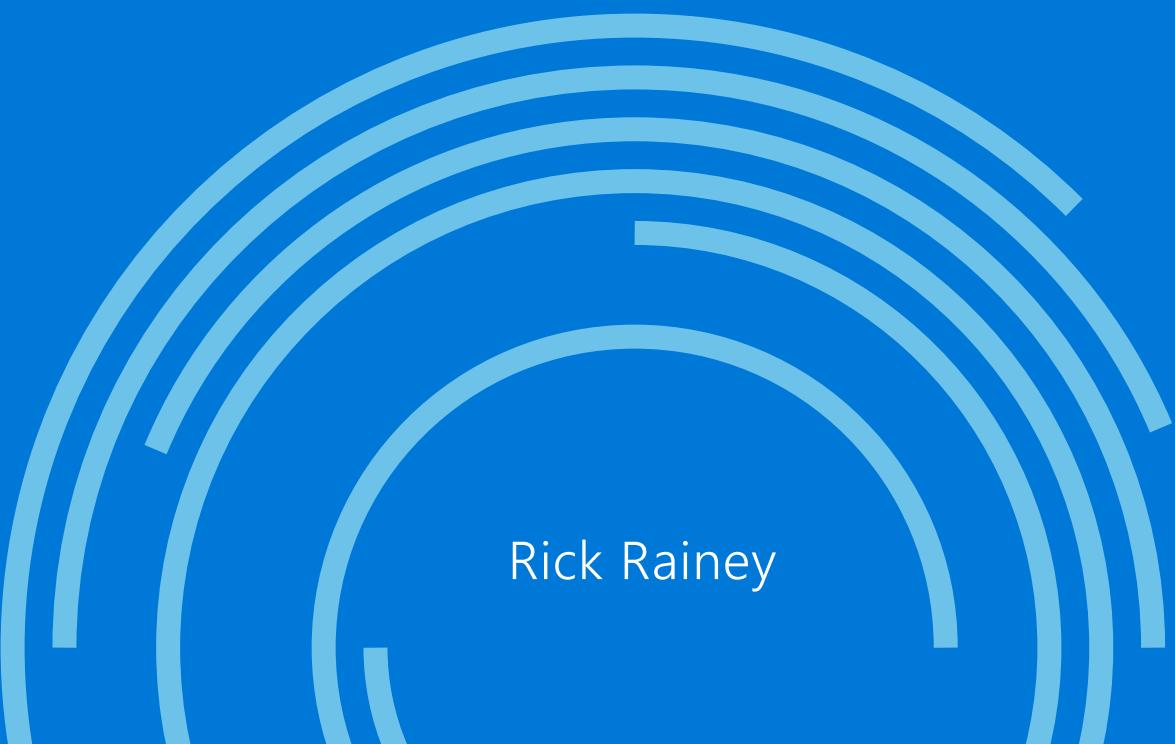


Azure Web Apps for Developers

Microsoft Azure Essentials

An abstract graphic consisting of several concentric, semi-circular arcs in varying shades of blue, creating a sense of depth and motion.

Rick Rainey

Visit us today at



microsoftpressstore.com

- **Hundreds of titles available** – Books, eBooks, and online resources from industry experts
- **Free U.S. shipping**
- **eBooks in multiple formats** – Read on your computer, tablet, mobile device, or e-reader
- **Print & eBook Best Value Packs**
- **eBook Deal of the Week** – Save up to 60% on featured titles
- **Newsletter and special offers** – Be the first to hear about new releases, specials, and more
- **Register your book** – Get additional benefits



Hear about it first.

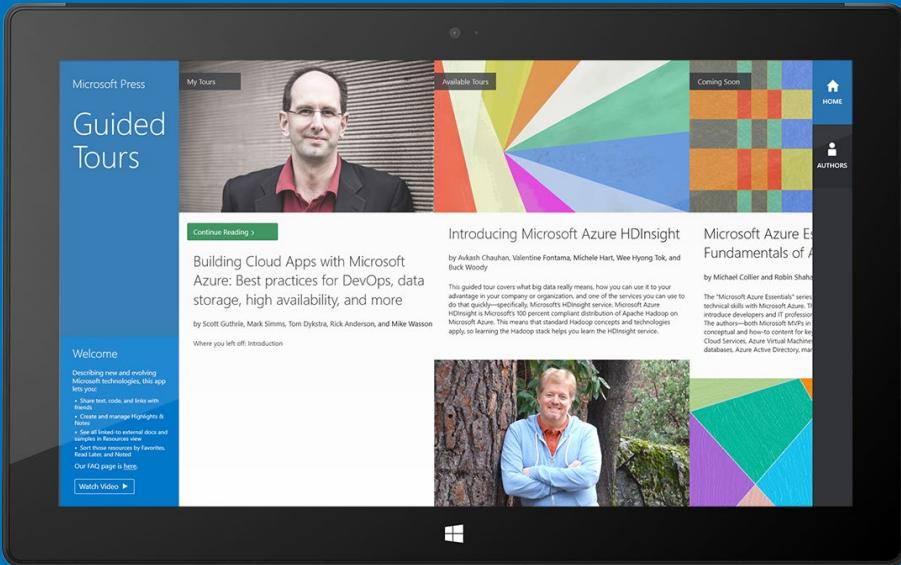


Get the latest news from Microsoft Press sent to your inbox.

- New and upcoming books
- Special offers
- Free eBooks
- How-to articles

Sign up today at MicrosoftPressStore.com/Newsletters

Wait, there's more...



Find more great content and resources in the **Microsoft Press Guided Tours** app.



The [Microsoft Press Guided Tours](#) app provides insightful tours by Microsoft Press authors of new and evolving Microsoft technologies.

- Share text, code, illustrations, videos, and links with peers and friends
- Create and manage highlights and notes
- View resources and download code samples
- Tag resources as favorites or to read later
- Watch explanatory videos
- Copy complete code listings and scripts



PUBLISHED BY
Microsoft Press
A division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Copyright © 2015 Microsoft Corporation. All rights reserved.

No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

ISBN: 978-1-5093-0059-4

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Support at mspinput@microsoft.com. Please tell us what you think of this book at <http://aka.ms/tellpress>.

This book is provided "as-is" and expresses the authors' views and opinions. The views, opinions, and information expressed in this book, including URL and other Internet website references, may change without notice.

Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in examples herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

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

Acquisitions, Developmental, and Project Editor: Devon Musgrave

Editorial Production: nSight, Inc.

Copyeditor: Ann Weaver

Cover: Twist Creative

Table of Contents

Foreword.....	6
Introduction	7
Who should read this book	7
Assumptions.....	7
This book might not be for you if.....	8
Organization of this book	8
Conventions and features in this book	9
System requirements.....	9
Acknowledgments.....	10
Errata, updates, & support	10
Free ebooks from Microsoft Press.....	10
Free training from Microsoft Virtual Academy	11
We want to hear from you.....	11
Stay in touch	11
Chapter 1 Microsoft Azure Web Apps	12
Introduction to Azure Resource Groups.....	12
Introduction to App Service Plans.....	13
Create an Azure Web App using the Azure portal.....	15
Create a Web App and SQL Database	15
Add an Azure Redis Cache to the Azure Resource Group	19
Create an Azure Web App using Visual Studio	21
Create a Web App by using Server Explorer	21
Create a Web App by using the ASP.NET Web Application template	24
Create a Web App using the Azure Resource Group template.....	26
Connection strings and application settings.....	34
Set connection strings and app settings in the environment	34
Retrieve connection strings and app settings from the environment	36
How connection strings and app settings are stored in the environment	37

Add a deployment slot for an Azure Web App.....	38
Scale to a Standard App Service Plan.....	39
Add a deployment slot.....	40
Set up continuous deployment with Visual Studio Online.....	41
Introduction to Visual Studio Online	41
Set up deployment from source control to a staging slot.....	43
Add Visual Studio solution to source control.....	45
Commit Visual Studio solution to source control.....	45
Role Based Access Control.....	46
Subscription-level roles	47
Resource-level roles.....	48
Summary.....	49
Chapter 2 Azure WebJobs	50
Introduction to Azure WebJobs.....	50
Create an Azure WebJob.....	51
Publish a web job from Visual Studio.....	52
Invoke a web job manually.....	54
View the WebJobs Dashboard.....	54
Create a web job from the Azure portal	55
Introduction to the Azure WebJobs SDK.....	57
WebJobs SDK .NET libraries and dependencies	58
Create a web job designed for use with Azure Storage Queues.....	58
Examine the web job project and code	60
Publish a web job to Azure	64
Examine new features in the WebJobs Dashboard.....	65
Summary	68
Chapter 3 Scaling Azure Web Apps.....	69
Scale Up.....	69
Scale Out.....	71
Dealing with the challenges of scaling out a web app.....	71

Scaling web apps using Autoscale	73
Autoscale based on CPU percentage.....	73
Autoscale based on a recurring schedule	74
Understanding Autoscale rules.....	77
Turn off Autoscale.....	79
Scale globally with Azure Traffic Manager	79
Create a Traffic Manager profile	81
Additional services for achieving massive scale.....	83
Scaling WebJobs.....	83
Summary.....	84
Chapter 4 Monitoring and diagnostics.....	86
Introduction to diagnostic logs.....	86
Enable application and site diagnostic logs	88
Store log files in the web app file system.....	88
Store log files in Azure Storage	89
Access and download diagnostic log files	91
Access log files stored in the web app file system.....	91
Access log files from Azure Storage	95
Log streaming.....	96
Log streaming using Visual Studio.....	96
Log streaming using command-line tools.....	97
Remote debugging	98
Diagnostics as a Service (DaaS)	100
Install the Diagnostics as a Service site extension.....	100
Run DaaS.....	101
View DaaS analysis reports	102
Site Admin Tools/Kudu.....	105
Install the Site Admin Tools/Kudu	105
Run the Site Admin Tools	105
Monitor web app endpoints externally using web tests.....	109

Create a URL ping web test	110
Monitoring	112
Monitor a resource group using the Azure portal	112
Application Insights	114
Add Application Insights to an existing ASP.NET MVC Web Application	115
Summary	119

Foreword

I'm thrilled to be able to share these Microsoft Azure Essentials ebooks with you. The power that Microsoft Azure gives you is thrilling but not unheard of from Microsoft. Many don't realize that Microsoft has been building and managing datacenters for over 25 years. Today, the company's cloud datacenters provide the core infrastructure and foundational technologies for its 200-plus online services, including Bing, MSN, Office 365, Xbox Live, Skype, OneDrive, and, of course, Microsoft Azure. The infrastructure is comprised of many hundreds of thousands of servers, content distribution networks, edge computing nodes, and fiber optic networks. Azure is built and managed by a team of experts working 24x7x365 to support services for millions of customers' businesses and living and working all over the globe.

Today, Azure is available in 141 countries, including China, and supports 10 languages and 19 currencies, all backed by Microsoft's \$15 billion investment in global datacenter infrastructure. Azure is continuously investing in the latest infrastructure technologies, with a focus on high reliability, operational excellence, cost-effectiveness, environmental sustainability, and a trustworthy online experience for customers and partners worldwide.

Microsoft Azure brings so many services to your fingertips in a reliable, secure, and environmentally sustainable way. You can do immense things with Azure, such as create a single VM with 32TB of storage driving more than 50,000 IOPS or utilize hundreds of thousands of CPU cores to solve your most difficult computational problems.

Perhaps you need to turn workloads on and off, or perhaps your company is growing fast! Some companies have workloads with unpredictable bursting, while others know when they are about to receive an influx of traffic. You pay only for what you use, and Azure is designed to work with common cloud computing patterns.

From Windows to Linux, SQL to NoSQL, Traffic Management to Virtual Networks, Cloud Services to Web Sites and beyond, we have so much to share with you in the coming months and years.

I hope you enjoy this Microsoft Azure Essentials series from Microsoft Press. The other ebooks in the series cover fundamentals of Azure, Azure Automation, and Azure Machine Learning. (Take a look at Microsoft Press's blog to find these.) And I hope you enjoy living and working with Microsoft Azure as much as we do.

Scott Guthrie

Executive Vice President

Cloud and Enterprise group, Microsoft Corporation

Introduction

Azure Web Apps is a fully managed platform that you can use to build mission-critical web applications that are highly available, secure, and scalable to global proportions. Combined with first-class tooling from Visual Studio and the Microsoft Azure Tools, the Azure Web Apps service is the fastest way to get your web application to production. Azure Web Apps is part of the Azure App Service that is designed to empower developers to build web and mobile applications for any device.

Developing web applications to host on Azure Web Apps is a familiar experience for developers accustomed to hosting web applications on Internet Information Services (IIS). Developers can use ASP.NET, Java, Node.js, PHP, and Python for their application development locally and easily deploy to Azure Web Apps. The environment supports continuous deployment to multiple staging environments, enabling development teams to deploy application updates rapidly and reliably.

Azure Web Apps is more than a host for web front-end applications. It also supports development of robust background processes using the Azure WebJobs feature. WebJobs can be invoked on demand, scheduled, or automatically invoked using a feature-rich WebJobs SDK.

The monitoring and diagnostics built into Azure Web Apps are exceptional. The Azure portal delivers a professional UI experience that you can use to interact with your monitoring and diagnostics data. Site extensions are available to further enhance this experience, and services such as Application Insights can be used to gain deeper insight into your application code running in Azure.

This ebook will guide you through these topics, point you to some best practices along the way, and provide detailed walkthroughs for you to gain hands-on experience.

Who should read this book

This book focuses on providing essential information about developing web applications hosted on Azure Web Apps. It is written with the developer who has experience using Visual Studio and the .NET Framework in mind. If Azure Web Apps is new to you, then this book is for you. If you have experience developing for Azure Web Apps, then this book is for you, too, because there are features and tools discussed in this text that are new to the platform.

Assumptions

It is expected that you have at least a minimal understanding of cloud computing concepts and basic web services. Some familiarity with developing web applications using Visual Studio and C# is not required but may help fast-track your learning. You should have general knowledge of how to use the Azure Preview portal at <http://portal.azure.com>.

This book might not be for you if...

This book might not be for you if you are looking for guidance developing ASP.NET MVC or Web API applications. Instead, this book focuses on the features and services of the Azure platform used to develop web-based cloud applications. Although ASP.NET MVC may be used to demonstrate concepts, it is only to the extent necessary to support the in-depth discussion on how to use Azure Web Apps to host your web application.

Organization of this book

This book provides information you can use to start building web applications using Azure Web Apps. It will guide you through development, deployment, and configuration tasks that are common for today's developer building cloud applications.

Each chapter stands alone; there is no requirement that you perform the hands-on demonstrations from previous chapters to understand any chapter. The topics explored in this book include the following:

- **Chapter 1, "Microsoft Azure Web Apps":** This chapter starts with an introduction to Azure Resource Groups and App Service Plans and progresses into essential tasks such as creating and configuring a web app. Learn best practices for storing and retrieving app settings and connection strings. Configure deployment slots and set up continuous deployment using Visual Studio Online. Wrap up with a discussion about Role Based Access Control (RBAC) and how you can use it to manage access to your Azure resources.
- **Chapter 2, "Azure WebJobs":** Learn everything you need to know to build and deploy background processing tasks using Azure WebJobs. You will learn the basics of the WebJobs feature and proceed into a deeper discussion on how to use the WebJobs SDK. You will learn about the Azure WebJobs Dashboard and how the WebJobs SDK enhances the dashboard experience.
- **Chapter 3, "Scaling Azure Web Apps":** Learn how to scale up and scale out your Azure web app and web jobs. You will learn how to configure Autoscale to scale your web app dynamically based on performance metrics and schedules. See how you can use Azure Traffic Manager to achieve global scale for your web apps.
- **Chapter 4, "Monitoring and diagnostics":** Learn about the many logging features built into the Azure Web Apps platform and how to configure logging to get the diagnostics data you need to troubleshoot issues. You will learn how to configure storage locations and retention policies for logs, how to view logs in real time using the log streaming service, and even how to debug your web app remotely while it is running in Azure. You will get an introduction to some

powerful site extensions you can use to view logs and perform analysis directly from your browser. Finally, you will learn how you can monitor your resource group down to individual resources and how you can use Application Insights to deliver a complete 360-degree view into your application code for monitoring and diagnostic purposes.

Conventions and features in this book

This book presents information using conventions designed to make the information readable and easy to follow:

- There currently are two management portals for Azure: the Azure Management Portal at <http://manage.windowsazure.com> and the new Azure Preview Portal at <http://portal.azure.com>. This book assumes the use of the new Azure Portal unless noted otherwise.
- A plus sign (+) between two key names means that you must press those keys at the same time. For example, "Press Alt+Tab" means that you hold down the Alt key while you press Tab.

System requirements

For many of the examples in this book, you will need a browser (Internet Explorer 10 or higher) to access the Azure portal, Visual Studio 2013 with Update 4, and the Microsoft Azure Tools v2.6. You can download a free copy of Visual Studio Express at the link below. Be sure to scroll down the page to the link for "Express 2013 for Windows Desktop": <http://www.visualstudio.com/en-us/products/visual-studio-express-vs.aspx>.

The system requirements are as follows:

- Windows 7 Service Pack 1, Windows 8, Windows 8.1, Windows Server 2008 R2 SP1, Windows Server 2012, or Windows Server 2012 R2
- Computer that has a 1.6 GHz or faster processor (2 GHz recommended)
- 1 GB (32 bit) or 2 GB (64 bit) RAM (Add 512 MB if running in a virtual machine)
- 20 GB of available hard disk space
- 5,400 RPM hard disk drive
- DirectX 9 capable video card running at 1,024 x 768 or higher-resolution display
- DVD-ROM drive (if installing Visual Studio from DVD)
- Internet connection

Depending on your Windows configuration, you might require Local Administrator rights to install or configure Visual Studio 2013.

Acknowledgments

It is with great pleasure to personally thank Rick Anderson and Tom Dykstra for their *expert* technical reviews, feedback, guidance, and encouragement. If you have used the online documentation at <http://azure.com> or <http://asp.net>, then you probably already know the fantastic work these two gentlemen produce. It was an honor to have them on the team for this project.

Special thanks to the entire team at Microsoft Press for their awesome support and guidance on this journey. Most of all, it was a pleasure to work with my editor, Devon Musgrave, who provided guidance from the very beginning when this was just an idea to the final copy you are about to read.

Errata, updates, & support

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

<http://aka.ms/AzureWebApps/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 mspinput@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 <http://support.microsoft.com>.

Free ebooks from Microsoft Press

From technical overviews to in-depth information on special topics, the free ebooks from Microsoft Press cover a wide range of topics. These ebooks are available in PDF, EPUB, and Mobi for Kindle formats, ready for you to download at:

<http://aka.ms/mspressfree>

Check back often to see what is new!

Free training from Microsoft Virtual Academy

The Microsoft Azure training courses from Microsoft Virtual Academy cover key technical topics to help developers gain the knowledge they need to be a success. Learn Microsoft Azure from the true experts. Microsoft Azure training includes courses focused on learning Azure Virtual Machines and virtual networks. In addition, gain insight into platform as a service (PaaS) implementation for IT Pros, including using PowerShell for automation and management, using Active Directory, migrating from on-premises to cloud infrastructure, and important licensing information.

<http://www.microsoftvirtualacademy.com/product-training/microsoft-azure>

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:

<http://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>

Chapter 1

Microsoft Azure Web Apps

Azure App Service Web Apps (formerly Azure Websites) is a platform-as-a-service (PaaS) offering that enables developers to build secure, mission-critical, and highly scalable web applications. Developers can choose from languages such as C#, HTML5, PHP, Java, Node.js, and Python to write their code and use familiar tools such as Visual Studio and platform-specific Azure SDKs to get started quickly.

This chapter will introduce you to essential knowledge to get started building Azure web apps. This text will present topics with the ASP.NET developer in mind and, therefore, will use C# for code samples and Visual Studio 2013 with the Azure SDK for .NET installed.

This book focuses exclusively on Azure Web Apps, not on essential aspects of the Microsoft Azure platform or the Azure portal. Therefore, it is recommended that the reader have some knowledge of the Microsoft Azure platform. An excellent resource to gain this knowledge is *Microsoft Azure Essentials: Fundamentals of Azure*, which you can download for free from http://blogs.msdn.com/b/microsoft_press/archive/2015/02/03/free-ebook-microsoft-azure-essentials-fundamentals-of-azure.aspx.

Introduction to Azure Resource Groups

When you create any resource in Azure, you will associate that resource with a new or existing resource group. Therefore, before creating a resource such as an Azure web app, it is important to understand Azure Resource Groups.

An Azure Resource Group is a logical container for grouping Azure resources. As an example, consider a typical website implementation that has a web front end with which users interact and a database in which to store data. The web front end and the database are individual resources that comprise the full website solution. An Azure Resource Group gives you a natural way to manage and monitor resources that comprise a solution. Figure 1-1 is an example of what an Azure Resource Group could look like for a web application consisting of an Azure web app, Redis Cache, SQL database, DocumentDB, and an Azure Storage account.

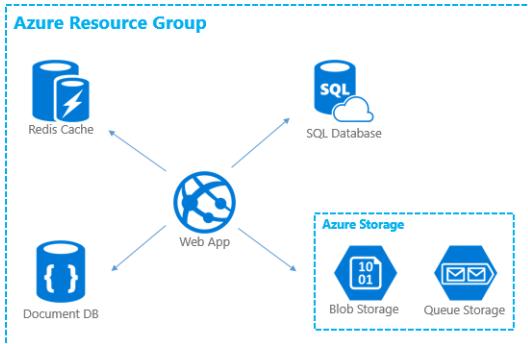


FIGURE 1-1 A hypothetical representation of an Azure Resource Group.

Grouping resources this way helps simplify the implementation, deployment, management, and monitoring of resources in the resource group. From a billing perspective, it gives you a way to view costs for the resource group rather than for individual resources, eliminating the need to figure out which resources are related. You can think of an Azure Resource Group as a unit of management.

The resources in a resource group can span regions if needed. For example, you may have a web app that is deployed in two or more regions for high availability in the unlikely event that an entire datacenter were to go down where your app is deployed. In this scenario, each web app could be part of the same resource group.

A single resource may exist in only one resource group. But it is possible for resources from different resource groups to be linked. These are referred to as *linked* resources in the Azure portal. An example of this could be a SQL database that is shared between web apps in different resource groups. Assume you have a SQL database named "sqldb1" that is a resource in a resource group named "resgrp1." Now, assume you have a second resource group named "resgrp2" with a web app resource that will use "sqldb1." In this example, "sqldb1" will appear as a linked resource in "resgrp2."

Introduction to App Service Plans

An App Service Plan provides a way for you to define the region and pricing tier (capacity and features) that can be shared across the app service types, which are Web Apps, Mobile Apps, Logic Apps, and API Apps. There are five pricing tiers available for Azure App Services:

- **Free** The Free tier is intended for evaluation purposes; this is why it is free. Web apps in this tier share machine resources with other web apps from other Azure subscribers. Because the resources are shared, you have limited daily compute and network bandwidth limits.
- **Shared** The Shared tier also shares machine resources with other web apps, but the daily compute and network bandwidth limits are increased. This tier is intended for low-traffic sites and enables support for a few features, such as custom domains and the ability to scale out to

multiple shared instances.

- **Basic** The Basic tier gives you dedicated machine instances to host your web app and, therefore, has no daily resource limits. In this tier, you are able to scale up the size of your dedicated virtual machine to increase the number of cores and RAM. This tier enables additional Azure Web App features and also is backed by a service level agreement (SLA).
- **Standard** The Standard tier has all the features of the Basic tier, and it enables all features available to Azure Web Apps. Features such as autoscale, deployment slots, automated backups, support for Microsoft Azure Traffic Manager, and more are enabled in this tier.
- **Premium** The Premium tier (in preview) provides the same set of features as the Standard tier but with some extra capacity for features such as the number of deployment slots, storage, and backups. It also enables BizTalk integration capabilities.

For details on the capacity, features, and costs associated with each pricing tier, see

<http://azure.microsoft.com/en-us/pricing/details/app-service/>.

An Azure web app may exist in a single App Service Plan at any given time. Or when needed, a web app may be moved to another App Service Plan. This gives you the flexibility of starting your web app development in an App Service Plan configured for the Free tier and then upgrading to an App Service Plan configured for a higher pricing tier as your application and development requirements expand.

An App Service Plan can be used for multiple Web Apps, Mobile Apps, Logic Apps, and API Apps. For example, you may have a plan that you use for development and testing your web apps that is configured with less capacity and fewer features. Or you may have a plan that you use for a common set of customer web apps in a specific region.

At any time, you can change the pricing tier for your App Service Plan. So it is not necessary to create a new one if you need to move to a higher or lower pricing tier.

Important Changing the pricing tier for an App Service Plan will change the tier for all web apps in the plan. For example, if you have five web apps running in a plan configured for the Basic tier and later change the pricing tier to Standard, then all five web apps will be upgraded to run on Standard Virtual Machines instead of Basic Virtual Machines.

An App Service Plan is a component of Azure Resource Groups, discussed in the previous section. At first, this may not seem intuitive, but if you think of an Azure web app as a resource in a resource group and an App Service Plan as a way to define the features and capacity available to your web app, then the relationship between the two is easier to comprehend. For more information on the relationship between Azure Resource Groups and App Service Plans, see
<http://azure.microsoft.com/en-us/documentation/articles/azure-web-sites-web-hosting-plans-in-depth-overview/>.

Create an Azure Web App using the Azure portal

The Azure portal at <https://portal.azure.com> provides a rich user interface to provision and manage resources in your Azure subscription. In this section, you will create an Azure web app with a SQL database and then add an Azure Redis Cache to the resource group. After completing the steps, you will have an Azure web app *environment* to which you can publish a web application. Publishing a web application such as an ASP.NET MVC application is covered later in this chapter.

Create a Web App and SQL Database

After signing in to the Azure portal, click the **+NEW** button in the lower-left corner of the page. In the Create blade, select the **Web + Mobile** option. In the Web + Mobile blade, you will see options for creating various resources in Azure App Service, such as Web Apps, Mobile Apps, Logic Apps, and API Apps.

Note If your goal is to create just an Azure web app, then you could select the option for Web Apps where you would have the opportunity to specify an Azure Resource Group and App Service Plan for the web app.

At the bottom of the Web + Mobile blade, click **Azure Marketplace**. In the Azure Marketplace, additional options are available for creating common Azure Web App environments, such as a web app with a SQL database or a web app with a MySQL database, as shown in Figure 1-2.



FIGURE 1-2 Web App solutions in the Azure Marketplace.

Tip There are many options for Web Apps you can choose from in the Azure Marketplace that provide solutions for popular configurations such as blogging sites, frameworks, ASP.NET Starter apps, and more. The full selection of solutions can be found in the Web Applications section of the Marketplace at <http://azure.microsoft.com/en-us/marketplace/web-applications/>. To create a web app using one of the solutions, click the solution. You will be redirected back to the Azure portal where you can configure the solution settings for your needs.

Select the Web App + SQL option, which will open a blade describing the resources that this solution will create and also provides links to relevant resources. Next, click the **Create** button at the bottom of the blade to begin configuring the web app and SQL database.

In the Web App + SQL blade, the first thing you must do is specify the Azure Resource Group name to create a new resource group. After entering a resource group name, select the **Web App (Configure Required Settings)** option to open the Web App blade.

Configure required Web App settings

In the Web App blade, you must specify a globally unique URL for the web app. All web apps are created in the *.azurewebsites.net domain and, therefore, must be unique to this domain. Later, you can configure a custom domain you own to map to this URL. For information on configuring custom domains, see <http://azure.microsoft.com/en-us/documentation/articles/web-sites-custom-domain-name/>.

Next, you need to specify an App Service Plan. You can choose an existing plan or create a new one. To create a new App Service Plan, enter a name in the text box. Creating a new App Service Plan will unlock the Pricing Tier and Location options in the Web App blade so you can configure those settings.

Click the Pricing Tier option and change it to D1 Shared. If you don't see the D1 Shared option, then click the **View All** link in the upper-right corner to show all the pricing tiers.

Click the Location option and select a region close to you.

Tip You can use <http://azurespeedtest.azurewebsites.net/> to find the lowest latency region for your client. The Web App + SQL and Web App blades will look similar to Figure 1-3.

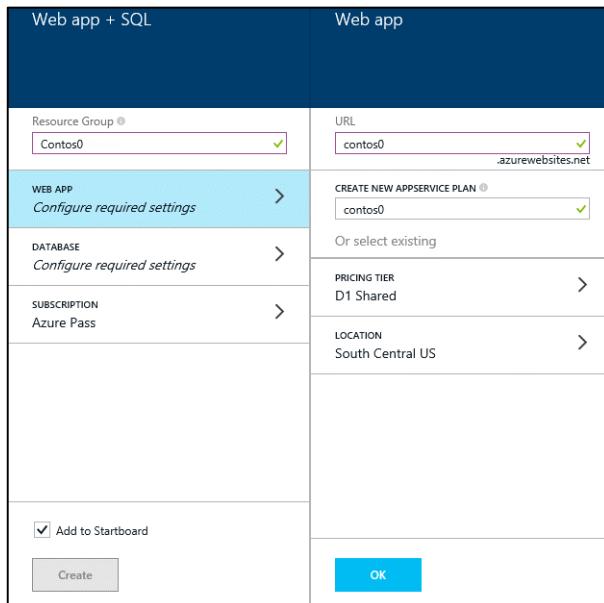


FIGURE 1-3 Web App + SQL blade and Web App blade.

Apply the required Web App settings by clicking the **OK** button at the bottom of the Web App blade. This will close the Web App blade and take you back to the Web App + SQL blade.

Configure required SQL Database settings

In the Web App + SQL blade, select the **Database (Configure Required Settings)** option to open the Database blade. If you have an existing SQL database, then you will have the option to select an existing database or create a new database. If you are presented with this option, select the option to create a new database, which will open the New Database blade.

Note Selecting an existing SQL database would result in the SQL database appearing as a "Linked" resource in the Summary section of the Azure Resource Group blade.

In the New Database blade, specify the name of the new SQL database you will create. Optionally, you may choose to change the Pricing Tier for your Database (not the web app) and Collation settings.

Next, select the **Server (Configure Required Settings)** option to open the New Server blade. In the New Server blade, specify a name for the server in which your database will be created and provide server admin credentials you later can use to sign in and manage the server.

The New Database blade and New Server blade will look similar to Figure 1-4.

The image shows two side-by-side configuration blades. The left blade is titled 'New database' and the right blade is titled 'New server'. Both blades have a dark header bar and a light blue footer bar containing an 'OK' button. The 'New database' blade has fields for 'Name' (contoso0-db), 'PRICING TIER' (Standard S0), and 'COLLATION' (SQL_Latin1_General_CI_AS). The 'New server' blade has fields for 'SERVER NAME' (contoso0-server.database.windows.net), 'SERVER ADMIN LOGIN' (adminuser), 'PASSWORD' (redacted), 'CONFIRM PASSWORD' (redacted), 'LOCATION' (South Central US), and checkboxes for 'CREATE V12 SERVER (LATEST UPDATE)' (Yes selected) and 'ALLOW AZURE SERVICES TO ACCESS SERVER' (checked). Both blades show green checkmarks in their respective input fields.

FIGURE 1-4 New Database blade and New Server blade.

Apply the required server settings by clicking the **OK** button at the bottom of the New Server blade. This will close the New Server blade and take you back to the New Database blade.

Apply the new database settings by clicking the **OK** button at the bottom of the New Database blade. This will close the New Database blade and take you back to the Web App + SQL blade.

The Create button at the bottom of the Web App + SQL blade will be enabled after all the required settings described above have been applied. Click the Create button to create the Web App and SQL database.

After a moment, the Azure Web App and SQL database (and server) will be provisioned and the Azure portal will open the Resource Group blade the resources were created in. In the Summary part of the blade, you can see all the resources that were created in the resource group, which will look similar to Figure 1-5.

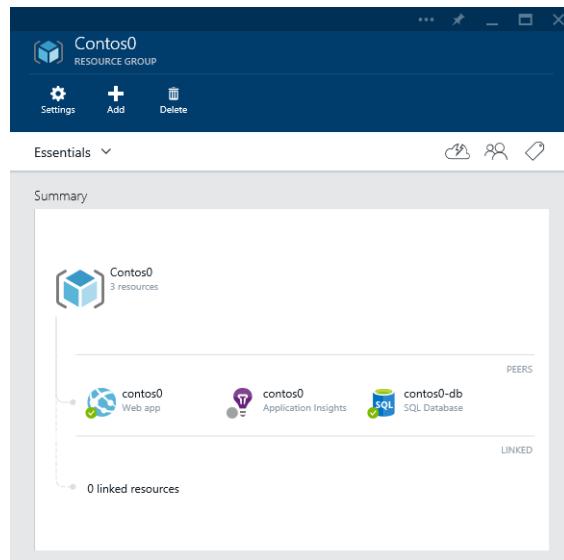


FIGURE 1-5 Summary part of the Resource Group blade.

Notice that an Application Insights resource was added by default in addition to the Azure Web App and SQL database resources. Application Insights is used to surface critical metrics from your web app that can be used for monitoring and troubleshooting. Application Insights will be discussed in Chapter 4, "Monitoring and diagnostics."

If you scroll through the blade, you will see web parts for monitoring, billing, and configuring alerts for the resource group. As you can see, the Resource Group blade provides the unit of management that enables you to drill down into the individual resources and apply resource-specific configuration settings.

Add an Azure Redis Cache to the Azure Resource Group

Many web apps will leverage caching to improve performance. For applications that need caching, Azure provides a fully managed Redis Cache as a service. The Azure Redis Cache is based on the popular open source Redis Cache that is a distributed, in-memory cache service. It provides a level of performance and features unmatched by previous cache offerings in a way that is easy to use. You can learn more about this service at <http://azure.microsoft.com/en-us/services/cache/>.

You can add an Azure Redis Cache in the Resource Group blade by clicking the **Add** button at the top of the blade, as shown in Figure 1-6.



FIGURE 1-6 Azure Resource Group toolbar.

In the New Resource blade, select the **Redis Cache** option from the list of resources, which will look similar to Figure 1-7. Selecting this option will open a Redis Cache blade describing the services, pricing tiers, uses for the cache, and links to relevant resources. Click the **Create** button at the bottom of the Redis Cache blade to begin configuring the resource.



FIGURE 1-7 Azure Redis Cache resource option.

In the New Redis Cache blade, enter a globally unique DNS name for your cache, choose a pricing tier, and choose a location. The Resource Group and Subscription will appear locked in the blade because the Redis Cache is being added to the current resource group. The New Redis Cache blade will look similar to Figure 1-8.

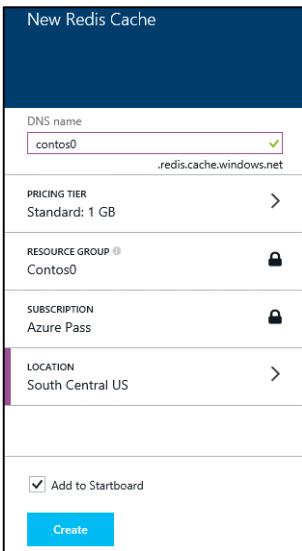


FIGURE 1-8 The New Redis Cache blade.

After adding the Redis Cache, you may not see the new resource in the Summary part of the Resource Group blade as you did with other resources shown in Figure 1-5. However, if you look closely at the Resource Group icon in the Summary part, you will see the total number of resources in the resource group under the name. Clicking the Resource Group icon will open the Resources blade, where you can see all the resources in the resource group, including the Redis Cache, as shown in Figure 1-9.

The screenshot shows the 'Resources' blade for the 'Contos0' resource group. It has a search bar at the top. Below it is a table with columns: NAME, RESOURCE GROUP, DESCRIPTION, and STATUS. The table lists four resources: 'contos0' (Redis Cache), 'contos0' (Application Insights), 'contos0-db' (SQL Database), and 'contos0' (Web app). All resources have a green checkmark in the STATUS column.

NAME	RESOURCE GROUP	DESCRIPTION	STATUS
contos0	Contos0	A redis cache	✓
contos0	Contos0	Application Insights	●
contos0-db	Contos0	SQL Database	✓
contos0	Contos0	Web app	✓

FIGURE 1-9 Resources blade.

Create an Azure Web App using Visual Studio

There are several ways to create a web app by using Visual Studio. For simple Dev/Test scenarios where you just want to quickly create an Azure web app environment and provision some resources, the Server Explorer window in Visual Studio can be very handy. It provides features for provisioning and managing resources in your Azure subscription from within your native development environment. It doesn't offer the same level of management capabilities the Azure portal does, but for the resources for which it does provide management capabilities, it is a significant time saver. Using Server Explorer to create a web app produces a web app environment to which you can publish a web application later.

Another approach for creating web apps is to use the ASP.NET Web Application template to create a new web application project. This is the experience ASP.NET developers have become very familiar with in recent years for starting web application development. With the Azure SDK Tools installed, this template also enables you to target Web Apps or Virtual Machines when hosting in Azure. Taking this approach to create a web app produces a web app environment and a web application project with the configuration needed to publish the web application to the host environment.

Perhaps the most feature-rich approach to creating a web app is to use the Cloud Deployment Project template to create your new web application project. This template further exposes the notion of Azure Resource Groups when creating a new project by producing both a web application project and a deployment project in the solution. The deployment project leverages the Azure Resource Manager to create the web app environment for your web application and includes Windows PowerShell scripts and JSON files describing the environment that you can use to automate the deployment to Azure.

Note You can download the Azure SDK Tools from <http://azure.microsoft.com/en-us/downloads/>. Under the SDK section, you can download language-specific SDKs and tools for your development environment. The steps demonstrated in this section are based on version 2.5.1 of Azure SDK Tools.

This section will discuss each of these techniques for creating an Azure web app using Visual Studio.

Create a Web App by using Server Explorer

The Server Explorer window in Visual Studio brings certain management capabilities directly into your Visual Studio environment. This is useful particularly in Dev/Test environments where you just want to provision some resources and quickly start developing or testing ideas without leaving your development environment.

Note If Server Explorer is not visible, you can open it from the main menu by selecting **View > Server Explorer**.

To use the Server Explorer with your Azure Subscription, you must first connect it to your Azure Subscription. Do this by right-clicking the Azure icon and selecting the option to **Connect To Microsoft Azure Subscription**, as shown in Figure 1-10.

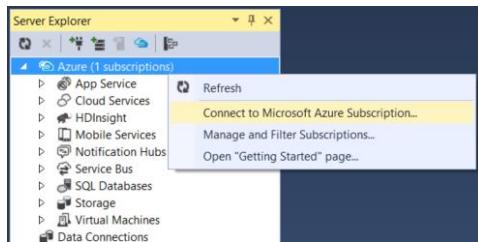


FIGURE 1-10 Connect Server Explorer to your Microsoft Azure Subscription.

When prompted to sign in to your Azure Subscription, use the same credentials you use to sign in to the Azure portal. After successfully authenticating, the Azure node in Server Explorer will display resources from each of your Azure Subscriptions.

If you have multiple subscriptions associated with your account, then you may want to filter the resources shown in the Azure node to a single subscription with which you are working. You can do this by right-clicking the Azure node in Server Explorer and selecting the option to **Manage Subscriptions**. This opens a dialogue where you can select the subscriptions and regions for which you want to filter Server Explorer's user interface, as shown in Figure 1-11.

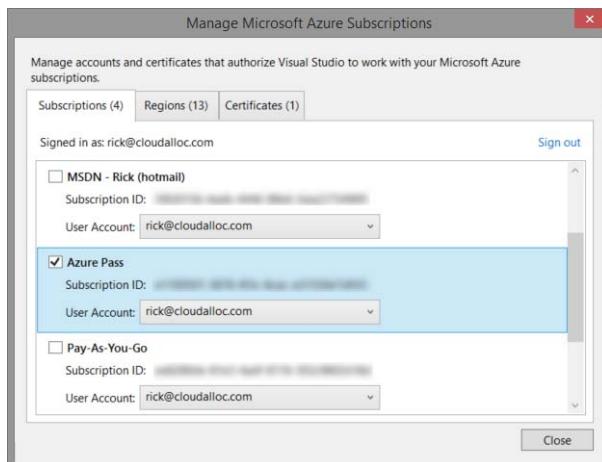


FIGURE 1-11 Dialog in Visual Studio for managing Azure Subscription settings.

To create a new web app and, optionally, a SQL database using Server Explorer, right-click the App Service node and select the option **Create New Site**. This will open a single page dialog where you can specify settings for the web app, as shown in Figure 1-12.

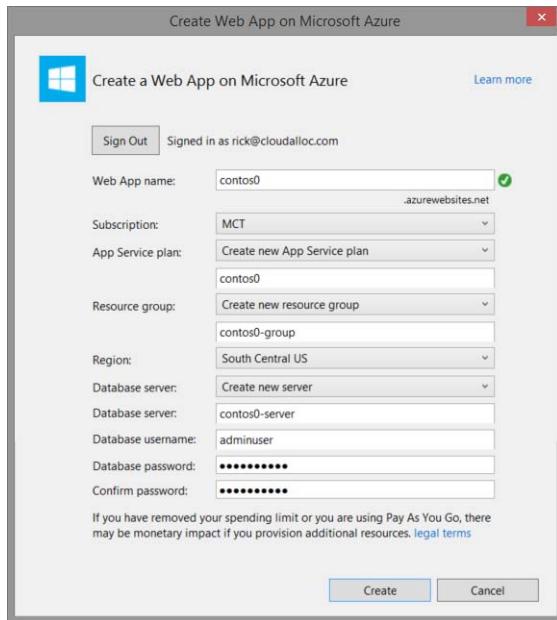


FIGURE 1-12 Create an Azure Web App from Server Explorer in Visual Studio.

Because there are few options for configuring the web app and SQL database, you should expect some limitations when taking this approach. Some of these limitations are as follows:

- For the App Service Plan, you have the option to select an existing plan or to specify the name to create a new plan. If you choose to create a new plan, the pricing tier will default to F1 Free. You later can change this to a pricing tier appropriate for your needs.
- Although you have the option to specify the Database Server name or select an existing server, the name of the SQL database will be set for you and defaults to <web app name>_db, where <web app name> is the web app name specified at the top of the dialog. This cannot be changed.
- You cannot specify the tier for your SQL database if you choose to create one. You later can change this to an appropriate tier.
- It is assumed that if you want a database, you want an Azure SQL database. In other words, there is not an option to select a MySQL database. You would need to use the Azure portal if you want a MySQL database.
- There is no option to add an Azure Redis Cache. But as demonstrated earlier, you can add an Azure Redis Cache later by using the Azure portal.

The main reason for showing this technique is to introduce you to Server Explorer, connect it to your Azure subscription, and then demonstrate how to create an Azure web app environment. Server Explorer will be referenced throughout the text, so having it connected to your Azure subscription will facilitate those topics later.

Create a Web App by using the ASP.NET Web Application template

Web application development traditionally starts with the developer going through the New ASP.NET Web Application Project Wizard in Visual Studio and selecting from a number of available templates. This experience is enhanced when the Azure SDK Tools for Visual Studio are installed, which enable you to configure and provision your Azure web app resources as you progress through the wizard.

To begin, from the Visual Studio main menu select **File > New > Project**.

In the New Project dialog, expand the Templates node on the left and click the Cloud node to view the available cloud templates. From the list of templates, select the **ASP.NET Web Application** template. The New Project dialog will look similar to Figure 1-13.

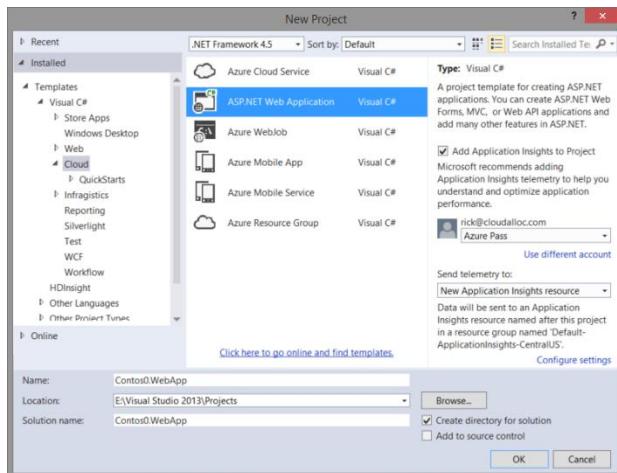


FIGURE 1-13 Create a new ASP.NET Web Application project.

If you select the option for Application Insights, then code and configuration are added to the project to capture telemetry data. This will be covered in more detail in the Application Insights section of Chapter 4.

Application Insights can be added to a web app later, so unchecking it does not prevent you from taking advantage of this feature later.

On the next page in the New Project Wizard, you can select from the many ASP.NET project

templates, such as the MVC template. With the Azure SDK Tools installed, the wizard allows you to change the authentication and Azure hosting options, as shown in Figure 1-14.

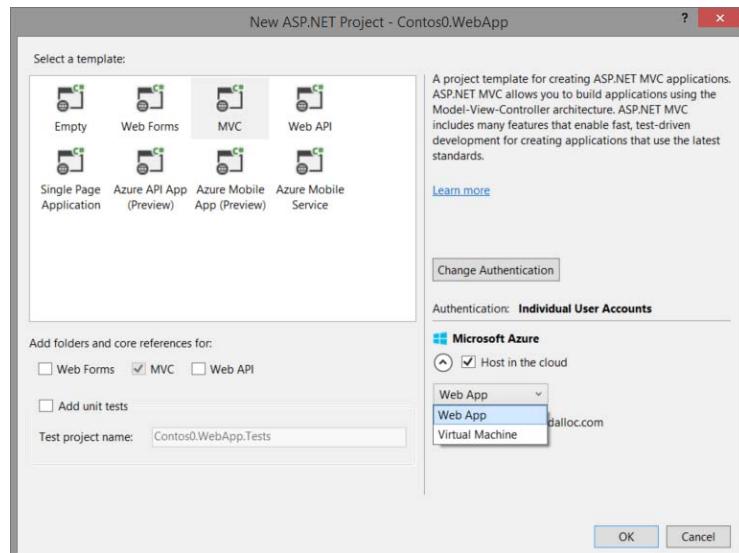


FIGURE 1-14 New ASP.NET Project templates for Visual Studio.

The authentication option defaults to Individual User Accounts, but you can change this by clicking the **Change Authentication** button. The authentication options are as follows:

- **No Authentication** For applications that do not require users to sign in.
- **Individual User Accounts** For applications that will store user profiles in a SQL database. This option also adds code to the project to support users registering and signing in using social networking accounts such as Facebook, Twitter, Google, and Microsoft. This option uses Microsoft's Open Web Interface for .NET (OWIN) implementation, OAuth 2.0, and cookie-based authentication to authenticate users. You can learn more about OWIN at <http://owin.org/>.
- **Work And School Accounts** For applications that will authenticate users against Active Directory, Azure Active Directory, or Office 365. Choosing this option enables you to specify the domain name for the organization and directory access permissions. This option uses an OWIN implementation of OpenID Connect and JSON Web Token (JWT) tokens when authenticating users.
- **Windows Authentication** For applications running in a traditional intranet environment. This option uses an OWIN implementation supporting Kerberos and NTLM protocols when authenticating users.

Selecting the option to Host In The Cloud using Microsoft Azure enables you to host your application using a Microsoft Azure Web App or Microsoft Azure Virtual Machine. The latter is similar

to what you may be used to in an on-premises environment where you have a cluster of servers (or virtual machines) running IIS that you want to run your application on. This is different from the Azure Web App option, which does not require you to manage the virtual machine. In other words, the Azure Web App option is a platform-as-a-Service (PaaS) approach, and the Virtual Machine option is an infrastructure-as-a-service (IaaS) approach to hosting your web application.

The last page in the New Project Wizard is the same as referenced in Figure 1-12. On this page, you can specify settings for the URL, Subscription, App Service Plan name, Resource Group name, Region, and SQL database credentials.

After completing the New Project Wizard, you will have a fully functioning ASP.NET MVC application that you can run locally. If you selected the option to Host In The Cloud with Azure Web Apps, then your web app environment also has been created in Azure and your solution has the information it needs to publish the application to that web app environment.

Publish from Visual Studio

When you are ready to publish the application, right-click the ASP.NET project in Solution Explorer and select the **Publish** option. Alternatively, you can select **Build > Publish <your project name>** from the main menu.

Create a Web App using the Azure Resource Group template

The Azure Resource Group template is relatively new. It was introduced in version 2.6 of the Azure SDK tools. This template contains JSON files (templates) used by the Azure Resource Manager to describe your resources and the resource group in which they are contained. It also includes a Windows PowerShell script you can use to automate creating the resources, making it easy to redeploy to multiple environments. And if you want the project to deploy artifacts such as an ASP.NET MVC Web Application project to your web app resource, you have the option to do so.

Important The Azure Resource Group project template uses Windows PowerShell scripts to communicate with the Azure Resource Manager when provisioning and deploying your resource group and resources. These scripts are executed from your local computer. Therefore, unless you have already done so, you will need to set your Windows PowerShell execution policy on your computer to allow Visual Studio to run the scripts. You can do this from the Windows PowerShell ISE or Windows PowerShell console using the following command: Set-ExecutionPolicy-ExecutionPolicy RemoteSigned-Force. For more information on this command, see <https://technet.microsoft.com/en-us/library/hh849812.aspx>.

Create an Azure Resource Group project

To begin, from the Visual Studio main menu select **File > New > Project**. In the New Project dialog, expand the Templates node on the left and click the **Cloud** node to view the available cloud templates. From the list of templates, select the **Azure Resource Group** template. The New Project dialog will look

similar to Figure 1-15.

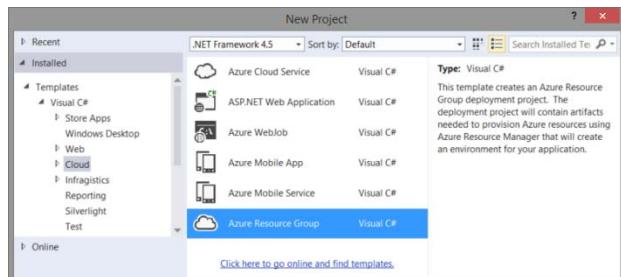


FIGURE 1-15 New Azure Resource Group template.

The next page in the New Project Wizard displays templates for common cloud configurations. At the top of the list are templates to create a Web App and Web App + SQL Database. To demonstrate how to create the environment from the previous section using the Azure portal, select **Web App + SQL**, as shown in Figure 1-16, and click **OK**.

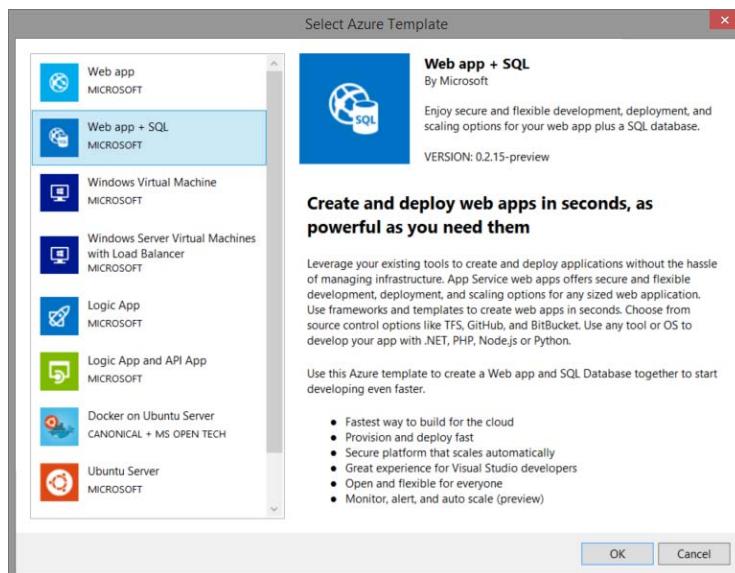


FIGURE 1-16 Dialog to select an Azure Resource Group template.

Visual Studio will create a single Azure Resource Group project that you will be able to see in the Solution Explorer window. Expand the folders in the deployment project to view the contents, as shown in Figure 1-17.

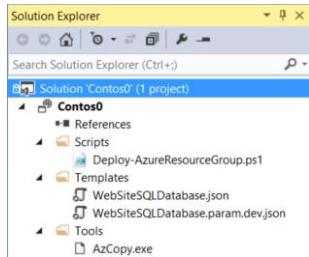


FIGURE 1-17 Solution Explorer with an Azure Resource Group project.

The contents of the deployment project are grouped in folders as follows:

- **Scripts** This folder contains a Windows PowerShell script that is used to fully automate creating the web app environment. If you open the script and look near the bottom of the script, you will see that it calls the Azure PowerShell command New-AzureResourceGroup to create the resource group and resources (web app and SQL database).
- **Templates** This folder contains two JSON files. The WebSiteSQLEDatabase.json file describes the Azure Resource Group and Azure Resources (web app and SQL database) that the Azure Resource Manager will use to provision the web app environment. The WebSiteSQLEDatabase.param.dev.json file is a parameters file that defines resource-specific parameters.
- **Tools** This folder contains a copy of AzCopy.exe and is used to copy files to an Azure Storage Account used during deployment. This is only used if you plan to deploy project artifacts such as an ASP.NET MVC Web Application using this deployment script in the Scripts folder.

See Also PowerShell tools for Visual Studio 2013 and Visual Studio 2015 are available that enable you to develop and debug PowerShell scripts directly in your Visual Studio environment. If you install the PowerShell Tools for Visual Studio, you also get syntax highlighting, brace matching, and IntelliSense, making it easier to read and edit PowerShell scripts. To learn more about the PowerShell Tools for Visual Studio and to install them, see <https://visualstudiogallery.msdn.microsoft.com/c9eb3ba8-0c59-4944-9a62-6eee37294597>.

Add a resource to a resource group

You can add additional resources, such as an Azure Redis Cache, to the resource group by editing the JSON template. To demonstrate this, double-click the **WebSiteSQLEDatabase.json** file to open it in the Visual Studio editor. This file is structured into two sections: *resources* (web app, SQL database, etc.) and *parameters* used to configure the resources. Figure 1-18 shows the two sections collapsed in the Visual Studio editor.

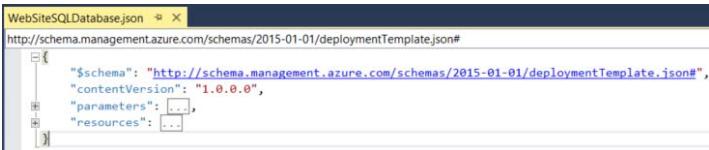


FIGURE 1-18 Azure resource group deployment template.

If you expand the resources section, you see the JSON descriptions of the resources for this template, which include the web app, SQL database, application insights, and more. Adding a resource such as an Azure Redis Cache involves adding the JSON description of the resource to the resources section of the file. Fortunately, Visual Studio provides a UI tool to do this called JSON Outline. Open this by selecting **View > Other Windows > JSON Outline** from the Visual Studio menu. The JSON Outline will appear as shown in Figure 1-19.

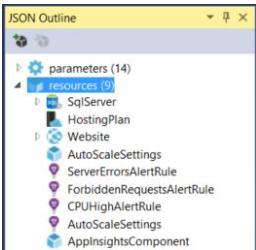


FIGURE 1-19 JSON Outline window showing resources from the resource group deployment template.

Notice the resources described in the deployment file are visualized in the JSON Outline window. To add an Azure Redis Cache, right-click the resources node and select **Add New Resource**. Scroll down the list of resource types and select the **Redis Cache** resource. Specify a name for the resource and click **Add** as shown in Figure 1-20.

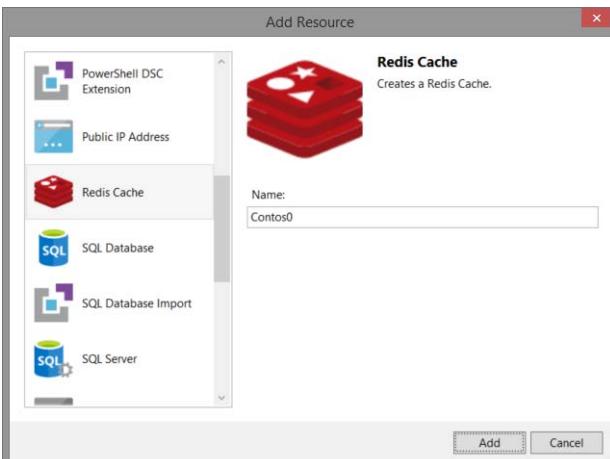


FIGURE 1-20 Add a resource to an Azure Resource Group project.

If you expand the resources section of the JSON deployment template file, you will see the resource description for the Redis Cache near the end of the section, as shown in Figure 1-21.

```
{  
  "name": "[parameters('Contos0Name')]",  
  "type": "Microsoft.Cache/Redis",  
  "location": "[parameters('Contos0Location')]",  
  "apiVersion": "2014-04-01-preview",  
  "dependsOn": [ ],  
  "tags": {  
    "displayName": "Contos0"  
  },  
  "properties": {  
    "sku": {  
      "name": "[parameters('Contos0SKUName')]",  
      "family": "[parameters('Contos0SKUFamily')]",  
      "capacity": "[parameters('Contos0SKUCapacity')]"  
    },  
    "redisVersion": "[parameters('Contos0RedisVersion')]"  
  }  
}
```

FIGURE 1-21 JSON description of an Azure Redis Cache.

Note If you want to remove a resource from a deployment template, you can do so using the JSON Outline window. Right-click on the resource you want to remove and select **Delete**. This will delete the resource in the resources section of the file.

Press **Ctrl+S** to save the changes to the JSON deployment template.

See Also For more information about editing resource manager deployment templates using the JSON Editor and JSON Outline, see the Azure SDK 2.6 announcement at
<http://azure.microsoft.com/blog/2015/04/29/announcing-the-azure-sdk-2-6-for-net/>.

Deploy a resource group using Visual Studio

The Deploy-AzureResourceGroup.ps1 PowerShell script is used to deploy your resources to Azure. It creates an Azure Resource Group and then sends the deployment template file and template parameter file (the two JSON files) to the Azure Resource Manager (ARM). ARM then takes this information and provisions the resources in Azure. You can run this deployment script from a Windows PowerShell console or you can invoke it directly from Visual Studio.

To demonstrate deploying from Visual Studio, right-click on the project and select **Deploy > New Deployment**. A dialog to deploy to a resource group will be displayed, as shown in Figure 1-22.

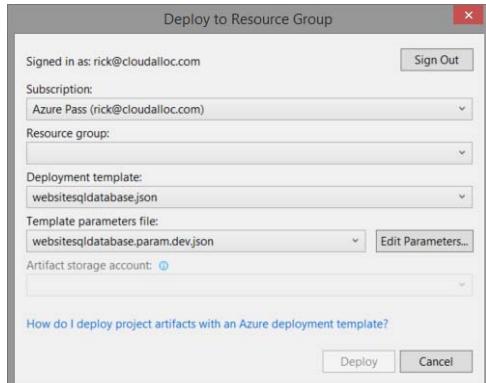


FIGURE 1-22 Deploy to Azure Resource Group.

In the Resource group field, you can choose an existing resource group to deploy to or create a new resource group. Select the option <**Create New...**>. This will open another dialog where you can specify the resource group name and region for the new resource group, as shown in Figure 1-23. Click **Create** to create the resource group and return to the previous dialog.



FIGURE 1-23 Create a new Azure Resource Group.

In the Deploy to Resource Group dialog, the fields for Deployment template and Template parameters file default to the two JSON files discussed earlier and, therefore, usually won't need to be changed.

Tip You can create different template parameter files for different environments, enabling you to automate the deployment to multiple environments.

The template parameters file in the Templates folder of the deployment project is where resource-specific settings for your deployment are stored. Because this is the first deployment, you must specify required settings that have not been provided yet. Click the **Edit Parameters** button. This will open the Edit Parameters dialog, as shown in Figure 1-24.

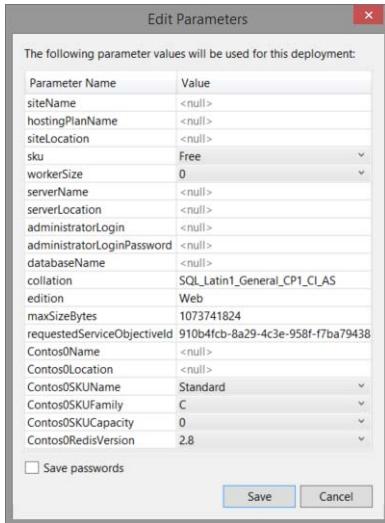


FIGURE 1-24 Missing parameters from the deployment template parameters JSON file.

The required parameters that need to be specified have a <null> value. Others have default values that you can change to meet your needs.

Note Location parameters such as siteLocation have to be typed instead of selected from a dropdown menu. Be careful to enter the location text exactly as Azure expects or you will have problems with your deployment. You can get the Azure region names and services available in each region at <http://azure.microsoft.com/en-us/regions/#services>.

After you fill in the required parameters, the Edit Parameters dialog will look similar to Figure 1-25.

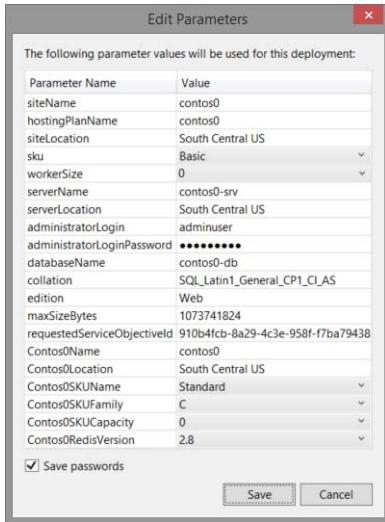


FIGURE 1-25 Parameters filled in to store in the deployment template parameters JSON file.

Click **Save** to save the changes to the template parameters file. The parameters entered in Figure 1-25 are stored in the JSON file named WebSiteSQLDatabase.param.dev.json in the Template folder of the project. At any time, you can go back and edit these values using the Visual Studio JSON editor or by using the Edit Parameters dialog.

In the Deploy to Resource Group dialog, click **Deploy** to start the deployment. The Output window will display logging as the resource group and resources are provisioned in Azure. The time it takes to deploy will depend on the number of resources and types of resources described in your deployment template. Figure 1-26 shows a brief section of the logging in the Output window as the resources are provisioned.

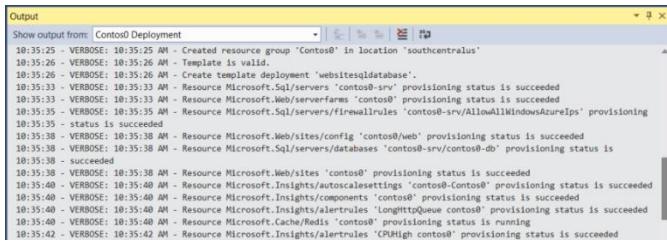


FIGURE 1-26 Output window showing resource group deployment logging during deployment.

Tip If you don't see the Output window in Visual Studio, select **View > Output** from the main menu.

After a few minutes, you will have a resource group provisioned that contains an Azure web app, SQL database, Azure Redis Cache, and Application Insights. To verify, open the Azure portal in your browser.

browser and select **Browse > Resource Groups**. Click on the resource group you just created and explore the resources in the group. You will see the same resources that you saw earlier in the chapter in Figure 1-9.

See Also For a detailed walkthrough creating and deploying Azure Resource Groups using the Azure Resource Group project template see <https://msdn.microsoft.com/en-us/library/azure/dn872471.aspx>.

Connection strings and application settings

In the past, it was common practice to store application settings and connection strings in the web.config file for an application. This led to another common practice: encrypting portions of the web.config file that contained sensitive information. This doesn't address the problem of having sensitive information checked into your source control system that your entire development team has access to.

Azure Web Apps offers a better solution whereby your application settings and connection strings can be stored in the web app *environment* and then retrieved by your application at runtime. This technique avoids the problems described above and offers some additional benefits. For example, your Dev/Test environment should be configured to use different resources (databases, non-production web services, and so on) from your production environment. By storing these settings in the environment, you can publish your application to different environments, knowing that it will be using the correct resource for each environment. In addition, you won't have to modify configuration files or perform configuration transforms when publishing the application to different environments. And, by using Role Based Access Control (RBAC), you can restrict access to the production environment to essential personnel who should have access to that environment. RBAC is discussed later in this chapter.

Set connection strings and app settings in the environment

The Cloud Deployment Project templates for Azure Web Apps take full advantage of this feature for you. If you look in the web.config file for the ASP.NET project, you will see the connection string for the LocalDB that is used for local development and testing. But you will not find connection string information for the SQL database that was provisioned because the templates stored it in your Azure web app environment. To see this, expand the App Service node in Server Explorer so you can see the web app that was created earlier. Right-click the web app and select the option to **View Settings**, as shown in Figure 1-27.

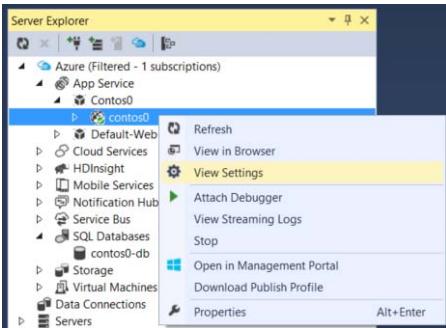


FIGURE 1-27 Access the Web App settings from Server Explorer.

In the Settings window, you can edit the configuration for the web app by clicking the **Configuration** tab on the left of the window. The Configuration tab is divided into sections for Web App Settings, Connection Strings, and Application Settings. Scroll down to the Connection Strings section to see the connection string for the SQL database provisioned in Azure, as shown in Figure 1-28.

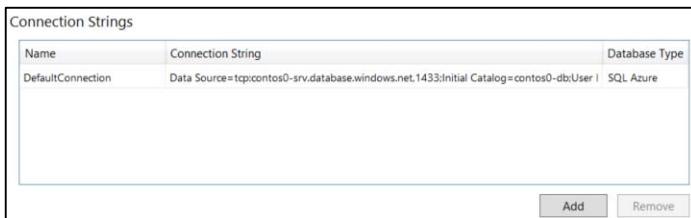


FIGURE 1-28 Connection Strings for Azure Web App.

Notice that the name of the connection string is the same as the name of the connection string in `web.config` for the LocalDB. This is intentional because connection strings and app settings defined in the *environment* will override settings of the same name defined in `web.config`. So when you are developing locally, the connection string to your LocalDB in `web.config` is used, but when the application is deployed and running in Azure, the connection string defined in the environment is used.

Another thing to notice is the value in the Database Type column, which is set to SQL Azure. This indicates that a SQL Azure database is the database the connection string is intended to be used for.

Note The SQL Azure database type is carried over from earlier brandings of what today is referred to as SQL Database. The Visual Studio tools still use this name, but the Azure portal correctly displays SQL Database as the name for this database type.

There are four database types as follows:

- **SQL Azure / SQL Database** Indicates the connection string is for a SQL database.
- **SQL Server** Indicates the connection string is for a SQL server such as an Azure Virtual Machine (or cluster) running SQL Server.
- **MySQL** Indicates the connection string is for a MySQL database.
- **Custom** A connection string to essentially any storage resource. For example, you could store the connection string the Azure Storage Client Libraries use for connecting to an Azure Storage account.

The Application Settings section essentially works the same as the Connection Strings section, except that this section is used to specify key/value pair settings that normally would be in the <appSettings> section of the web.config. This could be useful, for example, when storing the Redis Cache access key and endpoint address for the web app to use. To do this, click the **Add** button in the Application Settings section and add a key/value pair for the Redis Cache access key and another for the cache endpoint. The Name for the key/value pair can be any string you want. The Value can be retrieved from the Azure portal. Figure 1-29 shows what this section may look like.

Application Settings	
Name	Value
WEBSITE_NODE_DEFAULT_VERSION	0.10.32
RedisAccessKey	6e7LyUrqjU3Ix3ZC9rONsA5m51pnWm/sLsBDjT+jM=
RedisEndpoint	contos0.redis.cache.windows.net

Add Remove

FIGURE 1-29 Application Settings for Azure Web App.

After adding your settings, remember to click the **Save** button in your Visual Studio toolbar or press **Ctrl+S**. This will apply the settings to the web app environment in Azure.

Retrieve connection strings and app settings from the environment

In your application code, you can retrieve the connection strings and app settings by using the ConfigurationManager class. If you are running your application locally, it will retrieve the values for connection strings and app settings from your configuration files. If you are running the application in Azure, it will retrieve the values for connection strings and app settings from the environment. Using the settings from the previous section, you could have static variables defined somewhere in your application, as shown in Listing 1-1.

LISTING 1-1 Static variables to identify connection string and app setting key/value pairs.

```
// Name of app setting
```

```
private static string redisAccessKey = "RedisAccessKey";
private static string redisEndpoint = "RedisEndpoint";

// Name of connection string setting
private static string defaultConnection = "DefaultConnection";
```

Then, using ConfigurationManager, you can retrieve the values, as shown in Listing 1-2.

LISTING 1-2 Retrieve connection string and app setting values.

```
// Get app settings
string redisAccessKeyValue = ConfigurationManager.AppSettings[redisAccessKey];
string redisEndpointValue = ConfigurationManager.AppSettings[redisEndpoint];

// Get connection string
string contosoDbConnStr =
ConfigurationManager.ConnectionStrings[defaultConnection].ConnectionString;
```

It is important to note that this strategy works only if you use the same names for connection strings and app settings in your configuration files as you do in your web app environment. When you do, the Azure web app environment setting will take precedence over a setting of the same name in a configuration file. If the setting is not defined in the web app environment, then ConfigurationManager will return the value from the configuration file if it has been set.

As mentioned earlier in this section, storing sensitive data in web.config is not recommended. Guidance on storing and deploying sensitive data for ASP.NET Applications to Azure is available at <http://www.asp.net/identity/overview/features-api/best-practices-for-deploying-passwords-and-other-sensitive-data-to-aspnet-and-azure>. The technique described in this guidance recommends moving the connection strings and app settings to external files that are *referenced* from the web.config file and then never checking the external files holding these settings into source control.

How connection strings and app settings are stored in the environment

How connection strings and app settings are stored as environment variables is not necessarily important if you are using ASP.NET. As demonstrated in the previous sections, this is all transparent to you when using the ConfigurationManager class to retrieve the values. But if you are using other frameworks such as PHP, then you need to know this so you know the name of the environment variable you want to retrieve.

When you set connection strings and app settings in the environment, they are stored as environment variables. Connection string environment variables have a naming convention that identifies them with the database types discussed previously. The naming convention is <database type prefix><connection string name>, where the prefix for each of the database types is shown in Table 1-1.

TABLE 1-1 Connection string environment variable prefixes

Connection String Type	Environment Variable Prefix
SQL Database	SQLAZURECONNSTR_
SQL Server	SQLCONNSTR_
MySQL	MYSQLCONNSTR_
Custom	CUSTOMCONNSTR_

As an example, for the connection string named DefaultConnection for the SQL database discussed earlier, the environment variable will be named SQLAZURECONNSTR_DefaultConnection in the Azure Web App environment.

Similarly, application settings are stored as environment variables named APPSETTING_<setting name>, where <setting name> is the name of the key/value specified in the Application Settings section.

You can see these environment variables using the Site Control Manager ("Kudu") for your web app by browsing to <https://<appname>.scm.azurewebsites.net>, where <appname> is the name of your web app. If you are prompted to sign in, use the same credentials you use to sign in to the Azure portal. After signing in, click the **Environment** menu option at the top of the page and then scroll down to the Environment Variables section of the page. The application settings defined previously will be close to the top of the section, as shown in Figure 1-30.



FIGURE 1-30 App settings as environment variables in the Kudu portal.

Scroll down toward the bottom and you will see environment variables for the connection string, as shown in Figure 1-31.



FIGURE 1-31 Connection string as environment variables in the Kudu portal.

Add a deployment slot for an Azure Web App

One of the many benefits of an Azure web app running in a Standard or Premium pricing tier is the ability to add deployment slots. A deployment slot technically is an independent web app with its own content, configuration, and even a unique host name. So it functions just like any other web app.

There are many benefits to adding a deployment slot to your web app, including the following:

- You can deploy changes for your application to a staging deployment slot and test the changes without impacting users who are accessing the production deployment slot. When you are ready to move the new features into production, you can just swap the staging and production slots with no downtime.
- You can do A/B testing with a small set of users to try out new features of your application without impacting the majority of users who are using the production slot. This is an extremely powerful scenario, and you can learn more about it at <http://blogs.msdn.com/b/tomholl/archive/2014/11/10/a-b-testing-with-azure-websites.aspx>.
- You can “warm up” your application in a staging slot before swapping it into the production slot, avoiding the long delays a cold start of your application may incur because of some lengthy initialization code.
- You can swap back to the previous deployment if you realize that the new version of your application is not working as you expected.

Scale to a Standard App Service Plan

Deployment slots are available in Standard and Premium pricing tiers. The Standard plan allows up to 5 deployment slots, while the Premium plan provides up to 20. You can have different deployment slots for different stages of your application lifecycle such as development, test, user acceptance testing, integration, staging, and more.

To change the pricing tier for a web app using the Azure portal, click **Browse > Web Apps**. In the Web Apps blade, click the web app created previously to open the Web App blade. In the Web App blade, scroll down to the Usage section and click the **Pricing Tier** part, as shown in Figure 1-32.

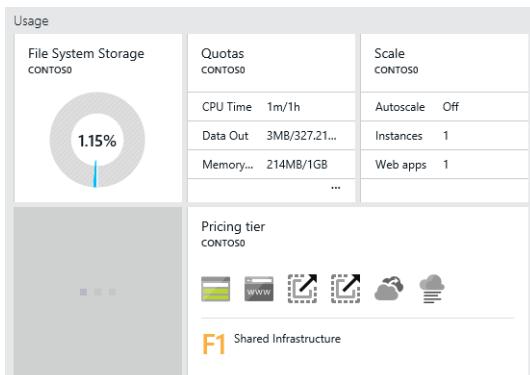


FIGURE 1-32 Usage section of Web App blade showing Pricing Tier part.

In the Pricing Tier blade, select the **S1 Standard pricing tier**, as shown in Figure 1-33, and then click

the **Select** button to apply the change.

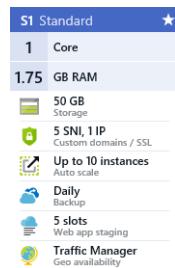


FIGURE 1-33 S1 Standard pricing tier.

Add a deployment slot

In the Web App blade, scroll down to the Deployment section and click the **Deployment slots** part, as shown in Figure 1-34.



FIGURE 1-34 Deployment slots part in the Web App blade.

In the Deployment slots blade, click the **Add slot** button in the toolbar. In the Add a slot blade, enter a name such as "staging" for the deployment slot name and click **OK**. After a few seconds, the new staging deployment slot will appear in the Deployment slots blade, as shown in Figure 1-35.



FIGURE 1-35 Deployment slots blade.

The web app now has two deployment slots: the production slot that is the default when the web app is first created and a staging slot. If you click the name of the staging slot, it will open the Web App blade where you can configure and manage the staging web app just as you would the production web app.

Tip Deployment slots for a web app incur costs just as the production slot does. So to minimize costs, you should consider stopping your non-production deployment slots when you are not using them. You can do this in the Web App blade for the deployment slot by clicking the **Stop** button in the

toolbar.

Close the Deployment slots blade to return to the Web App blade for the production slot. Scroll down to the Deployment section and observe that the Deployment slots part is now showing one slot.

For more information on adding, configuring, and managing deployment slots, see <http://azure.microsoft.com/en-us/documentation/articles/web-sites-staged-publishing>.

Set up continuous deployment with Visual Studio Online

Support for continuous deployment is available for Azure Web Apps running in any pricing tier, including the Free tier. By using continuous deployment, you can configure your web app to build and deploy your application automatically when you push changes into your source code repository. By using continuous deployment from source control, development and test teams can greatly increase the rate at which software changes are published to support business operations.

The continuous deployment feature of Azure Web Apps works with many popular source control systems, such as Visual Studio Online, GitHub, Bitbucket, and Dropbox. Combined with Team Explorer, which is part of Visual Studio, developers can push changes to source control quickly and monitor builds and deployments, all directly from Visual Studio.

When you set up continuous deployment for your Azure web app, you generally will want to target a deployment slot other than your production slot, such as a deployment slot designated for Dev/Test purposes. When changes are pushed into your source control repository, they can be compiled automatically and deployed to the Dev/Test slot and then swapped into staging or production after the changes have been verified. Or if your Dev/Test environment is one that takes many changes from many developers, you may want to have builds and deployments scheduled hourly. Both scenarios are possible with Visual Studio Online. Using the staging deployment slot added in the previous section, this section will demonstrate continuous deployment as changes are pushed into your source code repository.

Introduction to Visual Studio Online

Visual Studio Online is a complete application lifecycle management tool that is delivered as a service. If you are familiar with Team Foundation Server (TFS) in an on-premises environment, then you can think of Visual Studio Online as a software-as-a-service (SaaS) version of TFS. Visual Studio Online provides a build service to automate builds and features for team collaboration, work items, source control, user stories, backlogs, Kanban boards, load testing, reporting, and more. In this section, we look at the source control feature of Visual Studio Online and setting up continuous deployment for an Azure Web App.

Visual Studio Online supports two types of source control repositories as follows:

- **Team Foundation Version Control (TFVC)** A centralized server repository where developers check out, edit, and check in source code changes
- **Git** A distributed version control system where developers apply changes to a local repository and then share their changes by pushing and pulling changes through a remote shared repository

Today, most development teams are using Git for their source control needs. So Git will be the type of repository used for discussion in this section.

Although Visual Studio Online integrates seamlessly with Azure, you need a separate Visual Studio Online account to use it. If you don't already have one, you can get one for free at <https://www.visualstudio.com/get-started/setup/sign-up-for-visual-studio-online>.

Create a Visual Studio Online project with Git

Visual Studio's Team Explorer brings the features of Visual Studio Online directly into your development environment. If Team Explorer is not visible in your Visual Studio environment, select **View > Team Explorer** from the main menu. Next, click the plug icon in the Team Explorer toolbar, as shown in Figure 1-36.

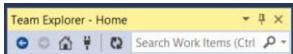


FIGURE 1-36 Team Explorer toolbar.

Next, click the link that says **Create Team Project**. This will launch your browser and take you to the Visual Studio Online portal. If you are prompted to sign in, use the credentials you used when you set up your Visual Studio Online account.

In the Create New Team Project dialog, specify a project name and change the Version control field to Git, as shown in Figure 1-37.

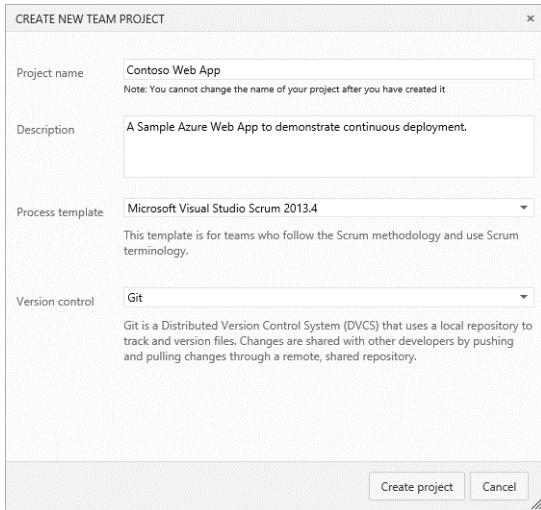


FIGURE 1-37 Create a new team project in Visual Studio Online.

Click the Create project button. In a few seconds, your project and Git repository will be ready in Visual Studio Online. When the project is ready, a dialog will show some team project information where you can navigate to the project or close the dialog. Click the Close button to close it.

Visual Studio Online is an amazing service with a massive number of features we cannot cover here. If you want to learn more about it, go to <http://azure.microsoft.com/en-us/services/visual-studio-online/>.

Set up deployment from source control to a staging slot

To set up deployment from source control so that deployments are published to a staging slot, you will need to sign in to the old portal at <https://manage.windowsazure.com> because the Azure portal does not support this capability at the time of this writing.

In the old portal, click **Web Apps** in the left navigation pane. For the web app created previously, click the arrow icon to the left of the name to expand the web app so you can see the staging deployment slot, as shown in Figure 1-38.

web apps	
NAME	STATUS
contoso0	✓ Running
contoso0(staging)	✓ Running

FIGURE 1-38 Web App with production and staging slot.

Click the staging deployment slot name and then click the **Dashboard** tab at the top of the page. In

the quick glance section of the dashboard page, click the link to **Set up deployment from source control**. This will launch a wizard, the first page of which will ask where your source code will reside, as shown in Figure 1-39.

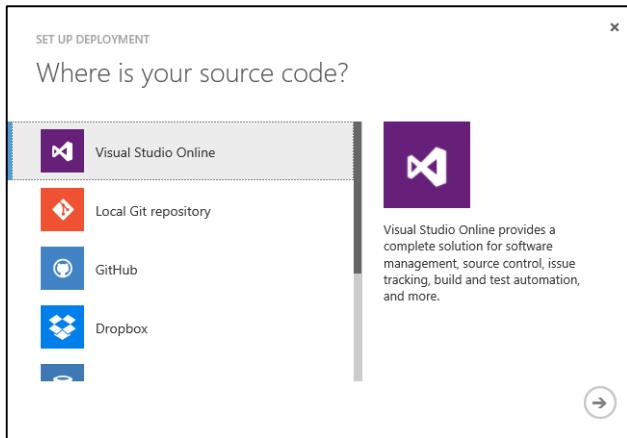


FIGURE 1-39 Set up deployment from source control.

Select the Visual Studio Online option and then click the right arrow to continue.

On the next page of the wizard, you can specify the name of your Visual Studio Online account and authorize Azure Web Apps to access the account for the purposes of publishing changes when you push them into source control. Enter the name of your account and then click the **Authorize Now** link. Visual Studio Online will ask you to confirm this authorization, as shown in Figure 1-40.

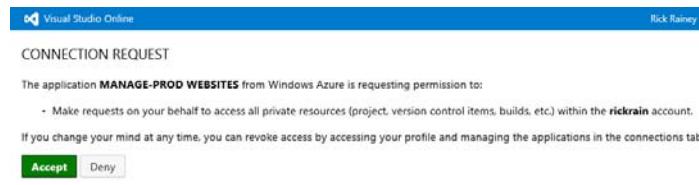


FIGURE 1-40 Visual Studio Online authorizing the connection request from Azure Web Apps.

Click **Accept** to accept the connection request.

The final page of the wizard will display a message indicating the authorization was successful and then ask you to select the repository you want Azure Web Apps to deploy from. Select the repository you previously created in Visual Studio Online, as shown in Figure 1-41.

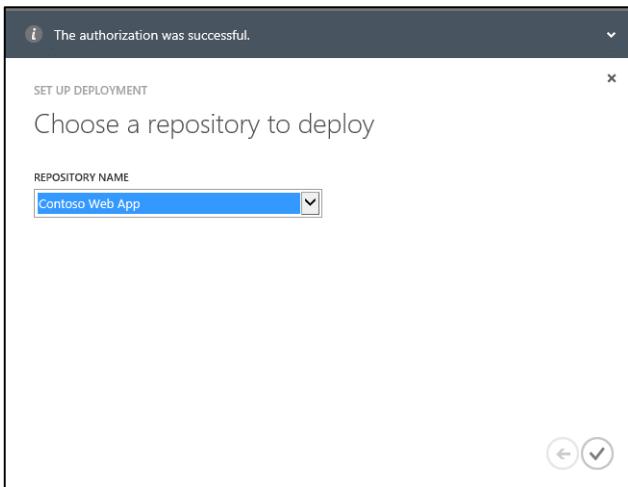


FIGURE 1-41 Choose a repository to deploy from

Click the check mark button to complete the wizard. You will be redirected to the DEPLOYMENTS page for the staging slot of the web app, where a message will indicate that your team project is linked and that Visual Studio Online will build and deploy your project on the next check-in.

The Azure web app now is configured for continuous deployment to your staging deployment slot.

Add Visual Studio solution to source control

The previous sections linked the Azure web app environment and your Visual Studio Online Git repository for the project so that Visual Studio Online can build and then publish deployments to your Azure web app staging slot. But you still need to link the Visual Studio solution for your web application to your Visual Studio Online project so you can push changes to the remote repository in Visual Studio Online.

In the Solution Explorer window in Visual Studio, right-click the solution and select the option to **Add Solution to Source Control**. When prompted to choose between Team Foundation Version Control or Git, choose Git.

Commit Visual Studio solution to source control

In the Team Explorer window, click the **Changes** option, as shown in Figure 1-42.

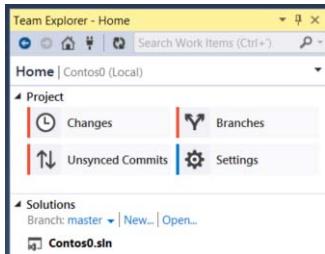


FIGURE 1-42 Team Explorer project options.

Enter a description such as “Initial Web App Commit” and click the **Commit** button to commit the solution to the local Git repository. Next, click the **Sync** link to sync the changes with the remote repository in your Visual Studio Online project. The first time you sync, you will need to provide the URL for the Git repository, which you can get from your Visual Studio Online portal in the Code page. Copy the Git URL and paste it into the remote repository field. The Team Explorer window will look similar to Figure 1-43.

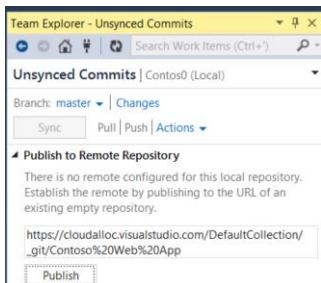


FIGURE 1-43 Remote repository URL in Unsynced Commits page of Team Explorer.

Click the Publish button to push the changes to the remote Git repository in your Visual Studio Online project. Because you linked the Visual Studio Online project to your Azure web app, Visual Studio Online will build and then publish the application to the staging deployment slot. When you are ready to move it to your production slot, you can click the **Swap** button at the bottom of the Dashboard page in the Azure portal.

Role Based Access Control

Role Based Access Control (RBAC) is an access control feature in the Azure platform that enables you to manage access to resources in your Azure subscription. With RBAC, you are able to get granular with your access permissions so that only users (or groups) have access to the resources they need. This is very different from previous access management scenarios in which the only ways to give a user management capabilities were to add the user as a co-administrator to the subscription or to create a management certificate, giving the user access to the entire subscription.

RBAC uses Azure Active Directory *exclusively* to authenticate users and authorize access to resources based on the user's roles. The RBAC feature is available in the new Azure portal (in preview) and programmatically through the Azure Resource Manager (ARM) REST APIs. The ARM REST API documentation is available at <https://msdn.microsoft.com/en-us/library/azure/dn790568.aspx>.

Note If you are using the Azure Service Management REST APIs to manage resources in your Azure subscription programmatically, then you should start getting familiar with the ARM REST APIs. The ARM REST APIs eventually will replace the Azure Service Management REST APIs.

Subscription-level roles

RBAC can be managed at the subscription level by using three built-in roles provided by the Azure platform:

- **Owner** Can create and manage resources of all types, including managing access for child resources.
- **Contributor** Can create and manage resources but cannot add or delete role assignments for users.
- **Reader** Can read resources but cannot read secrets. For Azure Web Apps, secrets would be things such as application settings, connection strings, and deployment credentials.

If you are a service administrator or co-administrator on a subscription, then you automatically are added to the Owners role. When you assign owners to these roles, the level of permissions for the role applies to all resources in the subscription.

Using the Azure portal, you can view these roles and manage users in these roles from the Subscription blade. To get to the Subscription blade, select **Browse > Subscriptions**. In the Subscriptions blade, click the subscription to open the Subscription blade. Scroll down to the Access section to locate the Roles and Users parts, as shown in Figure 1-44.

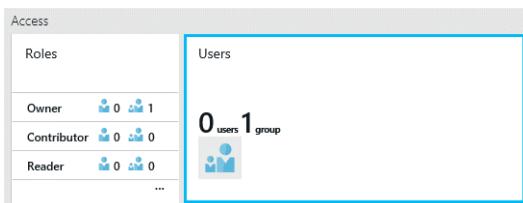


FIGURE 1-44 Roles and Users parts in the Azure Subscription blade.

To add a user to one of the roles, click the role to open the Role blade. Next, click the **Add** button at the top of the Role blade to open the Add Users blade. In the Add Users blade, you can add individual users or groups from the Azure Active Directory.

Tip You can add an external user to a role, provided the user has a Microsoft Account. When you do this, an external user will be added to your Azure Active Directory.

Resource-level roles

Perhaps the greatest benefit of RBAC is the ability to assign user access to a single resource. For example, if you want a user or group to have access to a single web app, then you could add the user to the Contributor role from the Web App blade. To do this, click the **Settings** icon in the toolbar for the Web App blade. In the Settings blade, click **Roles** to open the Roles blade, on which you can add a user or group to one of the roles for that resource. Notice in Figure 1-45 that the three built-in roles discussed earlier are present in addition to two roles that are specific to Azure Web Apps. These additional roles are made possible by the Azure Web App Resource provider. To learn more about resource providers, see <https://msdn.microsoft.com/en-us/library/azure/dn790572.aspx>.

The built-in roles are inherited from the subscription. However, adding users or groups at this level (the Web App Resource blade) limits their access to this web app only.



NAME	USERS	GROUPS
Owner	0	1
Contributor	0	0
Reader	0	0
User Access Administrator	0	0
Website Contributor	0	0

FIGURE 1-45 Built-in roles for Azure Web Apps.

For an example using this feature to assign a developer permissions to publish source code changes, see <http://azure.microsoft.com/blog/2015/01/05/rbac-and-azure-websites-publishing/>.

To learn more about RBAC, built-in roles, and managing permissions to resources, see <http://azure.microsoft.com/en-us/documentation/articles/role-based-access-control-configure/>.

Summary

This chapter started with an introduction to Azure Resource Groups and App Service Plans and then applied that knowledge to demonstrate different scenarios for creating Azure Web Apps and resources. We discussed a unique feature of Azure Web Apps for storing and retrieving application settings and connection strings. Next, we discussed how to add deployment slots and configure continuous deployment to a staging deployment slot by using Visual Studio Online. This chapter concluded with a discussion about RBAC and how it can be used to assign user permissions to resources.

Chapter 2

Azure WebJobs

One of the many features provided by Azure Web Apps is the ability to run background processes to augment your web application. This feature is called Azure WebJobs.

Azure WebJobs addresses a need common to many websites, which is to offload time-consuming or CPU-intensive tasks to another process. This helps ensure that the web front end that users are interacting with is able to respond to user requests quickly while asynchronously performing work in another process. Azure WebJobs also provides an excellent solution for running routine maintenance tasks such as moving log files or updating databases the web app uses.

Microsoft provides a .NET SDK specifically for Azure WebJobs that further enhances this feature. While use of the SDK is not technically required, it significantly simplifies some common patterns for invoking web job functions.

If you use Azure WebJobs as part of your web app solution, the Azure WebJobs Dashboard provides a nice web interface where you can view, monitor, and invoke your web jobs.

This chapter will introduce you to the Azure WebJobs feature and take you on a journey through each of these topics.

Introduction to Azure WebJobs

Because it is a feature of Azure Web Apps, you might assume that developing an Azure WebJob will involve using web frameworks such as ASP.NET, MVC, or Web API, to name a few. But this is not the case. In fact, the C# code in Listing 2-1 could be compiled and deployed as a web job. It certainly is not useful, but its simplicity is well received by developers.

LISTING 2-1 A simple .NET console application.

```
class Program
{
    static void Main()
    {
        Console.WriteLine("Hello from web job.");
    }
}
```

As enticing as this may be, it is not the only choice for building web jobs. A web job also could be implemented using a variety of scripting languages. The following are all valid candidates for implementing a web job:

- .cmd, .bat, .exe (Windows)
- .ps1 (PowerShell)
- .sh (Bash)
- .php
- .py (Python)
- .js (Node.js)

When you add a web job to your web app environment, you must specify how your web job should be invoked. Azure WebJobs provides you three options to indicate how your web job should run.

These options are as follows:

- **Continuous** A continuous web job is always running and therefore may be implemented using a looping structure such as a while loop. But as you will see later, there are more elegant ways of implementing this kind of web job using the Azure WebJobs SDK.
- **Scheduled** A scheduled web job is invoked on a schedule. The schedule can be one where your web job is invoked only once at a date and time you specify. Or it can be a recurring schedule where your web job is invoked sequentially with a specified time span between invocations. A recurring schedule can be defined so that the time span between invocations is specified in minutes, hours, days, weeks, or months. As an example, the code in Listing 2-1 could be put on a recurring schedule to run every two hours.
- **On-Demand** An on-demand web job will be invoked only when you specifically take action to invoke it. You can invoke the web job by using the Azure portal, programmatically by using the WebJobs REST API, or by using Windows PowerShell.

Tip To invoke a web job by using Windows PowerShell, use the Start-AzureWebsiteJob cmdlet and set the JobType parameter to Triggered. To learn how you can use the WebJobs REST API to invoke a web job, see the documentation at <https://github.com/projectkudu/kudu/wiki/WebJobs-API>.

Create an Azure WebJob

To demonstrate some essential concepts for Azure WebJobs, we will leverage the code from Listing 2-1 to create a very simple .NET console application using Visual Studio. From the main menu, select **File > New > Project** and choose the Console Application template. For the name of the project, enter "SimpleWebJob."

In the Main method of the application, add the WriteLine statement from Listing 2-1.

Press **Ctrl+F5** to build and run the application locally.

Publish a web job from Visual Studio

To run this simple console application in Azure as a web job, you can leverage the Azure SDK tools to publish it to a web app environment. To do this, right-click the SimpleWebJob project in Solution Explorer and select **Publish as Azure WebJob**, as shown in Figure 2-1.

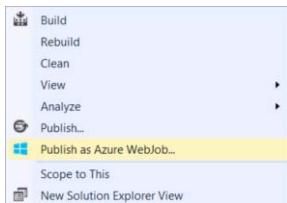


FIGURE 2-1 Publish as Azure WebJob from Visual Studio.

This will open a dialog where you can specify the name of the web job and configure how you want it to run. For this demonstration, configure the web job as follows:

Set the **WebJob run mode** setting to **Run on a Schedule**.

Set the **Recurrence** setting to **Recurring Job**.

Set the **Recur every** setting to **1 Hours**.

The Add Azure WebJob dialog will look similar to Figure 2-2. Click **OK** to proceed to the next dialog.

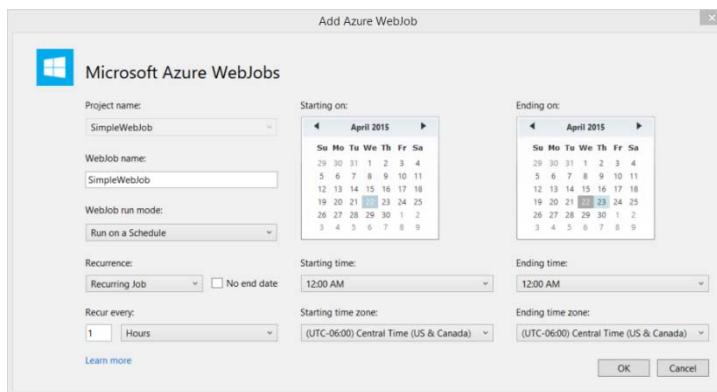


FIGURE 2-2 Add an Azure WebJob to run on a recurring schedule.

Note The ability to publish an Azure WebJob from Visual Studio is provided by the

Microsoft.Web.WebJobs.Publish NuGet package. When you click **OK** in the Add Azure WebJob dialog, you may see a dialog briefly appear, indicating that this package is being downloaded. It's quick, so you have to be paying attention or you may miss it. Whether you see it or not, this step adds a packages.config file to your console application project with a package reference to this NuGet package.

In the Publish Web dialog, click **Microsoft Azure Web Apps**, as shown in Figure 2-3.

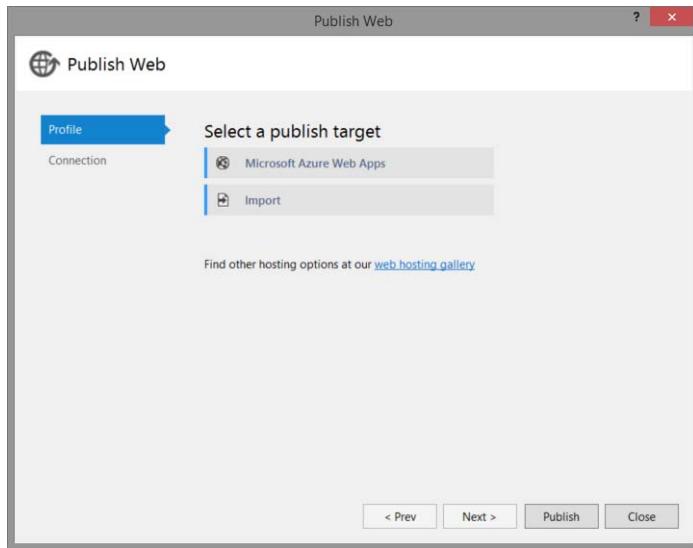


FIGURE 2-3 Select a publish target for the web job.

In the Select Existing Web App dialog, set **Existing Web Apps** to the web app created in Chapter 1, "Microsoft Azure Web Apps." If you don't have an existing web app, then click **New** to create a new web app. Click **OK** to continue.

Note A web job runs on the same machine hardware that the Azure web app is running on. For example, if you have an ASP.NET MVC application deployed in the web app environment and then add a web job to the same web app, the web job process (SimpleWebJob.exe) will share the machine resources the ASP.NET MVC application process (w3wp.exe) is running.

The ability to specify a new web app when publishing a web job gives you the option later of scaling out your web job instances independently where the web job is running in its own web app. In other words, you are not required to have a web application such as an ASP.NET MVC application in the web app environment to publish a web job. The web job can be the sole application running in the web app environment. A deployment strategy such as this may be appropriate if your web job is CPU-intensive and you don't want it consuming the resources of your web application.

After selecting an existing web app or creating a new web app, click **Publish** in the Publish Web

dialog to publish the web job to Azure. Because this is a scheduled web job, the Azure Scheduler service is used behind the scenes to invoke the web job on the schedule you configured. The Scheduler service uses the WebJobs REST API mentioned earlier to invoke the web job. If you are interested in learning more about the Azure Scheduler service, see <http://azure.microsoft.com/en-us/services/scheduler/>.

Tip If you want a frequency of less than once per hour for a scheduled web job, you will need to make sure your Scheduler service is configured for either the Standard or Premium plan. Additional information on the Scheduler limits and default settings is available at <https://msdn.microsoft.com/en-us/library/azure/dn479786.aspx>.

Invoke a web job manually

The web job you created in the previous section will be invoked by the Azure Scheduler service based on the schedule you specified. But you can also invoke it on demand just as you would an On-Demand web job.

To invoke the web job, expand the **WebJobs > On Demand & Scheduled** node in Server Explorer. Then, right-click the SimpleWebJob and select **Run**, as shown in Figure 2-4.

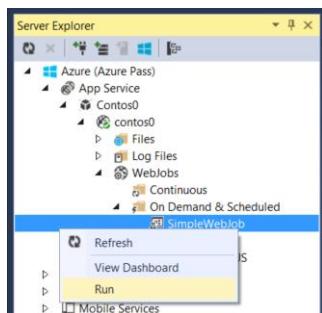


FIGURE 2-4 Invoke a web job from Visual Studio.

This web job is very small, so you may not even notice the “Starting...” text appear after the web job’s name as it starts and then quickly exits. So how can you tell if the web job ran successfully? That is what the WebJobs Dashboard is for, which will be discussed next.

View the WebJobs Dashboard

The WebJobs Dashboard is a site extension of the web app you can easily access from Visual Studio by right-clicking the web job in Server Explorer and selecting **View Dashboard**, as shown in Figure 2-4. This will launch your browser and place you directly into the details page for your web job in the WebJobs Dashboard. Your dashboard will look similar to Figure 2-5.

The screenshot shows the Microsoft Azure WebJobs dashboard. At the top, there's a header bar with the title "Microsoft Azure WebJobs". Below it, a navigation bar has "WebJobs" selected. The main content area is titled "WebJob Details" for "SimpleWebJob". A blue callout box contains the text "Run command: SimpleWebJob.exe". Under "Recent job runs", there's a table with two columns: "TIMING" and "STATUS". The first row shows "55 seconds ago (453 ms running time)" and "Success". Below the table, a note says: "Do more with [Microsoft Azure WebJobs SDK](#). The SDK integrates Microsoft Azure Storage, triggering a function in your program when items are added to Queues, Blobs, or Tables." The log output section shows a single entry: "Success a moment ago (297 ms running time) Run ID: 201504181504273118".

FIGURE 2-5 WebJob Details page in the WebJobs Dashboard.

This page shows the relative time your web job was run, how long it ran, and the status of the job run. The data shown in the Timing column is linkable, and clicking it will open the log file for that job run. If your application writes data to the console like this one does, you will see it in the output log file with other data that the host writes, as shown in Figure 2-6.

The screenshot shows the Microsoft Azure WebJobs dashboard. At the top, there's a header bar with the title "Microsoft Azure WebJobs". Below it, a navigation bar has "WebJobs / SimpleWebJob" selected. The main content area is titled "WebJob Run Details" for "SimpleWebJob". A green callout box contains the text "Success a moment ago (297 ms running time) Run ID: 201504181504273118". Below the log output section, a note says: "Do more with [Microsoft Azure WebJobs SDK](#). The SDK integrates Microsoft Azure Storage, triggering a function in your program when items are added to Queues, Blobs, or Tables." The log output section shows several entries from the host and the application itself:

```
[04/18/2015 15:04:27 > 4853c3: SYS INFO] Status changed to Initializing  
[04/18/2015 15:04:27 > 4853c3: SYS INFO] Run script 'SimpleWebJob.exe' with script host - 'WindowsScriptHost'  
[04/18/2015 15:04:27 > 4853c3: SYS INFO] Status changed to Running  
[04/18/2015 15:04:27 > 4853c3: INFO] Hello from web job.  
[04/18/2015 15:04:27 > 4853c3: SYS INFO] Status changed to Success
```

FIGURE 2-6 WebJob run details in the WebJobs Dashboard.

Create a web job from the Azure portal

You can add a web job to your web app using the Azure portal. This approach involves adding the web

job to the web app and then uploading the file that contains your web job application. If your web job is a script or .exe that has dependencies on other files such as scripts, DLLs, or configuration files, you must zip the files and upload the .zip file. When you upload a .zip file, the web app environment will automatically unzip the contents in a file location designated for web jobs.

Note If you want to learn more about where web jobs are stored and some of the environment settings specific to Azure WebJobs, see the WebJobs documentation at <https://github.com/projectkudu/kudu/wiki/Web-jobs>.

To demonstrate this approach using the Contoso0 web app, go to the web app blade in the Azure portal. In the toolbar of the web app blade, click the **Settings** icon.

In the Settings blade, click **WebJobs**. The WebJobs blade shows all the web jobs that exist for the web app. If you followed along in the previous section, the WebJobs blade will show the SimpleWebJob, as shown in Figure 2-7.

NAME	TYPE	STATUS	LOGS
SimpleWebJob	On Demand	Completed Just now	https://contoso0.scm.a...

FIGURE 2-7 WebJobs blade in the Azure portal.

Note If you click the link in the Logs column, a browser window will open and take you to the WebJobs Dashboard discussed in the previous section.

Create a new web job by clicking **Add** in the WebJobs blade toolbar. In the Add WebJob blade, configure the web job properties as follows:

1. Set the Name field to **SimpleWebJobFromPortal**.
2. Set the How To Run field to **On Demand**.
3. Click the **folder** icon next to the File Upload field and locate the SimpleWebJob.exe you compiled in the previous section. Because this application has no dependencies, you can upload the .exe without zipping it first.

The Add WebJob blade will look similar to Figure 2-8.

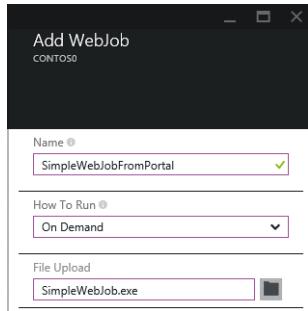


FIGURE 2-8 Add WebJob blade in the Azure portal.

Click **Create** at the bottom of the Add WebJob blade. This will return you to the WebJobs blade, and the SimpleWebJobFromPortal web job will be present.

If the web job is an On-Demand web job like the two that have been added so far, you can run the web job from the WebJobs blade by right-clicking the web job name and selecting **Run**. Similarly, you can delete a web job by right-clicking it and selecting **Delete**.

Introduction to the Azure WebJobs SDK

Using the Azure WebJobs SDK to build your web job simplifies some common application patterns by providing basic infrastructure that you otherwise would have to code yourself. For example, a common web pattern is one where a web application takes input from a user and then puts a message on a queue to be processed asynchronously. As messages are added to the queue, a separate background process pulls the messages off the queue to do the work. This helps eliminate bottlenecks in the web application by offloading time-consuming tasks to a separate background process. In Azure, a solution such as this may look similar to Figure 2-9.

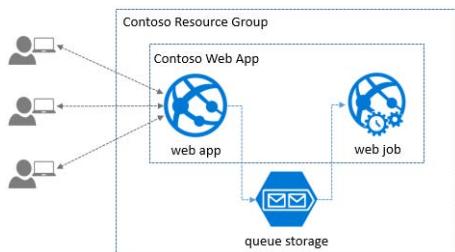


FIGURE 2-9 A queue-centric work pattern.

You could imagine extending the web job in the previous section to poll a queue for messages, keep track of a message's de-queue count, handle message visibility in the queue, delete the message after processing, and potentially move poison messages to a different queue. Or you could get all of

this for free by using the WebJobs SDK and spend more of your time coding the business logic of your web job and less time writing infrastructure code.

See Also For a complete tutorial on how to implement this type of pattern, see <http://azure.microsoft.com/en-us/documentation/articles/websites-dotnet-webjobs-sdk-get-started/>.

The WebJobs SDK puts in place infrastructure to automatically invoke methods in your web job that have been decorated with *trigger* attributes that are also provided by the SDK. These attributes are designed to work with Microsoft Azure Storage account services such as queues, blobs, and tables and with ServiceBus queues and topics. Methods in the web job that use these attributes are identified as *functions* in the web job and, as a result, become visible in the WebJobs Dashboard.

Using the WebJobs SDK lights up additional features in the WebJobs Dashboard, such as the ability to see basic statistics for each function in the web job. You can view function invocation logs and the data that was passed to the function for processing. You can even *replay* functions from the invocation log and edit the original data before it is passed to the function.

In this section, we will take a closer look at the web job infrastructure, attributes, and dashboard capabilities that become available when using the WebJobs SDK.

WebJobs SDK .NET libraries and dependencies

The WebJobs SDK is delivered through two NuGet packages: Microsoft Azure WebJobs and Microsoft Azure WebJobs Core. Installing the Microsoft.Azure.WebJobs package will automatically install the Microsoft.Azure.WebJobs.Core package for you. Because the WebJobs SDK is designed to work closely with services in your Azure Storage account, it will also install the Azure Storage NuGet package if you don't already have it.

You can start your web job development from a simple console application, add these packages to your project, and start coding. Or you can leverage the QuickStart templates that provide the client libraries, prerequisite code and configuration, and some sample methods that show some nifty ways to use the WebJobs SDK for common scenarios.

We will use the QuickStart templates in the following section to create a web job that uses the WebJobs SDK and then examine essential facets of the project.

Create a web job designed for use with Azure Storage Queues

From the Visual Studio main menu, select **File > New > Project**. In the New Project dialog, navigate to the **Cloud > QuickStarts > Compute** node and select the **Azure WebJobs SDK Queues** template, as shown in Figure 2-10.

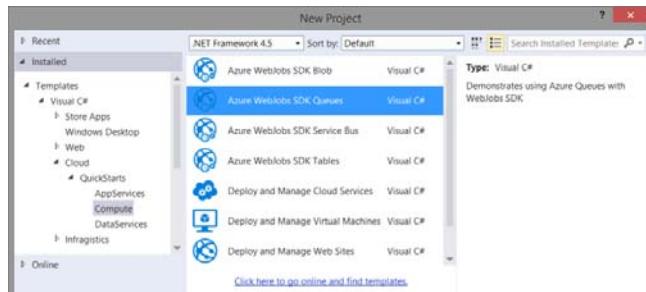


FIGURE 2-10 Create a new Azure WebJobs SDK Queue project in Visual Studio.

Set the **Name** for the project to **SampleQueueWebJob** and click **OK** to create the project.

From the main menu, select **Build > Build Solution**. This will download the NuGet packages needed for the project.

If you press **Ctrl+F5** to run the console application, you will get a message indicating that you need to add Azure Storage account credentials to your app.config.

Configure connection strings for Azure Storage

Open app.config in your project, and you will see two empty connection strings in the `<connectionStrings>` section named `AzureWebJobsDashboard` and `AzureWebJobsStorage`. Classes in the WebJobs SDK look for these two connection strings by name. The WebJobs Dashboard uses the `AzureWebJobsDashboard` connection string to store logs and parameter data surfaced by the dashboard. Your web job application uses the `AzureWebJobsStorage` connection string to bind methods in your web job to Azure Storage services such as queues, blobs, and tables.

You can configure both connection strings to use the same storage account, and some of the documentation you will see online does this out of convenience. This is also an acceptable practice for Dev/Test environments. But it is a recommended best practice to use separate storage accounts to keep application data separate from logging data. Doing so ensures that the entire capacity and throughput limits of your storage account are dedicated to your application and not used for logging purposes. This can be especially important for high-volume applications running in production.

See Also For information on Azure Storage capacity and limits, see <http://azure.microsoft.com/en-us/documentation/articles/azure-subscription-service-limits/>.

We will configure the `AzureWebJobsDashboard` and `AzureWebJobsStorage` connection strings to use the same storage account created for the web app in Chapter 1, "Microsoft Azure Web Apps." To get this value, expand the Storage node in Server Explorer and locate the storage account created for the web app. Right-click the storage account and select **Properties**. In the Properties window, click the **ellipsis** button in the Connection String property, as shown in Figure 2-11.

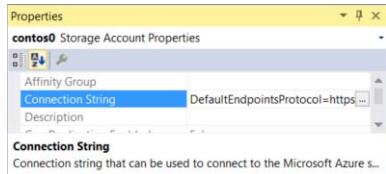


FIGURE 2-11 Storage account Properties window in Visual Studio.

In the Storage Account Connection String dialog, click the icon to copy the full connection string to your clipboard and then click **Close**. In app.config, paste the connection string into the connectionString setting for the AzureWebJobsDashboard and AzureWebJobsStorage connection strings.

Press **Ctrl+F5** to run the web job locally. The output will look similar to Figure 2-12.

```
C:\Windows\system32\cmd.exe
Creating Demo data
Functions will store logs in the 'azure-webjobs-hosts' container in the specified Azure storage account. The functions take in a TextWriter parameter for logging.
9.
Found the following functions:
SampleQueueWebJob.Functions.MultipleOutput
SampleQueueWebJob.Functions.QueueToBlob
SampleQueueWebJob.Functions.PropertyBinding
SampleQueueWebJob.Functions.FailAlways
SampleQueueWebJob.Functions.BindToPoisonQueue
Job host started
Executing: 'Functions.PropertyBinding' because New queue message detected on 'initialorderproperty'.
Executing: 'Functions.MultipleOutput' because New queue message detected on 'initialorder'.
Executing: 'Functions.QueueToBlob' because New queue message detected on 'orders'.
```

FIGURE 2-12 WebJobs SDK Queue sample application output.

One thing to notice about the output from the web job is that methods in the web job were *found* and listed by name as functions in the web job. The web job was able to identify these as a result of certain attributes from the WebJobs SDK being applied to these methods. Also notice that three of these methods were invoked as a result of a message being added to a queue. That is because the template used to create the project includes some code that generates sample data as part of the application so you can see how things work. Finally, notice that this application is still running. Web jobs developed using the WebJobs SDK frequently run continuously so that functions can be invoked when messages (in this case) are added to queues in Azure Storage. How this all works will become clear as we proceed through the rest of this chapter.

Press **Ctrl+C** to exit the application.

Examine the web job project and code

The general organization of project files that the WebJobs SDK project templates follow is one in which functions of the web job are contained in a Functions class in the file Functions.cs. This file also may include other classes used for sample data, such as the Order class, which is used to generate some data the sample code uses. You can extend these classes to fit your needs or remove them from your

project. Their purpose is to demonstrate how the WebJobs SDK could be used. The rest of the project is what you would expect to see for a simple console application, as shown in Figure 2-13.

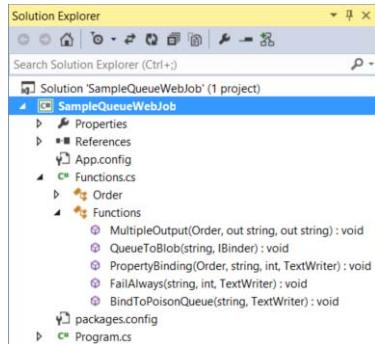


FIGURE 2-13 Web job project files in Solution Explorer.

Understanding the JobHost class

The workhorse of the WebJobs SDK is the **JobHost** class. At runtime, it uses reflection to locate methods in your web job that have been decorated with certain attributes from the WebJobs SDK. It uses the connection strings you provided earlier to connect to your Azure Storage account and listen for new messages being added to specified Azure Storage queues. When a new message is detected, it invokes the function that is designated to receive messages from the queue. All of this is delivered to your web job application with just two lines of code. If you open Program.cs and look in the **Main** method of the application, you will see where **JobHost** is instantiated, followed by a single call to **RunAndBlock**, as shown in Listing 2-2.

LISTING 2-2 Use of JobHost class in Main method.

```
static void Main()
{
    if (!VerifyConfiguration())
    {
        Console.ReadLine();
        return;
    }

    CreateDemoData();

    JobHost host = new JobHost();
    host.RunAndBlock();
}
```

The **RunAndBlock** method blocks the main thread of the application so that the process is kept running. The **JobHost** instance will invoke your web job functions on background worker threads.

If you wanted to make changes to some of the default configuration the JobHost class uses, you could instantiate an instance of the JobHostConfiguration class and define configuration such as connection string settings. Then, you could pass the JobHostConfiguration instance into the JobHost constructor.

If you need to control when the JobHost monitors your storage account for changes, you can use the Start/StartAsync and Stop/StopAsync methods.

If you want to manually invoke a function in your web job, you can use one of the Call/CallAsync methods. If you want to see how this works, create a new WebJob project using the Azure WebJobs SDK Tables QuickStart template.

Understanding the WebJobs SDK attributes

The WebJobs SDK provides a QueueTrigger attribute that you can use in your methods to indicate that you want that method invoked by the JobHost when a new message is added to a queue. Similarly, there is a BlobTrigger attribute and a ServiceBusTrigger attribute that can be used for those services.

Keeping with the queue-centric theme, suppose you have a queue named "neworders" and a method in your web job named "ProcessNewOrder" that you want called when a new message arrives in the neworders queue. This could be accomplished using the QueueTrigger attribute, as shown in Listing 2-3.

LISTING 2-3 Process orders from a queue.

```
public static void ProcessNewOrder([QueueTrigger("neworders")] Order newOrder)
{
    //
    // Do work to process the order.
    //
}
```

The QueueTrigger attribute takes as a parameter the name of the queue you want JobHost to monitor for new messages. The attribute is applied to the newOrder parameter so that when a message (Order) is added to the neworders queue, this method will be invoked and the newOrder parameter will contain the order that was added to the queue. In other words, you don't have to write any code to de-queue the message. The JobHost takes care of it. When your function completes, the JobHost will make sure the message is deleted from the queue.

Another powerful attribute is the Queue attribute, which can be used to bind a method parameter to a queue. Expanding on the previous example, suppose you have another queue named "processedorders" that you use to store messages for orders that have been processed successfully. Rather than writing code to insert a message into the processedorders queue, you can add an *out* parameter to the ProcessNewOrder method and use the Queue attribute to bind that parameter to the processedorders queue. Listing 2-4 is an example of how this could be achieved.

LISTING 2-4 Process orders from a queue and output to a separate queue.

```
public static void ProcessNewOrder(
    [QueueTrigger("neworders")] Order newOrder,
    [Queue("processedorders")] out string processedorders)
{
    //
    // Do work to process the order.
    //
    processedorders = newOrder.OrderId;
}
```

In this code, after the newOrder is processed, a message (this time a simple string) is added to the processedorders queue that contains the order ID of the order. You then could have another function that is invoked when messages are added to the processedorders queue to do any final work needed, such as updating a database or sending out email to the customer.

Similar to the Queue attribute, there are Blob, Table, and ServiceBus attributes that can be used to bind a parameter to those services.

The code in Listing 2-3 and Listing 2-4 is a simplified example only and is not part of the web job project created by the QuickStart templates. To see how the code generated by the QuickStart template uses these attributes, open Functions.cs and explore the methods in the Functions class.

Understanding how logging works

As you explore code in Functions.cs, you will notice that some of the methods take a `TextWriter` parameter. This parameter is optional. But if you want to do any logging in your functions and have the logging saved in log files that can be accessed later, this is the way to do it.

The `JobHost` creates an instance of `TextWriter` for you. All you have to do is ask for it by adding a parameter of this type to your method. Listing 2-5 demonstrates how logging in your function could be achieved by using `TextWriter`.

LISTING 2-5 Process orders and add logging capabilities by using `TextWriter`.

```
public static void ProcessNewOrder(
    [QueueTrigger("neworders")] Order newOrder,
    [Queue("processedorders")] out string processedorders,
    TextWriter log)
{
    log.WriteLine("Started processing order Id '{0}'.", newOrder.OrderId);
    //
    // Do work to process the order.
    //
    log.WriteLine("Completed processing order Id '{0}'.", newOrder.OrderId);

    processedorders = newOrder.OrderId;
```

}

This section covered some essential knowledge to introduce you to the power of the WebJobs SDK and get you started using essential features in your code. You are encouraged to fully explore the functions in Functions.cs to further understand the capabilities available to you.

Publish a web job to Azure

When you move your web job to Azure, you will need to make sure the AzureWebJobsDashboard and AzureWebJobsStorage connection strings are in your *web app* environment. You can accomplish this by adding the connection strings from app.config in your web job project to your web app environment as environment variables. This adheres to the best practice guidance for storing connection strings and will be demonstrated in the following section.

See Also More information on recommended best practices for working with connection strings and other sensitive information is available at <http://www.asp.net/identity/overview/features-api/best-practices-for-deploying-passwords-and-other-sensitive-data-to-aspnet-and-azure>.

Add storage account connection strings to the web app environment

In Server Explorer, expand the App Service node and then the resource group for the web app created in Chapter 1. Right-click the web app and select **View Settings**.

In the Connection Strings section, add the AzureWebJobsDashboard and AzureWebJobsStorage connection strings and set the Database Type to **Custom** for both. Your Connection Strings settings will look similar to Figure 2-14.

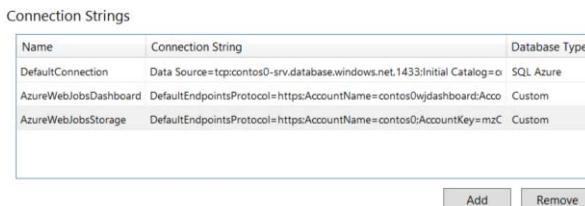


FIGURE 2-14 Configuring connection strings for a web app using Visual Studio.

Press **Ctrl+S** to save the changes to the web app environment.

Publish the web job to Azure

In the Visual Studio Solution Explorer window, right-click the web job project and select **Publish as Azure WebJob**.

In the Add Azure WebJob dialog, set the WebJob run mode to **Run Continuously** and click **OK** to continue to the next page.

Tip Web jobs built using the WebJobs SDK frequently run continuously. However, web apps may unload if they are idle for extended periods of time, which also would cause your web job to be unloaded. It is recommended that you enable the Always On feature for web apps in the Basic or Standard pricing tier to ensure reliable execution of your functions.

In the Publish Web dialog, click **Microsoft Azure Web Apps** and set the Existing Web Apps setting to the web app created in Chapter 1.

Click **Publish** to publish the web job. After a few seconds, the web job will be published to the web app environment.

In the Visual Studio Server Explorer window, right-click the **WebJobs** node for the web app and select **Refresh**. The web job will be under the Continuous folder, as shown in Figure 2-15.

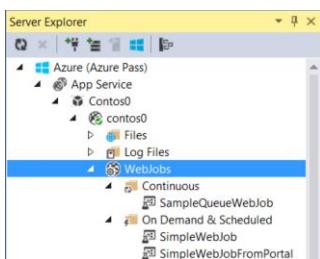


FIGURE 2-15 Server Explorer showing web jobs published to a web app.

Examine new features in the WebJobs Dashboard

In the Visual Studio Server Explorer window, right-click the **SampleQueueWebJob** and select **View Dashboard**. Because this web job is using the WebJobs SDK, the dashboard is able to show you which functions have been invoked, as shown in Figure 2-16.

A screenshot of the Microsoft Azure WebJobs Dashboard. The top navigation bar says 'Microsoft Azure WebJobs'. Below it is a sub-navigation bar with 'WebJobs' selected. The main content area is titled 'Continuous WebJob Details SampleQueueWebJob'. It shows the status as 'Running' with a 'Run command: ComputeWebJobsSDKQueue.exe'. There is a 'Toggle Output' button. A table titled 'Functions invoked' lists three entries: 'Functions.QueueToBlob (3e0f2debc7924518ad ...)' with a 'Success' status and '5 minutes ago (234 ms running time)'; 'Functions.MultipleOutput ([{"Name": "Alex", "Or ...})' with a 'Success' status and '5 minutes ago (424 ms running time)'; and 'Functions.PropertyBinding ([{"Name": "Alex", "Or ...})' with a 'Success' status and '5 minutes ago (188 ms running time)'. All three rows have a green success icon.

FIGURE 2-16 WebJobs Dashboard showing functions that have been invoked.

Each function invocation is linkable so that you can drill down into the details for that specific invocation. Locate the function invocation for the PropertyBinding function and click it. The invocation details show you why this function was invoked and the parameters that were passed to the function. Click the **Toggle Output** button at the bottom of the page to see the logging that was captured when the function was run. The Invocation Details page will look similar to Figure 2-17.

The screenshot shows the Microsoft Azure WebJobs Invocation Details page. At the top, there's a navigation bar with 'Microsoft Azure WebJobs' and links for 'Functions', 'Search Blobs', and 'About'. Below the navigation, the URL is shown as 'WebJobs / SampleQueueWebJob / Functions.PropertyBinding'. The main content area has a title 'Invocation Details' followed by the function name 'Functions.PropertyBinding ({"Name": "Alex", "Or ...})'. A blue 'Replay Function' button is visible. A green box indicates a success message: 'Success 17 minutes ago (188 ms running time)'. Below this, a note says 'New queue message detected on 'initialorderproperty''. A table lists parameters and their values:

PARAMETER	VALUE	NOTES
initialorder	{"Name": "Alex", "OrderId": "3e0f2debc7924518adbacd75f2f7c6ee"}	
Name	Alex	
dequeueCount	2	
log		

At the bottom, another blue 'Toggle Output' button is present. A log entry shows: 'download New order from: Alex Message dequeued 2 times'. A note at the bottom right says 'New queue message detected on 'initialorderproperty''.

FIGURE 2-17 Invocation details for a web job function.

A powerful feature in the dashboard made possible by the WebJobs SDK is the ability to replay a function with the same or different values. Click the **Replay Function** button near the top of the page. This will open a new page that shows the original values that were passed to the function. In this page, you have an opportunity to edit the values before replaying (invoking) the function. For example, change the initialorder field (the queue message) so that the Name property is "JohnDoe" and the OrderId property is "1010101." Set the Name to **JohnDoe**, as shown in Figure 2-18.

Functions

/ Functions.PropertyBinding

Replay Function Functions.PropertyBinding**initialOrder** Enter the queue message body

>{"Name":"JohnDoe","OrderId":1010101}

Name Enter a value

JohnDoe

dequeueCount Enter a value

2

Replay**FIGURE 2-18** Replying a function invocation with different values.Click **Replay** to replay the function with these values.

You can view the invocation log for this and other recent function invocations from the Functions page. To get to the Functions page, click **Functions** in the menu at the top of the page. This will show you some statistics for each function in the web job, such as the number of times the function was successful or failed, and an invocation log of recent function invocations. The items listed in both sections of this page also are linkable, so you can drill down into them for more details. Figure 2-19 shows what this page looks like for this web job. Notice the invocation log at the top of the Invocation Log section for the PropertyBinding function that was replayed earlier.

Functions		Invocation Log Recently executed functions		
FUNCTION NAME	STATISTICS	FUNCTION	STATUS	STATUS DETAIL
Functions.BindToPoisonQueue	9 0	Functions.PropertyBinding >{"Name": "JohnDoe", ...}	Success	9 minutes ago (228 ms running time)
Functions.FailAlways	9 0	Functions.QueueToBlob {...}	Success	16 minutes ago (234 ms running time)
Functions.MultipleOutput	2 0	Functions.MultipleOutput {"Name": "Alice", "Or ...}	Success	36 minutes ago (434 ms running time)
Functions.PropertyBinding	4 5	Functions.PropertyBinding >{"Name": "Alice", "Or ...}	Success	36 minutes ago (188 ms running time)
Functions.QueueToBlob	2 0			

FIGURE 2-19 Functions page of WebJobs Dashboard.

See Also An excellent narrative on how Azure WebJobs were used recently in a production application and the benefits gained from this feature is available at <http://www.troyhunt.com/2015/01/azure-webjobs-are-awesome-and-you.html>.

Summary

In this chapter, we discussed how WebJobs can be used to address background processing needs for an Azure web app. We started by creating a simple .NET console application and then published it to Azure to demonstrate how web jobs and the WebJobs Dashboard work at their most primitive level. Then, we introduced the WebJobs SDK and learned how common messaging patterns between a web app and a web job can be achieved with just a few lines of code and a couple of well-placed attributes. We then revisited the WebJobs Dashboard and explored new features available to web jobs developed using the WebJobs SDK.

This chapter covered essential knowledge developers need to get started building Azure WebJobs. More information, including sample applications, tutorials, blogs, videos, and scenarios you can implement using WebJobs is available at <http://azure.microsoft.com/en-us/documentation/articles/websites-webjobs-resources/>.

Chapter 3

Scaling Azure Web Apps

Azure Web Apps offers flexibility and features that you can use to scale your web app to handle the load your users place on it. Whether your application needs to handle a few hundred requests per day or a few million requests per day, the Azure Web Apps scalability features provide ways for you to deliver the right level of scale in a robust, cost-effective manner.

When you consider the scalability requirements of an application, you should look at its resource requirements vertically (scaling up) *and* horizontally (scaling out). When you *scale up* a web app, you increase the resource capacity, such as RAM and CPU cores, of the virtual machine on which your web app is running. This is an important concept whether you are running in the cloud or on-premises. It ensures that your application has the resources it needs to perform properly. In other words, you rightsize the machine resources for the application.

When you *scale out* a web app, you increase the number of virtual machine instances on which your web app is running. For the properly architected app, this means your web app can handle more load and therefore service more user requests. But an effective scale out strategy involves more than just increasing virtual machine instances. There are a few things you should consider carefully, which will be discussed shortly. Furthermore, by leveraging the breadth of scalability features that Azure Web Apps offers, you can achieve high availability and elastic scale that is common for many modern cloud applications.

This chapter will discuss essential knowledge you can apply to scale your web apps.

Scale Up

The ability to scale up a web app exists only for web apps configured for Basic, Standard, or Premium pricing tiers. These tiers give you a dedicated virtual machine instance for your web app and therefore some choices for the capacity of the virtual machine. Load testing your app will give you an idea of your application's requirements for resources such as CPU and RAM. Knowing this will help you determine an appropriate pricing tier for your application.

The compute capacity of Virtual Machines in the different pricing tiers is shown in Table 3-1 and includes the number of cores and amount of RAM.

TABLE 3-1 Virtual Machine core and RAM capacity in each pricing tier

Premium (P1) Standard (S1) Basic (B1)	Premium (P2) Standard (S2) Basic (B2)	Premium (P3) Standard (S3) Basic (B3)	Premium (P4)
1 Core	2 Cores	4 Cores	8 Cores
1.75 GB RAM	3.5 GB RAM	7 GB RAM	14 GB RAM

Similarly, the capacity for storage and web socket connections available in the different pricing tiers is shown in Table 3-2.

TABLE 3-2 Virtual Machine storage and connection capacity in each pricing tier

Basic (B1, B2, B3)	Standard (S1, S2, S3)	Premium (P1, P2, P3)	Premium (P4)
10 GB storage	50 GB storage	250 GB storage	500 GB storage

Note These tables do not represent the full matrix of features available in each pricing tier. Instead, they identify core resource limits that you may take into consideration when deciding how to scale your web app vertically. The complete list of limits for Web Apps is available at <http://azure.microsoft.com/en-us/documentation/articles/azure-subscription-service-limits/#web-apps-websites-limits>.

As an example, if you load test your app with anticipated load and determine you need a Virtual Machine with two cores and more than 2 GB of RAM to meet your response goals, then you should scale your web app to a Standard (S2) pricing tier. To do this using the Azure portal, from the Web App blade click the **Pricing tier** web part in the Usage section. In the Pricing Tier blade, select the **Standard S2** plan. Click **Select** to change the web app to the new pricing tier. The Pricing tier web part in the Web App blade will look similar to Figure 3-1.

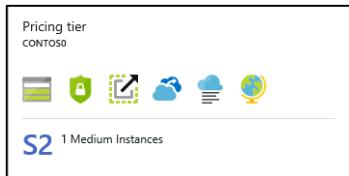


FIGURE 3-1 The Pricing tier web part in the Web App blade.

See Also At the time of this writing, Microsoft announced the App Service Environment (ASE) in preview, which is a Premium service plan offering additional security and scalability options. The Premium (P4) capacities listed in Tables 3-1 and 3-2 are specific to the App Service Environment. You can learn more about this at <http://azure.microsoft.com/en-us/documentation/articles/app-service-app-service-environment-intro/>.

Scale Out

To scale out is to increase the number of virtual machine instances on which your web app is running. The number of instances you can scale out is limited by the pricing tier configured for your web app. Table 3-3 shows the number of instances you can scale out to in the various pricing tiers.

TABLE 3-3 Maximum Virtual Machine instances available by pricing tier

Free & Shared	Basic	Standard	Premium	Premium w/ASE
1 Shared	3 Dedicated	10 Dedicated	20 Dedicated	50 Dedicated

You can scale out the number of instances manually by using the Azure portal. From the Web App blade, click the **Scale** web part in the Usage section. In the Scale setting blade, set **Scale by** to **an instance count that I enter manually**. Use the slider control to change the **Instances** value. Or type the number of instances (up to the maximum allowed) in the box to the right of the slider control. For example, to set the number of Standard instances to 5, your Scale setting blade will look similar to Figure 3-2.

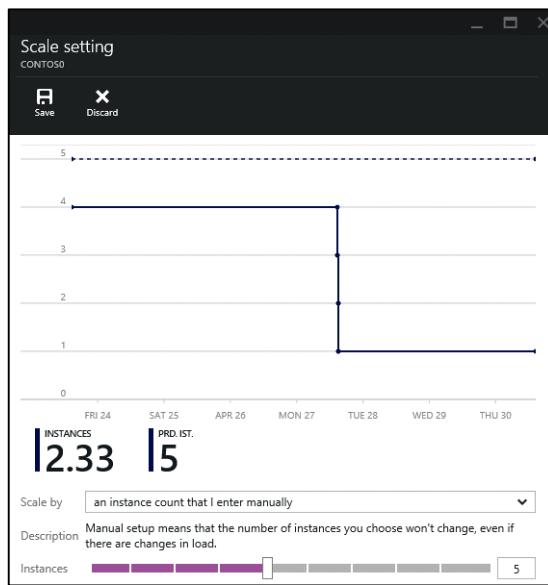


FIGURE 3-2 Manually setting number of instances in the Scale setting blade.

Dealing with the challenges of scaling out a web app

Scaling an app typically requires scaling out other resources the app depends on, and for most apps, the database is the resource that can be the greatest bottleneck. For example, a web app with a database may perform fine when there is just one instance of the web app. If you increase the

instances of your web app, you put more load on the database. To support the extra demand from the web app instances, you will need to consider scaling up and possibly scaling out your database. If you are using Azure SQL Database, there is very good documentation on how to assess performance of your SQL Database and how to scale the database at <https://msdn.microsoft.com/en-us/library/azure/dn741338.aspx>.

See Also To learn more about load testing and how you can use Visual Studio Online to load test your application, see <https://www.visualstudio.com/get-started/test/load-test-your-app-vs>.

Many web applications use session state, which can create a problem because the default in-process session state provider requires sticky sessions. Sticky sessions ensure that a user's requests are routed back to the same server instance where the session was created. Unfortunately, this results in less-than-optimal load balancing of traffic over time between the instances. Adding a distributed session cache is an effective way to deal with this challenge. The Azure Redis Cache is the recommended cache service in Azure. It is a distributed, in-memory cache delivered as a service, and Microsoft provides an ASP.NET Session State Provider for use with the Redis Cache.

The combination of caching and properly scaling your database can help you achieve massive scalability for your applications. For an example that shows how to use the Azure Redis Cache with a SQL database and use the ASP.NET Session State Provider, see

<http://azure.microsoft.com/blog/2014/06/05/mvc-movie-app-with-azure-redis-cache-in-15-minutes>.

See Also Azure Web Apps assumes by default that your web app uses sessions. It does this by using the Application Request Routing (ARR) feature in IIS and placing an 'ARRAffinity' cookie in the user's browser session. As requests arrive, an ARR front-end processes the cookie and then routes the request back to the server instance that is associated with the cookie. This ensures that the user is routed to the same instance when multiple instances are available.

If your application uses a distributed cache (such as Redis) or is completely stateless, then you can get better load balancing by disabling instance affinity. For more information on how Azure Web Apps implements this and how to disable it, see <http://azure.microsoft.com/blog/2013/11/18/disabling-arrs-instance-affinity-in-windows-azure-web-sites/>.

If your web app uses Microsoft Azure Storage in any way, then scaling out your web app will introduce bottlenecks if your web app instances start to exceed your storage account scalability targets. In this scenario, you may consider adding storage accounts and restructuring your code to use different storage accounts for different parts of the application. Or you may be able to leverage caching to solve the problem. More information about Azure Storage account limits is available at <http://azure.microsoft.com/en-us/documentation/articles/storage-scalability-targets/>.

There are many other resources in Azure that your web app could have dependencies on. This is not intended to be a complete list. The main takeaway from this section is that you realize that although scaling out your web app is achieved easily through a few simple clicks in the Azure portal, you first should give careful thought to the impact this could have on other resources in your resource group. Not doing so could potentially degrade the performance of your application.

Scaling web apps using Autoscale

Autoscale is a feature in the Azure platform that enables you to automatically scale out (and in) your cloud applications. This can deliver substantial cost savings on your cloud computing while giving you flexibility to scale out to capacity levels to support increased demand from end users.

The Autoscale feature is not unique to Azure Web Apps. It also is available to Cloud Services and Virtual Machines. Although there are some differences in how Autoscale is configured for Web Apps compared to other compute services, for the most part the concepts you will learn here can be applied to other compute services.

Autoscale lets you scale out your web app instances based on predefined schedules, metrics such as CPU, or both. To use Autoscale, you must be in a Basic pricing tier or higher, and you are limited to scaling out to the maximum number of instances available for your pricing tier, as mentioned earlier in Table 3-3.

Note Autoscale is a feature you can use to scale in/out your web app. You cannot use it to scale up/down your web app.

Autoscale based on CPU percentage

Autoscale may be configured in the **Scale setting** blade of the Azure portal, which you can access from the Web App blade by clicking the Scale web part in the Usage section.

To configure Autoscale to scale by CPU percentage, change the Scale by dropdown to **CPU Percentage**. As a result, the Instances setting will become a number range instead of a single number. You also will see a Target range setting appear under the Instances setting.

The Instances setting is used to define the minimum and maximum number of instances on which you want your web app running. As the CPU percentage exceeds a certain threshold, Autoscale will increase the number of instances up to the maximum instances you specify. As the CPU percentage falls below a certain threshold, Autoscale will decrease the number of instances down to the minimum instances you specify. Set Instances to be in the range of **3** and **8**.

The Target range setting defines the minimum and maximum CPU percentage to target. As long as the CPU percentage is within this range, Autoscale will not increase or decrease the number of instances. When the CPU percentage exceeds the maximum CPU percentage you specify, Autoscale will add an instance. If CPU percentage continues to exceed the maximum CPU specified, then Autoscale will add another instance. At no point will you have more than the maximum number of instances specified in the Instances setting. Similarly, when CPU percentage falls below the minimum CPU percentage you specify, Autoscale will remove an instance. If CPU percentage continues to fall below the minimum CPU percentage specified, then Autoscale will remove another instance. At no point will

you have fewer than the minimum number of instances specified in the Instances setting. Set the Target range to be between **20** and **80** percent. The bottom of the Scale setting blade will look similar to Figure 3-3.

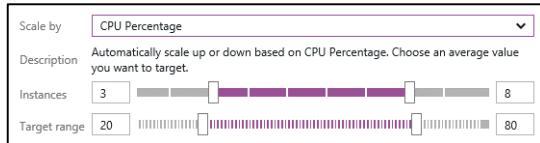


FIGURE 3-3 Scaling by CPU percentage in the Scale setting blade.

Click **Save** in the toolbar at the top of the Scale setting blade.

Note The CPU percentage is measured as an average across all instances. For example, if you have two instances, one of which is running at 50 percent CPU and the other of which is running at 100 percent CPU, then the CPU percentage would be 75 percent for all the instances at that point in time. Autoscale uses this value to determine when to scale instances up or down.

After several minutes, you will notice the number of instances step down to 3. The default configuration will remove an instance every five minutes until the instance count reaches the minimum instances configured for the range. Later, you will see where this can be changed.

Close the Scale setting blade and scroll down to the Usage section. The Scale web part should indicate that Autoscale is On and the Instances are 3, as shown in Figure 3-4.

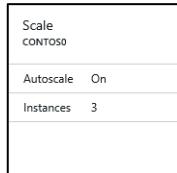


FIGURE 3-4 Scale web part in the Web App blade.

Autoscale based on a recurring schedule

Autoscale also can be configured based on a schedule. This can be particularly useful when demand for your web app is predictable. For example, if your web app provides services for an industry where most work is done Monday through Friday, then you could configure Autoscale to increase the number of instances during the week to support peak demand and decrease the number of instances on weekends when demand is very light.

You also can apply a schedule to a performance-based configuration, such as the CPU percentage metric discussed in the previous section. This gives you the flexibility of scaling out dynamically to support increasing traffic only during a time period you specify. To demonstrate, this concept, we'll

extend the Autoscale configuration based on CPU percentage to be bound to a particular time period.

In the Scale setting blade, change Scale by to **schedule and performance rules**. This will change the content in the blade so that the Settings section shows the Autoscale *profiles* and *rules*, as shown in Figure 3-5.

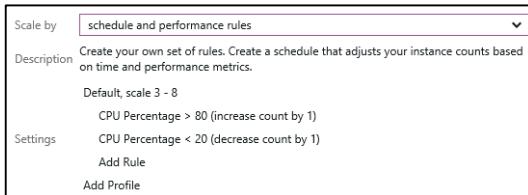


FIGURE 3-5 Configuring Autoscale in the Scale setting blade based on schedule and performance rules.

In Figure 3-5, there is one profile shown, which is identified as **Default, scale 3 – 8**. Under the profile are two rules summarizing the CPU percentage minimum and maximum configurations. Click the profile **Default, scale 3 – 8**. This will open the Scale profile blade, where you can rename the profile, change the target range, and apply a schedule by changing the Type. The Scale profile blade will look similar to Figure 3-6.

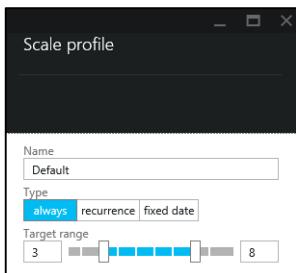


FIGURE 3-6 Scale profile blade in the Azure portal.

Set Type to **recurrence**.

Click the dropdown for Days and uncheck Saturday and Sunday, leaving Monday through Friday checked. The Scale profile blade will now look similar to Figure 3-7.

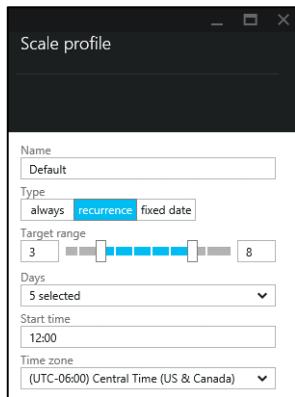


FIGURE 3-7 Scale profile blade with a recurring date configuration for weekdays.

Click **OK** in the Scale profile blade.

Click **Save** in the toolbar at the top of the Scale setting blade.

Now, the Autoscale based on CPU percentage will be in effect only Monday through Friday.

Note In this discussion, we are using a recurring schedule to demonstrate the scheduling features in Autoscale. You also could have selected a fixed date for the type. This would allow you to specify a specific starting date and time, and ending date and time for the profile.

Suppose now that you want to scale down the number of instances to just one on Saturday and Sunday. To achieve this, you would add a profile to the Autoscale configuration. To demonstrate this, click **Add Profile** in the Settings section of the Scale setting blade. Configure the Scale profile blade as follows:

Set Name to **Weekend**.

Set Type to **recurrence**.

Set the minimum and maximum of the Target range to **1**.

In the Days dropdown, check only **Saturday** and **Sunday**.

The Scale profile blade will look similar to Figure 3-8.

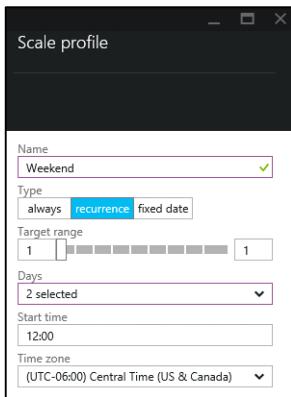


FIGURE 3-8 Scale profile blade with a recurring date configuration for weekends.

Click **OK** in the Scale profile blade.

Click **Save** in the toolbar at the top of the Scale setting blade.

You now have two profiles defined for Autoscale: one that will scale between 3 and 8 instances based on CPU percentage during the week and another that will scale down to 1 instance on the weekends. The Scale setting blade will look similar to Figure 3-9.

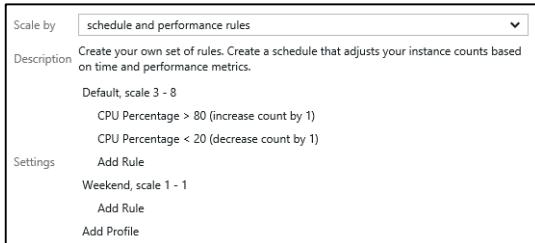


FIGURE 3-9 Configuring Autoscale in the Scale setting blade with two profiles.

Tip It is not possible to configure Autoscale to scale down to 0 instances. If you want to shut down your web app completely, then you could use the `Stop-AzureWebsite` command from the Azure PowerShell module. You also could automate this on a schedule by creating a Runbook and using Azure Automation, as explained at <http://stackoverflow.com/questions/25929339/how-do-you-turn-an-azure-website-on-and-off-on-a-schedule>.

Understanding Autoscale rules

Earlier, you saw how to configure Autoscale based on CPU percentage, and if you were patient, you noticed the number of instances decrease. This was based on default settings for the CPU Percentage scale rule. You can change these behaviors by editing the Autoscale rule. As an example, if you wanted to be more aggressive about scaling out your web app instances, you could have Autoscale add two

instances (or more) when it takes action to scale out your instances.

To demonstrate this, click the **CPU Percentage > 80 (increase count by 1)** rule in the Scale setting blade. In the Scale rule blade, change the Value to **2**, as shown in Figure 3-10.

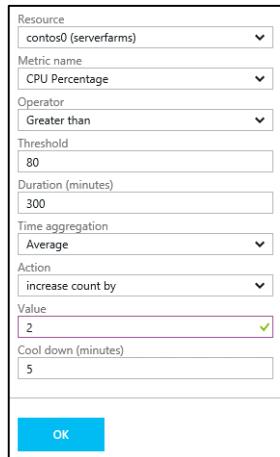


FIGURE 3-10 Scale rule blade in the Azure portal.

Click **OK** to save the Scale rule. Now, if the CPU percentage were to exceed 80 percent, then Autoscale will add two instances at a time up to the maximum number of instances you specified for the instances range.

Additional metrics by which to scale

The Scale rule blade gives you an opportunity to specify metrics by which to scale other than CPU percentage. In Figure 3-10, notice the Metric name field is a dropdown. You can select a different metric when editing an existing scale rule or adding a new one. The available metrics are as follows:

- CPU Percentage
- Memory Percentage
- Disk Queue Length
- Http Queue Length
- Data In
- Data Out

Another option for configuring Autoscale rules is to scale by queue depth for a queue in your Azure Storage account. To do this, in the Scale rule blade, set Resource to **Storage Queue**. This will add an option where you can select an existing Azure Storage queue. Set the Metric name to **Message Count**.

The Threshold field in this scenario defines the queue depth for which you want an Autoscale action triggered. For example, if you want instances added when your queue depth exceeds 125 messages, then you could set this to 125, and Autoscale will add the number of instances indicated in the Value field. A scale rule configuration such as this would look similar to Figure 3-11.

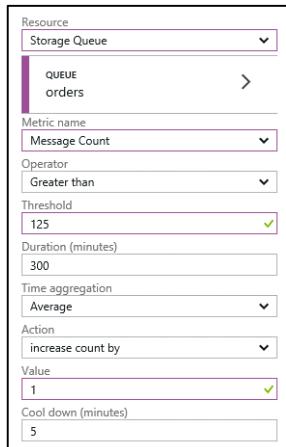


FIGURE 3-11 Scale rule blade configured to scale by queue depth in an Azure Storage queue.

You have a great amount of flexibility in how you scale your web apps and the behavior of Autoscale actions. This section introduced you to some common scaling strategies you can apply in your web apps. For more information on the properties that can be set for profiles and rules, consider looking at the REST API documentation for Autoscale at <https://msdn.microsoft.com/en-us/library/azure/dn510372.aspx>.

Turn off Autoscale

Regardless of how Autoscale has been configured, it can be turned off in the Scale setting blade by setting Scale by to **an instance count that I enter manually**. Next, set Instances to the number of instances you want and click **Save** in the toolbar at the top of the blade.

Scale globally with Azure Traffic Manager

The scalability discussion until now has been limited to scalability within a single deployment. For example, you have essential knowledge to scale your web app that is deployed in a single region, such as the Central US region. If your users are mostly in the United States and the peak demand for your web app is within the capacity of this single deployment, then you may have achieved your scalability utopia. But for applications that need to scale beyond the capacity of a single deployment or whose users are globally dispersed, additional scalability techniques should be considered.

Deploying your web app to multiple regions (or datacenters) is a scale-out strategy that can be used to achieve massive scalability for your web app. Assume, for example, that you have a web app deployment in the Central US region. If your users are dispersed around the world, then you may choose to deploy to the West US, East US, and North Europe regions as well. Doing so will significantly increase capacity of your web app. But this introduces some new challenges such as how users are routed to the different deployments and how the database is synchronized between deployments. Fortunately, Microsoft Azure provides services and features to achieve this level of scale and address the challenge of data synchronization between deployments.

The challenge of routing users to one of many web app deployments can be met by using Azure Traffic Manager. This is a networking service that can be used to achieve global scale for your web apps by allowing you to control how user traffic is routed to multiple deployments of your application. This service is not unique to Azure Web Apps. It can be used to direct traffic to deployments of Cloud Services, Web Apps, and even external HTTP/S endpoints. To use Azure Traffic Manager with your web apps, your web apps must be running in the Standard (or higher) pricing tier.

To use Traffic Manager, you first must create a Traffic Manager profile, which identifies a unique DNS name for the profile in the `*.trafficmanager.net` domain, a list of *endpoints* (web app deployments) that you specify, and a *load balancing method*, which can be one of the following:

- **Performance** This method will route users to the deployment endpoint that has the *lowest network latency* for the user. For example, if your web app is deployed in the West US, Central US, and East US regions and a user is located geographically somewhere in the eastern United States, then this method likely will resolve the user to the East US deployment.
- **Round Robin** This method will distribute users to endpoints in a round-robin fashion so that traffic is distributed evenly across all the endpoints.
- **Failover** This method is used to identify a primary endpoint to be used for all traffic. If the endpoint becomes unavailable, users are routed to the next endpoint listed in the profile. When the primary endpoint becomes available again, Traffic Manager will continue routing users to the primary endpoint.

See Also To learn more about how the Traffic Manager load balancing methods work, see <http://azure.microsoft.com/en-us/documentation/articles/traffic-manager-load-balancing-methods/>.

The DNS name for the profile is referenced in your DNS server as a CNAME record. For example, suppose your custom domain is contoso.com and you have a Traffic Manager profile with the DNS name contoso.trafficmanager.net. To use the Traffic Manager profile for your deployments, you would create a CNAME record that maps www.contoso.com to contoso.trafficmanager.net. When users make a request to www.contoso.com, a DNS query is invoked. The CNAME record results in the DNS query being forwarded to contoso.trafficmanager.net, where your Traffic Manager profile is defined. Based on the load balancing method you have configured for your profile, Traffic Manager selects an endpoint and returns the endpoint's DNS name, such as contoso-west-us.azurewebsites.net. The user's

DNS server resolves contoso-west-us.azurewebsites.net to an IP address and returns it to the user, where an HTTP request is then made against the web app endpoint.

Note Traffic Manager essentially is an extension of the DNS query process that occurs when users access web endpoints. It is *not* a device through which all traffic is routed, so no overhead is incurred beyond the initial DNS query.

Create a Traffic Manager profile

At the time of this writing, the Azure Traffic Manager was not available in the Azure portal, so this section will use the old portal at <https://manage.windowsazure.com>.

To create a new profile, select **NEW > NETWORK SERVICES > TRAFFIC MANAGER > QUICK CREATE**. Enter a unique DNS prefix name and set LOAD BALANCE METHOD to **Performance**, as shown in Figure 3-12.

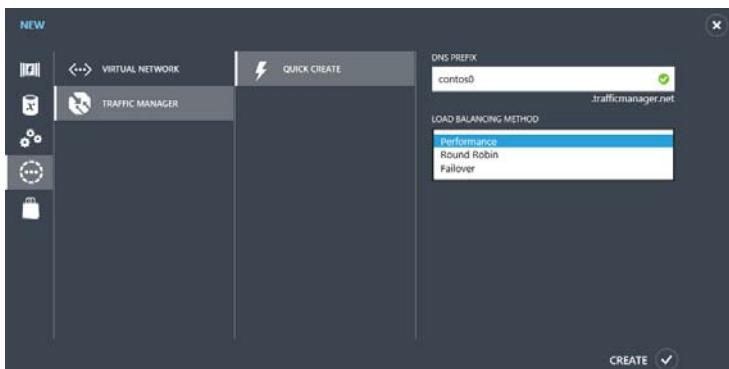


FIGURE 3-12 Create a new Traffic Manager profile in the Azure Management Portal.

Click **CREATE** to create the profile.

Add endpoints to the profile

After the Traffic Manager profile is created, click the **profile name** in the Traffic Manager page of the portal. Next, click the **ENDPOINTS** tab at the top of the page.

Note The steps demonstrated in this section assume the Contoso web app already has been published to three web app environments, as follows: contoso0.azurewebsites.net, contoso0-west.azurewebsites.net, and contoso0-east.azurewebsites.net.

In the ENDPOINTS page, click the **ADD** button at the bottom of the page.

In the ADD SERVICE ENDPOINTS dialog, set SERVICE TYPE to **Web app**.

In the Service Endpoints section, the dialog will list the available web app endpoints, and you can check the ones you want to add to the Traffic Manager profile. Select the check box for each endpoint you want to add. The ADD SERVICE ENDPOINTS dialog will look similar to Figure 3-13.

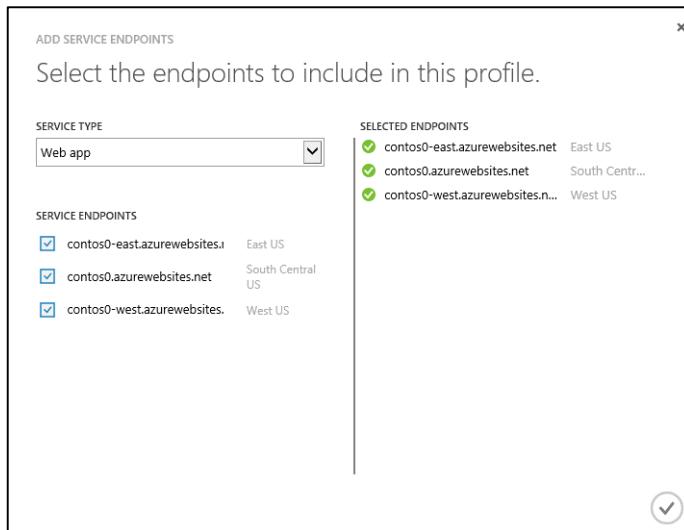


FIGURE 3-13 Adding web app endpoints to a Traffic Manager profile.

Click the check mark in the lower-right corner of the dialog to add the endpoints to the profile.

Note As the update to the profile is being made, Azure Traffic Manager will check each endpoint to make sure it is functioning. By default, this means issuing an HTTP request to the root of each site. If it receives an HTTP 200 (OK), then it is considered a healthy endpoint. If Traffic Manager is not able to get a successful response from an endpoint, then it will stop routing traffic to the endpoint. When the endpoint is determined to be healthy again, Traffic Manager will resume routing traffic to it.

After a few minutes, the endpoints will be added to the profile and will appear online, as shown in Figure 3-14.

contos0				
DASHBOARD ENDPOINTS CONFIGURE				
DNS NAME	STATUS	TYPE	LOCATION	
contos0-east.azurewebsites.net	Online	Web app	East US	
contos0.azurewebsites.net	Online	Web app	South Central US	
contos0-west.azurewebsites.net	Online	Web app	West US	

FIGURE 3-14 Endpoints shown as online in the Traffic Manager profile.

To test the configuration, click the **CONFIGURE** tab at the top of the page. In the general section of

the CONFIGURE page, copy the DNS NAME property. Open a browser and paste the DNS NAME into the URL. The default page from one of the three deployments will be returned and will look similar to Figure 3-15 if you are using the default MVC templates for your web app.

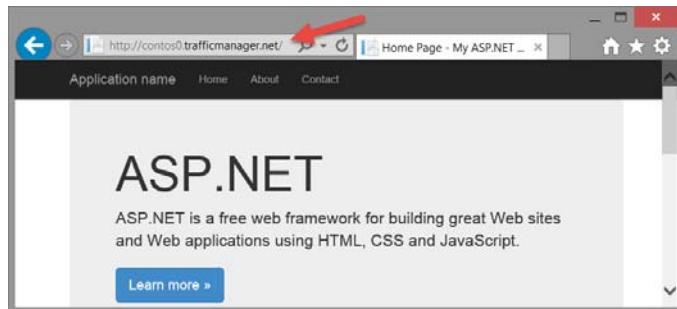


FIGURE 3-15 Default web app page rendered as a result of navigating to the Traffic Manager DNS name.

Additional services for achieving massive scale

When you scale your web app to multiple deployments, you should consider how your application's data is managed. For example, if you store customer data in a database and each deployment has its own database resource, then you need a way to synchronize the data between locations. Azure SQL Database provides an active geo-replication feature that can be used to achieve data continuity. More information about this feature is available at <https://msdn.microsoft.com/en-US/library/azure/dn741339.aspx>.

A content delivery network (CDN) can be used to cache static data used in your web app, such as media and JavaScript files. A CDN caches static data at locations closer to your end users. This can improve the experience for the end user by minimizing network latency. It also improves performance of your web app by eliminating the load on the web server to serve the static content. Azure provides a CDN service that you can use to cache data from Azure Blob storage and Azure Web Apps. More information about this service is available at <http://azure.microsoft.com/en-us/documentation/services/cdn/>.

Scaling WebJobs

Because web jobs run in the context of a web app, when you scale out your web app, your web job is scaled out with it. This is the default behavior; therefore, no action is needed to scale out your web jobs.

Note If you have multiple instances of a web job that use a trigger attribute to invoke a function, Azure will try to invoke the function on a web job where the server resources are most available. One way the load on a server is measured is via the request queue. If web traffic is high on one instance at

the time a function is being triggered, then Azure will find the instance with the lowest load and invoke the function on that instance.

In some situations, it may be preferable to have only one instance of your web job running even though you have multiple instances of your web app. For example, you may have a web job that performs maintenance work that doesn't need to be scaled out. This can be achieved by adding a file to your web job project called *settings.job*. The contents of the file must be formatted as JSON and must contain a property setting for *is_singleton*. Setting this property to true will result in only one instance of a web job running, as shown in Listing 3-1.

LISTING 3-1 settings.job.

```
{ "is_singleton" : true }
```

If you add this file to your web job project, be sure to set Copy to Output Directory to **Copy if newer**, as shown in Figure 3-16. Otherwise it won't get published when you publish the web app.

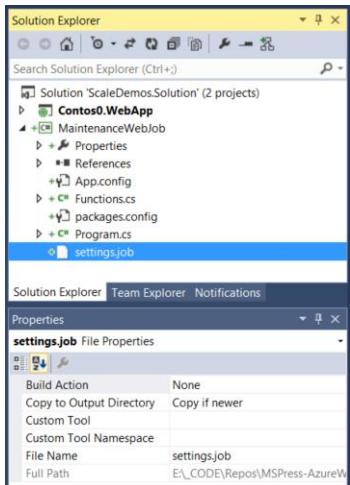


FIGURE 3-16 Web job project with settings.job file.

Summary

This chapter discussed the notion of scale up and scale out and provided essential information necessary to take advantage of the scalability features available in Azure Web Apps. You learned how Autoscale can be used to scale out your web app automatically based on various metrics and how to schedule Autoscale rules for web apps where traffic patterns are highly predictable. Next, you saw how you can achieve massive scalability using Azure Traffic Manager. We concluded with a brief review of

how web jobs scale out and showed how you can make your web jobs run as a singleton in a multi-instance deployment.

Throughout this chapter, you learned that scalability of an app is more than just increasing the capacity of a machine or increasing the number of instances. You also should give careful thought to the architecture of the application so that you can gain the maximum benefit. An excellent discussion on these concepts is available at <https://msdn.microsoft.com/en-us/magazine/dn786914.aspx>. For a discussion of scalability lessons learned from real production applications, see Mark Russinovich's presentation at the 2015 Strategic Architect Forum at <http://channel9.msdn.com/blogs/Microsoft-Architecture/Mark-Russinovich-Strategic-Architect-Forum-2015-Building-Applications-for-Azure-Lessons-from-Scale>.

Chapter 4

Monitoring and diagnostics

Microsoft Azure Web Apps, Visual Studio, and the Azure platform collectively provide a rich set of services and tools that you can use to monitor and troubleshoot your applications. You can monitor your application in real time and interact with near-real-time data using the Azure portal. Or you can have the platform alert you if performance degrades or your application becomes unavailable. If you need to debug your app live or post-mortem, you will find the Azure Web Apps platform rich with data and analysis to get you to the root cause and resolution as fast as possible.

Monitoring and diagnostics is a large topic and easily could be an entire book by itself. So the goal of this chapter is not to take you on a deep journey through every feature. Instead, the goal is to introduce you to features of the platform and give you essential knowledge that you can use to get started diagnosing and monitoring your web apps. We will start by examining the platform's logging features and then transition into monitoring and diagnostic services.

Introduction to diagnostic logs

Diagnostic logs for a web app generally fall into one of two categories: *application diagnostic logs* and *site diagnostic logs*. Application diagnostic logs are generated as a result of logging code you add to your application. Site diagnostic logs are generated automatically by a monitoring service configured on the web server on which your web app is running.

For demonstration purposes, assume the HomeController of an MVC application contains the following code as shown in Listing 4-1 and has been published to Azure Web Apps.

LISTING 4-1 Tracing code added to default page of HomeController.

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        Trace.TraceInformation("Entered {0}.", this.GetType().Name);
        Trace.TraceWarning("Something could be wrong here.");
        Trace.TraceError("Something is definitely wrong here.");
        Trace.TraceInformation("Leaving {0}.", this.GetType().Name);
        Debug.WriteLine("This is a debug only trace message.");
        return View();
    }
}
```

We will use this code to generate some log files and explore the types of logs available. The types of logs that can be enabled for a web app are as follows:

- **Application logging** These are logs that are written specifically from your application code using diagnostic classes such as the `System.Diagnostics.Trace` or the `System.Diagnostics.Debug`. The latter only works if you publish a debug build of your web app (something you should never do on a production app). When you enable application logging, you also must specify a log level, which can be *Error*, *Warning*, *Information*, or *Verbose*. Each log level includes the previous level. For example, if you want only error logs, then you would set the logging level to *Error*. If you want only error and warning logs, then you would set the logging level to *Warning*, and so on.
- **Web server logging** These are HTTP logs (that is, IIS logs) that are written by the web server on which your web app is running. Data in these logs contains fields defined in the W3C extended log file format defined at <https://msdn.microsoft.com/library/windows/desktop/aa814385.aspx> and includes things such as the time it took the server to process a request, cookies that were sent to the client or received by the client, the client's IP address, and much more.
- **Detailed error messages** These are HTML files written by the web server for any requests to the server that result in an HTTP status code 400 or above response. For example, if you request a resource that doesn't exist on the server, you will get an HTTP 404 (Not Found) response. With detailed error messages enabled, an HTML file also will be generated containing suggested causes, possible solutions, and additional details about the request.
- **Failed request tracing** These are XML files written by the web server containing a deeper level of trace information for failed requests. These logs contain visibility into the HTTP modules that were invoked when processing the request, time taken in each module, module tracing messages, and more. A new XML file is generated for each failed request and is named `fr<x>.xml`, where `<x>` is an incrementing number. Failed request logs are intended to be viewed using a browser, and Azure Web Apps facilitates this by generating a style sheet file named `freb.xsl` in the directory where these files are stored.

Tip The Verbose log-level in application logging will include logs written using the `System.Diagnostics.Debug` class only if you publish a debug build of your web app. For example, if you use `Debug.WriteLine()` as a debugging aid in your code, then setting the log level to Verbose will enable you to see these logs when your web app is running in Azure. This behavior does not apply to release builds of your web app.

Debug logging is intended primarily for local development and debugging, so you shouldn't deploy debug builds to Azure unless it is in a Dev/Test environment and you are troubleshooting an issue that requires additional debug logging.

Enable application and site diagnostic logs

Application and site diagnostic logs can be enabled from the Logs blade in the Azure portal. Starting from the Web app blade, click **Settings** in the toolbar. In the Settings blade, click **Application settings**. From here, you can enable the logging, as shown in Figure 4-1.

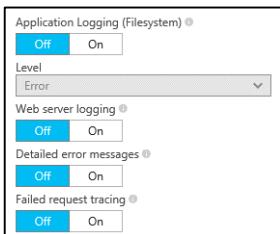


FIGURE 4-1 Application and site diagnostic log configuration in the Azure portal.

Logging also can be enabled from Visual Studio using the Server Explorer window. In Server Explorer, expand the App Services node and right-click the web app. Select **View Settings**. In the Web App Settings section, you can enable the logs using the drop-down boxes. To save the changes to your web app environment, click **Save** at the top of the window, as shown in Figure 4-2.

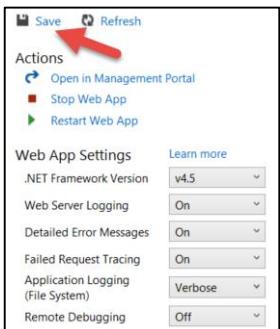


FIGURE 4-2 Application and site diagnostic log configuration in Visual Studio.

Store log files in the web app file system

Log files for an Azure web app are stored on the web server's file system hosting your web app. This is the default configuration. The file system generally is intended to be used for short-term storage of log files because it is limited to the capacity of the web hosting plan. Furthermore, when the virtual machine (VM) on which your web app is running is updated, these logs are lost as a result of your web app instance being moved to another VM instance.

The capacity available for site diagnostic logs is restricted by default to 35 megabytes (MB), but you can increase it up to 100 MB. The capacity of application diagnostic logs is limited by the local storage capacity your web hosting plan provides. As you reach capacity, existing log files are overwritten with

new log files as needed to preserve the storage space on the file system. Table 4-1 shows the path where each type of diagnostic log is stored. Later in the chapter, you will learn ways you can access these files.

TABLE 4-1 Diagnostic log file locations on a web server's file system

Diagnostic Log Type	File system Location
Application logs	D:\home\LogFiles\application
Web server logs	D:\home\LogFiles\http\RawLogs
Detailed error messages	D:\home\LogFiles\DetailedErrors
Failed request tracing	D:\home\LogFiles\W3SVC<x>, where <x> is a random number

Store log files in Azure Storage

Log files also can be stored in an Azure Storage account using table and blob storage. This has the advantage of giving you more storage space (limited only to the capacity of your storage account) to store your log files, more reliability (you don't lose data when the hosting VM is updated), more data logged for each event, and the ability to set retention policies for the logs. Additional data that is captured in the log files when using Azure Storage includes data such as the applicationName, instanceId, eventTickCount, and Thread ID (TID).

At the time of this writing, configuring application diagnostic logs and site diagnostic logs for storage in Azure Storage is not supported in the new Azure portal. To configure these logs for Azure Storage, you must use the old portal at <https://manage.windowsazure.com>.

To configure your web app to store diagnostic logs in Azure Storage, go to the CONFIGURE page of the web app in the Azure Management portal.

Application diagnostic logs are configured in the application diagnostics section. One thing unique to application diagnostics is the ability to enable logging and log levels per storage location. For example, you can have Verbose logs stored in the web app's file system, Error and Warning logs stored in Azure Table Storage, and Verbose logs stored in Azure Blob Storage all at the same time, as shown in Figure 4-3. The green buttons labeled Manage Table Storage and Manage Blob Storage will pop out a dialog where you can select a new or existing table or blob container in which to store the logs.

Tip Configuring logging and log levels per storage location has the added advantage of isolating log files by levels. For example, you probably want to respond to errors quickly. This will be easier for you to do if you are looking only at a log file of errors.

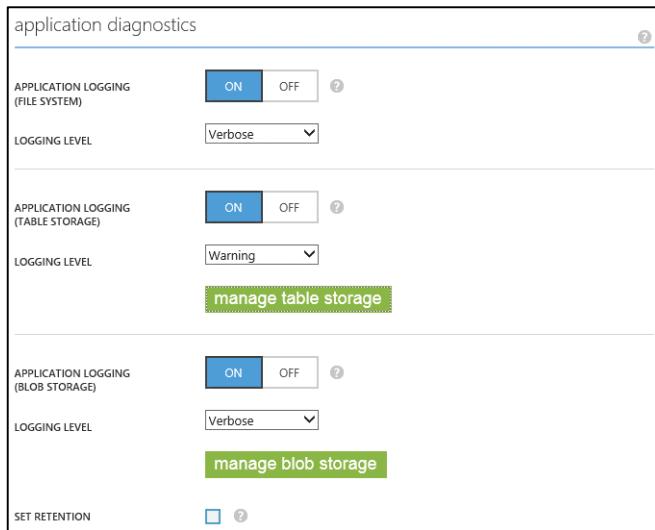


FIGURE 4-3 Application diagnostics configuration in the Azure Management portal.

Site diagnostic logs are configured in the site diagnostics section. Storage of site diagnostics logs in Azure Storage is limited to web server logging using Azure Blob Storage. It is not possible to store this log using Azure Table Storage. Detailed error messages and failed request tracing cannot be configured to use Azure Storage and therefore always will be written to the web app's file system. Figure 4-4 shows the configuration to store web server logging (IIS logs) in Azure Blob Storage with a retention policy of 14 days.

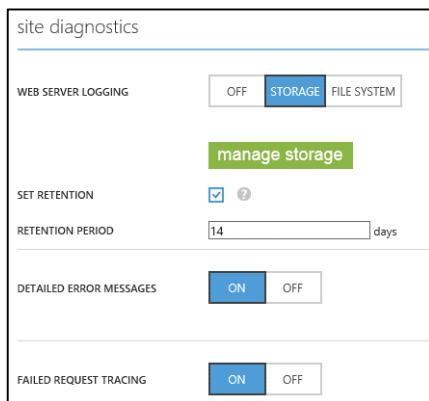


FIGURE 4-4 Site diagnostics configuration in the Azure Management portal.

As a general rule, it is recommended that *site diagnostic* logging be enabled for your web app and stored in Azure Storage. This ensures that you have diagnostics data with which to work when a problem occurs. Then, if you need additional information for troubleshooting, enable *application*

diagnostic logging at the appropriate level (Error, Warning, or Information) to capture additional application-specific information.

Access and download diagnostic log files

Whether your diagnostic logs are stored on your web app's file system or in Azure Storage, it is easy to get to your files to view or download them using a variety of tools.

Access log files stored in the web app file system

This section will introduce you to some common techniques for accessing files on your web app's file system.

Retrieve log files using FTP

You can use an FTP client to connect to your web app's file system and access your logs. To connect to the file system using an FTP client, you will need to configure your FTP client with the host name, user name, and password. These can be obtained from the Web App blade in the Azure portal. The host name and user name are available in the Essentials section at the top of the blade, as shown in Figure 4-5.

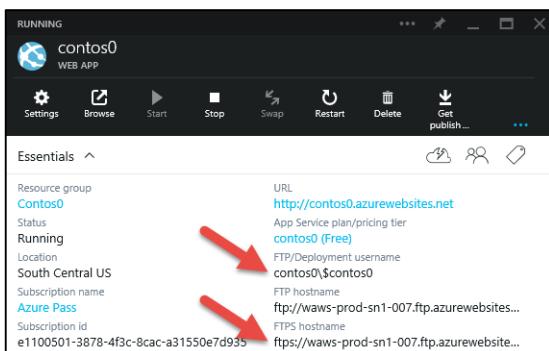


FIGURE 4-5 FTP host name and user name in Web App blade.

The password can be retrieved from the PublishSettings file by clicking the **Get PublishSettings** icon in the toolbar of the Web App blade. This will download the PublishSettings file for your web app. The PublishSettings file is an XML file, so you can open it with any text editor. In the PublishSettings file, locate the userPWD property in the <publishProfile> section.

Tip You should delete the PublishSettings file from your local drive after you are finished with it because it contains credentials used to access your web app's file system.

After connecting to the file system, drill down into the LogFiles folder, where you will find the application and site diagnostic logs. Figure 4-6 shows the failed request logs and the associated style sheet (.xsl) from an FTP client application.

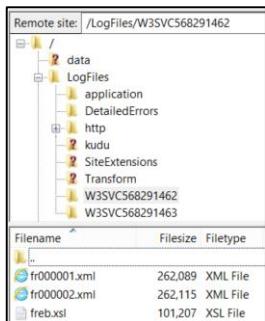


FIGURE 4-6 Accessing failed request logs using an FTP client.

Retrieve log files using Server Explorer

The Server Explorer window in Visual Studio gives you quick and easy access to your log files. Expand the App Service node for your web app and locate the LogFiles folder, where you can navigate into each of the diagnostic log folders. You can view the logs directly from Visual Studio or download them. Figure 4-7 shows the failed request log opened in Visual Studio. Notice also that Visual Studio properly applied the style sheet to the viewer for a richer viewing experience.

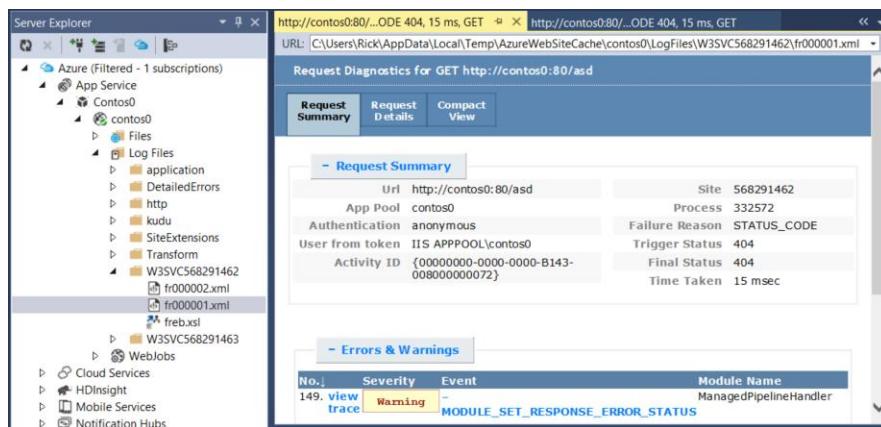


FIGURE 4-7 Viewing a failed request log from Server Explorer.

If you want to download all the logs at once, you can do so by right-clicking the web app in Server Explorer and selecting **View Settings**. Click on the **Logs** tab on the left of the settings dialog. Click the **Download Logs** link in the Actions section, as shown in Figure 4-8.

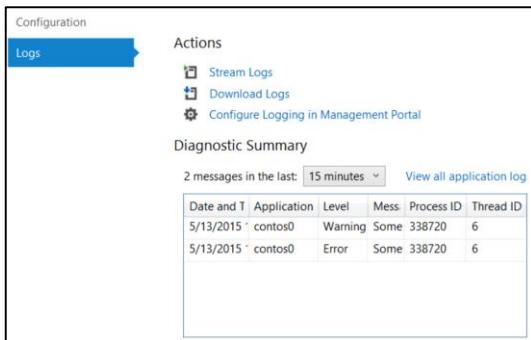


FIGURE 4-8 Logs dialog in Visual Studio.

Retrieve log files using the Azure Web App Logs Browser site extension

The Azure Web App Logs Browser is a site extension you can add to your web app, resulting in easy access to your files using your browser. To get to it, starting from the Web App blade of your web app, click **Settings** in the toolbar. In the Settings blade, click **Extensions**. In the toolbar of the Installed Web App Extensions blade, click **Add**. Click **Azure Web App Logs Browser**. Click **OK** to accept the legal terms. Click **OK** to install the extension. After a few seconds, the Installed Web App Extensions blade will look similar to Figure 4-9.

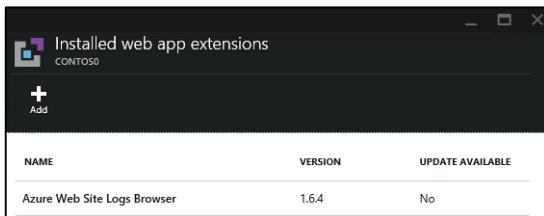


FIGURE 4-9 Installed Web App Extensions blade in the Azure portal.

Click the extension to open the Azure Web App Logs Browser blade. In the toolbar, click **Browse**. This opens a browser window and navigates directly to [https://\[web app name\].scm.azurewebsites.net/websitelogs/#](https://[web app name].scm.azurewebsites.net/websitelogs/#), where you can drill down into each of the folders. Figure 4-10 shows the failed request log folder. You can view the log files in the browser or download them to your computer by clicking the download icon to the left of the file name.

The screenshot shows the Azure Web Site Log Browser interface. At the top, it says "Azure Web Site Log Browser". Below that, it says "File System - Log Files Directory" and "D:\home\LogFiles\W3SVC568291452". There is a table with three columns: "Log File / Directory", "Date", and "Size". The table contains the following data:

Log File / Directory	Date	Size
fr000002.xml	05/12/2015 11:37 AM	262115
fr000001.xml	05/11/2015 4:29 PM	262089
frreb.xml	05/11/2015 4:29 PM	101207

FIGURE 4-10 Failed request logs viewed from the Azure Web Apps Log Browser site extension.

Retrieve log files using command-line tools

Two common command-line tools available for the Azure platform are the Azure PowerShell cmdlets and the Cross-Platform Command-Line Interface (CLI) tools. These can be used to manage resources in Azure from a command line or in an automated manner using script.

See Also For information on how to install and use the Azure PowerShell cmdlets, see <http://azure.microsoft.com/en-us/documentation/articles/powershell-install-configure/>. For information on how to install and use the Cross-Platform Command-Line Interface (CLI) tools, see <http://azure.microsoft.com/en-us/documentation/articles/xplat-cli/>.

You can download all the log files for your web app using the Azure PowerShell cmdlets. Listing 4-2 shows the commands you can run from the PowerShell ISE.

LISTING 4-2 Downloading web app diagnostic logs using Azure PowerShell cmdlets.

```
# Authenticate to Azure
Add-AzureAccount

# Save Log files to local drive
Save-AzureWebsiteLog -Name "Contos0" -Output e:\contos0_logs.zip
```

Note At the time of this writing, version 0.9.1 of the Azure PowerShell cmdlets has a quota limit that prevents you from downloading logs if the total size exceeds 64 kilobytes (KB).

The Azure CLI tools also can be used to download the logs. For example, to download the log files for a web app named "Contos0," enter the commands as shown in Listing 4-3.

LISTING 4-3 Downloading web app diagnostic logs using CLI tools.

```
azure login
azure site log download Contos0 -o e:\contos0_logs.zip
```

Access log files from Azure Storage

Most of the tools shown for retrieving log files from a web app's file system can be used to retrieve files from Azure Storage. The exception is using FTP because you cannot connect to Azure Storage using an FTP client.

Retrieve log files using Server Explorer

In the Server Explorer window, expand the Storage node for the storage account you configured for diagnostic logs. Expand the Blobs node. Locate and drill into the container where your application logs are stored. These log files are stored by the hour in different folders so you are likely to see several folders for a particular date. Click one, and you will see a .csv file similar to what is shown in Figure 4-11.

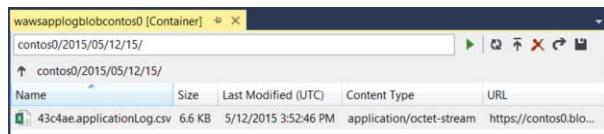


FIGURE 4-11 Application logs stored in Azure Blob Storage.

You can view the file directly in Visual Studio by double-clicking it. Or click the disk icon in the toolbar as shown in Figure 4-11 to download it to your local drive.

Web server logs are retrieved in the same manner. In Server Explorer, go back to the Blobs node and drill into the container where your web server logs are stored. These also are stored by the hour in different folders. As you drill down, you will find your web server logs (IIS logs) with a .log extension.

To view logs stored in Azure Table Storage, expand the Tables node and double-click the table. For example, Figure 4-12 shows application errors and warnings that were stored in Azure Table Storage.

Timestamp	EventTickCount	Application	Level	Event	Instance	Pid	Ti	Message
5/12/2015 1:52:46 PM	635670417...	contos0	Warning	0	43c4ae	338124	7	Something could be wrong here.
5/12/2015 1:52:46 PM	635670417...	contos0	Error	0	43c4ae	338124	7	Something is definitely wrong here.
5/12/2015 1:52:46 PM	635670453...	contos0	Warning	0	43c4ae	342848	7	Something could be wrong here.
5/12/2015 1:52:46 PM	635670453...	contos0	Error	0	43c4ae	342848	7	Something is definitely wrong here.

FIGURE 4-12 Application errors and warnings stored in Azure Table Storage.

Retrieve log files using the Azure Web App Logs Browser site extension

You can view log files in your browser by using the Web App Logs Browser site extension. To install the extension, go to the Web App blade in the Azure portal. Click **Settings** in the toolbar at the top of the blade. In the Settings blade, click **Extensions**. Click the **Add** button at the top of the Installed Web App Extensions blade. Select **Azure Web Site Logs Browser**. Click **OK** to accept the legal terms. Click **OK** to install the extension.

After the extension is installed, open the extension blade by clicking the extension in the Installed Web App Extensions blade. Click **Browse** in the toolbar. Next, click the folder titled Application Logs – Table Storage. The application logs stored in Azure Table Storage will look similar to Figure 4-13.

The screenshot shows the 'Azure Web Site Log Browser' interface. At the top, there's a logo and the title 'Azure Web Site Log Browser'. Below the title, it says 'View Logs From Azure Table Storage' and shows a date range 'May 12, 2015 12:00 AM - May 12, 2015 5:00 PM'. A search bar is present. The main area displays a table of log entries:

Date	Level	Instance	Activity	Message
12 May 2015 10:36	Warning	43c4ae		Something could be wrong here.
12 May 2015 10:36	Error	43c4ae		Something is definitely wrong here.
12 May 2015 11:35	Warning	43c4ae	00000000-0000-0000-7c45-00800100009a	Something could be wrong here.
12 May 2015 11:35	Error	43c4ae	00000000-0000-0000-7c45-00800100009a	Something is definitely wrong here.

At the bottom, it says 'Showing 1 to 4 of 4 entries' and has navigation buttons for 'Previous', '1', and 'Next'.

FIGURE 4-13 Application errors and warnings viewed from the Azure Web App Logs Browser site extension.

The web server logs (IIS logs) can be retrieved in a similar manner. Go back to the previous page, and this time click **HTTP Logs – Blob Storage**. Drill into this folder to view the individual web server logs.

Log streaming

The log-streaming service in Azure Web Apps enables you to view application logs, web server logs, and detailed error messages in nearly real time. You can connect to the log-streaming service and view the logs from Visual Studio and by using command-line tools.

Log streaming using Visual Studio

Open Server Explorer if it is not already open. Double-click through the App Service node to get to the web app. Right-click the web app and select **View Streaming Logs**, as shown in Figure 4-14.

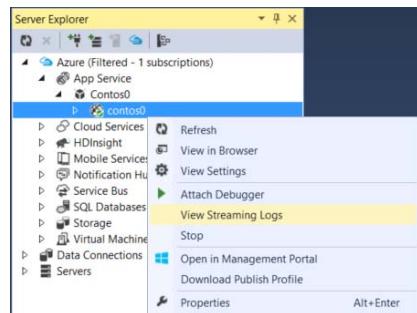


FIGURE 4-14 View streaming logs using Visual Studio.

In the Output window, you will see a message stating You Are Now Connected To Log-Streaming Service. To start viewing logs, browse to the default page of the web app using your browser. Almost immediately, you will see the logs appearing in the output window. Figure 4-15 shows what the output window may look like if you have Application Logging set to Verbose.

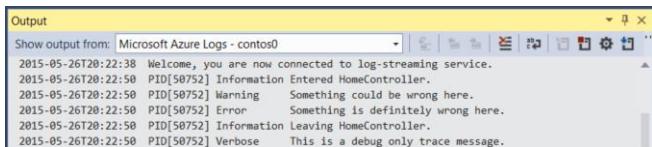


FIGURE 4-15 Application logs shown in Visual Studio using the log-streaming service.

You can change the logs the log-streaming service monitors by clicking the gear icon in the toolbar, as shown in Figure 4-15. This will open a dialog where you can select the logs you want the log-streaming service to capture. In the dialog, you can choose to monitor all logs or specifically the application and web server logs, as shown in Figure 4-16.

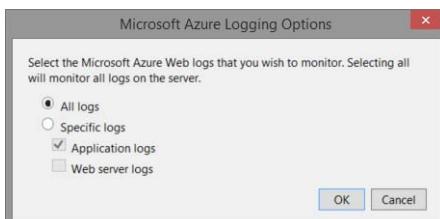


FIGURE 4-16 Azure logging options in Visual Studio.

To disconnect from the log-streaming service, click the stop icon in the toolbar of the Output window, as shown in Figure 4-17.



FIGURE 4-17 Stop monitoring real-time logs in Visual Studio.

Similarly, you can click the start icon in the toolbar to reconnect to the log streaming service.

Log streaming using command-line tools

Using the Azure PowerShell cmdlets, you can view streaming logs from a Windows PowerShell console window. To connect to the log-streaming service, use the `Get-AzureWebsiteLog` command and specify the `-Tail` parameter. Open a browser to the web app's default page, and shortly you will see output in the Windows PowerShell console, as shown in Figure 4-18.

```
PS C:\Windows\system32> Get-AzureWebsiteLog -Name "Contoso0" -Tail  
2015-05-12T22:40:43 Welcome, you are now connected to log-streaming service.  
2015-05-12T22:40:51 PID[329248] Information Entered HomeController.  
2015-05-12T22:40:51 PID[329248] Warning Something could be wrong here.  
2015-05-12T22:40:51 PID[329248] Error Something is definitely wrong here.  
2015-05-12T22:40:51 PID[329248] Information Leaving HomeController.  
2015-05-12T22:40:51 PID[329248] Verbose This is a debug only trace message.
```

FIGURE 4-18 Log streaming using Azure PowerShell cmdlets.

If you want to filter the output, you can apply the *-Message* parameter. For example, Figure 4-19 shows how to filter the output to show only *Information* messages.

```
PS C:\Windows\system32> Get-AzureWebsiteLog -Name "Contoso0" -Tail -Message Information  
2015-05-12T22:45:26 Welcome, you are now connected to log-streaming service.  
2015-05-12T22:45:31 PID[329248] Information Entered HomeController.  
2015-05-12T22:45:31 PID[329248] Information Leaving HomeController.
```

FIGURE 4-19 Log streaming using Azure PowerShell cmdlets and filtered to show only Information messages.

See Also For more information on how to use Azure PowerShell to stream log files and how to use the CLI tools to do the same, see <http://azure.microsoft.com/en-us/documentation/articles/web-sites-enable-diagnostic-log/>.

Remote debugging

Remote debugging enables you to debug your web app interactively while it is running in Azure. The experience is identical to debugging your web app locally, allowing you to set breakpoints, step through code, view variables in memory, and much more. Remote debugging is useful in scenarios in which your web app is behaving differently in Azure than when you run it locally.

Tip You should never perform remote debugging on a production web app. When you debug an application and set breakpoints using remote debugging, execution of the W3WP process in which your app is running is halted until you resume or advance to the next breakpoint. Remote debugging should be used in Dev/Test environments only.

To use remote debugging, you first need to enable it in your Azure web app environment. Starting from the Server Explorer window in Visual Studio, right-click the web app and select **View Settings**. Under the Web App Settings section, set Remote Debugging to On and click **Save** at the top of the window, as shown in Figure 4-20.

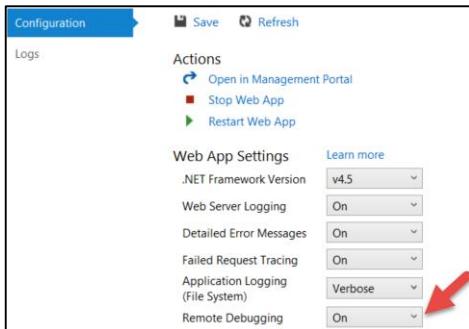


FIGURE 4-20 Enabling Remote Debugging feature from Visual Studio.

In Server Explorer, right-click the web app and select Attach Debugger. This step attaches the Visual Studio debugger to the w3wp.exe process in Azure where your web app is running and launches the web app in your browser.

Note The remote debugging feature is supported for versions of Visual Studio 2012 and newer. But it is not supported for Express editions of Visual Studio. Enabling the remote debugging feature from your Visual Studio environment will configure the feature automatically for the version of Visual Studio you are using.

Set a breakpoint in the default action of the HomeController by pressing **F9**. Next, refresh the browser. This will cause the breakpoint to be hit, as shown in Figure 4-21.

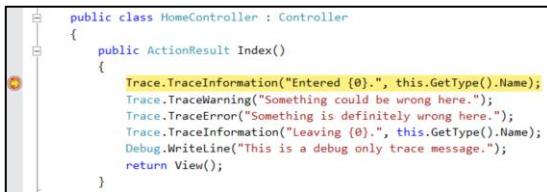


FIGURE 4-21 Visual Studio debugger stopped on a breakpoint during remote debugging.

You can debug the remote process using the Visual Studio debugging features you normally use for debugging a local process. For example, you can use the Immediate window to view the `HttpRequest` object for the current request by selecting **DEBUG > Windows > Immediate** from the main menu. In the Immediate window, type **this.Request** and press **Enter**. This will display the contents of the `HttpRequest` object, as shown in Figure 4-22. Additionally, you can view the Call Stack, Memory, Variables, and other debugging windows you prefer to use.

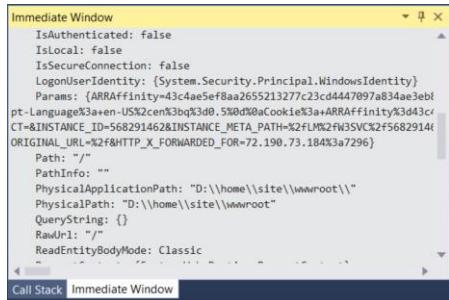


FIGURE 4-22 Immediate window in Visual Studio showing properties of the current request.

At any time, you can end your remote debugging session by clicking the stop icon in the Visual Studio toolbar or selecting **DEBUG > Stop Debugging** from the main menu.

See Also To learn more about troubleshooting a web app using Visual Studio, see <http://azure.microsoft.com/en-us/documentation/articles/web-sites-dotnet-troubleshoot-visual-studio/>.

Diagnostics as a Service (DaaS)

Diagnostics as a Service (DaaS) is a feature of Azure Web Apps that delivers extremely powerful diagnostics capabilities to you using just your browser. DaaS is an Azure site extension that you can install from the Azure portal. After you have installed it, you can collect event logs, web server logs, and even memory dumps from your web app environment. DaaS then will perform deep analysis on these data sources and produce rich reports through which you can navigate to find the data you need to diagnose your issues.

Note If you have ever used the Debug Diagnostics Tool (DebugDiag) to troubleshoot issues on-premises, then you will recognize the power and richness of data this service delivers.

DaaS is safe to use in production environments and is available for web apps in the Basic, Standard, and Premium pricing tiers.

Install the Diagnostics as a Service site extension

Starting from the Web App blade for the web app on which you want to use DaaS, click **Settings** in the toolbar at the top of the blade. In the Settings blade, click **Extensions**. In the Installed Web App Extensions blade, click **Add** in the toolbar and select the **Diagnostics as a Service** extension. Click **OK** to accept the legal terms and then **OK** to install the extension.

DaaS requires that web server logging be enabled and configured to use the file system (not Azure Storage) of your web app. To do this from the Azure portal, click **Settings** in the toolbar of the Web App blade. In the Settings blade, click **Diagnostic Logs**. In the Logs blade, switch Web Server Logging to On, as shown in Figure 4-23. Click **Save** in the toolbar to save the change.

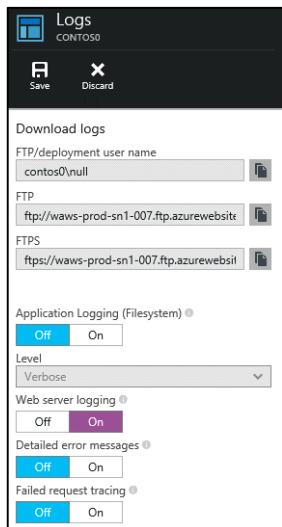


FIGURE 4-23 Diagnostic Logs blade in the Azure portal.

Run DaaS

Starting from the Settings blade, click **Extensions**. In the Installed Web App Extensions blade, click **Diagnostics As A Service**. In the Diagnostics As A Service blade, click **Browse** located in the toolbar.

Tip You can navigate directly to the DaaS site extension for your web app using the URL [https://\[your_web_app_name\].scm.azurewebsites.net/DaaS/](https://[your_web_app_name].scm.azurewebsites.net/DaaS/). If you are not already authenticated to Azure, you will be prompted to sign in. Sign in using the credentials you use to sign in to the Azure portal.

The DaaS landing page will load in your browser and look similar to Figure 4-24. From this page, you can start a diagnostics session immediately by clicking the **Diagnose Now** button or schedule a diagnostics session by clicking the **Schedule Analysis** button.



FIGURE 4-24 Diagnostics as a Service site extension.

Click the **Diagnose Now** button. This will start a new diagnostics session that will start pulling information from the event log and web server logs. It also will capture a memory dump of the w3wp.exe process in which your web app is running. If your web app is using PHP, then it also will collect information from the PHP host process.

Tip Azure web apps have an eventlog.xml file located on the file system at D:\home\LogFiles. This file is generated automatically for you and contains application events that often are useful when troubleshooting errors at run time.

After all the data is collected, DaaS will perform analysis of the data and generate a report for each of the data sources. This process can take a few minutes to complete. You can check the status of the session by expanding it to see what remains to be completed. An in-progress session will look similar to Figure 4-25.

Sessions		
StartTime	Collection Status	Analysis Status
2015-05-14 18:20:29	InProgress	WaitingForInputs
Event Viewer Logs		
Memory Dump		
Http Logs		
PHP Process Report		

FIGURE 4-25 DaaS session in progress.

After the collection and analysis of all the data sources is finished, you will see the status changed to Complete. To view the analysis reports, click the right arrow icon in the upper-right corner, as shown in Figure 4-25.

View DaaS analysis reports

You can view an analysis report by clicking the document icon in the analysis column for the data source in which you're interested. For example, to view the memory dump analysis, click the document

icon to show the link to the memory dump report. Similarly, to download and view any of the data that was collected, click the document icon in the collection column. Figure 4-26 shows the memory dump analysis and HTTP logs collection links.

Session: 2015-05-14 18:20:29		
Diagnoser	Collection Status	Analysis Status
Event Viewer Logs		
Memory Dump		
Http Logs		

FIGURE 4-26 Analysis report links and data links from DaaS session.

The event log analysis report will report any events that were found in the eventlog.xml file, a description of the event, and potentially a link to solution content. Figure 4-27 shows an example of an event log report where an error event was found. Also, notice that a link to solution content on MSDN has been identified as relevant to this particular error.

Event Log Processing Report	
File Processed: D:\local\Temp\Log\contoso0\15-05-14\RD0000D3A703FB4\EventViewer\150514_1820292641\150514_1822292641\eventlog.xml	
Processing Summary: Success	
Field	Rule Content
Title	Failed to enable remote debugger
Type	Error
Detected Condition	/Events/Event[System/EventID=1001 and System/Provider/@Name="Visual Studio Remote Debugger" and EventData/Data[server]]
Solution Description	Failed to enable remote debugger. Please check link in Solution Content Url field for more solution.
Solution Content Url	http://msdn.microsoft.com/en-us/library/ms161726.aspx
	Failed to enable remote debugger. The following events have been detected: Found 6 events:Event Found #1.....

FIGURE 4-27 Event log processing report generated from Daas.

The memory dump analysis report contains a massive amount of information, as it should, because its data source is a memory dump of the w3wp.exe process in which your web app is running. If your application is experiencing high memory usage, high CPU, or hanging as users send requests to it, this report will give you information that can help you determine the root cause. The report is structured into three sections, as follows:

- **Analysis Summary** Contains a summary of issues found during analysis, which can include known errors or just information that could be affecting the performance of your application. When possible, it also will provide recommendations for issues found.
 - **Analysis Details** This is the bulk of the analysis report. It includes information such as the top five threads consuming CPU time, HttpContext reports for incoming and outgoing HTTP requests, .NET exception objects that are present in memory at the time the memory dump was taken, a thread report showing the call stack of each thread, .NET CLR heap and memory information including which .NET objects are consuming the most memory, and more.

- Analysis Rule Summary** A table showing which rules were used to analyze the memory dump. Common rules include performing a crash analysis, hang analysis, and memory analysis on the memory dump file.

Figure 4-28 shows an example of an error reported from the Analysis Summary section of the memory dump analysis report, the recommended remediation, and a link to a blog explaining why this is considered an error.

Analysis Summary		
Type	Description	Recommendation
Error	In <code>w3wp_172_17ce_2015-05-14_18-20-30-359_07b4.dmp</code> debug is set to true for the runtime + <code>D:\home\site\wwwroot\</code>	For applications running in production Debug should never be set to true and it has great performance impact. For more details on debug setting please refer to this blog If your application is in production? then why_is_debug=true

FIGURE 4-28 Error reported in the Analysis Summary section of memory dump analysis report generated from DaaS.

The http logs analysis report is generated by examining the web server logs (IIS logs) on your web app's file system. It uses LogParser behind the scenes to run some common queries against the web server logs and compiles the results into a linkable report. This report is useful in many scenarios, such as identifying which pages in your app are slowest, which pages are most visited, requests per hour, and more. Figure 4-29 is an example of the beginning of an IIS log analysis report where each line links to a table showing the results for that section of the report.

FIGURE 4-29 Http log analysis report generated from DaaS.

At the time of this writing, the DaaS site extension doesn't provide a way for you to download all the data and reports from your DaaS sessions. It also doesn't provide a way for you to remove these files from your web app's file system. In the next section, you will learn about another site extension you can use to download these files easily and remove them if you want.

See Also To learn more about the *Diagnostics as a Service (DaaS)* site extension, see <http://azure.microsoft.com/blog/?p=157471>.

Site Admin Tools/Kudu

Site Admin Tools is a site extension that runs in the same environment in which your web app is running. Some know this tool as Kudu, and in some documentation it is referred to as Site Control Manager (SCM).

Using the Site Admin Tools site extension, you can explore the web app environment, access the file system for your web app, explore running processes in your environment using a built-in process explorer, and even install other site extensions.

See Also *Kudu has been a part of the Azure Web App platform since the beginning. Initially, its primary purpose was to provide support for Git deployments. But it also provides some powerful tools and visibility into your web app environment, all from a browser. To learn more about Kudu's capabilities, see <https://github.com/projectkudu/kudu/wiki>.*

Install the Site Admin Tools/Kudu

Starting from the Web App blade, click **Settings** in the toolbar at the top of the blade. In the Settings blade, click **Extensions**. In the Installed Web App Extensions blade, click **Add** in the toolbar and select the **Site Admin Tools** extension. Click **OK** to accept the legal terms and then **OK** to install the extension.

Note Technically it's not necessary to install the Kudu site extension. Every Azure web app includes this site extension by default. But going through the steps to install it makes it appear as an installed site extension for your web app that you later can use to access the tools from the Azure portal.

Run the Site Admin Tools

Starting from the Settings blade, click **Extensions**. In the Installed Web App Extensions blade, click **Site Admin Tools**. In the toolbar of the Site Admin Tools blade, click **Browse**.

Tip You can navigate directly to Kudu using the URL [https://\[your_web_app_name\].scm.azurewebsites.net/](https://[your_web_app_name].scm.azurewebsites.net/). If you're not authenticated to Azure, then you will be prompted to sign in using the credentials you use to sign in to the Azure portal.

The Kudu landing page will load in your browser and look similar to Figure 4-30.

The screenshot shows the Kudu interface with the 'Environment' tab selected. It displays system information such as Build (45.40512.1539.0), Site up time (08:01:07.3869060), Site folder (D:\home), and Temp folder (D:\local\Temp).

FIGURE 4-30 Site Admin Tools/Kudu site extension landing page.

Using the menu across the top, click **Environment**. The environment page will show you system information, app settings, and connection strings, which include the app settings and connection strings you have configured for your web app. It also includes environment and server variables that are available to your web app.

Debug console

When you click on Debug Console, a menu option gives you a choice to open a CMD console or PowerShell console window. Whichever you choose, the real power of the debug console is that it puts the file system of your web app at your fingertips. You can issue commands directly in the console or click through the file system using the UI. To demonstrate, click **Debug Console** and then **CMD**. After the console loads, you will see an explorer-like view of your file system at the top. The bottom part of the page will show the console window. You can navigate the file system directly in the console window or by using the UI. Regardless of which approach you take, notice that the two stay in sync. The UI at the top of the page will look similar to Figure 4-31.

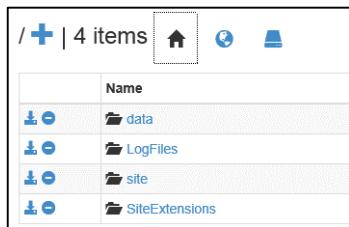


FIGURE 4-31 The home directory of the web app file system.

Earlier in this chapter, you learned that your application diagnostic and site diagnostic logs are stored in the web app's file system at d:\home\LogFiles, which you can see in Figure 4-31. You can click this folder and see the folders where the various types of logs are stored. If you want to download the entire LogFiles folder and subfolders, you can click the download icon left of the folder icon. Kudu will zip the directory contents and start a download. If you're using Internet Explorer, you will see the download notification at the bottom of the screen, as shown in Figure 4-32.



FIGURE 4-32 Downloading the LogFiles folder using the Site Admin Tools site extension.

The site folder is where your web app files are published when you publish from Visual Studio. For example, if your application is an ASP.NET MVC application, you will find your web.config, views (.cshtml), scripts, font files, and more in the d:\home\site\wwwroot folder.

In the previous section, you learned about the DaaS site extension. One of the data sources from which it collected data was the eventlog.xml file located in the d:\LogFiles folder. This is where DaaS looked to collect the data for the event log analysis report.

DaaS stores most of its data in the folder d:\home\data\DaaS. Using the UI or the console window, navigate to this directory. The top of your page will look similar to Figure 4-33.

... / DaaS + 6 items	
	Name
↳	bin
↳	Heartbeats
↳	Logs
↳	Reports
↳	Sessions
↳	PrivateSettings.xml

FIGURE 4-33 Contents of d:\home\data\DaaS as shown using Kudu's debug console.

The various analysis reports you viewed in the previous section are in the Reports folder. So you can click the download icon next to the Reports folder to download all your reports at once.

The Logs folder is where data that DaaS collected is stored. Recall that one of the data sources is memory dump files that DaaS generates and analyzes to produce the memory dump analysis report. Memory dump files can be large, and if you have a lot of these, they can consume a significant portion of your web app's file system quota. If you want to clean up these files after you have your analysis reports, you can do so by drilling down into the Logs folder. Memory dump files (.dmp) are located under a folder named MemoryDumpCollector, which will comprise part of the path to the file. Figure 4-34 shows an example of where one of the dump files for a DaaS memory dump analysis is stored. Your path will be different but similar.



FIGURE 4-34 Memory dump file in a web app's file system.

You also can upload individual files to your web app's file system or a zip file that contains multiple files and subdirectories. When you upload a zip file, Kudu will unzip the contents and place them in the folder to which you uploaded. To upload a file from your local drive, open Windows Explorer and just drag and drop the file into the folder displayed in your browser to which you want the files uploaded.

Process explorer

Kudu includes a process explorer feature that you can use to view processes in your environment. Using the menu across the top, click **Process Explorer**. The process explorer page will look similar to Figure 4-35.

Process Explorer							
<input type="button" value="Find Handle..."/> <input type="button" value="Refresh"/>							
name	pid	user_name	total_cpu_time	working_set	private_memory	thread_count	properties
w3wp.exe [admin]	1408	contoso	61 s	97,260 KB	85,980 KB	49	<input type="button" value="Properties"/>
cmd.exe [website]	4704	contoso	<1 s	176 KB	3,604 KB	2	<input type="button" value="Properties"/>
ComputeWebJobsSDKQueue.exe [website]	2004	contoso	2 s	13,580 KB	23,556 KB	11	<input type="button" value="Properties"/>
cmd.exe [website]	6980	contoso	<1 s	5,120 KB	3,596 KB	2	<input type="button" value="Properties"/>
DaaSRunner.exe [website]	5460	contoso	5 s	23,264 KB	16,512 KB	6	<input type="button" value="Properties"/>
w3wp.exe	1972	contoso	4 s	274,576 KB	49,124 KB	25	<input type="button" value="Properties"/>

FIGURE 4-35 Process explorer page in Site Admin Tools site extension.

The Properties button shown for each process will present detailed information about the process and includes things such as modules, handles, threads, and environment variables the process is using. Also, at the bottom of the properties page are buttons that you can use to kill a process or generate a memory dump of the process.

Monitor web app endpoints externally using web tests

Web tests enable you to test the availability of web app endpoints from geo-distributed locations around the world. This feature is available for web apps configured for Basic and higher pricing tiers.

There are two types of web tests you can create, as follows:

- **URL ping test** In this type of test, you specify a URL that you want to test, one or more locations around the world from which you want the endpoint tested, and what the success criteria are for the test. The success criteria can be a specific HTTP status code, such as HTTP 200 (OK), or they can be a content match in which the test is successful only if the content in the response contains the content that you indicate it should.
- **Multi-step test** In a multi-step test, you upload a *.webtest file that you create using Visual Studio. This is useful if you need to test multiple URLs of your web app. For example, for an expense-reporting app you may need to test the ability to view an expense summary page and detail page and the ability to add or remove expense items. The overall success of the test depends on whether each step in the .webtest file was successful.

An alert can be configured for your web tests to send an email if a test fails. The email can be sent to subscription admins or to a list of email addresses you specify in the alert. An alert has a sensitivity property that is used to control how often an alert is triggered. The sensitivity settings for an alert are as follows:

- **Low** The alert is triggered when a web test fails for all locations within a 15-minute period.
- **Medium** The alert is triggered when a web test fails for at least half of the locations in a 10-minute period.
- **High** The alert is triggered anytime a web test fails.

Web tests are configured from the Web Tests blade in the Azure portal. You can get to the Web Tests blade by clicking the **Web Tests** tile in the Web App blade.

If the Web Tests tile is not present on the Web App blade, then customize the blade and add it. This is done by right-clicking anywhere in the Web App blade and selecting **Customize**. Next, scroll down to the Monitoring section and click the + icon to add a new tile, as shown in Figure 4-36.

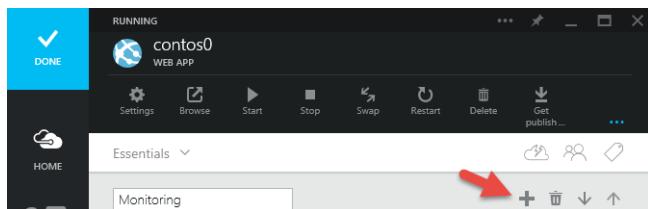


FIGURE 4-36 Customizing the Web App blade in the Azure portal.

In the Tile Gallery, select the **Web Tests** tile and drag it onto the Web App blade under the Monitoring section. Next, click the blue **DONE** button in the upper-left corner to save the customizations. The Monitoring section of the Web App blade will look similar to Figure 4-37.

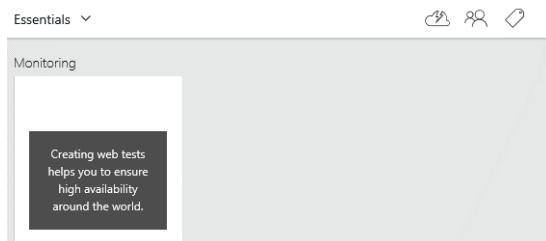


FIGURE 4-37 Web Tests tile added to the Monitoring section of the Web App blade.

See Also For more information on monitoring your web app and configuring alerts, see <http://azure.microsoft.com/en-us/documentation/articles/web-sites-monitor/>.

Create a URL ping web test

Click the **Web Tests** tile to create a test. In the Create Test blade, provide a name for the test, such as "Default Test." Click the **TEST LOCATIONS** box and select a couple of locations from which you want the test invoked and click **OK**. Click **Create** to create the web test and return to the Web App blade.

The web test will start getting invoked on a periodic schedule. After a few seconds, click the **Web Tests** tile to view the results of the tests. The Web Tests blade will look similar to Figure 4-38.

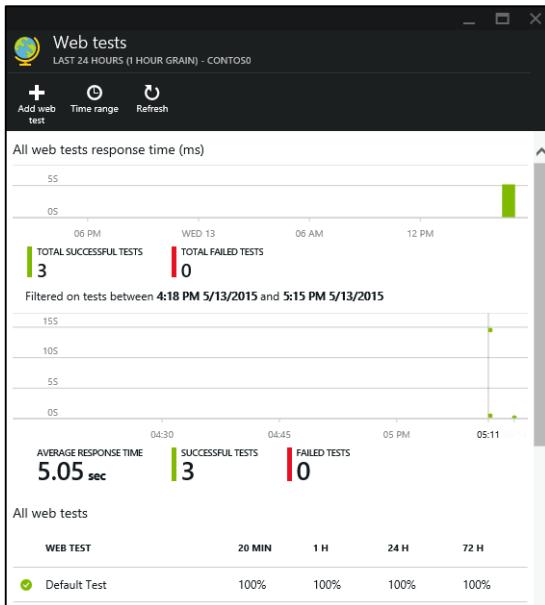


FIGURE 4-38 Web Tests blade in the Azure portal.

You can click the green dots in the scattergraph to get details on a specific web test. If there are any failures, they will be indicated as red dots.

At the bottom of the Web Tests blade, you can see the web tests that have been configured and the percentage of success for the last 20-minute, 1-hour, 24-hour, and 72-hour periods. You can click the web test to see the success ratios for each location from which the test is invoked and the response times for each.

See Also *It is a best practice to implement a health-check page in your web app that checks the availability of other resources on which your web app depends. For example, if your web app uses a database, then verifying the connection to the database in the health-check page would enable you to return an error response to the endpoint monitoring service to indicate there is a problem. For more information on this monitoring pattern and guidance on when and how to use it, see <https://msdn.microsoft.com/en-us/library/dn589789.aspx>.*

To demonstrate what a failed web test would look like, go back to the Web App blade and click **Stop** in the toolbar at the top of the blade. This will stop your web app, which will cause the web test to fail. Return to the Web Tests blade and click **Refresh** at the top of the blade. If you don't see a failed test, wait a few seconds and refresh again until you see a failure. You will notice a red dot in the scattergraph. Below the scattergraph, click **Default Test** under the All Web Tests section to see the detailed results for that test. At the bottom of the blade, you can see which location failed, as shown in Figure 4-39. You can click the location to see more details for that test location.

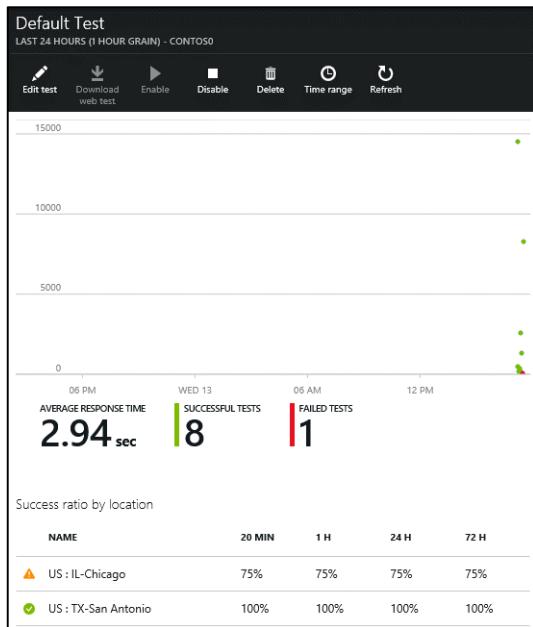


FIGURE 4-39 Web Tests blade showing success ratios by location.

See Also For more information on configuring web tests and how to set up a multi-step test, see <http://azure.microsoft.com/en-us/documentation/articles/app-insights-monitor-web-app-availability/>.

Monitoring

The Azure portal delivers a feature-rich user interface for monitoring your Azure resources. You can scope your monitoring to an entire resource group or perform resource-specific monitoring on individual resources such as a web app, database, or cache.

Monitor a resource group using the Azure portal

The Resource Group blade in the Azure portal is an ideal place to start exploring the monitoring capabilities. To get to the resource group, click **Browse** in the navigation bar on the left of the page. In the Browse blade, select **Resource Groups**. In the Resource Groups blade, select the resource group your web app is in. The Resource Group blade will be divided into sections, each containing one or more tiles. Scrolling down the blade, you will see the Monitoring and Billing sections that will look similar to Figure 4-40.

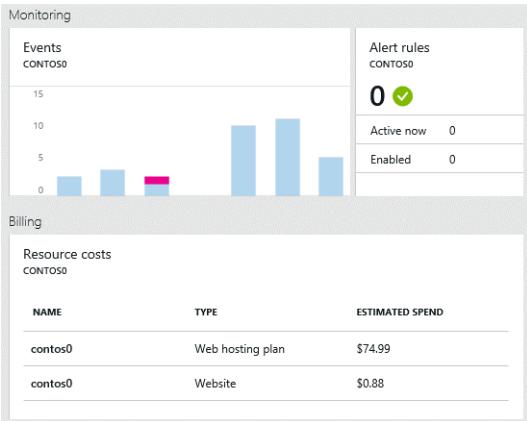


FIGURE 4-40 Monitoring and Billing sections of the Resource Group blade.

Each of the tiles shown can be drilled into by clicking them. For example, to see the event details that comprise the bar chart in the Events tile, click the Events tile. In the Events blade, you will be able to see all the events from all of the resources in the resource group. You can click individual events (operations) in the Events blade to drill down into a specific event. For example, an update website operation would look similar to Figure 4-41.

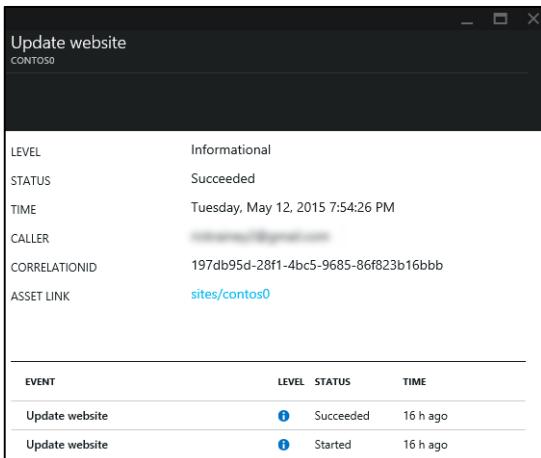


FIGURE 4-41 Update Website blade showing events specific to the operation.

Each event in the list shown can be explored further by clicking the event to open the Detail blade for the event.

See Also The Azure portal is highly customizable and allows you to create views to monitor resources that are important to you. You can customize the start board and individual blades by right-clicking the page and selecting **Customize**. For example, the Resource Group blade can be customized to show additional content tiles, resize the tiles, and rename, delete, add, and reorder sections. For examples of how you can customize the Azure portal to monitor web apps, see <http://azure.microsoft.com/en-us/documentation/articles/insights-how-to-customize-monitoring/>. For additional information and demonstrations showing how to get the most out of the Azure portal, see <http://blogs.msdn.com/b/brianke/archive/2014/04/10/building-your-dream-devops-dashboard-with-the-new-azure-preview-portal.aspx>.

Application Insights

Application Insights is a developer service that collects telemetry data from your application that you can use to monitor your application's availability, performance, reliability, exceptions, usage trends, and more. The telemetry data from your application is stored and processed in Azure in nearly real time.

When you create a new web app using Visual Studio or the Azure portal, you automatically get an Application Insights resource added to your resource group. The Application Insights resource includes an instrumentation key that is used to identify telemetry data from your application and is the resource you use to view and interact with the data using the Azure portal. If the project you create is an ASP.NET Web Application, you get an opportunity during project creation to let Visual Studio add the packages, code, and configuration to your project so you are ready to start using the service. Figure 4-42 shows a portion of the New Project dialog where you can choose to let Visual Studio set this up for you.

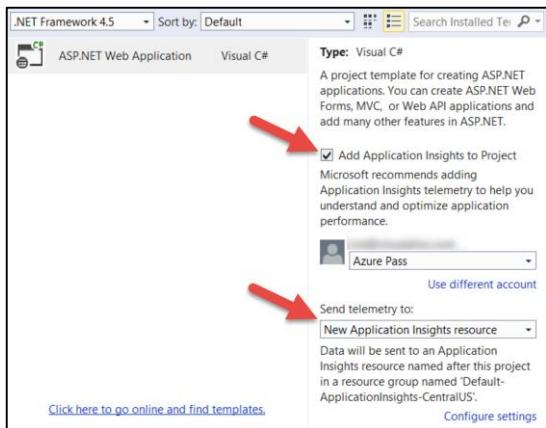


FIGURE 4-42 Adding Application Insights to a new ASP.NET Web Application project in Visual Studio.

If you know you will be using Application Insights with your project, this is the recommended way to add it to your project. But if you have an existing application to which you want to add Application Insights, you easily can do so by using Visual Studio.

See Also There is a lot of documentation online for Application Insights. To learn more about the service, see <http://azure.microsoft.com/en-us/documentation/services/application-insights/>.

This section will walk you through the steps to add this feature to an existing application and then show you how to start interacting with your application's telemetry data using the Azure portal. Taking this approach also will show you the code and configuration that is added to your project when you choose to add Application Insights.

Add Application Insights to an existing ASP.NET MVC Web Application

If you have an existing ASP.NET MVC Web Application to which you want to add Application Insights, open it in Visual Studio.

Note If you want to create a new project, then from the Visual Studio menu, select **FILE > New > Project**. In the New Project dialog, choose the ASP.NET Web Application template and deselect the option to Add Application Insights To Project. Proceed through the rest of the new project wizard to create your project.

From the Solution Explorer window, right-click the project and select **Add Application Insights Telemetry**. In the Application Insights dialog, you have an opportunity to choose the Application Insights resource to which you want your telemetry data sent or to create a new resource if one doesn't exist. Click the blue button labeled **Add Application Insights To Project** at the bottom of the dialog, as shown in Figure 4-43.

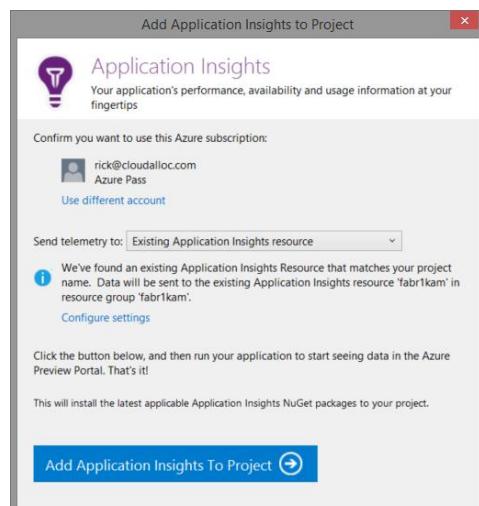


FIGURE 4-43 Adding Application Insights to a project in Visual Studio.

The following changes are made to your project:

- **NuGet Package References** The Microsoft.ApplicationInsights.* packages and necessary dependencies are added. You will see these in the packages.config file of your project.
- **ApplicationInsights.config** This file is added to your project and provides configuration used by the NuGet packages that were added to the project. At the bottom of this file, you will see the <InstrumentationKey>, which is used to identify your application's telemetry data with the Application Insights service.
- **Web.Config changes** The web.config file is updated so that ApplicationInsightsWebTracking is added to the HTTP modules.

Right-click the project in Solution Explorer and select **Publish**. After the changes have been published, browse to some of the pages in your web application a few times to generate some web traffic. In the URL field of your browser, append some random text such as "foobar" to the end of the URL to generate an error (HTTP 404).

In the Azure portal, open the Resource Group blade for your web app and click the Application Insights resource in the Summary section, as shown in Figure 4-44.

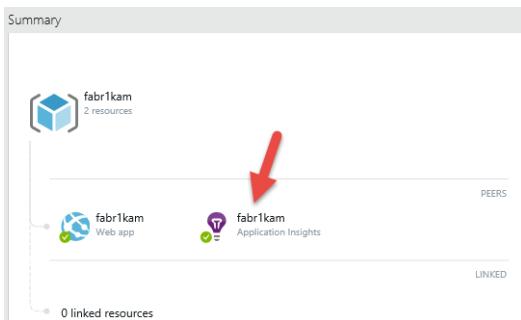


FIGURE 4-44 Summary section of the Resource Group blade in the Azure portal.

The Application Insights blade provides a rich and interactive view of the telemetry data your web app is generating. In the Application Health section of the blade, you will see graphs showing server response times, the number of server requests, and the number of failed requests, which will look similar to Figure 4-45.

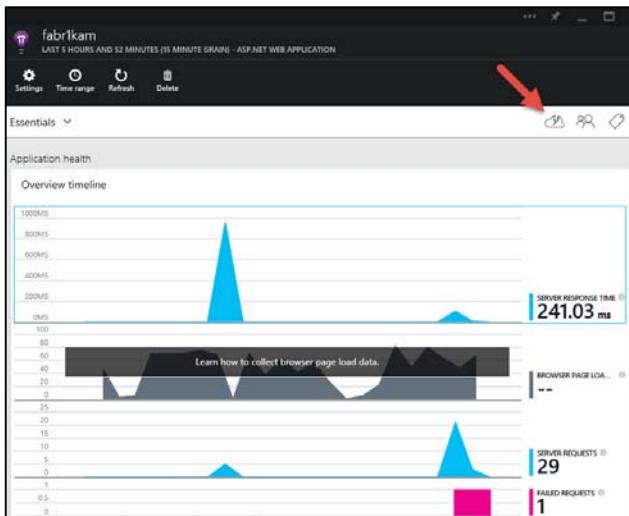


FIGURE 4-45 Application Health section of the Application Insights blade in the Azure portal.

Notice the graph for Browser Page Load is grayed out with a message inviting you to learn how to collect this data. You can click the message in the graph or click the Quick Start icon in the upper-right corner of the Essentials section, as shown in Figure 4-45. In the Quick Start blade, locate the section labeled Add Code To Monitor Web Pages and click the link labeled **Get Code To Monitor My Web Pages**, as shown in Figure 4-46.

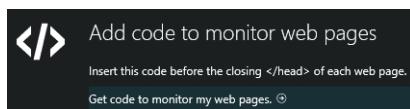
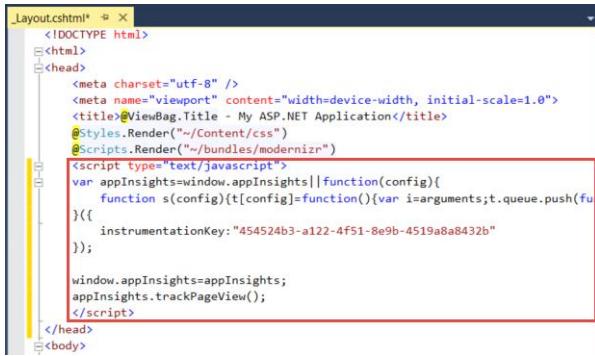


FIGURE 4-46 Link in Quick Start blade to get code to add to your web pages.

In the End-User Usage Analytics Code blade is JavaScript that includes your Application Insights resource instrumentation key and code to collect end-user usage analytics and send them to your Application Insights resource in Azure. Copy the `<script>` section to your Clipboard (Ctrl+C).

In Visual Studio, open _Layout.cshtml, which is located in the Views > Shared folder of your project. Paste the script copied to your Clipboard just before the `</head>` element near the top of the file, as shown in Figure 4-47.



```
<_Layout.cshtml>
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - My ASP.NET Application</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
<script type="text/javascript">
var appInsights>window.appInsights||function(config){
    function s(config){t[config]=function(){var i=arguments;t.queue.push(function() {
        instrumentationKey:"454524b3-a122-4f51-8e9b-4519a8a8432b"
    });
    window.appInsights=appInsights;
    appInsights.trackPageView();
    </script>
</head>
<body>
```

FIGURE 4-47 _Layout.cshtml with the Application Insights end-user analytics script added.

Save the changes (Ctrl+S). Right-click the project in Solution Explorer and select **Publish** to publish the change to your web app in Azure. After the updates are published, the browser will launch. Navigate some pages in your web app as you did before to generate some web traffic.

Return to the Application Insights blade in the Azure portal. The Browsers Page Load graph will start showing data. If you don't see it right away, wait a few seconds and refresh the blade. When you see the graph, click it to drill into the data using the Browsers blade. The Browsers blade will look similar to Figure 4-48 and shows graphs and data for client-side metrics.

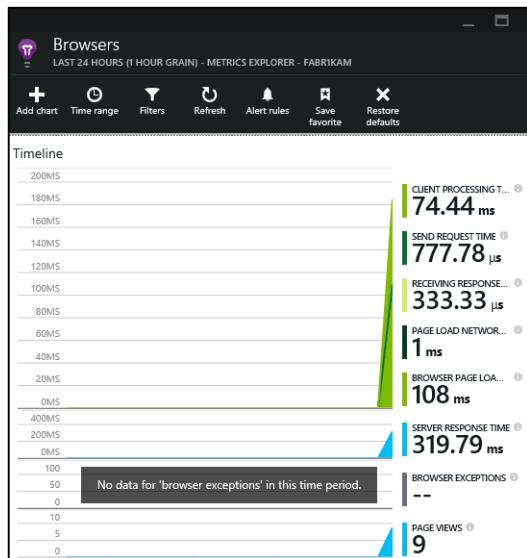


FIGURE 4-48 The Application Insights Browsers blade in the Azure portal.

See Also Application Insights can be used to monitor the availability and responsiveness of any website. To learn how, see <http://azure.microsoft.com/en-us/documentation/articles/app-insights-monitor-web-app-availability/>.

See Also Application Insights captures a large amount of telemetry data, and the Azure portal provides a way for you to see the data at a high level to identify application trends and event totals. To learn how to search and filter the data down to individual items for deeper analysis, see <http://azure.microsoft.com/en-us/documentation/articles/app-insights-diagnostic-search/>.

Summary

In this chapter, you gained essential information about application and site diagnostic logs including how to configure them, how and where they are stored, and several techniques you can use to access them. We discussed debugging techniques such as log streaming, remote debugging, and the Diagnostics as a Service (DaaS) site extension. The Site Admin Tools (Kudu) and how you can use them to explore the web app environment, file system, and processes hosting your web app were addressed further. We wrapped up by taking a brief look at built-in monitoring capabilities of the Azure portal and how you can use Application Insights to capture deep telemetry data to monitor and troubleshoot your web app in nearly real time.

See Also To learn about other solutions available for monitoring cloud applications, see <http://www.asp.net/aspnet/overview/developing-apps-with-windows-azure/building-real-world-cloud-apps-with-windows-azure/monitoring-and-telemetry>.

About the author



Rick Rainey is an independent consultant and owner of CloudAlloc, LLC. He specializes in helping customers migrate to and build new applications to run on the Microsoft Azure Platform. He has over 25 years of experience designing, developing, and supporting applications using Microsoft technologies.

Rick is a Microsoft Azure Insider and Advisor, Certified Trainer (MCT), speaker, blogger, and Azure community enthusiast. He worked for Microsoft in various developer-focused roles for 12 years with emphasis on helping ISV's develop solutions for the Windows and Azure platforms.

He is an active contributor in the Azure community. He speaks often at community events, organized and presented at the Dallas Global Azure Bootcamp, and helps support the broader Azure community by answering questions on Stack Overflow. He also writes about Azure on his blog at <http://rickrainey.com> and tweets Azure goodness on Twitter at [@RickRaineyTx](https://twitter.com/RickRaineyTx).

Rick resides with his family in Dallas, Texas. Outside of work he is an avid runner and an occasional biker.



From technical overviews to drilldowns on special topics, get *free* ebooks from Microsoft Press at:

www.microsoftvirtualacademy.com/ebooks

Download your free ebooks in PDF, EPUB, and/or Mobi for Kindle formats.

Look for other great resources at Microsoft Virtual Academy, where you can learn new skills and help advance your career with free Microsoft training delivered by experts.

Microsoft Press



Now that
you've
read the
book...

Tell us what you think!

Was it useful?

Did it teach you what you wanted to learn?

Was there room for improvement?

Let us know at <http://aka.ms/tellpress>

Your feedback goes directly to the staff at Microsoft Press,
and we read every one of your responses. Thanks in advance!



Microsoft