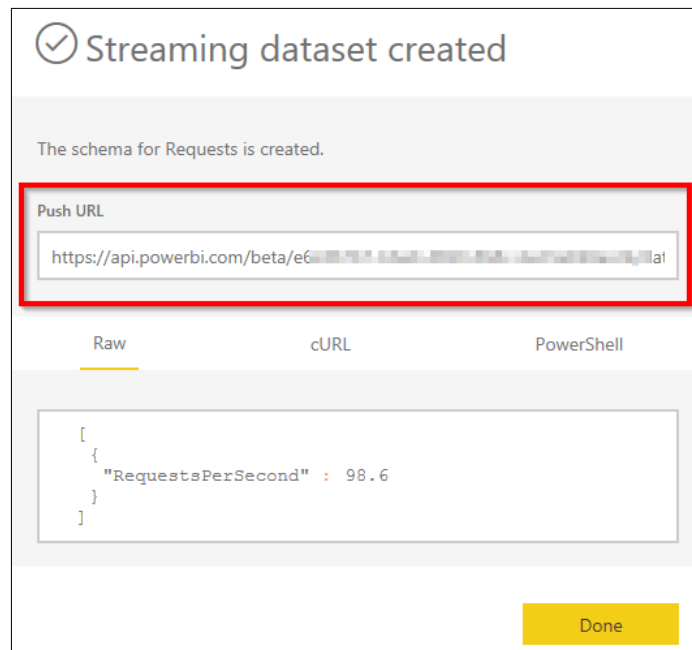
Historic data analysis

Back Create Cancel

Push URL
Push URL

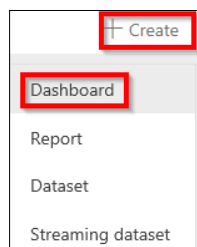
`RequestsPerSecond`

Done



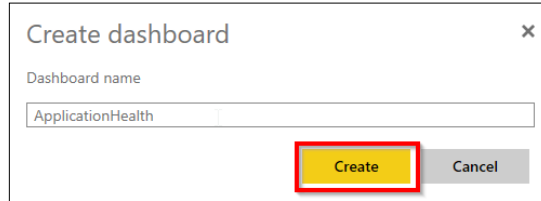
Create

Dashboard



Create dashboard

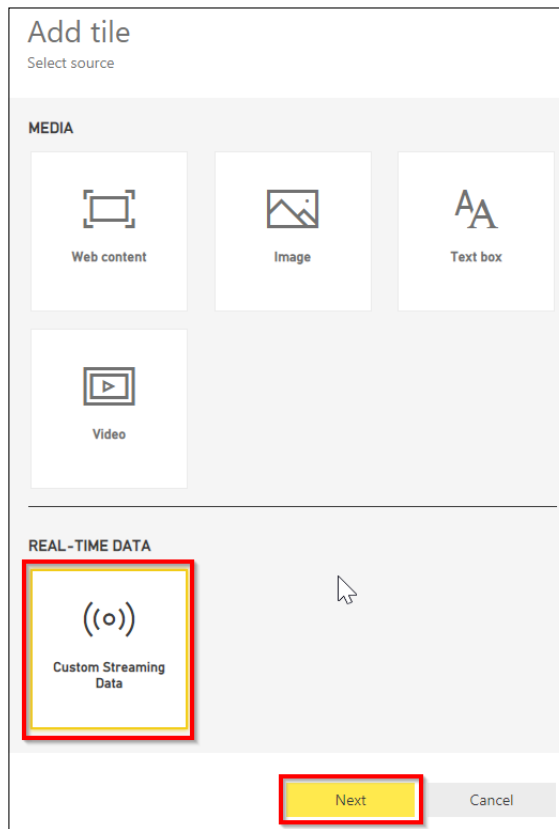
Create



A dialog box titled "Create dashboard" with a close button (X) in the top right corner. Below the title, it says "Dashboard name". There is a text input field containing the text "ApplicationHealth". At the bottom right, there are two buttons: "Create" (highlighted with a red border) and "Cancel" (disabled).

Add tile

Add tile



A dialog box titled "Add tile" with a close button (X) in the top right corner. Below the title, it says "Select source". The dialog is divided into two sections: "MEDIA" and "REAL-TIME DATA".

The "MEDIA" section contains four tiles:

- Web content (icon: rectangle with corner brackets)
- Image (icon: landscape photo)
- Text box (icon: two 'A's)
- Video (icon: play button in a square)

The "REAL-TIME DATA" section contains one tile:

- Custom Streaming Data (icon: ((o)))

At the bottom right, there are two buttons: "Next" (highlighted with a red border) and "Cancel" (disabled).

Custom Streaming Data

Next
Requests

Next

Add a custom streaming data tile

Choose a streaming dataset

+ Add streaming dataset

YOUR DATASETS

Requests

Manage datasets

Back Next Cancel

Visualisation Type Card

select the fields from the

Visualization Type

Card

Fields

RequestPerSecond

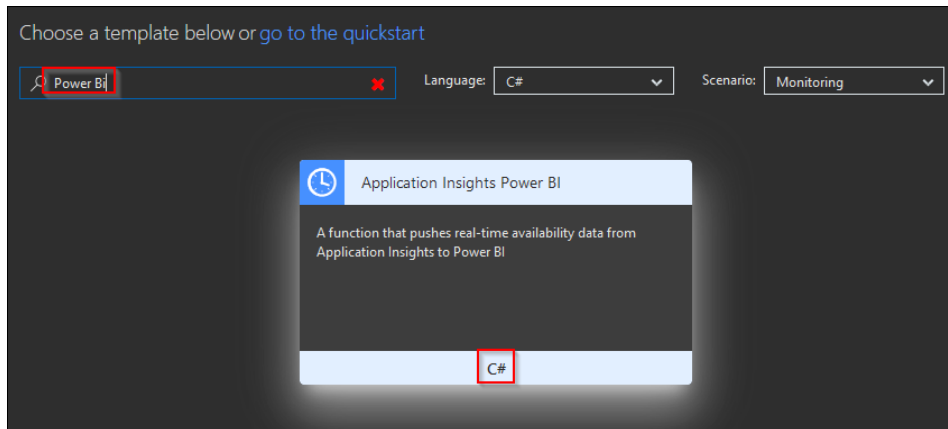
+ Add value

Manage datasets

Back Next Cancel

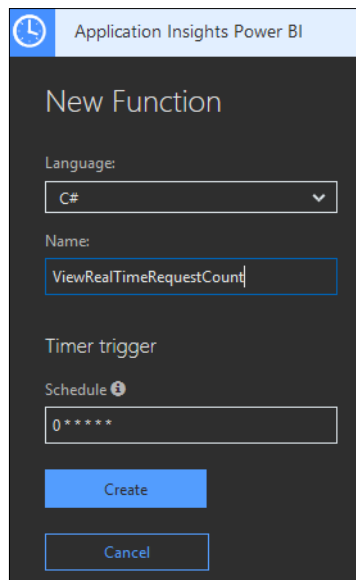
The final step is to provide a name for your tile. I have provided **RequestsPerSecond**

Creating an Azure Application Insights real-time Power BI – C# function



C#
Create

Name



configure the right value for which the analytics query should pull the data. In my case, I have provided five minutes (5m

```

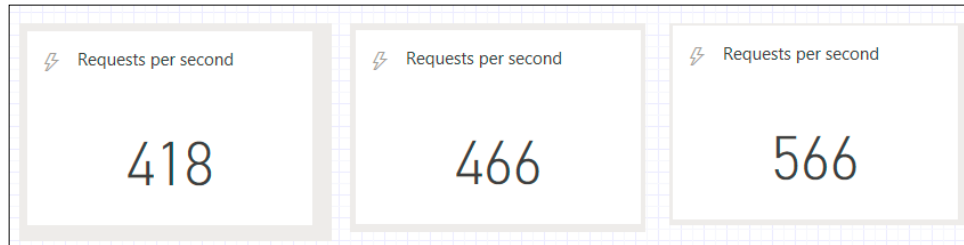
        #r "Newtonsoft.Json"
        using System.Configuration;
        using System.Text;
        using Newtonsoft.Json.Linq;
        private const string AppInsightsApi =
            "https://api.applicationinsights.io/beta/apps";
        private const string RealTimePushURL =
"PastethePushURLhere";
        private static readonly string AiAppId =
            ConfigurationManager.AppSettings["AI_APP_ID"];
        private static readonly string AiAppKey =
            ConfigurationManager.AppSettings["AI_APP_KEY"];

        public static async Task Run(TimerInfo myTimer,
TraceWriter
            log)
        {
            if (myTimer.IsPastDue)
            {
                log.Warning($"[Warning]: Timer is running late! Last
ran
                    at: {myTimer.ScheduleStatus.Last}");
            }
            await RealTimeFeedRun(
                query: @"
requests
| where timestamp > ago(5m)
| summarize passed = countif(success == true),
total = count()
| project passed
",
                log: log
            );
            log.Info($"Executing real-time Power BI run at:
                {DateTime.Now}");
        }

        private static async Task RealTimeFeedRun( string query,
TraceWriter log)
        {
            log.Info($"Feeding Data to Power BI has started at:
                {DateTime.Now}");
            string requestId = Guid.NewGuid().ToString();
            using (var httpClient = new HttpClient())
            {
                httpClient.DefaultRequestHeaders.Add("x-api-key",

```

```
        AiAppKey);
        httpClient.DefaultRequestHeaders.Add("x-ms-app",
        "FunctionTemplate");
        httpClient.DefaultRequestHeaders.Add("x-ms-client-
        request-id", requestId);
        string apiPath = $"{AppInsightsApi}/{AiAppId}/query?
        clientId={requestId}&timespan=P1D&query={query}";
        using (var httpResponse = await
        httpClient.GetAsync(apiPath))
        {
            httpResponse.EnsureSuccessStatusCode();
            var resultJson = await
            httpResponse.Content.ReadAsAsync<JToken>();
            double result;
            if (!double.TryParse(resultJson.SelectToken
            ("Tables[0].Rows[0][0]")?.ToString(), out
result))
            {
                throw new FormatException("Query must result
in a
                single metric number. Try it on Analytics
before
                scheduling.");
            }
            string postData = $"[[{"requests": "{result}"
            }]]";
            log.Verbose($"[Verbose]: Sending data:
{postData}");
            using (var response = await
            httpClient.PostAsync(RealTimePushURL, new
            ByteArrayContent(Encoding.UTF8.
GetBytes(postData))))
            {
                log.Verbose($"[Verbose]: Data sent with
response:
                {response.StatusCode}");
            }
        }
    }
}
```



How it works...

We have created the following in this specific order:

There's more...

[https://
powerbi.microsoft.com/documentation/powerbi-service-real-time-
streaming/](https://powerbi.microsoft.com/documentation/powerbi-service-real-time-streaming/)

[https://dev.
applicationinsights.io/documentation/Authorization/Rate-limits](https://dev.applicationinsights.io/documentation/Authorization/Rate-limits)


7

Developing Reliable Serverless Applications Using Durable Functions

Introduction

avoid persisting any data in the resource that is specific to any virtual machine (VM) instance provisioned to any Azure Service (for example, app service, the API and so on). If you do so, you cannot leverage some of the services, such as auto-scaling

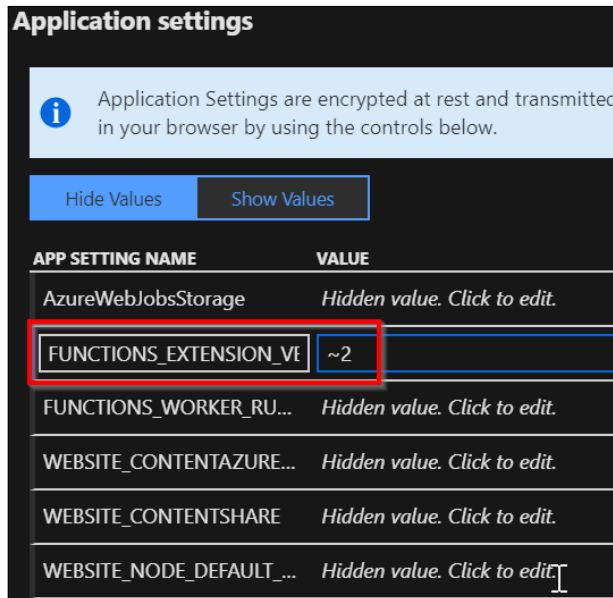
VM-specific resources, you will end up facing troubles with unexpected behaviours.

[ information about Durable Functions, check the official <https://docs.microsoft.com/en-us/azure/azure-functions/durable-functions-overview>]

Configuring Durable Functions in the Azure Management portal

Getting ready

~2 Application settings



Application settings

i Application Settings are encrypted at rest and transmitted in your browser by using the controls below.

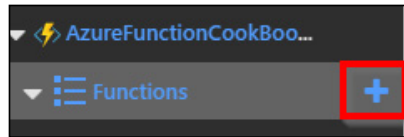
Hide Values Show Values

| APP SETTING NAME | VALUE |
|-------------------------------|------------------------------|
| AzureWebJobsStorage | Hidden value. Click to edit. |
| FUNCTIONS_EXTENSION_VE | ~2 |
| FUNCTIONS_WORKER_RU... | Hidden value. Click to edit. |
| WEBSITE_CONTENTAZURE... | Hidden value. Click to edit. |
| WEBSITE_CONTENTSHARE | Hidden value. Click to edit. |
| WEBSITE_NODE_DEFAULT_... | Hidden value. Click to edit. |

How to do it...

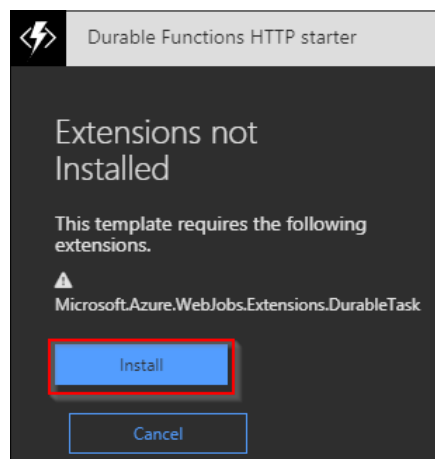
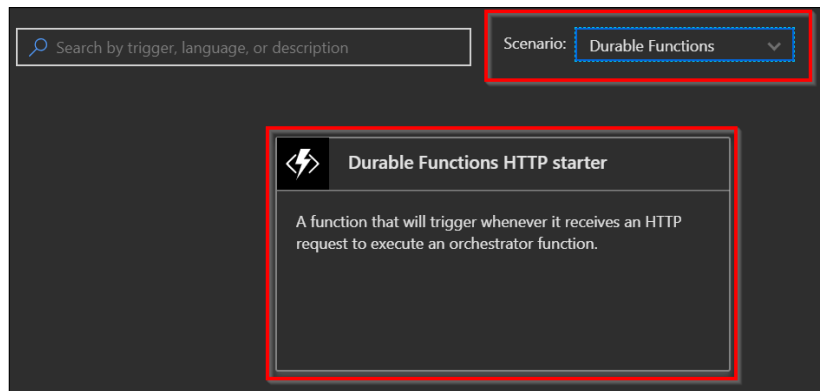
Perform

+

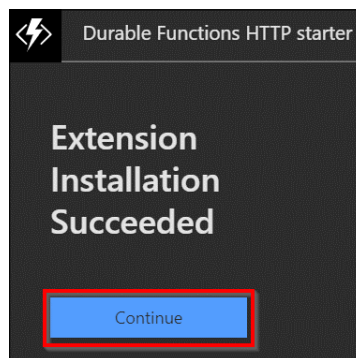
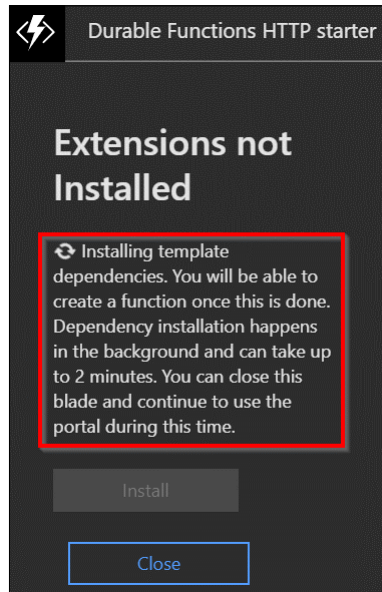


Durable Functions HTTP starter

Durable Functions Scenario



Install DurableTask



There's more...

Creating a Durable Function hello world app

it is to develop a workflow-based application using Durable Functions.

Getting ready

Download and install Postman from <https://www.getpostman.com/>

<https://docs.microsoft.com/en-us/azure/azure-functions/durable-functions-bindings>

How to do it...

Orchestrator client

Orchestrator function

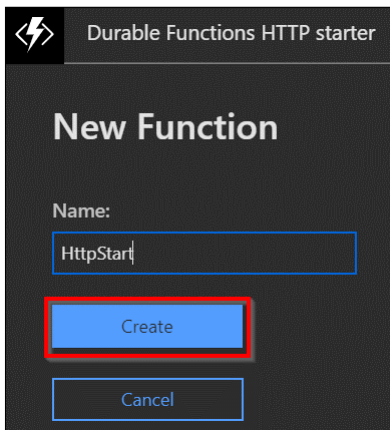
Azure Functions (named **Activity functions**),

Activity functions

Creating an HttpStart function in the Orchestrator client

Perform

Durable Functions HTTP starter function by choosing
Durable Functions Scenario
Durable Functions HTTP starter
Let's create a new HTTP function named `HttpStart`



is a HTTP trigger which accepts the name of the Function that needs to
StartNewAsync

DurableOrchestrationClient

```
#r "Microsoft.Azure.WebJobs.Extensions.DurableTask"
#r "Newtonsoft.Json"

using System.Net;

public static async Task<HttpResponseMessage> Run(
    HttpRequestMessage req,
    DurableOrchestrationClient starter,
    string functionName,
    ILogger log)
{
    // Function input comes from the request content.
    dynamic eventData = await req.Content.
ReadAsAsync<object>();
    string instanceId = await starter.
StartNewAsync(functionName,
```

```

        eventData);

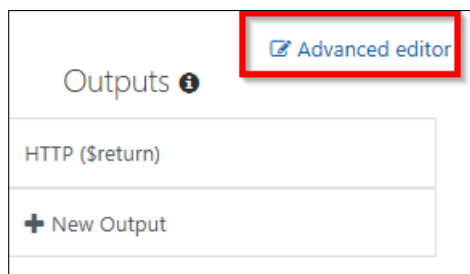
        log.LogInformation($"Started orchestration with ID =
        '{instanceId}'.");

        return starter.CreateCheckStatusResponse(req,
        instanceId);
    }

```

Integrate

Advanced editor



Advanced editor

```

{
  "bindings":
  [
    {
      "authLevel": "anonymous",
      "name": "req",
      "type": "httpTrigger",
      "direction": "in",
      "route": "orchestrators/{functionName}",
      "methods": [
        "post",
        "get"
      ]
    },
    {
      "name": "$return",
      "type": "http",
      "direction": "out"
    },
    {
      "name": "starter",

```

```
        "type": "orchestrationClient",  
        "direction": "in"  
      }  
    ]  
  }  
}
```

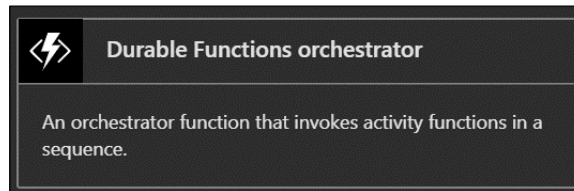
```
    [   HttpStart   ]  
    [   https://<durablefunctionname>.azurewebsites.net/api/  
    [   orchestrators/{functionName}  
    [   HttpStart  
    [   Orchestrator  
    [   {functionName}   route  
    [   attribute, defined in function.json   HttpStart   ]
```

Creating the Orchestrator function

Perform the following steps:

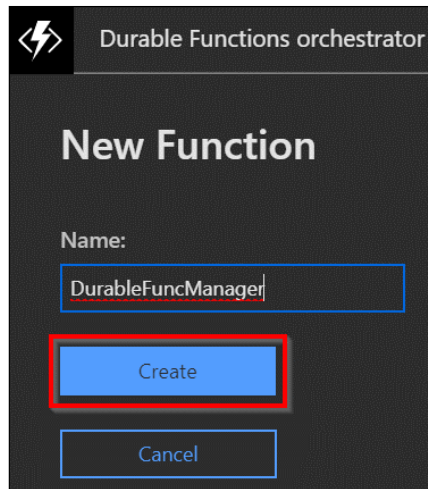
orchestrator

Durable Functions



Durable Functions orchestrator

Create



DurableFuncManager

Save

CallActivityAsync

DurableOrchestraionContext

```
#r "Microsoft.Azure.WebJobs.Extensions.DurableTask"
public static async Task<List<string>>
  Run(DurableOrchestrationContext context)
{
    var outputs = new List<string>();
    outputs.Add(await context.CallActivityAsync<string>
      ("ConveyGreeting", "Welcome Cookbook Readers"));
    return outputs;
}
```

Advanced editor

Integrate

```
{
  "bindings": [
    {
      "name": "context",
      "type": "orchestrationTrigger",
      "direction": "in"
    }
  ]
}
```

Creating an activity function

Perform the

activity **ConveyGreeting** **Durable Functions**



Save

```
#r "Microsoft.Azure.WebJobs.Extensions.DurableTask"
public static string Run(string name)
{
    return $"Hello Welcome Cookbook Readers!";
}
```

Advanced editor **Integrate**

```
{
  "bindings": [
    {
      "name": "name",
      "type": "activityTrigger",
      "direction": "in"
    }
  ]
}
```

How it works...

We first developed the Orchestrator client (in our case, `HttpStart`), which

```
DurableOrchestrationClient
```

```
StartNewAsync
```

-
-
-

VMs, then it can replace or resume service automatically

There's more...

`durabletask`

<https://github.com/Azure/>

Testing and troubleshooting Durable Functions

Getting ready

The Postman

<https://www.getpostman.com>

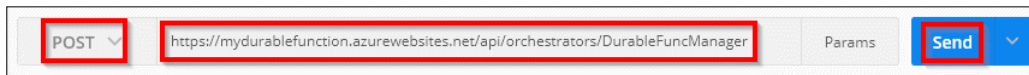
<http://storageexplorer.com>

How to do it...

Perform

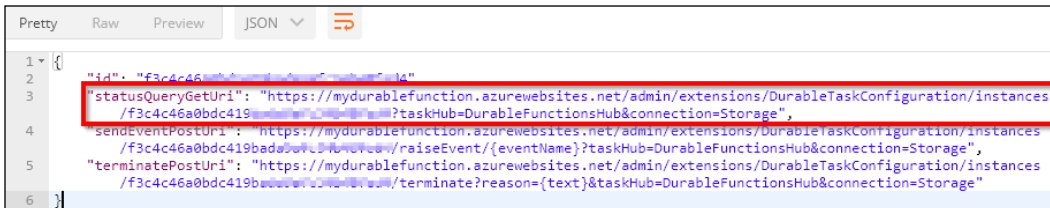
```
HttpStart
</>Get function URL {functionName}
DurableFuncManager
```

POST request using Postman:



Send

- o
- o
- o
- o



statusQueryGetUri

in a new tab within the Postman tool. Once the new tab is opened, click **Send**

```

1 {
2   "instanceId": "53a8eca4e20b43e59718516397e94f0e",
3   "runtimeStatus": "Completed",
4   "input": null,
5   "customStatus": null,
6   "output": [
7     "Hello Welcome Cookbook Readers!"
8   ],
9   "createTime": "2018-10-21T16:46:42Z",
10  "lastUpdateTime": "2018-10-21T16:46:49Z"
11 }

```

runtimeStatus Completed
 Postman, as shown in the preceding screenshot. You will also get eight

| EventType | ExecutionId |
|-----------------------|----------------------------------|
| OrchestratorStarted | a426ec4dabe44527a6aeae14290802c1 |
| ExecutionStarted | a426ec4dabe44527a6aeae14290802c1 |
| TaskScheduled | a426ec4dabe44527a6aeae14290802c1 |
| OrchestratorCompleted | a426ec4dabe44527a6aeae14290802c1 |
| OrchestratorStarted | a426ec4dabe44527a6aeae14290802c1 |
| TaskCompleted | a426ec4dabe44527a6aeae14290802c1 |
| ExecutionCompleted | a426ec4dabe44527a6aeae14290802c1 |
| OrchestratorCompleted | a426ec4dabe44527a6aeae14290802c1 |

Monitor

Implementing multithreaded reliable applications using Durable Functions

Assume that we have five customers (whose IDs are 1, 2, 3, 4 and 5 respectively) who approached us to generate a huge number of barcodes (say around 50,000).

auto-heals when the VM on which they are hosted goes down or is restarted.

Getting ready

The Postman tool, available from <https://www.getpostman.com/storageexplorer.com/>
<http://>

How to do it...

GenerateBARCode

- GetAllCustomers
- CreateBARCodeImagesPerCustomer

customer, we will randomly generate a number less than 50,000 and

Creating the Orchestrator function

Perform the

```

GenerateBARCode
Durable
Functions Orchestrator template
Save
GetAllCustomers

```

```

#r "Microsoft.Azure.WebJobs.Extensions.DurableTask"
public static async Task<int> Run(
    DurableOrchestrationContext context)
{
    int[] customers = await
        context.CallActivityAsync<int[]>("GetAllCustomers", null);

    var tasks = new Task<int>[customers.Length];
    for (int nCustomerIndex = 0; nCustomerIndex < customers.
        Length; nCustomerIndex++)
    {
        tasks[nCustomerIndex] = context.CallActivityAsync<int>
            ("CreateBARCodeImagesPerCustomer",
            customers[nCustomerIndex]);
    }
}

```

```
    }

    await Task.WhenAll(tasks);
    int nTotalItems = tasks.Sum(item => item.Result);

    return nTotalItems;
}
```

Advanced editor **Integrate**

```
{
  "bindings": [
    {
      "name": "context",
      "type": "orchestrationTrigger",
      "direction": "in"
    }
  ]
}
```

Creating a GetAllCustomers activity function

Perform the

Functions Activity template GetAllCustomers **Durable**
Save

```
#r "Microsoft.Azure.WebJobs.Extensions.DurableTask"
public static int[] Run(string name)
{
    int[] customers = new int[] {1,2,3,4,5};
    return customers;
}
```

Advanced editor **Integrate**

```
{
  "bindings": [
    {
      "name": "name",
      "type": "activityTrigger",
    }
  ]
}
```

```

        "direction": "in"
    }
  ]
}

```

Creating a CreateBarcodeImagesPerCustomer activity function

Perform the

CreateBarcodeImagesPerCustomer Durable Functions Activity template Save

```

#r "Microsoft.Azure.WebJobs.Extensions.DurableTask"
#r "Microsoft.WindowsAzure.Storage"
using Microsoft.WindowsAzure.Storage.Blob;

public static async Task<int> Run(DurableActivityContext
customerContext, ILogger log)
{
    int ncustomerId = Convert.ToInt32
(customerContext.GetInput<string>());
    Random objRandom = new Random(Guid.NewGuid().
GetHashCode());
    int nRandomValue = objRandom.Next(50000);
    for(int nProcessIndex = 0;nProcessIndex<=nRandomValue;
nProcessIndex++)
    {
        log.LogInformation($" running for {nProcessIndex}");
    }
    return nRandomValue;
}

```

Advanced editor Integrate

```

{
  "bindings": [
    {
      "name": "customerContext",
      "type": "activityTrigger",
      "direction": "in"
    }
  ]
}

```

Let's run the function using Postman. We will be stopping the App Service to simulate a restart of the VM where the function would be running and to see

POST request using Postman, as shown as follows:



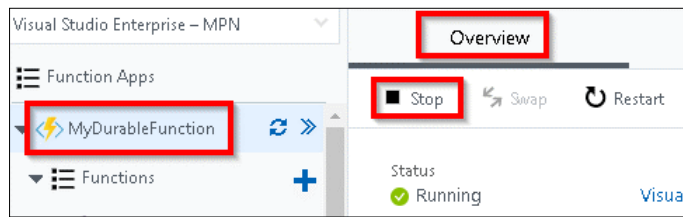
Send

statusQueryGetURi
statusQueryGetURi

Postman tool. Once the new tab is opened, click on the **Send**

Overview

Stop



DurableFunctionsHubHistory

| EventType | ExecutionId | IsPlayed | _Timestamp | Input | Name |
|-----------------------|----------------------------------|----------|--------------------------|-------|-----------------|
| OrchestratorStarted | cf36ba2c05344390896fe2dcbb898189 | false | 2018-01-19T16:34:56.005Z | | |
| ExecutionStarted | cf36ba2c05344390896fe2dcbb898189 | true | 2018-01-19T16:34:46.159Z | null | GenerateBARCode |
| TaskScheduled | cf36ba2c05344390896fe2dcbb898189 | false | 2018-01-19T16:35:06.009Z | | GetAllCustomers |
| OrchestratorCompleted | cf36ba2c05344390896fe2dcbb898189 | false | 2018-01-19T16:35:06.010Z | | |

After some time – in my case, after just five minutes – go back to the **Overview**

| EventType | ExecutionId | IsPlayed | _Timestamp | Input | Name |
|-----------------------|----------------------------------|----------|--------------------------|-------|--------------------------------|
| TaskScheduled | cf36ba2c05344390896fe2dcbb898189 | false | 2018-01-19T16:42:23.523Z | | CreateBARCodeImagesPerCustomer |
| OrchestratorStarted | cf36ba2c05344390896fe2dcbb898189 | false | 2018-01-19T16:34:56.005Z | | GetAllCustomers |
| TaskScheduled | cf36ba2c05344390896fe2dcbb898189 | false | 2018-01-19T16:35:06.009Z | | |
| OrchestratorCompleted | cf36ba2c05344390896fe2dcbb898189 | false | 2018-01-19T16:35:06.010Z | | |
| OrchestratorStarted | cf36ba2c05344390896fe2dcbb898189 | false | 2018-01-19T16:41:51.507Z | | |
| TaskCompleted | cf36ba2c05344390896fe2dcbb898189 | true | 2018-01-19T16:41:50.516Z | | |
| ExecutionStarted | cf36ba2c05344390896fe2dcbb898189 | true | 2018-01-19T16:34:46.159Z | null | GenerateBARCode |
| TaskScheduled | cf36ba2c05344390896fe2dcbb898189 | false | 2018-01-19T16:42:23.523Z | | CreateBARCodeImagesPerCustomer |
| TaskScheduled | cf36ba2c05344390896fe2dcbb898189 | false | 2018-01-19T16:42:23.523Z | | CreateBARCodeImagesPerCustomer |
| TaskScheduled | cf36ba2c05344390896fe2dcbb898189 | false | 2018-01-19T16:42:23.523Z | | CreateBARCodeImagesPerCustomer |
| TaskScheduled | cf36ba2c05344390896fe2dcbb898189 | false | 2018-01-19T16:42:23.523Z | | CreateBARCodeImagesPerCustomer |

How it works...

means that even if the VMs crashed or are restarted while the function is running,

checkpointing replaying



<https://docs.microsoft.com/en-us/azure/azure-functions/durable-functions-checkpointing-and-replay>

There's more...

404 Not Found

statusQueryGetUri

DurableFunctionsHubHistory

WEBSITE_CONTENTSHARE mydurablefunction8c72

You can find the storage account name in the **Application settings**

8


Bulk Import of Data Using Azure Durable Functions and Cosmos DB

Introduction

lets you write workflows by writing the minimum lines of code.

Orchestrator

Activity trigger

[ <https://docs.microsoft.com/en-us/azure/azure-functions/durable-functions-overview>]

Business problem

In general, every organisation will definitely be using various applications hosted

files) are used for exporting data from one application and importing it into another

You might think that exporting an Excel file from one application to another would

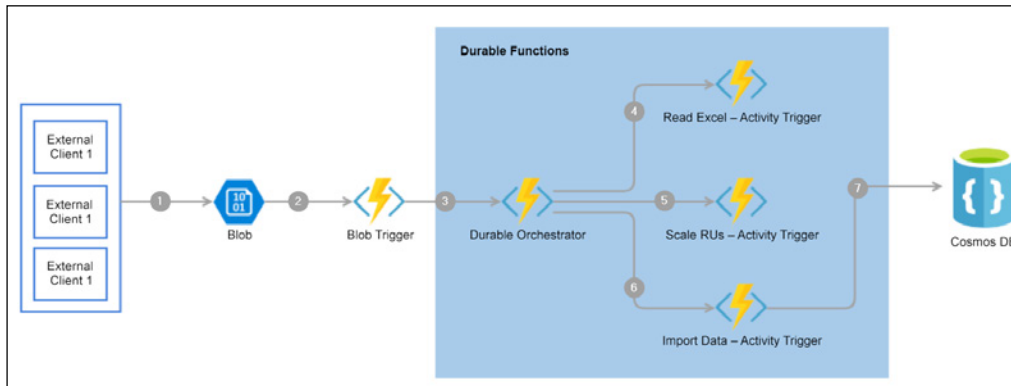
Reliable Serverless Applications Using Durable Functions

Chapter 7 Developing

Chapter 7

Developing Reliable Serverless Applications Using Durable Functions

Durable serverless way of implementing an Excel import



External clients or applications upload an Excel file to Blob Storage
Blob Trigger gets triggered after the Excel file is uploaded successfully

Blob Trigger

Read Excel - Activity Trigger

Scale RUs - Activity Trigger

Cosmos

DB

Import Data - Activity Trigger

Import Data - Activity Trigger
Cosmos DB

Uploading employee data into Blob Storage

Getting ready

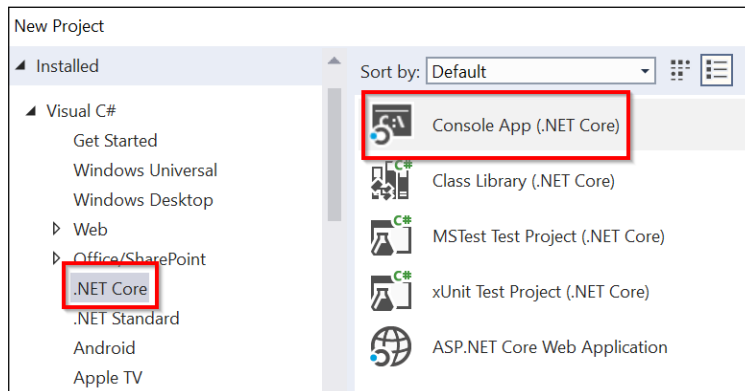
ExcelImports

Create an Excel file with some employee data as shown in the following

| Emp Id | Name | Email | PhoneNumber |
|--------|----------|--|-------------|
| 1 | Vijay | Vijay@vijay.com | 1111111111 |
| 2 | Vivek | Vivek@vivek.com | 2222222222 |
| 3 | Vineesha | vineesha@vineesha.com | 3333333333 |
| 4 | Praveen | Praveen@praveen.com | 4444444444 |
| 5 | Nishit | nishit@nishit.com | 5555555555 |
| 6 | Ragi | ragi@ragi.com | 6666666666 |
| 7 | Uma | uma@uma.com | 7777777777 |
| 8 | Harsha | harsha@harsha.com | 8888888888 |

How to do it...

ExcelImport.Client



```

Install-Package Microsoft.Azure.Storage.Blob
Install-Package Microsoft.Extensions.Configuration
Install-Package Microsoft.Extensions.Configuration.FileExtensions
Install-Package Microsoft.Extensions.Configuration.Json

```

Program.cs file:

```

using Microsoft.Extensions.Configuration;
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.Blob;
using System;
using System.IO;
using System.Threading.Tasks;

```

UploadBlob

uploads the Excel file into the blob container that we have created. For the sake of simplicity, the following code uploads the Excel file from a hardcoded location. However, in a typical real-time application, this file would be

Program.cs file of the ExcelImport.Client

```

private static async Task UploadBlob()
{
    var builder = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("appsettings.json", optional: true, reloadOnChange:
true);
    IConfigurationRoot configuration = builder.Build();
    CloudStorageAccount cloudStorageAccount = CloudStorageAccount.
Parse(configuration.GetConnectionString("StorageConnection"));
    CloudBlobClient cloudBlobClient = cloudStorageAccount.
CreateCloudBlobClient();
    CloudBlobContainer ExcelBlobContainer = cloudBlobClient.GetContain
erReference("Excelimports");
    await ExcelBlobContainer.CreateIfNotExistsAsync();
    CloudBlockBlob cloudBlockBlob = ExcelBlobContainer.GetBlockBlobRef
erence("EmployeeInformation" + Guid.NewGuid().ToString());
    await cloudBlockBlob.UploadFromFileAsync(@"C:\Users\
vmadmin\source\repos\POC\ImportExcelPOC\ImportExcelPOC\
EmployeeInformation.xlsx");
}

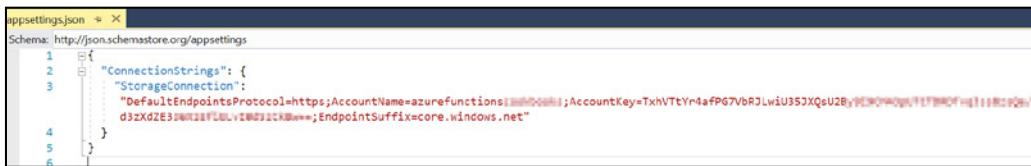
```

Main

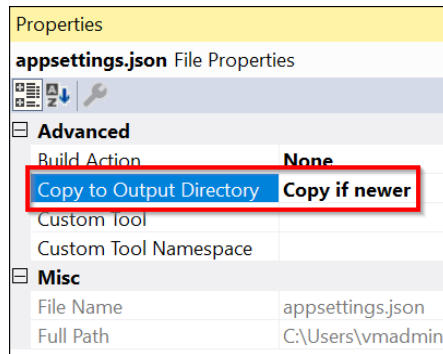
UploadBlob

```
{
UploadBlob().Wait();
}
catch (Exception ex)
{
Console.WriteLine("An Error has occurred with the message" +
ex.Message);
}
```

configuration file named appsettings.json



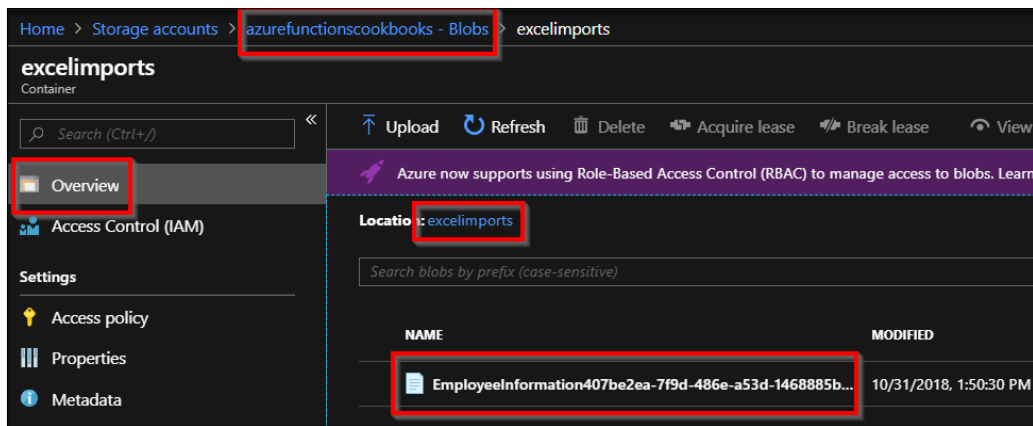
Output Directory appsettings.json file and change Copy to Copy if newer



Now, build the application and execute it. If you have configured everything,

```
Successfully Uploaded.  
Press any key to continue . . .
```


Excelimports, where you should see the Excel file that we have uploaded,



How it works...

to upload a blob (in our case, it is just an Excel file) to the designated blob container. Note that every time the application runs, a new file will get created in the blob container. In order to upload the Excel files with unique names, we are appending

There's more...

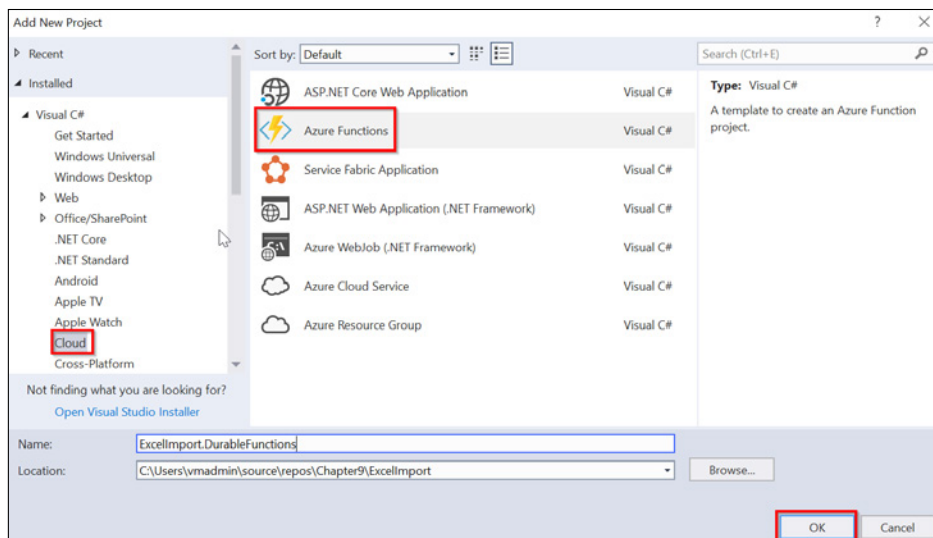
 This name may only contain lowercase letters, numbers and hyphens, and must begin with a letter or a number. Each hyphen must be preceded and followed by a non-hyphen character. The name must also be between 3 and 63 characters long.

Creating a Blob trigger

how the Blob trigger gets triggered when the Excel file is uploaded successfully

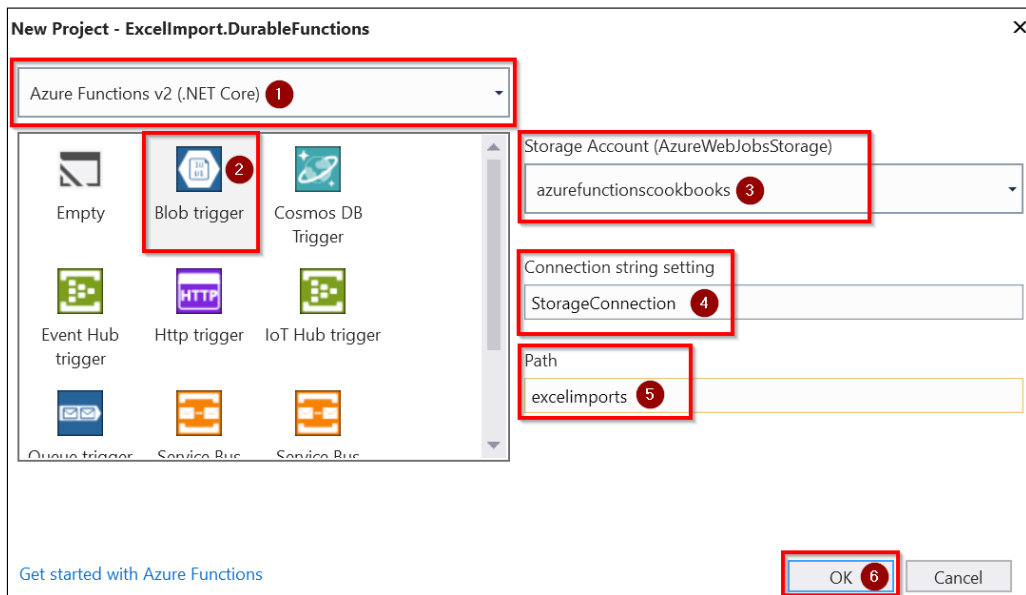
Getting ready

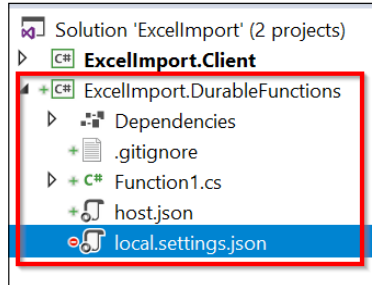
ExcelImport.DurableFunctions
Azure Functions



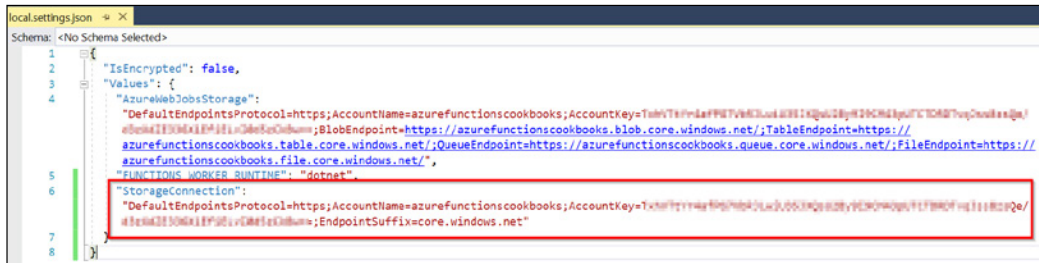
Azure Functions v2 (.NET Core) Blob trigger

- **Storage Account (AzureWebJobsStorage)**
- **Connection string setting**
- **Path**





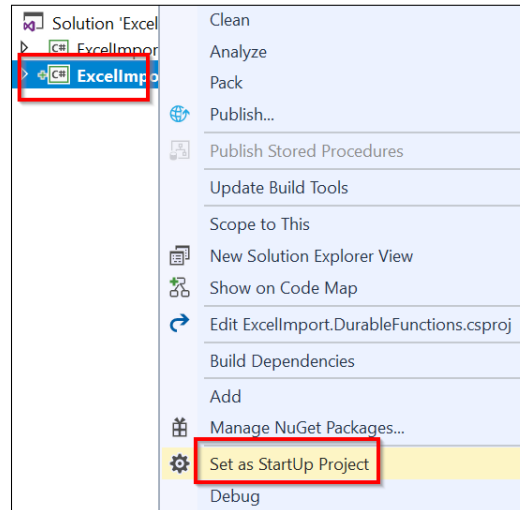
StorageConnection
we have used this in the connection string setting file in one of our earlier
local.settings.json



Function1.cs file and rename it to
ExcelImportBlobTrigger Function1
ExcelImportBlobTrigger



Configure ExcelImport.DurableFunctions



F5 `ExcelImportBlobTrigger`
everything is configured properly, you should see the

```

C:\Users\vmadmin\AppData\Local\AzureFunctionsTools\Releases\2.10.1\cli\func.exe
[10/31/2018 3:37:55 PM] Reading host configuration file 'C:\Users\vmadmin\source\repos\Chapter9\ExcelImport\ExcelImport.DurableFunctions\bin\Debug\netcoreapp2.1\host.json'
[10/31/2018 3:37:55 PM] Host configuration file read:
[10/31/2018 3:37:55 PM] {
[10/31/2018 3:37:55 PM]   "version": "2.0"
[10/31/2018 3:37:55 PM] }
[10/31/2018 3:37:56 PM] Initializing Host.
[10/31/2018 3:37:56 PM] Host initialization: ConsecutiveErrors=0, StartupCount=1
[10/31/2018 3:37:56 PM] Starting JobHost
[10/31/2018 3:37:57 PM] Starting Host (HostId=vm2017-1617097310, InstanceId=4f084e09-c991-42e1-8000-000000000000, ProcessId=15724, AppDomainId=1, Debug=False, FunctionsExtensionVersion=2.0.12134.0)
[10/31/2018 3:37:57 PM] Loading functions metadata
[10/31/2018 3:37:57 PM] 1 functions loaded
[10/31/2018 3:37:57 PM] Generating 1 job function(s)
[10/31/2018 3:37:57 PM] Found the following functions:
[10/31/2018 3:37:57 PM] ExcelImport.DurableFunctions.ExcelImportBlobTrigger.Run
[10/31/2018 3:37:57 PM]
[10/31/2018 3:37:57 PM] Host initialized (732ms)
[10/31/2018 3:38:01 PM] Host started (4664ms)
[10/31/2018 3:38:01 PM] Job host started
Hosting environment: Production
Content root path: C:\Users\vmadmin\source\repos\Chapter9\ExcelImport\ExcelImport.DurableFunctions
Now listening on: http://0.0.0.0:7071
Application started. Press Ctrl+C to shut down.
Listening on http://0.0.0.0:7071/
Hit CTRL-C to exit...
  
```

Let's now upload a new file by running the `ExcelImport.Client` application. Immediately after the file is uploaded, the Blob trigger will be fired, as shown in the following screenshot. Your breakpoints should

```
[10/31/2018 3:37:57 PM] Loading functions metadata
[10/31/2018 3:37:57 PM] 1 functions loaded
[10/31/2018 3:37:57 PM] Generating 1 job function(s)
[10/31/2018 3:37:57 PM] Found the following functions:
[10/31/2018 3:37:57 PM] ExcelImport.DurableFunctions.ExcelImportBlobTrigger.Run
[10/31/2018 3:37:57 PM]
[10/31/2018 3:37:57 PM] Host initialized (732ms)
[10/31/2018 3:38:01 PM] Host started (4664ms)
[10/31/2018 3:38:01 PM] Job host started
Hosting environment: Production
Content root path: C:\Users\vmadmin\source\repos\Chapter9\ExcelImport\ExcelImport.DurableFunctions\bin\Debug\netcoreapp2
.1
Now listening on: http://0.0.0.0:7071
Application started. Press Ctrl+C to shut down.
Listening on http://0.0.0.0:7071/
Hit CTRL-C to exit...
[10/31/2018 3:38:07 PM] Executing 'ExcelImportBlobTrigger' (Reason='New blob detected: excelimports/EmployeeInformation
77be2ea-7f9d-486e-a53d-1468885bc03b', Id=aa52d1dd-b538-4e8d-95c3-c2e873ab743c)
```

Blob trigger that gets fired whenever a new blob

How to do it...

a Blob trigger and configured it to run when a new Blob is added by configuring the `local.settings.json` configuration file to store the config values that are used in local development. After we created the `ExcelImport.Client` application to upload a file to validate

There's more...

All the configurations will be taken from the `local.settings.json` file while you the functions to Azure, all the configurations items (such as connection string and **Application Settings** Make sure that you create all the configuration items in the function app after you

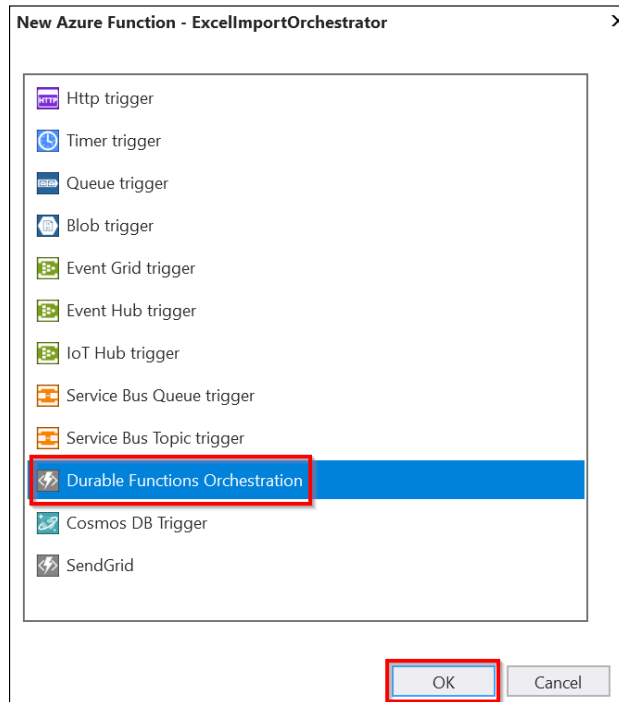
**New Azure Function
Orchestration**

**Durable Function
OK**

- HttpStart

ExcelImportBlobTrigger

- RunOrchestrator
- SayHello



ExcelImportBlobTrigger

-
-
- StartNewAsync DurableOrchestrationClient

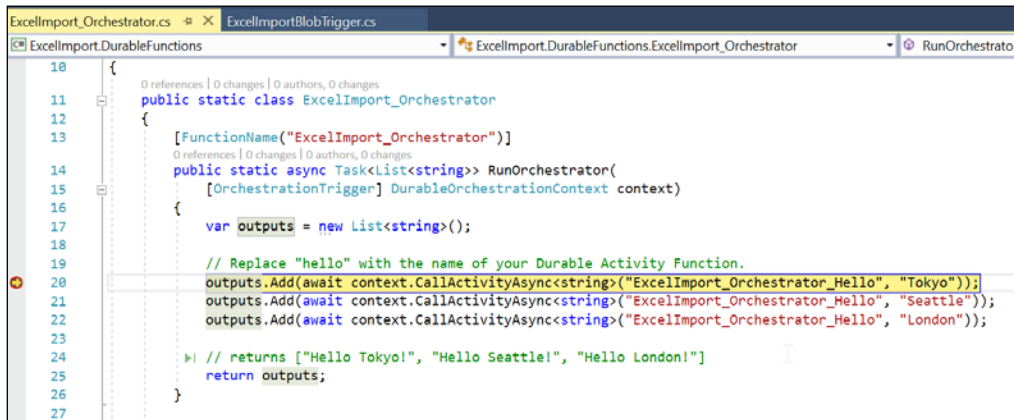
ExcelImportBlobTrigger

```
using System.IO;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Host;
using Microsoft.Extensions.Logging;
namespace ExcelImport.DurableFunctions
{
    public static class ExcelImportBlobTrigger
    {
        [FunctionName("ExcelImportBlobTrigger")]
        public static async void Run(
            [BlobTrigger("Excelimports/{name}", Connection =
            "StorageConnection")]Stream myBlob,
            string name,
            [OrchestrationClient]DurableOrchestrationClient starter,
            ILogger log)
        {
            string instanceId = await starter.StartNewAsync("ExcelImport_
            Orchestrator", name);
            log.LogInformation($"C# Blob trigger function Processed blob\n
            Name:{name} \n Size: {myBlob.Length} Bytes");
        }
    }
}
```

ExcelImport_Orchestrator

F5

Let's now upload a new file (while `ExcelImport.DurableFunctions`
`ExcelImport.Client`
also directly upload the Excel file from the Azure portal.) After the file
`ExcelImport_`
Orchestrator



```
10 {
11     0 references | 0 changes | 0 authors, 0 changes
12     public static class ExcelImport_Orchestrator
13     {
14         [FunctionName("ExcelImport_Orchestrator")]
15         0 references | 0 changes | 0 authors, 0 changes
16         public static async Task<List<string>> RunOrchestrator(
17             [OrchestrationTrigger] DurableOrchestrationContext context)
18         {
19             var outputs = new List<string>();
20             // Replace "hello" with the name of your Durable Activity Function.
21             outputs.Add(await context.CallActivityAsync<string>("ExcelImport_Orchestrator_Hello", "Tokyo"));
22             outputs.Add(await context.CallActivityAsync<string>("ExcelImport_Orchestrator_Hello", "Seattle"));
23             outputs.Add(await context.CallActivityAsync<string>("ExcelImport_Orchestrator_Hello", "London"));
24             // returns ["Hello Tokyo!", "Hello Seattle!", "Hello London!"]
25             return outputs;
26         }
27     }
```

How it works...

`ExcelImportBlobTrigger`
`OrchestratorClient`


There's more...

`DurableOrchestrationClient`

```
StartNewAsync
TerminateAsync
```

```
GetStatusAsync
```

```
RaiseEventAsync
```

```
[  https://docs.microsoft.com/en-us/azure/azure-functions/durable-functions-instance-management#sending-events-to-instances ]
```

Reading Excel data using activity functions

retrieve all data from specific Excel sheets by

Getting ready

```
ReadExcel_AT
```

```
ReadBlob
```

```
StorageManager
```

EPPlus

```
https://github.com/JanKallman/EPPlus
```

Returns the data from the Excel file as a collection of employee objects.


```
ExcelImport.DurableFunctions
```

```
Install-Package WindowsAzure.Storage
```

```
Install-Package EPPlus
```

How to do it...

If you think of Durable Functions as a workflow, then the activity trigger function can be treated a workflow step that takes some kind of optional input, performs

[ <https://docs.microsoft.com/en-us/azure/azure-functions/durable-functions-types-features-overview>]

Before we start creating the activity trigger function, let's first build the dependency

Read data from Blob Storage

StorageManager

This code connects to the specified storage account, reads the data from Stream

```
class StorageManager
{
    public async Task<Stream> ReadBlob(string BlobName)
    {
        var builder = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("local.settings.json", optional: true,
                reloadOnChange: true);
        IConfigurationRoot configuration = builder.Build();
        CloudStorageAccount cloudStorageAccount = CloudStorageAccount.
            Parse(configuration.GetConnectionString("StorageConnection"));
        CloudBlobClient cloudBlobClient = cloudStorageAccount.
            CreateCloudBlobClient();
        CloudBlobContainer ExcelBlobContainer = cloudBlobClient.
            GetContainerReference("Excel");
        CloudBlockBlob cloudBlockBlob = ExcelBlobContainer.GetBlockBlobRe
            ference(BlobName);
        return await cloudBlockBlob.OpenReadAsync();
    }
}
```

StorageManager

```

using Microsoft.Extensions.Configuration;
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.Blob;
using System.IO;
using System.Threading.Tasks;

```

local.settings.json file, as shown here:

```

1  {
2    "IsEncrypted": false,
3    "Values": {
4      "AzureWebJobsStorage":
5        "DefaultEndpointsProtocol=https;AccountName=azurefunctioncookbooks;AccountKey=TxhVtYr4afPG7VbR3LwU35JXQsU2B9I9CH43pUTC9RD7Vq3ss8zsQe/
6        d3zXdZE336X1EfiELvI0d5zCk0w==;BlobEndpoint=https://azurefunctioncookbooks.blob.core.windows.net;/TableEndpoint=https://
7        azurefunctioncookbooks.table.core.windows.net;/QueueEndpoint=https://azurefunctioncookbooks.queue.core.windows.net;/FileEndpoint=https://
8        azurefunctioncookbooks.file.core.windows.net/",
9      "FUNCTIONS_WORKER_RUNTIME": "dotnet",
10     "StorageConnection":
11       "DefaultEndpointsProtocol=https;AccountName=azurefunctioncookbooks;AccountKey=TxhVtYr4afPG7VbR3LwU35JXQsU2B9I9CH43pUTC9RD7Vq3ss8zsQe/
12       d3zXdZE336X1EfiELvI0d5zCk0w==;EndpointSuffix=core.windows.net"
13     }
14   }
15 }
16 }
17 }
18 }
19 }
20 }

```

Read Excel data from the stream

EPPlusExcelManager

ReadExcelData

EPPlus to read the data from the Excel file (.xlsx)

Employee

Employee

```

class EPPlusExcelManager
{
    public List<Employee> ReadExcelData(Stream stream)
    {
        List<Employee> employees = new List<Employee>();
        //FileInfo existingFile = new FileInfo("EmployeeInformation.
        xlsx");
        using (ExcelPackage package = new ExcelPackage(stream))
        {
            ExcelWorksheet ExcelWorksheet = package.Workbook.Worksheets[0];
            for (int EmployeeIndex = 2; EmployeeIndex < ExcelWorksheet.
            Dimension.Rows + 1; EmployeeIndex++)
            {

```

```
employees.Add(new Employee()
{
    EmpId = Convert.ToString(ExcelWorksheet.Cells[EmployeeIndex,
1].Value),
    Name = Convert.ToString(ExcelWorksheet.Cells[EmployeeIndex,
2].Value),
    Email = Convert.ToString(ExcelWorksheet.Cells[EmployeeIndex,
3].Value),
    PhoneNumber = Convert.ToString(ExcelWorksheet.
Cells[EmployeeIndex, 4].Value)
});
}
}
return employees;
}
}
```

Employee

```
public class Employee
{
    public string EmpId { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    public string PhoneNumber { get; set; }
}
```

developing the dependencies for our first activity trigger function. Let's now start

Create the activity function

```
StorageManager
ReadExcel_AT
EPPlusExcelManager
ExcelImport_Orchestrator
[FunctionName("ReadExcel_AT")]
public static async Task<List<Employee>> ReadExcel_AT(
[ActivityTrigger] string name,
ILogger log)
{
```

```
    log.LogInformation("Orchestration started");
    StorageManager storageManager = new StorageManager();
    Stream stream = null; ;
    log.LogInformation("Reading the Blob Started");
    stream = await storageManager.ReadBlob(name);
    log.LogInformation("Reading the Blob has Completed");
    EPPlusExcelManager ePPlusExcelManager = new EPPlusExcelManager();
    log.LogInformation("Reading the Excel Data Started");
    List<Employee> employees = ePPlusExcelManager.
ReadExcelData(stream);
    log.LogInformation("Reading the Blob has Completed");
    return employees;
}
```

System.IO


ExcelImport_Orchestrator

reads the data from the Excel file:

```
[FunctionName("ExcelImport_Orchestrator")]
public static async Task<List<string>> RunOrchestrator(
    [OrchestrationTrigger] DurableOrchestrationContext context)
{
    var outputs = new List<string>();
    string ExcelFileName = context.GetInput<string>();
    List<Employee> employees = await context.CallActivityAsync<List<E
mployee>>("ReadExcel_AT", ExcelFileName);
    return outputs;
}
```

Auto-scaling Cosmos DB throughput

capacity of the Cosmos DB collection is not sufficient.

| | | | |
|---|---|-----------------|-----------|
| [|  | Units RU | Request] |
| | https://docs.microsoft.com/en-us/azure/cosmos-db/request-units | | |

Getting ready

<https://docs.microsoft.com/en-us/azure/cosmos-db/create-sql-api-dotnet>

Create a Cosmos database and a collection with fixed storage and set the
400

The screenshot shows the 'Add Collection' dialog box with the following details:

- Database id:** cookbookdb (selected from a dropdown)
- Collection Id:** EmployeeCollection
- Storage capacity:** Fixed (10 GB) (selected, highlighted with a red box)
- Throughput (400 - 10,000 RU/s):** 400 (input field, highlighted with a red box)
- Estimated spend (USD):** \$0.032 hourly / \$0.77 daily.
- Unique keys:** + Add unique key
- Buttons:** OK (highlighted with a red box)

Fixed (10 GB) Storage capacity

Unlimited

Install-Package Microsoft.Azure.WebJobs.Extensions.CosmosDB

How to do it...

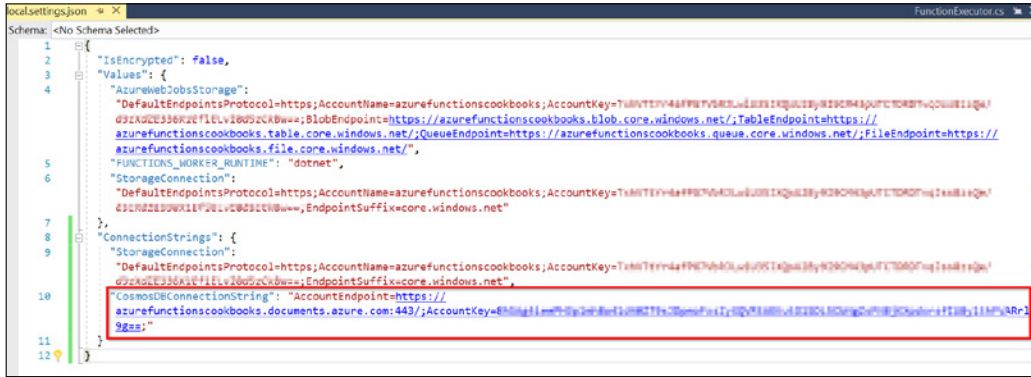
ScaleRU_AT ExcelImport_
Orchestrator.cs file. The function should look something like this

```
[FunctionName("ScaleRU_AT")]
public static async Task<string> ScaleRU_AT(
    [ActivityTrigger] int RequestUnits,
    [CosmosDB(ConnectionStringSetting = "CosmosDBConnectionString")]
    DocumentClient client
)
{
    DocumentCollection EmployeeCollection = await client.ReadDocument
tCollectionAsync(UriFactory.CreateDocumentCollectionUri("cookbook
db", "EmployeeCollection"));
    Offer offer = client.CreateOfferQuery().Where(o => o.ResourceLink
== EmployeeCollection.SelfLink).AsEnumerable().Single();
    Offer replaced = await client.ReplaceOfferAsync(new
OfferV2(offer, RequestUnits));
    return $"The RUs are scaled to 500 RUs!";
}
```

ExcelImport_Orchestrator.cs file:

```
using System.Linq;
using Microsoft.Azure.Documents;
using Microsoft.Azure.Documents.Client;
```

Keys

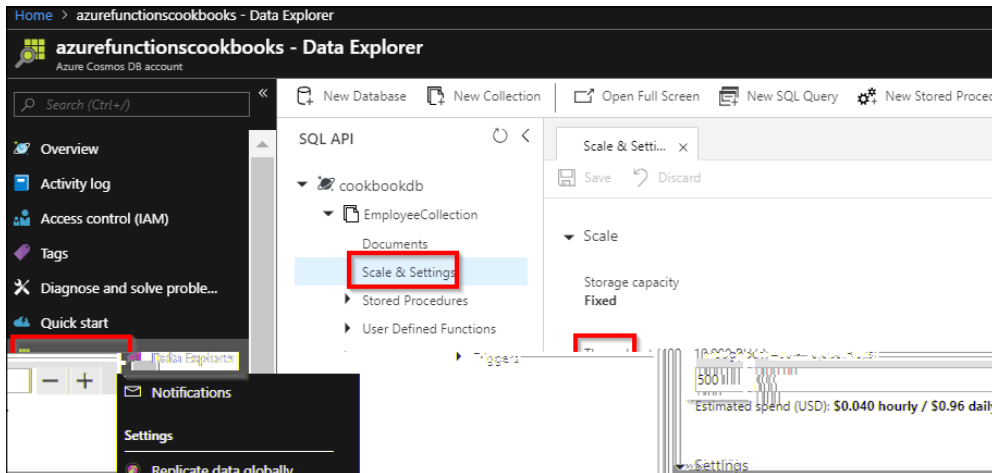


ExcelImport_Orchestrator
ScaleRU_AT 500

await context.CallActivityAsync<string>("ScaleRU_AT", 500);

Now, upload an Excel file to trigger the Orchestration, which internally
ScaleRU_AT

500
Data Explorer Scale & Settings



There's more...

offer

<https://docs.microsoft.com/en-us/rest/api/cosmos-db/offers>

Bulk inserting data into Cosmos DB

How to do it...

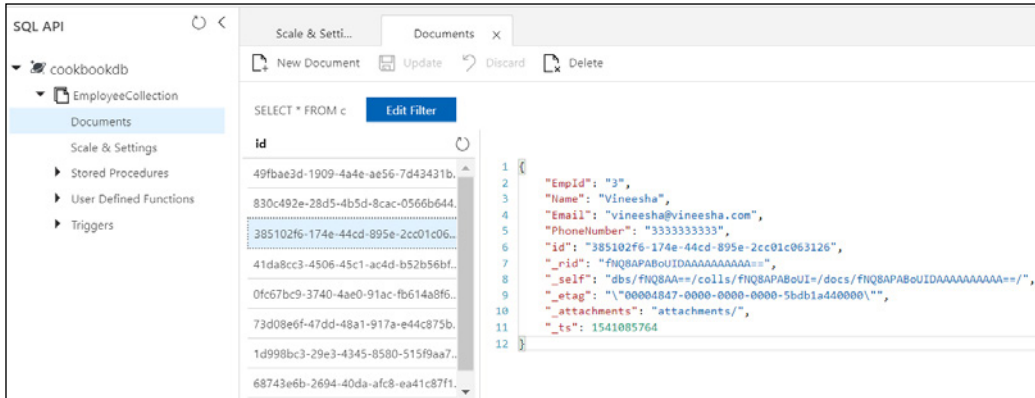
ImportData_AT

```
[FunctionName("ImportData_AT")]
public static async Task<string> ImportData_AT(
    [ActivityTrigger] List<Employee> employees,
    [CosmosDB(ConnectionStringSetting = "CosmosDBConnectionString")]
    DocumentClient client,
    ILogger log)
{
    foreach (Employee employee in employees)
    {
        await client.CreateDocumentAsync(UriFactory.CreateDocumentCollectionUri("cookbookdb", "EmployeeCollection"), employee);
        log.LogInformation($"Successfully inserted {employee.Name}.");
    }
    return $"Data has been imported to Cosmos DB Collection Successfully!";
}
```

ImportData_AT

```
await context.CallActivityAsync<string>("ImportData_AT",
    employees);
```

Let's run the application and upload an Excel file to test the functionality.



There's more...

<https://docs.microsoft.com/en-us/azure/cosmos-db/bulk-executor-overview>

to make it simple. You will have to configure them in the app settings file.

9

Implementing Best Practices for Azure Functions

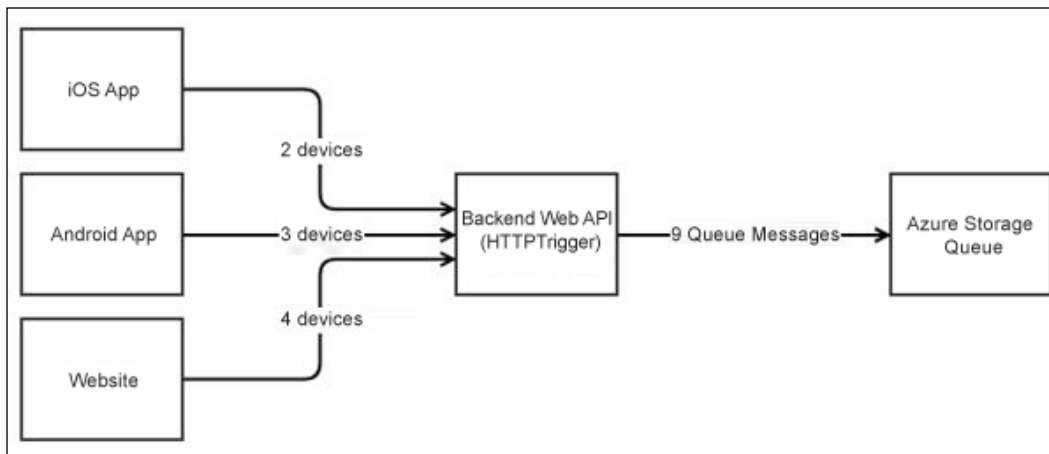
IAsyncCollector

Adding multiple messages to a queue using the IAsyncCollector function

In the first chapter, you learned

IAsyncCollector

Here is a diagram that depicts the data flow from different client applications



Backend Web API HTTPTrigger

Getting ready

<http://storageexplorer.com/>

How to do it...

BulkDeviceRegistrations

Save

```
#r "Newtonsoft.Json"
using System.Net;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Primitives;
using Newtonsoft.Json;
public static async Task<IActionResult> Run(HttpRequest req,
ILogger log )
{
log.LogInformation("C# HTTP trigger function processed a
request.");
string requestBody = await new StreamReader(req.Body).
ReadToEndAsync();
dynamic data = JsonConvert.DeserializeObject(requestBody);
string Device = string.Empty;
for(int nIndex=0;nIndex<data.devices.Count;nIndex++)
{
Device = Convert.ToString(data.devices[nIndex]);
log.LogInformation("devices data" + Device);
}
return (ActionResult)new OkObjectResult("Program has been executed
Successfully.");
}
```


Save

Save

```
public static async Task<IActionResult> Run(HttpRequest req,
ILogger log,
IAsyncCollector<string> DeviceQueue )
{
....
....
for(int nIndex=0;nIndex<data.devices.Count;nIndex++)
{
Device = Convert.ToString(data.devices[nIndex]);
DeviceQueue.AddAsync(Device); }
....
....
```

Test

```
{
  "devices":
  [
    {
      "type": "laptop",
      "brand": "lenovo",
      "model": "T440"
    },
    {
      "type": "mobile",
      "brand": "Mi",
      "model": "Red Mi 4"
    }
  ]
}
```

Run

| ID | Message Text |
|--------------------------------------|--|
| 2deabd46-73d0-4413-921e-251b5d03ef50 | { "type": "mobile", "brand": "Mi", "model": "Red Mi 4" } |
| 9fca2898-b6ad-4000-9410-701036f1a0e4 | { "type": "laptop", "brand": "lenovo", "model": "T440" } |

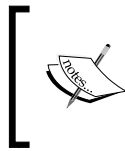
How it works...

```
IAsyncCollector<string>
```

There's more...

```
ICollector
```

```
IAsyncCollector
```



Implementing defensive applications using Azure Functions and queue triggers

Getting ready

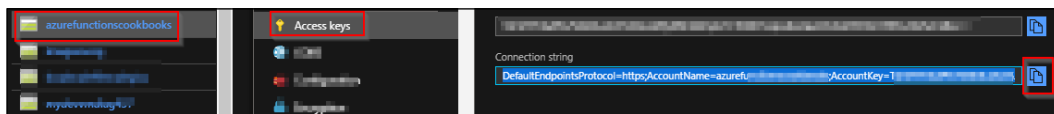
<http://storageexplorer.com/>

How to do it...

```
myqueuemessages
ProcessData
myqueuemessages
```

CreateQueueMessage – C# console application

```
StorageConnectionString
```



Install the Configuration and Queue Storage NuGet

```
Install-Package Microsoft.Azure.Storage.Queue
Install-Package System.Configuration.ConfigurationManager
```

```
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.Queue;
using System.Configuration;
```

CreateQueueMessages

```

static void CreateQueueMessages()
{
    CloudStorageAccount storageAccount =
        CloudStorageAccount.Parse(ConfigurationManager.
AppSettings
    ["StorageConnectionString"]);
    CloudQueueClient queueclient =
        storageAccount.CreateCloudQueueClient();

    CloudQueue queue = queueclient.GetQueueReference
        ("myqueuemessages");
    queue.CreateIfNotExists();

    CloudQueueMessage message = null;
    for(int nQueueMessageIndex = 0; nQueueMessageIndex <=
100;
        nQueueMessageIndex++)
    {
        message = new CloudQueueMessage(Convert.ToString
            (nQueueMessageIndex));
        queue.AddMessage(message);
        Console.WriteLine(nQueueMessageIndex);
    }
}

```

Developing the Azure Function – queue trigger

ProcessData

and configure the myqueuemessages

Azure Queue Storage trigger ✕ delete

Message parameter name ⓘ

Queue name ⓘ

Storage account connection ⓘ [show value](#)
 new

```
using System;
public static void Run(string myQueueItem,
    ILogger log)
{
    if (Convert.ToInt32(myQueueItem) > 50)
    {
        throw new Exception(myQueueItem);
    }
    else
    {
        log.LogInformation($"C# Queue trigger function
            processed: {myQueueItem}");
    }
}
```

a numerical index) for the first 50 messages and then throws an exception for the all

Running tests using the console application

Ctrl F5

myqueuemessages

Explorer display the first 32 messages. You need to use the C# Storage SDK

[ myqueuemessage]

(<OriginalQueueName>-Poison 50 myqueuemessages

| ID | Message T |
|--------------------------------------|-----------|
| edf51243-5857-4cf4-afd7-2f92b9be3f2d | 70 |
| 32946370-8667-401f-a01f-5d1c03b71472 | 52 |
| 04ada314-9e31-4bea-9e6c-0c8621cd743b | 53 |
| d7c46ef7-37aa-4172-ab2f-2c9672edd544 | 56 |
| e4aff6bb-cbb6-44f1-b22b-a3be302bb4f5 | 54 |
| 88c39a0f-37e1-46d8-bbaf-c704eb590bc1 | 55 |
| cecfe6ce-7964-470b-8a6e-3e6fcc8c8a41 | 57 |
| 51f89d3a-838a-42de-b691-133dc2ea04af | 58 |
| cf8b68d4-5a05-4643-93ee-4a0131358a6b | 60 |
| 58f3d7d4-f68b-4f6f-9243-0798d45dc75c | 59 |
| 75f941eb-9c76-4ff5-b921-610624af1275 | 61 |
| bc5e50a8-7547-4605-8dba-3322c83076d4 | 63 |
| 56632e73-f8a7-4d7f-a5a2-764016d475c8 | 62 |
| ee55a490-77ce-4632-9e62-d679080d94fb | 66 |
| e7bf0a18-5e08-4074-aa90-ca506dbb855b | 64 |
| | -- |

Showing 32 of 50 messages in queue

How it works...

<OriginalQueueName>-Poison

messages and take necessary actions to fix errors in the applications.

There's more...

tries to pick the message and process five times. You can learn about how this

```
dequecount          int          Run
```

Handling massive ingress using Event Hubs for IoT and other similar scenarios

Getting ready

Event Hubs
Event Hubs
Event Hubs
Event Hub
Consumer Group

Internet of Things
Overview
Event Hub
\$Default

How to do it...

Internet of Things IoT

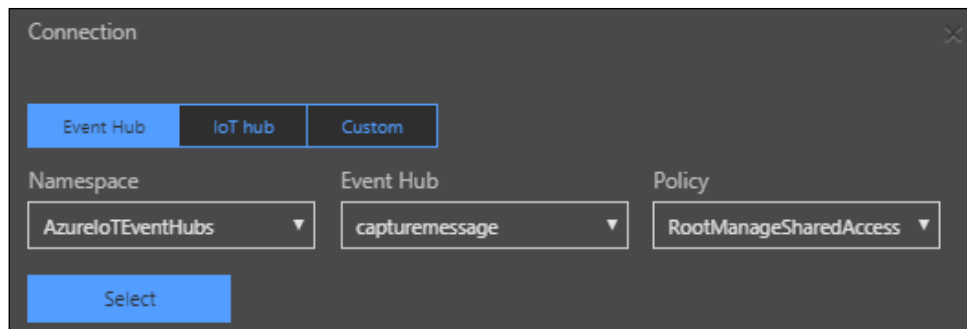
Creating an Azure Function event hub trigger

Azure Function Event Hub Trigger

`capturemessage`. If you don't have any connections configured yet, you

New
New
Event Hub
Select

Connection



Event Hub Connection

Create

Developing a console application that simulates IoT data

EventHubApp

NuGet

```
Install-Package Microsoft.Azure.EventHubs
Install-Package Newtonsoft.Json
```

System.Configuration.

dll

```
using Microsoft.Azure.EventHubs;
using System.Configuration;
```

App.config

App.config

ConnectionStrings

Overview

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<startup>
<supportedRuntime version="v4.0"
  sku=".NETFramework,Version=v4.6.1" />
</startup>
<appSettings>
<add key="EventHubConnection"
```



```
        value="Endpoint=sb://<event hub namespace
        here>.servicebus.windows.net/;Entitypath=<Event
Hubname>;
        SharedAccessKeyName= RootManageSharedAccessKey;
        SharedAccessKey=<Key here>"/>
</appSettings>
</configuration>
```

Create a new C# class file and place the following code in the new class file:

```
using System;
using System.Text;
using Microsoft.Azure.EventHubs;
using System.Configuration;
using System.Threading.Tasks;

namespace EventHubApp
{
    class EventHubHelper
    {
        static EventHubClient eventHubClient = null;
        public static async Task GenerateEventHubMessages()
        {
            EventHubsConnectionStringBuilder conBuilder = new
            EventHubsConnectionStringBuilder
            (ConfigurationManager.AppSettings
            ["EventHubConnection"].ToString());

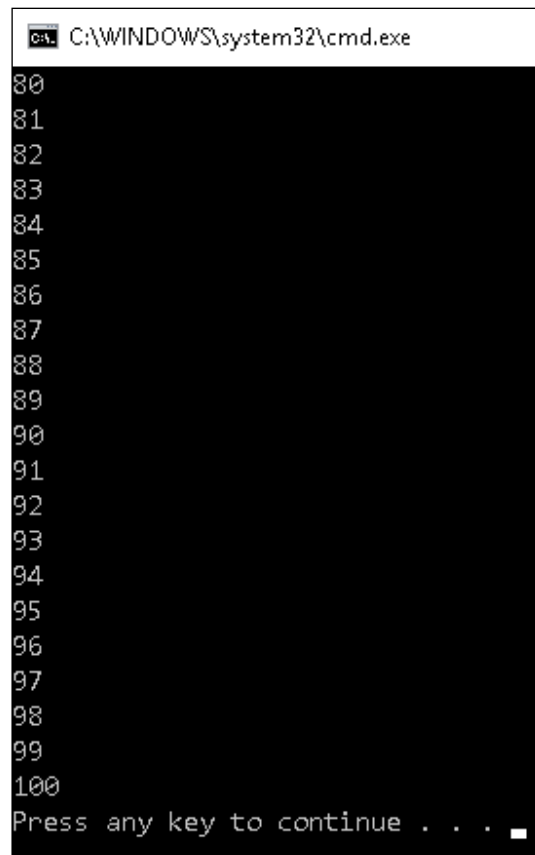
            eventHubClient =
            EventHubClient.CreateFromConnectionString
            (conBuilder.ToString());
            string strMessage = string.Empty;
            for (int nEventIndex = 0; nEventIndex <= 100;
            nEventIndex++)
            {
                strMessage = Convert.ToString(nEventIndex);
                await eventHubClient.SendAsync(new EventData
                (Encoding.UTF8.GetBytes(strMessage)));
                Console.WriteLine(strMessage);
            }
            await eventHubClient.CloseAsync();
        }
    }
}

main

namespace EventHubApp
{
```

```
class Program
{
    static void Main(string[] args)
    {
        EventHubHelper.GenerateEventHubMessages().Wait();
    }
}
```

Ctrl F5



```
cmd C:\WINDOWS\system32\cmd.exe
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
Press any key to continue . . .
```

Avoiding cold starts by warming the app at regular intervals

You will get all the benefits of serverless architecture only when you create the

<https://blogs.msdn.microsoft.com/appserviceteam/2018/02/07/understanding-serverless-cold-start/>



Getting ready

HttpALive
KeepFunctionAppWarm
HttpALive

there would be no concerns if your application had traffic regularly during the day. So, if we can ensure that the application has traffic all day, then the Azure Function

How to do it...

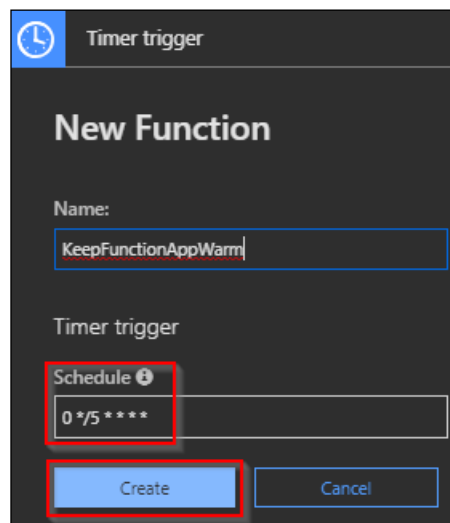
In this recipe, we will create a timer trigger that simulates traffic to the HTTP trigger,

Creating an HTTP trigger

```
using System.Net;
using Microsoft.AspNetCore.Mvc;
public static async Task<IActionResult> Run(HttpRequest req, ILogger
log)
{
    return (ActionResult)new OkObjectResult($"Hello User! Thanks for
keeping me Warm");
}
```

Creating a timer trigger

+
New Function Schedule
five



The screenshot shows the 'New Function' dialog in the Azure portal. The dialog is titled 'Timer trigger' and has a blue clock icon in the top left corner. The main title is 'New Function'. Below the title, there is a 'Name:' label and a text input field containing 'KeepFunctionAppWarm'. Underneath, there is a 'Timer trigger' section with a 'Schedule' label and a dropdown menu set to 'Schedule'. Below the dropdown is a text input field containing the cron expression '0 */5 * * * *'. At the bottom of the dialog, there are two buttons: 'Create' and 'Cancel'. Red boxes highlight the 'Schedule' dropdown, the cron expression input field, and the 'Create' button.

following code simulates traffic by making HTTP requests programmatically.

<<FunctionAppName>>

```
using System;
public async static void Run(TimerInfo myTimer, ILogger log)
{
    using (var httpClient = new HttpClient())
    {
        var response = await httpClient.GetAsync("https://<FunctionAppNa
me>>.azurewebsites.net/api/HttpALive");
    }
}
```

There's more...

Run-From-Package

See also

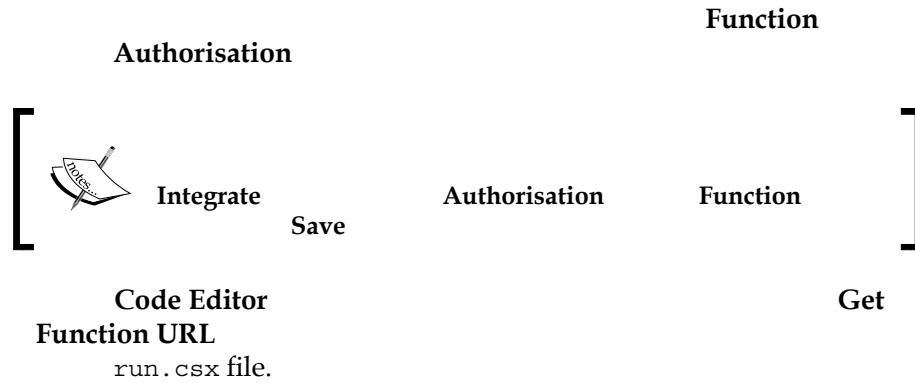
See the Deploying Azure Functions using Run From Package recipe of Configuring of Serverless Applications in the Production Environment

Enabling authorisation for function apps

Getting ready

<https://www.getpostman.com/>

How to do it...



- code

- name

401 Unauthorised

How it works...

first checks the presence of the header name code either in the query string collection
Request Body. If it finds it, then it validates the value of the code query

401

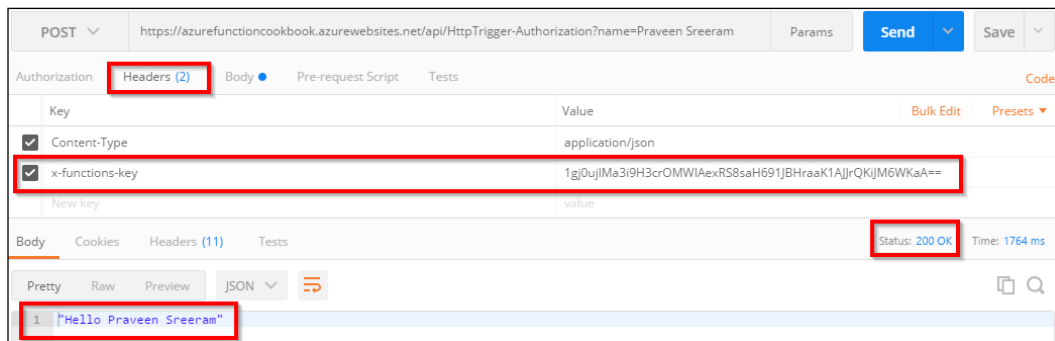
Unauthorized

There's more...

code

code

x-functions-key



Controlling access to Azure Functions using function keys

by setting the Anonymous Level field with the value
trigger

Integrate

fine-grained granular access to your Azure Function for your own applications or

How to do it...

Function keys

Host keys

Configuring the function key for each application

Manage

Add new function

WebApplication

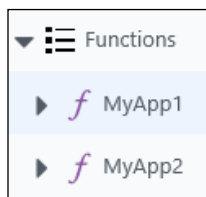
WebApplication

MobileApplication

MobileApplication

Configuring one host key for all the functions in a single function app

functions into a single function app and configure the authorisation for each function app instead of for each individual function. You can configure authorisation



Manage

MyApp1

- Functions +
- MyApp1
- Integrate
- Manage
- Monitor
- MyApp2
- Proxies (preview) +
- Slots (preview) +

Function Keys

| NAME | VALUE |
|---------|--------------------------------|
| default | Q5KfqD7wRS1zJBsN0ZvmePNjsQZ... |

Add new function key

Host Keys (All functions)

| NAME | VALUE |
|---------|------------------------------|
| _master | LMBAO482Zm61Ag0bXckG/KC... |
| default | aD6KDCDww9zWlYBVt7lJXMuak... |

Add new host key

MyApp2

- Functions +
- MyApp1
- MyApp2
- Integrate
- Manage
- Monitor
- Proxies (preview) +
- Slots (preview) +

Function Keys

| NAME | VALUE | ACTIONS |
|---------|-------|---|
| default | | Click to show Copy |

Add new function key

Host Keys (All functions)

| NAME | VALUE | ACTIONS |
|---------|------------------------------|---------|
| _master | LMBAO482Zm61Ag0bXckG/K0a... | Copy |
| default | aD6KDCDww9zWlYBVt7lJXMuak... | Copy |

Add new host key



There's more...

401

Unauthorized

| Key type | When should I use it? | Is it revocable (can it be deleted)? | Renewable? | Comments |
|----------|------------------------|--------------------------------------|------------|-------------|
| | Authorisation Admin | | | configured. |
| | Authorisation Function | | | |
| | Authorisation Function | | | |



Securing Azure Functions using Azure Active Directory

Active Directory AD

Functions provides an easy way to configure this called EasyAuth.

Getting ready

<https://docs.microsoft.com/azure/active-directory-b2c/active-directory-b2c-tutorials-web-app>

How to do it...

Configuring Azure AD to the function app

Platform features

Authentication/Authorisation

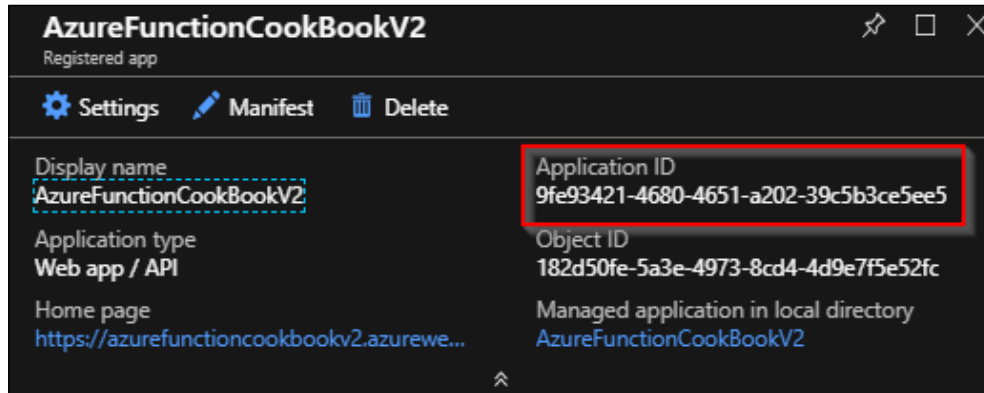
On

Login

Not Configured

Express

Management Mode field. Also, I opted **AzureFunctionCookbookV2** OK to save the configurations, which will take

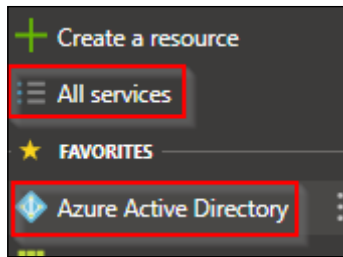


That's it. Without writing a single line of code, we are done with configuring

With the current configurations, none of the external client applications will

Registering the client app in Azure AD

All Services



**AD
application**

App Registrations

New

Fill in the fields as follows and click on the **Create**

`http://localhost`

Create

* Name ⓘ
PostmanAppRegistration ✓

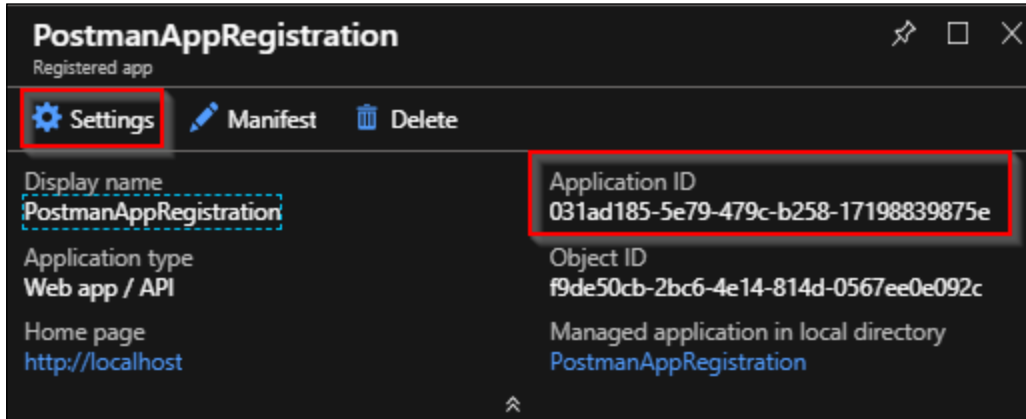
Application type ⓘ
Web app / API ▾

* Sign-on URL ⓘ
http://localhost ✓

Create

Application ID

Settings

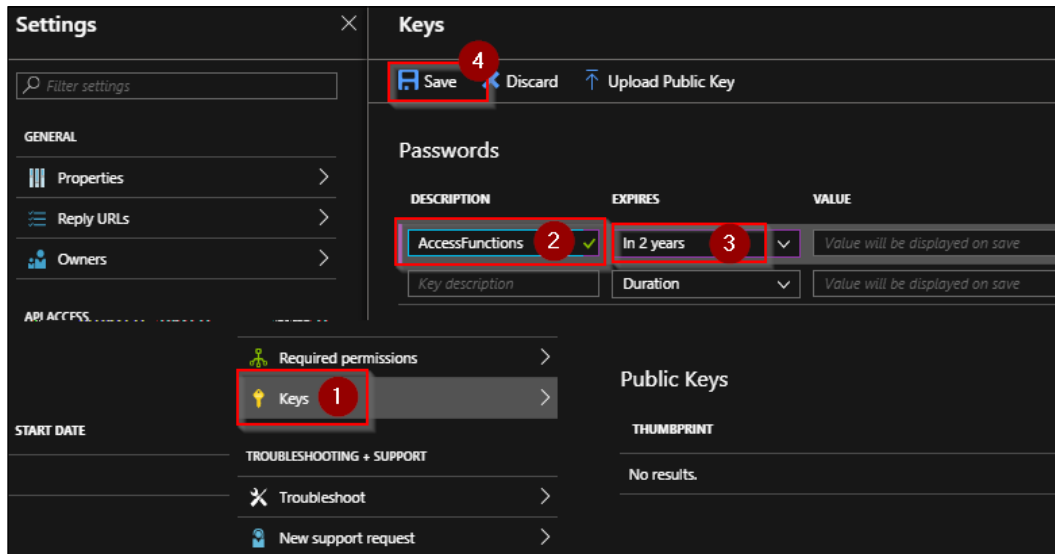


Settings

Keys

we will be passing from Postman. In order to generate the key, we first
Description **Duration**

Save button. The actual key is displayed to us in the value field only once
Save



Granting the client app access to the backend app

backend app. In this section, we will learn how to configure it:

Required Permissions
Add API Access

Select

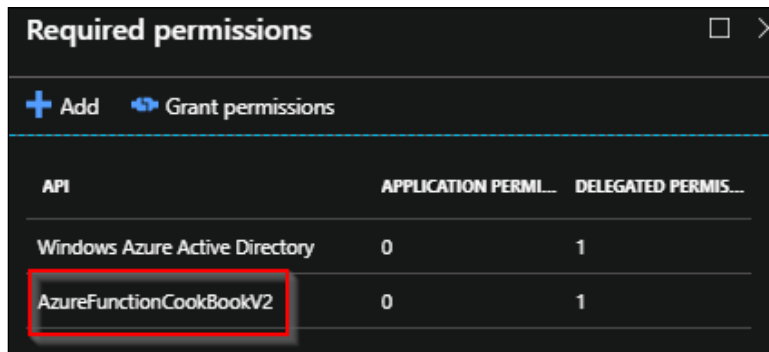
AzureFunctionCookBookV2
Select

Select

Permissions

<Backend App name> ,

Grant Permission



| API | APPLICATION PERMI... | DELEGATED PERMIS... |
|--------------------------------|----------------------|---------------------|
| Windows Azure Active Directory | 0 | 1 |
| AzureFunctionCookBookV2 | 0 | 1 |

Testing the authentication functionality using a JWT token

Endpoints

- Grant type `client_credentials`
- Client ID of the client application

Getting ready

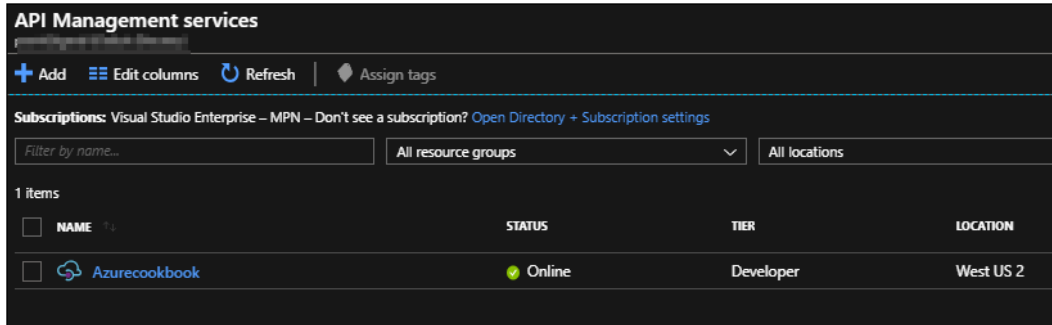
Standard/Premium Developer (No SLA) Developer Basic/
Create

The screenshot shows the 'API Management service' creation form in the Azure portal. The form is titled 'API Management service' and includes the following fields:

- Name:** Azurecookbook (with a green checkmark and a .azure-api.net domain suggestion)
- Subscription:** Visual Studio Enterprise – MPN
- Resource group:** AzureFunctionCookBooks (with a 'Create new' link)
- Location:** West US 2
- Organization name:** Azurecookbook (with a green checkmark)
- Administrator email:** [Redacted]
- Pricing tier:** Developer (No SLA) (with a 'View full pricing details' link)

At the bottom of the form, there is a blue 'Create' button and a link for 'Automation options'.

API Management



The screenshot shows the Azure API Management console interface. At the top, it says "API Management services". Below that, there are navigation options: "+ Add", "Edit columns", "Refresh", and "Assign tags". A "Subscriptions" section is visible, showing "Visual Studio Enterprise – MPN – Don't see a subscription? Open Directory + Subscription settings". There are filters for "Filter by name...", "All resource groups", and "All locations". Below the filters, it says "1 items". A table lists the API Management services with columns for NAME, STATUS, TIER, and LOCATION.

| | NAME | STATUS | TIER | LOCATION |
|--------------------------|---------------|--------|-----------|-----------|
| <input type="checkbox"/> | Azurecookbook | Online | Developer | West US 2 |

How to do it...

Integrating Azure Functions with API Management

API Management Instance

Create Function App

Browse

Azure Functions, where you can configure the function apps. Click on the **Configure Required Setting**

Select

API Management

Select

Import Azure Functions
API Management service

Don't see an Azure Function? Azure API Management requires Azure Functions to use the HTTP trigger and Function or Anonymous authorization level setting.

* Function App
AzureFunctionCookBookV2

Search to filter items...

| <input type="checkbox"/> | NAME | HTTP METHODS | URL TEMPLATE |
|-------------------------------------|------------------------------|--------------|------------------------------|
| <input type="checkbox"/> | BulkDeviceRegistrations | GET, POST | BulkDeviceRegistrations |
| | HttpTriggerCSharp1 | GET, POST | HttpTriggerCSharp1 |
| <input checked="" type="checkbox"/> | HttpTriggerTestUsingPost... | GET, POST | HttpTriggerTestUsingPost... |
| | MyApp1 | GET, POST | MyApp1 |
| | MyApp2 | GET, POST | MyApp2 |
| | RegisterUser | GET, POST | RegisterUser |
| | SaveJSONToSQLAzureData... | GET, POST | SaveJSONToSQLAzureData... |
| <input type="checkbox"/> | ValidateTwitterFollowerCo... | POST | ValidateTwitterFollowerCo... |

Select

pop-up **Create** **App**
Create

Create from Function App

Basic | Full

* Function App AzureFunctionCookBookV2 Browse

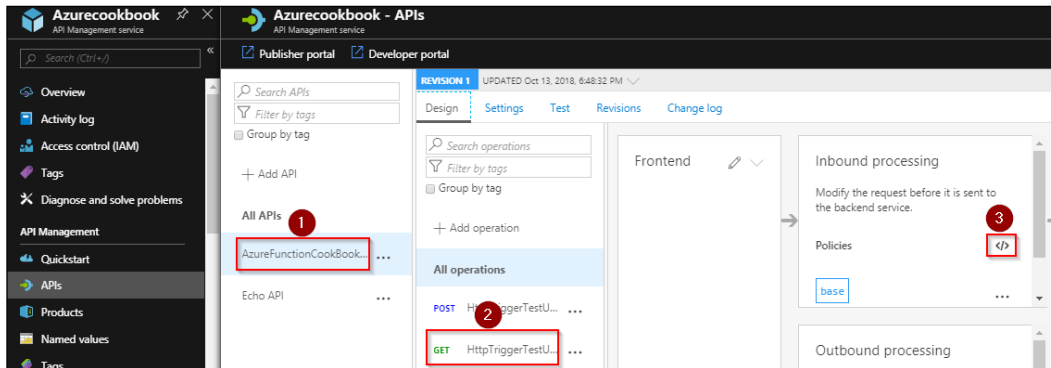
* Display name AzureFunctionCookBookV2

* Name azurefunctioncookbookv2

API URL suffix AzureFunctionCookBookV2

Create Cancel

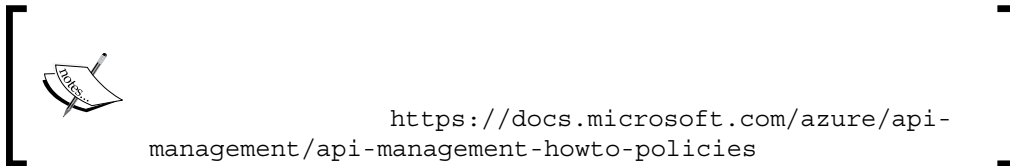
goes fine, you should get a screenshot as follows. Now we are **Azure Functions**



Configuring request throttling using inbound policies

GET

labelled 3



API Management
sending the request to the backend function app, we need to configure the

period attribute, and finally set the counter-key to the IP address of the

```
AzureFunctionCookBookV2 > HttpTriggerTestUsingPostman > Policies
1 <policies>
2   <inbound>
3     <base />
4     <rate-limit-by-key calls="1" renewal-period="60"
5       counter-key="@(<context.Request.IpAddress)" />
6   </inbound>
7   <backend>
8     <base />
9   </backend>
10  <outbound>
11    <base />
12  </outbound>
13  <on-error>
14    <base />
15  </on-error>
16 </policies>
```

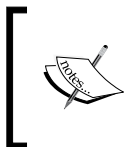
Save Discard



One final step

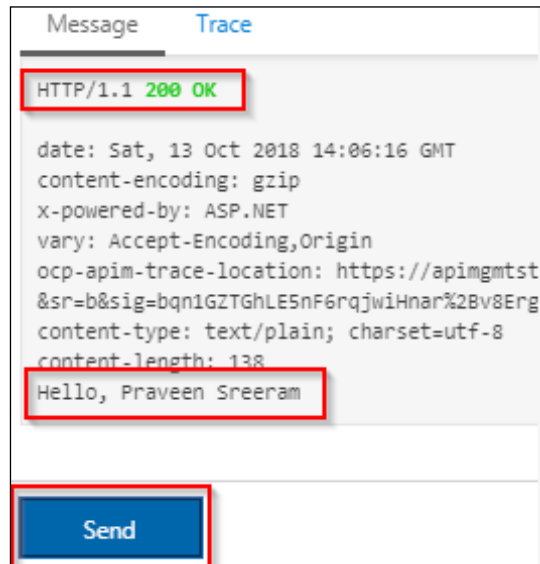
Save

The screenshot shows the 'Settings' tab in the Azure portal for an API. The interface includes a left-hand navigation pane with 'Search APIs', 'Filter by tags', and 'Group by tag' options. Below this, there is a list of APIs, including 'AzureFunctionCookBook...' and 'Echo API'. The main content area is titled 'REVISION 1' and 'UPDATED Oct 13, 2018, 7:28:33 PM'. It features tabs for 'Design', 'Settings', 'Test', 'Revisions', and 'Change log'. The 'Settings' tab is active, showing various configuration options: 'web service URL' (e.g. httpbin), 'URL scheme' (radio buttons for HTTP, HTTPs, Both), 'API URL suffix' (AzureFunctionCookBookV2), 'Base URL' (https://azurecookbook.azure-api.net/AzureFunctionCookBookV2), 'Tags' (e.g. Booking), and 'Products' (Starter). Below these are sections for 'Security' (User authorization: None, OAuth 2.0, OpenID connect) and 'Diagnostics Logs'. At the bottom, there are 'Save' and 'Discard' buttons.



Testing the rate limit inbound policy configuration

preceding step to make your first request. You should see something

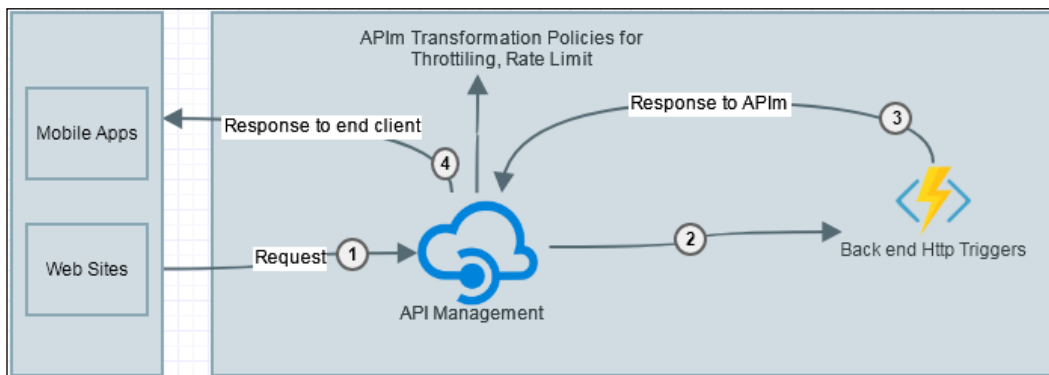


Send

```
Message Trace
HTTP/1.1 429 Too many requests
date: Sat, 13 Oct 2018 14:18:39 GMT
vary: Origin
ocp-apim-trace-location: https://apimgmtstdfzhl1n6s4bo5hx.blob.cor
&sr=b&sig=inKK2gAuB%2BXhhPP5131X%2B4IvHkQMYeKX8CHK56GR1Go%3D&se=20
content-type: application/json
content-length: 84
retry-after: 58
{
  "statusCode": 429,
  "message": "Rate limit is exceeded. Try again in 58 seconds."
}
```

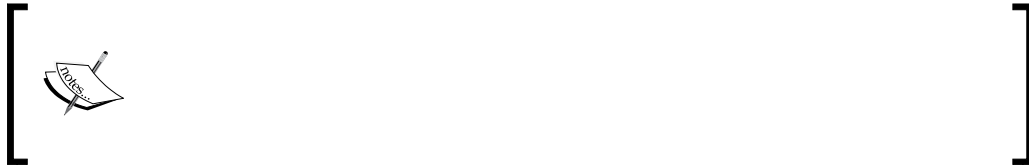
Send

How it works...



Securely accessing SQL Database from Azure Functions using Managed Service Identity

Seamless Integration of Azure Functions with Azure Services



Getting ready

How to do it...

Creating a function app using Visual Studio 2017 with V1 runtime

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

        string firstname = string.Empty, lastname = string.
Empty, email = string.Empty, devicelist = string.Empty;

        dynamic data = await req.Content.
ReadAsAsync<object>();
        firstname = data?.firstname;
        lastname = data?.lastname;
        email = data?.email;
        devicelist = data?.devicelist;

        SqlConnection con = null;
        try
        {
            string query = "INSERT INTO EmployeeInfo
(firstname,lastname, email, devicelist) " + "VALUES (@firstname,@
lastname, @email, @devicelist) ";

            con = new
                SqlConnection("Server=tcp:dbserver.database.
windows.net,1433;Initial Catalog=database;Persist Security Info=Fa
```

```
lse;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;");
        SqlCommand cmd = new SqlCommand(query, con);

        con.AccessToken = (new
AzureServiceTokenProvider()).GetAccessTokenAsync("https://
database.windows.net/").Result;

        cmd.Parameters.Add("@firstname", SqlDbType.
VarChar,
        50).Value = firstname;
        cmd.Parameters.Add("@lastname", SqlDbType.VarChar,
50)
        .Value = lastname;
        cmd.Parameters.Add("@email", SqlDbType.VarChar,
50)
        .Value = email;
        cmd.Parameters.Add("@devicelist", SqlDbType.
VarChar)
        .Value = devicelist;
        con.Open();
        cmd.ExecuteNonQuery();
    }
    catch (Exception ex)
    {
        throw ex;
    }
    finally
    {
        if (con != null)
        {
            con.Close();
        }
    }
    return req.CreateResponse(HttpStatusCode.OK, "Hello,
Successfully inserted the data");
}
```

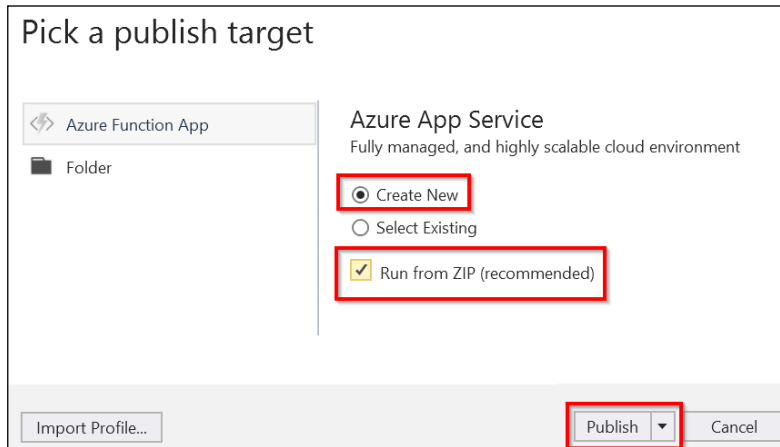
Seamless Integration of

Azure Functions with Azure Services

```
con.AccessToken = (new AzureServiceTokenProvider()).
GetAccessTokenAsync("https://database.windows.net/").Result;

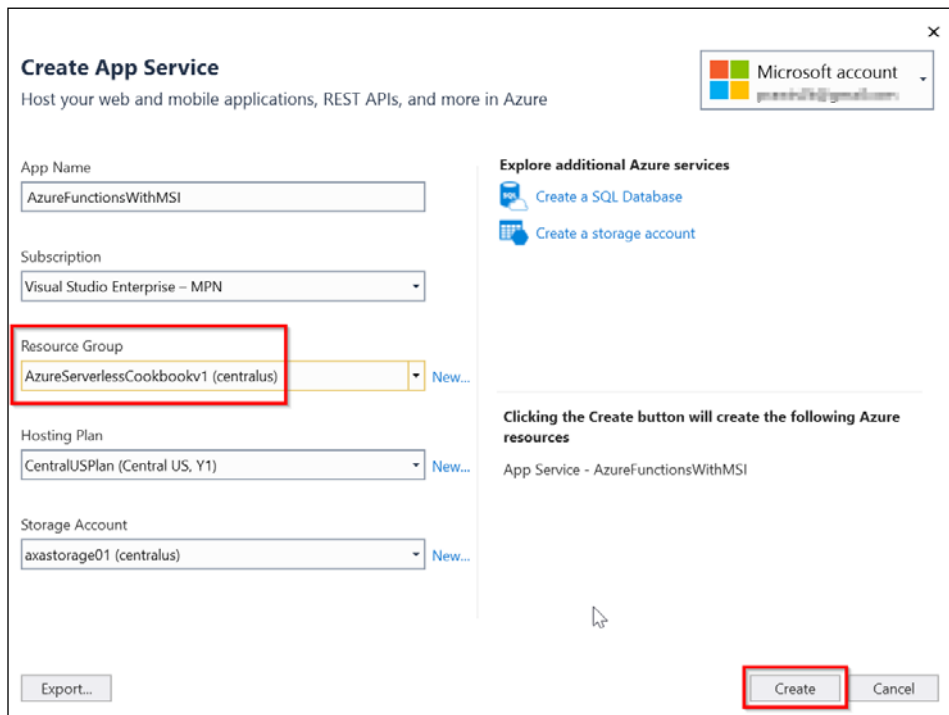
        NuGet
Install-Package Microsoft.IdentityModel.Clients.ActiveDirectory
Install-Package Microsoft.Azure.Services.AppAuthentication
```

Publish



Resource

Create

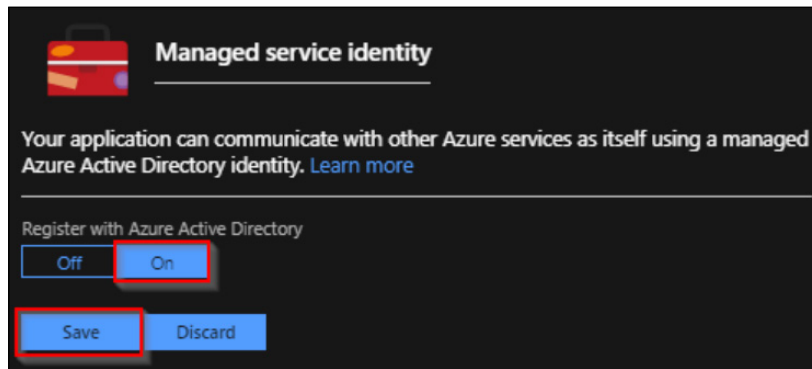


Creating a Logical SQL Server and a SQL Database

AzureServerlessCookbookv1

Enabling the managed service identity

Platform Managed
service identity
Managed service identity On Save



Retrieving Managed Service Identity information

```
az login
```

```
C:\Users\vmadmin>az login
Note, we have launched a browser for you to login. For old experience with device code, use "az login --use-device-code"
```

```
az resource show --name <<Function App Name>> --resource-group
<<Resource Group>> --resource-type Microsoft.Web/sites --query
identity
```

If you have configured the managed identity properly, you will see

```
C:\Users\vmadmin>az resource show --name AzureFunctions-CookBook --resource-group AzureFunctions-CookBook --resource-type
Microsoft.Web/sites --query identity
{
  "principalId": "6a1edb19-9084-4779-9482-7a39a0003469",
  "tenantId": "8ef7b61f-88aa-447e-bb2c-33a111111111",
  "type": "SystemAssigned",
  "userAssignedIdentities": null
}
```

principalId

Allowing SQL Server access to the new Managed Identity Service

principalId

```
az sql server ad-admin create --resource-group
AzureServerlessCookbookv1 --server-name azuresqlmsidbserver
--display-name sqladminuser --object-id <Principe Id>
```

EmployeeInfo

```
CREATE TABLE [dbo].[EmployeeInfo] (
  [PKEmployeeId] [bigint]
  IDENTITY(1,1) NOT NULL,
  [firstname] [varchar](50) NOT NULL,
  [lastname] [varchar](50) NULL,
  [email] [varchar](50) NOT NULL,
  [devicelist] [varchar](max) NULL,
  CONSTRAINT [PK_EmployeeInfo]
  PRIMARY KEY CLUSTERED (
    [PKEmployeeId] ASC
  ) )
```


Executing the HTTP trigger and testing

Postman

The screenshot shows the Postman interface for a POST request. The URL is `https://azurefunctionswithmsi.azurewebsites.net/api/HttpTriggerWithMSI`. The request body is set to 'raw' with the content type 'JSON (application/json)'. The body content is a JSON object:

```
{
  "firstname": "Praveen",
  "lastname": "Kumar",
  "email": "praveen@example.com",
  "devicelist": [
    {
      "Type": "Mobile Phone",
      "Company": "Microsoft"
    }
  ]
}
```

The response status is 200 OK, and the response body is `"Hello, Successfully inserted the data"`.

The screenshot shows a SQL query `select * from Employeeinfo` executed in a database tool. The results are displayed in a table:

| | PKEmployeeId | firstname | lastname | email | devicelist |
|---|--------------|-----------|----------|---------------------|--|
| 1 | 1 | Praveen | Kumar | praveen@example.com | [{ 'Type': 'Mobile Phone', 'Company': 'Microsoft' ... |

There's more...

```
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 System.Data.SqlClient;
using System.Data;
using System;
using Microsoft.Azure.Services.AppAuthentication;
```

See also

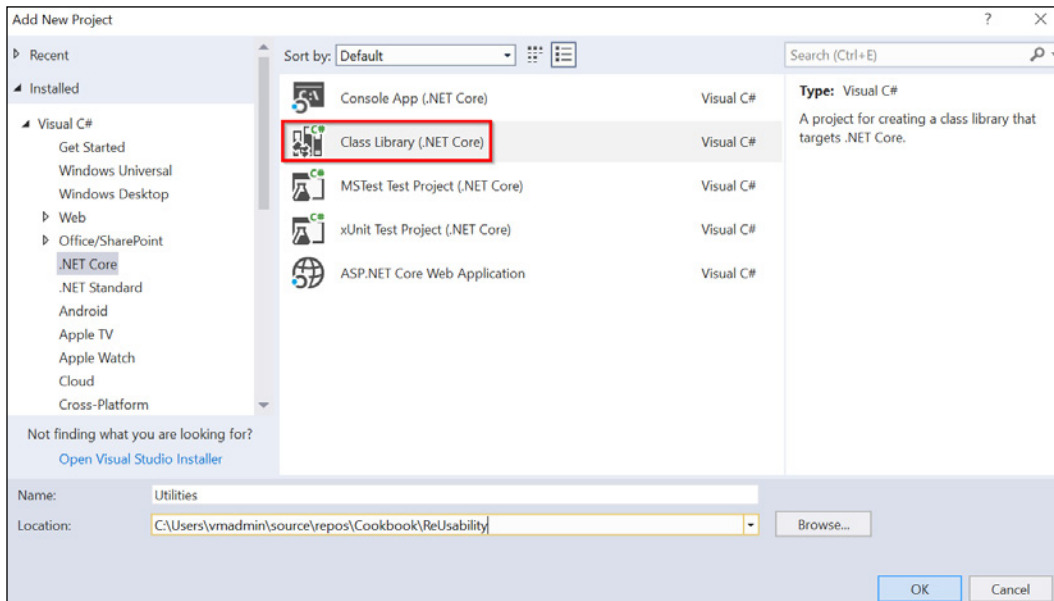
See the [Azure SQL Database interactions using Azure Functions recipe](#) of [Seamless Integration of Azure Functions with Azure Services](#)

Shared code across Azure Functions using class libraries

and create a new .dll file and you will learn how to use the classes and their methods

How to do it...

Class Library



Helper

class file:

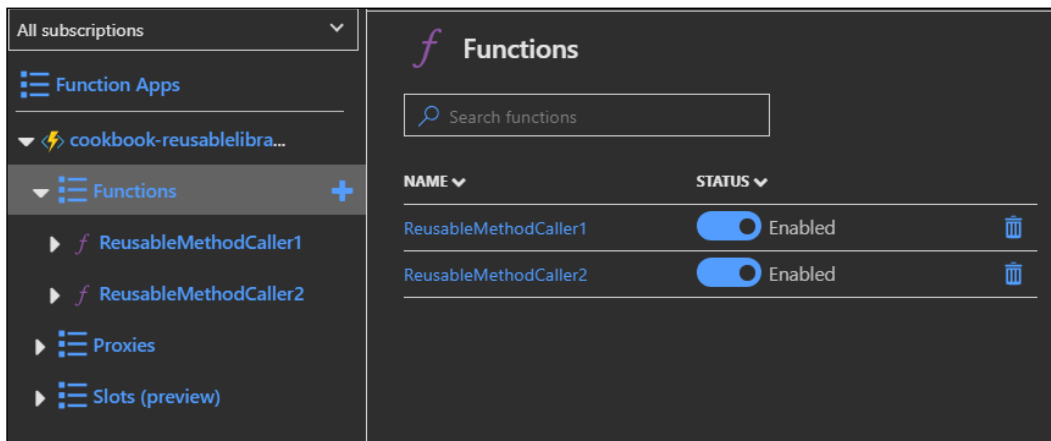
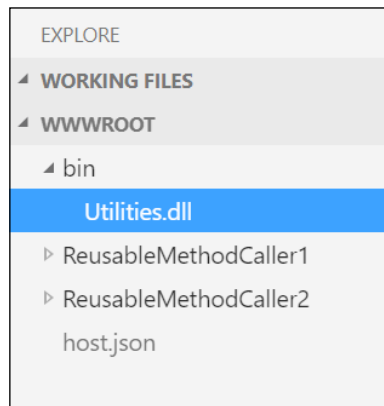
```
namespace Utilities
{
    public class Helper
    {
        public static string GetReusableFunctionOutput()
        {
            return "This is an output from a Resuable Library
across functions";
        }
    }
}
```

Change Build Configuration to Release and build the application to create .dll file, which will be used in our Azure Functions.

App Service Editor

files located in WWWROOT

Upload Files
.dll file that we created in Visual Studio
.dll file to the bin folder:



ReusableMethodCaller1

```
◦  
  
#r "../bin/Utilities.dll"  
  
◦  
  
using Utilities;  
  
GetReusableFunctionOutput  
  
log.LogInformation(Helper.GetReusableFunctionOutput());
```

```
2018-11-20T03:24:36.696 [Information] Compilation succeeded.  
2018-11-20T03:24:42.914 [Information] Executing 'Functions.ReusableMethodCaller1' (Reason='This function was  
programmatically called via the host APIs.', Id=556b0d73-07ed-4d34-bac3-4ad8252adbf9)  
2018-11-20T03:24:42.936 [Information] # HTTP trigger function processed a request.  
2018-11-20T03:24:42.936 [Information] This is an output from a Resuable Library across functions  
2018-11-20T03:24:42.937 [Information] Executed 'Functions.ReusableMethodCaller1' (Succeeded, Id=556b0d73-07ed-  
4d34-bac3-4ad8252adbf9)
```

ReusableMethodCaller2

```
2018-11-20T03:29:27 Welcome, you are now connected to log-streaming service.  
2018-11-20T03:29:53.251 [Information] Script for function 'ReusableMethodCaller2' changed. Reloading.  
2018-11-20T03:29:53.385 [Information] Compilation succeeded.  
2018-11-20T03:29:59.994 [Information] Executing 'Functions.ReusableMethodCaller2' (Reason='Timer fired at 2018-11-  
20T03:29:59.9938806+00:00', Id=357d4c51-1920-47f0-91d7-87aad0611955)  
2018-11-20T03:30:00.075 [Information] A timer trigger function executed successfully.  
2018-11-20T03:30:00.075 [Information] This is an output from a Resuable Library across functions  
2018-11-20T03:30:00.075 [Information] Executed 'Functions.ReusableMethodCaller2' (Succeeded, Id=357d4c51-1920-47f0-91d7-87aad0611955)
```

How it works...

.dll file that contains the reusable code that can be used in any .dll file.

.dll file was
.dll file to the bin folder.



There's more...

`bin` `.ddl` file in the required Azure Function folder.

Using strongly typed classes in Azure Functions

`RegisterUser`

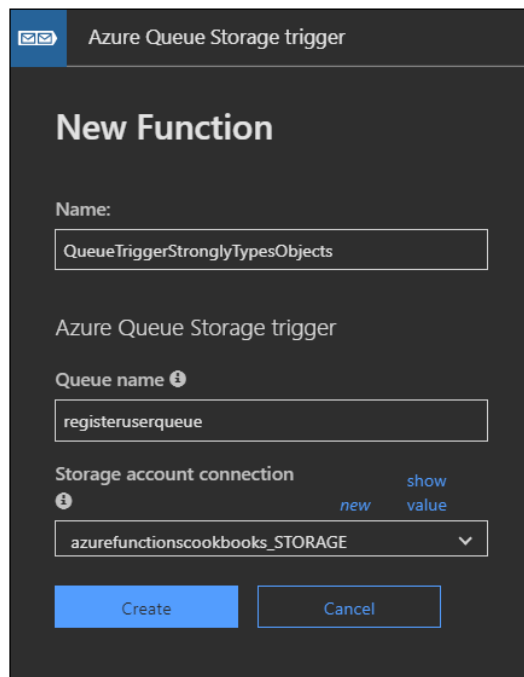
Getting ready

`azurefunctionscookbook`

Storage Explorer

How to do it...

```
registeruserqueue  
azurefunctionscookbook  
    registeruserqueue
```

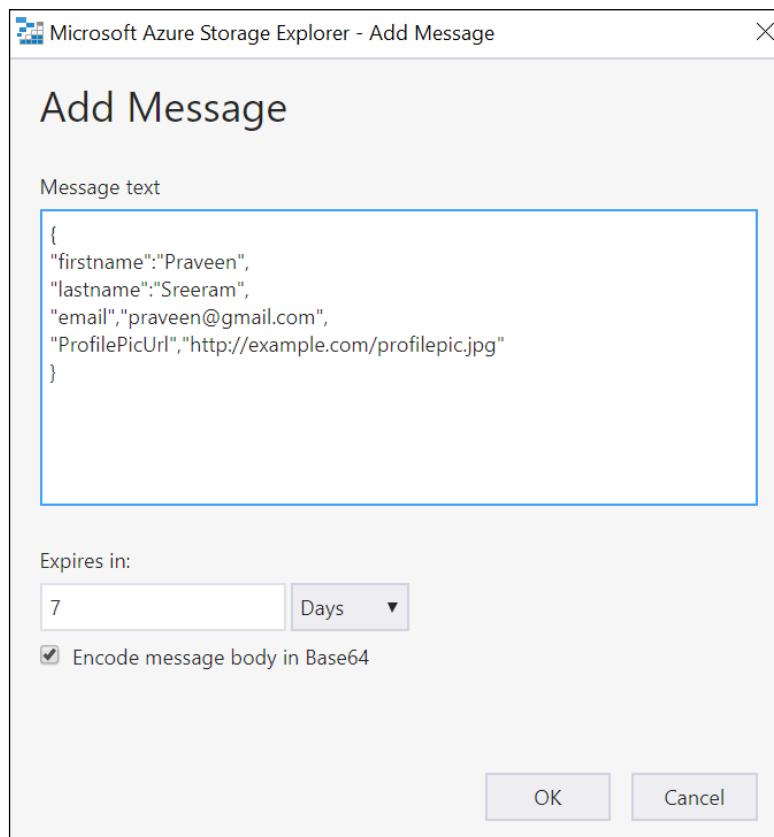


myQueueItem

```
using System;  
public static void Run(User myQueueItem, ILogger log)  
{
```

```
        log.LogInformation($"A Message has been created for a new
User");
        log.LogInformation($"First name: {myQueueItem.firstname}" );
        log.LogInformation($"Last name: {myQueueItem.lastname}" );
        log.LogInformation($"email: {myQueueItem.email}" );
        log.LogInformation($"Profile Url: {myQueueItem.ProfilePicUrl}"
);
    }
public class User
{
    public string firstname { get;set;}
    public string lastname { get;set;}
    public string email { get;set;}
    public string ProfilePicUrl { get;set;}
}
```

registeruserqueue



OK

```
2018-11-20T00:37:51.745 [Information] Executing 'Functions.QueueTriggerStronglyTypesObjects' (Reason='New queue message detected on 'registeruserqueue'.', Id=e043cc8e-dd49-42e7-8dd7-dd34db6021a6)
2018-11-20T00:37:51.747 [Information] A Message has been created for a new User
2018-11-20T00:37:51.747 [Information] First name: Praveen
2018-11-20T00:37:51.747 [Information] Last name: Sreeram
2018-11-20T00:37:51.747 [Information] email: praveen@gmail.com
2018-11-20T00:37:51.747 [Information] Profile Url: http://example.com/profilepic.jpg
2018-11-20T00:37:51.748 [Information] Executed 'Functions.QueueTriggerStronglyTypesObjects' (Succeeded, Id=e043cc8e-dd49-42e7-8dd7-dd34db6021a6)
```

How it works...

myQueueItem

There's more...

Run

User

10

Configuring of Serverless Applications in the Production Environment

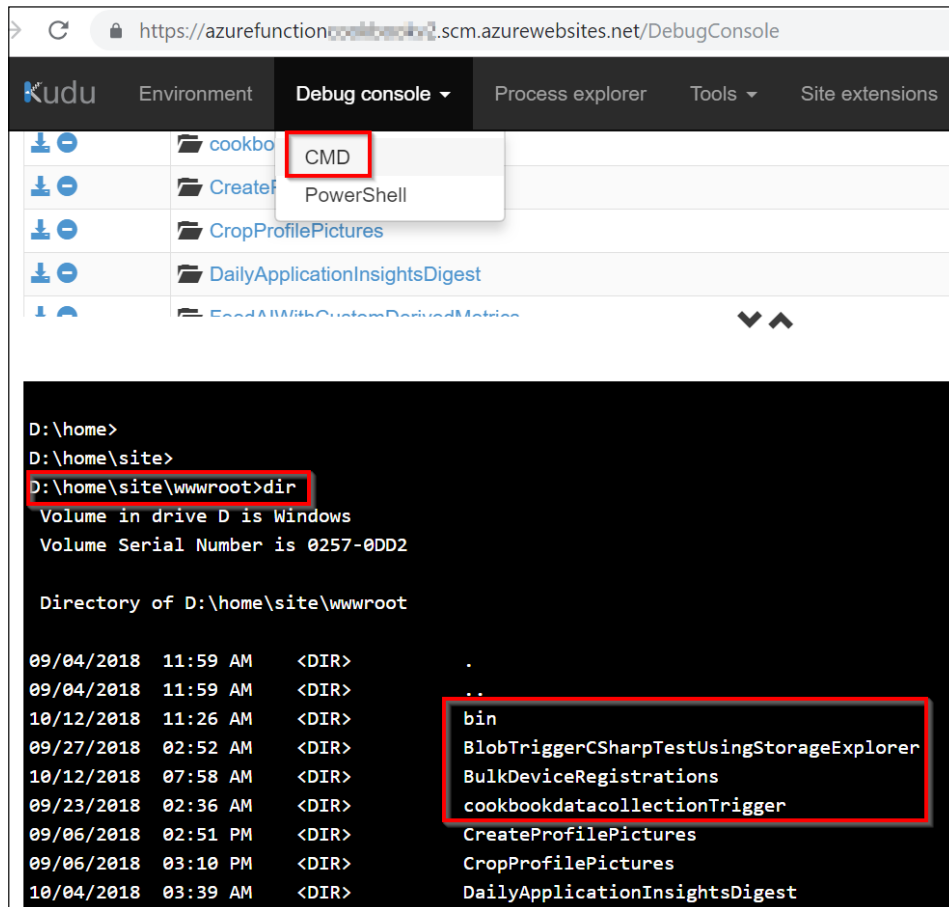
Introduction

configurations that one needs to make in a non-development environment

Deploying Azure Functions using the Run From Package

site\wwwroot

D:\home\



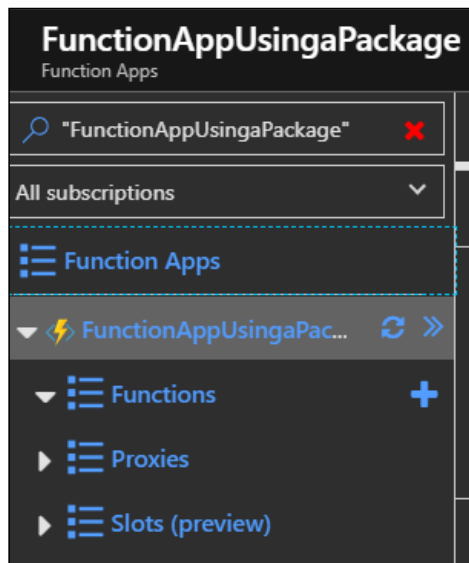
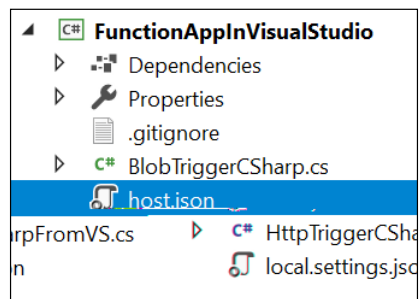
D:\home\site\wwwroot

binaries and all the configuration files that are required for executing the application.

Run From Zip

Run From Package

Getting ready

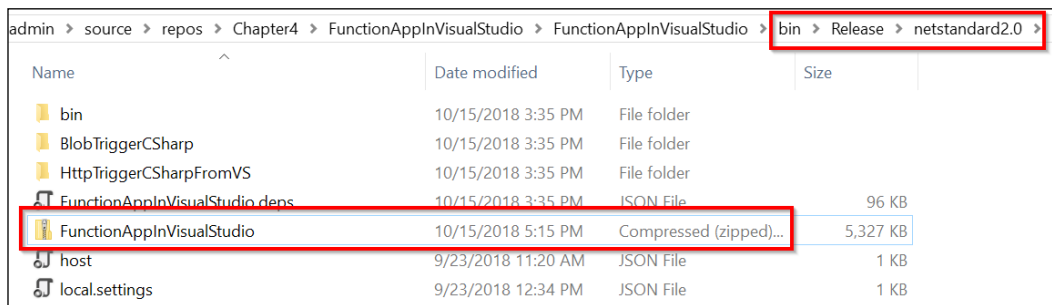


A storage account – we will upload the package file to the storage account.

How to do it...

Create a package file for the application. I'm using the same application that
Chapter 4 Understanding the Integrated Developer Experience of Visual Studio Tools

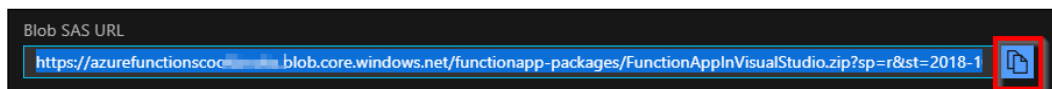
`bin`
files related to your functions. Create a `.zip` file out of the files, which



file either from the portal or by using Azure Storage Explorer.

shared access signature SAS

files located in the container. You can learn more about SAS
<https://docs.microsoft.com/azure/storage/common/storage-dotnet-shared-access-signature-part-1>

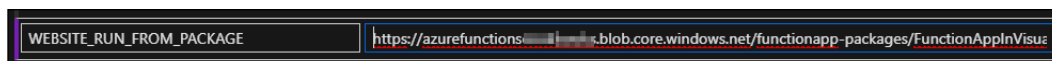


Application settings

WEBSITE_RUN_FROM_PACKAGE

Blob SAS URL

Save



That's it. After the preceding configuration, you can test the function:



How it works...

When the Azure Function runtime finds an app setting with the name `WEBSITE_RUN_FROM_PACKAGE` storage account. So, on the fly, the runtime downloads the files and uses them




There's more...

You can <https://github.com/Azure/app-service-announcements/issues/84>

Deploying Azure Function using ARM templates

Azure Resource Manager ARM

Getting ready

| | | | |
|--------------------------|---|------------------|------------|
| <input type="checkbox"/> |  azurefunctioncoba88 | Storage account | Central US |
| <input type="checkbox"/> |  azurefunctioncookbook-gateway | App Service | Central US |
| <input type="checkbox"/> |  CentralUSPlan | App Service plan | Central US |

azurefunctioncookbook-gateway

App Service Plan

App Service plan
Consumption plan
Storage account

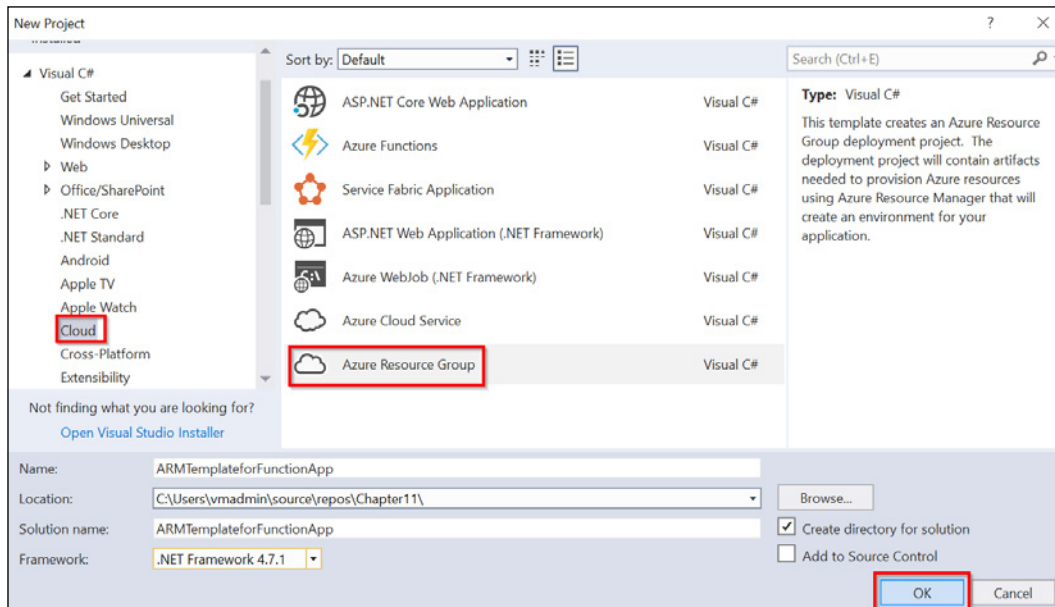
App Service plan

Application Insights

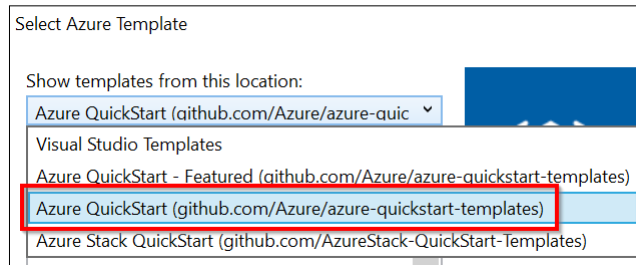
AzureWebJobsDashboard

How to do it...

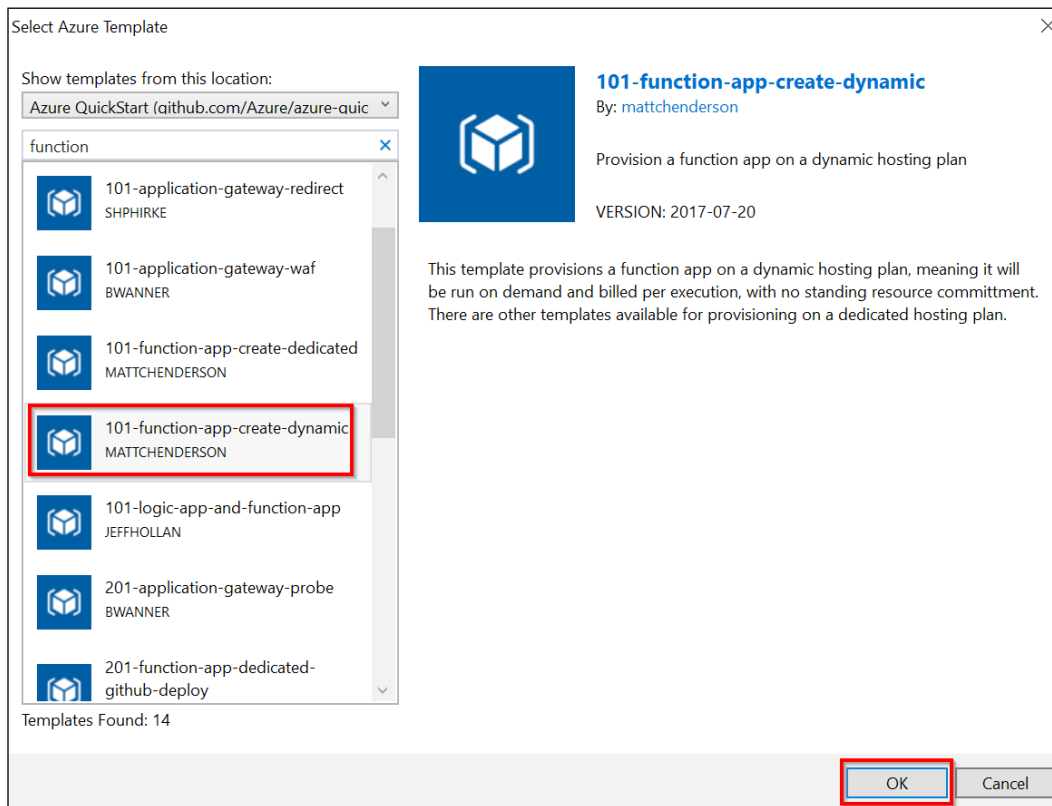
Azure Resource Group Visual C# Cloud



OK Select Azure
Template Azure Quick-Start (github.com/Azure/
azure-quickstart-templates)

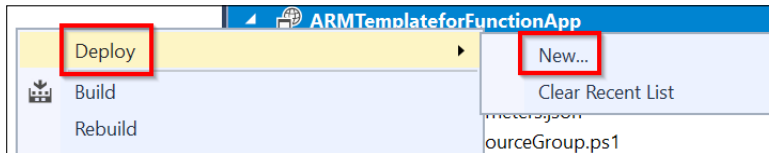


dynamic function 101-function-app-create-
plan Consumption

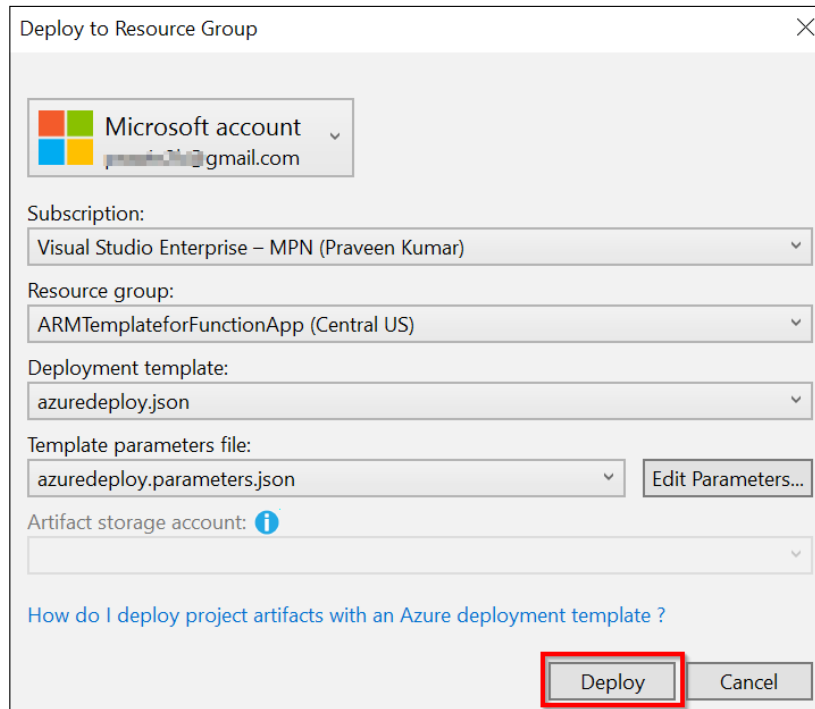


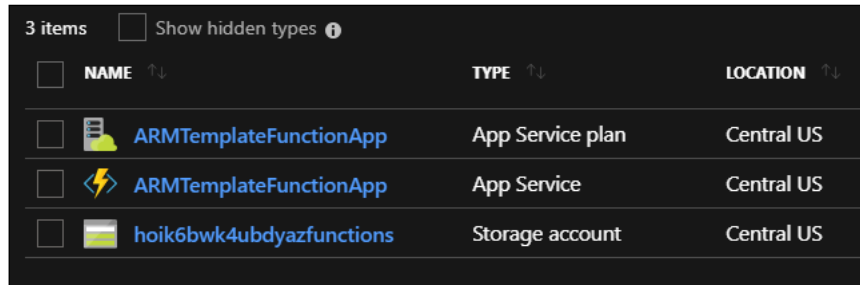
Visual Studio. You can learn <https://docs.microsoft.com/azure/azure-functions/functions-infrastructure-as-code>




resources. You can deploy it by right-clicking on the project name (in my case ARMTemplateforFunctionApp) and clicking **Deploy**.



Subscription, Resource group provisioning the function app. Choose all the mandatory fields and click **Deploy**.





| <input type="checkbox"/> | NAME ↑↓ | TYPE ↑↓ | LOCATION ↑↓ |
|--------------------------|--|------------------|-------------|
| <input type="checkbox"/> |  ARMTemplateFunctionApp | App Service plan | Central US |
| <input type="checkbox"/> |  ARMTemplateFunctionApp | App Service | Central US |
| <input type="checkbox"/> |  hoik6bwk4ubdyazfunctions | Storage account | Central US |

There's more...

push the files into some kind of version-control system, such as Git or TFS,

Configuring custom domain to Azure Functions

`functionappname.azurewebsites.net`

their own domains. Yes, it's possible to configure a custom domain for the function apps. In this recipe, we will learn how to configure it.

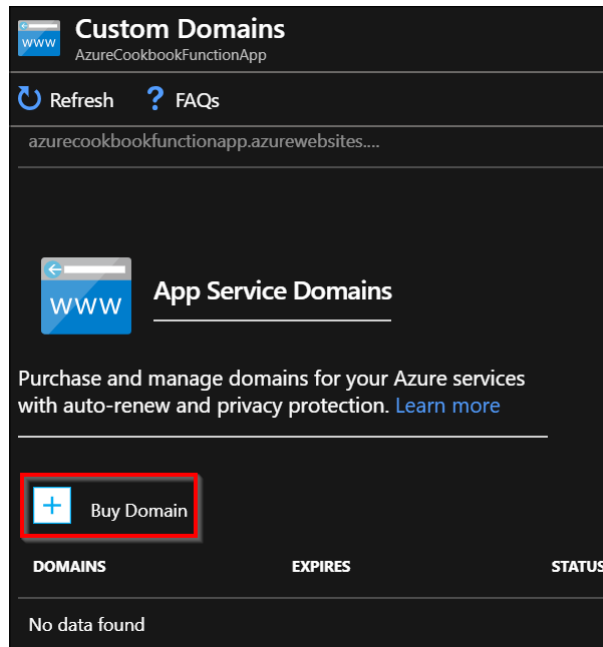
Getting ready

Create a domain with any of the domain registrars. You can also purchase a domain

Buy Domain

Custom

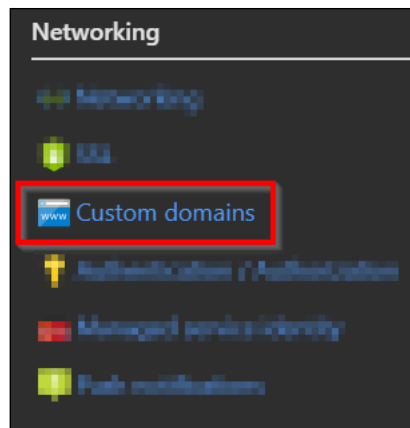
Domains



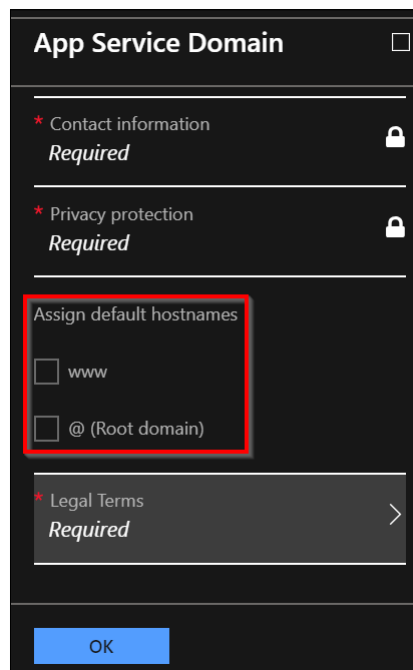
How to do it...

Custom domains

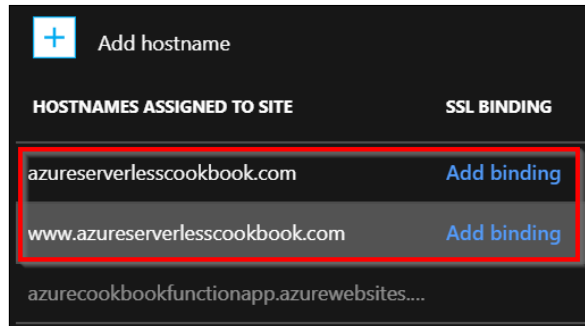
for which you would like to configure a domain:



App Service Domain



by the Azure Management portal. You can view the integration of



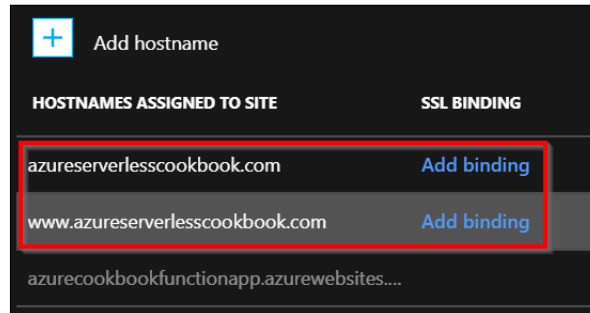
Configuring function app with an existing domain

Create a A record and CNAME record in the domain registrar.
You can get the IP address from the **Custom Domains**

The screenshot shows a table with a search bar at the top. The table has four columns: NAME, TYPE, TTL, and VALUE. The first row is highlighted with a red box. The second row is also highlighted with a red box. The third row is not highlighted. The fourth row is highlighted with a red box.

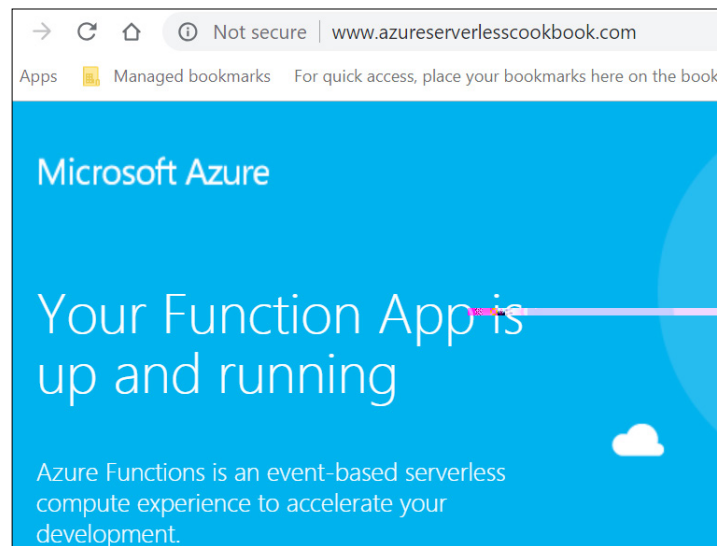
| NAME | TYPE | TTL | VALUE |
|------|-------|--------|---|
| @ | A | 3600 | 40.113.232.243 |
| | NS | 172800 | ns1-07.azure-dns.com ns2-07.azure-dns.net ns3-07.azure-dns.org ns4-07.azure-dns.info |
| | SOA | 3600 | Email: azurebns.hostmaster@microsoft.com Host: ns1-07.azure-dns.com Refresh: 3600 Retry: 300 Expire: 144000 Minimum TTL: 300 Serial number: 1 |
| www | CNAME | 3600 | AzureCookbookFunctionApp.azurewebsites.net |

Custom Domains



That's it. You

You can now browse your function app using the new domain instead of the



Techniques to access Application Settings

In every application, you will have at least a few configuration items that you might

In general, I would classify the configuration items into two categories:

Whatever might be the use of the configuration value, you need to have a place

In this recipe, we will learn how and where to store these configuration items

Getting ready

*Chapter 4 Understanding the
Integrated Developer Experience of Visual Studio Tools*

How to do it...

In this recipe, we will look at a few ways of accessing the configuration values.

Accessing Application Settings and connection strings in the Azure Function code

```
    a configuration items with the MyAppSetting
    ConnectionStrings          sqlldb_connection          local.
    settings.json file. local.settings.json
```

```

1  {
2    "IsEncrypted": false,
3    "FUNCTIONS_WORKER_RUNTIME": "dotnet",
4    "Values": {
5      "AzureWebJobsStorage":
6        "DefaultEndpointsProtocol=https;AccountName=azurefunctionscookb
7        d3zXdZE336X1EfiELvI0d5zCkBW==;EndpointSuffix=core.windows.net",
8      "AzureWebJobsDashboard": "UseDevelopmentStorage=true",
9      "MyAppSetting": "Hello! i'm a value from App Setting"
10   }
11   "ConnectionStrings": {
12     "sqlldb_connection": "connection_string_here"
13   }
14 }

```

that read the configuration values and the connection strings:

```

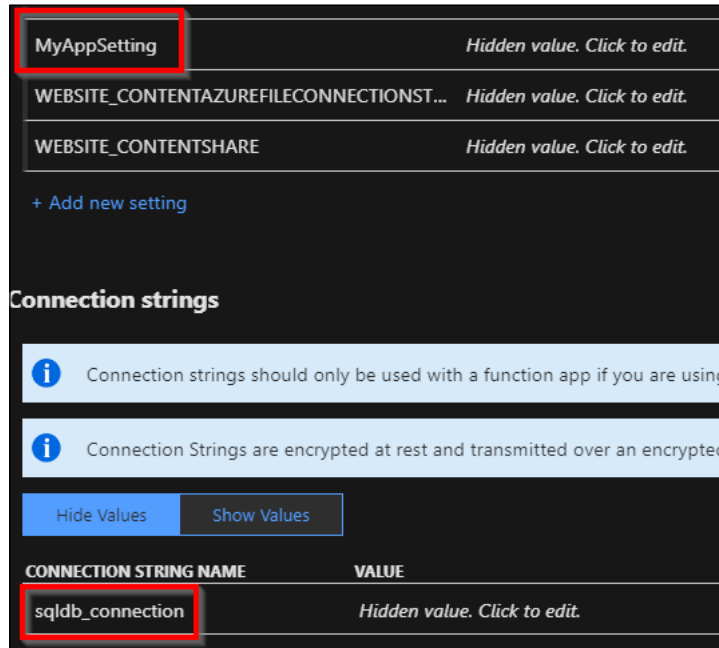
public class HttpTriggerCSharpFromVS
{
    [FunctionName("HttpTriggerCSharpFromVS")]
    public static IActionResult Run([HttpTrigger(AuthorizationLevel.
Anonymous, "get", "post", Route = null)]HttpRequest req, ILogger
logger)
    {
        var configuration = new ConfigurationBuilder()
            .AddEnvironmentVariables()
            .AddJsonFile("appsettings.json", true)
            .Build();
        var ValueFromGetConnectionStringOrSetting = configuration.GetConne
ctio
nStringOrSetting("MyAppSetting");
        logger.LogInformation("GetConnectionStringOrSetting" +
ValueFromGetConnectionStringOrSetting);
        var ValueFromConfigurationIndex= configuration["MyAppSetting"];
        logger.LogInformation("ValueFromConfigurationIndex" +
ValueFromConfigurationIndex);
        var ValueFromConnectionString = configuration.GetConnectionStringO
rSett
ing("ConnectionStrings:sqlldb_connection");
        logger.LogInformation("ConnectionStrings:sqlldb_connection" +
ValueFromConnectionString);
        string name = req.Query["name"];
        return name != null ? (ActionResult)new OkObjectResult($"Hello,
{name}")
            : new BadRequestObjectResult("Please pass a name on the query
string or in the request body");
    }
}

```


project to Azure by right-clicking on the project and then clicking

Publish

Add the configuration key and the connection string in the **Application Settings**



```
2018-10-20T13:17:33 Welcome, you are now connected to log-streaming service.
2018-10-20T13:17:54.660 [Information] Executing 'HttpTriggerCSharpFromVS' (Reason='This function was
programmatically called via the host APIs.', Id=95e43756-962e-4210-bf16-0303be4117f9)
2018-10-20T13:17:54.847 [Information] GetConnectionStringOrSettingHello! i'm a value from App Setting
2018-10-20T13:17:54.847 [Information] ValueFromConfigurationIndexHello! i'm a value from App Setting
2018-10-20T13:17:54.847 [Information] ConnectionStrings:sqlldb_connectionconnection_string_here
2018-10-20T13:17:54.858 [Information] Executed HttpTriggerCSharpFromVS (Succeeded, Id=95e43756-962e-4210-bf16-
0303be4117f9)
```

Application setting – binding expressions

In the previous section, we learned how to access the configuration settings from the code. Sometimes, you might want to configure some of the declarative items too. You could achieve that using binding expression. You

parameter for configuring the QueueTrigger

```
public static IActionResult Run(
    [HttpTrigger(AuthorizationLevel.Anonymous, "get", "post",
    ILogger logger,
    [QueueTrigger("hardcodedqueueName")] string queue)
    {
```

hardcodedqueueName

queue is not a good practice. In order to make it configurable, you need

```
public static IActionResult Run(
    [HttpTrigger(AuthorizationLevel.Anonymous, "get", "post", Route = null)]HttpRequest req,
    ILogger logger,
    [QueueTrigger("%queueName%")] string queue)
```

queueName

%...%
Application Settings


Creating and generating open API specifications using Swagger

there are many tools and standards/specifications that are available for proper documentation for the REST APIs. One such standard is known as the open API specification (it's popularly known as Swagger).

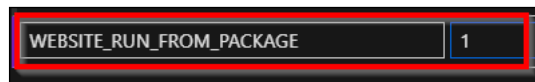
the open API definitions for our HTTP triggers. In this recipe, we will learn how

Getting ready

Get

[ Note that at the time of writing, the API definition feature is]

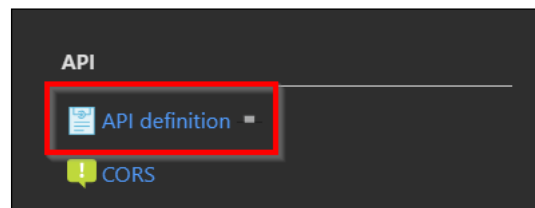
Ensure that the Azure function app is configured to point to runtime version 1,
Application Settings



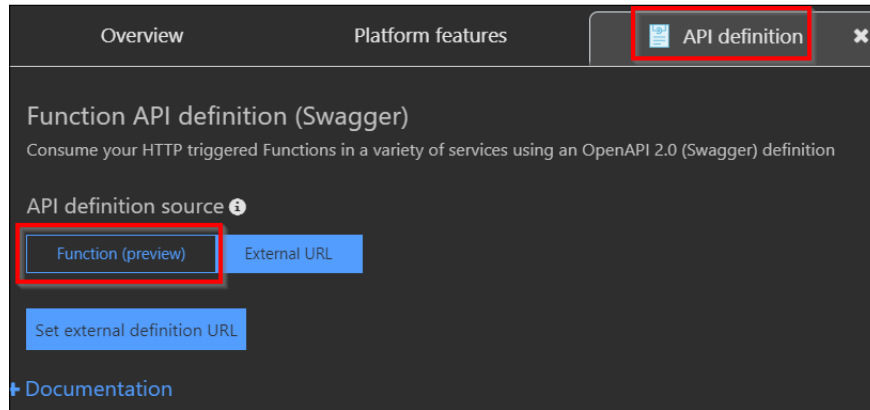
How to do it...

Platform features

API definition

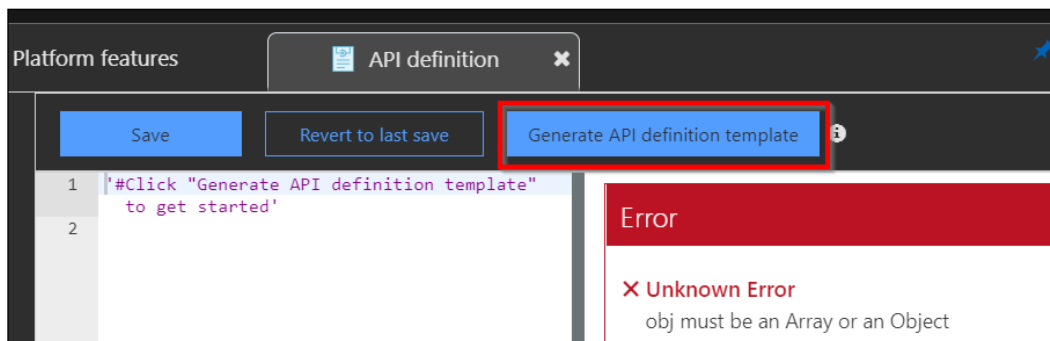


API definition **Function (preview)**
the source of the API definition:

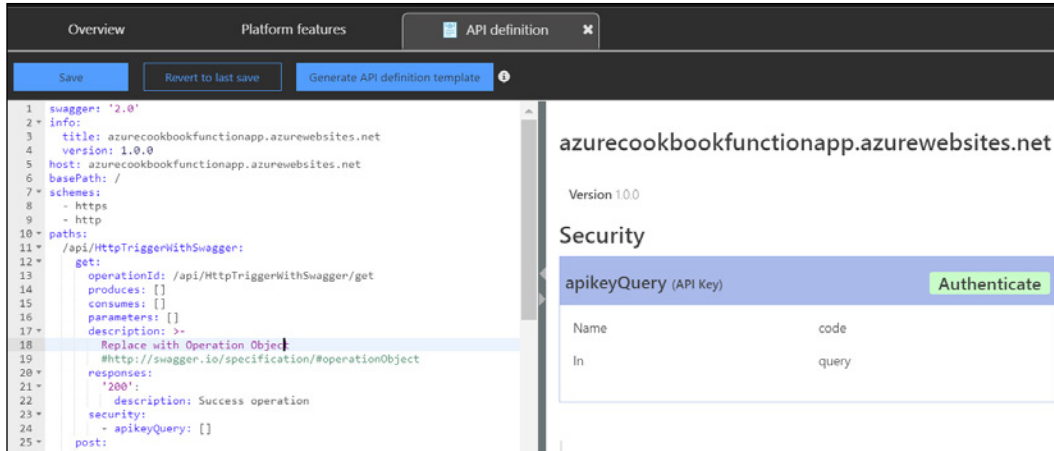


Function (preview)

Don't worry, as the feature is still in preview, Microsoft might fix it before
generally available (GA) **Generate API definition**
template

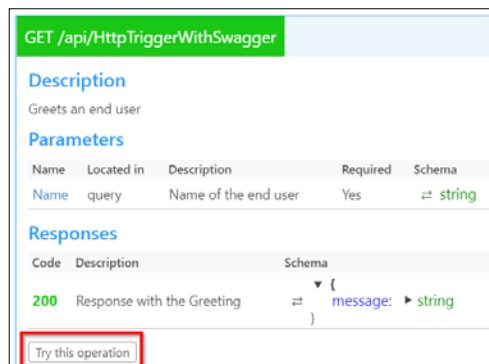


create a template of the open API definition. It's the cloud developer's responsibility to fill in the template, based on the APIs that



required for generating the Swagger definition based on the open specification. The right section shows how the Swagger UI looks. The

Save



Try this operation

Close

Request

Scheme
https

Accept
application/json

Parameters

| Name |
|----------------------|
| <input type="text"/> |

Name of the end user

GET <https://AzureCookbookFunctionApp.azurewebsites.net/api/HttpTriggerWithSwagger?Name=> HTTP/1.1

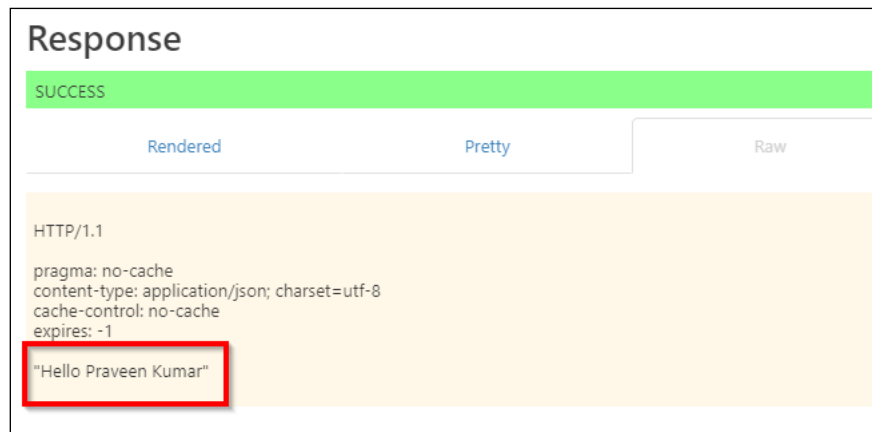
Host: AzureCookbookFunctionApp.azurewebsites.net
Accept: application/json
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US, en;q=0.8, fa;q=0.6, sv;q=0.4
Cache-Control: no-cache
Connection: keep-alive
Origin: https://functions.azure.com
Referer: https://functions.azure.com/node_modules/swagger-editor/index.html
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36

⚠ This is a cross-origin call. Make sure the server at AzureCookbookFunctionApp.azurewebsites.net accepts GET requests from functions.azure.com. [Learn more](#)

Send Request

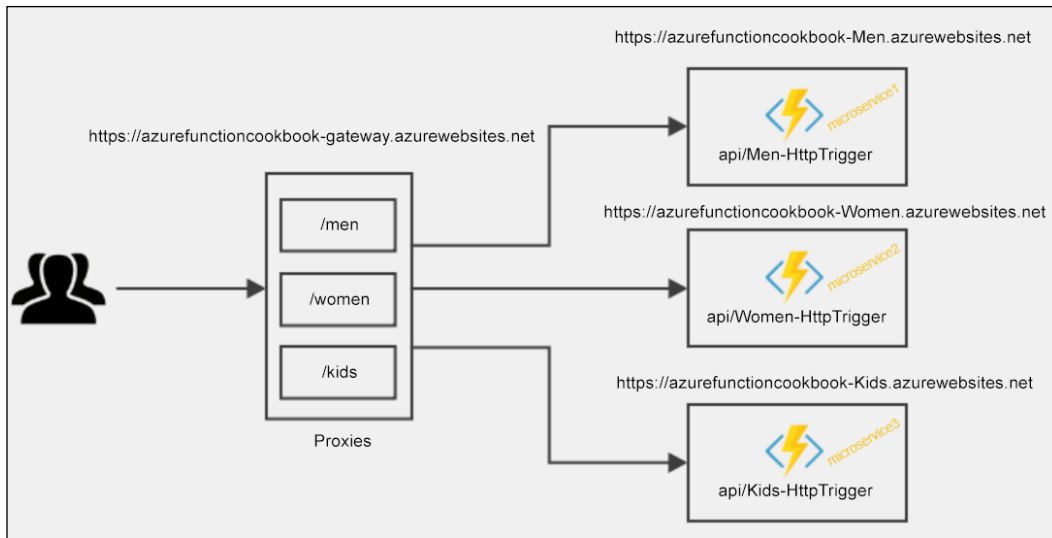
Praveen Kumar

Send Request



Breaking down large APIs into small subsets of APIs using proxies

Getting ready



Let's assume that we are working for an e-commerce portal where we just have

/men /women /kids

How to do it...

Create, proxy and configure the respective microservice

Creating microservices

| NAME | SUBSCRIPTION ID | RESOURCE GROUP |
|---|--------------------------------|-------------------------------|
| AzureCookbookFunctionApp | Visual Studio Enterprise – MPN | AzureCookbookFunctionApp |
| azurefunctioncookbook-gateway | Visual Studio Enterprise – MPN | azurefunctioncookbook-gateway |
| azurefunctioncookbook-Kids 3 | Visual Studio Enterprise – MPN | azurefunctioncookbook-Kids |
| azurefunctioncookbook-Men 1 | Visual Studio Enterprise – MPN | azurefunctioncookbook-Men |
| azurefunctioncookbook-Women | Visual Studio Enterprise – MPN | azurefunctioncookbook-Women |
| azurefunctioncookbook-Kids | Visual Studio Enterprise – MPN | azurefunctioncookbook-Kids |
| azurefunctioncookbook-Men | Visual Studio Enterprise – MPN | azurefunctioncookbook-Men |
| azurefunctioncookbook-Women | Visual Studio Enterprise – MPN | azurefunctioncookbook-Women |

| Http Trigger name | Output message |
|-------------------|--|
| Men-HttpTrigger | Hello <<Name>> - Welcome to the Men Microservice |
| Women-HttpTrigger | Hello <<Name>> - Welcome to the Women Microservice |
| Kids-HttpTrigger | Hello <<Name>> - Welcome to the Kids Microservice |

Creating the gateway proxies

New proxy

Name
Men

Route template
/Men

Allowed HTTP methods
All methods

Backend URL
https://azurefunctioncookbook-men.azurewebsites.net

+ Request override

+ Response override

Create

You will be taken

azurefunctioncookbook-gatew... x

Save Discard Delete proxy Advanced editor

All subscriptions

Function Apps

azurefunctioncookbook... >

Functions +

Proxies +

Kids

Men

Women

Kids

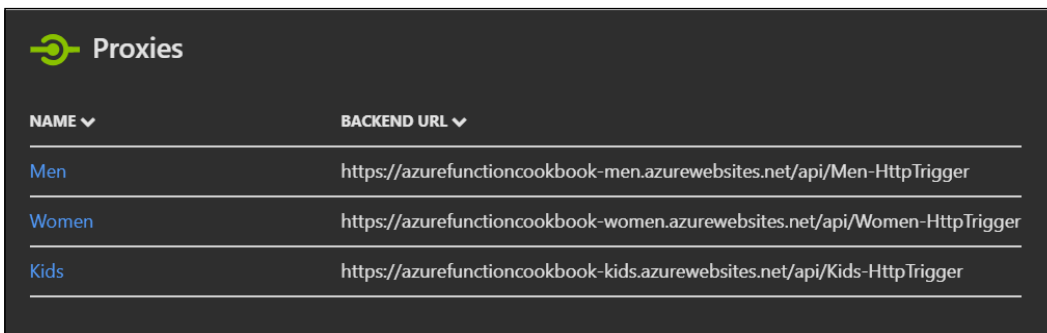
Proxy URL
https://azurefunctioncookbook-gateway.azurewebsites.net/Kids

Route template
/Kids

Allowed HTTP methods
All methods

Backend URL
https://azurefunctioncookbook-kids.azurewebsites.net/api/Kids-HttpTrigger

| Proxy name | Route template | Backend URL (the URLs of the HTTP triggers created in the previous step) |
|------------|----------------|---|
| | | https://azurefunctioncookbook-men.azurewebsites.net/api/Men-HttpTrigger |
| | | https://azurefunctioncookbook-women.azurewebsites.net/api/Women-HttpTrigger |
| | | https://azurefunctioncookbook-kids.azurewebsites.net/api/Kids-HttpTrigger |

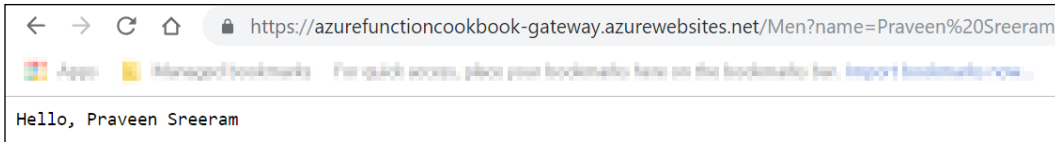


```
https://azurefunctioncookbook-gateway.azurewebsites.net/Men
https://azurefunctioncookbook-gateway.azurewebsites.net/Women
https://azurefunctioncookbook-gateway.azurewebsites.net/Kids
```

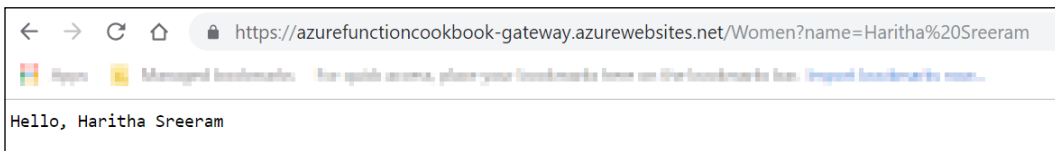
Testing the proxy URLs

name
name

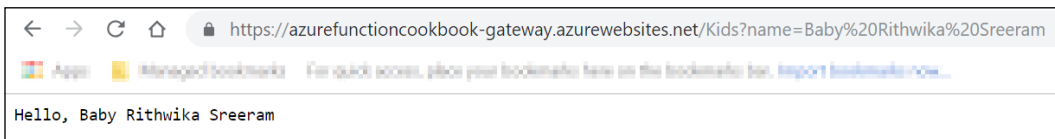
Men



Women



Kids



Observe the URLs of the three screenshots. You will notice that they look like they

There's more...

Best Practices for Azure Functions

Chapter 9 Implementing

client application. You can learn more about Azure Function proxies in the official <https://docs.microsoft.com/azure/azure-functions/functions-proxies>

See also

Controlling access to Azure Functions using function keys Chapter 9
Implementing Best Practices for Azure Functions

Securing Azure Functions using Azure AD recipe Chapter 9 *Implementing Best Practices for Azure Functions*

Configuring throttling of the Azure Functions using API Management
Chapter 9 *Implementing Best Practices for Azure Functions*

Moving configuration items from one environment to another using resources

Every application that you develop will have many configuration items (such as `Web.Config` files for all your .NET-based web applications.

In the traditional on-premises world, the `Web.Config` file would be located in the server and the file would be accessible to all people who have access to the server. Although it is possible to encrypt all the configuration items of `Web.Config`

`Web.Config` files and they work as they used to in the traditional on-premises world.

application settings, where you can configure these settings (either manually

Getting ready

MyApp-Dev

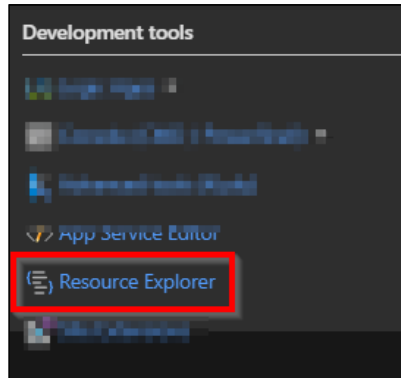
| APP SETTING NAME | VALUE |
|------------------|-------------------------------------|
| AppSetting0 | <i>Hidden value. Click to edit.</i> |
| AppSetting1 | <i>Hidden value. Click to edit.</i> |
| AppSetting2 | <i>Hidden value. Click to edit.</i> |
| AppSetting3 | <i>Hidden value. Click to edit.</i> |
| AppSetting4 | <i>Hidden value. Click to edit.</i> |
| AppSetting5 | <i>Hidden value. Click to edit.</i> |
| AppSetting6 | <i>Hidden value. Click to edit.</i> |
| AppSetting7 | <i>Hidden value. Click to edit.</i> |
| AppSetting8 | <i>Hidden value. Click to edit.</i> |
| AppSetting9 | <i>Hidden value. Click to edit.</i> |

MyApp-Prod

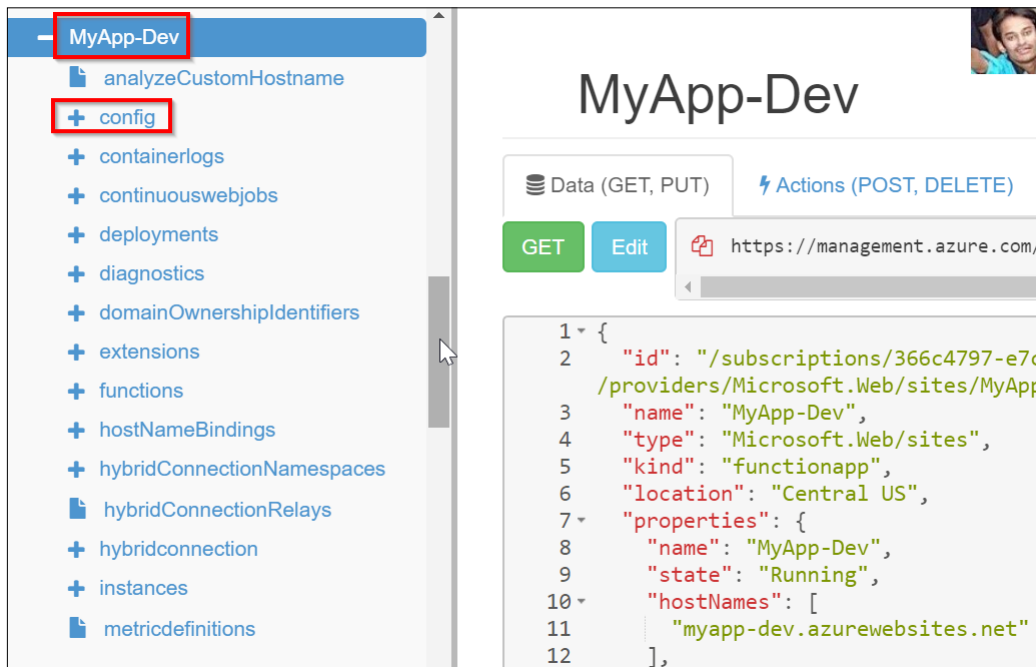
How to do it...

Platform features Resource Explorer

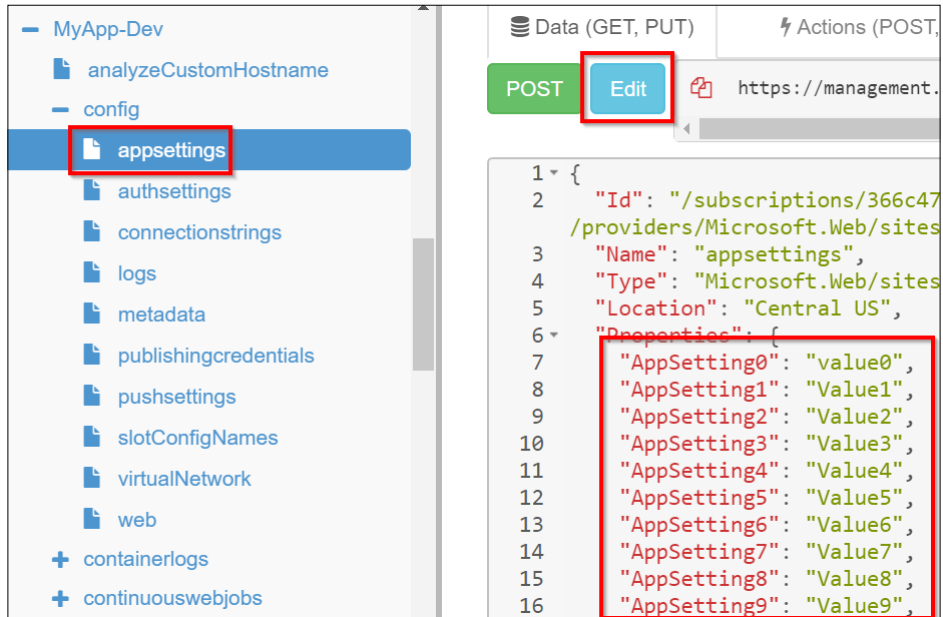
MyApp-Dev



Resource Explorer



config
opens all the items related to configurations:



The screenshot shows the Azure Resource Explorer interface. On the left, the 'config' folder is expanded, and the 'appsettings' resource is selected. The 'Edit' button is highlighted with a red box. The right pane shows the JSON representation of the resource, with the 'Properties' section highlighted in red. The JSON is as follows:

```
1 {
2   "Id": "/subscriptions/366c47
   /providers/Microsoft.Web/sites
3   "Name": "appsettings",
4   "Type": "Microsoft.Web/sites
5   "Location": "Central US",
6   "Properties": {
7     "AppSetting0": "value0",
8     "AppSetting1": "Value1",
9     "AppSetting2": "Value2",
10    "AppSetting3": "Value3",
11    "AppSetting4": "Value4",
12    "AppSetting5": "Value5",
13    "AppSetting6": "Value6",
14    "AppSetting7": "Value7",
15    "AppSetting8": "Value8",
16    "AppSetting9": "Value9",
```

Resource Explorer will display all the application settings in the right-hand
Edit

AppSetting0 AppSetting 9

MyApp - Prod

Resource

Explorer **config appsettings**

The screenshot displays the Azure portal interface for configuring a resource. On the left, the Explorer pane shows a tree view of resources under 'MyApp - Prod', with 'config' expanded and 'appsettings' selected. On the right, the configuration details for 'appsettings' are shown, including a 'PUT' button and a list of app settings. The 'PUT' button is highlighted with a red box. The app settings list includes 'AppSetting0' through 'AppSetting9', each with a value. The list is also highlighted with a red box.

```
1 {
2   "id": "/subscriptions/366c479
3   -Prod/config/appsettings",
4   "name": "appsettings",
5   "type": "Microsoft.Web/sites/
6   "location": "Central US",
7   "properties": {
8     "FUNCTIONS_WORKER_RUNTIME":
9     "AzureWebJobsStorage": "Def
10    W81apCnPLU6jvkY2f+2W53pF1JV03wy
11    "FUNCTIONS_EXTENSION_VERSIO
12    "WEBSITE_CONTENTAZUREFILECO
13    A+dAHVYXyhXjqqu463cU0W81apCnPLU
14    "WEBSITE_CONTENTSHARE": "my
15    "WEBSITE_NODE_DEFAULT VERSI
16    "AppSetting0": "value0",
17    "AppSetting1": "value1",
18    "AppSetting2": "value2",
19    "AppSetting3": "value3",
20    "AppSetting4": "value4",
21    "AppSetting5": "value5",
22    "AppSetting6": "value6",
23    "AppSetting7": "value7",
24    "AppSetting8": "value8",
25    "AppSetting9": "value9"
26  }
```

Edit

PUT

MyApp-Prod

| APP SETTING NAME | VALUE |
|------------------|-------------------------------------|
| AppSetting0 | <i>Hidden value. Click to edit.</i> |
| AppSetting1 | <i>Hidden value. Click to edit.</i> |
| AppSetting2 | <i>Hidden value. Click to edit.</i> |
| AppSetting3 | <i>Hidden value. Click to edit.</i> |
| AppSetting4 | <i>Hidden value. Click to edit.</i> |
| AppSetting5 | <i>Hidden value. Click to edit.</i> |
| AppSetting6 | <i>Hidden value. Click to edit.</i> |
| AppSetting7 | <i>Hidden value. Click to edit.</i> |
| AppSetting8 | <i>Hidden value. Click to edit.</i> |
| AppSetting9 | <i>Hidden value. Click to edit.</i> |

You should
Explorer

Resource

11

Implementing and Deploying Continuous Integration Using Azure DevOps

Introduction

Build the application and fix any errors



`https://dev.azure.com
visualstudio.com`

`https://www.`

`https://dev.azure.com`

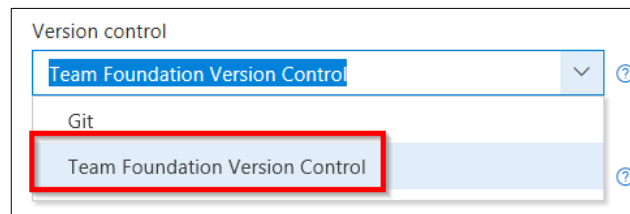
Prerequisites

`azure.com`

`https://dev.`

Control

**Git Team Foundation Version Control
Team Foundation Version**



Configure the Visual Studio project that you developed in Chapter 4, *Understanding the Integrated Developer Experience of Visual Studio Tools for Azure Functions*



Continuous integration – creating a build definition

A build definition is a set of tasks that are required to configure an automated build

Getting ready

Team Foundation Version Control

Create new project ✕

Project name *
AzureServerlessCookBook ✓

Description

Visibility

Public
Anyone on the internet can view the project. Certain features like TFVC are not supported.

Private
Only people you give access to will be able to view this project.

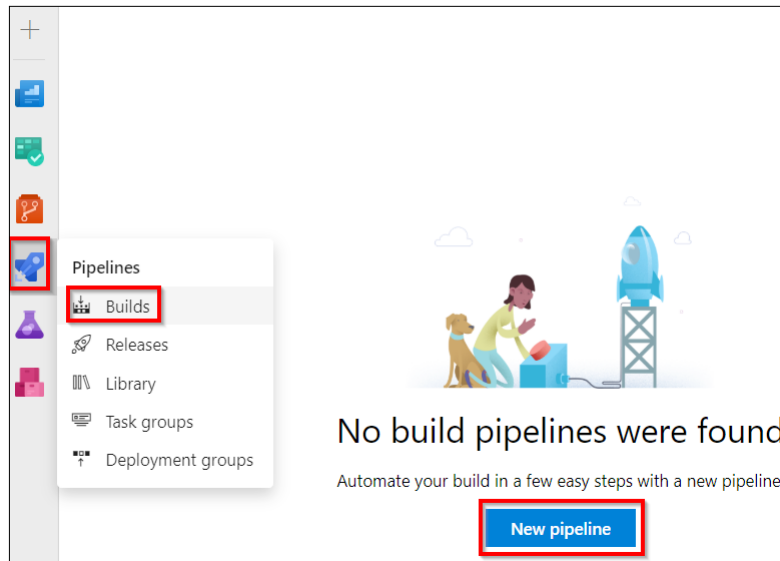
^ Advanced

Version control ?
Team Foundation Version Control ▾

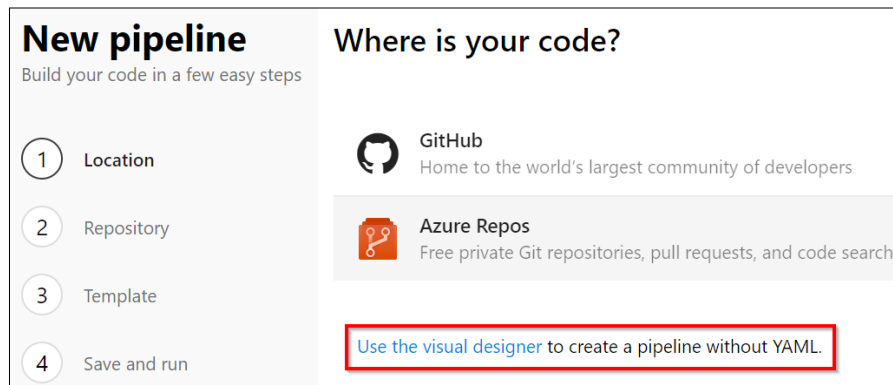
Work item process ?
Agile ▾

How to do it...

Pipelines
Builds **New pipeline**
build definition, as shown in the following screenshot:



Use a visual designer



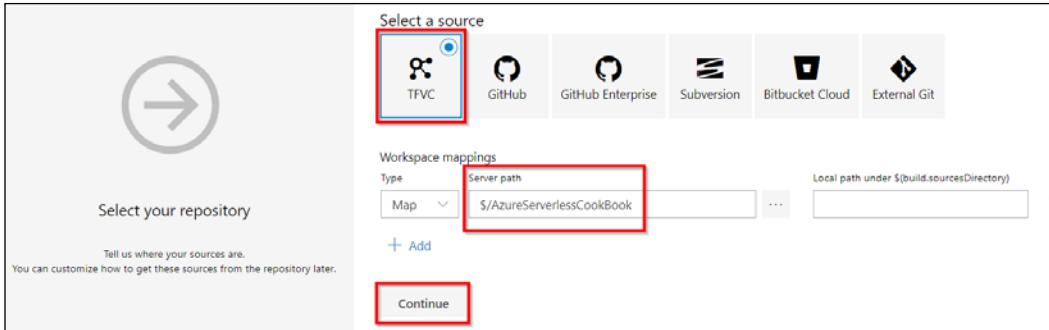
Select your repository

TFVC

TFVC

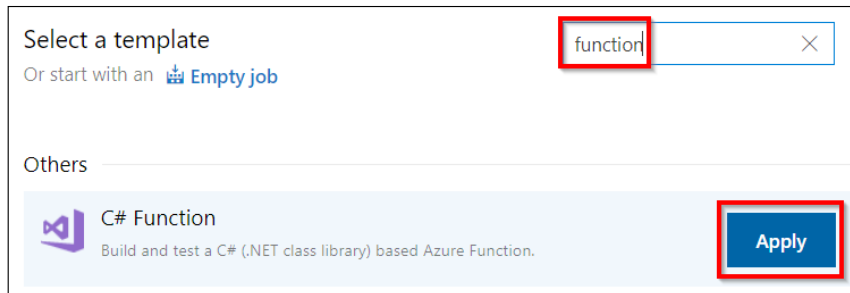
Continue

AzureServerlessCookBook



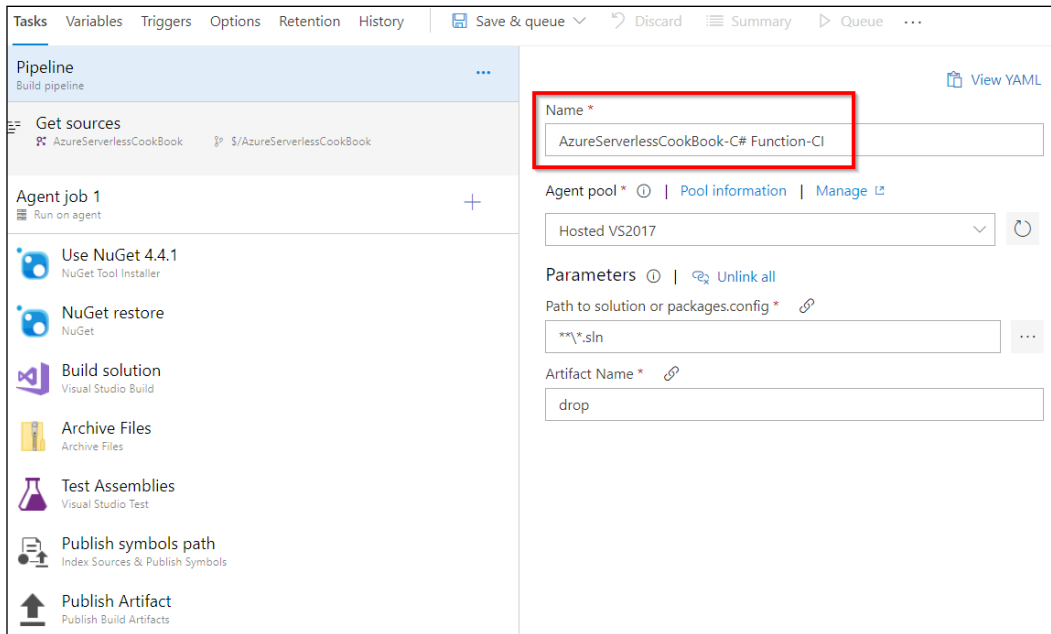
Select a template

C# Function
Apply



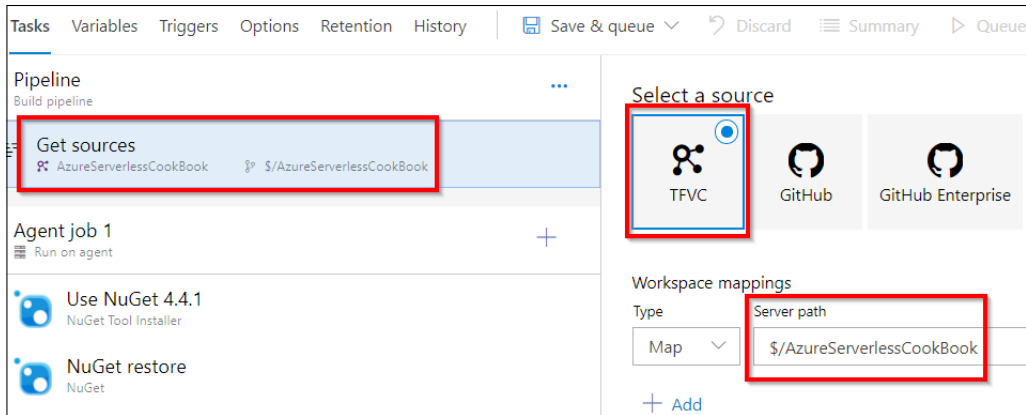
build step is a set of steps used to define the build template, where for each of those fields based on our requirements. Let's start by providing

Hosted VS2017 Agent Pool



Get sources

AzureServerlessCookBook



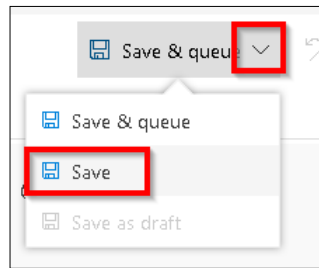
Leave the default

- **Use NuGet and NuGet restore**
- **Build solution**
- **Test assemblies**
- **Archive files**
- **Publish symbols path**
- **Publish artifact**

Continuous integration

– executing unit test cases in the pipeline

Once you review all the values in all the fields, click on Save, as shown in the following screenshot, and click on Save again in the Save build definition



How it works...

A build definition is just a blueprint of the tasks that are required for building the build definition. We can choose a blank template and create the definition by

build definition (either manually or automatically, which will be in which you have configured them. You can also rearrange the steps by dragging

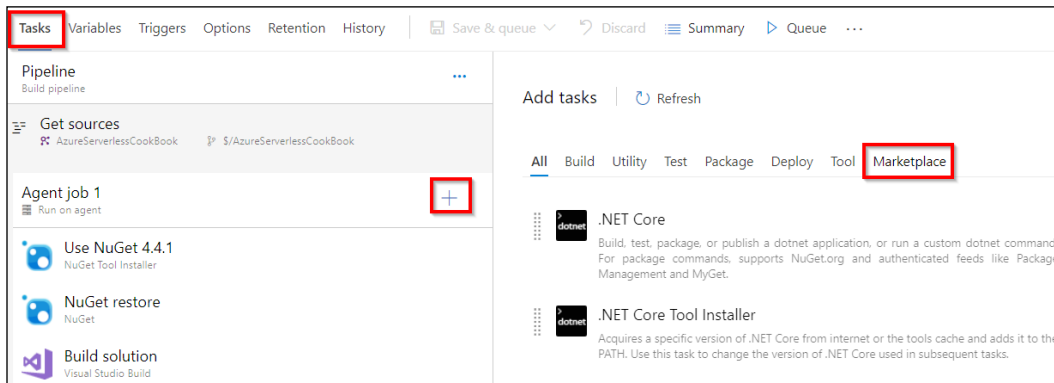
it in a folder configured for the `build.artifactstagingdirectory` **Path to publish** field of the **Publish artifact**

Variables

| Pipeline variables | | Name ↑ | Value | Settable at queue time |
|------------------------|--|---------------------|--------------------------------------|-------------------------------------|
| Variable groups | | BuildConfiguration | release | <input checked="" type="checkbox"/> |
| Predefined variables ↗ | | BuildPlatform | any cpu | <input checked="" type="checkbox"/> |
| | | system.collectionId | a00560d2-16b5-48e7-9eb3-601c04d7e9bd | |
| | | system.debug | false | <input checked="" type="checkbox"/> |
| | | system.definitionId | 7 | |
| | | system.teamProject | AzureServerlessCookBook | |

There's more...

Add Task (+)



Marketplace

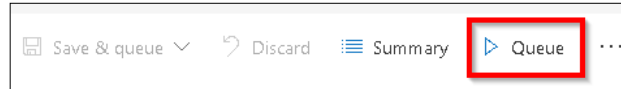
Continuous integration – queuing a build and triggering it manually

In the previous recipe, you learned how to create and configure the build definition.

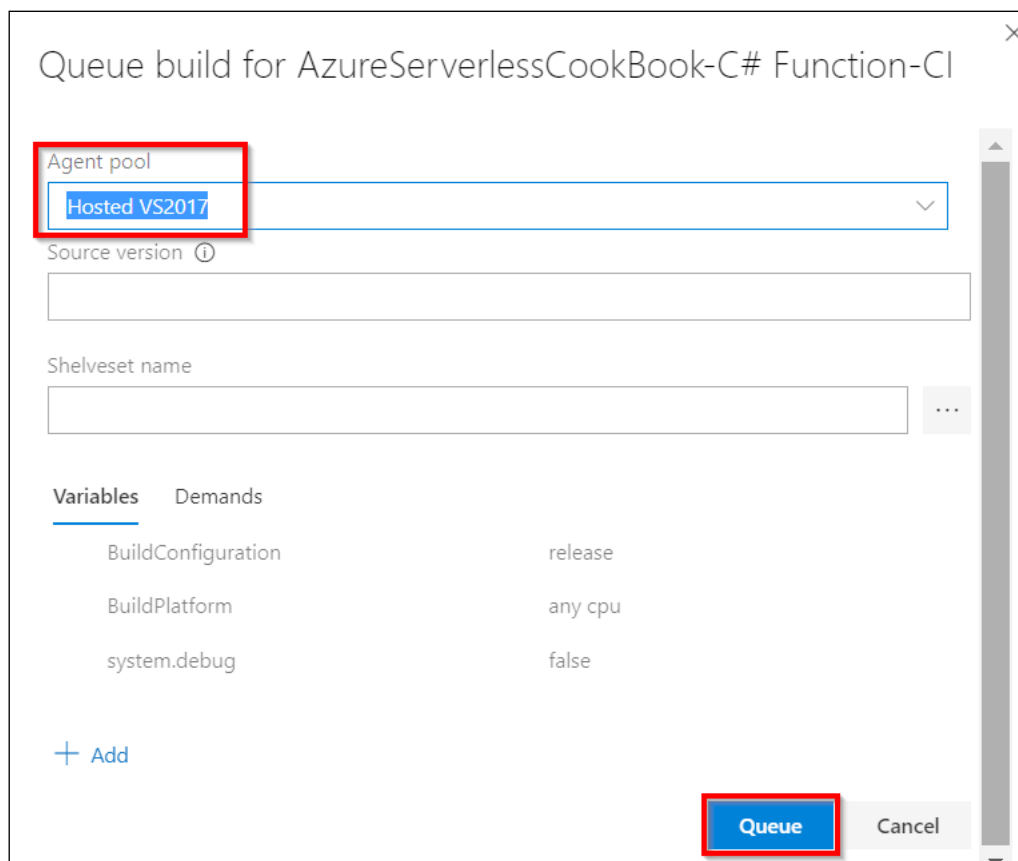
Getting ready

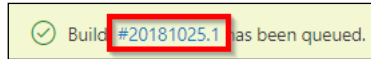
How to do it...

build definition named AzureServerlessCookBook-C#
Function-CI **Queue**



Queue build for AzureServerlessCookBook-C# Function-CI
Hosted VS2017 **Agent pool**
Queue





20181025.1

Test Assemblies

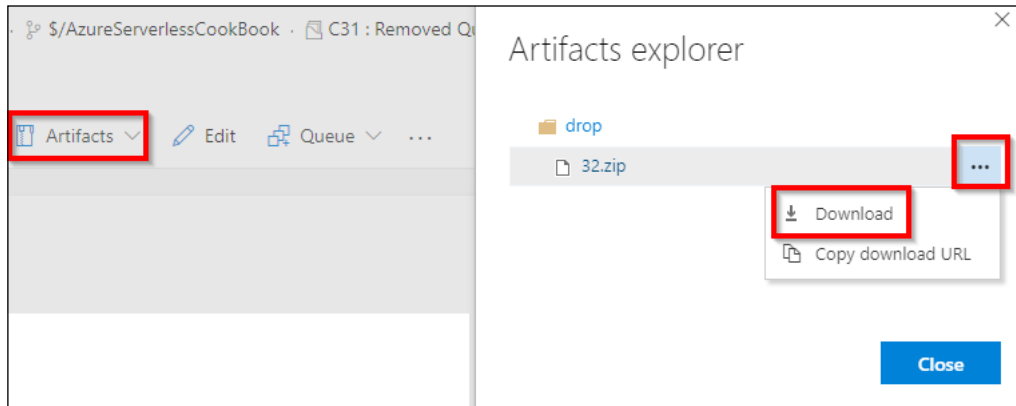
for now. We will fix this in the recipe *Continuous integration – executing unit test cases in the pipeline*

A screenshot of the Azure DevOps build interface. The top navigation bar includes "Logs", "Summary", "Tests", "Release", "Artifacts", "Edit", and "Queue". The main heading is "Agent job 1 Job" with subtext "Pool: Hosted VS2017 · Agent: Hosted Agent". A list of build steps is shown, each with a green checkmark and the word "succeeded". The "Test Assemblies" step is highlighted with a red box and includes a "1 warning" indicator. A "View" button is visible on the right side of the list.

| Step | Status |
|----------------------|---------------------|
| Prepare job | succeeded |
| Initialize Agent | succeeded |
| Initialize job | succeeded |
| Checkout | succeeded |
| Use NuGet 4.4.1 | succeeded |
| NuGet restore | succeeded |
| Build solution | succeeded |
| Archive Files | succeeded |
| Test Assemblies | succeeded 1 warning |
| Publish symbols path | succeeded |
| Publish Artifact | succeeded |
| Post-job: Checkout | succeeded |

Artifacts

files by clicking on the **Download**



Configuring and triggering an automated build

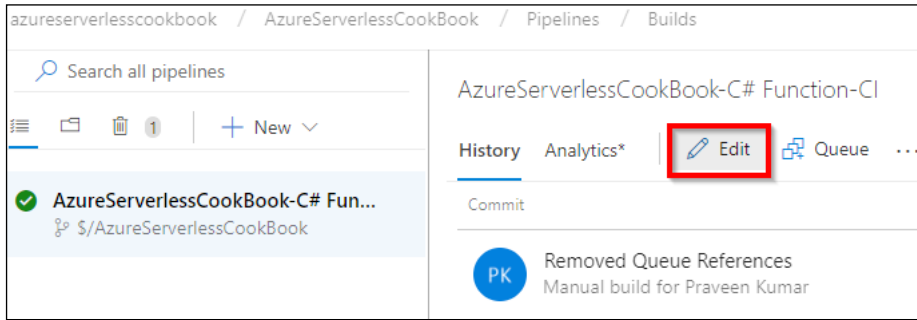
DevOps. It would make sense if we can configure **Continuous Integration CI**

In this recipe, you will learn how to configure continuous integration in Azure

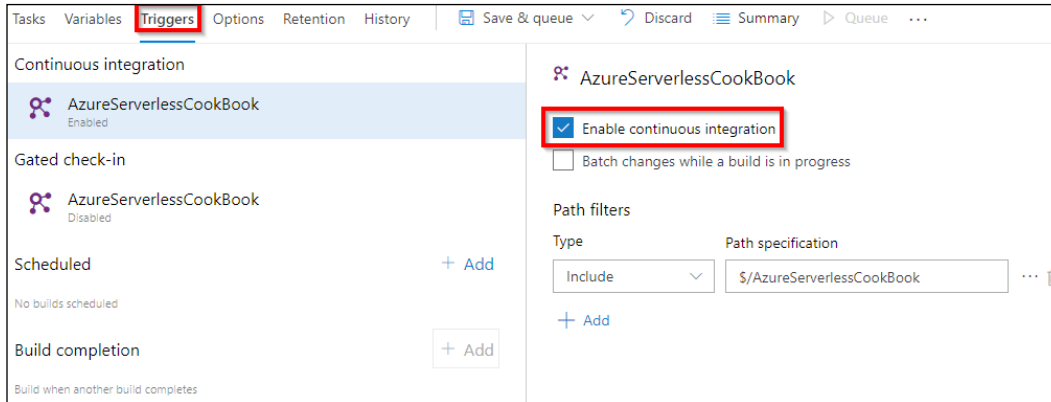
Understanding the Integrated Developer Experience of Visual Studio Tools for Azure Functions

How to do it...

Navigate to the build definition AzureServerlessCookBook-C# Function-CI
Edit



Once you are in the build definition, click on the **Triggers**

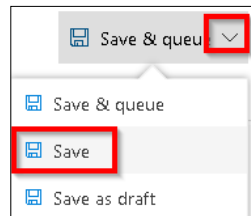


Enable continuous integration

& queue

Save

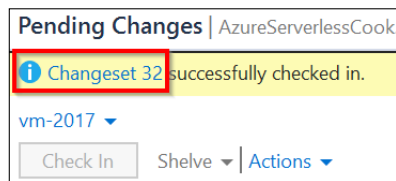
Save



Let's navigate to the Azure Function project in Visual Studio. Make a small
 Run
 hello Automated Build Trigger test by
 return name != null ? (ActionResult)new OkObjectResult(\$"Automated
 Build Trigger test by, { name}")
 : new BadRequestObjectResult("Please pass a name on the
 query string or in the request body");

Let's check in the code and commit the changes to the source control.

Changeset 32



Now, immediately navigate back to the Azure DevOps build definition to

| Commit | Build # | Branch | Queued ↓ |
|---|------------|------------------------|--------------------|
| PK Modified the Welcome Messages to demo CI CI build for Praveen Kumar | 20181025.2 | \$/AzureServerlessC... | 2018-10-25 · 10:15 |
| PK Removed Queue References Manual build for Praveen Kumar | 20181025.1 | \$/AzureServerlessC... | 2018-10-25 · 08:36 |

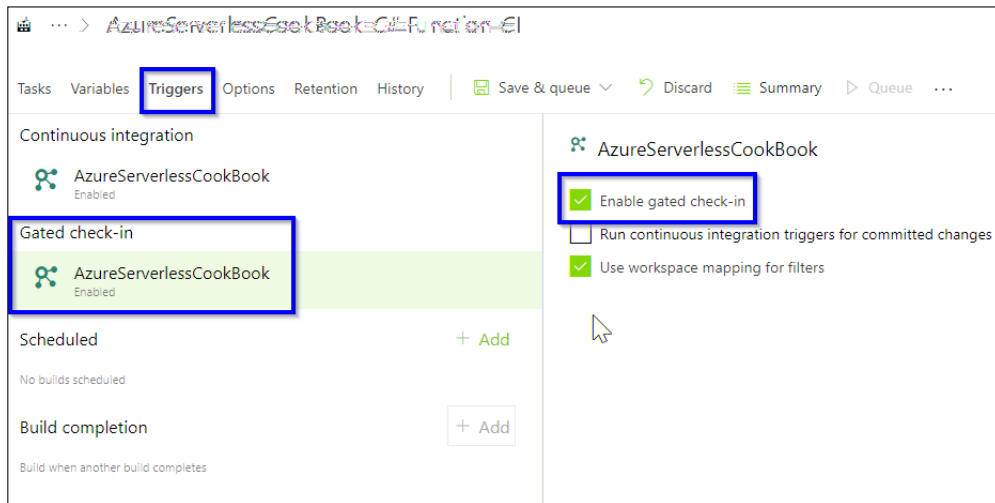
How it works...

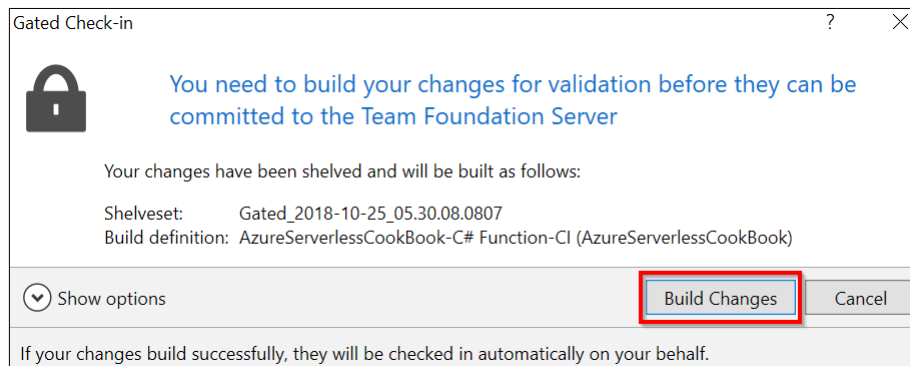
automatic build trigger for the build definition

There's more...

definition and then navigate to the **Triggers**

Enable gated check-in





Build Changes



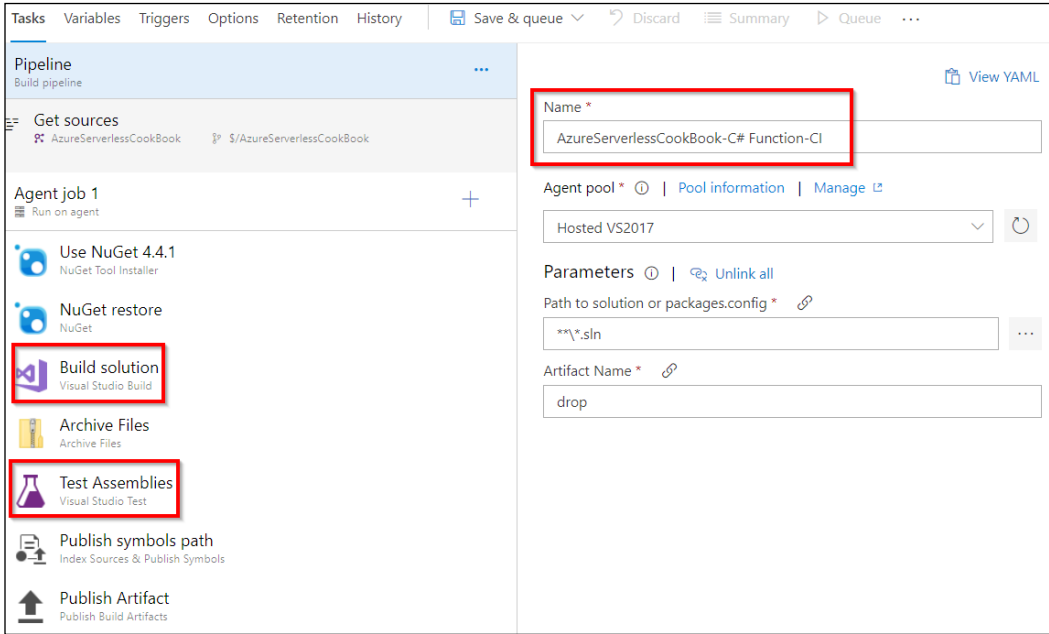
Continuous integration – executing unit test cases in the pipeline

*Developing Unit Tests for Azure Functions HTTP Triggers
Exploring Testing Tools for the Validation of Azure Functions*

How to do it...

Continuous integration – creating a build definition

Assemblies **Build solution** **Test**



Test Assemblies

configuration with the one

Test files field:

```
**\$(BuildConfiguration)\**\*test*.dll  
!**\obj\**  
!**\*TestAdapter.dll
```

The previous configuration settings let the test runner do the following:

```
o .dll Test  
release  
release  
$(BuildConfiguration) Variables
```

| Tasks | | Variables | Triggers | Options | Retention | History | Save & queue | Discard | Summary | Queue | ... |
|----------------------|--|---------------------|--------------------------------------|---------|-----------|---------|--------------|---------|---------|-------|-----|
| Pipeline variables | | Name ↑ | Value | | | | | | | | |
| Variable groups | | BuildConfiguration | release | | | | | | | | |
| Predefined variables | | BuildPlatform | any cpu | | | | | | | | |
| | | system.collectionId | a00560d2-16b5-48e7-9eb3-601c04d7e9bd | | | | | | | | |
| | | system.debug | false | | | | | | | | |
| | | system.definitionId | 9 | | | | | | | | |
| | | system.teamProject | AzureServerlessCookBook | | | | | | | | |

o .dll obj
 .dll release

That's it. Let's now queue the

Queue

| Logs | Summary | Tests | Release | Artifacts | Edit | Queue | ... |
|---|----------------------|-------|-----------|--------------------|------|-------|-----|
| Agent job 1 Job | | | | | | | |
| Pool: Hosted VS2017 · Agent: Hosted Agent | | | | | | | |
| ✓ | Prepare job | · | succeeded | | | | |
| ✓ | Initialize job | · | succeeded | | | | |
| ✓ | Initialize Agent | · | succeeded | View detailed logs | | | |
| ✓ | Checkout | · | succeeded | | | | |
| ✓ | Use NuGet 4.4.1 | · | succeeded | | | | |
| ✓ | NuGet restore | · | succeeded | | | | |
| ✓ | Build solution | · | succeeded | | | | |
| ✓ | Archive Files | · | succeeded | | | | |
| ✓ | Test Assemblies | · | succeeded | | | | |
| ✓ | Publish symbols path | · | succeeded | | | | |
| ✓ | Publish Artifact | · | succeeded | | | | |
| ✓ | Post-job: Checkout | · | succeeded | | | | |



There's more...

test
.dll file), then feel free to

```
**\$(BuildConfiguration)\**\*whateverwordyouhaveinthenameofthedllfile*.dll
```

* * * !, which are called file matching patterns. You can learn more about the file matching patterns <https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/file-matching-patterns?view=vsts>

Creating a release definition

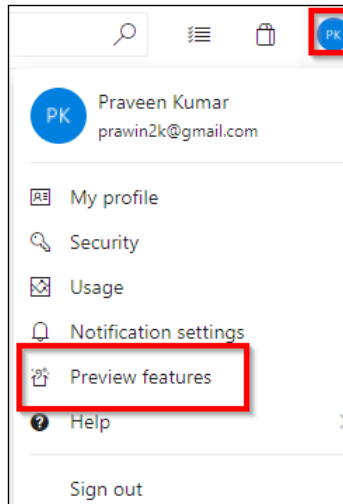
create a build definition and trigger an automated build

In order to do that, first we need to create a release definition in the same way that we created the build definitions.

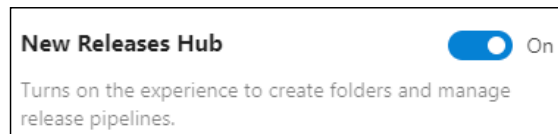
Getting ready

I have used the new release definition editor to visualise the deployment pipelines. The release definition editor is still in preview. By the time you read this, if it is still in preview, then you can enable it by clicking on the profile image and then clicking

Preview features



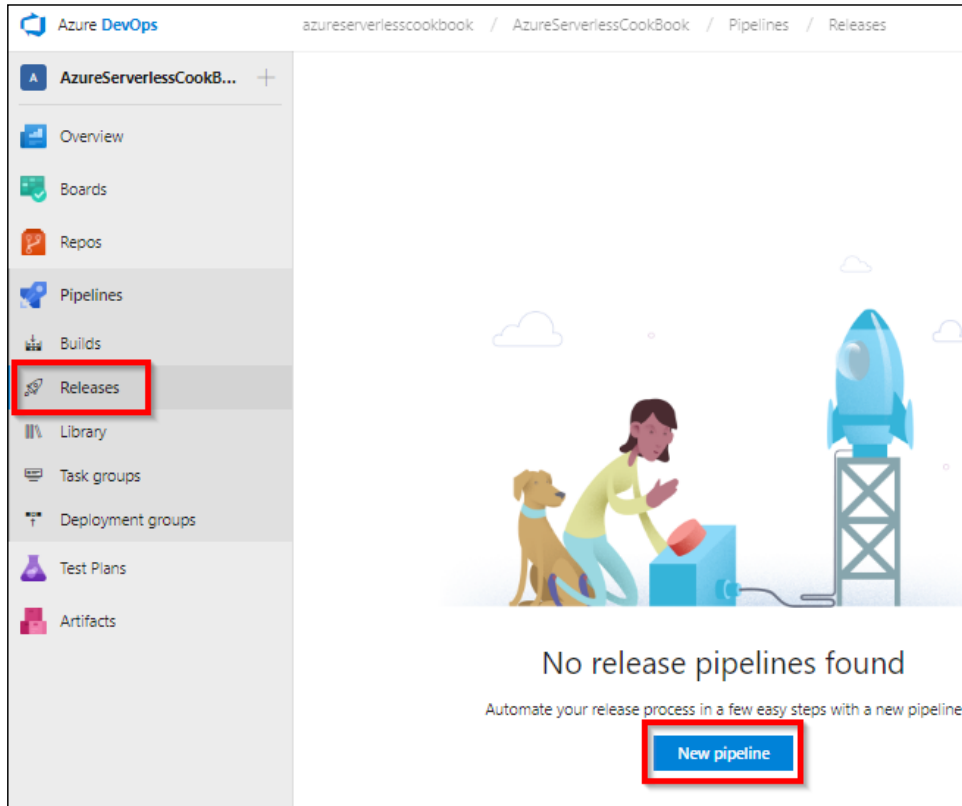
You can then enable new release definition editor, as shown in the following



Let's get started with creating a new release definition.

How to do it...

Releases New Pipeline



a template
Apply


Release definition
Azure App Service deployment
Apply

Select
Environment

Select a template

Or start with an [Empty job](#)

Featured

 **Azure App Service deployment**
Deploy your application to Azure App Service. Choose from Web App on Windows, Linux, containers, Function Apps, or WebJobs. [Apply](#)

Add

Artifacts

All pipelines > **New release pipeline**

Pipeline Tasks Variables Retention Options History

Artifacts | [+ Add](#)

+ Add an artifact

Schedule not set

Stages | [+ Add](#)

Stage 1
1 job, 1 task

Add an artifact

Source type **Build**

Project

Source (build definition)

Default Version **Latest**

Add an artifact

Source type

✓ Build Azure Repos ... GitHub TFVC

4 more artifact types ▾

Project * ⓘ
AzureServerlessCookBook ▾

Source (build pipeline) * ⓘ
AzureServerlessCookBook-C# Function-CI ▾

Default version * ⓘ
Latest ▾

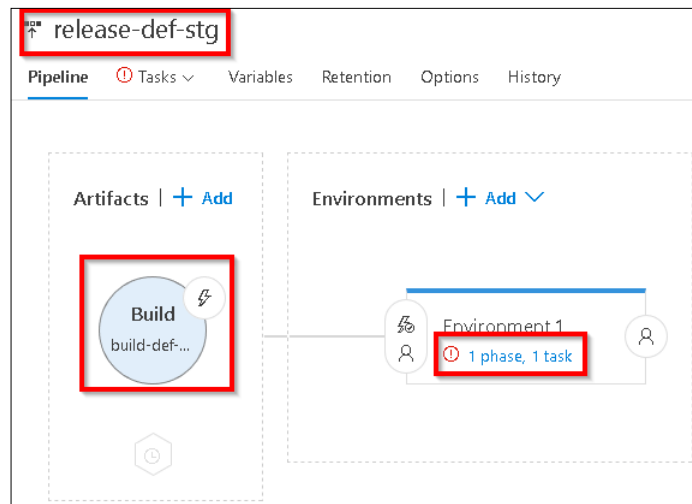
Source alias * ⓘ
_AzureServerlessCookBook-C# Function-CI

ⓘ The artifacts published by each version will be available for deployment in release pipelines. The latest successful build of **AzureServerlessCookBook-C# Function-CI** published the following artifacts: **drop**.

Add

Add

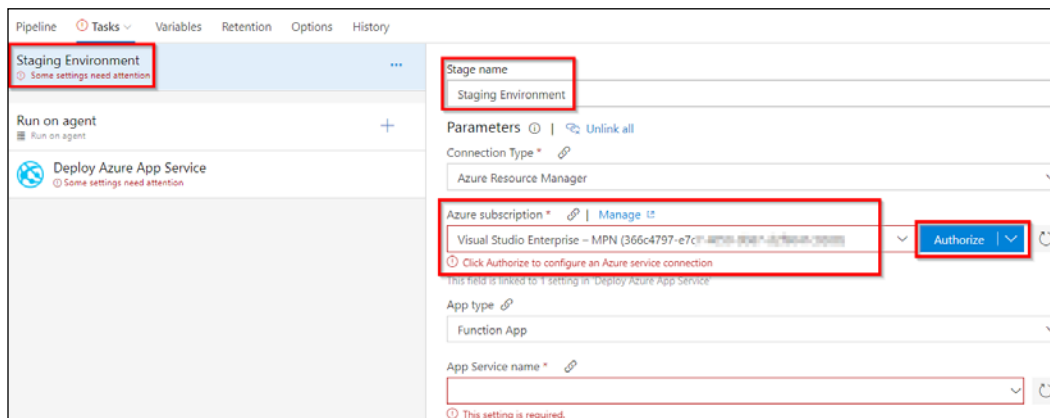
Once the artifact is added, the next step is to configure the stages, where **1 phase, 1 task** in the following screenshot: Also, change the name of the release definition `release-def-stg`



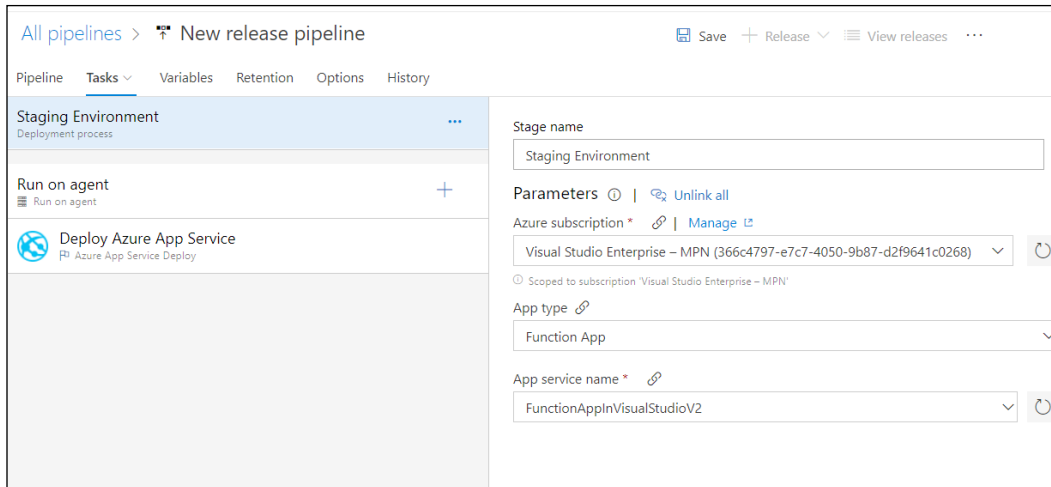
Tasks

Stage name field. I have provided the name Staging Environment

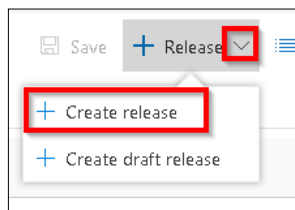
Authorize



App type Function App
App Service name



Save
definition and try to create new release by clicking on **Create release**



Create a new release

can configure the build definition that needs to be used. As we have only definition. You also need to choose the

Create

Create a new release
New release pipeline

Pipeline ^
Click on a stage to change its trigger from automated to manual.

Staging Envirc

Stages for a trigger change from automated to manual. ⓘ
Staging Environment

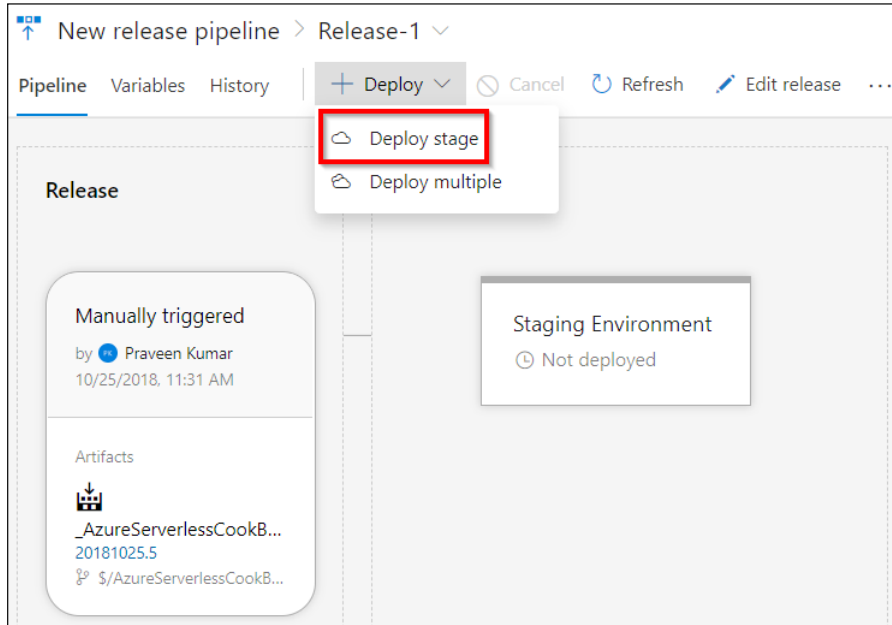
Artifacts ^
Select the version for the artifact sources for this release

| Source alias | Version |
|----------------------------------|------------|
| _AzureServerlessCookBook-C# F... | 20181025.5 |

Release description

Create Cancel

Create Deploy stage




Deploy

Staging Environment
Deploy release

[Overview](#) [Commits](#) [Work Items](#)

To be deployed (Deploying for the first time)
Release-1

Artifacts

 [_AzureServerlessCookBook-C# Function-CI / 20181025.5](#)
[\\$/AzureServerlessCookBook](#)

Comment

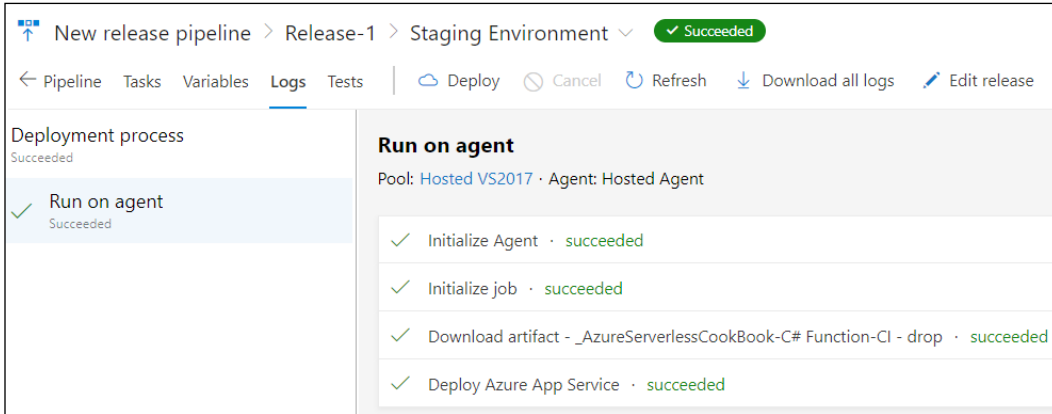
Deploy

In Progress

Stages

Staging Environment
▶ In progress

In Progress



How it works...

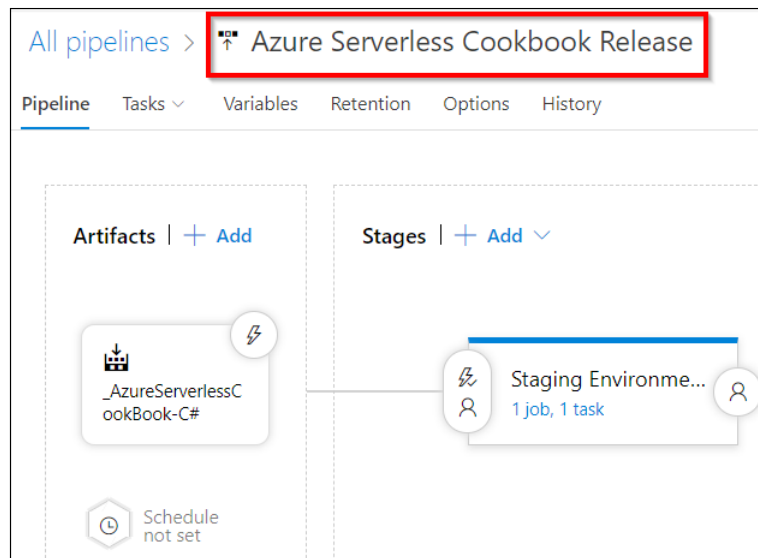
Pipeline

We have also configured the environment to have the Azure App Service related
*Understanding the Integrated
Developer Experience of Visual Studio Tools for Azure Functions*

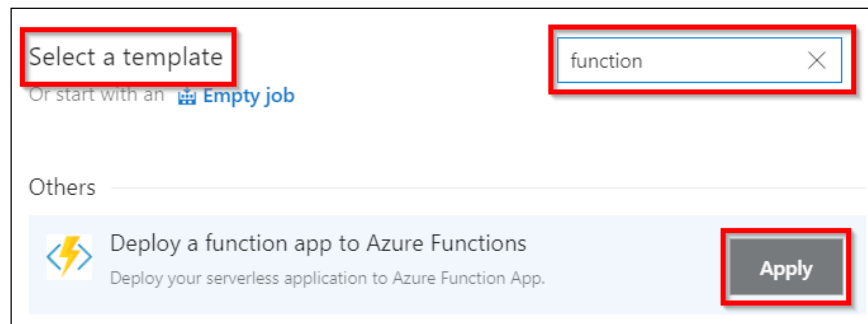
There's more...

If you are configuring continuous deployment for the first time, you might see a

Authorise Azure App Service deployment
Authorise



Currently, there is a template specific to Azure Functions, shown next. However,



See also

*Deploying an Azure Function app to Azure Cloud using Visual Studio
Understanding the Integrated Developer Experience of Visual Studio Tools
for Azure Functions*

Triggering the release automatically

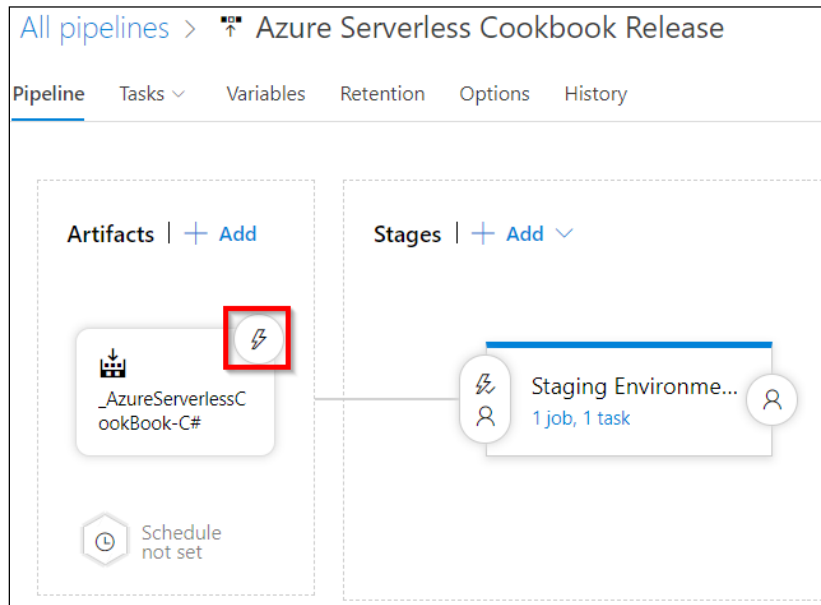
learn how to configure continuous deployment for an environment. In your project, you can configure dev/staging or any other pre-production environment, and configure continuous deployment to streamline

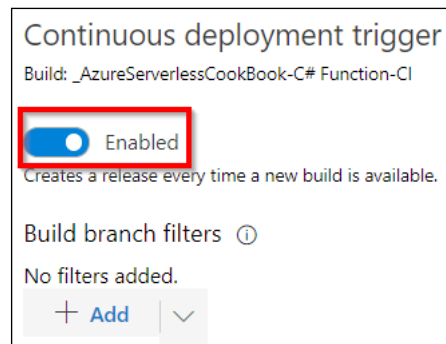
In general, it is not recommended that you configure continuous deployment for requirements. Be cautious and think about various scenarios before you configure

Getting ready

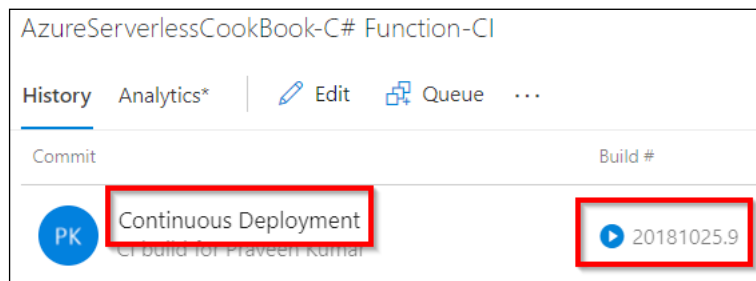
How to do it...

By default, the releases are configured to be pushed manually. Let's configure Continuous Deployment by navigating back to the **Pipeline Continuous deployment trigger**



Continuous deployment**trigger****Save**

```
return name != null ? (ActionResult)new OkObjectResult($"Automated
Build Trigger and Release test by, { name}")
    : new BadRequestObjectResult("Please pass a name on the
query string or in the request body");
```

Continuous Deployment**Builds**

Releases

| Releases | Created | Stages |
|--|------------------|---|
| Release-3 20181025... \$/AzureServerlessCookBook | 2018-10-25 12:27 | <input type="radio"/> Staging E... |
| Release-2 20181025... \$/AzureServerlessCookBook | 2018-10-25 12:20 | <input checked="" type="radio"/> Staging E... |
| Release-1 20181025... \$/AzureServerlessCookBook | 2018-10-25 12:04 | <input checked="" type="radio"/> Staging E... |

```
1 "Automated Build Trigger & Release Trigger test by Praveen Sreeram"
```

How it works...

Pipeline

Continuous deployment trigger

AzureServerlessCookBook-C# Function-CI
Azure Serverless Cookbook Release

There's more...

You can also create multiple environments and configure the definitions to release

Other Books You May Enjoy



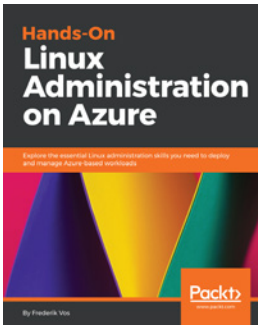
Architecting Microsoft Azure Solutions – Exam Guide 70-535

Sjoukje

- ▶
- ▶
- ▶

- ▶
- ▶
- ▶ Architect state-of-the-art solutions using Artificial Intelligence, IoT and Azure

- ▶ different automation solutions that are incorporated in the Azure platform



Hands-On Linux Administration on Azure

- ▶
- ▶ Master essential Linux skills and learn to find your way around the Linux
- ▶
- ▶ Use configuration management to manage Linux in Azure
- ▶
- ▶
- ▶
- ▶
- ▶

Leave a review – let other readers know what you think

review on this book's Amazon page. This is vital so that other potential readers can see

time, but is valuable to other potential customers, our authors and Packt. Thank you!

