



# Developing Microsoft Azure Solutions

SECOND EDITION

Exam Ref 70-532

Zoiner Tejada  
Michele Leroux Bustamante  
Ike Ellis

FREE SAMPLE CHAPTER

SHARE WITH OTHERS



# **Exam Ref 70-532**

# **Developing Microsoft**

# **Azure Solutions**

## **2nd Edition**

**Zoiner Tejada**  
**Michele Leroux Bustamante**  
**Ike Ellis**

## **Exam Ref 70-532 Developing Microsoft Azure Solutions, 2nd Edition**

**Published with the authorization of Microsoft Corporation by:  
Pearson Education, Inc.**

**Copyright © 2018 by Pearson Education**

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms, and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit [www.pearsoned.com/permissions/](http://www.pearsoned.com/permissions/). No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-1-5093-0459-2

ISBN-10: 1-5093-0459-X

Library of Congress Control Number: 2017953300

1 18

### **Trademarks**

Microsoft and the trademarks listed at <https://www.microsoft.com> on the “Trademarks” webpage are trademarks of the Microsoft group of companies. All other marks are property of their respective owners.

### **Warning and Disclaimer**

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The authors, the publisher, and Microsoft Corporation shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or programs accompanying it.

### **Special Sales**

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at [corpsales@pearsoned.com](mailto:corpsales@pearsoned.com) or (800) 382-3419.

For government sales inquiries, please contact [governmentsales@pearsoned.com](mailto:governmentsales@pearsoned.com).

For questions about sales outside the U.S., please contact [intlcs@pearson.com](mailto:intlcs@pearson.com).

<b>Editor-in-Chief</b>	Greg Wiegand
<b>Acquisitions Editor</b>	Laura Norman
<b>Development Editor</b>	Troy Mott
<b>Managing Editor</b>	Sandra Schroeder
<b>Senior Project Editor</b>	Tracey Croom
<b>Editorial Production</b>	Backstop Media
<b>Copy Editor</b>	Liv Bainbridge
<b>Indexer</b>	Julie Grady
<b>Proofreader</b>	Christina Rudloff
<b>Technical Editor</b>	Jason Haley
<b>Cover Designer</b>	Twist Creative, Seattle

# Contents at a glance

	<i>Introduction</i>	<i>xiii</i>
	<i>Preparing for the exam</i>	<i>xvii</i>
<b>CHAPTER 1</b>	<b>Create and manage virtual machines</b>	<b>1</b>
<b>CHAPTER 2</b>	<b>Design and implement a storage and data strategy</b>	<b>109</b>
<b>CHAPTER 3</b>	<b>Manage identity, application and network services</b>	<b>191</b>
<b>CHAPTER 4</b>	<b>Design and implement Azure PaaS compute and web and mobile services</b>	<b>281</b>
	<i>Index</i>	<i>413</i>



*This page intentionally left blank*

# Contents

<b>Introduction</b>	<b>xiii</b>
Organization of this book .....	.xiii
Microsoft certifications .....	.xiv
Acknowledgments .....	.xiv
Microsoft Virtual Academy .....	.xv
Quick access to online references .....	.xv
Errata, updates, & book support .....	.xv
We want to hear from you .....	.xvi
Stay in touch .....	.xvi
<i>Preparing for the exam</i>	<i>xvii</i>

<b>Chapter 1</b>	<b>Create and manage virtual machines</b>	<b>1</b>
Skill 1.1: Deploy workloads on Azure ARM virtual machines .....		1
Identify supported workloads		2
Create a Windows Server VM		3
Create a Linux VM		6
Create a SQL Server VM		7
Skill 1.2: Perform configuration management .....		7
Automate configuration management by using PowerShell Desired State Configuration (DSC) and the VM Agent (using custom script extensions)		7
Configure VMs with Custom Script Extension		8
Use PowerShell DSC		11
Configure VMs with DSC		13
Enable remote debugging		16

Skill 1.3: Scale ARM VMs . . . . .	16
Scale up and scale down VM sizes . . . . .	17
Deploy ARM VM Scale Sets (VMSS) . . . . .	18
Configure Autoscale . . . . .	25
Skill 1.4: Design and implement ARM VM storage . . . . .	29
Plan for storage capacity . . . . .	30
Configure storage pools . . . . .	32
Configure disk caching . . . . .	39
Configure geo-replication . . . . .	41
Configure shared storage using Azure File storage . . . . .	41
Implement ARM VMs with Standard and Premium Storage . . . . .	45
Implement Azure Disk Encryption for Windows and Linux ARM VMs . . . . .	46
Skill 1.5: Monitor VMs. . . . .	47
Configure monitoring and diagnostics for a new VM . . . . .	49
Configure monitoring and diagnostics for an existing VM . . . . .	50
Configure alerts . . . . .	55
Monitor metrics . . . . .	55
Skill 1.6: Manage ARM VM Availability . . . . .	57
Configure availability sets . . . . .	58
Combine the Load Balancer with availability sets . . . . .	60
Skill 1.7: Design and implement DevTest Labs . . . . .	67
Create a lab . . . . .	67
Add a VM to a lab . . . . .	70
Create and manage custom images and formulas . . . . .	72
Configure a lab to include policies and procedures . . . . .	83
Configure cost management . . . . .	92
Secure access to labs . . . . .	95
Use environments in a lab . . . . .	100
Thought experiment. . . . .	105
Thought experiment answer. . . . .	105
Chapter summary . . . . .	105

## Chapter 2 Design and implement a storage and data strategy 109

Skill 2.1: Implement Azure Storage blobs and Azure files . . . . .	109
Azure Storage blobs	110
Create a blob storage account	110
Read and change data	112
Set metadata on a container	113
Setting user-defined metadata	113
Reading user-defined metadata	114
Store data using block and page blobs	115
Stream data using blobs	115
Access blobs securely	115
Implement Async blob copy	116
Configure a Content Delivery Network with Azure Blob Storage	116
Design blob hierarchies	117
Configure custom domains	118
Scale blob storage	119
Azure files	119
Implement blob leasing	119
Create connections to files from on-premises or cloudbased Windows or, Linux machines	120
Shard large datasets	122
Skill 2.2: Implement Azure Storage tables, queues, and Azure Cosmos DB Table API . . . . .	122
Azure Table Storage	122
Using basic CRUD operations	123
Querying using ODATA	127
Designing, managing, and scaling table partitions	128
Azure Storage Queues	128
Adding messages to a queue	128
Processing messages	129
Retrieving a batch of messages	130
Scaling queues	130
Choose between Azure Storage Tables and Azure Cosmos DB Table API	131

Skill 2.3: Manage access and monitor storage. . . . .	132
Generate shared access signatures	132
Create stored access policies	135
Regenerate storage account keys	135
Configure and use Cross-Origin Resource Sharing	136
Analyze logs	141
Skill 2.4: Implement Azure SQL databases . . . . .	144
Choosing the appropriate database tier and performance level	144
Configuring and performing point in time recovery	147
Enabling geo-replication	149
Creating an offline secondary database	150
Creating an online secondary database	150
Creating an online secondary database	151
Import and export schema and data	151
Scale Azure SQL databases	155
Managed elastic pools, including DTUs and eDTUs	157
Implement Azure SQL Data Sync	159
Implement graph database functionality in Azure SQL Database	160
Skill 2.5: Implement Azure Cosmos DB DocumentDB . . . . .	162
Choose the Cosmos DB API surface	163
Create Cosmos DB API Database and Collections	164
Query documents	167
Run Cosmos DB queries	167
Create Graph API databases	168
Execute GraphDB queries	168
Implement MongoDB database	169
Manage scaling of Cosmos DB, including managing partitioning, consistency, and RUs	169
Manage multiple regions	171
Implement stored procedures	173
Access Cosmos DB from REST interface	174
Manage Cosmos DB security	174

Skill 2.6: Implement Redis caching . . . . .	177
Choose a cache tier	177
Implement data persistence	178
Implement security and network isolation	179
Tune cluster performance	180
Integrate Redis caching with ASP.NET session and cache providers	181
Skill 2.7: Implement Azure Search . . . . .	182
Create a service index	182
Add data	183
Search an index	185
Handle Search results	186
Thought experiment. . . . .	186
Thought experiment answers . . . . .	187
Chapter summary . . . . .	188

## **Chapter 3   Manage identity, application and network services   191**

Skill 3.1: Integrate an app with Azure AD . . . . .	191
Preparing to integrate an app with Azure AD	192
Develop apps that use WS-Federation, SAML-P, OpenID Connect and OAuth endpoints	198
Query the directory using Microsoft Graph API, MFA and MFA API	207
Skill 3.2: Develop apps that use Azure AD B2C and Azure AD B2B . . . . .	216
Design and implement apps that leverage social identity provider authentication	217
Leverage Azure AD B2B to design and implement applications that support partner-managed identities and enforce multi-factor authentication	225
Skill 3.3: Manage Secrets using Azure Key Vault . . . . .	225
Configure Azure Key Vault	226
Manage access, including tenants	228
Implement HSM protected keys	232
Implement logging	233
Implement key rotation	235



Skill 4.4: Develop Azure App Service Mobile Apps . . . . .	343
Create a mobile app . . . . .	343
Add authentication to a mobile app . . . . .	346
Add offline sync to a mobile app . . . . .	348
Add push notifications to a mobile app . . . . .	350
Skill 4.5: Implement API Management . . . . .	351
Create managed APIs . . . . .	352
Configure API Management policies . . . . .	356
Protect APIs with rate limits . . . . .	358
Add caching to improve performance . . . . .	360
Monitor APIs . . . . .	362
Customize the developer portal . . . . .	363
Skill 4.6: Implement Azure Functions and WebJobs . . . . .	366
Create Azure Functions . . . . .	367
Implement a Webhook function . . . . .	369
Create an event processing function . . . . .	371
Implement an Azure-connected function . . . . .	372
Integrate a function with storage . . . . .	374
Design and implement a custom binding . . . . .	376
Debug a Function . . . . .	377
Implement and configure proxies . . . . .	377
Integrate with App Service Plan . . . . .	379
Skill 4.7: Design and Implement Azure Service Fabric apps . . . . .	379
Create a Service Fabric application . . . . .	380
Add a web front end to a Service Fabric application . . . . .	383
Build an Actors-based service . . . . .	387
Monitor and diagnose services . . . . .	388
Deploy an application to a container . . . . .	388
Migrate apps from cloud services . . . . .	390
Scale a Service Fabric app . . . . .	390
Create, secure, upgrade, and scale Service Fabric Cluster in Azure . . . . .	391



Skill 4.8: Design and implement third-party Platform as a Service (PaaS) . . . . .	392
Implement Cloud Foundry . . . . .	392
Implement OpenShift . . . . .	394
Provision applications by using Azure Quickstart Templates . . . . .	397
Build applications that leverage Azure Marketplace solutions and services . . . . .	398
Skill 4.9: Design and implement DevOps . . . . .	399
Instrument an application with telemetry . . . . .	400
Discover application performance issues by using Application Insights . . . . .	401
Deploy Visual Studio Team Services with continuous integration (CI) and continuous development (CD) . . . . .	403
Deploy CI/CD with third-party platform tools (Jenkins, GitHub, Chef, Puppet, TeamCity) . . . . .	408
Thought experiment. . . . .	410
Thought experiment answers . . . . .	411
Chapter summary . . . . .	412
 <i>Index</i> . . . . .	 413

# Introduction

---

The 70-532 exam focuses the skills necessary to develop software on the Microsoft Azure Cloud. It covers Infrastructure-as-a-Service (IaaS) offerings like Azure VMs and Platform-as-a-Service (PaaS) offerings like Azure Storage, Azure CosmosDB, Azure Active Directory, Azure Service Bus, Azure Event Hub, Azure App Services, Azure Service Fabric, Azure Functions and other relevant marketplace applications. This book will help get started with these and other features of Azure so that you can begin developing and deploying Azure applications.

This book is geared toward cloud application developers who focus on Azure as the target host environment. It covers choosing from Azure compute options for IaaS and PaaS, incorporating storage and data platforms. It will help you choose when to use features such as Web Apps, API Apps, API Management, Logic Apps and Mobile Apps. It will explain your data storage options between Azure CosmosDB, Azure Redis Cache, Azure Search, and Azure SQL Database. It also covers how to secure applications with Azure Active Directory using B2C and B2B features for single sign-on based on OpenID Connect, OAuth2 and SAML-P protocols, and how to use Azure Vault to protect secrets.

This book covers every major topic area found on the exam, but it does not cover every exam question. Only the Microsoft exam team has access to the exam questions, and Microsoft regularly adds new questions to the exam, making it impossible to cover specific questions. You should consider this book a supplement to your relevant real-world experience and other study materials. If you encounter a topic in this book that you do not feel completely comfortable with, use the “Need more review?” links you’ll find in the text to find more information and take the time to research and study the topic. Great information is available on MSDN, TechNet, and in blogs and forums.

## Organization of this book

---

This book is organized by the “Skills measured” list published for the exam. The “Skills measured” list is available for each exam on the Microsoft Learning website: <https://aka.ms/examlist>. Each chapter in this book corresponds to a major topic area in the list, and the technical tasks in each topic area determine a chapter’s organization. If an exam covers six major topic areas, for example, the book will contain six chapters.

## Microsoft certifications

---

Microsoft certifications distinguish you by proving your command of a broad set of skills and experience with current Microsoft products and technologies. The exams and corresponding certifications are developed to validate your mastery of critical competencies as you design and develop, or implement and support, solutions with Microsoft products and technologies both on-premises and in the cloud. Certification brings a variety of benefits to the individual and to employers and organizations.

### **MORE INFO** ALL MICROSOFT CERTIFICATIONS

For information about Microsoft certifications, including a full list of available certifications, go to <https://www.microsoft.com/learning>.

## Acknowledgments

---

**Zoiner Tejada** A book of this scope takes a village, and I'm honored to have received the support of one in making this second edition happen. My deepest thanks to the team at Solliance who helped make this possible: my co-authors Michele Leroux Bustamante and Ike Ellis and the hidden heroes, and Joel Hulen and Kyle Bunting helped us with research and coverage on critical topics as the scope of the book grew with the fast pace of Azure. Laura Norman, our editor, thank you for helping us navigate the path to completion with structure and compassion. To my wife Ashley Tejada, my eternal thanks for supporting me in this effort, the little things count and they don't go unnoticed.

**Michele Leroux Bustamante** I want to thank Joel Hulen, Virgilio Esteves and Khaled Hikmat – who have been part of key Solliance projects in Azure, including this book – and this work and experience reflects in the guidance shared in the book. Thank you for being part of this journey! Thank you also to, Laura Norman, our editor – who was very supporting during challenging deadlines. A level head keeps us all sane. To my husband and son – thank you for tolerating the writing schedule – again. I owe you - again. Much love.

**Ike Ellis** First and foremost, I'd like to thank my wife, Margo Sloan, for her support in taking care of all the necessities of life while I wrote. Our editor, Laura Norman, had her hands full in wrangling three busy co-authors, and I'm very grateful for her diligence. I'm very grateful to my co-authors, Zoiner and Michele. It's a joy to work with them on all of our combined projects.

## Microsoft Virtual Academy

---

Build your knowledge of Microsoft technologies with free expert-led online training from Microsoft Virtual Academy (MVA). MVA offers a comprehensive library of videos, live events, and more to help you learn the latest technologies and prepare for certification exams. You'll find what you need here:

*<https://www.microsoftvirtualacademy.com>*

## Quick access to online references

---

Throughout this book are addresses to webpages that the author has recommended you visit for more information. Some of these addresses (also known as URLs) can be painstaking to type into a web browser, so we've compiled all of them into a single list that readers of the print edition can refer to while they read.

Download the list at *<https://aka.ms/examref5322E/downloads>*.

The URLs are organized by chapter and heading. Every time you come across a URL in the book, find the hyperlink in the list to go directly to the webpage.

## Errata, updates, & book support

---

We've made every effort to ensure the accuracy of this book and its companion content. You can access updates to this book—in the form of a list of submitted errata and their related corrections—at:

*<https://aka.ms/examref5322E/errata>*

If you discover an error that is not already listed, please submit it to us at the same page.

If you need additional support, email Microsoft Press Book Support at *[msspinput@microsoft.com](mailto:msspinput@microsoft.com)*.

Please note that product support for Microsoft software and hardware is not offered through the previous addresses. For help with Microsoft software or hardware, go to *<https://support.microsoft.com>*.

## We want to hear from you

---

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

*<https://aka.ms/tellpress>*

We know you're busy, so we've kept it short with just a few questions. Your answers go directly to the editors at Microsoft Press. (No personal information will be requested.) Thanks in advance for your input!

## Stay in touch

---

Let's keep the conversation going! We're on Twitter: *<http://twitter.com/MicrosoftPress>*.

## Preparing for the exam

Microsoft certification exams are a great way to build your resume and let the world know about your level of expertise. Certification exams validate your on-the-job experience and product knowledge. Although there is no substitute for on-the-job experience, preparation through study and hands-on practice can help you prepare for the exam. We recommend that you augment your exam preparation plan by using a combination of available study materials and courses. For example, you might use the Exam ref and another study guide for your “at home” preparation, and take a Microsoft Official Curriculum course for the classroom experience. Choose the combination that you think works best for you.

Note that this Exam Ref is based on publicly available information about the exam and the author’s experience. To safeguard the integrity of the exam, authors do not have access to the live exam.

*This page intentionally left blank*

# Design and implement Azure PaaS compute and web and mobile services

The Azure platform provides a rich set of Platform-as-a-Service (PaaS) capabilities for hosting web applications and services. The platform approach provides more than just a host for running your application logic; it also includes robust mechanisms for managing all aspects of your web application lifecycle, from configuring continuous and staged deployments to managing runtime configuration, monitoring health and diagnostic data, and of course, helping with scale and resilience. Azure App Services includes a number of features to manage web applications and services including Web Apps, Logic Apps, Mobile Apps and API Apps. API Management provides additional features with first class integration to APIs hosted in Azure. Azure Functions and Azure Service Fabric enable modern microservices architectures for your solutions, in addition to several third-party platforms that can be provisioned via Azure Quickstart Templates. These key features are of prime importance to the modern web application, and this chapter explores how to leverage them.

### Skills in this chapter:

- Skill 4.1: Design Azure App Service Web Apps
- Skill 4.2: Design Azure App Service API Apps
- Skill 4.3: Develop Azure App Service Logic Apps
- Skill 4.4: Develop Azure App Service Mobile Apps
- Skill 4.5: Implement API Management
- Skill 4.6: Implement Azure Functions and WebJobs
- Skill 4.7: Design and implement Azure Service Fabric Apps
- Skill 4.8: Design and implement third-party Platform as a Service (PaaS)
- Skill 4.9: Design and implement DevOps



## Skill 4.1: Design Azure App Service Web Apps

---

Azure App Service Web Apps (or, just Web Apps) provides a managed service for hosting your web applications and APIs with infrastructure services such as security, load balancing, and scaling provided as part of the service. In addition, Web Apps has an integrated DevOps experience from code repositories and from Docker image repositories. You pay for compute resources according to your App Service Plan and scale settings. This section covers key considerations for designing and deploying your applications as Web Apps.

### This skill covers how to:

- Define and manage App Service plans
- Configure Web App settings
- Configure Web App certificates and custom domains
- Manage Web Apps by using the API, Azure PowerShell, and Xplat-CLI
- Implement diagnostics, monitoring, and analytics
- Design and configure Web Apps for scale and resilience

## Define and manage App Service plans

An App Service plan defines the supported feature set and capacity of a group of virtual machine resources that are hosting one or more web apps, logic apps, mobile apps, or API apps (this section discusses web apps specifically, and the other resources are covered in later sections in this chapter).

Each App Service plan is configured with a pricing tier (for example, Free, Shared, Basic, and Standard), and each tier describes its own set of capabilities and cost. An App Service plan is unique to the region, resource group, and subscription. In other words, two web apps can participate in the same App Service plan only when they are created in the same subscription, resource group, and region (with the same pricing tier requirements).

This section describes how to create a new App Service plan without creating a web app, and how to create a new App Service plan while creating a web app. It also reviews some of the settings that can be useful for managing the App Service plan.

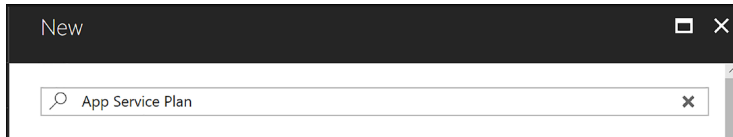
### **MORE INFO** APP SERVICES OVERVIEW

For an overview of App Services and Web App development see <https://docs.microsoft.com/en-us/azure/app-service/>.

## Creating a new App Service plan

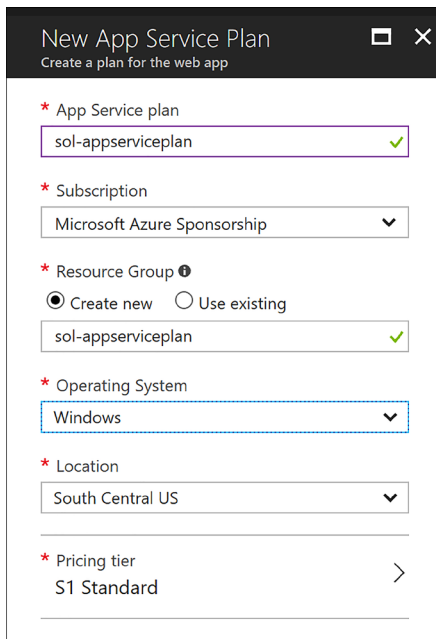
To create a new App Service plan in the portal, complete the following steps:

1. Navigate to the portal accessed via <https://portal.azure.com>.
2. Select New on the command bar.
3. Within the Marketplace (Figure 4-1) search text box, type **App Service Plan** and press Enter.



**FIGURE 4-1** The Marketplace search for App Service Plan.

4. Select App Service Plan from the results.
5. On the App Service Plan blade, select Create.
6. On the New App Service Plan blade (Figure 4-2), provide a name for your App Service plan, choose the subscription, resource group, operating system (Windows or Linux), and location into which you want to deploy. You should also confirm and select the desired pricing tier.

A screenshot of the 'New App Service Plan' blade in the Azure portal. The title bar says 'New App Service Plan' with a subtitle 'Create a plan for the web app'. The form contains several fields: 'App Service plan' with the value 'sol-appserviceplan' and a green checkmark; 'Subscription' with a dropdown menu showing 'Microsoft Azure Sponsorship'; 'Resource Group' with radio buttons for 'Create new' (selected) and 'Use existing', and a dropdown menu showing 'sol-appserviceplan' with a green checkmark; 'Operating System' with a dropdown menu showing 'Windows'; 'Location' with a dropdown menu showing 'South Central US'; and 'Pricing tier' with a dropdown menu showing 'S1 Standard' and a right arrow. Each field is preceded by a red asterisk indicating it's required.

**FIGURE 4-2** The settings for a new App Service Plan

7. Click Create to create the new App Service plan.

Following the creation of the new App Service plan, you can create a new web app and associate this with the previously created App Service plan. Or, as discussed in the next section, you can create a new App Service plan as you create a new web app.

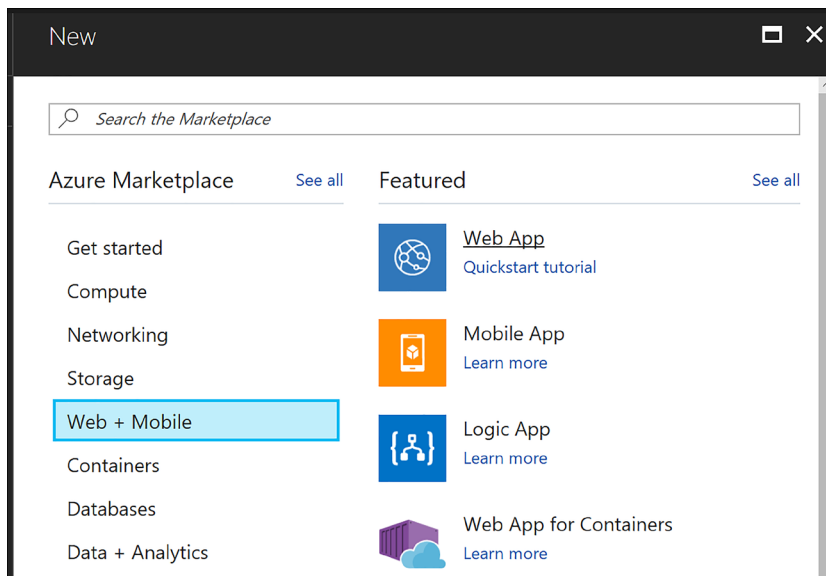
#### **MORE INFO APP SERVICE PRICING TIERS**

App Service plan pricing tiers range from Free, Shared, Basic, Standard, Premium, and Isolated tiers. It is important to understand the features offered by each tier related to custom domains, certificates, scale, deployment slots, and more. For more information see <https://azure.microsoft.com/en-us/pricing/details/app-service>.

## Creating a new Web App and App Service plan

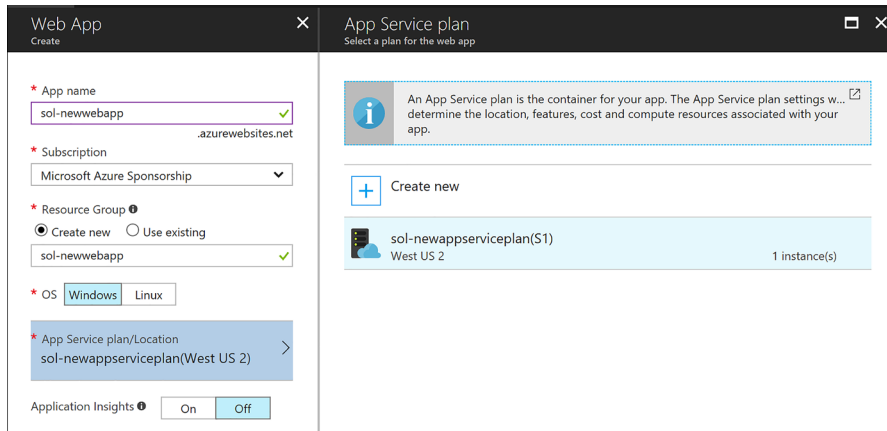
To create a new Web App and a new App Service plan in the portal, complete the following steps:

1. Navigate to the portal accessed via <https://portal.azure.com>.
2. Select New on the command bar.
3. Within the Marketplace list (Figure 4-3), select the Web + Mobile option.



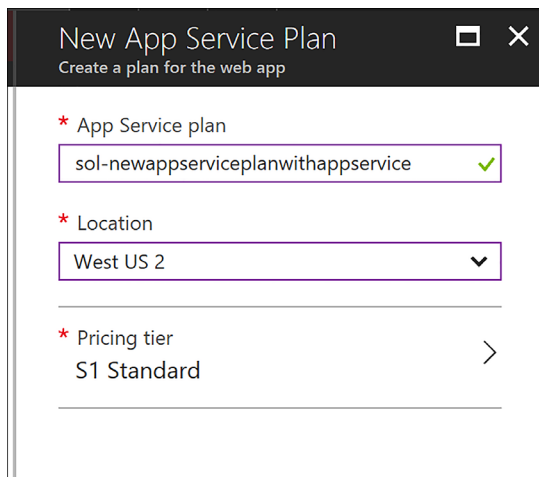
**FIGURE 4-3** The Marketplace list for Web + Mobile

4. On the Web + Mobile blade, select Web App.
5. On the Web App blade (Figure 4-4), provide an app name, choose the subscription, resource group, operating system (Windows or Linux), and choose a setting for Application Insights. You also select the App Service plan into which you want to deploy.



**FIGURE 4-4** The selections for a new App Service.

6. When you click the App Service plan selection, you can choose an existing App Service plan, or create a new App Service plan. To create a new App Service plan, click Create New from the App Service Plan blade.
7. From the New App Service Plan blade (Figure 4-5), choose a name for the App Service plan, select a location, and select a pricing tier. Click OK and the new App Service plan is created with these settings.



**FIGURE 4-5** Options for a new App Service Plan.

8. From the Web App blade, click Create to create the web app and associate it with the new App Service plan.

# Review App Service plan settings

Once you've created a new App Service plan, you can select the App Service plan in the portal and manage relevant settings including managing web apps and adjusting scale.

To manage an App Service plan, complete the following steps:

1. Navigate to the portal accessed via *https://portal.azure.com*.
2. Select More Services on the command bar.
3. In the filter text box, type **App Service Plans**, and select App Service Plans (Figure 4-6).

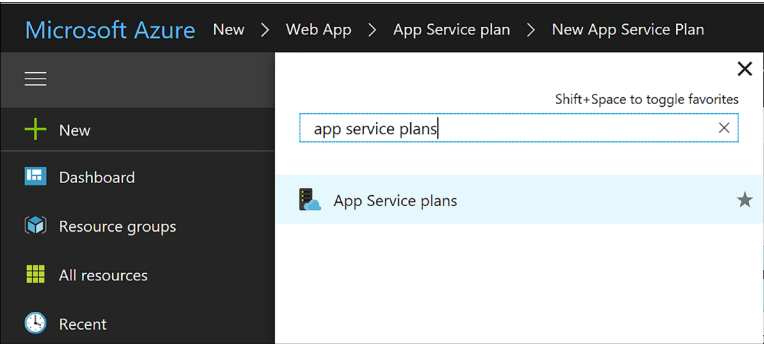


FIGURE 4-6 Search results for App Service plans

4. Review the list of App Service plans (Figure 4-7). Note the number of apps deployed to each is shown in the list. You can also see the pricing tiers. Select an App Service plan from the list to navigate to the App Service Plan blade.

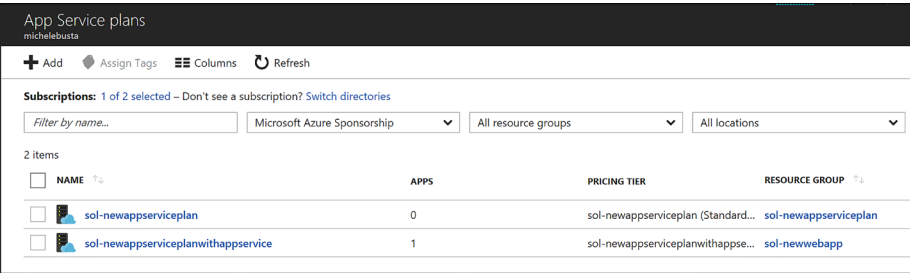


FIGURE 4-7 List of App Service plans

5. From the left navigation pane, select Apps to view the apps that are deployed to the App Service plan (Figure 4-8). You can select from the list of apps to navigate to the app blade and manage its settings.

NAME	TYPE	RESOURCE GROUP	STATUS
SOL-NEWWEBAPP			
sol-newwebapp	app	sol-newwebapp	Running

**FIGURE 4-8** List of apps deployed to the App Service plan.

6. From the left navigation pane, select Scale Up to choose a new pricing tier for the App Service plan.
7. From the left navigation pane, select Scale Out to increase or decrease the number of instances of the App Service plan, or to configure Autoscale settings.

## Configure Web App settings

Azure Web Apps provide a comprehensive collection of settings that you can adjust to establish the environment in which your web application runs, as well as tools to define and manage the values of settings used by your web application code. You can configure the following groups of settings for your applications:

- Application type and library versions
- Load balancing
- Slot management
- Debugging
- App settings and connection strings
- IIS related settings

To manage Web App settings follow these steps:

1. Navigate to the blade of your web app in the portal accessed via <https://portal.azure.com>.
2. Select the Application settings tab from the left navigation pane. The setting blade appears to the right.
3. Choose from the general settings required for the application:
  - A. Choose the required language support from .NET Framework, PHP, Java, or Python, and their associated versions.
  - B. Choose between 32bit and 64bit runtime execution.



- C. Choose web sockets if you are building a web application that leverages this feature from the browser.
- D. Choose Always On if you do not want the web application to be unloaded when idle. This reduces the load time required for the next request and is a required setting for web jobs to run effectively.
- E. Choose the type of managed pipeline for IIS. Integrated is the more modern pipeline and Classic would only be used for legacy applications (Figure 4-9).

General settings

.NET Framework version ⓘ v4.7 ▼

PHP version ⓘ Off ▼

Java version ⓘ Off ▼

 App Service supports installing newer versions of Python. [Click here to learn more.](#) 

Python version ⓘ Off ▼

Platform ⓘ 32-bit 64-bit

Web sockets ⓘ Off On

Always On ⓘ Off On

Managed Pipeline Version Integrated Classic

**FIGURE 4-9** General settings section for application settings

4. Choose your setting for ARR affinity (Figure 4-10). If you choose to enable ARR affinity your users will be tied to a particular host machine (creating a sticky session) for the duration of their session. If you disable this, your application will not create a sticky session and your application is expected to support load balancing between machines within a session.

ARR Affinity Off On

**FIGURE 4-10** ARR affinity settings

5. When you first create your web app, the auto swap settings are not available to configure. You must first create a new slot, and from the slot you may configure auto swap to another slot (Figure 4-11).

Auto Swap

Off On

Auto Swap Slot

**FIGURE 4-11** Auto Swap settings

6. Enable remote debugging (Figure 4-12) if you run into situations where deployed applications are not functioning as expected. You can enable remote debugging for Visual Studio versions 2012, 2013, 2015, and 2017.

Debugging

Remote debugging

Off On

Remote Visual Studio version

2012 2013 2015 2017

**FIGURE 4-12** Remote debugging settings for the web app

7. Configure the app settings required for your application. These app settings (Figure 4-13) override any settings matching the same name from your application.

App settings

Key	Value	Slot setting	
WEBSITE_NODE_DEFAULT_VERSION	6.9.1	<input type="checkbox"/>	...
ACCOUNT_API_PATH	https://api.contoso.com/Account	<input type="checkbox"/>	...
Key	Value	<input type="checkbox"/>	...

**FIGURE 4-13** Application settings

8. Configure any connection strings for your application (Figure 4-14). These connection string settings override any settings matching the same key name from your application configuration. For connection strings, once you create the settings, save, and later return to the application settings blade; those settings are hidden unless you select it to show the value again.

Connection strings

The connection string values are hidden [Show connection string values](#)

Name	Value	Slot setting	
DEFAULT_CONNECTION	< Hidden for Security >	<input type="checkbox"/>	...
Name	Value	<input type="checkbox"/>	...

**FIGURE 4-14** Connection string settings



9. Configure IIS settings related to default documents, handlers, and virtual applications and directories required for your application (Figure 4-15). This allows you to control these IIS features related to your application.

The screenshot shows the IIS Manager console with three main sections:

- Default documents:** A list of default document names with a three-dot menu icon to the right of each. The list includes: Default.htm, Default.html, Default.asp, index.htm, index.html, liststart.htm, default.aspx, index.php, and hostingstart.html. Below the list is an empty text box with a three-dot menu icon.
- Handler mappings:** A section titled "No results" with three input fields: "Extension", "Processor path", and "Additional arguments", each followed by a three-dot menu icon.
- Virtual applications and directories:** A table with two rows. The first row shows a path "/" with "site\wwwroot" and a checked "Application" checkbox. The second row shows a "Virtual directory" input field, a "Physical path relative to site root" input field, and an unchecked "Application" checkbox.

**FIGURE 4-15** IIS settings

#### **NOTE** ACCESS TO APP SETTINGS

App settings are represented as name-value pairs made available to your web application when it starts. The mechanism you use to access these values depends on the web platform in which your web application is programmed. If your application is built using ASP.NET 4.6, you access the values of app settings just as you would access the AppSettings values stored in web.config. If your web application is built using ASP.NET Core, you access the values as you would in your appsettings.json file. If your web application is built using another supported web platform, such as Node.js, PHP, Python, or Java, the app settings are presented to your application as environment variables.

#### **NOTE** ACCESSING CONNECTION STRINGS

Like app settings, connection strings represent name-value pairs, but they are used specifically for settings that define the connection string to a linked resource (typically a database) such as a SQL database, a SQL server, MySQL, or some other custom resource. Connection strings are given special treatment within the portal, beyond that offered to app settings, in that you can specify a type for the connection string to identify it as a SQL server, MySQL, a SQL database, or a custom connection string. Additionally, the values for connection strings are not displayed by default, requiring an additional effort to display the values so that their sensitive data is not displayed or editable until specifically requested by the portal user.

## Configure Web App certificates and custom domains

When you first create your web app, it is accessible through the subdomain you specified in the web app creation process, where it takes the form <yourwebappname>.azurewebsites.net. To map to a more user-friendly domain name (such as contoso.com), you must set up a custom domain name.

If your website will use HTTPS to secure communication between it and the browser using Transport Layer Security (TLS), more commonly (but less accurately) referred to in the industry as Secure Socket Layer (SSL), you need to utilize an SSL certificate. With Azure Web Apps, you can use an SSL certificate with your web app in one of two ways:

- You can use the “built-in” wildcard SSL certificate that is associated with the \*.azurewebsites.net domain.
- More commonly you use a certificate you purchase for your custom domain from a third-party certificate authority.

### NOTE

There are multiple types of SSL certificates, but the one you choose primarily depends on the number of different custom domains (or subdomains) that the certificate secures. Some certificates apply to only a single fully qualified domain name (sometimes referred to as *basic certs*), some certificates apply to a list of fully qualified domain names (also called *subjectAltName* or *UC certs*), and other certificates apply across an unlimited number of subdomains for a given domain name (usually referred to as *wildcard certs*).

## Mapping custom domain names

Web Apps support mapping to a custom domain that you purchase from a third-party registrar either by mapping the custom domain name to the virtual IP address of your website or by mapping it to the <yourwebappname>.azurewebsites.net address of your website. This mapping is captured in domain name system (DNS) records that are maintained by your domain registrar. Two types of DNS records effectively express this purpose:

- A records (or, address records) map your domain name to the IP address of your website.
- CNAME records (or, alias records) map a subdomain of your custom domain name to the canonical name of your website, expressed as <yourwebappname>.azurewebsites.net.

Table 4-1 shows some common scenarios along with the type of record, the typical record name, and an example value based on the requirements of the mapping.

**TABLE 4-1** Mapping domain name requirements to DNS record types, names, and values

Requirement	Type of Record	Record Name	Record Value
contoso.com should map to my web app IP address	A	@	138.91.240.81 IP address
contoso.com and all subdomains demo.contoso.com and www.contoso.com should map to my web app IP address	A	*	138.91.240.81 IP address
www.contoso.com should map to my web app IP address	A	www	138.91.240.81 IP address
www.contoso.com should map to my web app canonical name in Azure	CNAME	www	contoso.azurewebsites.net Canonical name in Azure

Note that whereas A records enable you to map the root of the domain (like *contoso.com*) and provide a wildcard mapping for all subdomains below the root (like *www.contoso.com* and *demo.contoso.com*), CNAME records enable you to map only subdomains (like the *www* in *www.contoso.com*).

## Configuring a custom domain

To configure a custom domain, you need access to your domain name registrar setup for the domain while also editing configuration for your web app in the Azure portal.

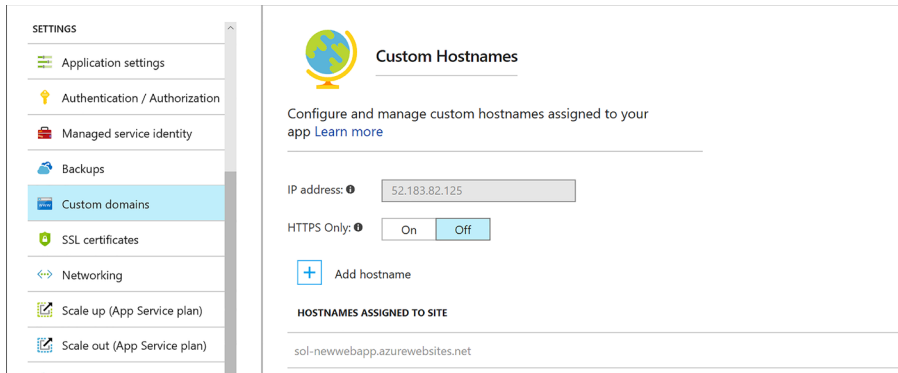


### EXAM TIP

Use of a custom domain name is not supported by the Free App Service plan pricing tier. All other pricing tiers including Shared, Basic, Standard, and Premium support custom domains.

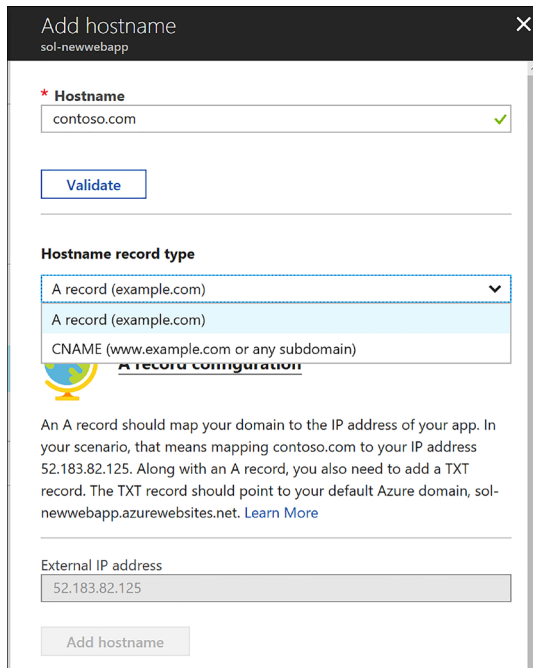
These are the high-level steps for creating a custom domain name for your web app:

1. Navigate to the blade of your web app in the portal accessed via <https://portal.azure.com>.
2. Ensure your web app uses an App Service plan that supports custom domains.
3. Click Custom Domains from the left navigation pane.
4. On the Custom Domains blade (Figure 4-16) note the external IP address of your web app.



**FIGURE 4-16** Part of the custom domain blade for the web app

5. Select Add Hostname to open the Add Hostname blade. Enter the hostname and click Validate for the portal to validate the state of the registrar setup with respect to your web app. You can then choose to set up an A record or CNAME record (Figure 4-17).



**FIGURE 4-17** Part of the Add hostname blade

6. To set up an A record, select A Record and follow the instructions provided in the blade. It guides you through the following steps for an A record setup:

- A.** You first add a TXT record at your domain name registrar, pointing to the default Azure domain for your web app, to verify you own the domain name. The new TXT record should point to <yourwebappname>.azurewebsites.net.
  - B.** In addition, you add an A record pointing to the IP address shown in the blade, for your web app.
- 7.** To set up a CNAME record, select CNAME record, and follow the instructions provided in the blade.
  - A.** If using a CNAME record, following the instructions provided by your domain name registrar, add a new CNAME record with the name of the subdomain, and for the value, specify your web app's default Azure domain with <yourwebappname>.azurewebsites.net.
- 8.** Save your DNS changes. Note that it may take some time for the changes to propagate across DNS. In most cases, your changes are visible within minutes, but in some cases, it may take up to 48 hours. You can check the status of your DNS changes by doing a DNS lookup using third-party websites like <http://mxtoolbox.com/DNSLookup.aspx>.
- 9.** After completing the domain name registrar setup, from the Custom Domains blade, click Add Hostname again to configure your custom domain. Enter the domain name and select Validate again. If validation has passed, select Add Hostname to complete the assignment.

#### **IMPORTANT IP ADDRESS CHANGES**

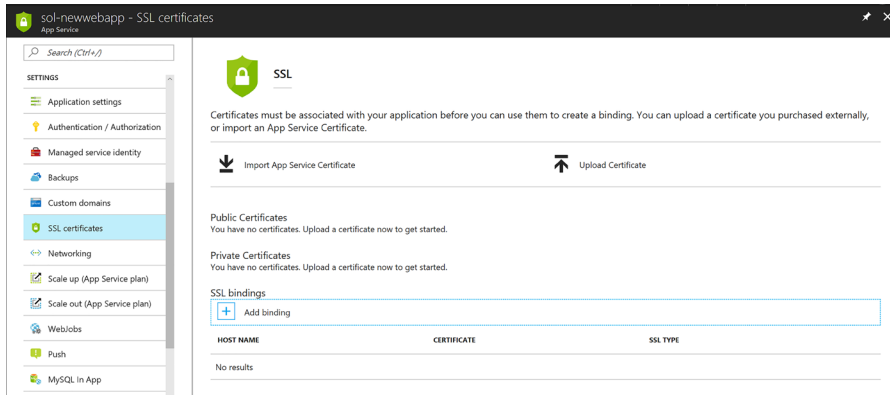
The IP address that you get by following the preceding steps will change if you move your web app to a Free web hosting plan, if you delete and recreate it, or potentially if you subsequently enable SSL with the IP Based type. This can also happen unintentionally if you reach your spending limit and the web app is changed to the Free web hosting plan mode. If the IP address changes and you are using an A record to map your custom domain to your web app, you will need to update the value of the A record to use the new IP address.

## Configuring SSL certificates

To configure SSL certificates for your custom domain, you first need to have access to an SSL certificate that includes your custom domain name, including the CNAME if it is not a wildcard certificate.

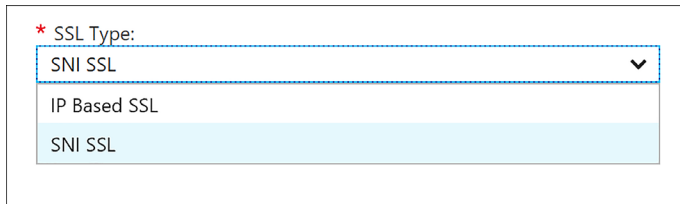
To assign an SSL certificate to your web app, follow these steps:

- 1.** Navigate to the blade of your web app in the portal accessed via <https://portal.azure.com>.
- 2.** Click SSL certificates from the left navigation pane.
- 3.** From the SSL certificates (Figure 4-18) blade you may choose to import an existing app service certificate, or upload a new certificate.



**FIGURE 4-18** SSL certificates blade

4. You can then select Add Binding to set up the correct binding. You can set up bindings that point at your naked domain (contoso.com), or to a particular CNAME (www.contoso.com, demo.contoso.com), so long as the certificate supports it.
5. You can choose between Server Name Indication (SNI) or IP based SSL when you create the binding for your custom domain (Figure 4-19).



**FIGURE 4-19** Part of the Add Binding blade

#### **MORE INFO** SSL CERTIFICATES AND BINDINGS

For more information on purchasing SSL certificates and setting up Web App certificates see <https://docs.microsoft.com/en-us/azure/app-service/web-sites-purchase-ssl-web-site>.

## Manage Web Apps by using the API, Azure PowerShell, and Xplat-CLI

In addition to configuring and managing Web Apps via the Azure portal, programmatic or script-based access is available for much of this functionality and can satisfy many development requirements.

The options for this include the following:

- **Azure Resource Manager (ARM)** Azure Resource Manager provides a consistent management layer for the management tasks you can perform using Azure PowerShell, Azure CLI, Azure portal, REST API, and other development tools. For more information on this see <https://docs.microsoft.com/en-us/azure/azure-resource-manager/>.
- **REST API** The REST API enables you to deploy and manage Azure infrastructure resources using HTTP request and JSON payloads. For more details on this see <https://docs.microsoft.com/en-us/rest/api/resources/>.
- **Azure PowerShell** Azure PowerShell provides cmdlets for interacting with Azure Resource Manager to manage infrastructure resources. The PowerShell modules can be installed to Windows, macOS, or Linux. For additional details see <https://docs.microsoft.com/en-us/powershell/azure/overview>.
- **Azure CLI** Azure CLI (also known as XplatCLI) is a command line experience for managing Azure resources. This is an open source SDK that works on Windows, macOS, and Linux platforms to create, manage, and monitor web apps. For details see <https://docs.microsoft.com/en-us/cli/azure/overview>.

#### **MORE INFO** MANAGING APP SERVICES

See the following links that provide samples for managing App Services using Azure and Azure CLI at: <https://docs.microsoft.com/en-us/azure/app-service/app-service-powershell-samples> and <https://docs.microsoft.com/en-us/azure/app-service/app-service-cli-samples>.

## Implement diagnostics, monitoring, and analytics

Without diagnostics, monitoring, and analytics, you cannot effectively investigate the cause of a failure, nor can you proactively prevent potential problems before your users experience them. Web Apps provide multiple forms of logs, features for monitoring availability and automatically sending email alerts when the availability crosses a threshold, features for monitoring your web app resource usage, and integration with Azure Analytics via Application Insights.



#### **EXAM TIP**

App Services are also governed by quotas depending on the App Service plan you have chosen. Free and Shared apps have CPU, memory, bandwidth, and filesystem quotas; when reached the web app no longer runs until the next cycle, or the App Service plan is changed. Basic, Standard, and Premium App Services are only limited by filesystem quotas based on the SKU size selected for the host.

#### **MORE INFO** QUOTAS

For the latest listing of specific quotas, limits, and features, visit <https://docs.microsoft.com/azure/azure-subscription-service-limits#app-service-limits>.

## Configure diagnostics logs

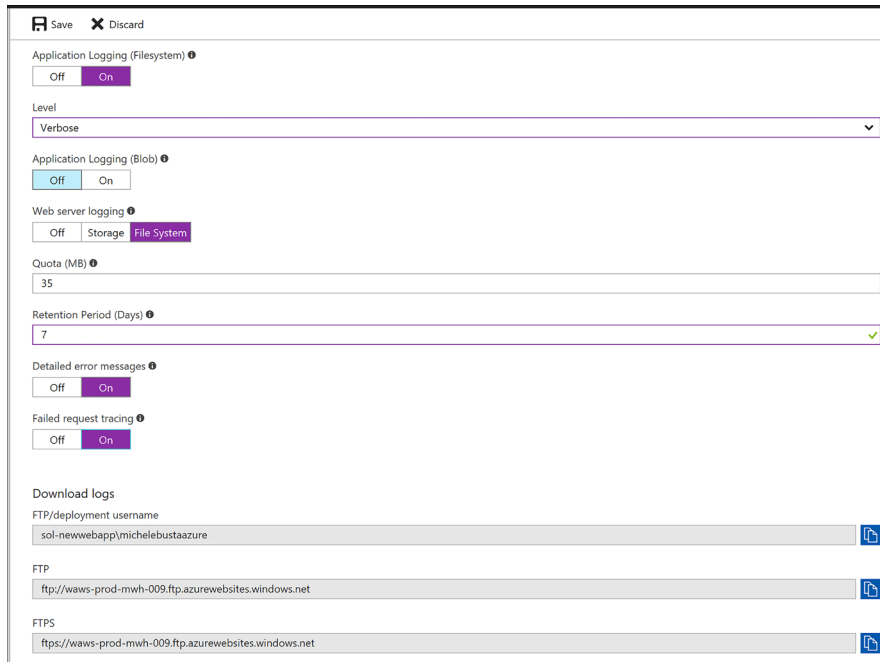
A web app can produce many different types of logs, each focused on presenting a particular source and format of diagnostic data. The following list describes each of these logs:

- **Event Log** The equivalent of the logs typically found in the Windows Event Log on a Windows Server machine, this is a single XML file on the local file system of the web application. In the context of web apps, the Event Log is particularly useful for capturing unhandled exceptions that may have escaped the application's exception handling logic and surfaced to the web server. Only one XML file is created per web app.
- **Web server logs** Web server logs are textual files that create a text entry for each HTTP request to the web app.
- **Detailed error message logs** These HTML files are generated by the web server and log the error messages for failed requests that result in an HTTP status code of 400 or higher. One error message is captured per HTML file.
- **Failed request tracing logs** In addition to the error message (captured by detailed error message logs), the stack trace that led to a failed HTTP request is captured in these XML documents that are presented with an XSL style sheet for in-browser consumption. One failed request trace is captured per XML file.
- **Application diagnostic logs** These text-based trace logs are created by web application code in a manner specific to the platform the application is built in using logging or tracing utilities.

To enable these diagnostic settings from the Azure portal, follow these steps:

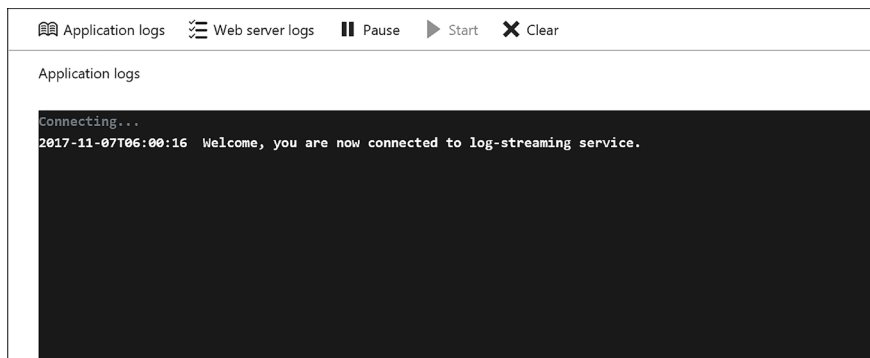
1. Navigate to the blade of your web app in the portal accessed via <https://portal.azure.com>.
2. Select the Diagnostics Logs tab from the left navigation pane. The Diagnostics Logs blade (Figure 4-20) will appear to the right. From this blade you can choose to configure the following:
  - A. Enable application logging to the file system for easy access through the portal.
  - B. Enable storing application logs to blob storage for longer term access.
  - C. Enable Web Server logging to the file system or to blob storage for longer term access.
  - D. Enable logging detailed error messages.
  - E. Enable logging failed request messages.





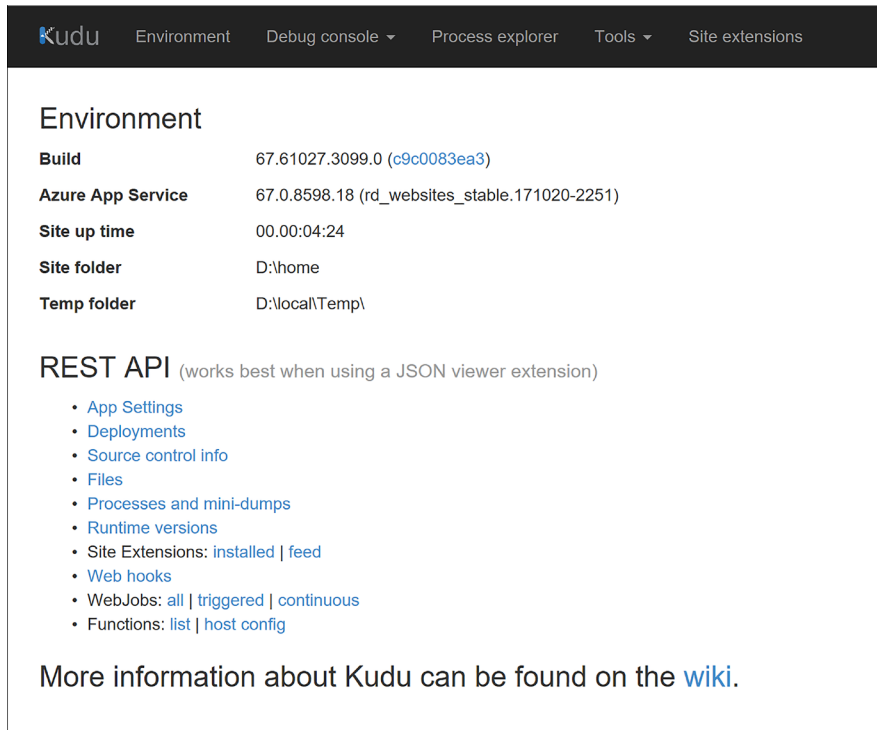
**FIGURE 4-20** The diagnostics logs blade

3. If you enable files system logs for application and Web Server logs, you can view those from the Log Streaming tab (Figure 4-21).



**FIGURE 4-21** The log streaming blade

4. You can access more advanced debugging and diagnostics tools from the Advanced Tools tab (Figure 4-22).



**FIGURE 4-22** The Kudu web site

Table 4-2 describes where to find each type of log when retrieving diagnostic data stored in the web app's local file system. The Log Files folder is physically located at D:\home\LogFiles.

**TABLE 4-2** Locations of the various logs on the web app's local file system

Log Type	Location
Event Log	\LogFiles\eventlog.xml
Web server logs	\LogFiles\http\RawLogs\*.log
Detailed error message logs	\LogFiles\DetailedErrors\ErrorPage#####.htm
Failed request tracing logs	\LogFiles\W3SVC\*.xml
Application diagnostic logs (.NET)	\LogFiles\Application\*.txt
Deployment logs	\LogFiles\Git. This folder contains logs generated by the internal deployment processes used by Azure web apps, as well as logs for Git deployments



#### EXAM TIP

You can retrieve diagnostics logs data by using Visual Studio Server Explorer, the Site Control Management (SCM) website (also known as Kudu), the command line in Windows PowerShell or the xplat-cli, or direct download via FTP to query Table or Blob storage.

## Configure endpoint monitoring

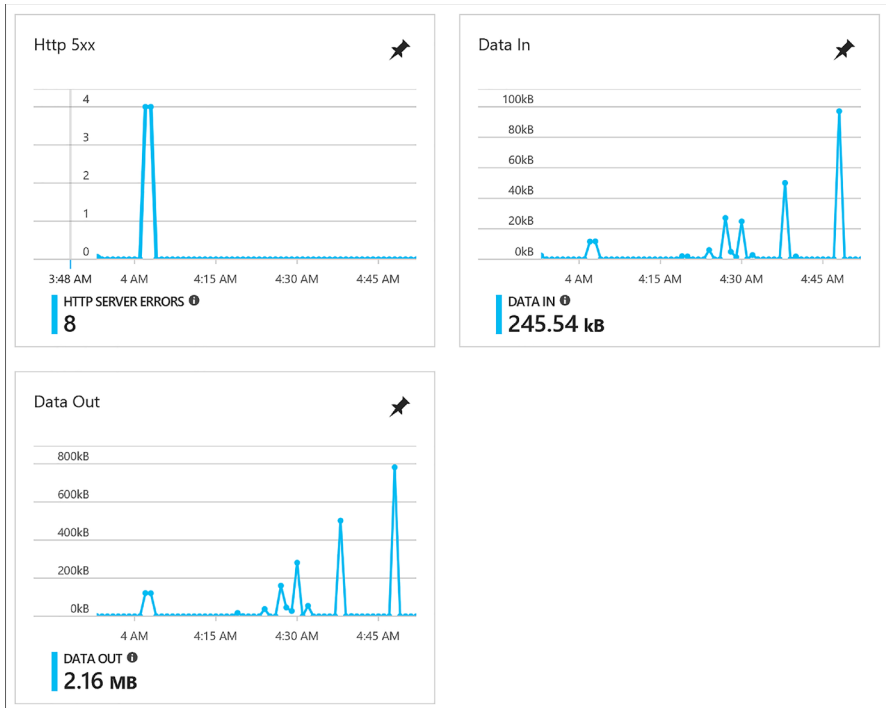
App Services provide features for monitoring your applications directly from the Azure portal. There are many metrics available for monitoring, as listed in Table 4-3.

**TABLE 4-3** List of available metrics that are monitored for your web apps

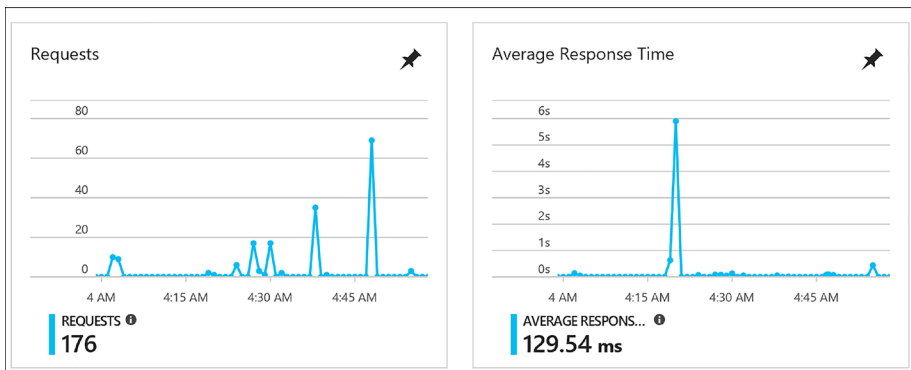
METRIC	DESCRIPTION
Average Response Time	The average time taken for the app to serve requests in ms.
Average memory working set	The average amount of memory in MiBs used by the app.
CPU Time	The amount of CPU in seconds consumed by the app.
Data In	The amount of incoming bandwidth consumed by the app in MiBs.
Data Out	The amount of outgoing bandwidth consumed by the app in MiBs.
Http 2xx	Count of requests resulting in a http status code $\geq 200$ but $< 300$ .
Http 3xx	Count of requests resulting in a http status code $\geq 300$ but $< 400$ .
Http 401	Count of requests resulting in HTTP 401 status code.
Http 403	Count of requests resulting in HTTP 403 status code.
Http 404	Count of requests resulting in HTTP 404 status code.
Http 406	Count of requests resulting in HTTP 406 status code.
Http 4xx	Count of requests resulting in a http status code $\geq 400$ but $< 500$ .
Http Server Errors	Count of requests resulting in a http status code $\geq 500$ but $< 600$ .
Memory working set	Current amount of memory used by the app in MiBs.
Requests	Total number of requests regardless of their resulting HTTP status code.

You can monitor metrics from the portal and customize which metrics should be shown by following these steps:

1. Navigate to the blade of your web app in the portal accessed via <https://portal.azure.com>.
2. Select the Overview tab from the left navigation pane. This pane shows a few default charts for metrics including server errors, data in and out, requests, and average response time (Figure 4-23 and 4-24).



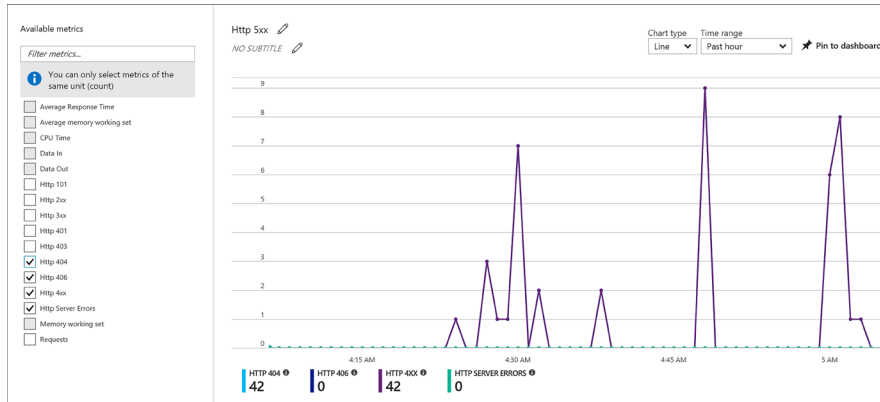
**FIGURE 4-23** Metrics showing http server errors, data in, and data out



**FIGURE 4-24** Metrics showing requests and average response time

3. You can customize the metrics (Figure 4-25) shown by creating new graphs and pinning those to your dashboard.
  - A. Click one of the graphs. You'll be taken to edit the metrics blade for the graph, limited to compatible metrics for the selection.

**B.** Select the metrics to add or remove from the graph.



**FIGURE 4-25** Selecting metrics to show on the graph

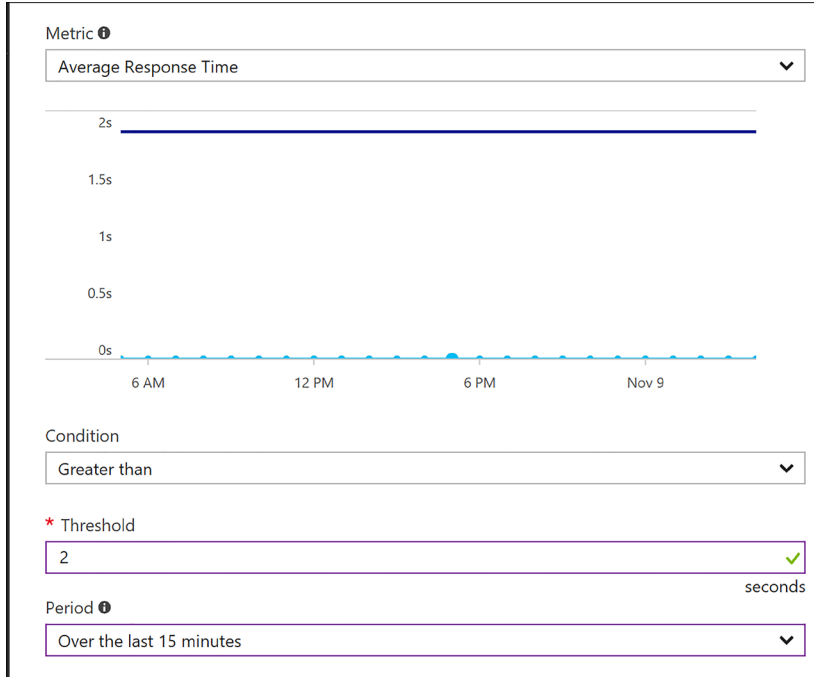
- C.** Save the graph to the dashboard. You can now navigate to your portal dashboard to view the selected metrics without having to navigate to the web app directly. From here you can also edit the graph by selecting it, editing metrics, and saving back to the same pinned graph.
- 4.** You can also add alerts for metrics. From the Metrics blade click Add Metric alert from the command bar at the top of the blade. This takes you to the Add Rule blade (Figure 4-26) where you can configure the alert. To configure an alert for slow requests, as an example, do the following:
- A.** Provide a name for the rule.
  - B.** Optionally change the subscription, resource group, and resource but it will default to the current web app.
  - C.** Choose Metrics for the alert type.

The 'Add rule' blade is shown with the following configuration:

- Name:** SlowPages
- Description:** (empty)
- Source:** Alert on: Metrics
- Criteria:** Subscription: Microsoft Azure Sponsorship, Resource group: sol-newwebapp, Resource: sol-newwebapp

**FIGURE 4-26** Part of the Add rule blade

- D. Choose the metric from the drop-down list (Figure 4-27), in this case Average Response Time with a condition greater than a threshold of 2 seconds over a 15 minute period.



**FIGURE 4-27** Part of the Add rule blade where you can set the metric values

- E. From the same blade you can also indicate who to notify, configure a web hook, or even configure a Logic App to produce a workflow based on the alert.
5. Click OK to complete the alert configuration.
  6. You can view the alerts from the Alerts tab of the navigation pane.

#### **NOTE** MONITORING QUOTAS

You can also monitor quotas by selecting the Quotas tab from the left navigation pane. This gives you an indication of where you stand with your quotas based on the App Service plan.

## Design and configure Web Apps for scale and resilience

App Services provide various mechanisms to scale your web apps up and down by adjusting the number of instances serving requests and by adjusting the instance size. You can, for example, increase the number of instances (scale out) to support the load you experience during business hours, but then decrease (scale in) the number of instances during less busy hours

to save costs. Web Apps enable you to scale the instance count manually, automatically via a schedule, or automatically according to key performance metrics. Within a datacenter, Azure load balances traffic between all of your Web Apps instances using a round-robin approach.

You can also scale a web app by deploying to multiple regions around the world and then utilizing Microsoft Azure Traffic Manager to direct web app traffic to the appropriate region based on a round robin strategy or according to performance (approximating the latency perceived by clients of your application). Alternately, you can configure Traffic Manager to use the alternate regions as targets for failover if the primary region becomes unavailable.

In addition to scaling instance counts, you can manually adjust your instance size (scale up or down). For example, you can scale up your web app to utilize more powerful VMs that have more RAM memory and more CPU cores to serve applications that are more demanding of memory consumption or CPU utilization, or scale down your VMs if you later discover your requirements are not as great.



#### EXAM TIP

Web Apps provide a high availability SLA of 99.9 percent using only a single standard instance. You do not need to provision more than one instance to benefit from this SLA.

To scale your web app, follow these steps:

1. Navigate to the blade of your web app in the portal accessed via <https://portal.azure.com>.
2. Select the App Service plan tab from the left navigation pane. This takes you to the App Service Plan blade.
3. Select the Scale Up tab from the left navigation pane and you'll be taken to a blade to select the new pricing tier for your web app VMs.
4. Select the Scale Out tab and you'll be taken to the Scale Out blade to choose the number of instances to scale out or into (Figure 4-28).

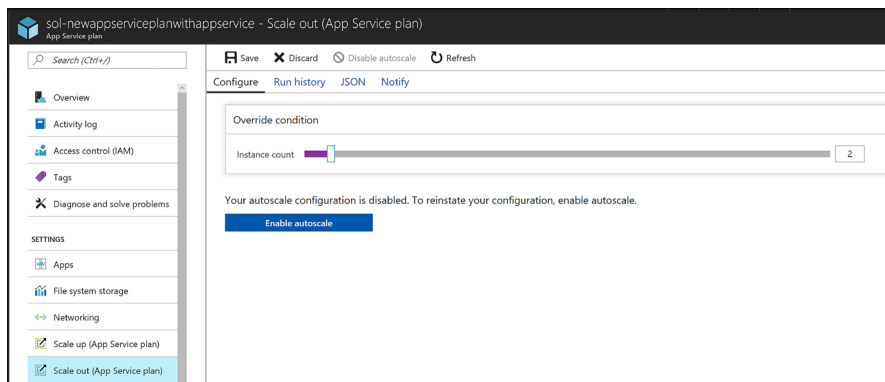


FIGURE 4-28 The scale out blade.

5. If you select Enable autoscale, you can create conditions based on metrics and rules in order for the site to automatically adjust instance count.

#### **MORE INFO MONITORING, ANALYTICS, AND AUTOSCALING**

For more information on monitoring web apps, analytics, and setting up autoscale, see:

<https://docs.microsoft.com/en-us/azure/log-analytics/log-analytics-azure-web-apps-analytics>, <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-analytics>, <https://docs.microsoft.com/en-us/Azure/monitoring-and-diagnostics/insights-autoscale-best-practices>, and <https://docs.microsoft.com/en-us/Azure/monitoring-and-diagnostics/insights-how-to-scale>.

## **Skill 4.2: Design Azure App Service API Apps**

Azure API Apps provide a quick and easy way to create and consume scalable RESTful APIs, using the language of your choice, in the cloud. As part of the Azure infrastructure, you can integrate API Apps with many Azure services such as API Management, Logic Apps, Functions, and many more. Securing your APIs can be done with a few clicks, whether you are using Azure Active Directory, OAuth, or social networks for single sign-on.

If you have existing APIs written in .NET, Node.js, Java, Python, or PHP, they can be brought into App Services as API Apps. When you need to consume these APIs, enable CORS support so you can access them from any client. Swagger support makes generating client code to use your API simple. Once you have your API App set up, and clients are consuming it, it is important to know how to monitor it to detect any issues early on.

#### **This skill covers how to:**

- Create and deploy API Apps
- Automate API discovery by using Swashbuckle
- Use Swagger API metadata to generate client code for an API app
- Monitor API Apps

## **Create and deploy API Apps**

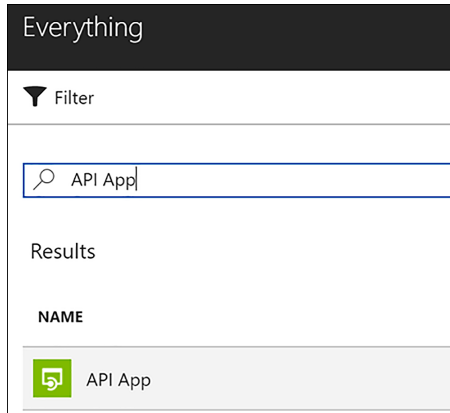
There are different ways you can create and deploy API Apps, depending on the language and development environment of choice. For instance, if you are using Visual Studio, you can create a new API Apps project and publish to a new API app, which provisions the service in Azure. If you are not using Visual Studio, you can provision a new API App service using the Azure portal, Azure CLI, or PowerShell.



## Creating a new API App from the portal

To create a new API app in the portal, complete the following steps:

1. Navigate to the portal accessed via <https://portal.azure.com>.
2. Select New on the command bar.
3. Within the Marketplace (Figure 4-29) search text box, type **API App**, and press Enter.



**FIGURE 4-29** Marketplace search for API App

4. Select API App from the results.
5. On the API App blade, select Create.
6. On the Create API App blade, choose your Azure subscription, select a Resource Group, select or create an App Service Plan, select whether you want to enable Application Insights, and then click Create.

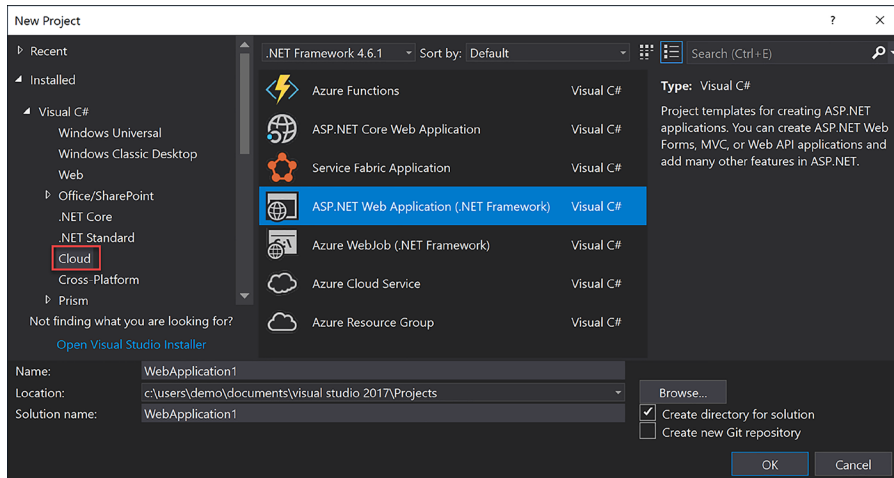
### **NOTE** SERVER-SIDE AND CLIENT-SIDE PROJECTS

After creating your API App service, you can quickly create sample ASP.NET, Node.js, or Java server-side and client-side projects using your new service, by selecting Quickstart from your API App blade in the portal.

## Creating and deploying a new API app with Visual Studio 2017

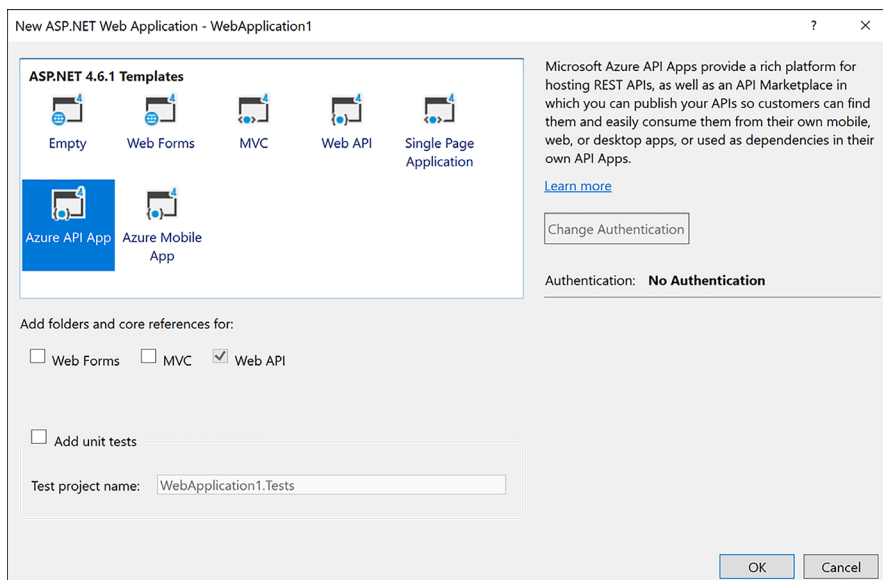
Visual Studio 2017 comes preconfigured with the ability to create an API app when you have installed the ASP.NET and web development, as well as Azure development workloads. Follow these steps to create a new API app with Visual Studio 2017:

1. Launch Visual Studio, and then select File > New > Project.
2. In the New Project dialog, select ASP.NET Web Application (.NET Framework) within the Cloud category (Figure 4-30). Provide a name and location for your new project, and then click OK.



**FIGURE 4-30** The ASP.NET Web Application Cloud project type

3. Select the Azure API App template (Figure 4-31), and then click OK.



**FIGURE 4-31** The Azure API App template

Visual Studio creates a new API App project within the specified directory, adding useful NuGet packages, such as:

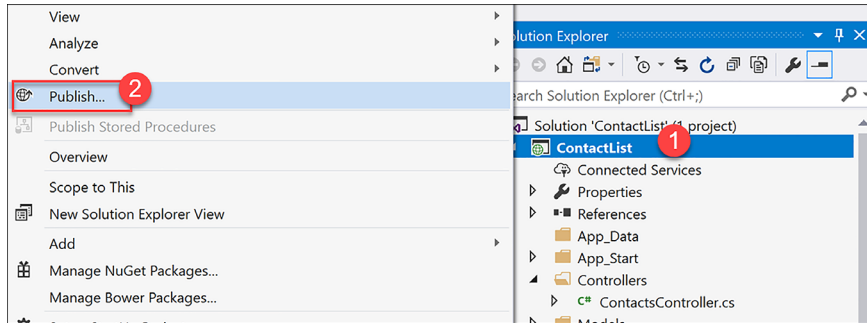
- Newtonsoft.Json for deserializing requests and serializing responses to and from your API app.

- Swashbuckle to add Swagger for rich discovery and documentation for your API REST endpoints.

In addition, Web API and Swagger configuration classes are created in the project's startup folder. All you need to do from this point, to deploy your API app is to complete your Controller actions, and publish from Visual Studio.

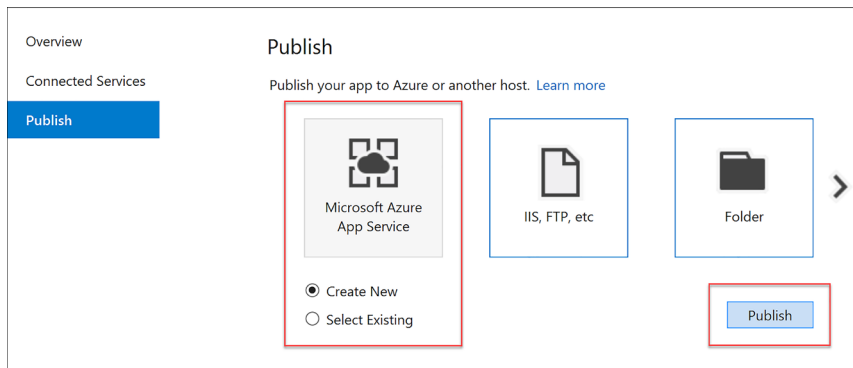
Follow these steps to deploy your API app from Visual Studio:

1. Right-click your project in the Visual Studio Solution Explorer (Figure 4-32), then click Publish.



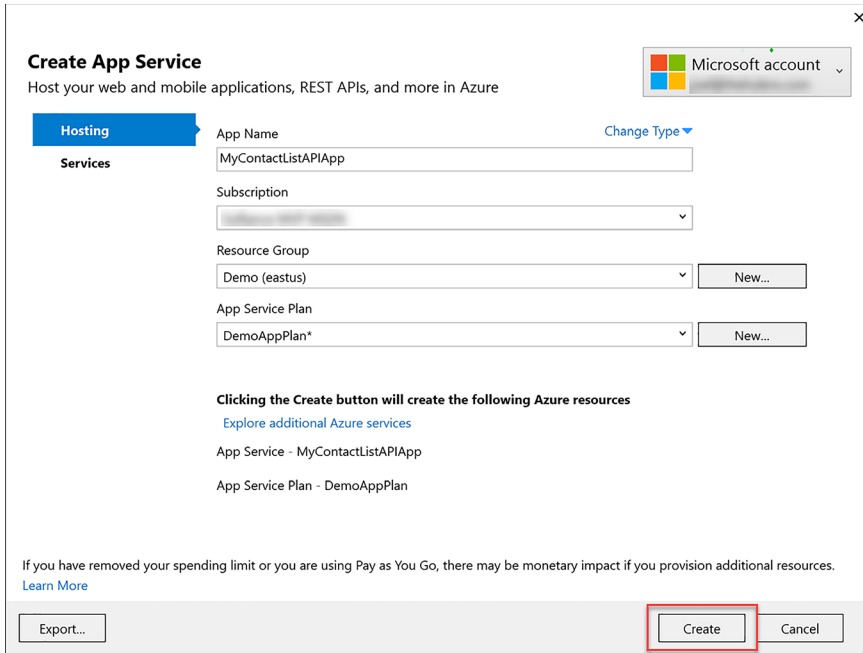
**FIGURE 4-32** Publish solution context menu

2. In the Publish dialog (Figure 4-33), select the Create New option underneath Microsoft Azure App Service, and then click Publish. This creates a new API app in Azure and publishes your solution to it. You could alternately select the Select Existing option to publish to an existing API App service.



**FIGURE 4-33** The Publish dialog

3. In the Create App Service dialog (Figure 4-34), provide a unique App name, select your Azure subscription and resource group, select or create an App Service Plan, and then click Create.



**Create App Service**  
Host your web and mobile applications, REST APIs, and more in Azure

Microsoft account

**Hosting**  
**Services**

App Name: MyContactListAPIApp [Change Type](#)

Subscription: [Dropdown]

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

App Service Plan: DemoAppPlan\* [New...](#)

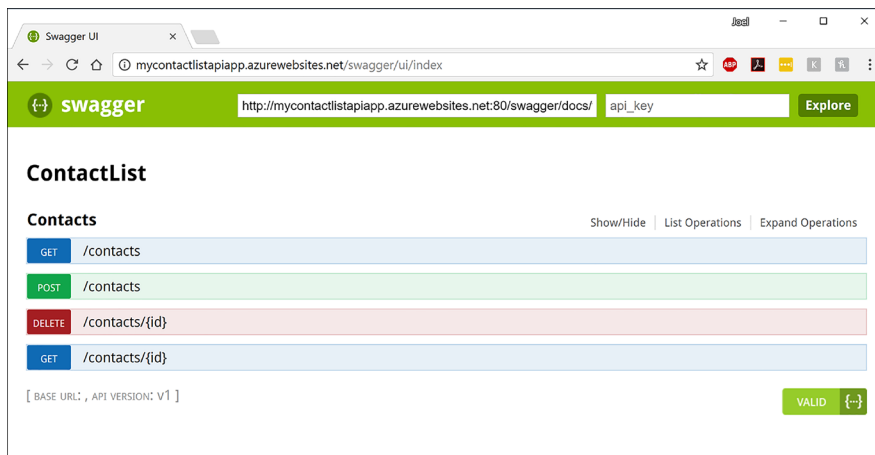
**Clicking the Create button will create the following Azure resources**  
[Explore additional Azure services](#)  
 App Service - MyContactListAPIApp  
 App Service Plan - DemoAppPlan

If you have removed your spending limit or you are using Pay as You Go, there may be monetary impact if you provision additional resources.  
[Learn More](#)

Export... **Create** Cancel

**FIGURE 4-34** Create App Service dialog

- When your API app is finished publishing, it will open in a new web browser. When the page is displayed, navigate to the /swagger path to view your generated API details, and to try out the REST methods. For example *http://<YOUR-API-APP>.azurewebsites.net/swagger/* (Figure 4-35).



**FIGURE 4-35** The Swagger interface for the published API App

**NOTE SWAGGER UI MAY NOT BE ENABLED BY DEFAULT IN ASP.NET PROJECT**

When you use the Swashbuckle NuGet package within an ASP.NET project, the Swagger UI may not be enabled by default. If this is the case, open `SwaggerConfig.cs` and uncomment the line that starts with `.EnableSwaggerUi(c =>.`

You do not need to uncomment any of the properties within the `EnableSwaggerUi` configuration to properly enable the UI.

**MORE INFO NODE.JS API APP TUTORIAL**

To follow a tutorial for creating and deploying an API App using Node.js, see <https://docs.microsoft.com/azure/app-service/app-service-web-tutorial-rest-api>.

## Automate API discovery by using Swashbuckle

Swagger is a popular, open source framework backed by a large ecosystem of tools that helps you design, build, document, and consume your RESTful APIs. The previous section included a screenshot of the Swagger page generated for an API App. This was generated by the Swashbuckle NuGet package.

**MORE INFO SWASHBUCKLE**

For more details on Swashbuckle, see <https://github.com/domaindrivendev/Swashbuckle>.

The core component of Swagger is the Swagger Specification, which is the API description metadata in the form of a JSON or YAML file. The specification creates the RESTful contract for your API, detailing all its resources and operations in a human and machine-readable format to simplify development, discovery, and integration with other services. This is a standardized OpenAPI Specification (OAS) for defining RESTful interfaces, which makes the generated metadata valuable when working with a wide range of consumers. Included in the list of consumers that can read the Swagger API metadata are several Azure services, such as Microsoft PowerApps, Microsoft Flow, and Logic Apps. Meaning, when you publish your API App service with Swagger, these Azure services and more immediately know how to interact with your API endpoints with no further effort on your part.

Beyond other Azure services being able to more easily use your API App, Swagger RESTful interfaces make it easier for other developers to consume your API endpoints. The API explorer that comes with `swagger-ui` makes it easy for other developers (and you) to test the endpoints and know what the data format looks like that need to be sent and should be returned in kind.

Generating this Swagger metadata manually can be a very tedious process. If you build your API using ASP.NET or ASP.NET Core, you can use the Swashbuckle NuGet package to automatically do this for you, saving a lot of time initially creating the metadata, and maintaining it. In

addition to its Swagger metadata generator engine, Swashbuckle also contains an embedded version of swagger-ui, which it will automatically serve up once Swashbuckle is installed.

## Use Swashbuckle in your API App project

Swashbuckle is provided by way of a set of NuGet packages: Swashbuckle and Swashbuckle.Core. When you create a new API App project using the Visual Studio template, these NuGet packages are already included. If you don't have them installed, follow these steps to add Swashbuckle to your API App project:

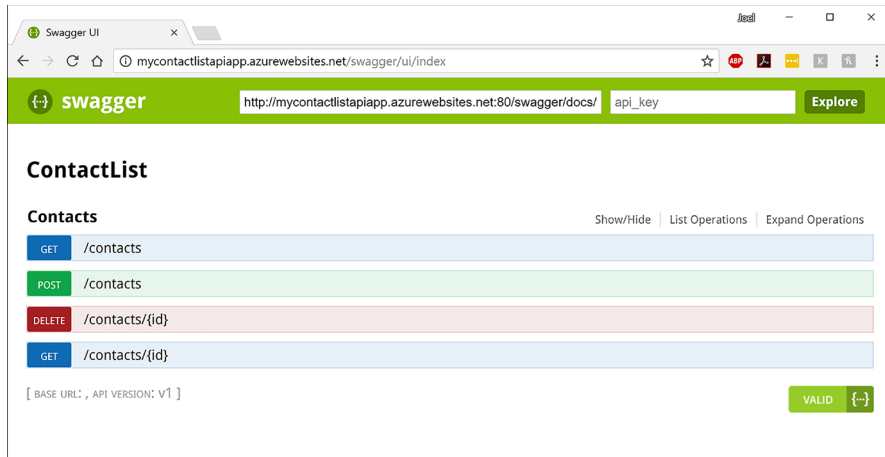
1. Install the Swashbuckle NuGet package, which includes Swashbuckle.Core as a dependency, by using the following command from the NuGet Package Manager Console:

```
Install-Package Swashbuckle
```

2. The NuGet package also installs a bootstrapper (App\_Start/SwaggerConfig.cs) that enables the Swagger routes on app start-up using WebActivatorEx. You can configure Swashbuckle's options by modifying the GlobalConfiguration.Configuration.EnableSwagger extension method in SwaggerConfig.cs. For example, to exclude API actions that are marked as Obsolete, add the following configuration:

```
public static void Register()
{
    var thisAssembly = typeof(SwaggerConfig).Assembly;
    GlobalConfiguration.Configuration
        .EnableSwagger(c =>
        {
            ...
            ...
            // Set this flag to omit descriptions for any actions
            decorated with the Obsolete attribute
            c.IgnoreObsoleteActions();
            ...
            ...
        });
}
```

3. Modify your project's controller actions to include Swagger attributes to aid the generator in building your Swagger metadata. Listing 4-1 illustrates the use of the SwaggerResponseAttribute at each controller method.
4. Swashbuckle is now configured to generate Swagger metadata for your API endpoints with a simple UI to explore that metadata. For example, the controller in Listing 4-1 may produce the UI shown in Figure 4-36.



**FIGURE 4-36** The Swagger interface for the published API App

**LISTING 4-1** C# code showing Swagger attributes added to the API App's controller actions

```

/// <summary>
/// Gets the list of contacts
/// </summary>
/// <returns>The contacts</returns>
[HttpGet]
[SwaggerResponse(HttpStatusCode.OK,
    Type = typeof(IEnumerable<Contact>))]
[Route("~/contacts")]
public async Task<IEnumerable<Contact>> Get()
{
    ...
}

/// <summary>
/// Gets a specific contact
/// </summary>
/// <param name="id">Identifier for the contact</param>
/// <returns>The requested contact</returns>
[HttpGet]
[SwaggerResponse(HttpStatusCode.OK,
    Description = "OK",
    Type = typeof(IEnumerable<Contact>))]
[SwaggerResponse(HttpStatusCode.NotFound,
    Description = "Contact not found",
    Type = typeof(IEnumerable<Contact>))]
[SwaggerOperation("GetContactById")]
[Route("~/contacts/{id}")]
public async Task<Contact> Get([FromUri] int id)
{
    ...
}

/// <summary>

```

```

/// Creates a new contact
/// </summary>
/// <param name="contact">The new contact</param>
/// <returns>The saved contact</returns>
[HttpPost]
[SwaggerResponse(HttpStatusCode.Created,
    Description = "Created",
    Type = typeof(Contact))]
[Route("~/contacts")]
public async Task<Contact> Post([FromBody] Contact contact)
{
    ...
}

```

You can test any of the API methods by selecting it from the list. Here we selected the /contacts/{id} GET method and tested it by entering a value of 2 in the id parameter, and clicking the Try It Out! button. Notice that Swagger details the return model schema, shows a Curl command and a Request URL for invoking the method, and shows the actual response body after clicking the button (Figure 4-37).

The screenshot shows the Swagger UI for the `GET /contacts/{id}` endpoint. The response class schema is defined as:

```

{
  {
    "Id": 0,
    "Name": "string",
    "EmailAddress": "string"
  }
}

```

The response content type is set to `application/json`. The parameters table shows:

Parameter	Value	Description	Parameter Type	Data Type
id	2		path	integer

The response messages table shows:

HTTP Status Code	Reason	Response Model	Headers
404	Contact not found		

The `Try it out!` button is highlighted with a red box. Below it, the curl command is:

```
curl -X GET --header "Accept: application/json" "http://mycontactlistapiapp.azurewebsites.net/contacts/2"
```

The request URL is:

```
http://mycontactlistapiapp.azurewebsites.net/contacts/2
```

The response body is:

```

{
  "Id": 2,
  "Name": "Lacy Barrera",
  "EmailAddress": "lacy@contoso.com"
}

```

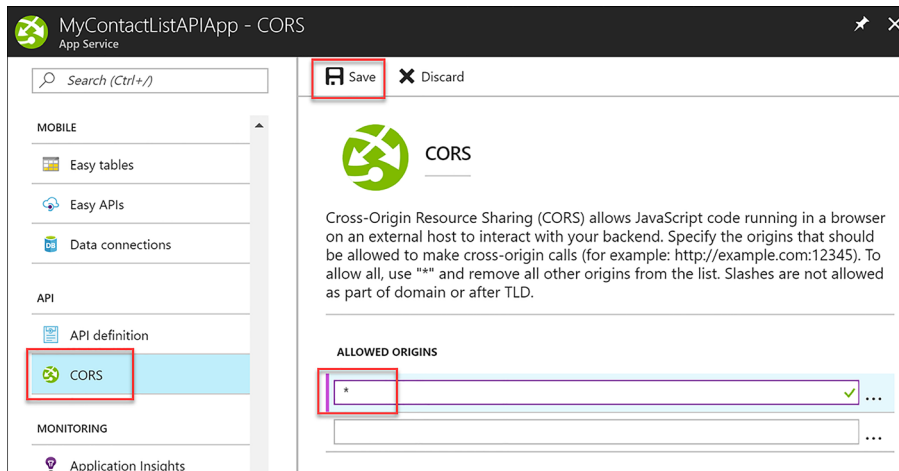
**FIGURE 4-37** An API method and result after testing with Swagger



## Enable CORS to allow clients to consume API and Swagger interface

Before clients, such as other web services or client code generators, can consume your API endpoints and Swagger interface, you need to enable CORS on the API App in Azure. To enable CORS, follow these steps:

1. Navigate to the portal accessed via <https://portal.azure.com>.
2. Open your API App service. You can find this by navigating to the Resource Group in which you published your service.
3. Select CORS from the left-hand menu (Figure 4-38). Enter one or more allowed origins, then select Save. To allow all origins, enter an asterisk (\*) in the Allowed Origins field and remove all other origins from the list.



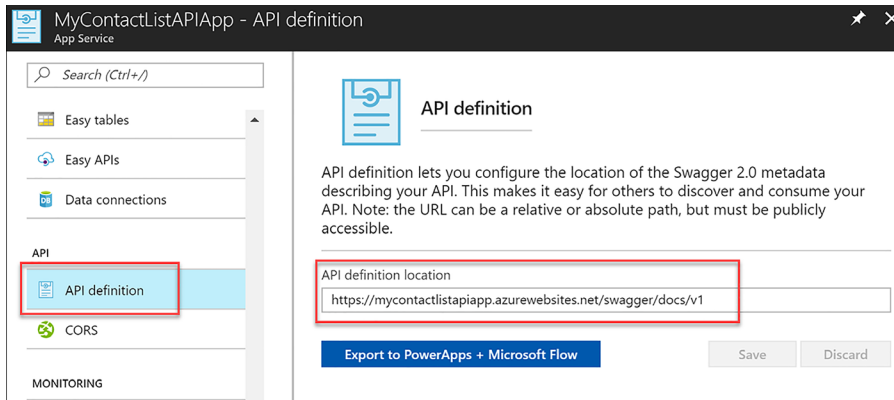
**FIGURE 4-38** Enabling cross-origin calls for all sources

## Use Swagger API metadata to generate client code for an API app

There are tools available to generate client code for your API Apps that have Swagger API definitions, like the [swagger.io](https://swagger.io) online editor. The previous section demonstrated how you can automatically generate the Swagger API metadata, using the Swashbuckle NuGet package.

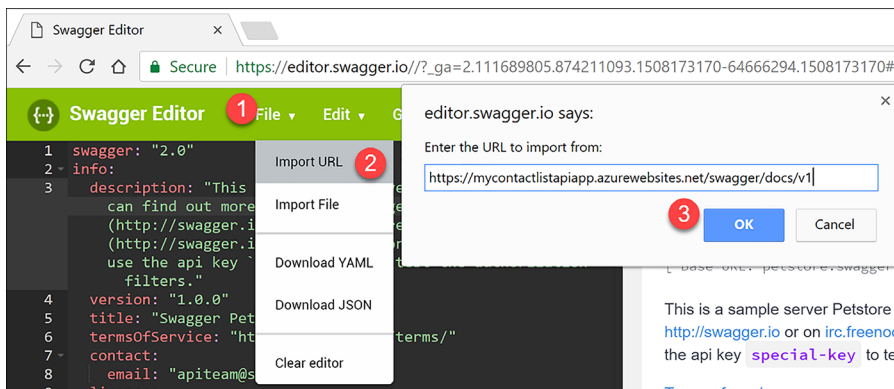
To generate client code for your API app that has Swagger API metadata, follow these steps:

1. Find your Swagger 2.0 API definition document by navigating to <http://<your-api-app>/swagger/docs/v1> (v1 is the API version). Alternately, you can find it by navigating to the Azure portal, opening your API App service, and selecting API definition from the left-hand menu. This displays your Swagger 2.0 API definition URL (Figure 4-39).



**FIGURE 4-39** Steps to find the API App's Swagger 2.0 metadata URL

2. Navigate to <https://editor.swagger.io> to use the Swagger.io Online Editor.
3. Select File > Import URL. Enter your Swagger 2.0 metadata URL in the dialog box and click OK (Figure 4-40).



**FIGURE 4-40** Steps to import the Swagger 2.0 metadata

4. After a few moments, your Swagger metadata appears on the left-hand side of the editor, and the discovered API endpoints will be displayed on the right. Verify that all desired API endpoints appear, and then select Generate Client from the top menu. Select the desired language or platform for the generated client app. This initiates a download of a zip file containing the client app (Figure 4-41).

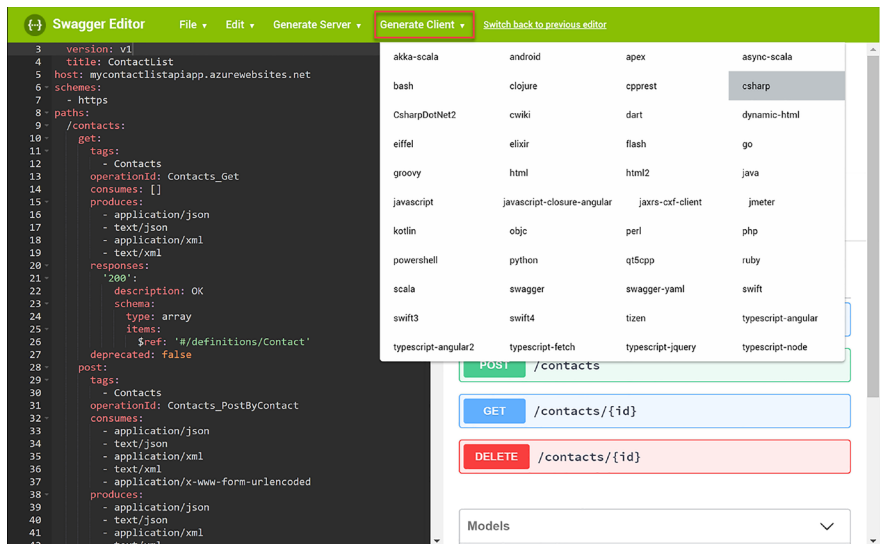


FIGURE 4-41 Steps to generate client code in Swagger.io

## Monitor API Apps

App Service, under which API Apps reside, provides built-in monitoring capabilities, such as resource quotas and metrics. You can also set up alerts and automatic scaling based on these metrics. In addition, Azure provides built-in diagnostics to assist with debugging an App Service web or API app. A combination of the monitoring capabilities and logging should provide you with the information you need to monitor the health of your API app, and determine whether it is able to meet capacity demands.

## Using quotas and metrics

API Apps are subject to certain limits on the resources they can use. The limits are defined by the App Service plan associated with the app. If the application is hosted in a Free or Shared plan, and then the limits on the resources the app can use are defined by Quotas, as discussed earlier for Web Apps.

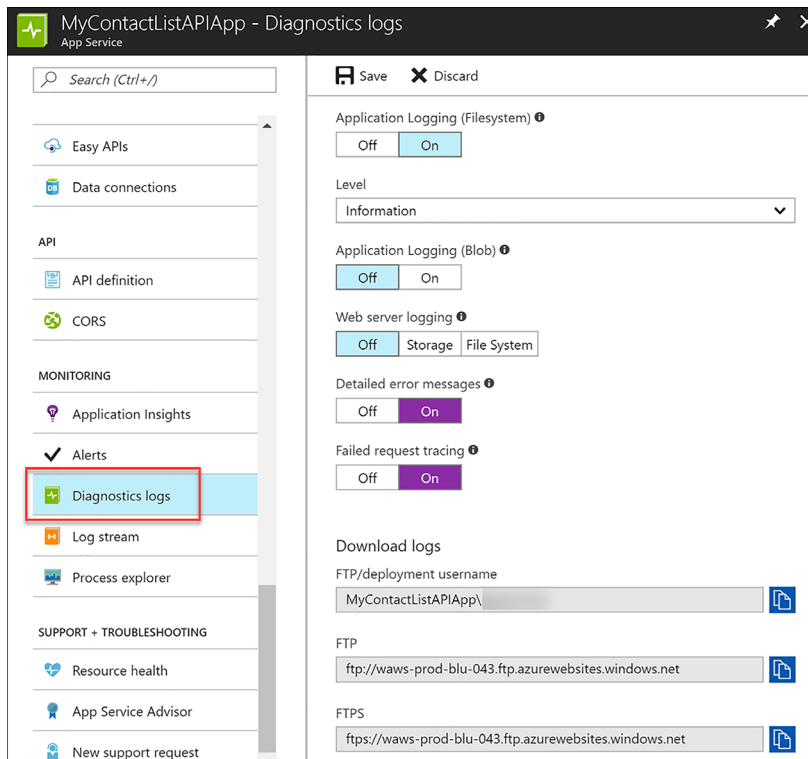
If you exceed the CPU and bandwidth quotas, your app will respond with a 403 HTTP error, so it's best to keep an eye on your resource usage. Exceeding memory quotas causes an application reset, and exceeding the filesystem quota will cause write operations to fail, even to logs. If you need to increase or remove any of these quotas, you can upgrade your App Service plan.

Metrics that you can view pertaining to your apps are the same as shown earlier in Table 4-3. As with Web Apps, metrics are accessed from the Overview blade of your API App within the Azure portal by clicking one of the metrics charts, such as Requests or Average Response Time. Once you click a chart, you can customize it by clicking it and selecting edit chart. From here you can change the time range, chart type, and metrics to display.

## Enable and review diagnostics logs

By default, when you provision a new API App, diagnostics logs are disabled. These are detailed server logs you can use to troubleshoot and debug your app. To enable diagnostics logging, perform the following steps:

1. Navigate to the portal accessed via <https://portal.azure.com>.
2. Open your API App service. You can find this by navigating to the Resource Group in which you published your service.
3. Select Diagnostics logs from the left-hand menu (Figure 4-42). Turn on any logs you wish to capture. When you enable application diagnostics, you also choose the Level. This setting allows you to filter the information captured to informational, warning, or error information. Setting this to verbose will log all information produced by the application. This is also where you can go to retrieve FTP information for downloading the logs.



**FIGURE 4-42** Steps to enable diagnostics logs

You can download the diagnostics logs via FTP, or they can be downloaded as a zip archive by using PowerShell or the Azure CLI.

The types of logs and structure for accessing logs follow that described for Web Apps and shown in Table 4-2.

**MORE INFO MONITOR AN API APP WITH WEB SERVER LOGS**

For more information about monitoring API Apps with web server logs, see: <https://docs.microsoft.com/azure/app-service/web-sites-enable-diagnostic-log>. To view sample CLI scripts you can use to enable and download logs, see: <https://docs.microsoft.com/azure/app-service/scripts/app-service-cli-monitor>. For information on troubleshooting your API Apps with Visual Studio, refer to: <https://docs.microsoft.com/azure/app-service/web-sites-dotnet-troubleshoot-visual-studio>.

**MORE INFO VIEWING METRICS AND QUOTAS FOR YOUR APP SERVICE**

For more information on viewing metrics and quotas for your App Service, such as an API App, see <https://docs.microsoft.com/azure/app-service/web-sites-monitor>.

**MORE INFO RECEIVING ALERT NOTIFICATIONS ON YOUR APP'S METRICS**

You can configure alert notifications that you can receive when certain metrics thresholds are reached. To find out how to do this, see: <https://docs.microsoft.com/azure/monitoring-and-diagnostics/insights-receive-alert-notifications>.

## Skill 4.3: Develop Azure App Service Logic Apps

Azure Logic Apps is a fully managed iPaaS (Integration Platform as a Service) that helps you simplify and implement scalable integrations and workflows in the cloud. As such, you don't have to worry about infrastructure, management, scalability, and availability because all of that is taken care of for you. Its Logic App Designer gives you a nice way to model and automate your process visually, as a series of steps known as a workflow. At its core, it allows you to quickly integrate with many services and protocols, inside of Azure, outside of Azure, as well as on-premises. When you create a Logic App, you start out with a trigger, like 'When an email arrives at this account,' and then you act on that trigger with many combinations of actions, condition logic, and conversions.

**MORE INFO LOGIC APP CONNECTORS**

There is a large list of connectors you can use to integrate with services and protocols that can be found at <https://docs.microsoft.com/azure/connectors/apis-list>.

**This skill covers how to:**

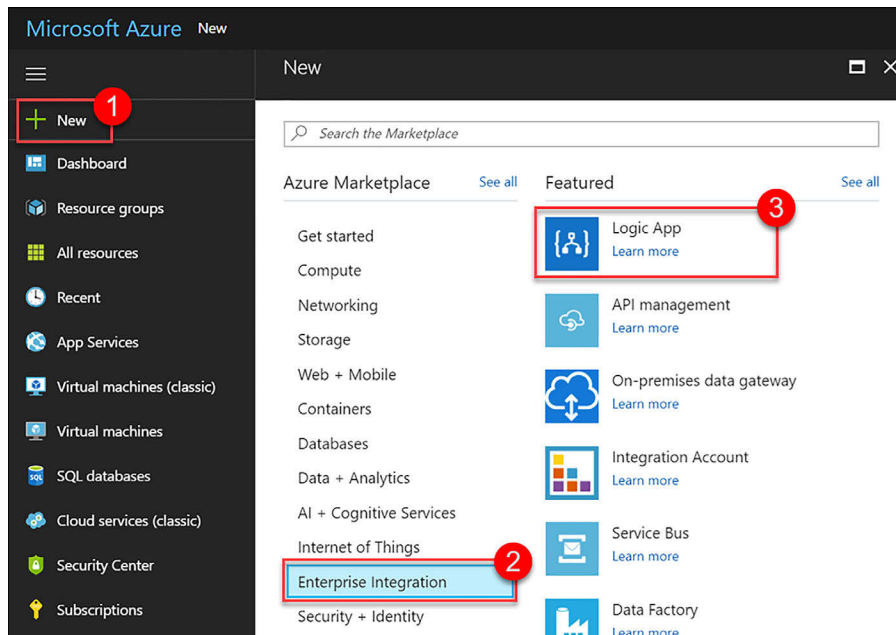
- Create a Logic App connecting SaaS services
- Create a Logic App with B2B capabilities
- Create a Logic App with XML capabilities
- Trigger a Logic App from another app
- Create custom and long-running actions
- Monitor Logic Apps

## Create a Logic App connecting SaaS services

One of the strengths of Logic Apps is its ability to connect a large number of SaaS (Software as a Service) services to create your own custom workflows. In this example, we will connect Twitter with an Outlook.com or hosted Office 365 mailbox to email certain tweets as they arrive.

To create a new Logic App in the portal, complete the following steps:

1. Navigate to the portal accessed via <https://portal.azure.com>.
2. Select New on the command bar.
3. Select Enterprise Integration, then Logic App (Figure 4-43).



**FIGURE 4-43** Creating a new Logic App from the Azure Portal

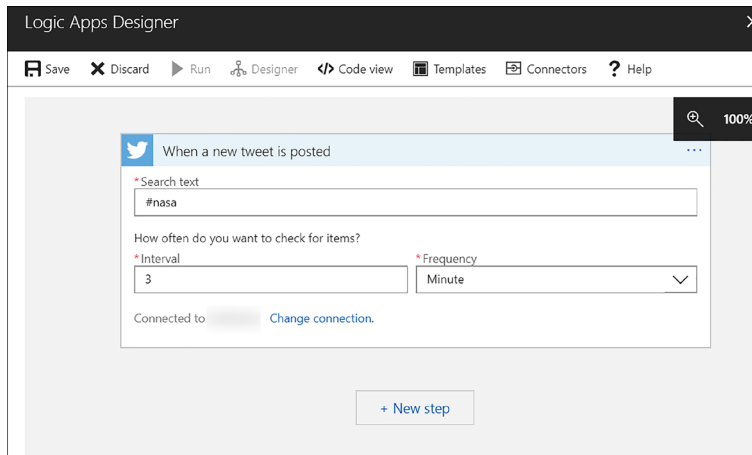
4. Provide a unique name, select a resource group and location, check Pin To Dashboard, and then click Create (Figure 4-44).

**FIGURE 4-44** The Create logic app form

Follow the above steps to create new Logic Apps as needed in the remaining segments for this skill.

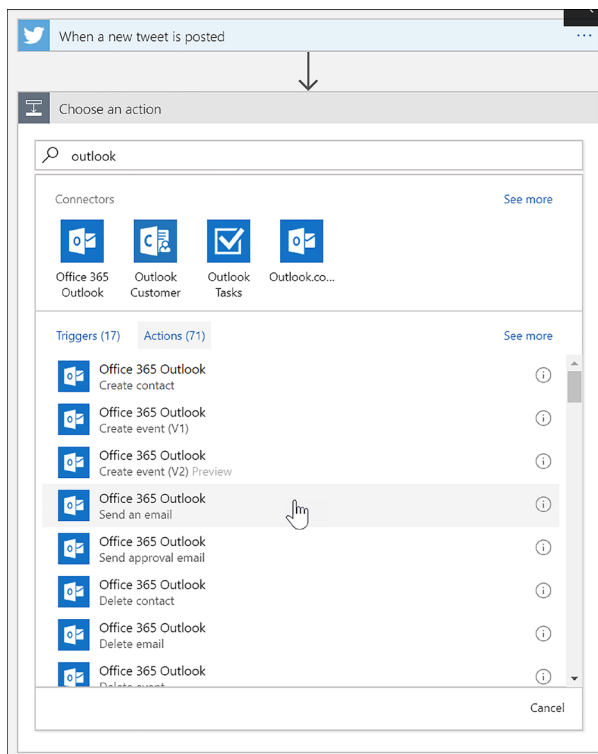
Once the Logic App has been provisioned, open it to view the Logic Apps Designer. This is where you design or modify your Logic App. You can select from a series of commonly used triggers, or from several templates you can use as a starting point. The following steps show how to create one from scratch.

1. Select Blank Logic App under Templates.
2. All Logic Apps start with a trigger. Search the list for Twitter, and then select it.
3. Click Sign in to create a connection to Twitter with your Twitter account. A dialog will appear where you sign in and authorize the Logic App to access your account.
4. In the Twitter trigger form on the designer (Figure 4-45), enter your search text to return certain tweets (such as #nasa), and select an interval and frequency, establishing how often you wish to check for items, returning all tweets during that time span.



**FIGURE 4-45** The Twitter trigger form in the Logic Apps Designer

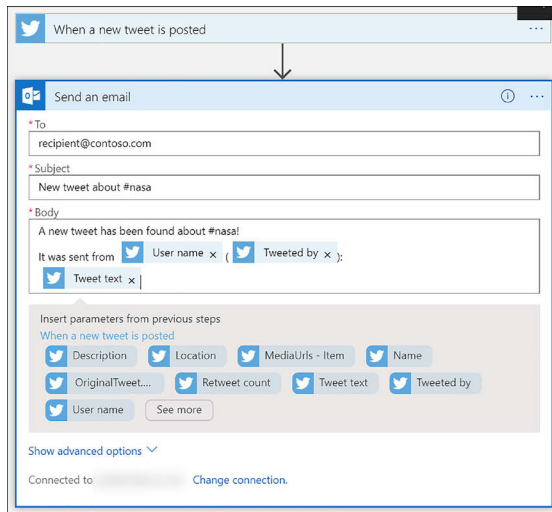
5. Select the + New Step button, and then choose Add An Action.
6. Type **outlook** in the search box, and then select Office 365 Outlook (Send An Email) from the results. Alternately, you can select Outlook.com from the list (Figure 4-46).



**FIGURE 4-46** Adding a new Office 365 Outlook action in the Logic Apps Designer



7. Click Sign In to create a connection to your Office 365 Outlook account (Figure 4-47).
8. In the Send An Email form, provide values for the email recipient, the subject of the email, and the body. In each of these fields, you can select parameters from the Twitter Connector, such as the tweet's text and who posted it.



**FIGURE 4-47** Adding details to a new Office 365 Outlook action in the Logic Apps Designer

9. Click Save in the Logic Apps Designer menu. Your Logic App is now live. If you wish to test right away and not wait for your trigger interval, click Run.

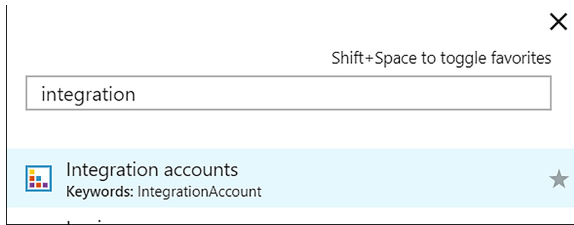
## Create a Logic App with B2B capabilities

Logic Apps support business-to-business (B2B) workflows and communication through the Enterprise Integration Pack. This allows organizations to exchange messages electronically, even if they use different protocols and formats. Enterprise integration allows you to store all your artifacts in one place, within your integration account, and secure messages through encryption and digital signatures. To access these artifacts from a logic app, you must first link it to your integration account. Your integration account needs both Partner and Agreement artifacts prior to creating B2B workflows for your logic app.

### Create an integration account

To get started with the Enterprise Integration Pack so you can create B2B workflows, you must first create an integration account, following these steps:

1. Navigate to the portal accessed via <https://portal.azure.com>.
2. Select More Services on the command bar.
3. In the filter box, type **integration**, and then select Integration Accounts in the results list (Figure 4-48).



**FIGURE 4-48** Navigating to the Integration accounts blade

4. At the top of the Integration Accounts blade, select + Add.
5. Provide a name for your Integration Account (Figure 4-49), select your resource group, location, and a pricing tier. Once validation has passed, click Create.

 A screenshot of the 'Integration account' creation form. The form has a dark header with the title 'Integration account' and a close button. The form contains several fields:
 

- Name:** A text input field containing 'MyIntegrationAccount' with a green checkmark icon to its right.
- Subscription:** A dropdown menu with a downward arrow.
- Resource group:** A section with two radio buttons: 'Create new' (unselected) and 'Use existing' (selected). Below the radio buttons is a dropdown menu containing 'Demo'.
- Pricing Tier:** A dropdown menu containing 'Free'.
- Location:** A dropdown menu containing 'West US'.
- Log Analytics:** A section with two buttons: 'On' and 'Off' (highlighted in blue).
- Pin to dashboard:** A checkbox that is currently unchecked.
- Create:** A blue button with the text 'Create'.
- Automation options:** A blue link with the text 'Automation options'.

**FIGURE 4-49** The create Integration account form

#### **NOTE** INTEGRATION ACCOUNT AND LOGIC APP

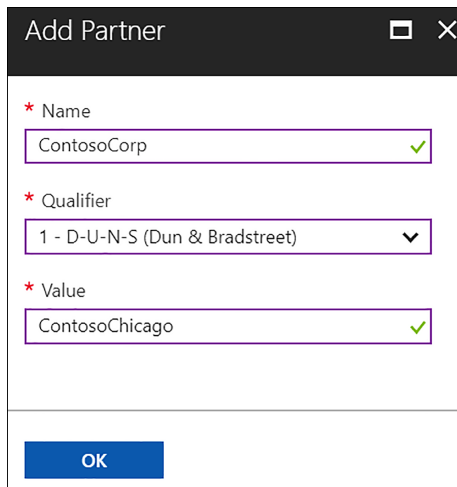
Your integration account and logic app must be in the same location before linking them.

## Add partners to your integration account

Partners are entities that participate in B2B transactions and exchange messages between each other. Before you can create partners that represent you and another organization in these transactions, you must both share information that identifies and validates messages sent by each other. After you discuss these details and are ready to start your business relationship, you can create partners in your integration account to represent you both. These message details are called agreements. You need at least two partners in your integration account to create an agreement. Your organization must be the host partner, and the other partner(s) guests. Guest partners can be outside organizations, or even a department in your own organization.

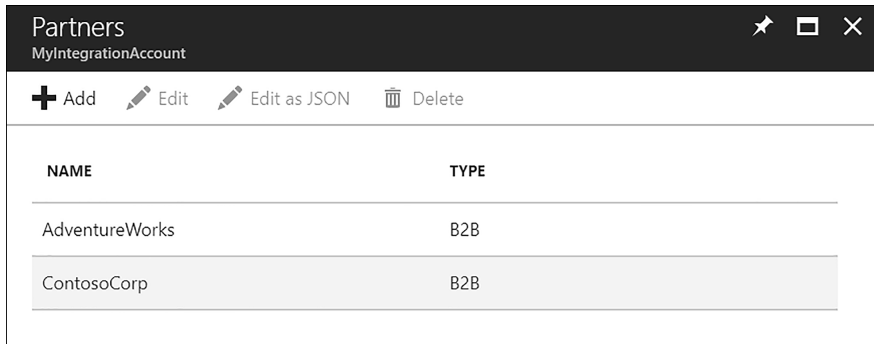
To add a partner to your integration account, follow these steps:

1. Navigate to the portal accessed via <https://portal.azure.com>.
2. Select More Services on the command bar.
3. In the filter box, type **integration**, then select Integration Accounts in the results list.
4. Select your integration account, and then select the Partners tile.
5. In the Partners blade, select + Add.
6. Provide a name for your partner (Figure 4-50), select a Qualifier, and then enter a Value to help identify documents that transfer through your apps. When finished, click OK.



**FIGURE 4-50** Adding a partner to an Integration account

7. After a few moments, the new partner (Figure 4-51) will appear in your list of partners.



NAME	TYPE
AdventureWorks	B2B
ContosoCorp	B2B

**FIGURE 4-51** Partners added to an Integration account

## Add an agreement

Now that you have partners associated with your integration account, you can allow them to communicate seamlessly using industry standard protocols through agreements. These agreements are based on the type of information exchanged, and through which protocol or transport standards they will communicate: AS2, X12, or EDIFACT.

Follow these steps to create an AS2 agreement:

1. Navigate to the portal accessed via <https://portal.azure.com>.
2. Select More Services on the command bar.
3. In the filter box, type **integration**, and then select Integration Accounts in the results list (Figure 4-52).
4. Select your integration account, and then select the Agreements tile.
5. In the Agreements blade, select + Add.
6. Provide a name for your agreement and select AS2 for the agreement type. Now select the Host Partner, Host Identity, Guest Partner, and Guest Identity. You can override send and receive settings as desired. Click OK.

Add

\* Name  
AS2Agreement ✓

\* Agreement type  
AS2 ▼

\* Host Partner  
ContosoCorp ▼

\* Host Identity  
AS2Identity : ContosoChicago ▼

\* Guest Partner  
AdventureWorks ▼

\* Guest Identity  
AS2Identity : AdventureWorks ▼

Receive Settings >

Send Settings >

OK

**FIGURE 4-52** Adding an agreement to an Integration account

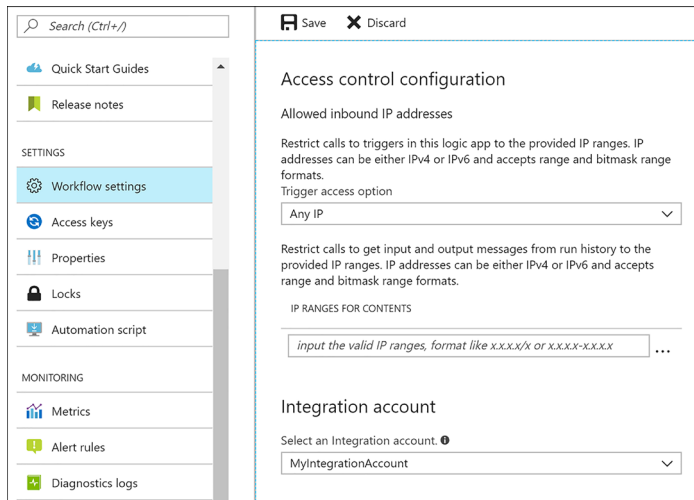
## Link your Logic app to your Enterprise Integration account

You will need to link your Logic app to your integration account so you can create B2B workflows using the partners and agreements you've created in your integration account. You must make sure that both the integration account and Logic app are in the same Azure region before linking.

To link, follow these steps:

1. Navigate to the portal accessed via <https://portal.azure.com>.
2. Select More Services on the command bar.
3. In the filter box, type **logic**, and then select Logic Apps in the results list.
4. Select your logic app, and then select Workflow settings.

5. In the Workflow settings blade, select your integration account from the select list, and click Save (Figure 4-53).

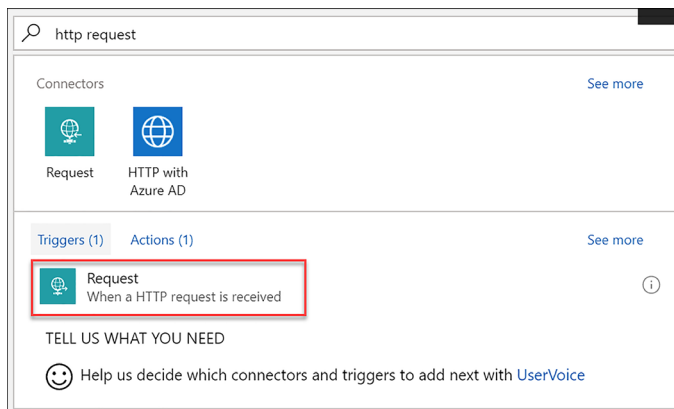


**FIGURE 4-53** Linking an integration account with a logic app

## Use B2B features to receive data in a Logic App

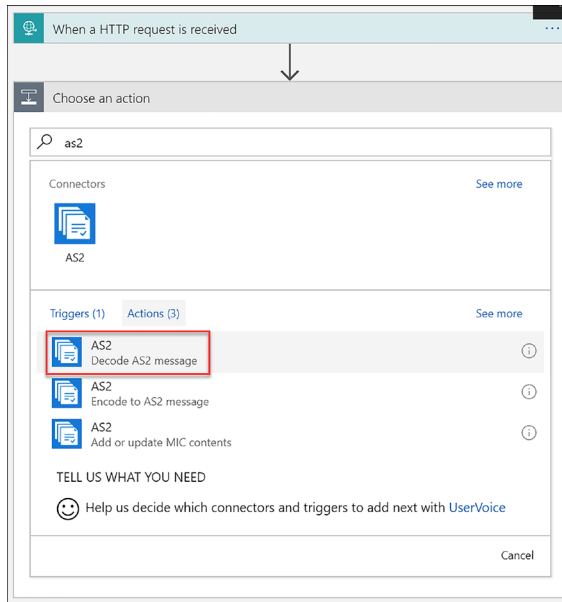
After creating an integration account, adding partners and agreements to it, and linking it to a Logic app, you can now create a B2B workflow using the Enterprise Integration Pack, following these steps:

1. Open the Logic App Designer on the Logic app that has a linked integration account.
2. Select Blank Logic App under Templates.
3. Search for “http request” in the trigger filter, and then select Request (When an HTTP request is received) from the list of results (Figure 4-54).



**FIGURE 4-54** Selecting a Request trigger in the Logic App Designer

4. Select the + New Step button, and then choose Add An Action.
5. Type **as2** in the search box, and then select AS2 (Decode AS2 Message) from the results (Figure 4-55).



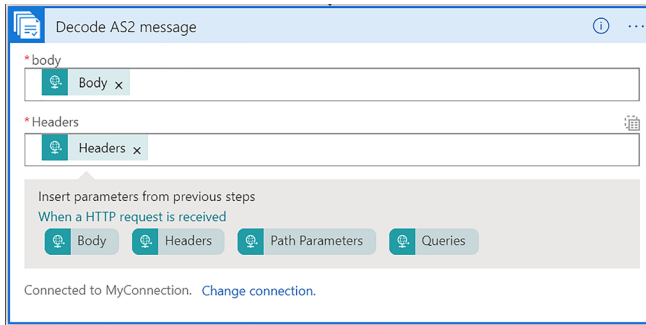
**FIGURE 4-55** Selecting a Decode AS2 Message action in the Logic App Designer

6. In the form that follows, provide a connection name, and then select your integration account, and click Create (Figure 4-56).

The screenshot shows the 'Decode AS2 message' configuration form. It has a 'Connection Name' field with the text 'MyConnection'. Below that is an 'Integration Account' section with a table. The table has three columns: 'Name', 'Resource Group', and 'Location'. There is one row with the values 'MyIntegrationAccount', 'Demo', and 'westus'. At the bottom of the form, there is a blue 'Create' button and a link that says 'Manually enter connection information'.

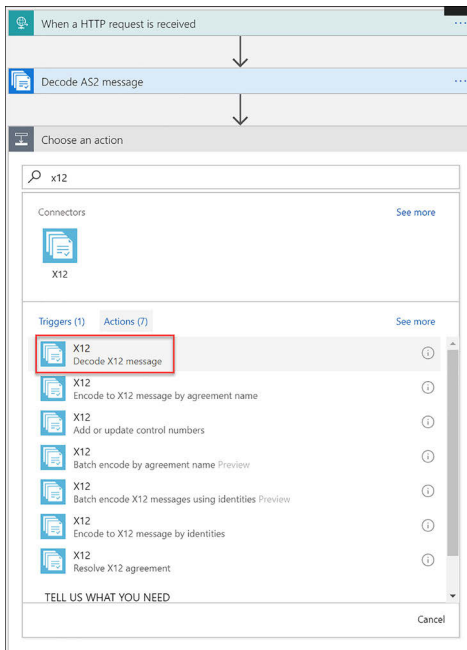
**FIGURE 4-56** Setting the Decode AS2 Message connection information form in the Logic App Designer

7. Add the Body that you want to use as input. In this example, we selected the body of the HTTP request that triggers the Logic app. Add the required Headers for AS2. In this example, we selected the headers of the HTTP request that triggers the Logic app (Figure 4-57).



**FIGURE 4-57** Setting the Decode AS2 Message body and headers information form in the Logic App Designer

8. Select the + New Step button, and then choose Add An Action.
9. Type **x12** in the search box, and then select X12 (Decode X12 Message) from the results (Figure 4-58).

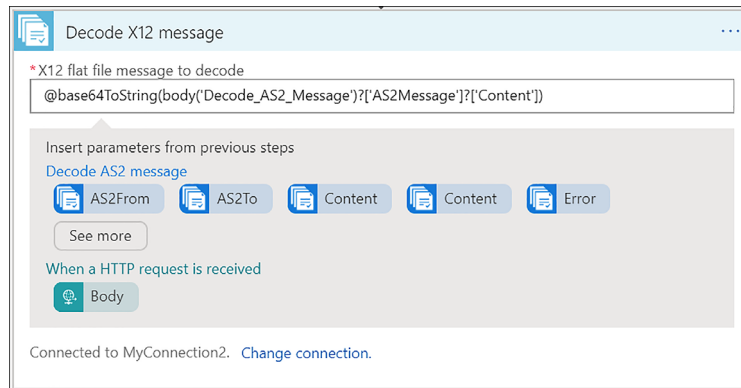


**FIGURE 4-58** Selecting a Decode X12 Message action in the Logic App Designer

10. In the form that follows, provide a connection name, and then select your integration account as before, and click Create (Figure 4-59).
11. The input for this new action is the output for the previous AS2 action. Because the actual message content is JSON-formatted and base64-encoded, you must specify an expression as the input. To do this, you type the following into the X12 Flat File Message

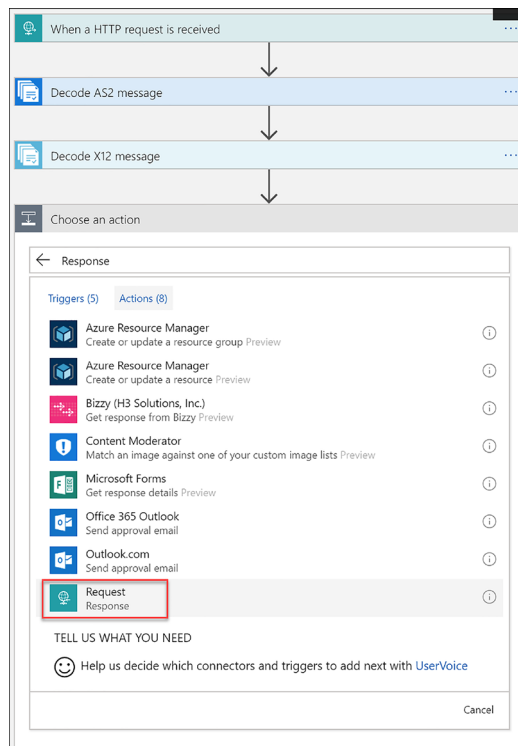


to Decode field: **@base64ToString(body('Decode\_AS2\_Message')?['AS2Message']?['Content'])**



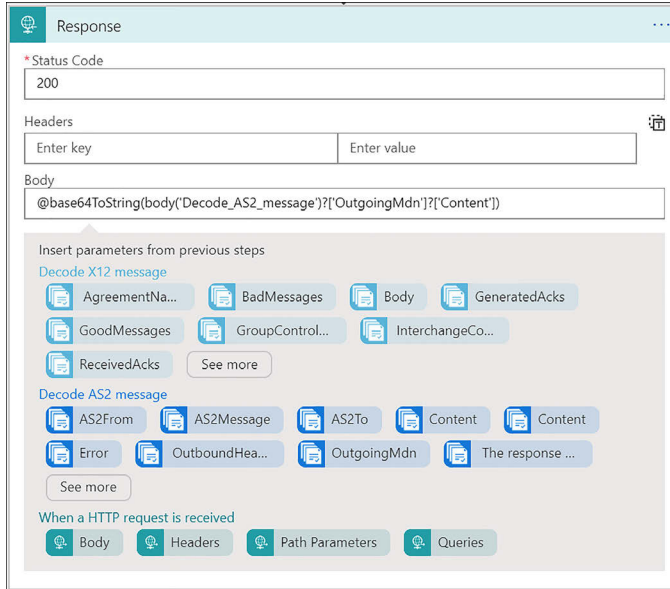
**FIGURE 4-59** Setting the Decode X12 flat file message to decode the information form in the Logic App Designer

12. Select the + New Step button, and then choose Add An Action (Figure 4-60).
13. Type **response** in the search box, and then select Request (Response) from the results.



**FIGURE 4-60** Selecting a Request (Response) action in the Logic App Designer

14. The response body should include the MDN from the output of the Decode X12 Message action (Figure 4-61). To do this, we type the following into the Body field: **@base64ToString(body('Decode\_AS2\_message')?['OutgoingMdn']?['Content'])**



The screenshot shows the 'Response' configuration pane in the Logic App Designer. At the top, the 'Status Code' is set to 200. Below it, the 'Headers' section is empty. The 'Body' section contains the expression `@base64ToString(body('Decode_AS2_message')?['OutgoingMdn']?['Content'])`. A large panel below the body field, titled 'Insert parameters from previous steps', provides a visual library of variables and outputs from previous actions. It is organized into three sections: 'Decode X12 message' (with outputs like AgreementNa..., BadMessages, Body, GeneratedAcks, GoodMessages, GroupControl..., InterchangeCo..., ReceivedAcks), 'Decode AS2 message' (with outputs like AS2From, AS2Message, AS2To, Content, Content, Error, OutboundHea..., OutgoingMdn, The response ...), and 'When a HTTP request is received' (with outputs like Body, Headers, Path Parameters, Queries). Each output is represented by a small icon and a text label.

FIGURE 4-61 Setting the body in the Response form in the Logic App Designer

15. Click Save in the Logic Apps Designer menu.

## Create a Logic App with XML capabilities

Oftentimes, businesses send and receive data between one or more organizations in XML format. Due to the dynamic nature of XML documents, schemas are used to confirm that the documents received are valid and are in the correct format. Schemas are also used to transform data from one format to another. Transforms are also known as maps, which consist of source and target XML schemas. When you link your logic app with an integration account, the schema and map artifacts within enable your Logic app to use these Enterprise Integration Pack XML capabilities.

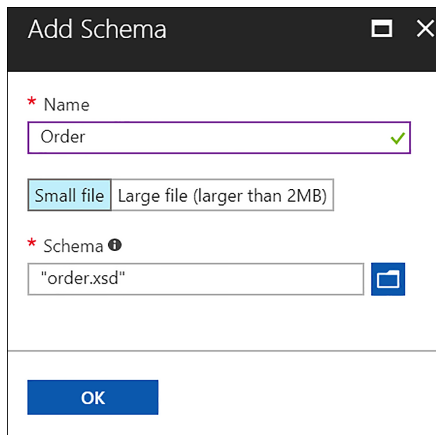
The XML features included with the Enterprise Integration pack are:

- **XML validation** Used to validate incoming and outgoing XML messages against a specific schema.
- **XML transform** Used to convert data from one format to another.
- **Flat file encoding/decoding** Used to encode XML content prior to sending, or to convert XML content to flat files.
- **XPath** Used to extract specific properties from a message, using an xpath expression.

## Add schemas to your integration account

Since schemas are used to validate and transform XML messages, you must add one or more to your integration account before working with the Enterprise Integration Pack XML features within your linked logic app. To add a new schema, follow these steps:

1. Navigate to the portal accessed via <https://portal.azure.com>.
2. Select More Services on the command bar.
3. In the filter box, type **integration**, and then select Integration Accounts in the results list (Figure 4-62).
4. Select your integration account, and then select the Schemas tile.
5. In the Schemas blade, select + Add.
6. Provide a name for your schema and select whether it is a small file (<= 2MB) or a large file (> 2MB). If it is a small file, you can upload it here. If you select Large file, then you need to provide a publicly accessible URI to the file. In this case, we're uploading a small file. Click the Browse button underneath Schema to select a local XSD file to upload. Click OK.



**FIGURE 4-62** Adding a schema to an Integration account

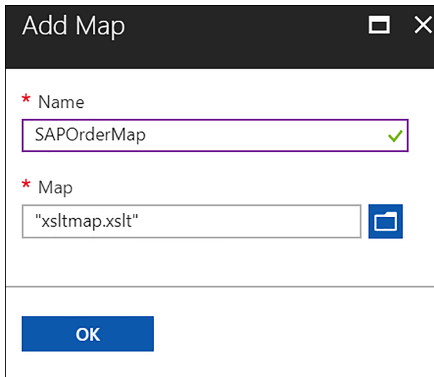
## Add maps to your Integration account

When you want your Logic app to transform data from one format to another, you first add a map (schema) to your linked Integration account.

To add a new schema, follow these steps:

1. Navigate to the portal accessed via <https://portal.azure.com>.
2. Select More Services on the command bar.
3. In the filter box, type **integration**, then select Integration Accounts in the results list.

4. Select your integration account, and then select the Maps tile.
5. In the Maps blade, select + Add.
6. Provide a name for your map and click the Browse button underneath Map to select a local XSLT file to upload. Click OK (Figure 4-63).



**FIGURE 4-63** Adding a map to an Integration account

#### **MORE INFO** HOW TO CREATE A TRANSFORM/MAP

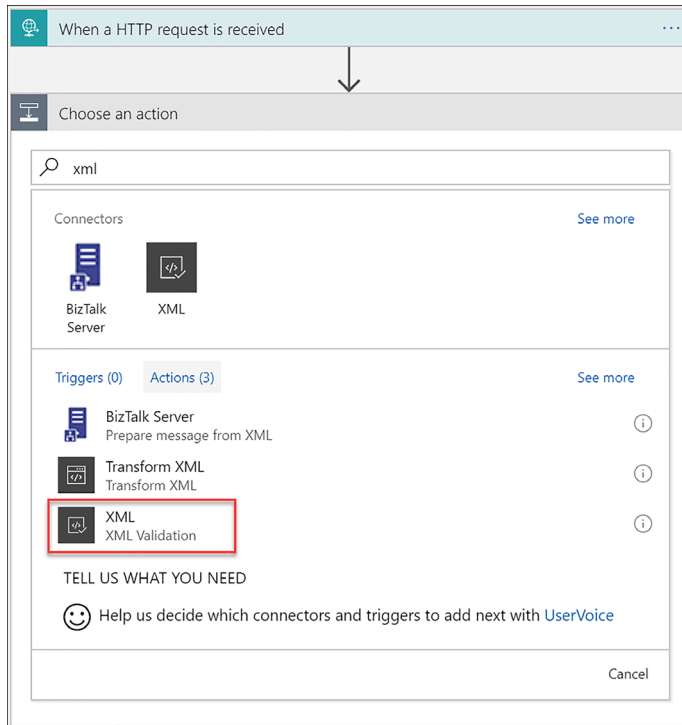
You can create the map that you upload to your Integration account by using the Visual Studio Enterprise Integration SDK at <https://aka.ms/vsmapsandschemas>.

## Add XML capabilities to the linked Logic App

After adding an XML schema and map to the Integration account, you are ready to use the Enterprise Integration Pack's XML validation, XPath Extract, and Transform XML operations in a Logic App.

Once your LogicApp has been linked to the Integration account with these artifacts, follow these steps to use the XML capabilities in your Logic App:

1. Open the Logic App Designer on the Logic pp that has a linked Integration account.
2. Select Blank Logic App under Templates.
3. Search for "http request" in the trigger filter, and then select Request (When An HTTP Request Is Received) from the list of results (Figure 4-64).
4. Select the + New Step button, and then choose Add An Action.
5. Type **xml** in the search box, and then select XML (XML Validation) from the results.



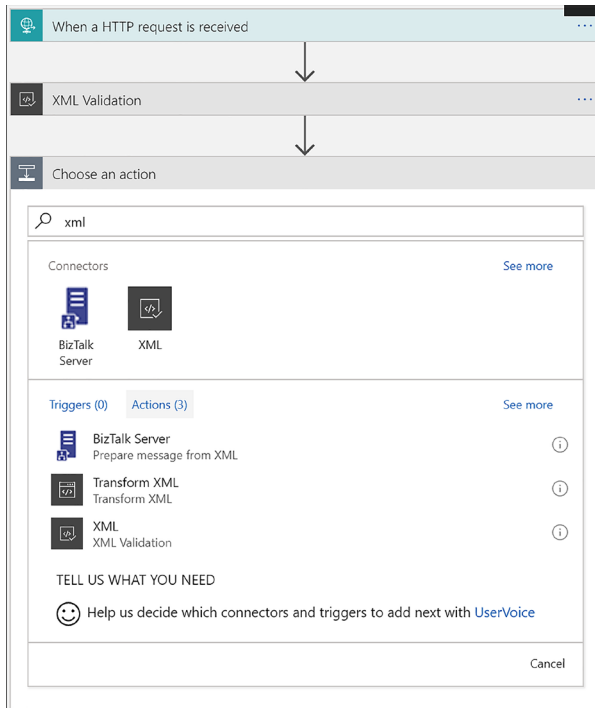
**FIGURE 4-64** Selecting an XML Validation action in the Logic App Designer

6. In the form that follows, select the Body parameter from the HTTP request trigger for the Content value. Select the Order schema in the Schema Name select list, which is the schema we added to the Integration account (Figure 4-65).



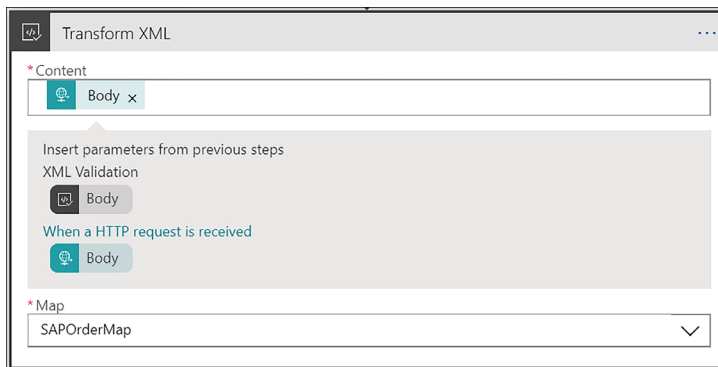
**FIGURE 4-65** Selecting an XML Validation form values in the Logic App Designer

7. Select the + New Step button, and then choose Add An Action.
8. Type **xml** in the search box, and then select Transform XML from the results (Figure 4-66).



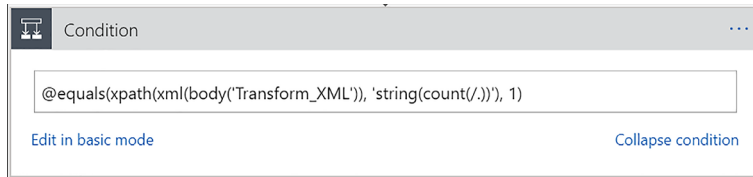
**FIGURE 4-66** Selecting an Transform XML action in the Logic App Designer

9. In the form that follows, select the Body parameter from the HTTP request trigger for the Content value. Select the SAPOrderMap map in the Map select list, which is the map we added to the Integration account (Figure 4-67).



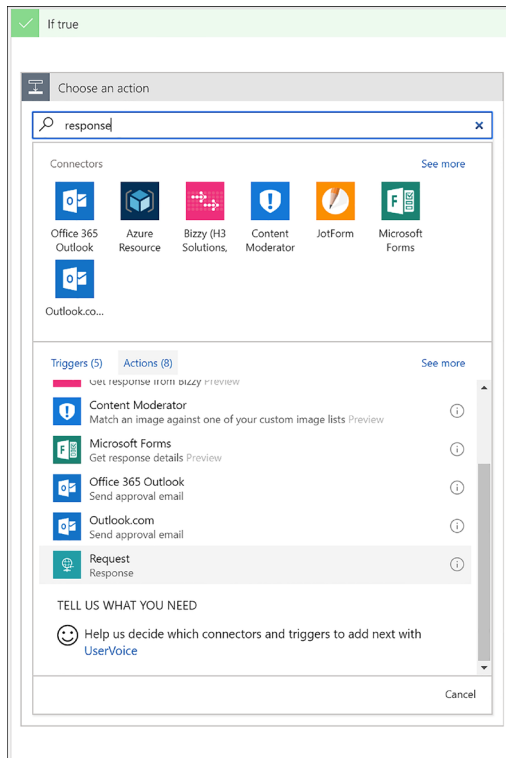
**FIGURE 4-67** Setting the Transform XML form values in the Logic App Designer

10. In the Condition form that appears, select the Edit In Advanced Mode link, and then type in your XPath expression. In our case, we type in the following (Figure 4-68): **@equ**  
**als(xpath(xml(body('Transform\_XML')), 'string(count(/.))', '1')**



**FIGURE 4-68** Setting the XPath expression for the new condition in the Logic App Designer

11. In the “If true” condition block beneath, select Add An Action. Search for “response,” and then select Request (Response) from the resulting list of actions (Figure 4-69).



**FIGURE 4-69** Selecting a Response action for the new condition’s “If true” block in the Logic App Designer

12. In the Response form, select the Transformed XML parameter from the previous Transform XML step. This returns a 200 HTTP response containing the transformed XML (an SAP order) within the body (Figure 4-70).

✓ If true

+

Response

\* Status Code

200

Headers

Enter key Enter value

Body

Transformed X... x

Insert parameters from previous steps

Transform XML

Transformed X...

XML Validation

Body

When a HTTP request is received

Body Headers Path Parameters Queries

Add an action ... More

**FIGURE 4-70** Completing the Response action form for the new condition's "If true" block in the Logic App Designer

13. Click Save in the Logic Apps Designer menu.

#### **MORE INFO** DEPLOY THIS LOGIC APP

Visit the GitHub project page for this Azure Quickstart template to deploy the Logic App in your Azure account at: <https://github.com/Azure/azure-quickstart-templates/tree/master/201-logic-app-veter-pipeline>.

#### **MORE INFO** USING XML CAPABILITIES IN LOGIC APPS

For more information about working with XML capabilities in Logic Apps, see: <https://docs.microsoft.com/azure/logic-apps/logic-apps-enterprise-integration-xml>.

## Trigger a Logic App from another app

There are many triggers that can be added to a Logic App. Triggers are what kick off the workflow within. The most common type of triggers you can use to trigger, or call, your Logic Apps from another app, are those that create HTTP endpoints. Triggers based on HTTP endpoints tend to be more widely used due to the simplicity of making REST-based calls from practically any web-enabled development platform.



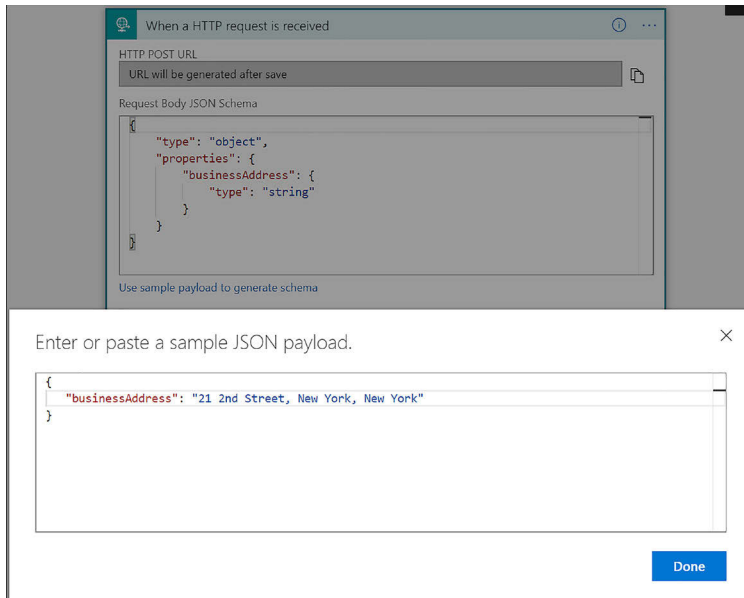
These are the triggers that create HTTP endpoints:

- **Request** Responds to incoming HTTP requests to start the Logic App's workflow in real time. Very versatile, in that it can be called from any web-based application, external webhook events, even from another Logic App with a request and response action.
- **HTTP Webhook** Event-based trigger that does not rely on polling for new items. Register subscribe and unsubscribe methods with a callback URL used to trigger the logic app. Whenever your external app or service makes an HTTP POST to the callback URL, the logic app fires, and includes any data passed into the request.
- **API Connection Webhook** The API connection trigger is similar to the HTTP trigger in its basic functionality. However, the parameters for identifying the action are slightly different.

## Create an HTTP endpoint for your logic app

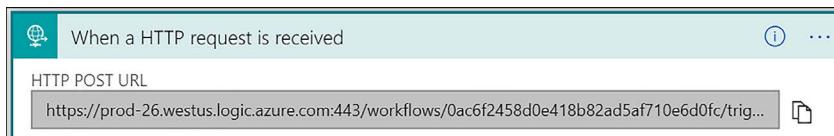
To create an HTTP endpoint to receive incoming requests for a Request Trigger, follow these steps:

1. Open the Logic App Designer on the logic app to which you will be adding an HTTP endpoint.
2. Select Blank Logic App under Templates.
3. Search for "http request" in the trigger filter, and then select Request (When An HTTP Request Is Received) from the list of results.
4. You can optionally enter a JSON schema for the payload, or data, that you expect to be sent to the trigger. This schema can be added to the Request Body JSON Schema field. To generate the schema, select the Use Sample Payload To Generate Schema link at the bottom of the form. This displays a dialog where you can type in or paste a sample JSON payload. This generates the schema when you click Done. The advantage to having a schema defined is that the designer will use the schema to generate tokens that your logic app can use to consume, parse, and pass data from the trigger through your workflow (Figure 4-71).



**FIGURE 4-71** Adding a Request trigger with a request body JSON schema

5. Click Save in the Logic Apps Designer menu.
6. After saving, the HTTP POST URL is generated on the Receive trigger (Figure 4-72). This is the URL your app or service uses to trigger your logic app. The URL contains a Shared Access Signature (SAS) key used to authenticate the incoming requests.



**FIGURE 4-72** The generated HTTP POST URL on the Request trigger

#### **MORE INFO** CALL, TRIGGER, OR NEST WORKFLOWS WITH HTTP ENDPOINTS IN LOGIC APPS

For more information on the topic of using HTTP endpoints to call, trigger, or nest workflows in Logic Apps see: <https://docs.microsoft.com/azure/logic-apps/logic-apps-http-endpoint>.

#### **MORE INFO** CREATE AN API THAT FOLLOWS THE WEBHOOK SUBSCRIBE/UNSUBSCRIBE PATTERN

For more information on how to create an API that follows the webhook subscribe and unsubscribe pattern in logic apps see <https://docs.microsoft.com/azure/logic-apps/logic-apps-create-api-app#webhook-triggers>.

## Create custom and long-running actions

You can create your own APIs that provide custom actions and triggers. Because these are web-based APIs that use REST HTTP endpoints, you can build them in any language framework like .NET, Node.js, or Java. You can also host your APIs on Azure App Service as either web apps or API apps. However, API apps are preferred because they will make it easier to build, host, and consume your APIs used by Logic Apps. Another recommendation is to provide an OpenAPI (previously Swagger) specification to describe your RESTful API endpoints, their operations, and parameters. This makes it much easier to reference your custom API from a logic app workflow because all of the endpoints are selectable within the designer. You can use libraries like Swashbuckle to automatically generate the OpenAPI (Swagger) file for you.

If your custom API has long-running tasks to perform, it is more than likely that your logic app will time out waiting for the operation to complete. This is because Logic Apps will only wait around two minutes before timing out a request. If your long-running task takes several minutes, or hours to complete, you need to implement a REST-based async pattern on your API. These types of patterns are already fully supported natively by the Logic Apps workflow engine, so you don't need to worry about the implementation there.

### **MORE INFO** USE SWASHBUCKLE TO AUTOMATICALLY GENERATE OPENAPI (SWAGGER)

Swashbuckle makes it easy to automatically generate the OpenAPI (Swagger) specification file for you. For more information see <https://github.com/domaindrivendev/Swashbuckle>.

## Long-running action patterns

Your custom API operations serve as endpoints for the actions in your Logic App's workflow. At a basic level, the endpoints accept an HTTP request and return an HTTP response within the Logic App's request timeout limit. When your custom action executes a long-running operation that will exceed this timeout, you can follow either the asynchronous polling pattern or the asynchronous webhook pattern. These patterns allow your logic app to wait for these long-running tasks to finish.

## Asynchronous polling

The way the asynchronous polling pattern works is as follows:

1. When your API receives the initial request to start work, it starts a new thread with the long-running task, and immediately returns an HTTP Response "202 Accepted" with a location header. This immediate response prevents the request from timing out, and causes the workflow engine to start polling for changes.
2. The location header points to the URL for the Logic Apps to check the status of the long-running job. By default, the engine checks every 20 seconds, but you can also add a "Retry-after" header to specify the number of seconds until the next poll.

3. After the allotted time (20 seconds), the engine polls the URL on the location header. If the long-running job is still going, you should return another “202 Accepted” with a location header. If the job has completed, you should return a “200 OK” along with any relevant data. This is what the Logic Apps engine will continue the workflow with.

#### **MORE INFO ASYNCHRONOUS POLLING PATTERN**

For more information on the asynchronous polling pattern see <https://docs.microsoft.com/azure/logic-apps/logic-apps-create-api-app#async-pattern>.

## Asynchronous Webhooks

The asynchronous webhook pattern works by creating two endpoints on your API controller:

- **Subscribe** The Logic Apps engine calls the subscribe endpoint defined in the workflow action for your API. Included in this call is a callback URL created by the logic app that your API stores for when work is complete. When your long-running task is complete, your API calls back with an HTTP POST method to the URL, along with any returned content and headers, as input to the logic app.
- **Unsubscribe** This endpoint is called any time the logic app run is cancelled. When your API receives a request to this endpoint, it should unregister the callback URL and stop any running processes.

#### **MORE INFO ASYNCHRONOUS WEBHOOK PATTERN**

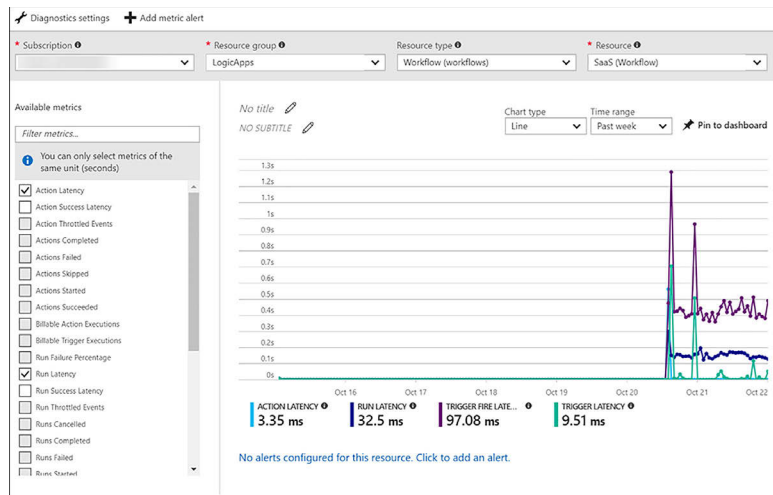
For more information on the asynchronous webhook pattern see <https://docs.microsoft.com/azure/logic-apps/logic-apps-create-api-app#webhook-actions>.

## Monitor Logic Apps

When you create a logic app, you can use out-of-the-box tools within Logic Apps to monitor your app and detect any issues it may have, such as failures. You can view runs and trigger history, overall status, and performance.

If you want real-time event monitoring, as well as richer debugging, you can enable diagnostics on your logic app and send events to OMS with Log Analytics, or to other services, such as Azure Storage and Event Hubs.

Select Metrics (Figure 4-73) under Monitoring in the left-hand menu of your logic app to view performance information and the overall state, such as how many actions succeeded or failed, over the specified time period. It will display an interactive chart based on the selected metrics.



**FIGURE 4-73** Metrics for a logic app

Select Alert Rules under Monitoring to create alerts based on metrics (such as any time failures occur over a 1-hour period), activity logs (with categories such as security, service health, autoscale, etc.), and near real time metrics, based on the data captured by your Logic App's metrics, in time periods spanning from one minute to 24 hours. Alerts can be emailed to one or more recipients, route alerts to a webhook, or run a logic app.

The overview blade of your logic app displays both Runs History and Trigger History (Figure 4-74). This view lets you see at a glance how often the app was called, and whether those operations succeeded. Select a run history to see its details, including any data it received.

Runs history				Trigger History		
<div> <div>All</div> <div>Start time ea...</div> <div>Pick a date</div> <div>Pick a time</div> </div> <div>Specify the run identifier to open monitor view directly</div>				<div> <div>All</div> <div>Start time ea...</div> <div>Pick a date</div> <div>Pick a time</div> </div> <div>When_a_new_tweet_is_posted</div>		
STATUS	START TIME	IDENTIFIER	DURATION	STATUS	START TI...	FIRE
✓ Succeeded...	10/22/2017, 1:22 AM	085869295711096932...	96 Milliseconds	✓ Succeeded	10/22/2...	Fired
✓ Succeeded...	10/22/2017, 1:22 AM	085869295711096932...	88 Milliseconds	✓ Succeeded	10/22/2...	Fired
✓ Succeeded...	10/22/2017, 1:21 AM	085869295717139636...	99 Milliseconds	✓ Succeeded	10/22/2...	Fired
✓ Succeeded...	10/22/2017, 1:21 AM	085869295717139636...	171 Milliseconds	✓ Succeeded	10/22/2...	Fired
✓ Succeeded...	10/22/2017, 1:20 AM	08586929572318705...	95 Milliseconds	✓ Succeeded	10/22/2...	Fired
✓ Succeeded...	10/22/2017, 1:20 AM	08586929572318705...	220 Milliseconds	✓ Succeeded	10/22/2...	Fired
✓ Succeeded...	10/22/2017, 1:20 AM	08586929572318705...	78 Milliseconds	✓ Succeeded	10/22/2...	Fired
✓ Succeeded...	10/22/2017, 1:19 AM	08586929572922095...	160 Milliseconds	✓ Succeeded	10/22/2...	Fired

**FIGURE 4-74** The Runs history and Trigger History of a logic app

### **MORE INFO** MONITOR STATUS AND SET UP DIAGNOSTICS LOGGING FOR LOGIC APPS

To learn more about how to monitor status, set up diagnostics logging, and turn on alerts for Logic Apps see <https://docs.microsoft.com/azure/logic-apps/logic-apps-monitor-your-logic-apps>.

## Skill 4.4: Develop Azure App Service Mobile Apps

Mobile Apps in Azure App Service provides a platform for the development of mobile applications, providing a combination of back-end Azure hosted services with device side development frameworks that streamline the integration of the back-end services.

### This skill covers how to:

- Create a mobile app
- Add authentication to a mobile app
- Add offline sync to a mobile app
- Add push notifications to a mobile app

Mobile Apps enables the development of applications across a variety of platforms, targeting native iOS, Android, and Windows apps, cross-platform Xamarin (Android, Forms and iOS) and Cordova. Mobile Apps includes a comprehensive set of open source SDKs for each of the aforementioned platforms, and together with the services provided in Azure provide functionality for:

- **Authentication and authorization** Enables integration with identity providers including Azure Active Directory, Facebook, Google, Twitter, and Microsoft Account.
- **Data access** Enables access to tabular data stored in an Azure SQL Database or an on-premises SQL Server (via a hybrid connection) via an automatically provisioned and mobile-friendly OData v3 data source.
- **Offline sync** Enables reads as well as create, update, and delete activity to happen against the supporting tables even when the device is not connected to a network, and coordinates the synchronization of data between local and cloud stores as dictated by the application logic (e.g., network connectivity is detected or the user presses a “Sync” button).
- **Push notifications** Enables the sending of push notifications to app users via Azure Notifications Hubs, which in turn supports the sending of notifications across the most popular push notifications services for Apple (APNS), Google (GCM), Windows (WNS), Windows Phone (MPNS), Amazon (ADM) and Baidu (Android China) devices.

## Create a mobile app

From a high level, the process for creating a mobile app is as follows:

1. Identify the target device platforms you want your app to target.
2. Prepare your development environment.
3. Deploy an Azure Mobile App Service instance.
4. Configure the Azure Mobile App Service.
5. Configure your client application.

6. Augment your project with authentication/authorization, offline data sync, or push notification capabilities.

The sections that follow cover each of these steps in greater detail.

## Identify the target device platforms

The first decision you make when creating an mobile app is choosing which device platforms to support. For device platforms, you can choose from the set that includes native Android, Cordova, native iOS (Objective-C or Swift), Windows (C#), Xamarin Android, Xamarin Forms and Xamarin iOS.

Because each device platform brings with it a set of requirements, it can make getting started an almost overwhelming setup experience. One way to approach this is to start with one device platform so that you can complete the end-to-end process, and then layer on additional platforms after you have laid the foundation for one platform. Additionally, if you choose to use Xamarin or Cordova as your starting platform you gain the advantage that these platforms can themselves target multiple device platforms, allowing you to write portable code libraries once that is shared by projects that are specific to each target device.

## Prepare your development environment

The requirements for your development environment vary depending on the device platforms you wish to target. The pre-requisites here include the supported operating system (e.g., macOS, Windows), the integrated development environment (e.g., Android Studio, Visual Studio for Windows, Visual Studio for Mac or Xcode) and the devices (e.g., the emulators/simulators or physical devices used for testing your app from the development environment of your choice).

Table 4-4 summarizes key requirements by device platform.

**TABLE 4-4** Requirements for each target platform

Target Platform	Requirements
Android	OS: macOS or Windows IDE: Android Studio Devices: Android emulator and devices
Cordova	OS: macOS and Windows IDE: Visual Studio for Windows Devices: Android, iOS*, Windows emulators and devices.
iOS	OS: macOS IDE: Xcode Devices: iOS simulator and devices
Windows	OS: Windows IDE: Visual Studio for Windows Devices: Windows desktop and phone
Xamarin.Android	OS: macOS or Windows IDE: Visual Studio for mac or Windows Devices: Android emulators and devices.
Xamarin.Forms	OS: macOS and Windows IDE: Visual Studio for mac or Windows Devices: Android, iOS*, Windows emulators and devices.
Xamarin.iOS	OS: macOS IDE: Visual Studio for mac or Windows Devices: iOS* simulator and devices

*\* Running the iOS simulator or connecting to an iOS device requires a computer running macOS that is reachable across the network from the Windows development computer, or running the indicated IDE on a macOS.*

## Deploy an Azure Mobile App Service

With the aforementioned decisions in place, you are now ready to deploy an Azure Mobile App Service instance to provide the backend services to your app. Follow these steps:

1. In the Azure Portal, select New, and search for Mobile App, and select the Mobile App entry.
2. Select Create.
3. Provide a unique name for your Mobile App.
4. Select an Azure subscription and Resource Group.
5. Select an existing App Service Plan or create a new one.
6. Select Create to deploy the mobile app.

## Configure the mobile app

Once you have deployed your mobile app, you need to configure where it will store its tabular data and the language (your options are C# or Node.js) in which the backend APIs are implemented (which affects the programming language you use when customizing the backend behavior). The following steps walk you through preparing the quick start solution, which you can use as a starting point for your mobile app. Follow these steps:

1. In the Azure Portal, navigate to the blade for your mobile app.
2. From the menu, under the Deployment heading, select Quick Start.
3. On the General listing, select the platform you wish to target first.
4. On the Quick Start blade, select the button underneath the header 1 Connect a database that reads You Will Need A Database In Order To Complete This Quickstart. Click Here To Create One."
5. On the Data Connections blade, select + Add.
6. On the Add Data Connection blade, leave the Type drop-down at SQL Database.
7. Select SQL Database - Configure Required Settings.
8. On the Database blade, select an existing Azure SQL Database, or create a new database (and optionally a new SQL Database Server).
9. Back on the Add Data Connection blade, select Connection String.
10. Provide the name to use for referring to this connection string in configuration.
11. Select OK.
12. Select OK once more to add the data connection (and create the SQL Database if so configured).



13. In a few minutes (when creating a new SQL Database), the new entry appears in the Data Connections blade. When it does, close the Data Connections blade.
14. On the Quick Start blade, underneath the header, Create A Table API, choose Node.js and select the check box I Acknowledge That This Will Overwrite All Site Contents. Then select the Create TodoItem table button that is enabled. If you choose to use C#, note that you will have to download the zip provided, extract it, open it in Visual Studio, compile and then publish the App Service to your Mobile App instance. This is performed in the same way as you deploy Web Apps as described previously.
15. Leave the Quick Start blade open and continue to the next section.

## Configure your client application

Now that you have a basic mobile app backend deployed, you are now ready to create the application that will run on your targeted devices. You can create a new application from a generated quick start project or by connecting an existing application:

1. From the Quick Start blade of your mobile app, underneath the header, Configure Your Client Application, set the toggle to create A New App If You Want To Create A Solution or Connect An Existing App If You Already Have A Solution Built and just need to connect it to the mobile app.
2. If you select Create A New App, you will be provided with instructions specific to the device platform you selected previously as well as a download link from which you can download a generated solution that includes the code customized for access to the deployed mobile app backend. For example, if you selected Xamarin.Forms as your platform, you are provided with a zip file that contains a personalized project that you can open in Visual Studio for Windows or Visual Studio for macOS, which has been pre-configured to connect to your mobile app backend.
3. If you select Connect An Existing App, you are provided with instructions and code you can copy and paste into your project to connect it to the mobile app backend.
4. Once you have completed the steps for either option, you can open and run the project in the IDE and start working against your mobile app backend.

## Add authentication to a mobile app

Once you have your project in place and connected to your mobile app backend, you can enable authentication and authorization. Recall that this enables integration with identity providers including Azure Active Directory, Facebook, Google, Twitter and Microsoft Account such that your app users need to sign in using credentials from one of these providers. To do so, follow these steps.

1. Identify the set of identity providers you want to support.
2. For each identity provider, you need to follow the provider's specific instructions to register your app and retrieve the credentials needed to authenticate using that provider. The up-to-date instructions for each provider are available:

- A.** Azure Active Directory: <https://docs.microsoft.com/en-us/azure/app-service-mobile/app-service-mobile-how-to-configure-active-directory-authentication>
  - B.** Facebook: <https://docs.microsoft.com/en-us/azure/app-service-mobile/app-service-mobile-how-to-configure-facebook-authentication>
  - C.** Google: <https://docs.microsoft.com/en-us/azure/app-service-mobile/app-service-mobile-how-to-configure-google-authentication>
  - D.** Microsoft: <https://docs.microsoft.com/en-us/azure/app-service-mobile/app-service-mobile-how-to-configure-microsoft-authentication>
  - E.** Twitter: <https://docs.microsoft.com/en-us/azure/app-service-mobile/app-service-mobile-how-to-configure-twitter-authentication>
- 3.** Configure authentication / authorization in your mobile app.
  - 4.** Navigate to the blade of your mobile app in the Azure Portal.
  - 5.** From the menu, under the Settings header, select Authentication / Authorization.
  - 6.** Under the Allowed External Redirect URLs header, in the text box provide a callback URL that will be used to invoke your application. It should be of the form [scheme]://easyauth.callback where the value of [scheme] is a string you specify that starts with a letter and consists of only letters and numbers. For example, myapp://easyauth.callback.
  - 7.** Select Save from the command bar.
  - 8.** Restrict permissions to authenticated users on the service side. The approach you take varies depending on how you configured your backend language and if you have deployed custom backend code.
  - 9.** If you are using the Node.js backend created through the quick start in the Azure Portal, you can control access to data on a table-by-table basis. From your Mobile App blade, in the menu select Easy Tables, and then select the table you want to secure. For all of the permission options, set the value to Authenticated Access Only and select Save.
  - 10.** If you deployed a C# backend, in the controller for your project that inherits from TableController, decorate the class with the Authorize attribute. For example:
 

```
[Authorize]

public class TodoItemController : TableController<TodoItem>
```
  - 11.** If you have deployed a customized Node.js backend, you need to modify the code accessing the table and set the access property to authenticated. For example:
 

```
table.access = 'authenticated';
```

**MORE INFO DETAILED STEP BY STEP FOR REQUIRING AUTHENTICATION FOR ACCESS TO TABLES**

Coverage of the implementation details for every platform supported by Mobile Apps is out of scope for this book. To read the implementation details for your particular platform navigate to <https://docs.microsoft.com/en-us/azure/app-service-mobile/app-service-mobile-node-backend-how-to-use-server-sdk#howto-tables-auth> and use the drop-down at the top of the article to select your platform.

12. Add the authentication logic to your app project. The specific steps to take vary based upon the target platform for your app, but in general they amount adding user interface elements to initiate sign-in and handling the authentication events. An important step in the configuration of the authentication is providing the value of your scheme you defined for the Allowed External Redirects URL (e.g., myapp).

**MORE INFO ADDING AUTHENTICATION LOGIC**

For the detailed steps and boilerplate code to use for each platform, see <https://docs.microsoft.com/en-us/azure/app-service-mobile/app-service-mobile-xamarin-forms-get-started-users> and using the drop-down list at the top select your target platform.

13. Run your application in your local simulator or device to verify the authentication flow.

## Add offline sync to a mobile app

The offline data sync capability comes from a mix of client-side SDK and service-side features. This capability enables reads as well as create, update and delete activity to happen against the supporting tables even when the device is not connected to a network, and coordinates the synchronization of data between local and cloud stores as dictated by the application logic (e.g., network connectivity is detected or the user presses a “Sync” button). The feature includes support for conflict detection when the same record is changed on both the client and the backend, and it allows for the conflicts to be resolved on either the client side or service side.

- On the Mobile App service side, you need a table that leverages Mobile App easy tables. This is typically a table in SQL Database that is exposed by Mobile Apps using the OData endpoint. Easy tables can be managed in the Mobile App blade in the portal, including adjusting their schema, setting permissions, and modifying the service side script (for Node.js backends) that processes the create, read, update, delete (CRUD) operations.
- On the client side, the Azure Mobile App SDKs provide an interface referred to as a SyncTable that wraps access to the remote easy table. When using a SyncTable all the CRUD operations work from a local store, whose implementation is device platform specific. The local store provides the data persistence capability on the client device. In iOS

the local store is based on Core Data, and for Windows, Xamarin, and Android the local store is based on SQL lite.

Changes to the data are made through a sync context object that tracks the changes that are made across all of the tables. This sync context maintains an operation queue that is an ordered list of create, update and delete operations that have been performed against the data locally.

- To modify the backend table data with the changes performed against the local store, you have to perform a push. To populate the local store with data from the backend, you have to perform a pull. A push operation executes a series of REST calls to your mobile app backend that applies all the CUD changes since the last push. It's important to note that when you push changes, you are always pushing a set containing at least one operation; you are not pushing a specific table. This restriction ensures that multiple operations against the context that may span across multiple tables are replayed against the backend table in the correct order.
- There is a notion of an implicit push; this occurs when you execute a pull operation but have pending operations to push. In this case, the pull will first execute a push against the sync context.
- Offline sync supports incremental sync, whereby each time you pull records from the source only the source records that are new or have changed are retrieved (as opposed to downloading the entire table worth of data every time). You can clear the contents of the local store by performing a purge.

You can enable Offline Sync by following these high-level steps:

1. Modify the client code that accesses your easy tables to use objects of the SyncTable variety.
2. Implement a method that is run when your application first launches that defines the table schema and initializes the local store with data from the remote table.
3. Implement a method that launches initiate sync operation. This could be triggered from a button or refresh gesture.
4. You can test the offline behavior of your app by:
5. Running the application once as normal and adding data to your table.
6. Modifying the application's configuration so that it no longer points to the correct URI of your mobile app backend.
7. Run the application again. This time the offline behavior should take affect. Make some modifications to the data.
8. Restore the application's configuration.
9. Run the application again and verify that the changes you made while offline appear in your easy table. To do this, navigate to the blade of your mobile app, select Easy Tables from the menu, and then select your table to view its contents.

#### **MORE INFO** ADDING OFFLINE SYNC LOGIC

Coverage of the implementation details of Offline Sync for every platform supported by Mobile Apps is out of scope for this book. To read the implementation details for your particular platform navigate to <https://docs.microsoft.com/en-us/azure/app-service-mobile/app-service-mobile-xamarin-forms-get-started-offline-data> and use the dropdown at the top of the article to select your platform.

## Add push notifications to a mobile app

Push notifications enable you to send app-specific messages to your app running across a variety of platforms. In Azure Mobile Apps, push notification capabilities are provided by Azure Notification Hubs, which is accessed using the Mobile Apps SDKs for the platform of choice. Notification Hubs, in turn, abstract your application from the complexities of dealing with the various push notification systems (PNS) that are specific to each platform, which includes challenges like device registration with the PNS, backend services to send messages to the PNS, and provides for routing of messages to targeted users or groups of users (which requires maintaining a mapping of users to devices), and scaling to support such functions across a huge base of devices. Notifications Hubs supports the sending of notifications across the most popular push notifications services for Apple (APNS), Google (GCM), Windows (WNS), Windows Phone (MPNS), Amazon (ADM), and Baidu (Android China) devices.

To add push notifications, follow these steps:

1. Deploy a Notification Hub with your mobile app.
2. Navigate to the blade of your mobile app, and on the menu under the Settings heading, select Push.
3. From the Command bar, select Connect.
4. On the Notification Hub blade, choose an existing Notification Hub or provision a new one. If you choose to provision a new Notification Hub, provide a name for the hub, a name for the new namespace, and select the desired pricing tier, and then select OK.
5. Select the link Configure Push Notification Services.
6. On the Push Notification Services blade, select the PNS to which you want to connect the Notification Hub.
7. On the blade for the PNS, enter the PNS specific configuration, and select Save.
8. Configure your backend server project to send push notifications.

#### **MORE INFO** SENDING PUSH NOTIFICATIONS FROM THE SERVER SIDE

Coverage of the implementation details of sending push notifications for every platform supported by Mobile Apps is out of scope for this book. To read the implementation details for your particular platform navigate to <https://docs.microsoft.com/en-us/azure/app-service-mobile/app-service-mobile-xamarin-forms-get-started-push#update-the-server-project-to-send-push-notifications> and use the drop-down at the top of the article to select your platform.

9. Modify the app project to respond to push notifications.

#### **MORE INFO RECEIVING PUSH NOTIFICATIONS IN THE CLIENT APP**

Coverage of the implementation details of receiving push notifications for every platform supported by Mobile Apps is out of scope for this book. To read the implementation details for your particular platform navigate to <https://docs.microsoft.com/en-us/azure/app-service-mobile/app-service-mobile-xamarin-forms-get-started-push#configure-and-run-the-android-project-optional> and use the drop-down at the top of the article to select your platform.

## **Skill 4.5: Implement API Management**

---

Azure API Management is a turnkey solution for publishing, managing, securing, and analyzing APIs to both external and internal customers in minutes. You can create an API gateway for back-end services hosted anywhere, not just those hosted on Azure. Many modern APIs protect themselves by rate-limiting consumers, meaning, limiting how many requests can be made in a certain amount of time. Traditionally, there is a lot of work that goes into that process. When you use API Management to manage your API, you can easily secure it and protect it from abuse and overuse with an API key, JWT validation, IP filtering, and through quotas and rate limits.

If you have several APIs as part of your solution, and they are hosted across several services or platforms, you can group them all behind a single static IP and domain, simplifying communication, protection, and reducing maintenance of consumer software due to API locations changing. You also can scale API Management on demand in one or more geographical locations. Its built-in response caching also helps with improving latency and scale.

Hosting your APIs on the API Management platform also makes it easier for developers to use your APIs, by offering self-service API key management, and an auto-generated API catalog through the developer portal. APIs are also documented and come with code examples, reducing developer on-boarding time using your APIs.

API Management is made up of the following components:

- The *API gateway* is the endpoint that:
  - Accepts API calls and routes them to your backends.
  - Verifies API keys, JWT tokens, certificates, and other credentials.
  - Enforces usage quotas and rate limits.
  - Transforms your API on the fly without code modifications.
  - Caches backend responses where set up.
  - Logs call metadata for analytics purposes.

- The *publisher portal* is the administrative interface where you set up your API program. Use it to:
  - Define or import API schema.
  - Package APIs into products.
  - Set up policies like quotas or transformations on the APIs.
  - Get insights from analytics.
  - Manage users.
- The *developer portal* serves as the main web presence for developers, where they can:
  - Read API documentation.
  - Try out an API via the interactive console.
  - Create an account and subscribe to get API keys.
  - Access analytics on their own usage.

**This skill covers how to:**

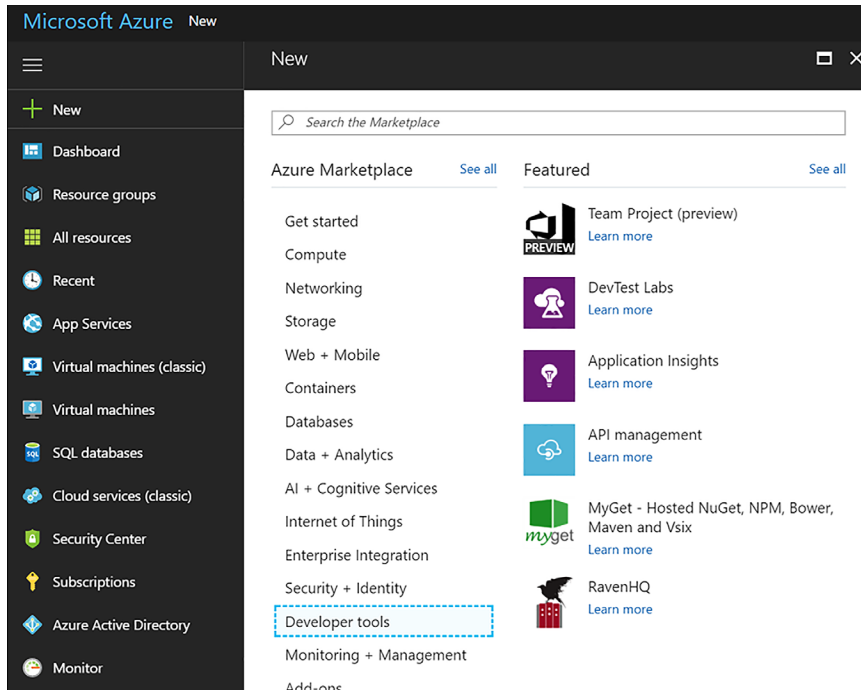
- Create managed APIs
- Configure API management policies
- Protect APIs with rate limits
- Add caching to improve performance
- Monitor APIs
- Customize the Developer Portal

## Create managed APIs

The API Management service is the platform on which the API gateway, publisher portal, and developer portal are hosted. As such, before you can create APIs, you must first create a service instance.

### Create an API Management service

1. Navigate to the portal accessed via <https://portal.azure.com>.
2. Select New on the command bar.
3. Select Developer Tools, and then API Management (Figure 4-75).



**FIGURE 4-75** Creating a new API Management service instance from the Azure Portal

4. Provide a unique name, select a resource group and location, enter an organization name that will appear on the developer portal and emails, an administrator email, your pricing tier, select Pin To Dashboard, and then click Create.

## Add a product

Before you can publish an API, it needs to be added to a product. A product in API Management contains one or more APIs, as well as constraints such as a usage quota and terms of use. This is a great way to add API access levels, like starter (limit to five calls/minute) or unlimited. You can create several products to group APIs with their own usage rules. Developers can subscribe to a product once it is published, and then begin using its APIs.

Follow these steps to add and publish a new product:

1. Navigate to your API Management service on the portal.
2. Select Publisher Portal on the top of the overview blade.
3. Select Products on the left-hand menu, and then click Add Product.
4. Within the new product form, provide a Title, which should be a descriptive name for your product that appears on the developer and admin portals. Provide a Description that explains the product's purpose and any other information you want to display. The remaining fields allow you to set your level of protection, meaning, whether your product requires a subscription, and if so, whether the subscription needs to be approved by



an administrator, and whether developers can subscribe more than once. Once finished, click Save.

5. Once the product has been added, you need to add one or more APIs to it before you can publish it. Select a product, and then click the Add API To Product link. This gives you a list of APIs that you can assign to the product.

## Create a new API

1. Navigate to your API Management service on the portal.
2. Select Publisher Portal on the top of the overview blade.
3. Select APIs on the left-hand menu, and then click Add API.
4. Within the new product form (Figure 4-76):
  - A. Provide a unique Web API Name, which should be a descriptive name for your API that appears on the developer and publisher portals.
  - B. Enter the Web Service URL, which is the HTTP endpoint for your API.
  - C. Enter the Web Service URL suffix, which is unique to your API, and is the last part of the API's public URL.
  - D. Select the desired Web API URL Scheme (HTTP or HTTPS (default)).
  - E. Select the product you created and any others you want to add it to.
  - F. When finished, click OK.

**Add new API**

Web API name

Web service URL

Web API URL suffix

Web API URL scheme  
☐ HTTP ☒ HTTPS

This is what the URL is going to look like:  
**https://api-this.azure-api.net/contoso-contacts**

Products (optional)

**Save** **Cancel**

Public name of the API as it would appear on the developer and admin portals.

A URL of the web service exposing the API. This URL will be used by Azure API Management only, and will not be made public.

Last part of the API's public URL. This URL will be used by API consumers for sending requests to the web service.

Add this API to one or more existing products.

**FIGURE 4-76** Completing the Response action form for the logic app

## Add an operation to your API

Before you can use your new API, you must add one or more operations. These operations do things like enable service documentation, the interactive API console, set per operation limits, set request/response validation, and configure operation-level statistics.

1. Navigate to your API Management service on the portal.
2. Select Publisher Portal on the top of the overview blade.
3. Select APIs on the left-hand menu, select your API from the list, and then select the Operations tab.
4. Click + Add Operation.
5. By default, the Signature tab will be selected. The Signature is the URL template used to send requests to the underlying API. Here you select (Figure 4-77):
  - A. The HTTP verb (GET, POST, etc.).
  - B. Type in the URL template (e.g. /contacts/{id}).
  - C. Type in a display name, and description.
  - D. You can also add a rewrite URL template to call the back-end with a converted URL.

The screenshot shows the 'New operation' form with the 'Signature' tab selected. The form is divided into several sections: 'Signature', 'Caching', 'REQUEST', 'Parameters', and 'RESPONSES'. In the 'Signature' section, the 'HTTP verb\*' is set to 'GET' and the 'URL template\*' is '/contacts/{id}'. Below these fields is an example URL: 'Example: customers/{cid}/orders/{oid}/?date={date}'. There is also a 'Rewrite URL template' field which is currently empty. A note below this field states: 'When specified, rewrite template will be used to make requests to the web service. It can contain all of the template parameters specified in the original template.' In the 'RESPONSES' section, 'Code 200' is selected, and 'Code 404' is also visible with an 'ADD' button next to it. The 'Display name\*' is '/contacts' and the 'Description' is 'Returns all contacts.' A 'Save' button is located at the bottom right of the form.

**FIGURE 4-77** Adding a new operation to a managed API

6. Select the Parameters tab. New query parameters are automatically generated based on the URL template defined in the signature. In our case, an id template parameter was generated because the URL template of our signature for this operation is /contacts/{id}. Specify the type (string, number, etc.) and provide a description for each query parameter (Figure 4-78).

NAME*	DESCRIPTION	TYPE	VALUES
id	The contact Id	number	

**Query parameters**  
 + ADD QUERY PARAMETER

**FIGURE 4-78** URL template parameters

7. You can optionally use the other tabs to specify caching and responses for the operation. Click Save when finished.

## Publish your product to make your API available

The last step to making your API available to other developers is to publish your product to which this and any other APIs have been added.

To publish your product, follow these steps:

1. Navigate to your API Management service on the portal.
2. Select Publisher portal on the top of the overview blade.
3. Select Products on the left-hand menu, and then click select your product from the list.
4. The summary tab will indicate whether your product has been published, and any associated APIs. You must have at least one API added before you can publish. Click the Publish link.
5. When the confirmation appears, click Yes, and then publish it.
6. After publishing, select the Visibility tab. Choose which roles, such as developers, you want to be able to see the product on the developer portal and subscribe to the product. Click Save when finished.

### **MORE INFO** ADD AND PUBLISH AN API PRODUCT

To learn more about creating and publishing a product in API Management see <https://docs.microsoft.com/azure/api-management/api-management-howto-add-products>.

## Configure API Management policies

API Management policies allow you, as the publisher, to determine the behavior of your APIs through configuration, requiring no code changes. You define a policy definition, which is a collection of statements that are executed sequentially on the request or response of your API. There are many policies you can select from, such as whether to allow cross domain calls, how to authenticate requests, find and replace strings in the body, setting rate limits, and many more.

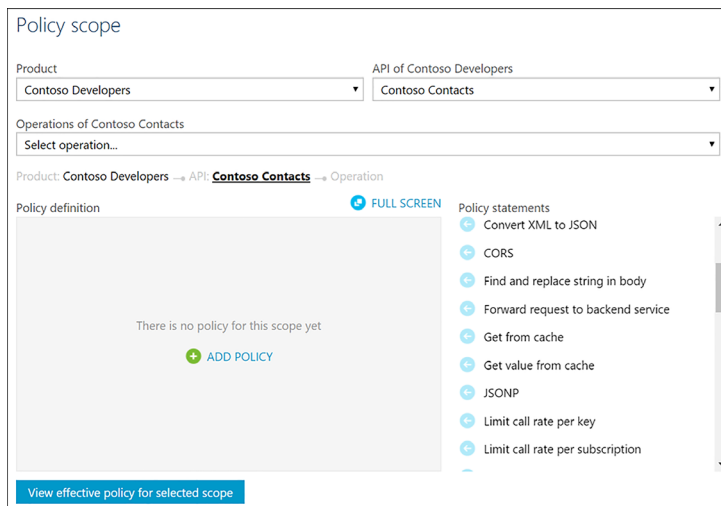
## MORE INFO FULL LIST OF POLICY STATEMENTS

See the Policy Reference for a full list of policy statements and their settings at <https://docs.microsoft.com/azure/api-management/api-management-policy-reference>.

Because the API gateway receives all requests to your APIs, the policies you defined are applied at this level. The policies statements you choose affect both inbound requests and outbound responses. Policies can be applied globally, or scoped to the Product, API, or Operation level.

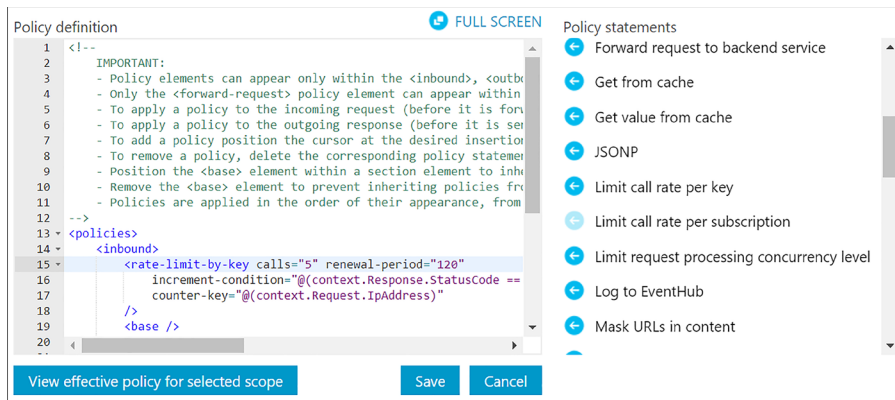
To configure a policy, follow these steps:

1. Navigate to your API Management service on the portal.
2. Select Publisher Portal on the top of the overview blade.
3. Select Policies on the left-hand menu.
4. At the top of the policies page, you will find select lists to define the policy scope at the Product, API, and Operations levels. If you do not select a specific operation, all operations are included in this policy. To create a policy scoped globally, simply deselect any options from these select lists (Figure 4-79).



**FIGURE 4-79** Policies page for an API Management service in the Publisher portal

5. To add a new policy to the selected policy scope, select + Add Policy link in the Policy definition area.
6. The policy definition will appear in XML format. To add an inbound policy that limits the call rate per key, place your cursor just inside the content of the inbound XML element, and then click the Limit Call Rate Per Key policy statement on the right. This adds the statement to rate limit inbound requests to the number of calls you specify within your defined period of time in seconds, and any other conditions you desire (Figure 4-80).



**FIGURE 4-80** Editing the policy definition for an API Management service in the Publisher portal

7. When you are finished, click Save. Your changes will be immediately applied to the API Management gateway.

#### **MORE INFO** APPLYING POLICIES IN API MANAGEMENT

For more information about how to apply policies in API Management see: <https://docs.microsoft.com/azure/api-management/api-management-howto-policies>.

## Protect APIs with rate limits

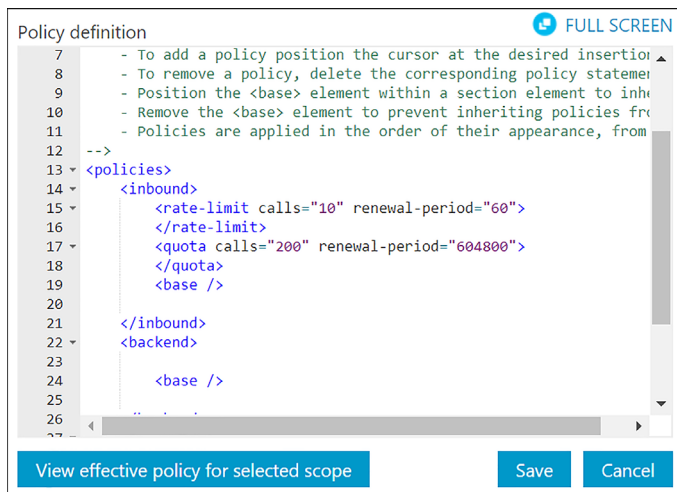
Protecting your published APIs by throttling incoming requests is one of the most attractive offerings of API Management. When you open up your API for others to use, it is difficult to guarantee a promised level of service if you cannot control the demand on your resources. Or, you may be interested in controlling your resource costs by limiting requests, preventing you from unnecessarily scaling up your services to meet unexpected demand. Rate limiting, or throttling, is common practice when providing APIs. Oftentimes, API publishers offer varying levels of access to their APIs. For instance, you may choose to offer a free tier with very restrictive rate limits, and various paid tiers offering higher request rates. This is where API Management's products come into play. Define products for your varying service levels, and apply rate limiting policies to each product, accordingly.

### Create a product to scope rate limits to a group of APIs

The following steps show how to create a free trial, adding APIs that developers can use on a rate-limited free trial basis:

1. Navigate to your API Management service on the portal.
2. Select Publisher Portal on the top of the Overview blade.
3. Create a new product named Free Trial.

4. Set the description to Subscribers Will Be Able To Run 10 Calls/Minute Up To A Maximum Of 200 Calls/Week.
5. Set the visibility to Developers.
6. Add your APIs to the product and publish it.
7. Go to Policies and set the policy scope to the free trial product.
8. Click + Add Policy.
9. Position the cursor within the inbound element.
10. Scroll through the list of policy statements and select Limit Call Rate Per Subscription. Modify the XML to set calls to 10 and renewal-period to 60. You can delete the API and operation elements because they are not needed in this scenario.
11. Position your cursor immediately below the rate-limit element you added. Select Set Usage Quota Per Subscription in the list of policy statements. Modify the XML to set calls to 200 and renewal-period to 604800. You can delete the API and operation elements because they are not needed in this scenario.
12. Save your changes. In the end, your inbound policy should look as follows (Figure 4-81):



**FIGURE 4-81** Editing the policy definition to set rate limits on a product

## Advanced rate limiting

In its simplest implementation, you can control the rate of requests or the total requests/data transferred. These constraints do not help when individual end-users of your API consume exponentially more of the quota than other users. If you want to avoid having high-usage consumers limit access to occasional users, by using up the pool of available resources, consider using the new rate-limit-by-key and quota-by-key policies. These are more flexible rate limit-

ing policies that allow you to define expressions to track traffic usage by user-level information, such as IP address and user identity.

Here is an example of rate and quota limiting by IP address:

```
<rate-limit-by-key calls="10"
    renewal-period="60"
    counter-key="@Context.Request.IpAddress)" />

<quota-by-key calls="1000000"
    bandwidth="10000"
    renewal-period="2629800"
    counter-key="@Context.Request.IpAddress)" />
```

#### **MORE INFO ADVANCED RATE LIMITING**

For more information about advanced rate limiting through flexible request throttling see <https://docs.microsoft.com/azure/api-management/api-management-sample-flexible-throttling>.

## Add caching to improve performance

Caching is a great way to limit your resource consumption, like bandwidth, as well as reduce latency for infrequently changing data. API Management allows you to configure response caching on operations.

Follow these steps to add response caching for your API (Figure 4-82), and review caching policies:

1. Navigate to your API Management service on the portal.
2. Select Publisher portal on the top of the overview blade.
3. Select APIs on the left-hand menu.
4. Select the ECHO API, which is automatically added to new API Management services.
5. Select the Operations tab, and then select GET Retrieve Resource (Cached) from the list.

Summary

Settings

Operations

Security

## Operations

Define service operations to enable service documentation, and operation-level statistics.

+

ADD OPERATION

POST	Create resource
PUT	Modify Resource
DELETE	Remove resource
HEAD	Retrieve header only
GET	Retrieve resource
GET	Retrieve resource (cached)

**FIGURE 4-82** The API operations tab

- Select the Caching tab (Figure 4-83) to view the caching settings. To enable caching on an operation, select the Enable check box. You can modify the keyed operation responses by setting values in the Vary By Query String Parameters and Vary By Headers fields. In this case, cache keys are being computed on two different headers: Accept and Accept-Charset. Duration sets the cache duration in seconds. Here it is set to 3600 seconds.

Operation - Retrieve resource (cached)

Signature

Caching

REQUEST

Parameters

RESPONSES

Code 200

+

ADD

Caching

Response caching can significantly reduce API latency, bandwidth consumption, and web service load.

Enable

Vary by query string parameters

Semicolon-delimited list of parameter names used to compute cache keys. If none specified, complete URLs are used as keys

Vary by headers

AcceptAccept-Charset

Semicolon-delimited list of header names used to compute cache keys.

Duration

3600

Cache duration (TTL) in seconds.

**FIGURE 4-83** Caching settings for the GET operation of the Echo API

Skill 4.5: Implement API Management **CHAPTER 4** **361**



7. Select Policies from the left-hand menu of the publisher portal.
8. Select Echo API from the API select list, and then Retrieve Resource (Cached) from the Operation select list.
9. Here you see that the caching policies in the policy editor reflect the values in the Caching tab of the operation. Any changes here are reflected on the Caching tab, and vice-versa.

#### **MORE INFO CUSTOM CACHING IN API MANAGEMENT**

To learn how to implement custom caching see <https://docs.microsoft.com/azure/api-management/api-management-sample-cache-by-key>.

## Monitor APIs

API Management provides a few methods by which you can monitor resource usage, service health, activities, and analytics. If you want real-time monitoring, as well as richer debugging, you can enable diagnostics on your logic app and send events to OMS with Log Analytics, or to other services, such as Azure Storage, and Event Hubs. Select Diagnostics Logs from the left-hand menu of your API Management service, and then select Turn On Diagnostics to archive your gateway logs and metrics to a storage account, stream to an event hub, or send to Log Analytics on OMS.

Activity logs provide insight into the operations that were performed on your API Management services, so you can determine the “what, who, and when” for any write operations taken on your API Management services. Select Activity Log from the left-hand menu to filter and view these logs. From here, you can select Export to archive these logs in a storage account or send them to an event hub. You can also select Log Analytics to send the logs to OMS.

- Select Metrics under Monitoring in the left-hand menu of your API Management service to view the state and health of your APIs in near real-time. These metrics are emitted every minute. You can monitor gateway requests, determine which of those were successful or failed, and also view unauthorized gateway requests. It displays an interactive chart based on the selected metrics.
- Select Alert rules under Monitoring to create alerts based on metrics (such as any time failed gateway requests occur over a one-hour period), activity logs (with categories such as security, service health, autoscale, etc.), and near real time metrics, based on the data captured by your API Management service’s metrics, in time periods spanning from one minute to 24 hours. Alerts can be emailed to one or more recipients, route alerts to a webhook, or run a logic app.

Open the publisher portal to view Analytics. This shows an overview of usage by developers, top products, top subscriptions, top APIs, and top operations. Each of these categories show the number of successful calls versus blocked or failed calls, as well as bandwidth used and average response time, when applicable. The usage tab shows number of calls and bandwidth

by region, highlighting countries on a map, corresponding with the origin of the requests. You can select any continent or country to drill down further into the selected region. The health tab shows statistics about status codes, caching, API response time, and Service response time. Finally, the activity tab shows more detailed information about requests by developers, on products, by subscriptions, for APIs, and on which operations.

**MORE INFO MONITOR API MANAGEMENT**

To learn more about how to monitor an API Management service see <https://docs.microsoft.com/azure/api-management/api-management-howto-use-azure-monitor>.

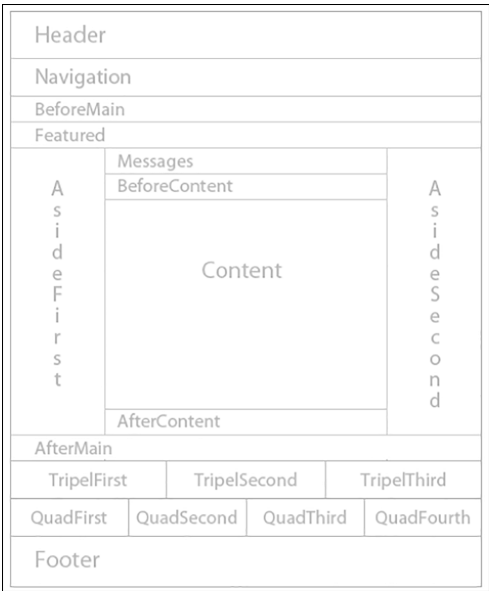
## Customize the developer portal

The API Management developer portal is built on top of a content management system (CMS), which gives you flexibility on ways you can customize its layout, content, and styles. Because this is the portal through which developers discover, subscribe to, and learn more about your APIs, you may wish to alter the look and feel to more closely match your company's website, or craft the experience for your end users in general.

There are three different methods by which you can customize the developer portal.

### Edit static page content and layout elements

The layout of every page of the developer portal is based on small page elements called widgets (Figure 4-84).



**FIGURE 4-84** The widget layout of the developer portal

The content area on the page is specific to an individual page's contents. Any Contents widget can be edited to modify that page's content. The page layout elements are comprised of the remaining widgets. Any edits made to these layout widgets are applied to all pages within the portal.

To edit the contents of a layout widget, perform the following steps:

1. Navigate to your API Management service on the portal.
2. Select Publisher portal on the top of the overview blade.
3. Select Widgets on the left-hand menu, underneath the DEVELOPER PORTAL section.
4. Select the widget you wish to edit, such as Banner.
5. The Edit Widget form allows you to select the zone for the widget, layer, position, title, name (used for CSS), and its HTML.
6. Make changes as desired, and then click Save. You immediately see your changes on the developer portal.

To edit the contents of a page, perform the following steps:

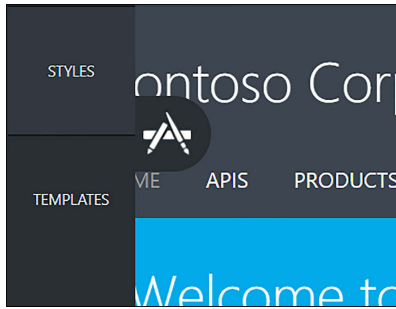
1. Navigate to your API Management service on the portal.
2. Select Publisher portal on the top of the overview blade.
3. Select Content on the left-hand menu, underneath the DEVELOPER PORTAL section.
4. Select the page you wish to edit, such as Welcome.
5. The Edit Page form allows you change the page title, select whether you wish to display the title on the front-end, and its HTML.
6. Make changes as desired, and then click Save. When you are satisfied with your changes, click Publish Now to make those changes visible to everyone. You immediately see your changes on the developer portal.

Using these tools, you can add new layout widgets, as well as new pages. Use the Navigation area to create custom menu links or rearrange their order.

## Customize the styling

Change the colors, fonts, spacing, and other styles by altering the style rules in the developer portal. For instance, change the colors and fonts to match your company's website. To change these style rules, you need to be logged in to the developer portal as an administrator. This requires opening the developer portal from the publisher portal.

1. Navigate to your API Management service on the portal.
2. Select Publisher portal on the top of the overview blade.
3. Select Developer portal from the top-right of the page.
4. On the developer portal, hover your mouse over the customization icon to display the customization toolbar (Figure 4-85), and then select Styles from the toolbar.



**FIGURE 4-85** The customization toolbar in the developer portal

5. In the list of editable styles that appear, you can either look through the list and change style values as you see fit, or click the Select An Element On The Page button, and then select any element on the page to view only its styles.
6. When you are finished making edits, click the Publish button at the bottom of the customization toolbar. This will show a preview of your changes. When satisfied, click the Publish Customizations button to make your changes publicly available.

## Customize using templates

Use templates to customize the system-generated developer pages, such as API docs, user authentication, products, etc. Template markup uses the DotLiquid syntax, based on Ruby's Liquid markup, to alter the appearance and behavior of the corresponding page. Dynamic content in the template is controlled through tokenized strings. When you select a template to edit, there are three panes that are displayed. The top pane is a preview of the corresponding page. On the bottom left is the template editing pane where you edit the markup, and on the bottom right is the template data pane. This pane serves as a guide to the data model for the entities available in the selected template. You can reference the template data when adding tokenized strings to the template beside it.

To edit templates, follow these steps:

1. Navigate to your API Management service on the portal.
2. Select Publisher portal on the top of the overview blade.
3. Select Developer portal from the top-right of the page.
4. On the developer portal, hover your mouse over the customization icon to display the customization toolbar, and then select Templates from the toolbar.
5. Select the template you wish to edit from the list.
6. Alter the template markup, using the bottom-left template editing pane. Here you can use a mix of HTML and tokenized strings. Reference the template data to the right to view tokenized strings you can add to the template, and the values they will display if you reference them. All changes will update the preview pane on top in real time.
7. When finished editing, click the save icon in the template editing pane.

8. Saved templates can be published either individually, or all together. To publish an individual template, click Publish in the template editor.
9. Click Yes to confirm and make your changes to the template live on the developer portal.

**MORE INFO EDIT STATIC PAGE CONTENT AND LAYOUT ELEMENTS**

To learn more about editing static page content and layout elements on the developer portal see <https://docs.microsoft.com/azure/api-management/api-management-modify-content-layout>.

**MORE INFO CUSTOMIZE THE STYLING**

For more information on how customize the styling of the developer portal, see <https://docs.microsoft.com/azure/api-management/api-management-customize-styles>.

**MORE INFO CUSTOMIZE USING TEMPLATES**

For more information on how to customize the developer portal using templates see <https://docs.microsoft.com/azure/api-management/api-management-developer-portal-templates>.

## Skill 4.6: Implement Azure Functions and WebJobs

---

Azure Functions is a serverless compute service that enables you to run code on-demand without having to explicitly provision or manage infrastructure. Use Azure Functions to run a script or piece of code in response to a variety of events from sources such as:

- HTTP requests
- Timers
- Webhooks
- Azure Cosmos DB
- Blob
- Queues
- Event Hub

When it comes to implementing background processing tasks, the main options in Azure are Azure Functions and WebJobs. It is important to mention, however, that Functions are actually built on top of WebJobs. The choice to use one or the other really depends on the problem you are trying to solve. For example, if you already have an app service running a website or a web API and you require a background process to run in the same context, a WebJob makes the most sense. Here are two examples that may drive you to using a WebJob:

- **The Service Plan** You want to share compute resources between the website or API and the WebJob.
- **Shared libraries** The WebJob should share libraries that run the website or API.

Otherwise, for situations where you want to externalize a process so that it runs and scales independently from your web application or API environment, or you are implementing an event handler in response to some external event (i.e., a Webhook); Azure Functions are the more modern serverless technology to choose.

#### **MORE INFO** AZURE FUNCTIONS

For a general references on Azure Functions see <https://docs.microsoft.com/en-us/azure/azure-functions/>.

#### **This skill covers how to:**

- Create Azure Functions
- Implement a webhook function
- Create an event processing function
- Implement an Azure-connected function
- Integrate a Function with storage
- Debug a Function
- Design and implement a custom binding
- Implement and configure proxies
- Integrate with App Service Plan

## Create Azure Functions

The Azure portal gives you a quick and easy way to create a functions app, add functions based on a template and test the function.

#### **NOTE** VISUAL STUDIO 2017

You can also develop, test, and publish functions using Visual Studio 2017.

To create a function app in the portal follow these steps (Figure 4-86):

1. Navigate to the portal accessed via <https://portal.azure.com>.
2. Select New on the command bar.
3. Select Compute, and then Function App.
4. Click Create and supply the app name, subscription, resource group, hosting plan, location, and storage plan (if you select Consumption plan).

## NOTE CONSUMPTION PLANS

Consumption plan means that resources are added dynamically as required by your function.

The screenshot shows the 'Function App Create' blade with the following configuration:

- App name:** sol-newfunctionapp (with a green checkmark and .azurewebsites.net suffix)
- Subscription:** Microsoft Azure Sponsorship
- Resource Group:** sol-functionsskill (with a green checkmark)
- Hosting Plan:** Consumption Plan
- Location:** South Central US
- Storage:** solnewfunctionapp (with a green checkmark)
- Application Insights:** Off

FIGURE 4-86 The Create Function App blade

5. After a few minutes, the Functions App is created (Figure 4-87).

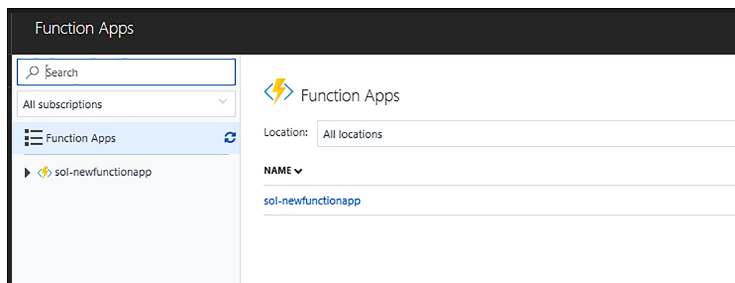


FIGURE 4-87 A new function app

## MORE INFO CREATING FUNCTIONS WITH AZURE CLI

You can also create functions using Azure CLI and from Visual Studio. See these references at: <https://docs.microsoft.com/en-us/azure/azure-functions/functions-create-first-azure-function-azure-cli> and <https://docs.microsoft.com/en-us/azure/azure-functions/functions-create-your-first-function-visual-studio>.

# Implement a Webhook function

Visual Studio provides a complete development and debugging environment for Azure Functions with the addition of Azure Functions Extension. To create a Webhook function using Visual Studio 2017, follow these steps:

1. Ensure you have the Functions App Visual Studio Extension installed first (Figure 4-88).

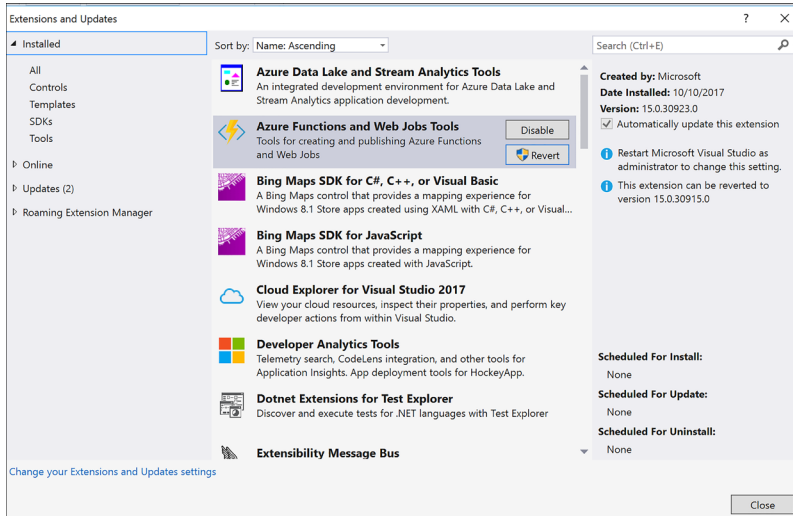


FIGURE 4-88 Azure Functions and WebJobs Tools

2. In the New Project dialog, expand Visual C# > Cloud node, select Azure Functions, type a Name for your project, and click OK (Figure 4-89).

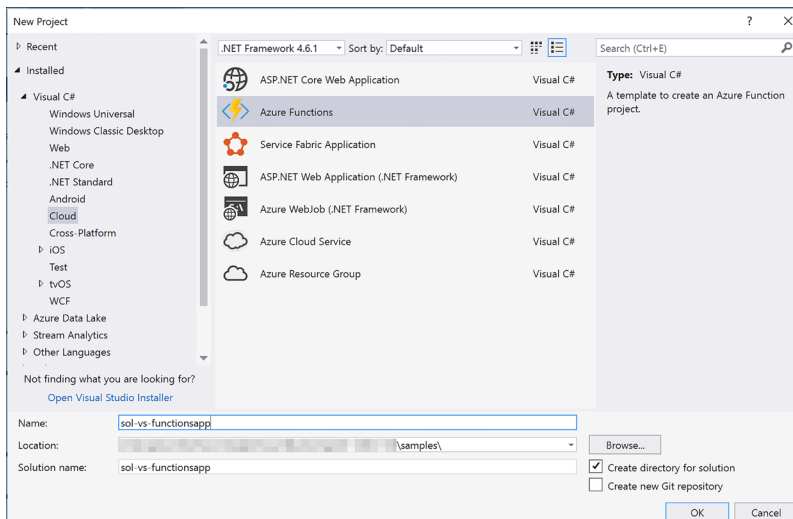
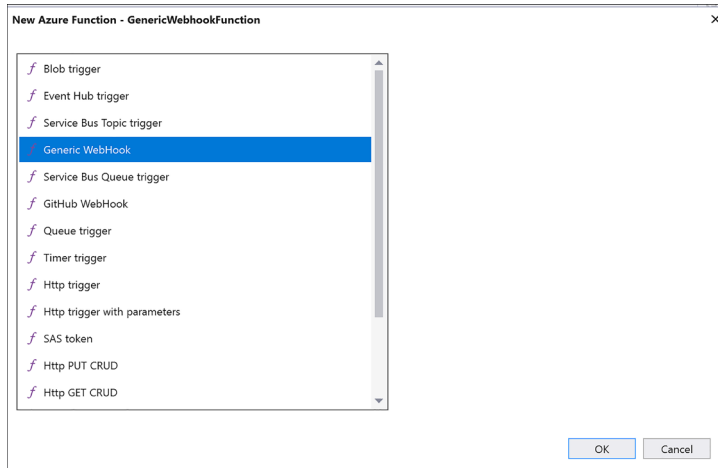


FIGURE 4-89 Selecting Azure Functions from the New Project dialog



3. This creates a new Functions App in your subscription. You may have to log in to the Azure portal to complete the process.
4. From Visual Studio, go to Solution Explorer, right-click the project node, and select Add > New Item. Select Azure Function, and click Add.
5. From the New Azure Function dialog, select Generic WebHook, type the function name, and click OK (Figure 4-90).



**FIGURE 4-90** Selecting the type of Azure Function

6. This generates an initial implementation for your function. The `FunctionName` attribute sets the name of your function. The `HttpTrigger(WebHookType = "genericJson")` attribute indicates the message that triggers the function.

```
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Host;
using Newtonsoft.Json;
using System.Net;
using System.Net.Http;
using System.Threading.Tasks;
namespace SolVsFunctionapp
{
    public static class GenericWebhookFunction
    {
        [FunctionName("GenericWebhookFunction")]
        public static async Task<object> Run([HttpTrigger(WebHookType =
"genericJson")]HttpRequestMessage req, TraceWriter log)
        {
            log.Info($"Webhook was triggered!");

            string jsonContent = await req.Content.ReadAsStringAsync();
            dynamic data = JsonConvert.DeserializeObject(jsonContent);

            if (data.first == null || data.last == null)
            {
```

```

        return req.CreateResponse(HttpStatusCode.BadRequest, new
        {
            error = "Please pass first/last properties in the input
object"
        });
    }

    return req.CreateResponse(HttpStatusCode.OK, new
    {
        greeting = $"Hello {data.first} {data.last}!"
    });
}
}
}
}

```

7. You can run the function from Visual Studio directly using Azure Functions Tools. Press F5 to run. If prompted, accept the download and install Azure Functions Core tools.
8. You can copy the URL of your function from the Azure Function runtime output (Figure 4-91).



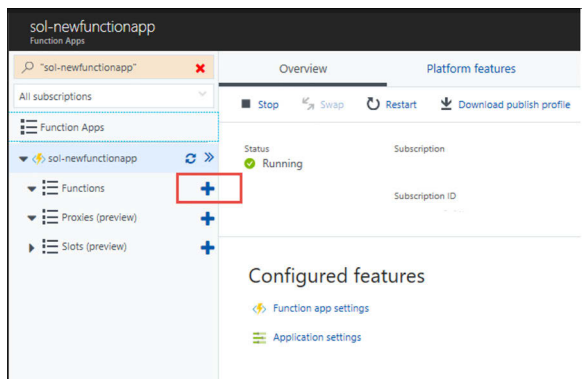
**FIGURE 4-91** The console output after running a Webhook function from Visual Studio

9. You can now post a JSON payload to the function using any tool that can issue HTTP requests to test the function.

## Create an event processing function

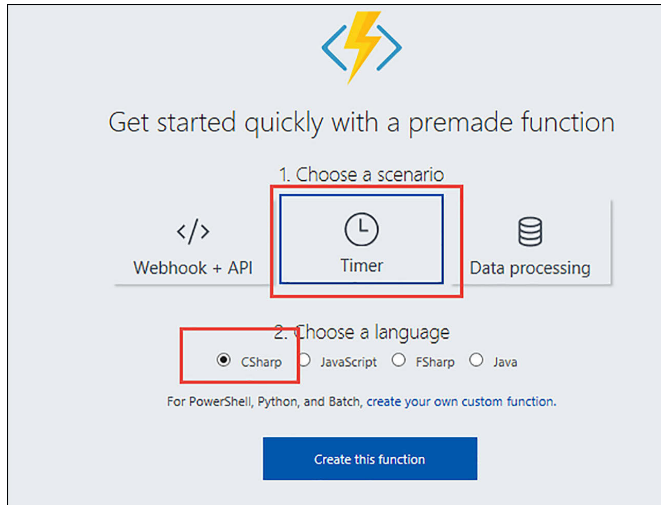
To create an event processing function, please complete these steps:

1. Navigate to the portal accessed via <https://portal.azure.com>.
2. Go to your Function App, such as the one created in the previous section, and click the + sign to create a new function (Figure 4-92).



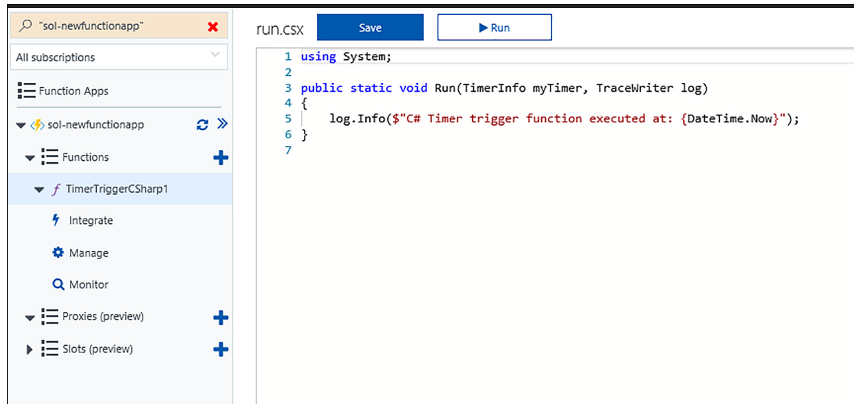
**FIGURE 4-92** The Function Apps blade where you can create a new function

3. Select Timer and CSharp, and select Create This Function (Figure 4-93).



**FIGURE 4-93** The Function Apps blade where you can choose the type of function

4. This creates a skeleton function that runs based on a timer. You can edit the function. json file to adjust settings for the function (Figure 4-94).



**FIGURE 4-94** A new timer-based function

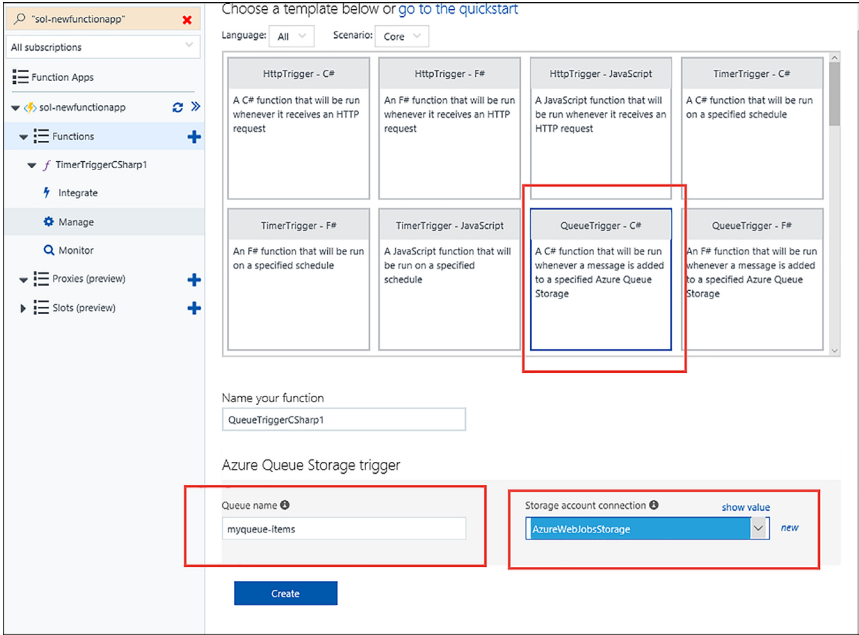
5. You can view the output of the function and any logs emitted as it executes.

## Implement an Azure-connected function

To create an Azure-connected function using Azure Queues, follow these steps:

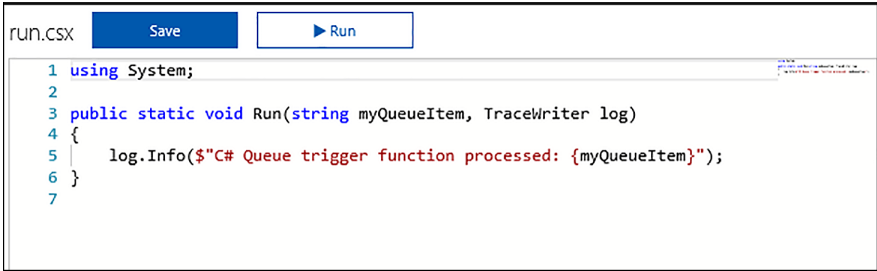
1. Navigate to the portal accessed via <https://portal.azure.com>.
2. Go to your Function App, such as the one used in the previous section, and click the + sign to create a new function.

3. Select QueueTrigger - C#, provide a name for the function, provide the name of the queue and the storage account that it belongs to. Click Create to create the function (Figure 4-95).



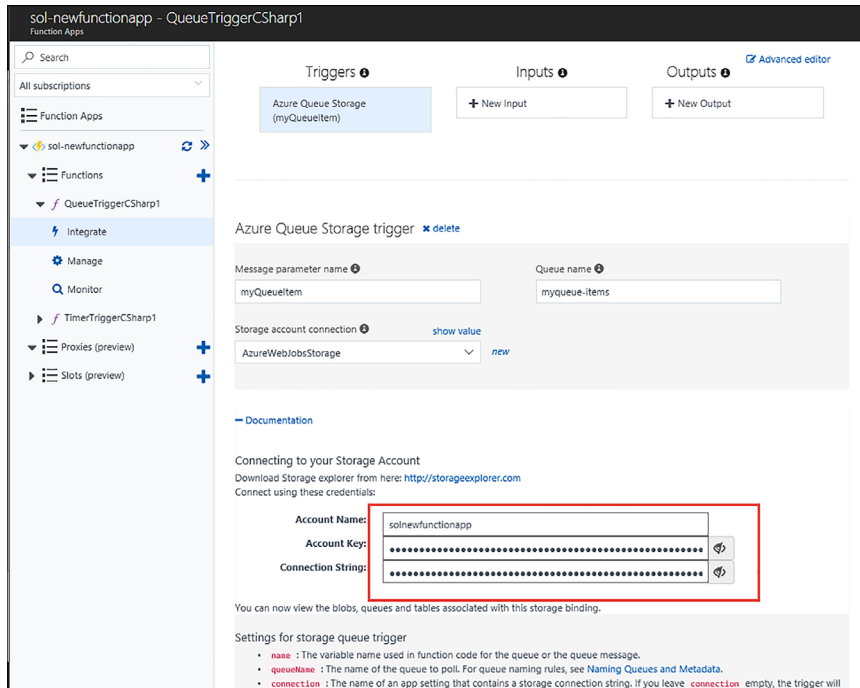
**FIGURE 4-95** The setup for a QueueTrigger

4. A skeleton implementation for the function is created. This is triggered for each message written to the specified queue (Figure 4-96).



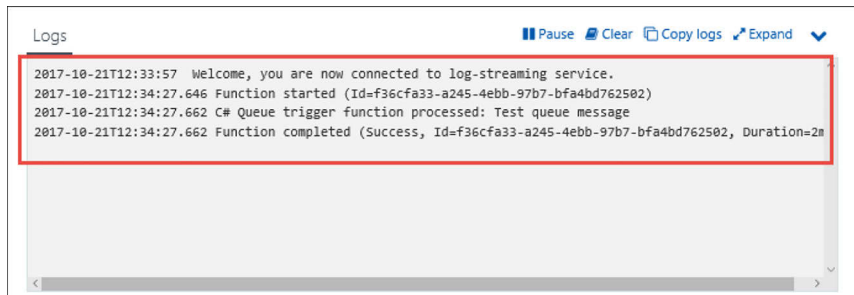
**FIGURE 4-96** The code behind the QueueTrigger function

5. To complete the integration, create the storage account and queue that you specified when creating the function. From the function app definition, select the Integrate tab, and select the storage queue under Triggers. Expand the Documentation link and enter the storage account name and key. The function will use these credentials to connect to the storage account (Figure 4-97).



**FIGURE 4-97** The integration blade for setting up the storage queue trigger credentials

To test the function, add a message to the queue. After a few seconds the function log in the portal shows output from processing the message (Figure 4-98).



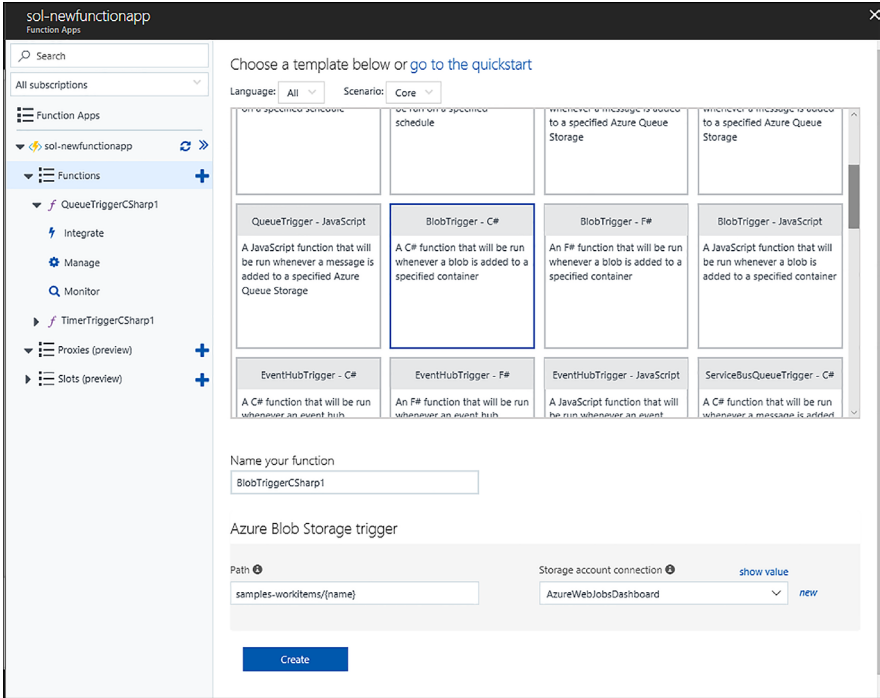
**FIGURE 4-98** The log output for the function after processing a single message

## Integrate a function with storage

To create a function integrated with Azure Storage Blobs, follow these steps:

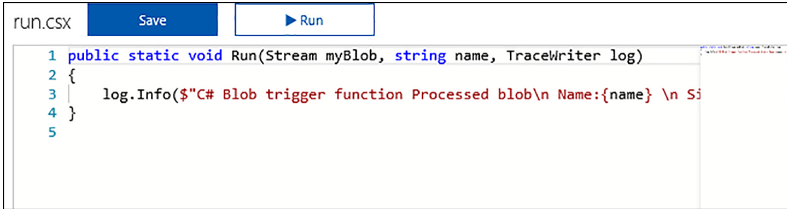
1. Navigate to the portal accessed via <https://portal.azure.com>.
2. Go to your Function App, such as the one used in the previous section, and click the + sign to create a new function.

3. Select BlobTrigger - C#, provide a name for the function, provide the path to the blob container item and the storage account that it belongs to. Click Create to create the function (Figure 4-99).



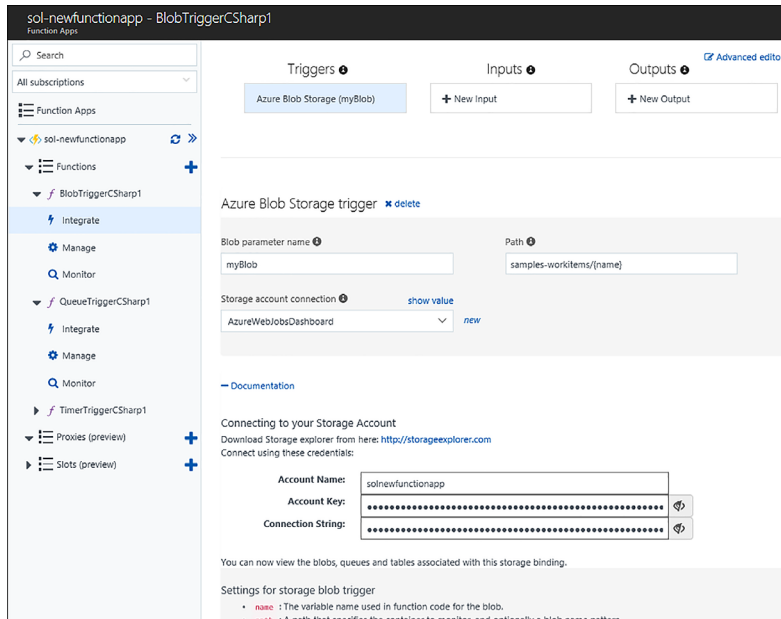
**FIGURE 4-99** The setup for a BlobTrigger

4. A skeleton implementation for the function is created. This is triggered for each blob written to the specified storage container (Figure 4-100).



**FIGURE 4-100** The code behind the BlobTrigger function

5. To complete the integration, create the storage account and blob container that you specified when creating the function. From the function app definition, select the Integrate tab, and select Azure Blob Storage under Triggers. Expand the Documentation link, and enter the storage account name and key. The function uses these credentials to connect to the storage account (Figure 4-101).



**FIGURE 4-101** The integration blade for setting up the blob trigger credentials

6. To test the function, add a file to the blob container. After a few seconds the function log in the portal shows output from processing the message, as illustrated in the previous section for Azure storage queues.

## Design and implement a custom binding

Function triggers indicate how a function is invoked. There are a number of predefined triggers, some already discussed in previous sections, including:

- HTTP triggers
- Event triggers
- Queues and topic triggers
- Storage triggers

Every function must have one trigger. The trigger is usually associated with a data payload that is supplied to the function. Bindings are a declarative way to map data to and from function code. Using the Integrate tab (as shown in previous sections to connect a Queue to a function, for example) you can provide connection settings for such a data binding activity.

### **MORE INFO** TRIGGERS AND BINDINGS

For additional details on triggers and bindings available to Azure Functions, and how they work, see <https://docs.microsoft.com/en-us/azure/azure-functions/functions-triggers-bindings>.



#### EXAM TIP

You can also create custom input and output bindings to assist with reducing code bloat in your functions by encapsulating reusable, declarative work into the binding. For details on how to implement custom bindings see <https://github.com/Azure/azure-webjobs-sdk/wiki/Creating-custom-input-and-output-bindings>.

## Debug a Function

You can use VS Code or Visual Studio 2017 to debug an Azure Function. For more information on working with local Functions projects and local debugging, see: <https://docs.microsoft.com/en-us/azure/azure-functions/functions-run-local>.

## Implement and configure proxies

If you have a solution with many functions you'll find it can become work to manage given the different URLs, naming, and versioning potentially related to each function. An API Proxy acts as a single point of entry to functions from the outside world. Instead of calling the individual function URLs, you provide a proxy as a facade to your different function URLs.

#### NOTE API PROXIES

API Proxies make sense in HTTP-bound Azure Functions. They may work for other event-driven functions, however, HTTP triggers are best suited for their functionality. In addition, API Proxies are in preview at the time of this writing and do not include any security features. As an alternative, you can use API Management for a fully featured solution.

To create a simple API Proxy, follow these steps (Figure 4-102):

1. Consider an existing function that includes the function code (API key) and any query string parameters in the URL such as the following example:

```
https://so1-newfunctionapp.azurewebsites.net/api/AirplanesApi?code=N8eJPFEkD1Mk0eQngOqRsaLVxeHRQ4QcxacFRdLtMDBdak3eeN/kNQ==&id=0099991
```

2. API proxies require two important pieces of information:
  - A. **The Route Template** Provides a template of how the proxies are triggered, for example a REST-compliant API path that removes the need for the function code and query string parameters:

```
/api/airplanes/86327
```

- B. **The Backend URL** The function URL to match to.



New proxy

Name  
ApiProxy

Route template  
api/{rest}/{id}

Allowed HTTP methods  
All methods

Backend URL  
pp.azurewebsites.net/api/AirplanesApi?code=N8eJPFEkD1MkOeQngOqRsaLVxeHRQ4QcxacFRdLtMD8dak3eeN/kNQ==&id=0099991

+ Request override

+ Response override

Create

**FIGURE 4-102** The settings while creating a new API proxy

3. Update the Backend URL too so that it uses the variables provided in the route template.

`https://so1-newfunctionapp.azurewebsites.net/api/{rest}Api?code=q/vTyTaw4wTzyFuY16wuMOnUPEhJLzRFqKRDxaChGz3/HzS0myMaNw==&id={id}`.

4. When you request the URL, the variables in the route template (i.e., {rest} and {id}) are replaced with whatever is passed in the request. For example, this URL:

`https://so1-newfunctionapp.azurewebsites.net/api/airplanes/3434`

Routes to this URL:

`https://so1-newfunctionapp.azurewebsites.net/api/airplanesApi?code=q/vTyTaw4wTzyFuY16wuMOnUPEhJLzRFqKRDxaChGz3/HzS0myMaNw==&id=3434`



#### **EXAM TIP**

API proxies have the ability to modify the requests and responses on the fly.

#### **MORE INFO API PROXIES**

For more details about API Proxies see <https://docs.microsoft.com/en-us/azure/azure-functions/functions-proxies>.

## Integrate with App Service Plan

Functions can operate in two different modes:

- **Consumption Plan** Where your function is allocated dynamically to the amount of compute power required to execute under the current load.
- **App Service Plan** Where your function is assigned a specific app service hosting plan and is limited to the resources available to that hosting plan.

For more information about the difference between Consumption and App Service Plans see: <https://docs.microsoft.com/en-us/azure/azure-functions/functions-scale>. For more information about setting up an App Service Plan see: <https://docs.microsoft.com/en-us/azure/app-service/azure-web-sites-web-hosting-plans-in-depth-overview>.

## Skill 4.7: Design and Implement Azure Service Fabric apps

---

Azure Service Fabric is a platform that makes it easy to package, deploy, and manage distributed solutions at scale. It provides an easy programming model for building microservices solutions with a simple, familiar, and easy to understand development experience that supports stateless and stateful services, and actor patterns. In addition, to providing a packaging and deployment solution for these native components, Service Fabric also supports the deployment of guest executables and containers as part of the same managed and distributed system.

The following list summarizes these native and executable components:

- **Stateless Services** Stateless Fabric-aware services that run without managed state.
- **Stateful Services** Stateful Fabric-aware services that run with managed state where the state is close to the compute.
- **Actors** A higher level programming model built on top of stateful services.
- **Guest Executable** Can be any application or service that may be cognizant or not cognizant of Service Fabric.
- **Containers** Both Linux and Windows containers are supported by Service Fabric and may be cognizant or not cognizant of Service Fabric.

This skill provides an overview of the Service Fabric programming experience.

### **MORE INFO** SERVICE FABRIC OVERVIEW

For an overview of Service Fabric see <https://docs.microsoft.com/en-us/azure/service-fabric>.

**This skill covers how to:**

- Create a Service Fabric application
- Add a web front end to a Service Fabric application
- Build an Actors-based service
- Monitor and diagnose services
- Deploy an application to a container
- Migrate apps from cloud services
- Scale a Service Fabric app
- Create, secure, upgrade, and scale Service Fabric Cluster in Azure

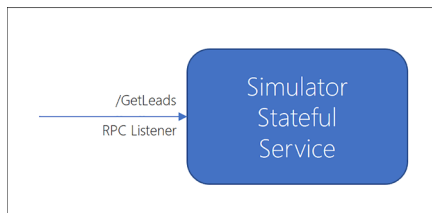
## Create a Service Fabric application

A Service Fabric application can consist of one or more services. The application defines the deployment package for the services, and each service can have its own configuration, code, and data. A Service Fabric cluster can host multiple applications, and each has its own independent deployment and upgrade lifecycle.

### **MORE INFO** SERVICE FABRIC APPLICATIONS

The following reference has additional information about the Service Fabric application and related concepts at <https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-application-model>.

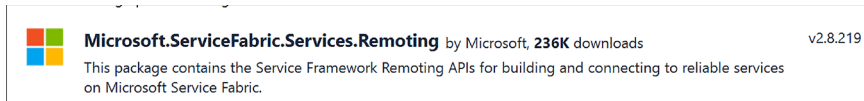
In this skill you create a new Service Fabric application that has a stateful service. This service is reachable via RPC and is called by a web front end created in the next section. The service is called Lead Generator and returns the current count for the number of leads that have been generated and persisted with the service. Figure 4-103 illustrates the service endpoint.



**FIGURE 4-103** A simple stateful service endpoint supporting RPC communication

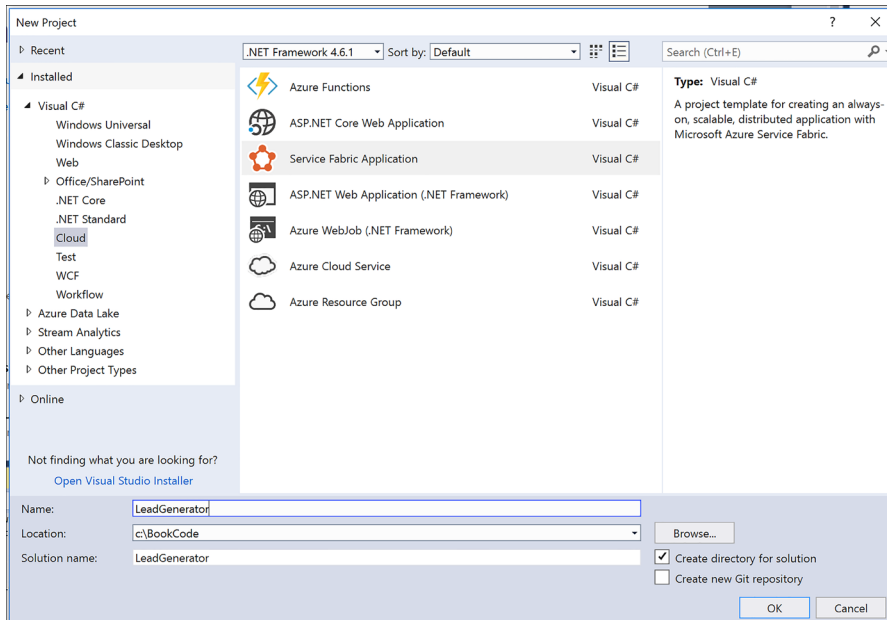
To create a new Service Fabric application, follow these steps:

1. Launch Visual Studio, and then select File > New > Project.
2. In the New Project dialog, select Service Fabric Application within the Cloud category. Provide a name and location for your new project, and then click OK. In this example the name is LeadGenerator (Figure 4-104).



**FIGURE 4-104** The New Project dialog where you can select Service Fabric Application as the project type

3. Select Stateful Service from the list of service templates and provide a name, `LeadGenerator.Simulator` as shown here.



**FIGURE 4-105** The New Service Fabric Service dialog where you can select Stateful Service as the service template

4. From Solution Explorer, expand the new `LeadGenerator.Simulator` node and expand the `PackageRoot` folder where you'll find `ServiceManifest.xml`. This file describes the service deployment package and related information. It includes a section that describes the service type that is initialized when the Service Fabric runtime starts the service:

```
<ServiceTypes>
  <StatefulServiceType ServiceTypeName="SimulatorType" HasPersistedState="true" />
</ServiceTypes>
```

5. A service type is created for the project; in this case the type is defined in the `Simulator.cs` file. This service type is registered when the program starts, in `Program.cs`, so that the Service Fabric runtime knows which type to initialize when it creates an instance of the service.

```

private static void Main()
{
    try
    {
        ServiceRuntime.RegisterServiceAsync("SimulatorType",
            context => new Simulator(context)).GetAwaiter().GetResult();
        ServiceEventSource.Current.ServiceTypeRegistered(Process.
GetCurrentProcess().Id,
        typeof(Simulator).Name);
        Thread.Sleep(Timeout.Infinite);
    }
    catch (Exception e)
    {
        ServiceEventSource.Current.ServiceHostInitializationFailed(e.ToString());
        throw;
    }
}

```

6. The template produces a default implementation for the service type, with a `RunAsync` method that increments a counter every second. This counter value is persisted with the service in a dictionary using the `StateManager`, available through the service base type `StatefulService`. This counter is used to represent the number of leads generated for the purpose of this example.

```

protected override async Task RunAsync(CancellationToken cancellationToken)
{
    var myDictionary = await this.StateManager.GetOrAddAsync<IReliableDictionary<string, long>>("myDictionary");
    while (true)
    {
        cancellationToken.ThrowIfCancellationRequested();
        using (var tx = this.StateManager.CreateTransaction())
        {
            var result = await myDictionary.TryGetValueAsync(tx, "Counter");
            ServiceEventSource.Current.ServiceMessage(this.Context, "Current Counter Value: {0}",
                result.HasValue ? result.Value.ToString() : "Value does not exist.");
            await myDictionary.AddOrUpdateAsync(tx, "Counter", 0, (key, value) => ++value);
            await tx.CommitAsync();
        }
        await Task.Delay(TimeSpan.FromSeconds(1), cancellationToken);
    }
}

```

7. This service will run, and increment the counter as it runs persisting the value, but by default this service does not expose any methods for a client to call it. Before you can create an RPC listener you add the required nuget package, `Microsoft.ServiceFabric.Services.Remoting`.

8. Create a new service interface using the `IService` marker interface from the `Microsoft.ServiceFabric.Services.Remoting` namespace, that indicates this service can be called remotely:

```
using Microsoft.ServiceFabric.Services.Remoting;
using System.Threading.Tasks;
public interface ISimulatorService : IService
{
    Task<long> GetLeads();
}
```

9. Implement this interface on the `Simulator` service type, and include an implementation of the `GetLeads` method to return the value of the counter:

```
public async Task<long> GetLeads()
{
    var myDictionary = await StateManager.GetOrAddAsync<IReliableDictionary<string, long>>("myDictionary");
    using (var tx = StateManager.CreateTransaction())
    {
        var result = await myDictionary.TryGetValueAsync(tx, "Counter");
        await tx.CommitAsync();
        return result.HasValue ? result.Value : 0;
    }
}
```

10. To expose this method to clients, add an RPC listener to the service. Modify the `CreateServiceReplicaListeners()` method in the `Simulator` service type implementation, to add a call to `CreateServiceReplicaListeners()` as shown here:

```
protected override IEnumerable<ServiceReplicaListener>
CreateServiceReplicaListeners()
{
    yield return new ServiceReplicaListener(this,
CreateServiceRemotingListener);
}
```

#### **MORE INFO** SERVICE FABRIC COMMUNICATION

For more information related to setting up listeners for Service Fabric stateful services see <https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-reliable-services-communication>.

## Add a web front end to a Service Fabric application

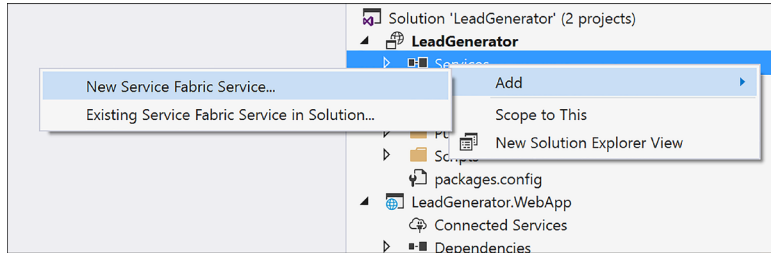
The previous section reviewed creating a simple stateful service that returns the value of a counter over RPC. To illustrate calling this service from a client application, this section reviews how to create a web front end and call a stateful service endpoint, as illustrated in Figure 4-106.



**FIGURE 4-106** An HTTP listener-based web app calling a stateful service over RPC

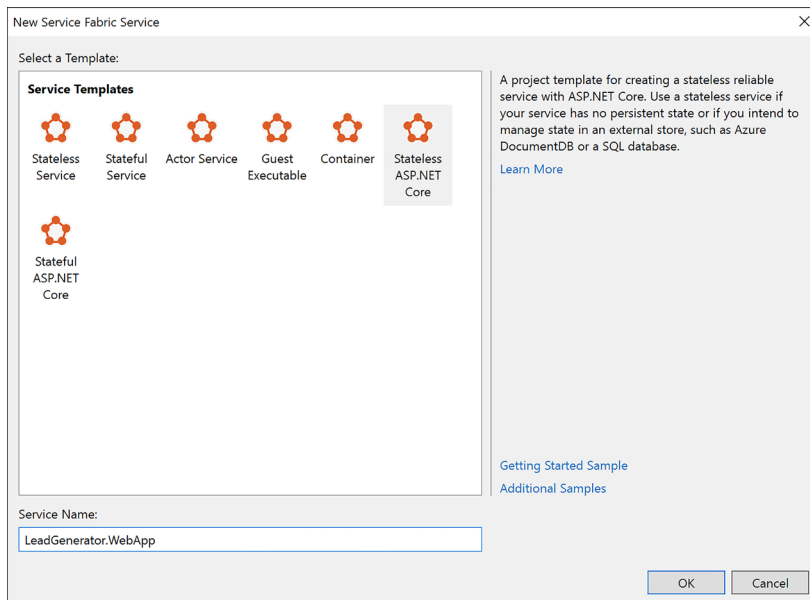
Follow these steps to add a web app to an existing Service Fabric application:

1. From the Solution Explorer in Visual Studio, expand the Service Fabric application node. Right-click the Services node, and select New Service Fabric Service (Figure 4-107).



**FIGURE 4-107** The context menu for adding a new Service Fabric service to the existing application services

2. From the New Service Fabric Service dialog, select Stateless ASP.NET Core for the service template. Supply the service name LeadGenerator.WebApp, and click OK (Figure 4-108).



**FIGURE 4-108** The New Service Fabric Service dialog where you can choose the Stateless ASP.NET Core template

3. From the New ASP.NET Core Web Application dialog select Web Application (Model-View-Controller) template. Click OK.
4. From Solution Explorer, expand the new LeadGenerator.WebApp node, and expand the PackageRoot folder where you'll find ServiceManifest.xml. Alongside the service type definition there is a section that describes the HTTP endpoint where the web app will listen for requests:

```
<Endpoints>"
  <Endpoint Protocol="http" Name="ServiceEndpoint" Type="Input" Port="8168" />
</Endpoints>
```

5. The new WebApp type is defined in WebApp.cs, which inherits StatelessService. For the service to listen for HTTP requests, the CreateServiceInstanceListeners() method sets up the WebListener as shown in this listing for the type:

```
internal sealed class WebApp : StatelessService
{
    public WebApp(StatelessServiceContext context) : base(context)
    { }
    protected override IEnumerable<ServiceInstanceListener>
    CreateServiceInstanceListeners()
    {
        return new ServiceInstanceListener[]
        {
            new ServiceInstanceListener(serviceContext =>
                new WebListenerCommunicationListener(serviceContext,
                    "ServiceEndpoint", (url, listener) =>
                    {
                        ServiceEventSource.Current.ServiceMessage(serviceContext,
                            $"Starting WebListener on {url}");
                        return new WebHostBuilder().UseWebListener()
                            .ConfigureServices(services =>
                                services
                                    .AddSingleton<StatelessServiceContext>(serviceCon
text))
                                    .UseContentRoot(Directory.GetCurrentDirectory())
                                    .UseStartup<Startup>()
                                    .UseApplicationInsights()
                                    .UseServiceFabricIntegration(listener,
ServiceFabricIntegrationOptions.None)
                                    .UseUrls(url)
                                    .Build();
                    })
                )))
        };
    }
}
```

Next you call the stateful service that returns the leads counter value, from the stateless web application just created.

1. Make a copy of the service interface defined for the service type, in this case ISimulatorService:



```
public interface ISimulatorService : IService
{
    Task<long> GetLeads();
}
```

2. Modify the `ConfigureServices` instruction in `WebApp.cs` to inject an instance of the `FabricClient` type (change shown in bold):

```
return new WebHostBuilder().UseWebListener()
    .ConfigureServices(services => {
        services
            .AddSingleton<StatelessServiceContext>(serviceContext)
            .AddSingleton(new FabricClient());
    })
```

3. Now that `FabricClient` is available for dependency injection, modify the `HomeController` to use it:

```
private FabricClient _fabricClient;
public HomeController(FabricClient client) { _fabricClient = client; }
```

4. Modify the `Index` method in the `HomeController` to use the `FabricClient` instance to call the `Simulator` service:

```
public async Task<IActionResult> Index()
{
    ViewData["Message"] = "Your home page.";
    var model = new Dictionary<Guid, long>();
    var serviceUrl = new Uri("fabric:/LeadGenerator/Simulator");
    foreach (var partition in await
        _fabricClient.QueryManager.GetPartitionListAsync(serviceUrl))
    {
        var partitionKey = new ServicePartitionKey
            (((Int64RangePartitionInformation)partition.PartitionInformation).LowKey);
        var proxy = ServiceProxy.Create<ISimulatorService>(serviceUrl,
            partitionKey);
        var leads = await proxy.GetLeads();
        model.Add(partition.PartitionInformation.Id, leads);
    }
    return View(model);
}
```

5. Update `Index.cshtml` to display the counter for each partition:

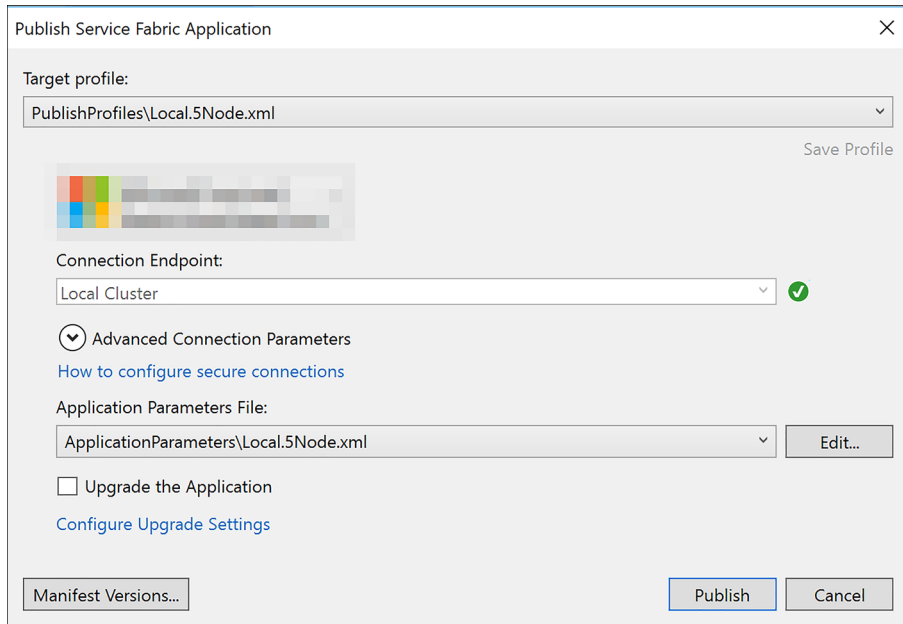
```
@model IDictionary<Guid, long>
<h2>@ViewData["Title"].</h2>
<h3>@ViewData["Message"]</h3>
<table class="table-bordered">
    <tr>
        <td><strong>PARTITION ID</strong></td>
        <td><strong># LEADS</strong></td>
    </tr>
    @foreach (var partition in Model)
    {
        <tr>
```

```

        <td>@partition.Key.ToString()</td>
        <td>@partition.Value</td>
    </tr>
}
</table>

```

6. To run the web app and stateful service, you can publish it to the local Service Fabric cluster. Right-click the Service Fabric application node from the Solution Explorer and select Publish. From the Publish Service Fabric Application dialog, select a target profile matching one of the local cluster options, and click Publish (Figure 4-109).



**FIGURE 4-109** The Publish Service Fabric Application dialog

7. Once the application is deployed, you can access the web app at <http://localhost:8162> (or, whatever the indicated port is in the service manifest for the web app. The home page triggers a call to the stateful service, which will increment as the counter is updated while it runs.

## Build an Actors-based service

The actor model is a superset of the Service Fabric stateful model. Actors are simple POJO objects that have many features that make them isolated, independent unit of compute and state with single-thread execution.

To create a new Service Fabric application based on the Actor service template, follow these steps:

1. Launch Visual Studio, then select File > New > Project.
2. In the New Project dialog, select Service Fabric Application within the Cloud category. Provide a name and location for your new project, and then click OK.
3. Select Actor Service from the list of service templates and provide a name, such as SimpleActor.
4. This generates a default implementation of the Actor Service.

#### **MORE INFO SERVICE FABRIC RELIABLE ACTORS**

For more information on the implementation of the actor pattern in Service Fabric see <https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-reliable-actors-introduction>.

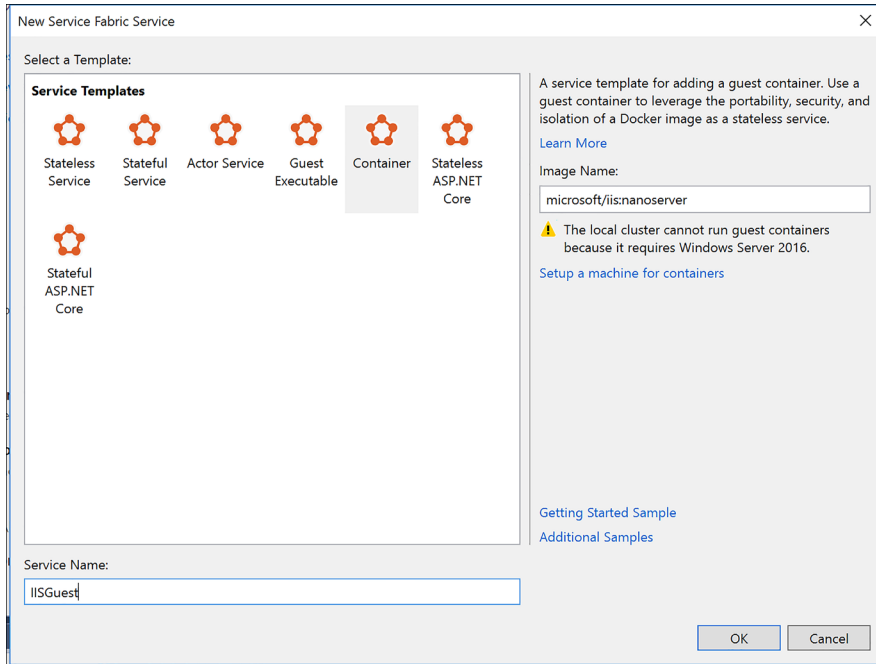
## Monitor and diagnose services

All applications benefit from monitoring and diagnostics to assist with troubleshooting issues, evaluating performance or resource consumption, and gathering useful information about the application at runtime. For more information about Service Fabric specific approaches to this, see <https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-diagnostics-overview>.

## Deploy an application to a container

Service Fabric can run processes and containers side by side, and containers can be Linux or Windows based containers. If you have an existing container image and wish to deploy this to an existing Service Fabric cluster, you can follow these steps to create a new Service Fabric application and set it up to deploy and run the container in your cluster:

1. Launch Visual Studio, nd then select File > New > Project.
2. In the New Project dialog, select Service Fabric Application within the Cloud category. Provide a name and location for your new project, and then click OK.
3. From the New Service Fabric Service dialog, choose Container for the list of templates and supply a container image and name for the guest executable to be created (Figure 4-110).



**FIGURE 4-110** The New Service Fabric Service dialog with Container selected, and an image name specified

4. From Solution Explorer, open the ServiceManifest.xml file and modify the <Resources> section to provide a UriScheme, Port and Protocol setting for the service endpoint.

```
<Resources>
  <Endpoints>
    <Endpoint Name="IISGuestTypeEndpoint" UriScheme="http" Port="80"
Protocol="http"/>
  </Endpoints>
</Resources>
```

5. From Solution Explorer, open the ApplicationManifest.xml file. Create a policy for container to host <PortBinding> policy by adding this <Policies> section to the <ServiceManifestImports> section. Indicate the container port for your container. In this example the container port is 80.

```
<ServiceManifestImport>
  <ServiceManifestRef ServiceManifestName="IISGuestPkg"
ServiceManifestVersion="1.0.0" />
  <ConfigOverrides />
  <Policies>
    <ContainerHostPolicies CodePackageRef="Code">
      <PortBinding ContainerPort="80" EndpointRef="IISGuestTypeEndpoint"/>
    </ContainerHostPolicies>
  </Policies>
</ServiceManifestImport>
```

6. Now that you have the application configured, you can publish and run the service.



#### EXAM TIP

Currently, you cannot run containers in the local Service Fabric cluster because it requires Windows Server 2016 with container support.

#### MORE INFO WINDOWS CONTAINERS

For more information regarding working with Windows containers both locally and in Windows Server environments see <https://docs.microsoft.com/en-us/virtualization/windowscontainers/index>.

## Migrate apps from cloud services

You can migrate your existing cloud services, both web and worker roles, to Service Fabric applications following instructions in the following reference at <https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-cloud-services-migration-worker-role-stateless-service>.

## Scale a Service Fabric app

In order to scale a Service Fabric app, the following terms are important to understand: Instances, Partitions, and Replicas.

By default, the Service Fabric tooling produces three publish profiles that you can use to deploy your application:

- **Local.1Node.xml** To deploy against the local 1-node cluster.
- **Local.5Node.xml** To deploy against the local 5-node cluster.
- **Cloud.xml** To deploy against a Cloud cluster.

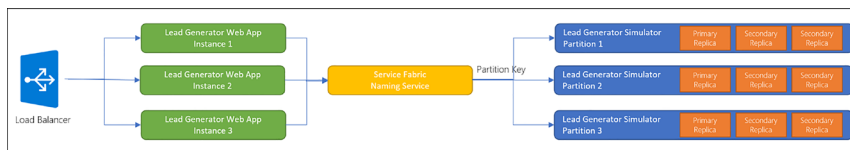
These publish profiles indicate the settings for the number of instances and partitions for each service. Consider this example of the parameters to a Local.5Node.xml:

```
<Parameters>
  <Parameter Name="WebApp_InstanceCount" Value="3" />
  <Parameter Name="Simulator_PartitionCount" Value="3" />
  <Parameter Name="Simulator_MinReplicaSetSize" Value="3" />
  <Parameter Name="Simulator_TargetReplicaSetSize" Value="3" />
</Parameters>
```

- **WebApp\_InstanceCount** Specifies the number of instances the WebApp service must have within the cluster.
- **Simulator\_PartitionCount** Specifies the number of partitions (for the stateful service) the Simulator service must have within the cluster.
- **Simulator\_MinReplicaSetSize** Specifies the minimum number of replicas required for each partition that the WebApp service should have within the cluster.

- **Simulator\_TargetReplicaSetSize** Specifies the number of target replicas required for each partition that the WebApp service should have within the cluster.

Consider the following diagram illustrating the instances and partitions associated with the stateless Web App and stateful simulator service, as shown in the Local.5Node.xml configuration (Figure 4-111).



**FIGURE 4-111** The instances for a stateless service, and partitions for a stateful service

- The Web App instance count is set to 3. As the diagram illustrates, when published to a Service Fabric cluster in Azure requests would be load balanced across those three instances.
- The Simulator service is assigned three partitions, each of which have replicas to ensure durability of each instance's state.



#### EXAM TIP

Sometimes the terms instances and replicas are used interchangeably, however, instances are for stateless services whereas replicas are for stateful services.

## Create, secure, upgrade, and scale Service Fabric Cluster in Azure

To publish your Service Fabric application to the Azure in production, you'll create a cluster, learn how to secure it, learn how to upgrade applications with zero downtime, and configure the application to scale following some of the practices already discussed. The following references will start you off with these topics:

- For an introduction to creating a Service Fabric Cluster see:
  - <https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-get-started-azure-cluster>
  - <https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-deploy-anywhere>
- For details on securing Azure Service Fabric Clusters in production, see this reference:
  - <https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-cluster-security>
- For details on upgrading clusters, see this reference:
  - <https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-cluster-upgrade>
- You can scale clusters manually or programmatically as described in these references:

- <https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-cluster-scale-up-down>
- <https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-cluster-programmatic-scaling>

## Skill 4.8: Design and implement third-party Platform as a Service (PaaS)

---

Azure supports many third-party PaaS offerings and services through the Azure Marketplace. These can be deployed through the Azure portal, using ARM, or using other CLI tools. This skill helps you navigate those offerings.

### This skill covers how to:

- Implement Cloud Foundry
- Implement OpenShift
- Provision applications by using Azure Quickstart Templates
- Build applications that leverage Azure Marketplace solutions and services

## Implement Cloud Foundry

Cloud Foundry is an open-source PaaS for building, deploying, and operating 12-factor applications developed in various languages and frameworks. It is a mature container-based application platform allowing you to easily deploy and manage production-grade applications on a platform that supports continuous delivery and horizontal scale, and supports hybrid and multi-cloud scenarios.

There are two forms of Cloud Foundry available to run on Azure:

- **Open-source Cloud Foundry (OSS CF)** An entirely open-source version of Cloud Foundry managed by the Cloud Foundry Foundation.
- **Pivotal Cloud Foundry (PCF)** An enterprise distribution of Cloud Foundry from Pivotal Software Inc., which adds on a set of proprietary management tools and enterprise support.

### **MORE INFO** AZURE SERVICE PRINCIPALS

Before you can create a Cloud Foundry cluster in Azure you must first create an Azure Service Principal, following the instructions found at: <https://github.com/cloudfoundry-incubator/bosh-azure-cpi-release/blob/master/docs/get-started/create-service-principal.md>.

To deploy a basic Pivotal Cloud Foundry on Azure from the Azure Marketplace, follow these steps:

1. Navigate to the portal accessed via <https://portal.azure.com>.
2. Select Marketplace from the Azure Dashboard.
3. Search for "Pivotal Cloud Foundry," and select Pivotal Cloud Foundry On Azure.
4. From within the Pivotal Cloud Foundry On Azure blade, click Create (Figure 4-112).
5. On the Basics blade, provide a storage account name prefix, paste your SSH public key, upload the azure-credentials.json Service Principal file, enter the Pivotal Network API token, choose a resource group, and location for the cluster. Click OK.

The screenshot shows the 'Create Pivotal Cloud Foundry on...' portal. The 'Basics' blade is active, showing the following configuration details:

- Storage Account Name Prefix:** mypivotal
- SSH public key:** A long alphanumeric string starting with 'ssh-rsa'.
- Service Principal:** azure-credentials.json
- Pivotal Network Token:** A long alphanumeric string.
- Subscription:** A dropdown menu.
- Resource group:** Create new (selected), CloudFoundry
- Location:** West US

**FIGURE 4-112** The selections for a new Pivotal Cloud Foundry cluster in the portal

6. On the Summary blade, wait for the validation to pass and click OK.
7. On the Buy blade, click Purchase.

To deploy the open-sourced version of Cloud Foundry on Azure, you deploy BOSH and then Cloud Foundry. The steps can be performed manually, or via Azure Resource Manager (ARM) templates. Detailed instructions can be found at <https://github.com/cloudfoundry-incubator/bosh-azure-cpi-release/tree/master/docs>.

#### **MORE INFO SSH KEYS**

For more information about creating SSH keys for creating clusters see: <https://docs.microsoft.com/en-us/azure/virtual-machines/linux/ssh-from-windows>.

#### **MORE INFO DEPLOYING AN APP TO CLOUD FOUNDRY**

For more information about deploying apps to your Cloud Foundry cluster see: <https://docs.microsoft.com/azure/virtual-machines/linux/cloudfoundry-deploy-your-first-app>.



# Implement OpenShift

The OpenShift Container Platform is a PaaS offering from Red Hat built on Kubernetes. It brings together Docker and Kubernetes, and provides an API to manage these services. OpenShift simplifies the process of deploying, scaling, and operating multi-tenant applications onto containers.

There are two forms of OpenShift that you can deploy to Azure:

- The open-source OpenShift Origin
- The enterprise-grade Red Hat OpenShift Container Platform

Both are built on the same open source technologies, with the Red Hat OpenShift Container Platform offering enterprise-grade security, compliance, and container management.

Prerequisites for installing both forms of OpenShift include:

1. Generate an SSH key pair (Public / Private), ensuring that you do not include a passphrase with the private key.
2. Create a Key Vault to store the SSH Private Key.
3. Create an Azure Active Directory Service Principal.
4. Install and configure the OpenShift CLI to manage the cluster.

Some specific prerequisites for deploying Red Hat OpenShift Container Platform include:

5. OpenShift Container Platform subscription eligible for use in Azure. You need to specify the Pool ID that contains your entitlements for OpenShift.
6. Red Hat Customer Portal login credentials. You may use either an Organization ID and Activation Key, or a Username and Password. It is more secure to use the Organization ID and Activation Key.

You can deploy both from the Azure Marketplace templates, or using ARM templates.

To deploy Red Hat OpenShift Container Platform on Azure from the Azure Marketplace, perform the following steps (Figure 4-113):

1. Navigate to the portal accessed via <https://portal.azure.com>.
2. Select Marketplace from the Azure Dashboard.
3. Search for “OpenShift,” and select Red Hat OpenShift Container Platform (BYOL).
4. From within the Red Hat OpenShift Container Platform (BYOL) blade, click Create.
5. On the Basics blade, provide the VM Admin user name, paste the SSH public key, choose a resource group and location for the platform. Click OK.

**Create Red Hat OpenShift Container Platform** Basics

- Basics  
Configure basic settings
- Infrastructure Settings  
Configure Infrastructure Settings
- OpenShift Container Platform Settings  
Configure OpenShift Container Platform
- Summary  
Red Hat OpenShift Container Platform
- Buy

\* VM Admin User Name  
clusteradmin

\* SSH Public Key for VM Admin User  
qzG0GbYx6LeLqZnc1IScVLfb3FUV6GvM  
tJ4TNouYIV2IhOMK8uTYwIUd2JM+ngzK  
8Fq+Dd8nOBzbqQ== rsa-key-

Subscription  
[Dropdown]

\* Resource group  
☒ Create new ☐ Use existing  
OpenShift

\* Location  
West US 2

**FIGURE 4-113** The selections in the Basics blade for a new Red Hat OpenShift Container Platform

- On the Infrastructure Settings blade, provide an OCP cluster name prefix, select a cluster size, provide the resource group name for your Key Vault, as well as the Key Vault name and its secret name you specified in the prerequisites. Click OK (Figure 4-114).

**Create Red Hat OpenShift Container Platform** Infrastructure Settings

- Basics  
Done
- Infrastructure Settings  
Configure Infrastructure Settings
- OpenShift Container Platform Settings  
Configure OpenShift Container Platform
- Summary  
Red Hat OpenShift Container Platform
- Buy

\* OCP Cluster Name Prefix  
ocpcluster

OpenShift Cluster Size  
Small Medium Large

Medium Cluster Configuration  
1 Bastion Node of size Standard DS2v2  
3 Master Nodes of size Standard DS3v2  
2 Infra Nodes of size Standard DS3v2  
4 App Nodes of size Standard DS3v2  
256 GB Data Disk for Docker Volume per VM  
Total Cores Required: 38

\* Key Vault Resource Group Name  
OpenShift

\* Key Vault Name  
OSKV

\* Secret Name  
OpenShiftKey

**FIGURE 4-114** The selections in the Infrastructure Settings blade for a new Red Hat OpenShift Container Platform in the portal

- On the OpenShift Container Platform Settings blade, provide an OpenShift Admin user password, enter your Red Hat subscription manager credentials, specify whether you want to configure an Azure Cloud Provider, and select your default router subdomain. Click OK (Figure 4-115).

Step	Label	Status
1	Basics Done	✓
2	Infrastructure Settings Done	✓
3	OpenShift Container Platform S... Configure OpenShift Container...	Active
4	Summary Red Hat OpenShift Container P...	>
5	Buy	>

**\* OpenShift Admin User Password** ✓

**\* Confirm OpenShift Admin User Password** ✓

**\* Red Hat Subscription Manager User Name** ✓

**\* Red Hat Subscription Manager User Password** ✓

**\* Red Hat Subscription Manager Pool ID** ✓

Configure Azure Cloud Provider

Yes No

Default Router Subdomain

nipro

**FIGURE 4-115** The selections in the OpenShift Container Platform Settings blade for a new Red Hat OpenShift Container Platform in the portal

- On the Summary blade, wait for the validation to pass, and click OK.
- On the Buy blade, click Purchase.

#### **MORE INFO** OPENSIFT CONTAINER PLATFORM PREREQUISITES

For an alternative method to deploy the OpenShift Container Platform using ARM templates instead of the marketplace, as well as detailed steps to complete the prerequisites see <https://github.com/Microsoft/openshift-container-platform>.

#### **MORE INFO** DEPLOYING OPENSIFT ORIGIN ON AZURE

For step-by-step instructions on how to deploy OpenShift Origin on Azure, including completing the prerequisites see <https://docs.microsoft.com/en-us/azure/virtual-machines/linux/openshift-get-started>.

## Provision applications by using Azure Quickstart Templates

Azure Quickstart Templates are community-contributed Azure Resource Manager (ARM) templates that help you quickly provision applications and solutions with minimal effort. You can search available Quickstart Templates in the gallery located at <https://azure.microsoft.com/resources/templates>.

Resources that are deployed as part of a Quickstart template can be thought of as related and interdependent parts of a single entity. ARM templates allow you to deploy, update, or delete all of the resources within the solution in a single, coordinated operation. You use a template for deployment and that template can work for different environments such as testing, staging, and production, while ensuring your resources are deployed in a consistent state.

Depending on the Quickstart Template you select, you will provide a set of parameters that get passed into the deployment command.

You can deploy a Quickstart Template using one of these methods (based on the example at <https://azure.microsoft.com/resources/templates/101-hdinsight-hbase-replication-geo>):

1. Using PowerShell, use the `New-AzureRmResourceGroupDeployment` cmdlet. You are prompted to supply values for the parameters. For example:

```
New-AzureRmResourceGroupDeployment -Name <deployment-name> -ResourceGroupName  
<resource-group-name> -TemplateUri https://raw.githubusercontent.com/azure/azure-  
quickstart-templates/master/101-hdinsight-hbase-replication-geo/azuredeploy.json
```

2. Using the Azure Command-Line Interface (CLI), use the `group deployment create` command. You are prompted to supply values for the parameters. For example:

```
azure config mode arm
```

```
azure group deployment create <my-resource-group> <my-deployment-name> --template-  
uri https://raw.githubusercontent.com/azure/azure-quickstart-templates/master/101-  
hdinsight-hbase-replication-geo/azuredeploy.json
```

3. Click the Deploy to Azure button, if provided. This opens a form for the Quickstart template in Azure, allowing you to enter the parameter values from within the portal (Figure 4-116).

Deploy HBase geo replication

Azure quickstart template

TEMPLATE

101-hdinsight-hbase-replication-geo

12 resources

Edit template

Edit parameters

Learn more

BASICS

\* Subscription

\* Resource group

Create new

Use existing

\* Location

West US 2

SETTINGS

\* Cluster Name Prefix ⓘ

\* Cluster Login User Name ⓘ

\* Cluster Login Password ⓘ

\* Ssh User Name ⓘ

\* Ssh Password ⓘ

☐ Pin to dashboard

Purchase

**FIGURE 4-116** An Azure Quickstart Template form in the Azure Portal after clicking a Deploy to Azure button

**MORE INFO    AZURE QUICKSTART TEMPLATE GALLERY**

Browse and search Quickstart Templates contributed by the community at <https://azure.microsoft.com/resources/templates>.

## Build applications that leverage Azure Marketplace solutions and services

The Azure Marketplace is an online applications and services marketplace that enables start-ups and independent software vendors (ISVs) to offer their solutions to Azure customers around the world. The marketplace makes it easier for consumers to search, purchase, and deploy a wide range of applications and services in just a few clicks. Some such applications and

**398    CHAPTER 4**    Design and implement Azure PaaS compute and web and mobile services

services include virtual machine images and extensions, APIs, applications, Machine Learning services, and data services.

You can subscribe to and deploy a product from the Azure Marketplace by visiting <https://azuremarketplace.microsoft.com/> or by clicking the Marketplace tile on the Azure Portal dashboard.

Pricing varies based on product types. ISV software charges and Azure infrastructure costs are charged separately through your Azure subscription. Pricing models include:

- **BYOL Model** Bring-your-own-license. You obtain outside of the Azure Marketplace the right to access or use the offering and are not charged Azure Marketplace fees for use of the offering in the Azure Marketplace.
- **Free** Free SKU. Customers are not charged Azure Marketplace fees for use of the offering.
- **Free Software Trial (Try it now)** Full-featured version of the offer that is promotionally free for a limited period of time. You are not charged Azure Marketplace fees for use of the offering through a trial period. Upon expiration of the trial period, customers are automatically be charged based on standard rates for use of the offering.
- **Usage-Based** You are charged or billed based on the extent of your use of the offering. For Virtual Machines Images, you are charged an hourly Azure Marketplace fee. For Data Services, Developer services, and APIs, you are charged per unit of measurement as defined by the offering.
- **Monthly Fee** You are charged or billed a fixed monthly fee for a subscription to the offering (from date of subscription start for that particular plan). The monthly fee is not prorated for mid-month cancellations or unused services.

You can find the offer-specific pricing details on the solution details page.

## Skill 4.9: Design and implement DevOps

---

DevOps is a combination of Development (Dev) and information technology Operations (Ops). It describes a set of practices emphasizing the collaboration between both teams, while automating software delivery and infrastructure changes with the ultimate goal of reliability and repeatability of these processes. Automation and repeatability allows for increased deployment frequency, as the manual burden of tending to all of the steps involved in deploying to one or more target environments has been removed. Some organizations use DevOps practices to deploy hundreds of times a day, which would otherwise be nearly impossible. DevOps improves reliability by ensuring each step of the software delivery or infrastructure change process is monitored, and any automated tests successfully pass.

**This skill covers how to:**

- Instrument an application with telemetry
- Discover application performance issues by using Application Insights
- Deploy Visual Studio Team Services with Continuous integration (CI) and Continuous development (CD)
- Deploy CI/CD with third-party platform tools (Jenkins, GitHub, Chef, Puppet, TeamCity)

## Instrument an application with telemetry

Application Insights is an extensible analytics service for application developers on multiple platforms that helps you understand the performance and usage of your live applications. With it, you can monitor your web application, collect custom telemetry, automatically detect performance anomalies, and use its powerful analytics tools to help you diagnose issues and understand what users actually do with your app. It works with web applications hosted on Azure, on-premises, or in another cloud provider. You can use it from web applications developed on multiple platforms, like .NET, Node.js, and J2EE. To get started, you just need to provision an Application Insights resource in Azure, and then install a small instrumentation package in your application. The things you can instrument are not limited just to the web application, but also any background components, and JavaScript within its web pages. You can also pull telemetry from host environments, such as performance counters, Docker logs, or Azure diagnostics.

Here is a comprehensive list of telemetry that can be collected by Application Insights.

From server web apps:

- HTTP requests
- Dependencies such as calls to SQL Databases; HTTP calls to external services; Azure Cosmos DB, table, blob storage, and queue
- Exceptions and stack traces
- Performance Counters, if you use Status Monitor, Azure monitoring, or the Application Insights collected writer
- Custom events and metrics that you code
- Trace logs if you configure the appropriate collector

From client web pages:

- Page view counts
- AJAX calls requests made from a running script
- Page view load data
- User and session counts
- Authenticated user IDs

From other sources, if you configure them:

- Azure diagnostics
- Docker containers
- Import tables to Analytics
- OMS (Log Analytics)
- Logstash

The standard telemetry modules that run “out of the box” when using the Application Insights SDK send load, performance and usage metrics, exception reports, client information such as IP address, and calls to external services. If you install the SDK in development, this allows you to send your own telemetry, in addition to the standard modules. This custom telemetry can include any data you wish to send.

#### **MORE INFO ABOUT APPLICATION INSIGHTS**

For additional information about Application Insights see <https://docs.microsoft.com/azure/application-insights>.

#### **MORE INFO SETTING UP APPLICATION INSIGHTS**

For more information about setting up Application Insights on the portal and within your application see <https://docs.microsoft.com/azure/application-insights/app-insights-create-new-resource>.

#### **MORE INFO COLLECT CUSTOM EVENTS AND METRICS IN APPLICATION INSIGHTS**

A good resource for collecting custom event and metrics telemetry in Application Insights see <https://docs.microsoft.com/azure/application-insights/app-insights-api-custom-events-metrics>.

## Discover application performance issues by using Application Insights

System performance depends on several factors. Each factor is typically measured through key performance indicators (KPIs), such as the number of database transactions per second or the volume of network requests your application can handle within a specified time frame. You can gather your application’s KPIs through specific performance measures, or a combination of metrics.

Application Insights can help you quickly identify any application failures. It also tells you about any performance issues and exceptions. With the right configuration and tooling, Application Insights can also help you find and diagnose the root causes of slowdowns and failures.



When you open any Application Insights resource you see basic performance data on the overview blade. Clicking on any of the charts allows you to drill down into the related data to see more detail and related requests, as well as viewing different time ranges.

**NOTE PERFORMANCE METRICS**

Earlier in this chapter performance metrics were discussed for API Apps and Logic Apps - and these are also similar across other resource blades in the Azure Portal.

Application Insights offers a full-screen, interactive performance investigator through the Performance blade. The dashboard arranges a set of performance-related metrics that you can use to quickly explore possible performance bottlenecks, and adds additional insights, such as common properties of selected requests. The common properties are the users' location, performance bucket (in milliseconds), and cloud role of the resource. This information can help you find common variables that affect groups of users, such as response times being lengthier for users coming from certain countries or regions (Figure 4-117).

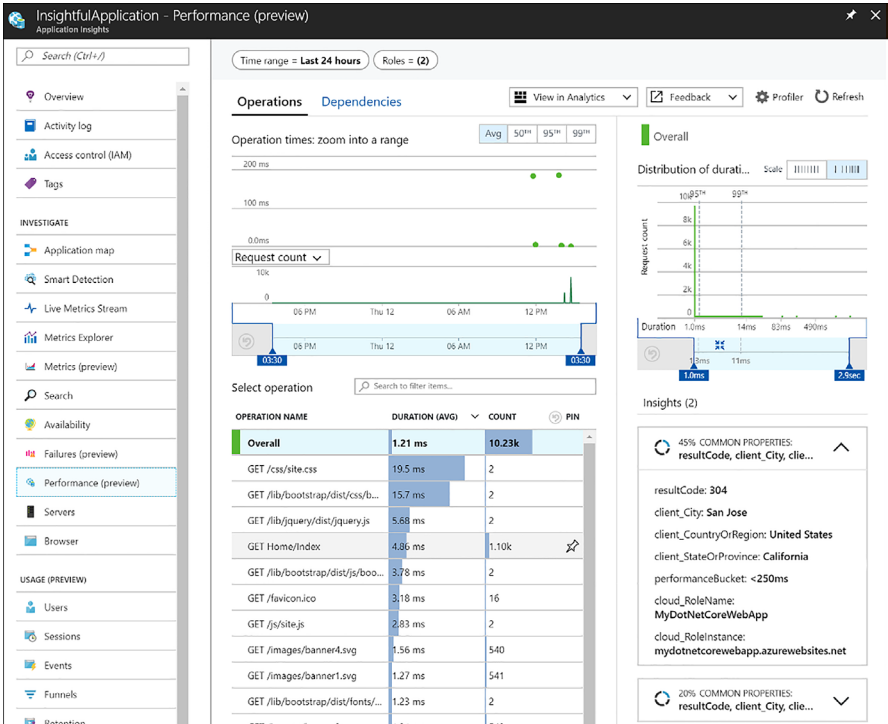


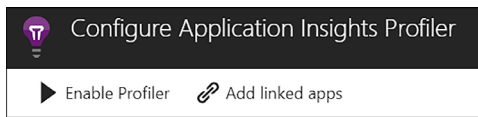
FIGURE 4-117 The Application Insights Performance blade

If your web application is built on ASP.NET or ASP.NET Core, you can turn on Application Insight's profiling tool to view detailed profiles of live requests. In addition to displaying 'hot paths' that are using the most response times, the Profiler shows which lines in the application

code slowed down performance. You can view the profile request details to see trace information, showing the call stack through your application. This level of detail allows you to quickly pinpoint issues and address them faster than digging through logs alone. There is little overhead running the profiler because it executes for two minutes per hour, but should provide a satisfactory sample set of data.

To enable the Profiler, follow these steps:

1. From the Application Insights resource in Azure, select Performance from the left-hand menu.
2. Select Profiler Rules from the top of the Performance blade.
3. Select Add Linked Apps from the top of the Configure Application Insights Profiler blade.
4. Select the application you wish to link to see all its available slots. Click Add to link them to the current Application Insights resource.
5. After linking your desired apps, select Enable Profiler from the top of the Configure Application Insights Profiler blade. Note, linked applications require Basic or above service plans to enable the profiler (Figure 4-118).



**FIGURE 4-118** The Application Insights Profiler actions to add linked apps and enable the Profiler

#### **MORE INFO ABOUT APPLICATION INSIGHTS PROFILER**

For additional information about using the Application Insights Profiler, see this reference: <https://docs.microsoft.com/azure/application-insights/app-insights-profiler>.

#### **MORE INFO MONITOR PERFORMANCE IN WEB APPLICATIONS**

For more information about using Application Insights to monitor performance in your web applications see <https://docs.microsoft.com/azure/application-insights/app-insights-web-monitor-performance>.

## Deploy Visual Studio Team Services with continuous integration (CI) and continuous development (CD)

Visual Studio Team Services (VSTS) is a collection of hosted DevOps services for application developers, including Build and Release services, which help you manage continuous integration and delivery of your applications.

Continuous Integration (CI) is a practice by which the development team members integrate their work frequently, usually daily. An automated build verifies each integration, typically along with tests to detect integration errors quickly, when it's easier and less costly to fix. Output, or artifacts, generated by the CI systems are fed to the release pipelines to streamline and enable frequent deployments. The Build service in VSTS helps you set up and manage CI for your applications.

Continuous Delivery (CD) is a process where the full software delivery lifecycle is automated, including tests, and deployed to one or more test and production environments. Azure App Services supports deployment slots, into which you can deploy development, staging, and production builds from the CD process. Automated release pipelines consume the artifacts that the CI systems produce, and deploys them as new versions and fixes to existing systems. Monitoring and alerting systems run continually to drive visibility into the entire CD process. The Release service in VSTS helps you set up and manage CD for your applications.

Because a key component of the Build system is integrating code changes and automating builds, you must host your source code in a version control system. VSTS provides two different version control systems:

- Git
- Team Foundation Version Control

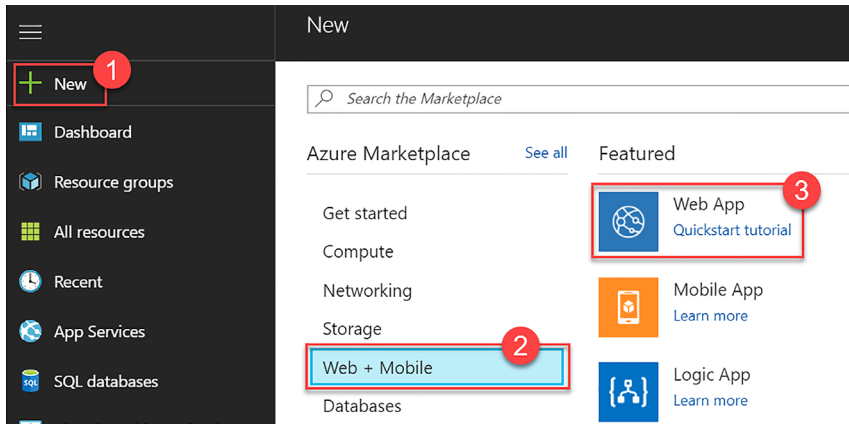
You can also host your source code in GitHub, Subversion, Bitbucket, or any other Git repository. The Build service can integrate with any one of these options.

VSTS build services provide preconfigured tasks to build many application types, such as .NET, Java, Node, Android, XCode, and C++. You can also run command line, PowerShell, or Shell scripts in your automation to support almost any type of application.

Azure App Services was mentioned earlier as a deployment target for the VSTS Release service. VSTS Release services can deploy to virtual machines, containers, on-premises and cloud platforms, or PaaS services. You can also publish your mobile applications to a store.

The following steps show one way to configure the CI/CD pipeline from the Azure portal (Figure 4-119):

1. Navigate to the portal accessed via *<https://portal.azure.com>*.
2. Select New on the command bar.
3. Select Web + Mobile, and then Web App.



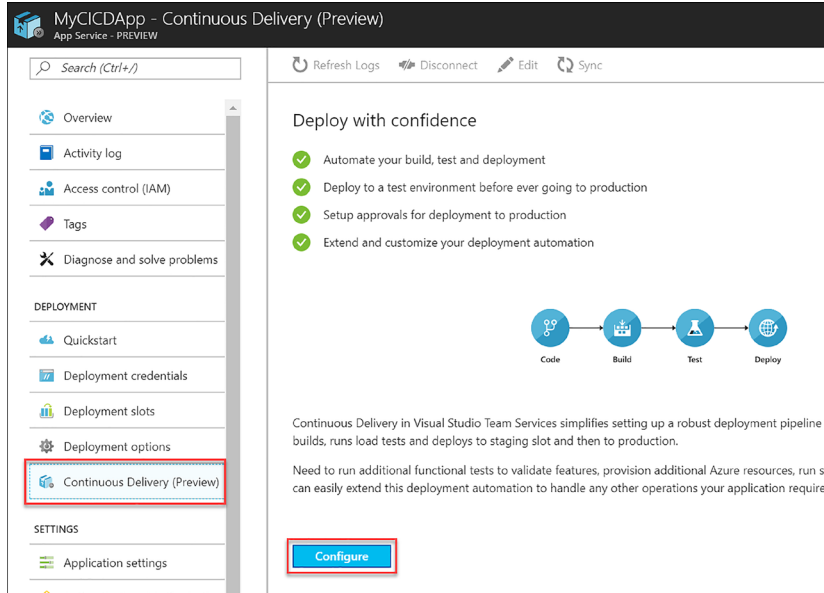
**FIGURE 4-119** Completing the Response action form for the new condition's "If true" block in the Logic App Designer

4. Provide a unique name for your web app, and then click Create (Figure 4-120).

The screenshot shows the 'Web App Create' blade. The 'App name' field is highlighted with a red circle and the number 1, and the 'Create' button is highlighted with a red circle and the number 2.

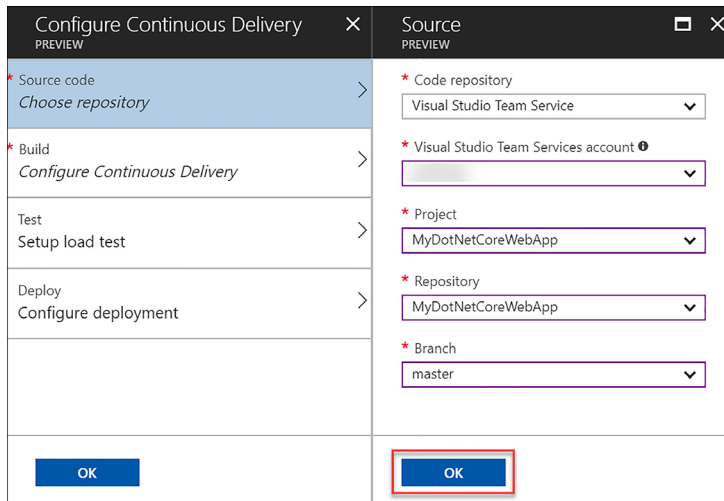
**FIGURE 4-120** The create Web App blade

5. After the new web app is provisioned open it in Azure portal, and then select Continuous Delivery from the left-hand menu. Click Configure on the Continuous Delivery blade (Figure 4-121).



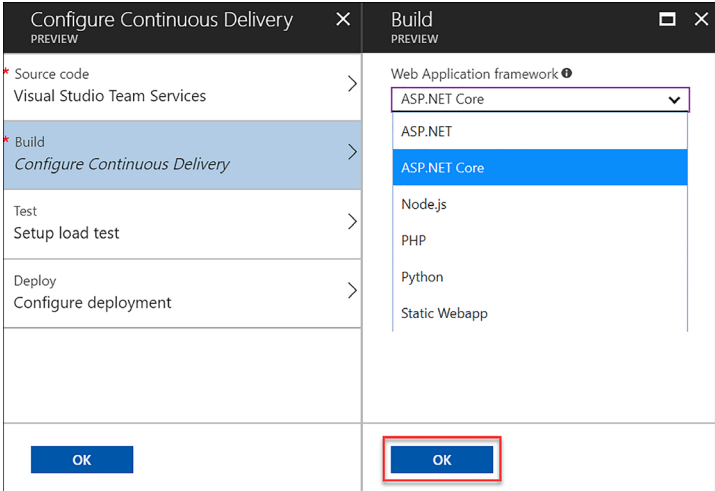
**FIGURE 4-121** The Continuous Delivery blade on the provisioned web app

6. Select Choose repository, and then select VSTS for the code repository. Select the VSTS account, project, repository, and source code branch from which you wish to deploy. Click OK (Figure 4-122).



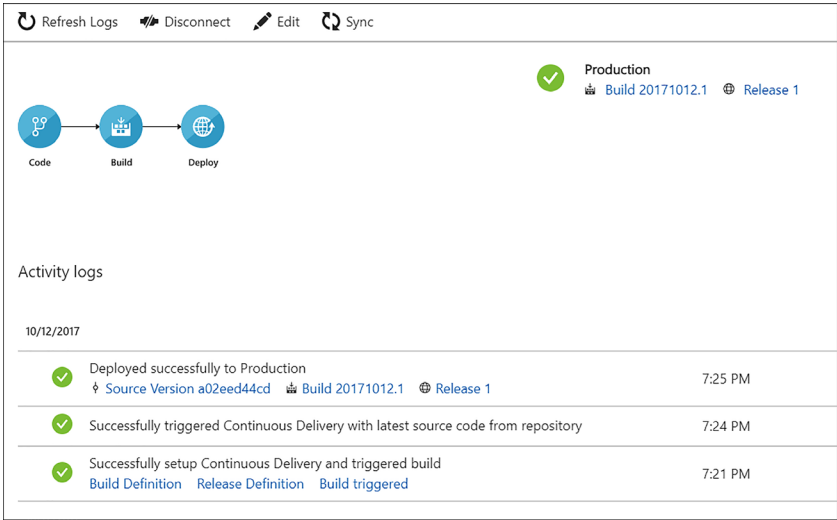
**FIGURE 4-122** The Continuous Delivery source code configuration options

7. Select **Configure Continuous Delivery**, and then your web application framework. In our example, we selected **ASP.NET Core**. Click **OK**. Skip the other two steps for now, and then click **OK** to complete the configuration (Figure 4-123).



**FIGURE 4-123** The Continuous Delivery build options

8. At this point, Azure Continuous Delivery configures and executes a build and deployment in VSTS. After the build completes, the deployment is automatically initiated. When you commit a change to the source code repository, the automated deployment appears in the Continuous Delivery application logs on your web app, as shown in Figure 4-124.



**FIGURE 4-124** The Continuous Delivery blade with activity logs showing the initial build

#### **MORE INFO ABOUT VSTS BUILD AND RELEASE SERVICES**

For additional information about the VSTS Build and Release services see <https://docs.microsoft.com/vsts/build-release>.

#### **MORE INFO THE MULTI-STAGE CONTINUOUS DEPLOYMENT (CD) PROCESS**

VSTS supports releasing to multiple environments, such as development, staging, QA, and production. To learn more about defining your multi-stage CD process see <https://docs.microsoft.com/vsts/build-release/actions/define-multistage-release-process>.

#### **MORE INFO TUTORIAL ON CREATING A CI PIPELINE WITH VSTS AND IIS**

To follow a tutorial showing how to create a continuous integration (CI) pipeline with VSTS and IIS on a VM see <https://docs.microsoft.com/azure/virtual-machines/windows/tutorial-vsts-iis-cicd>.

## Deploy CI/CD with third-party platform tools (Jenkins, GitHub, Chef, Puppet, TeamCity)

Azure allows you to continuously integrate and deploy with any of the leading DevOps tools, targeting any Azure service. Whether you are following your organization's established CI/CD procedures, or just getting started with DevOps, use the tools best-suited for your team.

If you are using VSTS to host your source code or as your CI service, you can use various build services, like Jenkins, through service hooks. In this way, you can use Jenkins for your continuous integration builds, or use both VSTS and Jenkins as for building parts of your solution. Refer to this tutorial for more information: <https://docs.microsoft.com/vsts/service-hooks/services/jenkins>.

In addition, Table 4-5 lists some popular DevOps tools that work with Azure.

**TABLE 4-5** References for using third-party DevOps tools with Azure

Tool	Description	More Information and Tutorials
Chef	Use Chef to automate workloads on Azure, whether IaaS, PaaS, cloud or hybrid, Windows or Linux	<a href="https://www.chef.io/implementations/azure/">https://www.chef.io/implementations/azure/</a> <a href="https://docs.microsoft.com/azure/virtual-machines/windows/chef-automation">https://docs.microsoft.com/azure/virtual-machines/windows/chef-automation</a>
Puppet	Use Puppet to automate the life-cycle of your entire Azure infrastructure	<a href="https://azuremarketplace.microsoft.com/marketplace/apps/PuppetLabs.PuppetEnterprise37">https://azuremarketplace.microsoft.com/marketplace/apps/PuppetLabs.PuppetEnterprise37</a> <a href="https://puppet.com/resources/whitepaper/getting-started-deploying-puppet-enterprise-microsoft-azure">https://puppet.com/resources/whitepaper/getting-started-deploying-puppet-enterprise-microsoft-azure</a>

Tool	Description	More Information and Tutorials
Jenkins	The Jenkins and Azure teams have been collaborating on making tighter integrations between the two. Benefit from the extensive tooling as a result	<a href="https://docs.microsoft.com/azure/virtual-machines/linux/tutorial-jenkins-github-docker-cicd">https://docs.microsoft.com/azure/virtual-machines/linux/tutorial-jenkins-github-docker-cicd</a> <a href="https://docs.microsoft.com/azure/jenkins/">https://docs.microsoft.com/azure/jenkins/</a> <a href="https://docs.microsoft.com/azure/storage/common/storage-java-jenkins-continuous-integration-solution">https://docs.microsoft.com/azure/storage/common/storage-java-jenkins-continuous-integration-solution</a>
TeamCity	Use TeamCity with Azure for a variety of DevOps processes, such as deploying Azure services or scaling out your build farm by having it automatically start agents on Azure when you need more power, and stop them, when they are no longer needed	<a href="https://confluence.jetbrains.com/display/TW/Microsoft+Azure+cloud">https://confluence.jetbrains.com/display/TW/Microsoft+Azure+cloud</a> <a href="https://blog.jetbrains.com/teamcity/2016/11/teamcity-dotnet-core/">https://blog.jetbrains.com/teamcity/2016/11/teamcity-dotnet-core/</a>

Out of the box, Azure App Services integrates with source code repositories such as GitHub to enable a continuous deployment workflow. This is the simplest way to integrate a CD process without the need for installing and configuring additional tools and services. Follow these simple steps to enable continuous deployment from a GitHub repository:

1. Publish your application source code to GitHub.
2. Open your app's Menu blade in the Azure portal, and then select Deployment Options under Deployment in the left-hand menu.
3. In the Deployment option blade, select Choose Source, and then select GitHub from the list of sources.
4. Select Authorization, and then click the Authorize button to enter your GitHub credentials. When authorized, click OK.
5. In the Deployment Option blade, select your project and branch from which you wish to deploy your app, and click OK.

App Service creates an association with the selected repository, pulls in the files from the specified branch, and maintains a clone of your repository for your App Service app. Now, when you push a change to your repository, your app is automatically updated with the latest changes. More information about this process can be found at: <https://docs.microsoft.com/azure/app-service/app-service-continuous-deployment>.



## Thought experiment

---

In this thought experiment, apply what you've learned about implementing App Services, Azure Functions, Azure Service Fabric, third-party PaaS, and DevOps to evaluate and determine a recommended set of features to use in a particular solution implementation.

You can find answers to this thought experiment in the next section. The following paragraphs describe the solution and the questions to answer.

You are designing a solution that issues certificates of insurance for end users. You are expecting insurance companies who you partner with to provide this service to their clients, your end user, through your solution. The following describes core components in the solution, and other requirements:

- Insurance companies can sign up with your service so that they can call your Policy Sync APIs and send insurance policy data using the X12 EDI standard. Their license with your API determines how much policy data they can upload to your service. This policy data is what supports certificate issuance to the end user owning the policy.
- Insurance companies can manage access to those policies through a Policy Management web application that allows them to create users who can later login and request certificates of insurance for their policy data.
- End users will, once invited by the insurance company, be able to login to the Certificate Issuance web application to request certificates of insurance on demand for their policies.
- When a certificate is requested, a workflow should be kicked off to generate a PDF from the policy data, save the PDF to a secure location from where it can be securely shared, and email a secure link to the PDF to a specified email address.
- While this is a new service, it is possible that many 100,000s of requests can be processed by a single insurance company per week so there is potential for large scale growth and the design must be ready to grow with demand.
- You are expecting to use a third-party Java-based executable component for PDF generation, alongside the other work, which will be based on ASP.NET Core.
- As a startup, you are looking for a solution that allows you to contain costs now, but grow into an architecture that can scale with your business growth.

Consider how you would answer the following questions for this solution:

1. How would you evaluate the core platform tools and hosting environment that you will use for the web apps and APIs? Consider these aspects:
  - A. Cost containment early on with potential for growth.
  - B. Manageability with a small team.
  - C. Support for polyglot development and third-party application components.

2. How will you control the onboarding process to use your Policy Sync APIs and subsequent throttling of their use by license?
3. How will you handle the inbound EDI requests and store those for the partner?
4. How will you prepare to scale the requests for certificates of insurance based on the potential growth?

## Thought experiment answers

---

This section contains the solution to the thought experiment.

1. Consider the following:
  - deploying the application to Web Apps on an App Service Plan that can scale as needed.
  - Consider if the main components of the application can be deployed as containers—in particular verifying that the Java component can be containerized. If so, standardizing around container deployments to Web Apps will keep things consistent and enable a future deployment to a container orchestration platform. If not, traditional Web App deployments for the ASP.NET Core applications will still reduce management overhead. The Java application may require a VM if it cannot be deployed to a Linux-based Web App due to underlying requirements.
  - Consider moving to a container orchestration platform, or Service Fabric cluster as the application needs to scale. Keep in mind the Service Fabric can support deployment of both ASP.NET Core applications alongside guest executables such as the Java application.
2. Consider using API Management for onboarding partners, setting up licensing, throttling access to the EDI process through licensing, and providing statistics on usage.
3. Consider using Logic App to handle X12 EDI transforms from API Management initiated calls. The Logic App can convert this payload to the target data format required for the application.
4. Look to scale out the requests for certificates of insurance by writing requests to a queue that triggers a Logic App to handle calls to generate PDFs and send emails through a workflow. Make sure the Java component is deployed to a tier that can scale independently given the potential for scale.

## Chapter summary

---

- Azure App Services provide a simple PaaS solution for deploying, managing, and scaling web applications, APIs, API Apps, Logic Apps, and Mobile Apps.
- API Apps and API Management both provide ways to publish APIs for partner integration. API Management provides richer features for partner management, licensing, throttling, security, and related management tools.
- Logic Apps provide an easy way to create workflows, modern integrations, and even legacy integration with EDI formats.
- Azure Functions provide an easy way to trigger workloads that can scale based on consumption or a hosting plan. There are many integration points for triggering functions including queues, HTTP requests, and data triggers.
- Azure Service Fabric is a modern orchestration platform that can support native services that leverage unique features such as stateful services and actor patterns, in addition to guest and container processes.
- Azure supports several third-party PaaS platforms for containers and microservices including Cloud Foundry and OpenShift.
- You have many choices for DevOps and CI/CD workflows in Azure including Application Insights for diagnostics, monitoring and alerts; and VSTS, Jenkins, Chef, Puppet and more for CI/CD integration.

# Index

## A

### access control

- anonymous access 133
- Azure Key Vault 228–232
- blobs 115–116
- DevTest Labs 95–100
- role-based 95–96
- shared access signatures 132–135
- storage 132–136
- stored access policies 135

### access keys 111–112, 179

### access policies 113, 262

### Active Directory (AD) 176–177

### activity logs 362

### actors 379, 387–388

### AD. *See* Active Directory

### ADE. *See* Azure Disk Encryption

### Advanced Message Queuing Protocol (AMQP) 239

### advanced rate limiting 359–360

### alerts

- configuration 55

### AMQP. *See* Advanced Message Queuing Protocol

### anonymous access 127, 133

### anonymous logs 141–142

### API Apps 305–318

- client code generation 314–316
- creating and deploying 305–310
- diagnostic logs 317–318
- discovery automation using Swashbuckle 310–314
- enabling CORS 314
- metrics 316, 318
- monitoring 316–318
- quotas 316

### API Management 281, 351–366

- adding product 353–354

### APIs

- adding operation to 355–356

- creating 352–356

- monitoring 362–363

- publishing 356

- rate limits 358–360

### caching 360–362

### components 351–352

### developer portal 363–366

### overview 351–352

### policies 356–358

### service creation 352–353

### API Proxies 377–378

### append blobs 115

### Application Insights 400–403

- performance issues and 401–403

### Profiler 403

### telemetry 400–401

### application logs 49, 52, 56–57

### applications

### ASP.NET 181

### availability of 57–66

### Azure Marketplace and 398–399

### Azure Service Fabric 379–392

### directory queries 207–216

### enterprise 265

### instrumenting, with telemetry 400–401

### integration with Azure AD 191–216

- directory creation 194–195

- preparation for 192–198

- querying directory 207–216

- viewing endpoints 197–198

- with OAuth 202–203

- with OpenID Connect 199–202

- with SAML-P 206–207

- with WS-Federation 203–206

### listener 242–245

### Microsoft Application Registry 208–209

### mobile 343–351

### multi-tier 58

## application tiers

- passwords 209
  - performance issues 401–403
  - provisioning, with Azure Quickstart Templates 397–398
  - registering 195–197, 221–223
  - remote debugging 16
  - sender 245–246
  - single page 192
  - using Azure AD B2B 225
  - using Azure AD B2C 216–225
  - using social identity provider authentication 217–225
  - web 281
    - Web/API 195–196
  - application tiers 58
  - App Service plans 282–287
    - creating 283–285
    - function integration with 379
    - pricing tiers 282
    - settings 286–287
  - A records 291–292, 293–294
  - ARM. *See* Azure Resource Manager
  - ARR affinity settings 288
  - ASP.NET 306
  - ASP.NET applications 181
  - asynchronous polling 340–341
  - asynchronous webhooks 341
  - authenticated logs 141–142
  - authentication
    - Azure AD 192–193
    - mobile apps 343, 346–348
    - multi-factor 210–216, 225
    - scenarios 193
    - social identity provider 217–225
    - storage account 135–136
    - users 203–206
  - authorization
    - mobile apps 343
  - authorization protocols 202–203
  - automatic asynchronous replication 150
  - automatic failover 172–173
  - Autoscale
    - configuration 25–29
  - AutoScale 18
  - auto-shutdown policy 87–89
  - auto-start policy 90–91
  - Auto Swap settings 289
  - availability
    - high 59
    - sets 19
      - application tiers and 58
      - configuration 58–60
      - Load Balancer and 60–66
    - virtual machines 57–66
  - AZCopy 44
  - Azure Active Directory (Azure AD)
    - application integration 191–216
      - directory creation 194–195
      - preparation for 192–198
      - querying directory 207–216
      - registering application 195–197
      - viewing endpoints 197–198
    - with OAuth 202–203
    - with OpenID Connect 199–202
    - with SAML-P 206–207
    - WS-Federation 203–206
  - B2B 225
  - B2C 216–225
  - code samples 194
  - documentation 192
  - PowerShell with 192
  - uses of 191
- Azure AD B2B 225
  - Azure AD B2C 216–225
    - application registration 221–223
    - identity provider configuration 223–224
    - policy configuration 224
    - tenant creation 218–221
  - Azure AD Connect 195
  - Azure AD Graph API 207
  - Azure App Services 281, 404
    - API Apps 305–318
    - integration with source code repositories 409
    - Logic Apps 318–342
    - Mobile Apps 343–351
    - plans 282–287
    - quotas 296
    - Web Apps 282–305
  - Azure Autoscale. *See* Autoscale
  - Azure Command Line Interface (Azure CLI) 7–8
    - Web Apps and 296
  - Azure-connected functions 372–374
  - Azure Cosmos DB accounts
    - creating 164
  - Azure Cosmos DB DocumentDB 160, 162–177
    - accessing from REST API 174
    - choosing surface 163
    - consistency 170

- database and collections creation 164–167
- Graph API database creation 168
- GraphDB API queries 168
- MongoDB database and 169
- multiple regions, managing 171–173
- query documents 167–168
- scaling 169–171
- security 174–176
- stored procedures 173–174
- users and permissions 175
- Azure Cosmos DB Table API 131, 163
- Azure Disk Encryption (ADE) 46–47
- Azure Files 109
  - connections to 120–121
  - storage 119
- Azure File storage 41–45
- Azure Functions 281, 366–379
  - Azure-connected functions 372–374
  - creating 367–368
  - custom bindings 376–377
  - debugging 377
  - event processing 371–372
  - integration with App Service plan 379
  - integration with storage 374–376
  - overview 366–367
  - proxies 377–378
  - triggers 376–377
  - webhook function, implementing 369–371
- Azure Key Vault 46, 225–236
  - access management 228–232
  - configuration 226–228
  - HSM protected keys 232–233
  - key rotation 235–236
  - logging implementation 233–235
  - uses of 225
- Azure Marketplace 2, 398–399
- Azure Portal
  - adding owners and users to lab with 97–98
  - API app creation from 306
  - Autoscale configuration with 25–29
  - custom image creation with 74–75
  - load balancing with 61–66
  - metrics monitoring with 55–56
  - monitoring configuration with 49–54
  - Scale Set deployment using 19–21
  - scaling VMs using 17
  - VM configuration using 14–15
- Azure queues 253
- Azure Queues 372–373
- Azure Quickstart Templates 281, 397–398
- Azure Relay 236, 239–253
  - Hybrid Connections 240–247
  - namespaces 240
  - scaling 273–274
  - WCF Relay 247–253
- Azure Resource Manager (ARM)
  - deployment 111
  - templates 2, 22, 100–104, 393, 397–398
  - virtual machines
    - availability 57–66
    - configuration management 7–16
    - DevTest Labs 67–104
    - load balancing 61–67
    - monitoring 47–57
    - scaling 16–29
    - storage 29–47
    - workload deployment 1–7
  - Web Apps and 296
- Azure Samples 210
- Azure Search 182–186
  - adding data 183–184
  - index search 185
  - search results 186
  - service indexes 182–183
- Azure Service Fabric 281, 379–392
  - actors-based service 387–388
  - applications
    - adding web front end to 383–387
    - creating 380–383
    - deployment to container 388–390
    - migration from cloud services 390
    - scaling 390–391
  - clusters 391–392
  - monitoring and diagnose services 388
  - overview 379–380
- Azure SQL Database 123
  - backups 147
  - database tiers, choosing 144–147
  - geo-replication 149–150
  - graph database functionality in 160–161
  - implementation 144–161
  - managed elastic pools 157–159
  - performance level, choosing 144–147
  - point in time recovery 147–149
  - scaling 155–157
  - schema and data, import and export 151–155
  - secondary databases
    - offline 150

- online 150–151
- SQL Data Sync 159–160
- vs. Azure Tables 123
- Azure Storage. *See* storage
- Azure Storage accounts 42
- Azure Storage Analytics 132
- Azure Storage Queue 128–131
  - adding messages to 128–129
  - batch message retrieval 130
  - processing messages 129–130
  - scaling queues 130–131
- Azure Storage Tables 48–49
- Azure Tables 122–128
  - creating 123–124
  - CRUD operations 123–127
  - deleting records 127
  - inserting multiple records 125–126
  - partitions 123–124, 128–129
  - querying, using OData 127
  - record insertion 124–125
  - records in partitions 126
  - transactions 125–126
  - updating records 126–127
  - vs. Azure Cosmos DB Table API 131
  - vs. Azure SQL Database 123
- Azure Virtual Machine Agent. *See* VM Agent

## B

- back off polling 131
- backups
  - Azure SQL Database 147
- BACPAC files 151–155
- batch messages 130
- blobs 30, 42, 109–122
  - about 110
  - access control 132
  - append 115
  - block 115, 141
  - containers 112–113, 117, 122
  - copying 116
  - geo-replication for 41
  - hierarchies 117–118
  - integration of function with 374–376
  - leasing 119–120
  - metadata 113–114
  - page 115
  - partition keys 122

- read and change data 112–113
- SAS tokens 133
- scaling 119–120
- secure access 115–116
- storage account creation 110–112
- streaming 115
- types of 115
- URIs 113
- Blob storage 30, 111
  - Content Delivery Network with 116–117
  - naming requirements 42
- block blobs 115, 141
- boot diagnostic logs 57
- BrokeredMessage type 257
- business-to-business (B2B) workflows
  - Logic Apps supporting 322–331

## C

- cache
  - CDN 116–117
  - configuration 39–41
  - expiry period 117
  - host 39–41
  - local 39–41
  - providers 181
  - Redis 177–182
  - tiers 177–178
- caching
  - adding 360–362
- capacity metrics 137
- CDN. *See* Content Delivery Network
- certificate authority (CA) 291
- certificate permissions 230
- certificates
  - SSL 291, 294–295
- Chef 408
- cifs-utils package 121
- CLI. *See* Azure Command Line Interface
- client-side logging 141
- Cloud Foundry 392–393
- cloud services 390
- clusters
  - Redis 180
  - Service Fabric 391–392
- CNAME records 291, 292, 294
- collections

- Cosmos DB API 164–167, 169–170
- compute resources 282
- compute time 119, 123
- configuration
  - alerts 55
  - API Management policies 356–358
  - Autoscale 25–29
  - availability sets 58–60
  - Azure AD B2C policies 224
  - Azure Key Vault 226–228
  - Content Delivery Network 116–117
  - custom domains 118, 292–294
  - DevTest Labs
    - cost management 92–95
    - policies and procedures 83–91
  - diagnostics 49–54
  - disk caching 39–41
  - endpoint monitoring 300–303
  - geo-replication 41
  - identity providers 223–224
  - Load Balancer 61–67
  - Mobile Apps 345–346
  - monitoring 49–54
  - proxies 377–378
  - shared storage 41–45
  - SSL certificates 294–295
  - Storage Analytics Logging 140–141
  - Storage Analytics Metrics 137–140
  - storage pools 32–39
  - Web Apps 287–295
- Configuration keyword 11–12
- configuration management
  - virtual machines 7–16
    - using Azure Portal 14–15
    - using DSC 13–15
    - with Custom Script Extension 8–10
    - with DSC 11–12
- configuration scripts 13
- connection strings
  - accessing 290
  - settings 289
- connectivity issues 239
- consistency 131, 170, 171
- consumer groups 269
- Consumption plans 379
- containers 112–113, 117, 122, 379
  - Service Fabric application deployment to 388–390
  - Windows 390
- Content Delivery Network (CDN) 116–117

- continuous development (CD)
  - VSTS with 404–409
  - with third-party platform tools 408–409
- continuous integration (CI)
  - VSTS with 404–409
  - with third-party platform tools 408–409
- CORS 314
- cost by resource 95
- cost management
  - DevTest Labs 92–95
- Cost Trend chart 92
- crash dumps 49, 53–54
- Create Alert Rule dialog box 146–147
- CreateServiceReplicaListeners() 383
- credentials
  - Event Hub 267
  - Service Bus queue 255–256
  - Service Bus topic 262
  - WCF Relay 249–250
- Cross-Origin Resource Sharing (CORS) 136–141
- CRUD operations 348
- custom actions
  - in Logic Apps 340–341
- custom domains
  - configuration 118, 292–294
  - mapping names 291–292
- custom images
  - creating 72–76
    - from provisioned VM 72–74
    - with Azure Portal 74–75
    - with PowerShell 76
  - deleting 77
  - pros and cons of 72
  - Scale Set deployment using 22–24
- custom resources 12
- Custom Script Extension
  - VM configuration with 8–10

## D

- data
  - consistency 170, 171
  - import and export 151–155
  - loading into storage account 112
  - logging. *See* logs
  - persistence 178–179
  - read and change 112–113
  - redundancy 149
  - replication 111, 171–172



## data access

- storing
  - using blobs 115
- streaming 115
- validation 135
- data access 343
- databases
  - Cosmos DB API 164–167
  - graph 163, 168
  - graph database functionality 160–161
  - relational 161, 171
  - sharding 156
- database throughput units (DTUs) 144–146, 157–159
- data products 109
- datasets
  - sharding large 122–123
- dead letter queues 259
- deployment
  - API Apps 305–310
  - ARM templates 22
  - Azure Relay namespaces 240
  - Azure Resource Manager 111
  - Hybrid Connection 240–241
  - Mobile Apps 345
  - Service Fabric applications 388–390
  - Virtual Machine Scale Sets 18–24
  - WCF Relay 248–249
- Desired State Configuration (DSC) 7
  - Configuration keyword 11–12
  - configuration management 7–8, 11–15
  - custom resources 12
  - Local Configuration Manager 12–13
  - resources 11
- developer portal 363–366
- DevOps 399–409
  - Application Insights 400–403
  - overview 399
  - telemetry 400–401
  - third-party platform tools 408–409
  - Visual Studio Team Services 403–408
- DevTest Labs 67–104
  - adding owner or user 97–99
  - adding VM 70–71
  - ARM templates 100–104
  - configuration
    - cost management 92–95
    - policies and procedures 83–91
  - custom images
    - creating 72–76
    - deleting 77
  - environments 100–104
  - formulas
    - creating 77–81
    - deleting 83
    - modifying 81–82
    - pros and cons of 77
  - lab creation 67–70
  - lab settings 99–100
  - policies and procedures
    - auto-shutdown policy 87–89
    - auto-start policy 90–91
    - per lab policy 86–87
    - per user policy 85–86
    - set expiration date policy 91
    - virtual machine sizes policy 83–85
  - security access 95–100
- diagnostic infrastructure logs 49, 54, 56–57
- diagnostic logs 317–318, 362
- diagnostics 48–50
  - boot 57
  - configuration 49–54
  - services 388
  - Web Apps 296–300
- Diagnostics extension 48
- differential backups 147
- directories
  - creating 194–195
  - premium 195
  - querying 207–216
- disaster recovery 149–150
- disk caching
  - configuration 39–41
- disks
  - encryption 46–47
  - managed 30–31
  - premium 30–31, 45
  - standard 30–31, 45
  - storage 30–32
  - unmanaged 30–31
- Docker 394
- DocumentDB API 173
- documents
  - retrieving from Azure Cosmos DB DocumentDB 167–168
  - searching 185
- domain name system (DNS) records 291
- domains
  - custom 118, 291–294
  - fault 58
  - update 58

domain specific language (DSL) 161  
 DSC. *See* Desired State Configuration  
 dump files 49  
 duplicate logs 140

## E

easy tables 348  
 eDTUs 157–159  
 Elastic Database Tools 156–157  
 elastic Database Transaction Units (eDTUs) 157–159  
 elastic pools 157–159  
 encryption  
   at rest 174  
   Azure Disk Encryption 46–47  
   disk 46–48  
   in flight 174  
   storage service 111  
   Storage Service Encryption 46–47  
 endpoints  
   HTTP 338–339  
   listener 250–252  
   monitoring 300–303  
   OAuth 202  
   relay 250–252  
   SAML-P 207  
   WS-Federation 206  
 enterprise applications 265  
 Enterprise Integration Pack 322–323, 331, 333. *See also* integration accounts  
 environments  
   DevTest Lab 100–104  
 error message logs 297  
 ETag 124  
 Event Hubs 237, 265–270  
   connection strings 268  
   creating 266–267  
   credentials 267  
   monitoring 276–277  
   overview 265  
   pricing tiers 272  
   properties 266  
   receiving messages from consumer groups 269–270  
   scaling 275  
   sending messages to 268  
   when to use 277–278  
 event logs 52–53, 56–57, 297  
 EventProcessorHost 270, 278

event tracing 49  
 external users  
   adding to DevTest Labs 97–99

## F

failed request trace logs 297  
 fault domains 58  
 file locking 43  
 files. *See also* Azure Files  
   accessing 42  
   BACPAC 151–155  
   connections to 120–121  
 file shares 42–45  
   accessing files 44–45  
   creating 43  
   mounting 43–44  
 file storage. *See* storage  
 firewalls  
   network 175  
 First In First Out (FIFO) buffer 253  
 formulas  
   creating 77–81  
   deleting 83  
   modifying 81–82  
   pros and cons of 77  
 full backups 147  
 full text search 182  
 functions 164. *See* Azure Functions

## G

General Purpose storage 111  
 geo-replication 149–150  
   configuration 41  
 Get-AzureRmAdUser cmdlet 99  
 Get-AzureRmResource cmdlet 76, 99  
 GetContainerReference() 114  
 GetMessage() 129  
 GetMessages() 130  
 Git 404  
 GitHub 409  
 Graph API databases  
   creating 168  
 graph databases 160–161, 163  
 GraphDB API queries 168  
 Gremlin 161, 163, 168  
 guest executables 379

## H

- Hardware Security Module (HSM) protected keys 232–233
- HDD disks 30
- high availability 59
- high availability/disaster recovery (HADR) scenarios 171
- host cache 39–41
- HTTP endpoints 338–339
- HTTP protocol 239, 248
- HTTP requests 115
- HTTPS requests 115, 174
- Hybrid Connections 240–247
  - Azure Relay namespace deployment 240
  - configuration retrieval 241–242
  - deployment 240–241
  - listener application creation 242–245
  - running applications 246–247
  - sender application creation 245–246

## I

- IaaS.Diagnostics extension 48
- identity providers 223–224. *See also* social identity provider authentication
- IIS logs 49, 57
- IIS settings 290
- ImageToUpload variable 112
- incremental log backups 147
- Infrastructure-as-a-Service (IaaS) 1
- InsertOrReplace() 126
- integration accounts
  - adding agreements 325–326
  - adding maps to 332–333
  - adding partners to 324–325
  - adding schemas to 332
  - creating 322–324
  - linking Logic app to 326–327
- Internet of Things (IoT) 265, 278
- IP addresses
  - changes in 294

## J

- Jenkins 408, 409
- JSON document storage 160, 162–163, 171. *See also* Azure Cosmos DB DocumentDB

## K

- key performance indicators (KPIs) 401
- key permissions 229
- key-value stores 160, 177
- Key Vault 46, 225–236
- Kubernetes 394
- Kudu 299, 300

## L

- lambda LINQ 167
- leases
  - blob 119–120
- LINQ queries 167
- Linux virtual machines
  - creating 6
  - metrics data 48
- listener applications 242–245
- listener endpoints 250–252
- Load Balancer
  - availability sets and 60–66
- local cache 39–41
- Local Configuration Manager 12–13
- locally redundant replication 41
- Locally Redundant Storage (LRS) 111
- Logic App Designer 318
- Logic Apps 318–342
  - creating
    - connecting SaaS services 319–322
    - with B2B capabilities 322–331
    - with XML capabilities 331–337
  - custom and long-running actions 340–341
  - HTTP endpoints for 338–339
  - integration accounts
    - adding agreement 325–326
    - adding maps to 332–333
    - adding partners to 324–325
    - adding schemas to 332
    - creating 322–324
    - linking to 326–327
  - metric 341–342
  - monitoring 341–342
  - overview 318
  - receiving data in 327–331
  - triggering from another app 337–339
- Login-AzureRmAccount cmdlet 98
- logs
  - activity 362

- analyzing 141–143
- anonymous 141
- API Apps 317–318
- application 49, 52, 56–57
- authenticated 141
- boot diagnostics 57
- client-side 141
- configuration 49–54
- diagnostic 297–300, 317–318, 362
- diagnostic infrastructure 49, 54, 56–57
- duplicate 140
- error message 297
- event 49, 52–53, 56–57, 297
- failed request tracing 297
- finding 142–143
- IIS 49, 57
- Key Vault 233–235
- metadata 143
- operation 142
- retention 140
- status messages 142
- storage 140–143
- Storage Analytics 132, 140–141
- system 48
- viewing 56–57
- viewing, with Microsoft Excel 143
- Web Apps 296–300
- web server 297, 318
- long-running actions 340–341
- LRS. *See* Locally Redundant Storage

## M

- managed disks 30
- managed elastic pools 157–159
- maps
  - adding to integration account 332–333
  - XML 331
- messages
  - adding to queue 128–129
  - batch 130
  - batching 264, 274
  - duplicate 259
  - filtering 264–265
  - identifiers 129
  - invisibility 129
  - pre-fetching 274
  - processing 129
  - receiving
    - from consumer group 269–270
    - from queues 257–259
    - from subscriptions 263–264
  - sending
    - through relay 252
    - to a topic 262–263
    - to Event Hubs 268
    - to queues 256–257
- messaging protocols 238–239
- messaging strategy 236–278
  - Azure Relay 236, 239–253
  - Event Hubs 237, 265–270
  - Hybrid Connections 240–247
  - Notification Hubs 237, 270–271
  - scaling and monitoring 271–277
  - Service Bus queues 237, 253–259
  - Service Bus topics and subscriptions 259–265
  - WCF Relay 247–253
- metadata
  - log 143
  - reading 114
  - setting 113–114
  - system properties 113, 114
  - user-defined 113–114
  - WS-Federation 206
- metrics 48, 132
  - analyzing 139
  - API Apps 316, 318
  - capacity 137
  - levels of 137–138, 138
  - Logic Apps 341–342
  - monitoring 55–57, 139
  - performance 402
  - retention 138
  - storage 137–140
  - transaction 137
  - Web Apps 300–303
- MFA. *See* multi-factor authentication
- Microsoft Application Registry 208–209
- Microsoft Azure Traffic Manager 304
- Microsoft Excel
  - viewing logs with 143
- Microsoft Graph API 207–210
- Microsoft SQL Server 164
- minidumps 49
- Mobile Apps 343–351
  - authentication 346–348
  - client application 346

- configuration 345–346
- creating 343–346
- deployment 345
- development environment 344–345
- offline sync for 348–350
- overview 343
- push notifications 350–351
- target device platforms 344
- MongoDB database 163, 169
- monitoring
  - alerts 55
  - API Apps 316–318
  - APIs 362–363
  - diagnostics 47–49
  - Event Hubs 276–277
  - Logic Apps 341–342
  - metrics 55–57
  - Notification Hubs 277
  - Service Bus features 275–277
  - services 388
  - storage metrics 139
  - viewing logs 56–57
  - virtual machines 47–57
    - configuration 49–54
  - Web Apps 296
- Monthly Estimated Cost Trend chart 92
- multi-factor authentication (MFA) 210–216, 225
- multi-tier applications 58

## N

- namespaces 274
  - Azure Relay 240
  - Event Hubs 275
  - Service Bus 237–238, 273
- .NET Storage Client Library 141
- NetTcpRelayBinding relay 248
- network firewalls 175
- network isolation 179–180
- New-AzureRmResourceGroupDeployment cmdlet 76
- New-AzureRmRoleAssignment cmdlet 99
- Newtonsoft.Json 307
- Node.js 310
- nodes 160–161
- Notification Hubs 237, 270–271, 272
  - monitoring 277
  - when to use 277–278

## O

- OAuth 2.0 198, 202–203
- OData
  - querying using 127
- offline secondary databases 150
- offline sync 343, 348–350
- online secondary databases 150–151
- OpenAPI Specification (OAS) 310
- OpenID Connect 192–193, 198, 199–202, 217
- OpenShift Container Platform 394–396
- OpenShift Origin 394, 396
- Open-source Cloud Foundry (OSS CF) 392
- operation logs 142
- owners
  - adding to DevTest Labs 97–98

## P

- page blobs 115
- partition keys 123–125, 126–128, 170, 269
- partitions 128–129, 169–170, 269, 274, 275
- partner-managed identities 225
- passwords
  - application 209
- PeekLock mode 257
- Performance Counters 51
- performance metrics 402
- permissions 95
  - certificate 230
  - Cosmos DB 175
  - key 229
  - secret 230
- Pivotal Cloud Foundry (PCF) 392–393
- Placement groups 19
- plain-old CLR objects (POCOs) 166
- Platform-as-a-Service (PaaS) 1, 281
  - Azure Marketplace 398–399
  - Azure Quickstart Templates 397–398
  - Cloud Foundry 392–393
  - OpenShift Container Platform 394–396
  - third-party 392–399
- point in time restores 147–149
- PowerShell
  - accessing file share using 44
  - adding external users to lab with 98–99
  - availability set configuration using 60

- Azure AD management with 192
- custom image creation with 76
- disk encryption using 46–47
- scaling VMs with 17
- Web Apps and 296
- PowerShell Desired State Configuration. *See* Desired State Configuration
- pre-fetching messages 274
- premium directories 195
- premium disks 30–31, 45
- premium storage 45
- pricing tier 271–272, 282
- primary keys 132
- proxies 377–378
- Publish-AzureRMVmDscConfiguration cmdlet 13
- Publish-AzureVMDscConfiguration cmdlet 14
- Puppet 408
- push notifications 343, 350–351

## Q

- queues 128–131
  - Azure 253
  - SAS tokens for 134–135
  - Service Bus 237, 253–259
- QueueSender 256
- Quickstart Templates 397–398
- quotas
  - API Apps 316

## R

- rate limits
  - for APIs 358–360
- RBAC. *See* Role-Based Access Control
- ReceiveAndDelete mode 257
- ReceiveBatch() 264
- ReceiveBatchAsync() 264
- receiver keys 250
- records
  - deleting 127
  - in partitions 126
  - inserting, into tables 124–125
  - inserting multiple 125–126
  - updating 126
- Redis caching 177–182
- Redis clusters 180
- relational databases 161, 171

- relationships 160–161
- relays 237. *See also* Azure Relay; *See also* WCF Relay scaling 273–274
- relay service endpoints 250–252
- remote debugging 16, 289
- Remote Desktop (RDP) 44
- replication
  - automatic asynchronous 150
  - data 111, 171–172
  - geo-replication 41, 149–150
  - locally redundant 41
  - options 111
- Request Units (RUs) 131
- resilience
  - Web Apps 303–305
- resources
  - custom 12
  - DSC 11
- REST API 174
- REST APIs 43, 45
- RESTful APIs 296, 305, 310, 340
- restores
  - point in time 147–149
- REST services 248
- retention
  - backups 147
- Role-Based Access Control (RBAC) 95–96
- roles 95, 96
- row keys 123, 124
- RPC listeners 383
- RUs. *See* Request Units

## S

- SAML 2.0 Protocol (SAML-P) 192, 198, 206–207
- SAS. *See* secure access signature
- SBMP. *See* Service Bus Messaging Protocol
- Scale Sets 18–24
- scaling
  - Azure Cosmos DB DocumentDB 169–171
  - Azure SQL Database 155–157
  - blob storage 119–120
  - Event Hubs 275
  - queues 130
  - relays 273–274
  - Service Bus features 272–273
  - Service Bus queues 274
  - Service Bus topics 274

- Service Fabric apps 390–391
- Web Apps 303–305
- schema
  - import and export 151–155
- schemas
  - adding to integration account 332
  - XML 331
- scopes 95
- search
  - Azure Search 182–186
  - full text 182
- Search Units (SUs) 182
- secondary databases
  - offline 150
  - online 150–151
- secondary keys 132
- secret permissions 230
- secrets
  - managing, with Key Vault 225–236
- secure access signature (SAS)
  - data validation 135
  - tokens
    - recommendations for 135
    - renewing 135
- Secure Socket Layer (SSL) 291
- security. *See also* access control; *See also* authentication
  - Cosmos DB 174–176
  - DevTest Labs 95–100
  - Redis 179–180
- Select-AzureRmSubscription cmdlet 76, 98
- sender applications 245–246
- sender keys 250
- Server Manager 32
- Server Message Block (SMB) protocol 42
- Service Bus
  - messaging protocols 238–239
  - monitoring 275–277
  - namespaces 237–238, 273
  - pricing tier 271–272
  - quotas 273
  - scaling features 272–273
  - when to use 277–278
- Service Bus Messaging Protocol (SBMP) 239
- Service Bus queues 237, 253–259, 278
  - connection strings 256
  - creating 255
  - credentials 255–256
  - dead letter 259
  - duplicate messages and 259
  - monitoring 275
  - properties of 254
  - receiving messages 257–259
  - scaling 274
  - sending messages to 256–257
- Service Bus subscriptions 237, 259–265, 278
  - creating 261
  - filtering messages 264–265
  - properties 260
  - receiving messages from 263–264
- Service Bus topics 237, 259–265, 278
  - creating 261
  - credentials 262
  - filtering messages 264–265
  - monitoring 276
  - properties 260
  - scaling 274
  - sending messages to 262–263
- Service Fabric. *See* Azure Service Fabric
- Service Tiers 144–147
- session state 181
- SetAzureRmVmDscExtension cmdlet 13
- set expiration date policy 91
- shard maps 156
- shared access signatures 116
- Shared Access Signature (SAS) 127
- shared access signatures (SAS) 132–135
- Shared Key 116
- Shared Key Lite 116
- shared storage
  - configuration 41–45
- single page applications (SPAs) 192
- Site Control Management (SCM) website 300
- SOAP protocol 248
- social identity provider authentication 217–225
- Software as a Service (SaaS) 319–322
- SQL Data Sync 159–160
- SQL queries 167
- SQL Server
  - virtual machines
    - creating 7
- SQL Server Management Studio (SSMS) 151–155
- SSD disks 30
- SSH keys 393
- SSH public keys
  - generation of 6
- SSL certificates 291, 294–295

- standard disks 30–31, 45
- standard storage 45
- stateful Fabric-aware services 379
- stateless Fabric-aware services 379
- status messages 142
- storage
  - access control 132–136
  - access policies 113
  - accounts 42
    - blob 110–112
      - CDN configuration 116–117
      - geographic location 111
      - key regeneration 135–136
      - read and change data 112–113
    - types 111
  - Azure File 41–45
  - Azure Files 119
  - Azure Storage Queue 128–131
  - Azure Tables 122–128
  - blob 30, 42
  - blobs 109–122
    - append 115
    - block 115
    - copying 116
    - hierarchies 117–118
    - leasing 119–120
    - metadata 113–114
    - page 115
    - read and change data 112–113
    - scaling 119–120
    - secure access 115–116
    - streaming data 115
    - types of 115
  - containers 112–113, 117, 122
  - Cross-Origin Resource Sharing 136–141
  - custom domains 118
  - disk caching 39–41
  - disk encryption 46–48
  - geo-replication 41
  - integration of function with 374–376
  - locally redundant 111
  - logs 140–143
  - metrics 132
  - pools 32–39
  - shared 41–45
  - virtual machines 29–47
    - capacity planning 30–32
    - premium 45
    - standard 45
- storage access signatures (SAS)
  - tokens

- creating 133–134
- Storage Analytics Logging 140–141
- Storage Analytics Metrics 132
  - analysis 139
  - configuration 137–140
  - monitoring 139
- Storage API 114
- Storage Client Library 44–45, 133
- Storage Service Encryption (SSE) 46–47
- stored access policies 135
- stored procedures 164, 170, 173–174
- Stream Analytics 278
- Swagger 308–310, 312–316, 340
- Swagger Specification 310
- Swashbuckle 308, 310–314, 340
- Sync Groups 159
- Sync Schemas 159
- SyncTable 348
- Syslog 48
- system logs 48
- system properties
  - metadata 113
  - reading 114

## T

- tables. *See also* Azure Tables
  - easy 348
  - SAS tokens for 134
- TeamCity 409
- Team Foundation Version Control 404
- telemetry 400–401
- temp drive 30
- third-party Platform-as-a-Service (PaaS) 392–399
  - Azure Marketplace 398–399
  - Azure Quickstart Templates 397–398
  - Cloud Foundry 392–393
  - OpenShift Container Platform 394–396
- throughput units 275
- tiered pricing 144
- Timestamp 124
- Time-to-Live (TTL) 117, 265
- tokens 203
- transaction metrics 137
- transforms 331
- Transform XML 334–336
- Transport Layer Security (TLS) 291
- triggers 164, 173, 373–374, 376–377



for Logic Apps 337–339  
T-SQL 161

## U

UDFs. *See* user-defined functions  
Universal Naming Convention (UNC) 41  
unmanaged disks 30  
update domains 58  
URIs 111, 113  
user defined functions (UDFs) 173  
user-defined metadata 113–114  
users  
    adding to DevTest Labs 97–99  
    authentication of 192, 203–206  
    Cosmos DB 175

## V

vent processing functions 371–372  
version control 404  
virtual hard disks (VHDs) 30, 72  
    custom image creation from 74–76  
virtual machine disks 30  
Virtual Machine Scale Sets (VMSS)  
    configuring Autoscale on existing 27–29  
    configuring Autoscale when provisioning 25–26  
    deployment 18–24  
virtual machines (VMs) 1–108  
    alerts, configuration 55  
    auto-shutdown policy 87–89  
    auto-start policy 90–91  
    availability 57–66  
    configuration management 7–16  
        with Azure Portal 14–15  
        with Custom Script Extension 8–10  
        with DSC 7–8, 11–12, 13–15  
        with VM Agent 7–8  
    creating  
        Linux 6  
        SQL Server 7  
        Windows Server 3–5  
DevTest Labs 67–104  
    adding VM to lab 70–71  
    cost management 92–95  
    custom images 72–78  
    environments 100–104  
    formulas 77–83  
    lab creation 67–70

    policies and procedures 83–91  
    security access 95–100  
disks  
    creating generalized 22  
extensions 7–8  
images 2  
load balancing 61–67  
monitoring 47–57  
    configuration 49–54  
    diagnostics 47–49  
    metrics 55–57  
per lab policy 86–87  
per user policy 85–86  
remote debugging 16  
scaling 16–29  
set expiration date policy 91  
sizes 17  
sizes policy 83–85  
storage 29–47  
    capacity planning 30–32  
    disk caching 39–41  
    disk encryption 46–48  
    geo-replication 41  
    pools 32–39  
    premium 45  
    shared 41–45  
    standard 45  
workload deployment 1–7  
virtual networks (VNet) 179  
Visual Studio 2017  
    API app creation with 306–310  
Visual Studio Server Explorer 300  
Visual Studio Team Services (VSTS) 403–408  
VM Agent 7, 47  
    configuration management using 7–8  
VM Depot 2

## W

WADDiagnosticInfrastructureLogsTable 49  
WADETWEventTable 49  
WADLogsTable 49  
WADPerformanceCountersTable 48  
WCF Relay 240, 247–253  
    credentials 249–250  
    deployment 248–249  
    protocols 247  
    relay and listener endpoints 250–252  
    sending messages 252–253  
Web/API applications 195–196

- web applications 281
- Web Apps 282–305
  - analytics 296
  - configuration
    - certificates 291, 294
    - settings 287–290
  - creating 284–285
  - custom domains 291–294
  - diagnostics 296–300
  - managing 295–296
  - monitoring 296, 300–303
  - resilience 303–305
  - scaling 303–305
- webhook functions 369–371
- Webhooks 94, 341
- WebJobs 366–367
- web server logs 297, 318
- web services 281
- WebSockets 239
- Windows containers 390
- Windows virtual machines
  - creating 3–5, 45
  - metrics data 48
  - remote debugging 16
- workloads
  - deployment, on ARM VMs 1–7
  - identifying supported 2–3
  - requirements 3–4
- WS-Federation 192, 198, 203–206

## X

- XML capabilities
  - Logic Apps with 331–337
- XML documents 331
- XML Validation 333–334
- XplatCLI 296

*This page intentionally left blank*

# About the authors

**ZOINER TEJADA** has more than 18 years of experience in the software industry as a software architect, CTO, and start-up CEO, with particular expertise in cloud computing, big data, analytics, and machine learning. He was among the first to receive a Microsoft Azure MVP (“Most Valuable Professional”) designation and has since been awarded the MVP for six consecutive years, and most recently a dual MVP award for Azure and Data Platform. Additionally, he was recently recognized by Microsoft as a Microsoft Regional Director.

**MICHELE LEROUX BUSTAMANTE** is cofounder / CIO of Solliance, a Microsoft Regional Director and Azure MVP, has been awarded Azure Elite and Azure Insider status as well as the ASP.NET Insider designation. Michele is a respected technology executive / thought leader, who builds high performance teams and infrastructure. With over 20 years of experience Michele has held senior executive positions, assembled software development teams and implemented processes for all aspects of the software development lifecycle, and actively facilitated large-scale enterprise application deployments. Michele is a recognized expert in many fields including distributed systems architecture, cloud computing and identity and access management – the latter, an area with very few deep technical experts. Today, Michele specializes in delivering cloud-enabled solutions at scale, cloud migration, security, compliance, and micro-services platforms.

**IKE ELLIS** is a data architect who stays current on many database technologies. He specializes in the Microsoft Data Platform, including DocumentDB, Azure SQL Datawarehouse, and Azure Data Lake. He also loves visualizing data using Microsoft tools like Power BI, SQL Server Reporting Services, and mobile dashboarding. Ike is a current Microsoft MVP for the data platform team.