

# AZ-204T00

Developing Solutions  
for Microsoft Azure

AZ-204T00

**Developing Solutions  
for Microsoft Azure**

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is

intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links may be provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

© 2019 Microsoft Corporation. All rights reserved.

Microsoft and the trademarks listed at <http://www.microsoft.com/trademarks><sup>1</sup> are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

<sup>1</sup> <http://www.microsoft.com/trademarks>

## **MICROSOFT INSTRUCTOR-LED COURSEWARE**

These license terms are an agreement between Microsoft Corporation (or based on where you live, one of its affiliates) and you. Please read them. They apply to your use of the content accompanying this agreement which includes the media on which you received it, if any. These license terms also apply to Trainer Content and any updates and supplements for the Licensed Content unless other terms accompany those items. If so, those terms apply.

**BY ACCESSING, DOWNLOADING OR USING THE LICENSED CONTENT, YOU ACCEPT THESE TERMS. IF YOU DO NOT ACCEPT THEM, DO NOT ACCESS, DOWNLOAD OR USE THE LICENSED CONTENT.**

**If you comply with these license terms, you have the rights below for each license you acquire.** 1. **DEFINITIONS.**

1. "Authorized Learning Center" means a Microsoft Imagine Academy (MSIA) Program Member, Microsoft Learning Competency Member, or such other entity as Microsoft may designate from time to time.
2. "Authorized Training Session" means the instructor-led training class using Microsoft Instructor-Led Courseware conducted by a Trainer at or through an Authorized Learning Center.
3. "Classroom Device" means one (1) dedicated, secure computer that an Authorized Learning Center owns or controls that is located at an Authorized Learning Center's training facilities that meets or exceeds the hardware level specified for the particular Microsoft Instructor-Led Courseware.
4. "End User" means an individual who is (i) duly enrolled in and attending an Authorized Training Session or Private Training Session, (ii) an employee of an MPN Member (defined below), or (iii) a Microsoft full-time employee, a Microsoft Imagine Academy (MSIA) Program Member, or a Microsoft Learn for Educators – Validated Educator.
5. "Licensed Content" means the content accompanying this agreement which may include the Microsoft Instructor-Led Courseware or Trainer Content.
6. "Microsoft Certified Trainer" or "MCT" means an individual who is (i) engaged to teach a training session to End Users on behalf of an Authorized Learning Center or MPN Member, and (ii) currently certified as a Microsoft Certified Trainer under the Microsoft Certification Program.
7. "Microsoft Instructor-Led Courseware" means the Microsoft-branded instructor-led training course that educates IT professionals, developers, students at an academic institution, and other learners on Microsoft technologies. A Microsoft Instructor-Led Courseware title may be branded as MOC, Microsoft Dynamics, or Microsoft Business Group courseware.
8. "Microsoft Imagine Academy (MSIA) Program Member" means an active member of the Microsoft Imagine Academy Program.
9. "Microsoft Learn for Educators – Validated Educator" means an educator who has been validated through the Microsoft Learn for Educators program as an active educator at a college, university, community college, polytechnic or K-12 institution.
10. "Microsoft Learning Competency Member" means an active member of the Microsoft Partner Network program in good standing that currently holds the Learning Competency status.
11. "MOC" means the "Official Microsoft Learning Product" instructor-led courseware known as Microsoft Official Course that educates IT professionals, developers, students at an academic institution, and other learners on Microsoft technologies.
12. "MPN Member" means an active Microsoft Partner Network program member in good standing.
13. "Personal Device" means one (1) personal computer, device, workstation or other digital electronic device that you personally own or control that meets or exceeds the hardware level specified for the particular Microsoft Instructor-Led Courseware.
14. "Private Training Session" means the instructor-led training classes provided by MPN Members for corporate customers to teach a predefined learning objective using Microsoft Instructor-Led

Courseware. These classes are not advertised or promoted to the general public and class attendance is restricted to individuals employed by or contracted by the corporate customer.

15. "Trainer" means (i) an academically accredited educator engaged by a Microsoft Imagine Academy Program Member to teach an Authorized Training Session, (ii) an academically accredited educator validated as a Microsoft Learn for Educators – Validated Educator, and/or (iii) a MCT.

16. "Trainer Content" means the trainer version of the Microsoft Instructor-Led Courseware and additional supplemental content designated solely for Trainers' use to teach a training session using the Microsoft Instructor-Led Courseware. Trainer Content may include Microsoft PowerPoint presentations, trainer preparation guide, train the trainer materials, Microsoft One Note packs, classroom setup guide and Pre-release course feedback form. To clarify, Trainer Content does not include any software, virtual hard disks or virtual machines.

2. **USE RIGHTS.** The Licensed Content is licensed, not sold. The Licensed Content is licensed on a **one copy per user basis**, such that you must acquire a license for each individual that accesses or uses the Licensed Content.

• 2.1 Below are five separate sets of use rights. Only one set of rights apply to you.

1. **If you are a Microsoft Imagine Academy (MSIA) Program Member:**

1. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.

2. For each license you acquire on behalf of an End User or Trainer, you may either:

1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User who is enrolled in the Authorized Training Session, and only immediately prior to the commencement of the Authorized Training Session that is the subject matter of the Microsoft Instructor-Led Courseware being provided, **or**
2. provide one (1) End User with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
3. provide one (1) Trainer with the unique redemption code and instructions on how they can access one (1) Trainer Content.

3. For each license you acquire, you must comply with the following:

1. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
2. you will ensure each End User attending an Authorized Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Authorized Training Session,
3. you will ensure that each End User provided with the hard-copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End

EULA **V**

User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,

4. you will ensure that each Trainer teaching an Authorized Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Authorized Training Session,

5. you will only use qualified Trainers who have in-depth knowledge of and experience with the Microsoft technology that is the subject of the Microsoft Instructor-Led Courseware being taught for all your Authorized Training Sessions,
6. you will only deliver a maximum of 15 hours of training per week for each Authorized Training Session that uses a MOC title, and
7. you acknowledge that Trainers that are not MCTs will not have access to all of the trainer resources for the Microsoft Instructor-Led Courseware.

**2. If you are a Microsoft Learning Competency Member:**

1. Each license acquired may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
2. For each license you acquire on behalf of an End User or MCT, you may either:
  1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User attending the Authorized Training Session and only immediately prior to the commencement of the Authorized Training Session that is the subject matter of the Microsoft Instructor-Led Courseware provided, **or**
  2. provide one (1) End User attending the Authorized Training Session with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
  3. you will provide one (1) MCT with the unique redemption code and instructions on how they can access one (1) Trainer Content.
3. For each license you acquire, you must comply with the following:
  1. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
  2. you will ensure that each End User attending an Authorized Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Authorized Training Session,
  3. you will ensure that each End User provided with a hard-copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,

**VI EULA**

4. you will ensure that each MCT teaching an Authorized Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Authorized Training Session,
5. you will only use qualified MCTs who also hold the applicable Microsoft Certification credential that is the subject of the MOC title being taught for all your Authorized Training Sessions using MOC,
6. you will only provide access to the Microsoft Instructor-Led Courseware to End Users, and
7. you will only provide access to the Trainer Content to MCTs.

**3. If you are a MPN Member:**

1. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instruc

tor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.

2. For each license you acquire on behalf of an End User or Trainer, you may either:

1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User attending the Private Training Session, and only immediately prior to the commencement of the Private Training Session that is the subject matter of the Microsoft Instructor-Led Courseware being provided, **or**
2. provide one (1) End User who is attending the Private Training Session with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
3. you will provide one (1) Trainer who is teaching the Private Training Session with the unique redemption code and instructions on how they can access one (1) Trainer Content.

3. For each license you acquire, you must comply with the following:

1. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
2. you will ensure that each End User attending an Private Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Private Training Session,
3. you will ensure that each End User provided with a hard copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,
4. you will ensure that each Trainer teaching an Private Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Private Training Session,

EULA VII

5. you will only use qualified Trainers who hold the applicable Microsoft Certification credential that is the subject of the Microsoft Instructor-Led Courseware being taught for all your Private Training Sessions,
6. you will only use qualified MCTs who hold the applicable Microsoft Certification credential that is the subject of the MOC title being taught for all your Private Training Sessions using MOC,
7. you will only provide access to the Microsoft Instructor-Led Courseware to End Users, and
8. you will only provide access to the Trainer Content to Trainers.

**4. If you are an End User:**

For each license you acquire, you may use the Microsoft Instructor-Led Courseware solely for your personal training use. If the Microsoft Instructor-Led Courseware is in digital format, you may access the Microsoft Instructor-Led Courseware online using the unique redemption code provided to you by the training provider and install and use one (1) copy of the Microsoft Instructor-Led Courseware on up to three (3) Personal Devices. You may also print one (1) copy of the Microsoft Instructor-Led Courseware. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.

**5. If you are a Trainer.**

1. For each license you acquire, you may install and use one (1) copy of the Trainer Content in the form provided to you on one (1) Personal Device solely to prepare and deliver an Authorized Training Session or Private Training Session, and install one (1) additional copy on another Personal Device as a backup copy, which may be used only to reinstall the Trainer Content. You may not install or use a copy of the Trainer Content on a device you do not own or control. You may also print one (1) copy of the Trainer Content solely to prepare for and deliver an Authorized Training Session or Private Training Session.
  2. If you are an MCT, you may customize the written portions of the Trainer Content that are logically associated with instruction of a training session in accordance with the most recent version of the MCT agreement.
  3. If you elect to exercise the foregoing rights, you agree to comply with the following: (i) customizations may only be used for teaching Authorized Training Sessions and Private Training Sessions, and (ii) all customizations will comply with this agreement. For clarity, any use of “customize” refers only to changing the order of slides and content, and/or not using all the slides or content, it does not mean changing or modifying any slide or content.
- **2.2 Separation of Components.** The Licensed Content is licensed as a single unit and you may not separate their components and install them on different devices.
  - **2.3 Redistribution of Licensed Content.** Except as expressly provided in the use rights above, you may not distribute any Licensed Content or any portion thereof (including any permitted modifications) to any third parties without the express written permission of Microsoft.
  - **2.4 Third Party Notices.** The Licensed Content may include third party code that Microsoft, the third party, licenses to you under this agreement. Notices, if any, for the third party code are included for your information only.
  - **2.5 Additional Terms.** Some Licensed Content may contain components with additional terms, conditions, and licenses regarding its use. Any non-conflicting terms in those conditions and licenses also apply to your use of that respective component and supplements the terms described in this agreement.

## VIII EULA

3. **LICENSED CONTENT BASED ON PRE-RELEASE TECHNOLOGY.** If the Licensed Content’s subject matter is based on a pre-release version of Microsoft technology (“**Pre-release**”), then in addition to the other provisions in this agreement, these terms also apply:
  1. **Pre-Release Licensed Content.** This Licensed Content subject matter is on the Pre-release version of the Microsoft technology. The technology may not work the way a final version of the technology will and we may change the technology for the final version. We also may not release a final version. Licensed Content based on the final version of the technology may not contain the same information as the Licensed Content based on the Pre-release version. Microsoft is under no obligation to provide you with any further content, including any Licensed Content based on the final version of the technology.
  2. **Feedback.** If you agree to give feedback about the Licensed Content to Microsoft, either directly or through its third party designee, you give to Microsoft without charge, the right to use, share and commercialize your feedback in any way and for any purpose. You also give to third parties, without charge, any patent rights needed for their products, technologies and services to use or interface with any specific parts of a Microsoft technology, Microsoft product, or service that includes the feedback. You will not give feedback that is subject to a license that requires Microsoft to license its technology, technologies, or products to third parties because we include your feedback in them. These rights survive this agreement.
  3. **Pre-release Term.** If you are an Microsoft Imagine Academy Program Member, Microsoft Learning Competency Member, MPN Member, Microsoft Learn for Educators – Validated Educator, or Trainer, you will cease using all copies of the Licensed Content on the Pre-release technology upon (i) the date which Microsoft informs you is the end date for using the Licensed Content on the Pre-release technology, or (ii) sixty (60) days after the commercial release of the technology that is the subject of the Licensed Content, whichever is earliest (“**Pre-release term**”). Upon expiration or termination of the Pre-release term, you will irretrievably delete and destroy all copies

of the Licensed Content in your possession or under your control.

4. **SCOPE OF LICENSE.** The Licensed Content is licensed, not sold. This agreement only gives you some rights to use the Licensed Content. Microsoft reserves all other rights. Unless applicable law gives you more rights despite this limitation, you may use the Licensed Content only as expressly permitted in this agreement. In doing so, you must comply with any technical limitations in the Licensed Content that only allows you to use it in certain ways. Except as expressly permitted in this agreement, you may not:

- access or allow any individual to access the Licensed Content if they have not acquired a valid license for the Licensed Content,
- alter, remove or obscure any copyright or other protective notices (including watermarks), branding or identifications contained in the Licensed Content,
- modify or create a derivative work of any Licensed Content,
- publicly display, or make the Licensed Content available for others to access or use,
- copy, print, install, sell, publish, transmit, lend, adapt, reuse, link to or post, make available or distribute the Licensed Content to any third party,
- work around any technical limitations in the Licensed Content, or
- reverse engineer, decompile, remove or otherwise thwart any protections or disassemble the Licensed Content except and only to the extent that applicable law expressly permits, despite this limitation.

5. **RESERVATION OF RIGHTS AND OWNERSHIP.** Microsoft reserves all rights not expressly granted to you in this agreement. The Licensed Content is protected by copyright and other intellectual property

EULA IX

laws and treaties. Microsoft or its suppliers own the title, copyright, and other intellectual property rights in the Licensed Content.

6. **EXPORT RESTRICTIONS.** The Licensed Content is subject to United States export laws and regulations. You must comply with all domestic and international export laws and regulations that apply to the Licensed Content. These laws include restrictions on destinations, end users and end use. For additional information, see [www.microsoft.com/exporting](http://www.microsoft.com/exporting).

7. **SUPPORT SERVICES.** Because the Licensed Content is provided “as is”, we are not obligated to provide support services for it.

8. **TERMINATION.** Without prejudice to any other rights, Microsoft may terminate this agreement if you fail to comply with the terms and conditions of this agreement. Upon termination of this agreement for any reason, you will immediately stop all use of and delete and destroy all copies of the Licensed Content in your possession or under your control.

9. **LINKS TO THIRD PARTY SITES.** You may link to third party sites through the use of the Licensed Content. The third party sites are not under the control of Microsoft, and Microsoft is not responsible for the contents of any third party sites, any links contained in third party sites, or any changes or updates to third party sites. Microsoft is not responsible for webcasting or any other form of transmission received from any third party sites. Microsoft is providing these links to third party sites to you only as a convenience, and the inclusion of any link does not imply an endorsement by Microsoft of the third party site.

10. **ENTIRE AGREEMENT.** This agreement, and any additional terms for the Trainer Content, updates and supplements are the entire agreement for the Licensed Content, updates and supplements.

11. **APPLICABLE LAW.**

1. United States. If you acquired the Licensed Content in the United States, Washington state law governs the interpretation of this agreement and applies to claims for breach of it, regardless of conflict of laws principles. The laws of the state where you live govern all other claims, including claims under state consumer protection laws, unfair competition laws, and in tort.
2. Outside the United States. If you acquired the Licensed Content in any other country, the laws of that country apply.



12. **LEGAL EFFECT.** This agreement describes certain legal rights. You may have other rights under the laws of your country. You may also have rights with respect to the party from whom you acquired the Licensed Content. This agreement does not change your rights under the laws of your country if the laws of your country do not permit it to do so.
13. **DISCLAIMER OF WARRANTY. THE LICENSED CONTENT IS LICENSED "AS-IS" AND "AS AVAILABLE." YOU BEAR THE RISK OF USING IT. MICROSOFT AND ITS RESPECTIVE AFFILIATES GIVES NO EXPRESS WARRANTIES, GUARANTEES, OR CONDITIONS. YOU MAY HAVE ADDITIONAL CONSUMER RIGHTS UNDER YOUR LOCAL LAWS WHICH THIS AGREEMENT CANNOT CHANGE. TO THE EXTENT PERMITTED UNDER YOUR LOCAL LAWS, MICROSOFT AND ITS RESPECTIVE AFFILIATES EXCLUDES ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.**
14. **LIMITATION ON AND EXCLUSION OF REMEDIES AND DAMAGES. YOU CAN RECOVER FROM MICROSOFT, ITS RESPECTIVE AFFILIATES AND ITS SUPPLIERS ONLY DIRECT DAMAGES UP TO US\$5.00. YOU CANNOT RECOVER ANY OTHER DAMAGES, INCLUDING CONSEQUENTIAL, LOST PROFITS, SPECIAL, INDIRECT OR INCIDENTAL DAMAGES.**

X EULA

This limitation applies to

- anything related to the Licensed Content, services, content (including code) on third party Internet sites or third-party programs; and
- claims for breach of contract, breach of warranty, guarantee or condition, strict liability, negligence, or other tort to the extent permitted by applicable law.

It also applies even if Microsoft knew or should have known about the possibility of the damages. The above limitation or exclusion may not apply to you because your country may not allow the exclusion or limitation of incidental, consequential, or other damages.

**Please note: As this Licensed Content is distributed in Quebec, Canada, some of the clauses in this agreement are provided below in French.**

**Remarque : Ce le contenu sous licence étant distribué au Québec, Canada, certaines des clauses dans ce contrat sont fournies ci-dessous en français.**

**EXONÉRATION DE GARANTIE.** Le contenu sous licence visé par une licence est offert « tel quel ». Toute utilisation de ce contenu sous licence est à votre seule risque et péril. Microsoft n'accorde aucune autre garantie expresse. Vous pouvez bénéficier de droits additionnels en vertu du droit local sur la protection des consommateurs, que ce contrat ne peut modifier. La ou elles sont permises par le droit locale, les garanties implicites de qualité marchande, d'adéquation à un usage particulier et d'absence de contre façon sont exclues.

**LIMITATION DES DOMMAGES-INTÉRÊTS ET EXCLUSION DE RESPONSABILITÉ POUR LES DOMMAGES.** Vous pouvez obtenir de Microsoft et de ses fournisseurs une indemnisation en cas de dommages directs uniquement à hauteur de 5,00 \$ US. Vous ne pouvez prétendre à aucune indemnisation pour les autres dommages, y compris les dommages spéciaux, indirects ou accessoires et pertes de bénéfices.

Cette limitation concerne:

- tout ce qui est relié au le contenu sous licence, aux services ou au contenu (y compris le code) figurant sur des sites Internet tiers ou dans des programmes tiers; et.
- les réclamations au titre de violation de contrat ou de garantie, ou au titre de responsabilité stricte, de négligence ou d'une autre faute dans la limite autorisée par la loi en vigueur.

Elle s'applique également, même si Microsoft connaissait ou devrait connaître l'éventualité d'un tel dommage. Si votre pays n'autorise pas l'exclusion ou la limitation de responsabilité pour les dommages indirects, accessoires ou de quelque nature que ce soit, il se peut que la limitation ou l'exclusion ci-dessus ne s'appliquera pas à votre égard.

**EFFET JURIDIQUE.** Le présent contrat décrit certains droits juridiques. Vous pourriez avoir d'autres droits prévus par les lois de votre pays. Le présent contrat ne modifie pas les droits que vous confèrent les lois de votre pays si celles-ci ne le permettent pas.

# Contents

■ <b>Module 0 Course introduction</b> . . . . .	<b>1</b> About this course . . . . .	<b>1</b>
■ <b>Module 1 Explore Azure App Service</b> . . . . .	<b>7</b> Explore Azure App Service . . . . .	<b>7</b>
	Configure web app settings . . . . .	
	<b>18</b> Scale apps in Azure App Service . . . . .	
	. . . <b>31</b> Explore Azure App Service deployment slots . . . . .	
	. . . <b>43</b>	
■ <b>Module 2 Implement Azure Functions</b> . . . . .	<b>53</b> Explore Azure Functions . . . . .	<b>53</b>
	Develop Azure Functions . . . . .	
	<b>59</b> Implement Durable Functions . . . . .	
	. . . <b>71</b>	
■ <b>Module 3 Develop solutions that use Blob storage</b> . . . . .	<b>87</b> Explore Azure Blob storage . . . . .	<b>87</b>
	Manage the Azure Blob storage lifecycle . . . . .	
	<b>95</b> Work with Azure Blob storage . . . . .	
	. . . <b>103</b>	
■ <b>Module 4 Develop solutions that use Azure Cosmos DB</b> . . . . .	<b>117</b> Explore Azure Cosmos DB . . . . .	<b>117</b>
	Implement partitioning in Azure Cosmos DB . . . . .	
	<b>128</b> Work with Azure Cosmos DB . . . . .	
	. . . <b>133</b>	
■ <b>Module 5 Implement infrastructure as a service solutions</b> . . . . .	<b>149</b> Provision virtual machines in Azure . . . . .	<b>149</b>
	<b>149</b> Create and deploy Azure Resource Manager templates . . . . .	
	. . . <b>157</b> Manage container images in Azure Container Registry . . . . .	
	. . . <b>170</b> Run container images in Azure Container Instances . . . . .	
	. . . <b>179</b>	
■ <b>Module 6 Implement user authentication and authorization</b> . . . . .	<b>191</b> Explore the Microsoft identity platform . . . . .	<b>191</b>
	Implement authentication by using the Microsoft Authentication Library . . . . .	<b>198</b>
	Implement shared access signatures . . . . .	<b>207</b> Explore Microsoft Graph . . . . .
	. . . <b>212</b>	
■ <b>Module 7 Implement secure cloud solutions</b> . . . . .	<b>223</b> Implement Azure Key Vault . . . . .	
	. . . <b>223</b> Implement managed identities . . . . .	<b>229</b>
	Implement Azure App Configuration . . . . .	<b>238</b>
■ <b>Module 8 Implement API Management</b> . . . . .	<b>249</b> Explore API Management . . . . .	<b>249</b>
■ <b>Module 9 Develop event-based solutions</b> . . . . .	<b>269</b> Explore Azure Event Grid . . . . .	<b>269</b>
	Explore Azure Event Hubs . . . . .	
	<b>284</b>	
■ <b>Module 10 Develop message-based solutions</b> . . . . .	<b>297</b> Discover Azure message queues . . . . .	
	<b>297</b>	

■ <b>Module 11 Instrument solutions to support monitoring and logging</b> .....	<b>317</b>	Monitor app performance .....	<b>317</b>
■ <b>Module 12 Integrate caching and content delivery within solutions</b> .....	<b>329</b>	Develop for Azure Cache for Redis .....	<b>329</b>
Develop for storage on CDNs .....	<b>342</b>		

# Module 0 Course introduction

## About this course

## About this course

Welcome to the **Developing Solutions for Microsoft Azure** course. This course teaches developers how to create solutions that are hosted in, and utilize, Azure services.

**Level:** Intermediate

## Audience

This course is for Azure Developers. They design and build cloud solutions such as applications and services. They participate in all phases of development, from solution design, to development and deployment, to testing and maintenance. They partner with cloud solution architects, cloud DBAs, cloud administrators, and clients to implement the solution.

## Prerequisites

This course assumes you have already acquired the following skills and experience:

- At least one year of experience developing scalable solutions through all phases of software development.
- Be skilled in at least one cloud-supported programming language. Much of the course focuses on C#, .NET Framework, HTML, and using REST in applications.
- Have a base understanding of Azure and cloud concepts, services, and the Azure Portal. If you need to ramp up you can start with the **Azure Fundamentals**<sup>1</sup> course which is freely available.

<sup>1</sup> <https://docs.microsoft.com/learn/paths/azure-fundamentals/>

## Labs and exercises

The labs and demonstrations in this course are designed to be performed in the lab environments provided to the students as part of the course.

Many of the exercises include links to tools you may need to install if you want to practice the on your own.

# Course syllabus

The course content includes a mix of content, exercises, hands-on labs, and reference links.

## Module 01: Create Azure App Service web apps

In this module you will learn how Azure App Service functions and how to create and update an app. You will also explore App Service authentication and authorization, how to configure app settings, scale apps, and how to use deployment slots. This module includes:

- Explore Azure App Service
- Configure web app settings
- Scale apps in Azure App Service
- Explore Azure App Service deployment slots
- Lab 01: Build a web application on Azure platform as a service offerings

## Module 02: Implement Azure Functions

In this module you will learn how to create and deploy Azure Functions. Explore hosting options, bindings, triggers, and how to use Durable Functions to define stateful workflows. This module includes:

- Explore Azure Functions
- Develop Azure Functions
- Implement Durable Functions
- Lab 02: Implement task processing logic by using Azure Functions

## Module 03: Develop solutions that use Blob storage

In this module you will learn how to create Azure Blob storage resources, manage data through the blob storage lifecycle, and work with containers and items by using the Azure Blob storage client library V12 for .NET. This module includes:

- Explore Azure Blob storage
- Manage the Azure Blob storage lifecycle
- Work with Azure Blob storage
- Lab 03: Retrieve Azure Storage resources and metadata by using the Azure Storage SDK for .NET

3

## Module 04: Develop solutions that use Azure Cosmos DB

In this module you will learn how to create Azure Cosmos DB resources with the appropriate consistency levels, choose and create a partition key, and perform data operations by using the .NET SDK V3 for Azure Cosmos DB. This module includes:

- Explore Azure Cosmos DB
- Implement partitioning in Azure Cosmos DB
- Work with Azure Cosmos DB
- Lab 04: Construct a polyglot data solution

## Module 05: Implement infrastructure as a service solutions

In this module you will learn how to create and deploy virtual machine, deploy resources using Azure Resource Manager templates, and manage and deploy containers. This module includes:

- Provision virtual machine in Azure
- Create and deploy Azure Resource Manager templates
- Manage container images in Azure Container Registry
- Run container images in Azure Container Instances
- Lab 05: Deploy compute workloads by using images and containers

## Module 06: Implement user authentication and authorization

In this module you will learn how to implement authentication and authorization to resources by using the Microsoft identity platform, Microsoft Authentication Library, shared access signatures, and use Microsoft Graph. This module includes:

- Explore the Microsoft identity platform
- Implement authentication by using the Microsoft Authentication Library
- Implement shared access signatures
- Explore Microsoft Graph
- Lab 06: Authenticate by using OpenID Connect, MSAL, and .NET SDKs

## Module 07: Implement secure cloud solutions

In this module you will learn how to more securely deploy apps in Azure by using Azure Key Vault, managed identities, and Azure App Configuration. This module includes:

- Implement Azure Key Vault
- Implement managed identities
- Implement Azure App Configuration
- Lab 07: Access resource secrets more securely across services

4

## Module 08: Implement API Management

In this module you will learn how the API Management service functions, how to transform and secure APIs, and how to create a backend API. This module includes:

- Explore API Management
- Lab 08: Create a multi-tier solution by using Azure services

## Module 09: Develop event-based solutions

In this module you will learn how to build applications with event-based architectures by integrating Azure Event Grid and Azure Event Hubs in to your solution. This module includes:

- Explore Azure Event Grid
- Explore Azure Event Hubs
- Lab 09: Publish and subscribe to Event Grid events

## Module 10: Develop message-based solutions

In this module you will learn how to build applications with message-based architectures by integrating Azure Service Bus and Azure Queue Storage in to your solution. This module includes:

- Discover Azure message queues

- Lab 10: Asynchronously process messages by using Azure Service Bus Queues

## Module 11: Instrument solutions to support monitoring and logging

In this module you will learn how to instrument apps to enable Application Insights to monitor performance and help troubleshoot issues. This module includes:

- Monitor app performance
- Lab 11: Monitor services that are deployed to Azure

## Module 12: Integrate caching and content delivery within solutions

In this module you will learn how to improve the performance and scalability of your applications by integrating Azure Cache for Redis and Azure Content Delivery Network in to your solution. This module includes:

- Develop for Azure Cache for Redis
- Develop for storage on CDNs
- Lab 12: Enhance a web application by using the Azure Content Delivery Network

5

## Certification exam preparation

This course helps you prepare for the **AZ-204: Developing Solutions for Microsoft Azure**<sup>2</sup> certification exam.

AZ-204 includes six study areas, as shown in the table. The percentages indicate the relative weight of each area on the exam. The higher the percentage, the more questions you are likely to see in that area.

AZ-204 Study Areas	Weight
Develop Azure compute solutions	25-30%
Develop for Azure storage	15-20%
Implement Azure security	20-25%
Monitor, troubleshoot, and optimize Azure solutions	15-20%
Connect to and consume Azure and third-party services	15-20%

**Note:** The relative weightings are subject to change. For the latest information visit the **exam page**<sup>3</sup> and review the Skills measured section.

Passing the exam will earn you the **Microsoft Certified: Azure Developer Associate**<sup>4</sup> certification.

The modules in the course are mapped to the objectives listed in each study area on the Skills Measured section of the exam page to make it easier for you to focus on areas of the exam you choose to revisit.

## Course resources

There are a lot of resources to help you learn about Azure. We recommend you bookmark these pages. • **Azure Community Support**<sup>5</sup>

- **Azure Documentation**<sup>6</sup>
- **Microsoft Azure Blog**<sup>7</sup>
- **Microsoft Learn**<sup>8</sup>
- **Microsoft Learn Blog**<sup>9</sup>

2 <https://docs.microsoft.com/en-us/learn/certifications/exams/az-204>

3 <https://docs.microsoft.com/learn/certifications/exams/az-204>

4 <https://docs.microsoft.com/learn/certifications/azure-developer>

5 <https://azure.microsoft.com/support/community/>

6 <https://docs.microsoft.com/azure/>

7 <https://azure.microsoft.com/blog/>

8 <https://docs.microsoft.com/learn/>

9 <https://techcommunity.microsoft.com/t5/microsoft-learn-blog/bg-p/MicrosoftLearnBlog>

# Module 1 Explore Azure App Service

## Explore Azure App Service

### Introduction

Your company is considering moving their web apps to the cloud and has asked you to evaluate Azure App Service.

### Learning objectives

After completing this module, you'll be able to:

- Describe Azure App Service key components and value.
- Explain how Azure App Service manages authentication and authorization.
- Identify methods to control inbound and outbound traffic to your web app.
- Deploy an app to App Service using Azure CLI commands.

## Examine Azure App Service

Azure App Service is an HTTP-based service for hosting web applications, REST APIs, and mobile back ends. You can develop in your favorite language, be it .NET, .NET Core, Java, Ruby, Node.js, PHP, or Python. Applications run and scale with ease on both Windows and Linux-based environments.

## Built-in auto scale support

Baked into Azure App Service is the ability to scale up/down or scale out/in. Depending on the usage of the web app, you can scale your app up/down the resources of the underlying machine that is hosting your web app. Resources include the number of cores or the amount of RAM available. Scaling out/in is the ability to increase, or decrease, the number of machine instances that are running your web app.

8

## Continuous integration/deployment support

The Azure portal provides out-of-the-box continuous integration and deployment with Azure DevOps, GitHub, Bitbucket, FTP, or a local Git repository on your development machine. Connect your web app with any of the above sources and App Service will do the rest for you by auto-syncing code and any future changes on the code into the web app.

## Deployment slots

Using the Azure portal, or command-line tools, you can easily add deployment slots to an App Service web app. For instance, you can create a staging deployment slot where you can push your code to test on Azure. Once you are happy with your code, you can easily swap the staging deployment slot with the production slot. You do all this with a few simple mouse clicks in the Azure portal.

**Note:** Deployment slots are only available in the Standard and Premium plan tiers.

## App Service on Linux

App Service can also host web apps natively on Linux for supported application stacks. It can also run custom Linux containers (also known as Web App for Containers). App Service on Linux supports a number of language specific built-in images. Just deploy your code. Supported languages include: Node.js, Java (JRE 8 & JRE 11), PHP, Python, .NET Core, and Ruby. If the runtime your application requires is not supported in the built-in images, you can deploy it with a custom container.

The languages, and their supported versions, are updated on a regular basis. You can retrieve the current list by using the following command in the Cloud Shell.

```
az webapp list-runtimes --linux
```

## Limitations

App Service on Linux does have some limitations:

- App Service on Linux is not supported on Shared pricing tier.
- You can't mix Windows and Linux apps in the same App Service plan.
- Historically, you could not mix Windows and Linux apps in the same resource group. However, all resource groups created on or after January 21, 2021 do support this scenario. Support for resource groups created before January 21, 2021 will be rolled out across Azure regions (including National cloud regions) soon.
- The Azure portal shows only features that currently work for Linux apps. As features are enabled, they're activated on the portal.

## Examine Azure App Service plans

In App Service, an app (Web Apps, API Apps, or Mobile Apps) always runs in an *App Service plan*. An App Service plan defines a set of compute resources for a web app to run. One or more apps can be configured to run on the same computing resources (or in the same App Service plan). In addition, Azure Functions also has the option of running in an App Service plan.



When you create an App Service plan in a certain region (for example, West Europe), a set of compute resources is created for that plan in that region. Whatever apps you put into this App Service plan run on these compute resources as defined by your App Service plan. Each App Service plan defines:

- Region (West US, East US, etc.)
- Number of VM instances
- Size of VM instances (Small, Medium, Large)
- Pricing tier (Free, Shared, Basic, Standard, Premium, PremiumV2, PremiumV3, Isolated)

The *pricing tier* of an App Service plan determines what App Service features you get and how much you pay for the plan. There are a few categories of pricing tiers:

- **Shared compute:** Both **Free** and **Shared** share the resource pools of your apps with the apps of other customers. These tiers allocate CPU quotas to each app that runs on the shared resources, and the resources can't scale out.
- **Dedicated compute:** The **Basic**, **Standard**, **Premium**, **PremiumV2**, and **PremiumV3** tiers run apps on dedicated Azure VMs. Only apps in the same App Service plan share the same compute resources. The higher the tier, the more VM instances are available to you for scale-out.
- **Isolated:** This tier runs dedicated Azure VMs on dedicated Azure Virtual Networks. It provides network isolation on top of compute isolation to your apps. It provides the maximum scale-out capabilities.
- **Consumption:** This tier is only available to *function apps*. It scales the functions dynamically depending on workload.

**Note:** App Service Free and Shared (preview) hosting plans are base tiers that run on the same Azure virtual machines as other App Service apps. Some apps might belong to other customers. These tiers are intended to be used only for development and testing purposes.

## How does my app run and scale?

In the **Free** and **Shared** tiers, an app receives CPU minutes on a shared VM instance and can't scale out. In other tiers, an app runs and scales as follows:

- An app runs on all the VM instances configured in the App Service plan.
- If multiple apps are in the same App Service plan, they all share the same VM instances.
- If you have multiple deployment slots for an app, all deployment slots also run on the same VM instances.
- If you enable diagnostic logs, perform backups, or run WebJobs, they also use CPU cycles and memory on these VM instances.

In this way, the App Service plan is the **scale unit** of the App Service apps. If the plan is configured to run five VM instances, then all apps in the plan run on all five instances. If the plan is configured for autoscaling, then all apps in the plan are scaled out together based on the autoscale settings.

## What if my app needs more capabilities or features?

Your App Service plan can be scaled up and down at any time. It is as simple as changing the pricing tier of the plan. If your app is in the same App Service plan with other apps, you may want to improve the app's performance by isolating the compute resources. You can do it by moving the app into a separate App Service plan.

You can potentially save money by putting multiple apps into one App Service plan. However, since apps in the same App Service plan all share the same compute resources you need to understand the capacity of the existing App Service plan and the expected load for the new app.

Isolate your app into a new App Service plan when:

- The app is resource-intensive.
- You want to scale the app independently from the other apps in the existing plan.
- The app needs resource in a different geographical region.

This way you can allocate a new set of resources for your app and gain greater control of your

## apps. **Deploy to App Service**

Every development team has unique requirements that can make implementing an efficient deployment pipeline difficult on any cloud service. App Service supports both automated and manual deployment.

### **Automated deployment**

Automated deployment, or continuous integration, is a process used to push out new features and bug fixes in a fast and repetitive pattern with minimal impact on end users.

Azure supports automated deployment directly from several sources. The following options are available:

- **Azure DevOps:** You can push your code to Azure DevOps, build your code in the cloud, run the tests, generate a release from the code, and finally, push your code to an Azure Web App.
- **GitHub:** Azure supports automated deployment directly from GitHub. When you connect your GitHub repository to Azure for automated deployment, any changes you push to your production branch on GitHub will be automatically deployed for you.
- **Bitbucket:** With its similarities to GitHub, you can configure an automated deployment with Bitbucket.

### **Manual deployment**

There are a few options that you can use to manually push your code to Azure:

- **Git:** App Service web apps feature a Git URL that you can add as a remote repository. Pushing to the remote repository will deploy your app.
- **CLI:** `webapp up` is a feature of the `az` command-line interface that packages your app and deploys it. Unlike other deployment methods, `az webapp up` can create a new App Service web app for you if you haven't already created one.
- **Zip deploy:** Use `curl` or a similar HTTP utility to send a ZIP of your application files to App Service.
- **FTP/S:** FTP or FTPS is a traditional way of pushing your code to many hosting environments, including App Service.

### **Use deployment slots**

Whenever possible, use deployment slots when deploying a new production build. When using a Standard App Service Plan tier or better, you can deploy your app to a staging environment and then swap your staging and production slots. The swap operation warms up the necessary worker instances to match your production scale, thus eliminating downtime.

11

## **Explore authentication and authorization in App Service**

Azure App Service provides built-in authentication and authorization support, so you can sign in users and access data by writing minimal or no code in your web app, API, and mobile back end, and also Azure Functions.

## Why use the built-in authentication?

You're not required to use App Service for authentication and authorization. Many web frameworks are bundled with security features, and you can use them if you like. If you need more flexibility than App Service provides, you can also write your own utilities.

The built-in authentication feature for App Service and Azure Functions can save you time and effort by providing out-of-the-box authentication with federated identity providers, allowing you to focus on the rest of your application.

- Azure App Service allows you to integrate a variety of auth capabilities into your web app or API without implementing them yourself.
- It's built directly into the platform and doesn't require any particular language, SDK, security expertise, or even any code to utilize.
- You can integrate with multiple login providers. For example, Azure AD, Facebook, Google, Twitter.

## Identity providers

App Service uses federated identity, in which a third-party identity provider manages the user identities and authentication flow for you. The following identity providers are available by default:

Provider	Sign-in endpoint	How-To guidance
Microsoft Identity Platform	<code>/.auth/login/aad</code>	<b>App Service Microsoft Identity Platform login</b> ( <a href="https://docs.microsoft.com/azure/app-service/configure-authentication-provider-aad">https://docs.microsoft.com/azure/app-service/configure-authentication-provider-aad</a> )
Facebook	<code>/.auth/login/facebook</code>	<b>App Service Facebook login</b> ( <a href="https://docs.microsoft.com/azure/app-service/configure-authentication-provider-facebook">https://docs.microsoft.com/azure/app-service/configure-authentication-provider-facebook</a> )
Google	<code>/.auth/login/google</code>	<b>App Service Google login</b> ( <a href="https://docs.microsoft.com/azure/app-service/configure-authentication-provider-google">https://docs.microsoft.com/azure/app-service/configure-authentication-provider-google</a> )
Twitter	<code>/.auth/login/twitter</code>	<b>App Service Twitter login</b> ( <a href="https://docs.microsoft.com/azure/app-service/configure-authentication-provider-twitter">https://docs.microsoft.com/azure/app-service/configure-authentication-provider-twitter</a> )

Provider	Sign-in endpoint	How-To guidance
----------	------------------	-----------------

Any OpenID Connect provider	<code>/.auth/login/&lt;provider Name&gt;</code>	<b>App Service OpenID Connect login</b> ( <a href="https://docs.microsoft.com/azure/app-service/configure-authentication-provider-openid-connect">https://docs.microsoft.com/azure/app-service/configure-authentication-provider-openid-connect</a> )
-----------------------------	---	--

When you enable authentication and authorization with one of these providers, its sign-in endpoint is available for user authentication and for validation of authentication tokens from the provider. You can provide your users with any number of these sign-in options.

## How it works

The authentication and authorization module runs in the same sandbox as your application code. When it's enabled, every incoming HTTP request passes through it before being handled by your application code. This module handles several things for your app:

- Authenticates users with the specified provider
- Validates, stores, and refreshes tokens
- Manages the authenticated session
- Injects identity information into request headers

The module runs separately from your application code and is configured using app settings. No SDKs, specific languages, or changes to your application code are required.

**Note:** In Linux and containers the authentication and authorization module runs in a separate container, isolated from your application code. Because it does not run in-process, no direct integration with specific language frameworks is possible.

## Authentication flow

The authentication flow is the same for all providers, but differs depending on whether you want to sign in with the provider's SDK.

- Without provider SDK: The application delegates federated sign-in to App Service. This is typically the case with browser apps, which can present the provider's login page to the user. The server code manages the sign-in process, so it is also called *server-directed flow* or *server flow*.
- With provider SDK: The application signs users in to the provider manually and then submits the authentication token to App Service for validation. This is typically the case with browser-less apps, which can't present the provider's sign-in page to the user. The application code manages the sign-in process, so it is also called *client-directed flow* or *client flow*. This applies to REST APIs, Azure Functions, JavaScript browser clients, and native mobile apps that sign users in using the provider's SDK.

The table below shows the steps of the authentication flow.

Step	Without provider SDK	With provider SDK
Sign user in	Redirects client to <code>/.auth/login/&lt;provider&gt;</code> .	Client code signs user in directly with provider's SDK and receives an authentication token. For information, see the provider's documentation.

Step	Without provider SDK	With provider SDK
Post-authentication	Provider redirects client to <code>/.auth/login/&lt;provider&gt;/callback</code> .	Client code posts token from provider to <code>/.auth/login/&lt;provider&gt;</code> for validation.
Establish authenticated session	App Service adds authenticated cookie to response.	App Service returns its own authentication token to client code.
Serve authenticated content	Client includes authentication cookie in subsequent requests (automatically handled by browser).	Client code presents authentication token in <code>X-ZUMO-AUTH</code> header (automatically handled by Mobile Apps client SDKs).

For client browsers, App Service can automatically direct all unauthenticated users to `/.auth/login/<provider>`. You can also present users with one or more `/.auth/login/<provider>` links to sign in to your app using their provider of choice.

## Authorization behavior

In the Azure portal, you can configure App Service with a number of behaviors when an incoming request is not authenticated.

- **Allow unauthenticated requests:** This option defers authorization of unauthenticated traffic to your application code. For authenticated requests, App Service also passes along authentication information in the HTTP headers. This option provides more flexibility in handling anonymous requests. It lets you present multiple sign-in providers to your users.
- **Require authentication:** This option will reject any unauthenticated traffic to your application. This rejection can be a redirect action to one of the configured identity providers. In these cases, a browser client is redirected to `/.auth/login/<provider>` for the provider you choose. If the anonymous request comes from a native mobile app, the returned response is an HTTP 401 Unauthorized. You can also configure the rejection to be an HTTP 401 Unauthorized or HTTP 403 Forbidden for all requests.

**Caution:** Restricting access in this way applies to all calls to your app, which may not be desirable for apps wanting a publicly available home page, as in many single-page applications.

## Discover App Service networking features

By default, apps hosted in App Service are accessible directly through the internet and can reach only internet-hosted endpoints. But for many applications, you need to control the inbound and outbound network traffic.

There are two main deployment types for Azure App Service. The multitenant public service hosts App Service plans in the Free, Shared, Basic, Standard, Premium, PremiumV2, and PremiumV3 pricing SKUs. There is also the single-tenant App Service Environment (ASE) hosts Isolated SKU App Service plans directly in your Azure virtual network.

## Multitenant App Service networking features

Azure App Service is a distributed system. The roles that handle incoming HTTP or HTTPS requests are called *front ends*. The roles that host the customer workload are called *workers*. All the roles in an App

Service deployment exist in a multitenant network. Because there are many different customers in the same App Service scale unit, you can't connect the App Service network directly to your network.

Instead of connecting the networks, you need features to handle the various aspects of application communication. The features that handle requests to your app can't be used to solve problems when you're making calls from your app. Likewise, the features that solve problems for calls from your app can't be used to solve problems to your app.

Inbound features	Outbound features
App-assigned address	Hybrid Connections
Access restrictions	Gateway-required VNet Integration
Service endpoints	VNet Integration
Private endpoints	

You can mix the features to solve your problems with a few exceptions. The following inbound use cases are examples of how to use App Service networking features to control traffic inbound to your app.

Inbound use case	Feature
Support IP-based SSL needs for your app	App-assigned address
Support unshared dedicated inbound address for your app	App-assigned address
Restrict access to your app from a set of well-defined addresses	Access restrictions

## Default networking behavior

Azure App Service scale units support many customers in each deployment. The Free and Shared SKU plans host customer workloads on multitenant workers. The Basic and higher plans host customer workloads that are dedicated to only one App Service plan. If you have a Standard App Service plan, all the apps in that plan will run on the same worker. If you scale out the worker, all the apps in that App Service plan will be replicated on a new worker for each instance in your App Service plan.

## Outbound addresses

The worker VMs are broken down in large part by the App Service plans. The Free, Shared, Basic, Standard, and Premium plans all use the same worker VM type. The PremiumV2 plan uses another VM type. PremiumV3 uses yet another VM type. When you change the VM family, you get a different set of outbound addresses. If you scale from Standard to PremiumV2, your outbound addresses will change. If you scale from PremiumV2 to PremiumV3, your outbound addresses will change. In some older scale units, both the inbound and outbound addresses will change when you scale from Standard to PremiumV2.

There are a number of addresses that are used for outbound calls. The outbound addresses used by your app for making outbound calls are listed in the properties for your app. These addresses are shared by all the apps running on the same worker VM family in the App Service deployment. If you want to see all the addresses that your app might use in a scale unit, there's property called `possibleOutboundAddresses` that will list them.

## Find outbound IPs

To find the outbound IP addresses currently used by your app in the Azure portal, click **Properties** in your app's left-hand navigation.

15

You can find the same information by running the following command in the Cloud Shell. They are listed in the **Additional Outbound IP Addresses** field.

```
az webapp show \
  --resource-group <group_name> \
  --name <app_name> \
  --query outboundIpAddresses \
  --output tsv
```

To find all possible outbound IP addresses for your app, regardless of pricing tiers, run the following command in the Cloud Shell.

```
az webapp show \
  --resource-group <group_name> \
  --name <app_name> \
  --query possibleOutboundIpAddresses \
  --output tsv
```

## Exercise: Create a static HTML web app by using Azure Cloud Shell

After gathering information about App Service you've decided to create and update a simple web app to try it out. In this exercise you'll deploy a basic HTML+CSS site to Azure App Service by using the Azure CLI `az webapp up` command. You will then update the code and redeploy it by using the same command.

The `az webapp up` command makes it easy to create and update web apps. When executed it performs the following actions:

- Create a default resource group.
- Create a default app service plan.
- Create an app with the specified name.
- Zip deploy files from the current working directory to the web app.

## Prerequisites

Before you begin make sure you have the following requirements in place:

- An Azure account with an active subscription. If you don't already have one, you can sign up for a free trial at <https://azure.com/free>.

## Login to Azure and download the sample

**app** 1. Login to the **Azure portal**<sup>1</sup> and open the Cloud Shell.



2. After the shell opens be sure to select the **Bash** environment.



3. In the Cloud Shell, create a directory and then navigate to it.

```
mkdir htmlapp
```

```
cd htmlapp
```

4. Run the following `git` command to clone the sample app repository to your `htmlapp` directory.

```
git clone https://github.com/Azure-Samples/html-docs-hello-world.git
```

## Create the web app

1. Change to the directory that contains the sample code and run the `az webapp up` command. In the following example, replace `<myAppName>` with a unique app name, and `<myLocation>` with a region near you.

```
cd html-docs-hello-world
```

```
az webapp up --location <myLocation> --name <myAppName> --html
```

This command may take a few minutes to run. While running, it displays information similar to the example below. Make a note of the `resourceGroup` value. You need it for the *Clean up resources* section later.

```
{
  "app_url": "https://<myAppName>.azurewebsites.net",
  "location": "westeurope",
  "name": "<app_name>",
  "os": "Windows",
  "resourcegroup": "<resource_group_name>",
  "serverfarm": "appsvc_asp_Windows_westeurope",
  "sku": "FREE",
  "src_path": "/home/<username>/demoHTML/html-docs-hello-world ",
  < JSON data removed for brevity. >
}
```

2. Open a browser and navigate to the app URL (`http://<myAppName>.azurewebsites.net`) and verify the app is running - take note of the title at the top of the page. Leave the browser open on the app for the next section.

## Update and redeploy the app

1. In the Cloud Shell, type `code index.html` to open the editor. In the `<h1>` heading tag, change *Azure App Service - Sample Static HTML Site* to *Azure App Service Updated* - or to anything else that you'd like.
2. Use the commands **ctrl-s** to save and **ctrl-q** to exit.

3. Redeploy the app with the same `az webapp up` command. Be sure to use the same values



for <myLocation> and <myAppName> as you used earlier.

```
az webapp up --location <myLocation> --name <myAppName> --html
```

**Tip:** You can use the up arrow on your keyboard to scroll through previous commands.

4. Once deployment is completed switch back to the browser from step 2 in the “Create the web app” section above and refresh the page.

## Clean up resources

If you no longer need the resources you created in this exercise you can delete the resource group using the `az group delete` command below. Replace <resource\_group> with resource group name you noted in step 1 of the “Create the web app” section above.

```
az group delete --name <resource_group> --no-wait
```

## Knowledge check

### Multiple choice

*Which of the following App Service plans supports only function apps?*

- Ⓔ Dedicated
- Ⓔ Isolated
- Ⓔ Consumption

### Multiple choice

*Which of the following networking features of App Service can be used to control outbound network traffic?* Ⓔ App-assigned address

- Ⓔ Hybrid Connections
- Ⓔ Service endpoints

## Summary

In this module, you learned how to:

- Describe Azure App Service key components and value.
- Explain how Azure App Service manages authentication and authorization.
- Identify methods to control inbound and outbound traffic to your web app.
- Deploy an app to App Service using Azure CLI commands.

18

## Configure web app settings

### Introduction

In App Service, app settings are variables passed as environment variables to the application code.

### Learning objectives

After completing this module, you'll be able to:

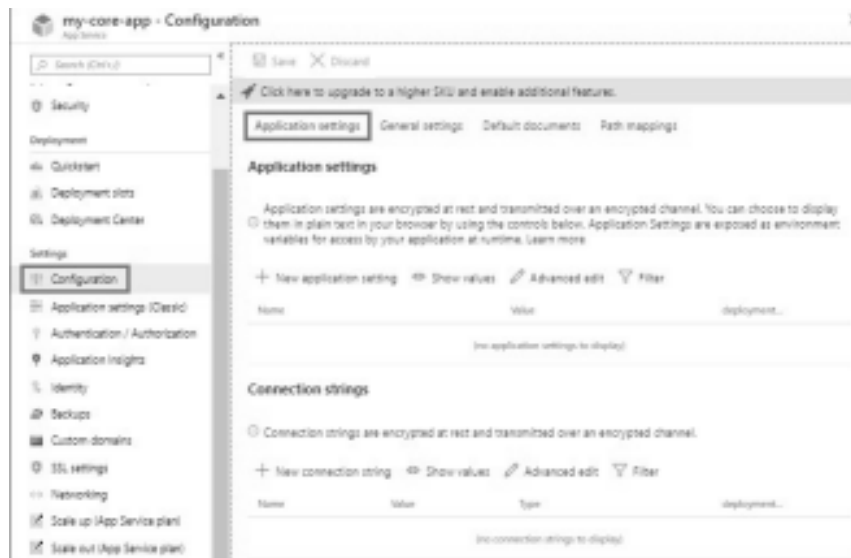
- Create application settings that are bound to deployment slots.

- Explain the options for installing SSL/TLS certificates for your app.
- Enable diagnostic logging for your app to aid in monitoring and debugging.
- Create virtual app to directory mappings.

## Configure application settings

In App Service, app settings are variables passed as environment variables to the application code. For Linux apps and custom containers, App Service passes app settings to the container using the `--env` flag to set the environment variable in the container.

Application settings can be accessed by navigating to your app's management page and selecting **Configuration > Application Settings**.



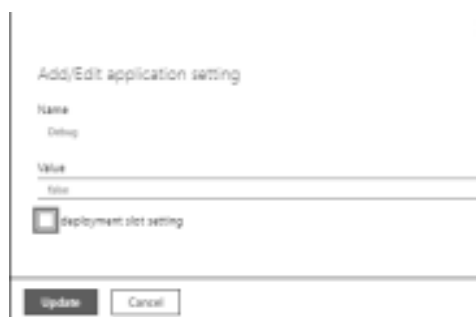
For ASP.NET and ASP.NET Core developers, setting app settings in App Service are like setting them in `<appSettings>` in *Web.config* or *appsettings.json*, but the values in App Service override the ones in *Web.config* or *appsettings.json*. You can keep development settings (for example, local MySQL password) in *Web.config* or *appsettings.json*, but production secrets (for example, Azure MySQL database password) safe in App Service. The same code uses your development settings when you debug locally, and it uses your production secrets when deployed to Azure.

App settings are always encrypted when stored (encrypted-at-rest).

19

## Adding and editing settings

To add a new app setting, click **New application setting**. If you are using deployment slots you can specify if your setting is swappable or not. In the dialog, you can stick the setting to the current slot.



To edit a setting, click the **Edit** button on the right side.

When finished, click **Update**. Don't forget to click **Save** back in the **Configuration** page.

**Note:** In a default, or custom, Linux container any nested JSON key structure in the app setting name like `ApplicationInsights:InstrumentationKey` needs to be configured in App Service as

ApplicationInsights\_\_InstrumentationKey for the key name. In other words, any : should be replaced by \_\_ (double underscore).

## Editing application settings in bulk

To add or edit app settings in bulk, click the **Advanced** edit button. When finished, click **Update**. App settings have the following JSON formatting:

```
[
  {
    "name": "<key-1>",
    "value": "<value-1>",
    "slotSetting": false
  },
  {
    "name": "<key-2>",
    "value": "<value-2>",
    "slotSetting": false
  },
  ...
]
```

## Configure connection strings

For ASP.NET and ASP.NET Core developers the values you set in App Service override the ones in *Web.config*. For other language stacks, it's better to use app settings instead, because connection strings require special formatting in the variable keys in order to access the values. Connection strings are always encrypted when stored (encrypted-at-rest).

20

**Tip:** There is one case where you may want to use connection strings instead of app settings for non-.NET languages: certain Azure database types are backed up along with the app only if you configure a connection string for the database in your App Service app.

Adding and editing connection strings follow the same principles as other app settings and they can also be tied to deployment slots. Below is an example of connection strings in JSON formatting that you would use for bulk adding or editing.

```
[
  {
    "name": "name-1",
    "value": "conn-string-1",
    "type": "SQLServer",
    "slotSetting": false
  },
  {
    "name": "name-2",
    "value": "conn-string-2",
    "type": "PostgreSQL",
    "slotSetting": false
  },
  ...
]
```

## Configure general settings

In the **Configuration > General settings** section you can configure some common settings for your

app. Some settings require you to scale up to higher pricing tiers.

Below is a list of the currently available settings:

- **Stack settings:** The software stack to run the app, including the language and SDK versions. For Linux apps and custom container apps, you can also set an optional start-up command or file.



- **Platform settings:** Lets you configure settings for the hosting platform, including:

- **Bitness:** 32-bit or 64-bit.
- **WebSocket protocol:** For ASP.NET SignalR or socket.io, for example.
- **Always On:** Keep the app loaded even when there's no traffic. By default, **Always On** is not enabled and the app is unloaded after 20 minutes without any incoming requests. It's required for continuous WebJobs or for WebJobs that are triggered using a CRON expression.

21

- **Managed pipeline version:** The IIS pipeline mode. Set it to **Classic** if you have a legacy app that requires an older version of IIS.
- **HTTP version:** Set to 2.0 to enable support for HTTPS/2 protocol.
- **ARR affinity:** In a multi-instance deployment, ensure that the client is routed to the same instance for the life of the session. You can set this option to **Off** for stateless applications.
- **Debugging:** Enable remote debugging for ASP.NET, ASP.NET Core, or Node.js apps. This option turns off automatically after 48 hours.
- **Incoming client certificates:** require client certificates in mutual authentication. TLS mutual authentication is used to restrict access to your app by enabling different types of authentication for it.

## Configure path mappings

In the **Configuration > Path mappings** section you can configure handler mappings, and virtual application and directory mappings. The **Path mappings** page will display different options based on the OS type.

## Windows apps (uncontainerized)

For Windows apps, you can customize the IIS handler mappings and virtual applications and directories.

Handler mappings let you add custom script processors to handle requests for specific file extensions. To add a custom handler, select **New handler**. Configure the handler as follows:

- **Extension:** The file extension you want to handle, such as *\*.php* or *handler.fcgi*.
- **Script processor:** The absolute path of the script processor. Requests to files that match the file extension are processed by the script processor. Use the path `D:\home\site\wwwroot` to refer to your app's root directory.
- **Arguments:** Optional command-line arguments for the script processor.

Each app has the default root path (/) mapped to `D:\home\site\wwwroot`, where your code is deployed by default. If your app root is in a different folder, or if your repository has more than one application, you can edit or add virtual applications and directories.

You can configure virtual applications and directories by specifying each virtual directory and its corresponding physical path relative to the website root (D:\home). To mark a virtual directory as a web application, clear the **Directory** check box.

## Linux and containerized apps

You can add custom storage for your containerized app. Containerized apps include all Linux apps and also the Windows and Linux custom containers running on App Service. Click **New Azure Storage Mount** and configure your custom storage as follows:

- **Name:** The display name.
- **Configuration options:** Basic or Advanced.
- **Storage accounts:** The storage account with the container you want.
- **Storage type:** **Azure Blobs** or **Azure Files**. Windows container apps only support Azure Files.
- **Storage container:** For basic configuration, the container you want.
- **Share name:** For advanced configuration, the file share name.

22

- **Access key:** For advanced configuration, the access key.
- **Mount path:** The absolute path in your container to mount the custom storage.

## Enable diagnostic logging

There are built-in diagnostics to assist with debugging an App Service app. In this lesson, you will learn how to enable diagnostic logging and add instrumentation to your application, as well as how to access the information logged by Azure.

The table below shows the types of logging, the platforms supported, and where the logs can be stored and located for accessing the information.

Type	Platform	Location	Description
Application logging	Windows, Linux	App Service file system and/or Azure Storage blobs	Logs messages generated by your application code. The messages can be generated by the web framework you choose, or from your application code directly using the standard logging pattern of your language. Each message is assigned one of the following categories: <b>Critical</b> , <b>Error</b> , <b>Warning</b> , <b>Info</b> , <b>Debug</b> , and <b>Trace</b> .

Web server logging	Windows	App Service file system or Azure Storage blobs	Raw HTTP request data in the W3C extended log file format. Each log message includes data like the HTTP method, resource URI, client IP, client port, user agent, response code, and so on.
Detailed error logging	Windows	App Service file system	Copies of the <i>.htm</i> error pages that would have been sent to the client browser. For security reasons, detailed error pages shouldn't be sent to clients in production, but App Service can save the error page each time an application error occurs that has HTTP code 400 or greater.

Type	Platform	Location	Description
Failed request tracing	Windows	App Service file system	Detailed tracing information on failed requests, including a trace of the IIS components used to process the request and the time taken in each component. One folder is generated for each failed request, which contains the XML log file, and the XSL stylesheet to view the log file with.
Deployment logging	Windows, Linux	App Service file system	Helps determine why a deployment failed. Deployment logging happens automatically and there are no configurable settings

			for deployment logging.
--	--	--	-------------------------

## Enable application logging (Windows)

1. To enable application logging for Windows apps in the Azure portal, navigate to your app and select **App Service logs**.
2. Select **On** for either **Application Logging (Filesystem)** or **Application Logging (Blob)**, or both. The **Filesystem** option is for temporary debugging purposes, and turns itself off in 12 hours. The **Blob** option is for long-term logging, and needs a blob storage container to write logs to.
3. You can also set the **Level** of details included in the log as shown in the table below.

Level	Included categories
<b>Disabled</b>	None
<b>Error</b>	Error, Critical
<b>Warning</b>	Warning, Error, Critical
<b>Information</b>	Info, Warning, Error, Critical
<b>Verbose</b>	Trace, Debug, Info, Warning, Error, Critical (all categories)

4. When finished, select **Save**.

## Enable application logging (Linux/Container)

1. In **App Service logs** set the **Application logging** option to **File System**.
2. In **Quota (MB)**, specify the disk quota for the application logs. In **Retention Period (Days)**, set the number of days the logs should be retained.
3. When finished, select **Save**.

24

## Enable web server logging

1. For **Web server logging**, select **Storage** to store logs on blob storage, or **File System** to store logs on the App Service file system.
2. In **Retention Period (Days)**, set the number of days the logs should be retained.
3. When finished, select **Save**.

## Add log messages in code

In your application code, you use the usual logging facilities to send log messages to the application logs. For example:

- ASP.NET applications can use the `System.Diagnostics.Trace` class to log information to the application diagnostics log. For example:  

```
System.Diagnostics.Trace.TraceError("If you're seeing this, something bad happened");
```

- By default, ASP.NET Core uses the `Microsoft.Extensions.Logging.AzureAppServices` logging provider.

## Stream logs

Before you stream logs in real time, enable the log type that you want. Any information written to files ending in .txt, .log, or .htm that are stored in the `/LogFiles` directory (`d:/home/logfiles`) is streamed by App Service.

**Note:** Some types of logging buffer write to the log file, which can result in out of order events in the stream. For example, an application log entry that occurs when a user visits a page may be displayed in the stream before the corresponding HTTP log entry for the page request.

- Azure portal - To stream logs in the Azure portal, navigate to your app and select **Log stream**.
- Azure CLI - To stream logs live in Cloud Shell, use the following command:

```
az webapp log tail --name appname --resource-group myResourceGroup
```

- Local console - To stream logs in the local console, install Azure CLI and sign in to your account. Once signed in, follow the instructions for Azure CLI above.

## Access log files

If you configure the Azure Storage blobs option for a log type, you need a client tool that works with Azure Storage.

For logs stored in the App Service file system, the easiest way is to download the ZIP file in the browser at:

- Linux/container apps: `https://<app-name>.scm.azurewebsites.net/api/logs/docker/zip`
- Windows apps: `https://<app-name>.scm.azurewebsites.net/api/dump`

For Linux/container apps, the ZIP file contains console output logs for both the docker host and the docker container. For a scaled-out app, the ZIP file contains one set of logs for each instance. In the App Service file system, these log files are the contents of the `/home/LogFiles` directory.

25

## Configure security certificates

You have been asked to help secure information being transmitted between your company's app and the customer. Azure App Service has tools that let you create, upload, or import a private certificate or a public certificate into App Service.

A certificate uploaded into an app is stored in a deployment unit that is bound to the app service plan's resource group and region combination (internally called a *webspace*). This makes the certificate accessible to other apps in the same resource group and region combination.

The table below details the options you have for adding certificates in App Service:

Option	Description
Create a free App Service managed certificate	A private certificate that's free of charge and easy to use if you just need to secure your custom domain in App Service.
Purchase an App Service certificate	A private certificate that's managed by Azure. It combines the simplicity of automated certificate management and the flexibility of renewal and export options.



Import a certificate from Key Vault	Useful if you use Azure Key Vault to manage your certificates.
Upload a private certificate	If you already have a private certificate from a third-party provider, you can upload it.
Upload a public certificate	Public certificates are not used to secure custom domains, but you can load them into your code if you need them to access remote resources.

## Private certificate requirements

The free **App Service managed certificate** and the **App Service certificate** already satisfy the requirements of App Service. If you want to use a private certificate in App Service, your certificate must meet the following requirements:

- Exported as a password-protected PFX file, encrypted using triple DES.
- Contains private key at least 2048 bits long
- Contains all intermediate certificates in the certificate chain

To secure a custom domain in a TLS binding, the certificate has additional requirements:

- Contains an Extended Key Usage for server authentication (OID = 1.3.6.1.5.5.7.3.1)
- Signed by a trusted certificate authority

## Creating a free managed certificate

To create custom TLS/SSL bindings or enable client certificates for your App Service app, your App Service plan must be in the **Basic**, **Standard**, **Premium**, or **Isolated** tier. Custom SSL is not supported in the **F1** or **D1** tier.

The free App Service managed certificate is a turn-key solution for securing your custom DNS name in App Service. It's a TLS/SSL server certificate that's fully managed by App Service and renewed continu-

ously and automatically in six-month increments, 45 days before expiration. You create the certificate and bind it to a custom domain, and let App Service do the rest.

The free certificate comes with the following limitations:

- Does not support wildcard certificates.
- Does not support usage as a client certificate by certificate thumbprint.
- Is not exportable.
- Is not supported on App Service Environment (ASE).
- Is not supported with root domains that are integrated with Traffic Manager.
- If a certificate is for a CNAME-mapped domain, the CNAME must be mapped directly to `<app name>.azurewebsites.net`.

## Import an App Service Certificate

If you purchase an App Service Certificate from Azure, Azure manages the following tasks:

- Takes care of the purchase process from GoDaddy.

- Performs domain verification of the certificate.
- Maintains the certificate in Azure Key Vault.
- Manages certificate renewal.
- Synchronize the certificate automatically with the imported copies in App Service apps.

If you already have a working App Service certificate, you can:

- Import the certificate into App Service.
- Manage the certificate, such as renew, rekey, and export it.

**Note:** App Service Certificates are not supported in Azure National Clouds at this time.

## Upload a private certificate

If your certificate authority gives you multiple certificates in the certificate chain, you need to merge the certificates in order. Then you can Export your merged TLS/SSL certificate with the private key that your certificate request was generated with.

If you generated your certificate request using OpenSSL, then you have created a private key file. To export your certificate to PFX, run the following command. Replace the placeholders <private-key file> and <merged-certificate-file> with the paths to your private key and your merged certificate file.

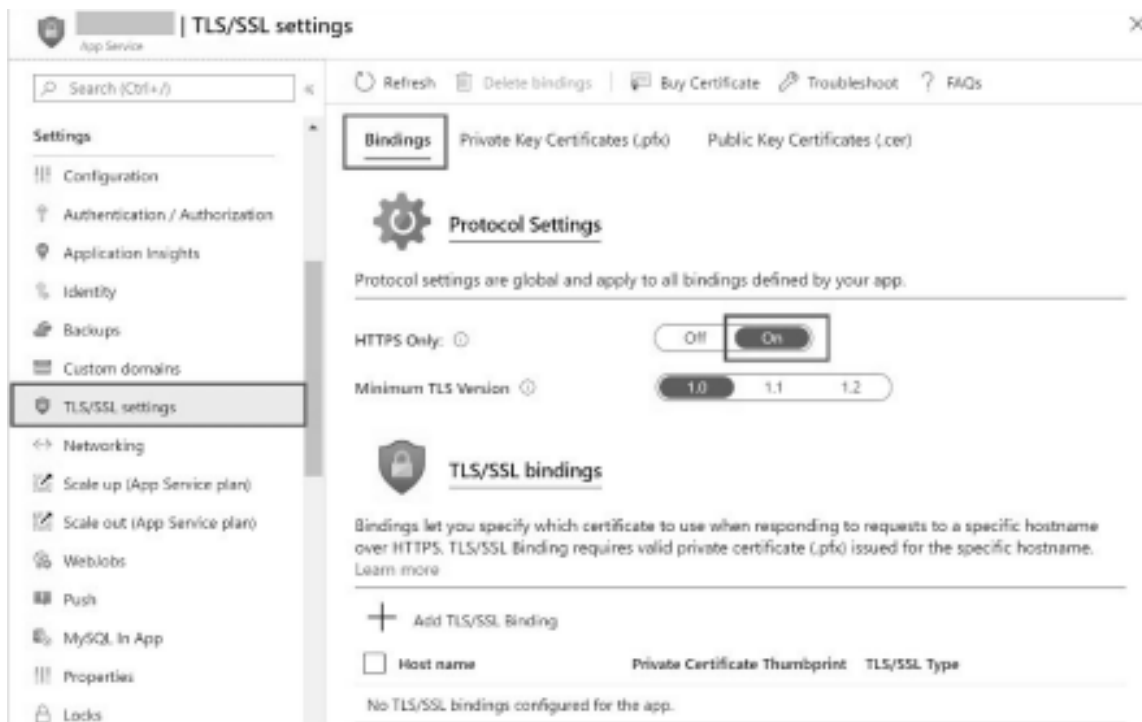
```
openssl pkcs12 -export -out myserver.pfx -inkey <private-key-file>
-in <merged-certificate-file>
```

When prompted, define an export password. You'll use this password when uploading your TLS/SSL certificate to App Service.

27

## Enforce HTTPS

By default, anyone can still access your app using HTTP. You can redirect all HTTP requests to the HTTPS port by navigating to your app page and, in the left navigation, select **TLS/SSL settings**. Then, in **HTTPS Only**, select **On**.



## Manage app features

Feature management is a modern software-development practice that decouples feature release from code deployment and enables quick changes to feature availability on demand. It uses a technique called `feature flags` (also known as `feature toggles`, `feature switches`, and so on) to dynamically administer a feature's lifecycle.

## Basic concepts

Here are several new terms related to feature management:

- **Feature flag:** A feature flag is a variable with a binary state of *on* or *off*. The feature flag also has an associated code block. The state of the feature flag triggers whether the code block runs or not.
- **Feature manager:** A feature manager is an application package that handles the lifecycle of all the feature flags in an application. The feature manager typically provides additional functionality, such as caching feature flags and updating their states.
- **Filter:** A filter is a rule for evaluating the state of a feature flag. A user group, a device or browser type, a geographic location, and a time window are all examples of what a filter can represent.

An effective implementation of feature management consists of at least two components working in concert:

- An application that makes use of feature flags.

28

- A separate repository that stores the feature flags and their current states.

How these components interact is illustrated in the following examples.

## Feature flag usage in code

The basic pattern for implementing feature flags in an application is simple. You can think of a feature flag as a Boolean state variable used with an `if` conditional statement in your code:

```
if (featureFlag) {  
    // Run the following code  
}
```

In this case, if `featureFlag` is set to `True`, the enclosed code block is executed; otherwise, it's skipped. You can set the value of `featureFlag` statically, as in the following code example:

```
bool featureFlag = true;
```

You can also evaluate the flag's state based on certain rules:

```
bool featureFlag = isBetaUser();
```

A slightly more complicated feature flag pattern includes an `else` statement as well:

```
if (featureFlag) {  
    // This following code will run if the featureFlag value is true }  
else {  
    // This following code will run if the featureFlag value is false }
```

## Feature flag declaration

Each feature flag has two parts: a name and a list of one or more filters that are used to evaluate if a feature's state is *on* (that is, when its value is `True`). A filter defines a use case for when a feature should be turned on.

When a feature flag has multiple filters, the filter list is traversed in order until one of the filters deter

mines the feature should be enabled. At that point, the feature flag is *on*, and any remaining filter results are skipped. If no filter indicates the feature should be enabled, the feature flag is *off*.

The feature manager supports *appsettings.json* as a configuration source for feature flags. The following example shows how to set up feature flags in a JSON file:

```
"FeatureManagement": {  
  "FeatureA": true, // Feature flag set to on  
  "FeatureB": false, // Feature flag set to off  
  "FeatureC": {  
    "EnabledFor": [  
      {  
        "Name": "Percentage",  
        "Parameters": {  
          "Value": 50  
        }  
      }  
    ]  
  }  
}
```

29

## Feature flag repository

To use feature flags effectively, you need to externalize all the feature flags used in an application. This approach allows you to change feature flag states without modifying and redeploying the application itself.

Azure App Configuration is designed to be a centralized repository for feature flags. You can use it to define different kinds of feature flags and manipulate their states quickly and confidently. You can then use the App Configuration libraries for various programming language frameworks to easily access these feature flags from your application.

## Knowledge check

### Multiple choice

*In which of the app configuration settings categories below would you set the language and SDK version?* Ⓐ Application settings

Ⓑ Path mappings

Ⓒ General settings

### Multiple choice

*Which of the following types of application logging is supported on the Linux platform?* Ⓐ Web server logging

Ⓑ Failed request tracing

Ⓒ Deployment logging

### Multiple choice

*Which of the following choices correctly lists the two parts of a feature flag?*

Ⓐ Name, App Settings

- ⑥ Name, one or more filters
- ⑥ Feature manager, one or more filters

## Summary

In this module, you learned how to:

- Create application settings that are bound to deployment slots.
- Explain the options for installing SSL/TLS certificates for your app.

30

- Enable diagnostic logging for your app to aid in monitoring and debugging. •

Create virtual app to directory mappings.

31

## Scale apps in Azure App Service

### Introduction

Autoscaling enables a system to adjust the resources required to meet the varying demand from users, while controlling the costs associated with these resources. You can use autoscaling with many Azure services, including web applications. Autoscaling requires you to configure autoscale rules that specify the conditions under which resources should be added or removed.

### Learning objectives

After completing this module, you'll be able to:

- Identify scenarios for which autoscaling is an appropriate solution
- Create autoscaling rules for a web app
- Monitor the effects of autoscaling

### Examine autoscale factors

Autoscaling can be triggered according to a schedule, or by assessing whether the system is running short on resources. For example, autoscaling could be triggered if CPU utilization grows, memory occupancy increases, the number of incoming requests to a service appears to be surging, or some combination of factors.

### What is autoscaling?

Autoscaling is a cloud system or process that adjusts available resources based on the current demand. Autoscaling performs scaling *in and out*, as opposed to scaling *up and down*.

### Azure App Service Autoscaling

Autoscaling in Azure App Service monitors the resource metrics of a web app as it runs. It detects situations where additional resources are required to handle an increasing workload, and ensures those resources are available before the system becomes overloaded.

Autoscaling responds to changes in the environment by adding or removing web servers and balancing the load between them. Autoscaling doesn't have any effect on the CPU power, memory, or storage capacity of the web servers powering the app, it only changes the number of web servers.

## Autoscaling rules

Autoscaling makes its decisions based on rules that you define. A rule specifies the threshold for a metric, and triggers an autoscale event when this threshold is crossed. Autoscaling can also deallocate resources when the workload has diminished.

Define your autoscaling rules carefully. For example, a Denial of Service attack will likely result in a large-scale influx of incoming traffic. Trying to handle a surge in requests caused by a DoS attack would be fruitless and expensive. These requests aren't genuine, and should be discarded rather than processed. A better solution is to implement detection and filtering of requests that occur during such an attack before they reach your service.

32

## When should you consider autoscaling?

Autoscaling provides elasticity for your services. It's a suitable solution when hosting any application when you can't easily predict the workload in advance, or when the workload is likely to vary by date or time. For example, you might expect increased/reduced activity for a business app during holidays.

Autoscaling improves availability and fault tolerance. It can help ensure that client requests to a service won't be denied because an instance is either not able to acknowledge the request in a timely manner, or because an overloaded instance has crashed.

Autoscaling works by adding or removing web servers. If your web apps perform resource-intensive processing as part of each request, then autoscaling might not be an effective approach. In these situations, manually scaling up may be necessary. For example, if a request sent to a web app involves performing complex processing over a large dataset, depending on the instance size, this single request could exhaust the processing and memory capacity of the instance.

Autoscaling isn't the best approach to handling long-term growth. You might have a web app that starts with a small number of users, but increases in popularity over time. Autoscaling has an overhead associated with monitoring resources and determining whether to trigger a scaling event. In this scenario, if you can anticipate the rate of growth, manually scaling the system over time may be a more cost effective approach.

The number of instances of a service is also a factor. You might expect to run only a few instances of a service most of the time. However, in this situation, your service will always be susceptible to downtime or lack of availability whether autoscaling is enabled or not. The fewer the number of instances initially, the less capacity you have to handle an increasing workload while autoscaling spins up additional instances.

## Identify autoscale factors

Autoscaling enables you to specify the conditions under which a web app should be scaled out, and back in again. Effective autoscaling ensures sufficient resources are available to handle large volumes of requests at peak times, while managing costs when the demand drops.

You can configure autoscaling to detect when to scale in and out according to a combination of factors, based on resource usage. You can also configure autoscaling to occur according to a schedule.

In this unit, you'll learn how to specify the factors that can be used to autoscale a service.

## Autoscaling and the App Service Plan

Autoscaling is a feature of the App Service Plan used by the web app. When the web app scales out, Azure starts new instances of the hardware defined by the App Service Plan to the app.

To prevent runaway autoscaling, an App Service Plan has an instance limit. Plans in more expensive pricing tiers have a higher limit. Autoscaling cannot create more instances than this limit.

**Note:** Not all App Service Plan pricing tiers support autoscaling.

# Autoscale conditions

You indicate how to autoscale by creating autoscale conditions. Azure provides two options for autoscaling:

- Scale based on a metric, such as the length of the disk queue, or the number of HTTP requests awaiting processing.

33

- Scale to a specific instance count according to a schedule. For example, you can arrange to scale out at a particular time of day, or on a specific date or day of the week. You also specify an end date, and the system will scale back in at this time.

Scaling to a specific instance count only enables you to scale out to a defined number of instances. If you need to scale out incrementally, you can combine metric and schedule-based autoscaling in the same autoscale condition. So, you could arrange for the system to scale out if the number of HTTP requests exceeds some threshold, but only between certain hours of the day.

You can create multiple autoscale conditions to handle different schedules and metrics. Azure will autoscale your service when any of these conditions apply. An App Service Plan also has a default condition that will be used if none of the other conditions are applicable. This condition is always active and doesn't have a schedule.

## Metrics for autoscale rules

Autoscaling by metric requires that you define one or more autoscale rules. An autoscale rule specifies a metric to monitor, and how autoscaling should respond when this metric crosses a defined threshold. The metrics you can monitor for a web app are:

- **CPU Percentage.** This metric is an indication of the CPU utilization across all instances. A high value shows that instances are becoming CPU-bound, which could cause delays in processing client requests.
- **Memory Percentage.** This metric captures the memory occupancy of the application across all instances. A high value indicates that free memory could be running low, and could cause one or more instances to fail.
- **Disk Queue Length.** This metric is a measure of the number of outstanding I/O requests across all instances. A high value means that disk contention could be occurring.
- **Http Queue Length.** This metric shows how many client requests are waiting for processing by the web app. If this number is large, client requests might fail with HTTP 408 (Timeout) errors.
- **Data In.** This metric is the number of bytes received across all instances.
- **Data Out.** This metric is the number of bytes sent by all instances.

You can also scale based on metrics for other Azure services. For example, if the web app processes requests received from a Service Bus Queue, you might want to spin up additional instances of a web app if the number of items held in an Azure Service Bus Queue exceeds a critical length.

## How an autoscale rule analyzes metrics

Autoscaling works by analyzing trends in metric values over time across all instances. Analysis is a multi-step process.

In the first step, an autoscale rule aggregates the values retrieved for a metric for all instances across a period of time known as the *time grain*. Each metric has its own intrinsic time grain, but in most cases this period is 1 minute. The aggregated value is known as the *time aggregation*. The options available are *Average*, *Minimum*, *Maximum*, *Total*, *Last*, and *Count*.

An interval of one minute is a very short interval in which to determine whether any change in metric is long-lasting enough to make autoscaling worthwhile. So, an autoscale rule performs a second step that

performs a further aggregation of the value calculated by the *time aggregation* over a longer, user-specified period, known as the *Duration*. The minimum *Duration* is 5 minutes. If the *Duration* is set to 10 minutes for example, the autoscale rule will aggregate the 10 values calculated for the *time grain*.

34

The aggregation calculation for the *Duration* can be different for that of the *time grain*. For example, if the *time aggregation* is *Average* and the statistic gathered is *CPU Percentage* across a one-minute *time grain*, each minute the average CPU percentage utilization across all instances for that minute will be calculated. If the *time grain statistic* is set to *Maximum*, and the *Duration* of the rule is set to 10 minutes, the maximum of the 10 average values for the CPU percentage utilization will be used to determine whether the rule threshold has been crossed.

## Autoscale actions

When an autoscale rule detects that a metric has crossed a threshold, it can perform an autoscale action. An autoscale action can be *scale-out* or *scale-in*. A scale-out action increases the number of instances, and a scale-in action reduces the instance count. An autoscale action uses an operator (such as *less than*, *greater than*, *equal to*, and so on) to determine how to react to the threshold. Scale-out actions typically use the *greater than* operator to compare the metric value to the threshold. Scale-in actions tend to compare the metric value to the threshold with the *less than* operator. An autoscale action can also set the instance count to a specific level, rather than incrementing or decrementing the number available.

An autoscale action has a *cool down* period, specified in minutes. During this interval, the scale rule won't be triggered again. This is to allow the system to stabilize between autoscale events. Remember that it takes time to start up or shut down instances, and so any metrics gathered might not show any significant changes for several minutes. The minimum cool down period is five minutes.

## Pairing autoscale rules

You should plan for scaling-in when a workload decreases. Consider defining autoscale rules in pairs in the same autoscale condition. One autoscale rule should indicate how to scale the system out when a metric exceeds an upper threshold. Then other rule should define how to scale the system back in again when the same metric drops below a lower threshold.

## Combining autoscale rules

A single autoscale condition can contain several autoscale rules (for example, a scale-out rule and the corresponding scale-in rule). However, the autoscale rules in an autoscale condition don't have to be directly related. You could define the following four rules in the same autoscale condition:

- If the HTTP queue length exceeds 10, scale out by 1
- If the CPU utilization exceeds 70%, scale out by 1
- If the HTTP queue length is zero, scale in by 1
- If the CPU utilization drops below 50%, scale in by 1

When determining whether to scale out, the autoscale action will be performed if **any** of the scale-out rules are met (HTTP queue length exceeds 10 **or** CPU utilization exceeds 70%). When scaling in, the autoscale action will run **only if all** of the scale-in rules are met (HTTP queue length drops to zero **and** CPU utilization falls below 50%). If you need to scale in if only one the scale-in rules are met, you must define the rules in separate autoscale conditions.

## Enable autoscale in App Service

In this unit, you will learn how to enable autoscaling, create autoscale rules, and monitor autoscaling activity

35



## Enable autoscaling

To get started with autoscaling navigate to your App Service plan in the Azure portal and select **Scale out (App Service plan)** in the **Settings** group in the left navigation pane.

**Note:** Not all pricing tiers support autoscaling. The development pricing tiers are either limited to a single instance (the **F1** and **D1** tiers), or they only provide manual scaling (the **B1** tier). If you've selected one of these tiers, you must first scale up to the **S1** or any of the **P** level production tiers.

By default, an App Service Plan only implements manual scaling. Selecting **Custom autoscale** reveals condition groups you can use to manage your scale settings.

The screenshot shows the 'Scale out' configuration page in the Azure portal. At the top, there are buttons for 'Save', 'Discard', 'Refresh', 'Logs', and 'Feedback'. Below these are tabs for 'Configure', 'Run history', 'JSON', 'Notify', and 'Diagnostic settings'. A descriptive paragraph explains that Autoscale is a built-in feature for scaling based on metrics, schedule, or manually. The 'Choose how to scale your resource' section has two options: 'Manual scale' (with a radio button) and 'Custom autoscale' (with a radio button and a box around it). Below the 'Manual scale' option, there is a section titled 'Manual scale' with an 'Override condition' input field and an 'Instance count' slider set to 1.

## Add scale conditions

Once you enable autoscaling, you can edit the automatically created default scale condition, and you can add your own custom scale conditions. Remember that each scale condition can either scale based on a metric, or scale to a specific instance count.

The Default scale condition is executed when none of the other scale conditions are active.

**Default\***
Auto created default scale condition

Scale mode
☐ Scale based on a metric
☒ Scale to a specific instance count

Instance count\*

Schedule
**This scale condition is executed when none of the other scale condition(s) match**

Auto created scale condition 1

Scale mode
☒ Scale based on a metric
☐ Scale to a specific instance count

Rules

*No metric rules defined; click Add a rule to scale out and scale in your instances based on rules. For example: 'Add a rule that increases instance count by 1 when CPU percentage is above 70%'. If you save the setting without any rules defined, no scaling will occur.*

+ Add a rule

Instance limits

Minimum
Maximum
Default

Schedule
☒ Specify start/end dates
☐ Repeat specific days

Timezone

Start date

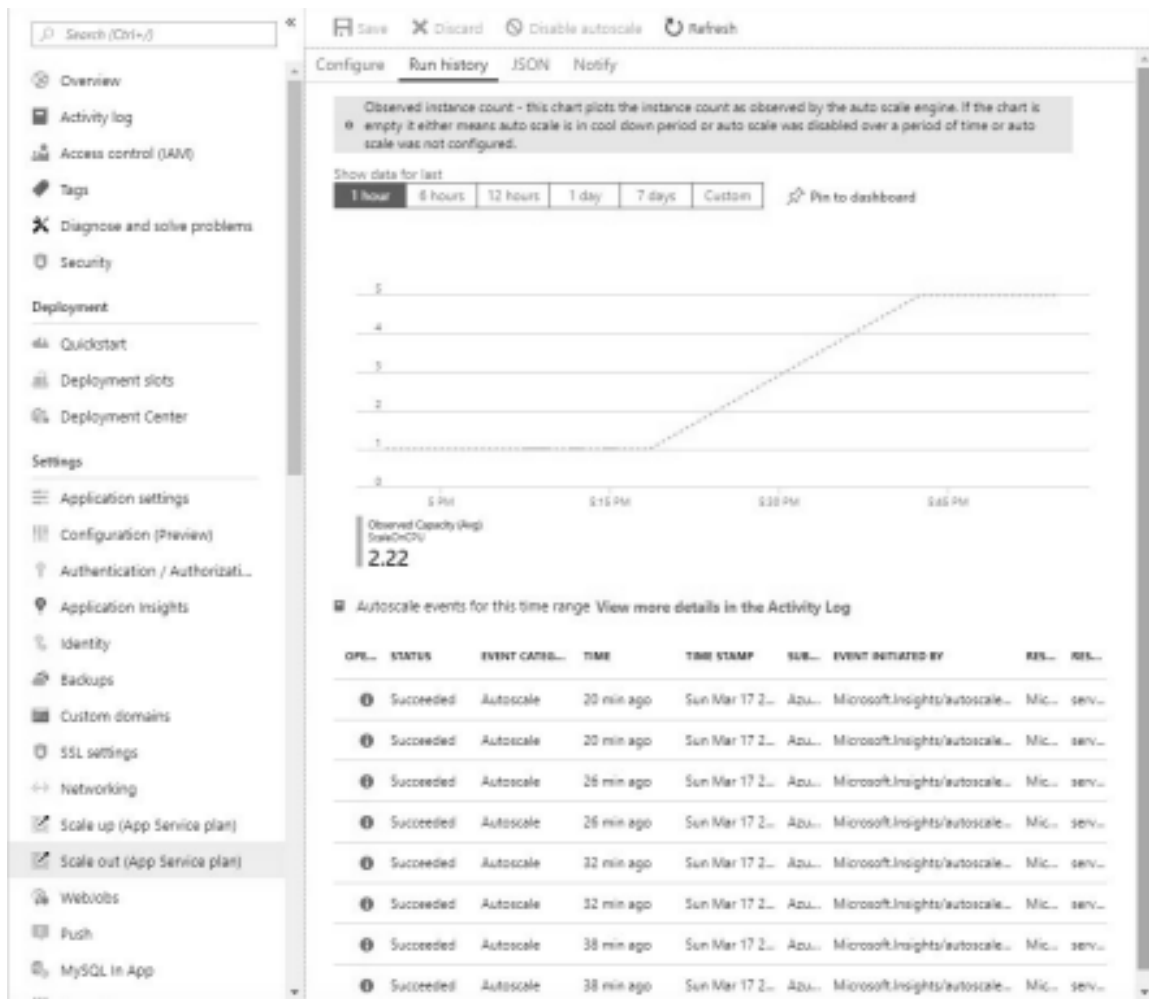
End date

A metric-based scale condition can also specify the minimum and maximum number of instances to create. The maximum number can't exceed the limits defined by the pricing tier. Additionally, all scale conditions other than the default may include a schedule indicating when the condition should be applied.

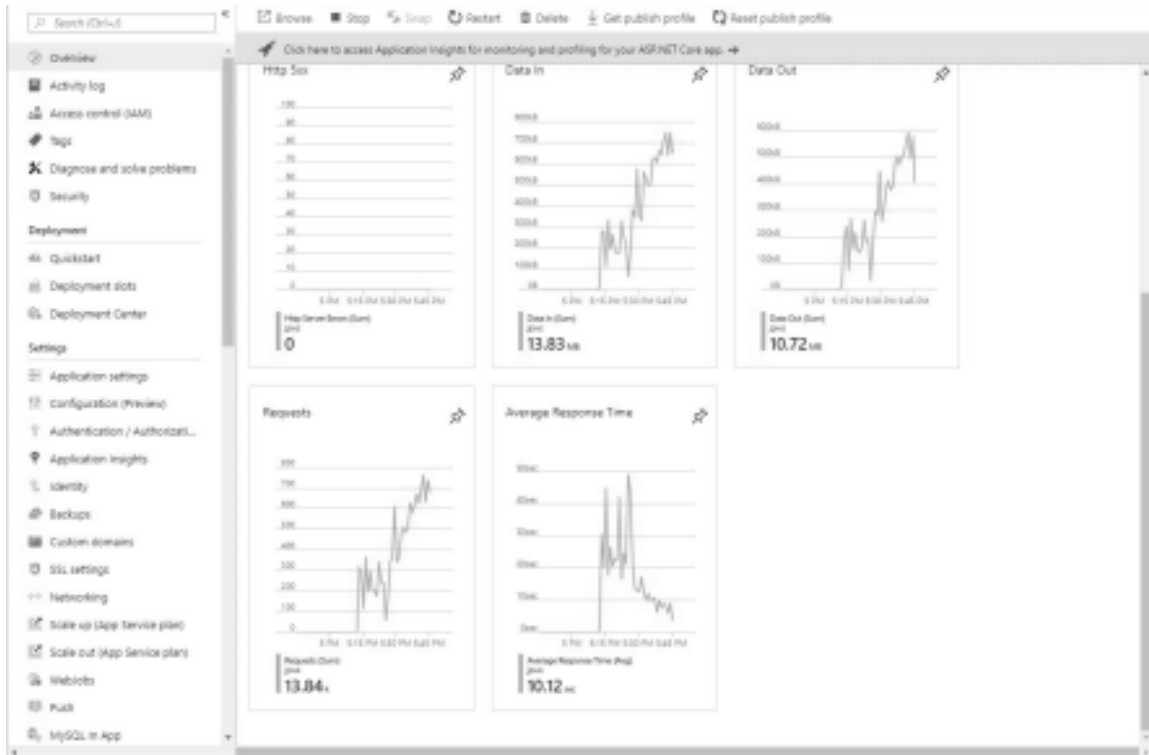
## Create scale rules

A metric-based scale condition contains one or more scale rules. You use the **Add a rule** link to add your own custom rules. You define the criteria that indicate when a rule should trigger an autoscale action, and the autoscale action to be performed (scale out or scale in) using the metrics, aggregations, operators, and thresholds described earlier.





You can use the **Run history** chart in conjunction with the metrics shown on the **Overview** page to correlate the autoscaling events with resource utilization.



## Explore autoscale best practices

If you're not following good practices when creating autoscale settings you can create conditions that lead to undesirable results. In this unit you will learn how to avoid creating rules that conflict with each other.

### Autoscale concepts

- An autoscale setting scales instances horizontally, which is *out* by increasing the instances and *in* by decreasing the number of instances. An autoscale setting has a maximum, minimum, and default value of instances.
- An autoscale job always reads the associated metric to scale by, checking if it has crossed the configured threshold for scale-out or scale-in.
- All thresholds are calculated at an instance level. For example, “scale out by one instance when average CPU > 80% when instance count is 2”, means scale-out when the average CPU across all instances is greater than 80%.
- All autoscale successes and failures are logged to the Activity Log. You can then configure an activity log alert so that you can be notified via email, SMS, or webhooks whenever there is activity.

### Autoscale best practices

Use the following best practices as you create your autoscale rules.

### Ensure the maximum and minimum values are different and have an adequate margin between them

If you have a setting that has minimum=2, maximum=2 and the current instance count is 2, no scale action can occur. Keep an adequate margin between the maximum and minimum instance counts, which are inclusive. Autoscale always scales between these limits.

# Choose the appropriate statistic for your diagnostics metric

For diagnostics metrics, you can choose among *Average*, *Minimum*, *Maximum* and *Total* as a metric to scale by. The most common statistic is Average.

## Choose the thresholds carefully for all metric types

We recommend carefully choosing different thresholds for scale-out and scale-in based on practical situations.

We *do not recommend* autoscale settings like the examples below with the same or very similar threshold values for out and in conditions:

- Increase instances by 1 count when Thread Count  $\geq 600$
- Decrease instances by 1 count when Thread Count  $\leq 600$

Let's look at an example of what can lead to a behavior that may seem confusing. Consider the following sequence.

1. Assume there are two instances to begin with and then the average number of threads per instance grows to 625.
2. Autoscale scales out adding a third instance.
3. Next, assume that the average thread count across instance falls to 575.
4. Before scaling in, autoscale tries to estimate what the final state will be if it scaled in. For example,  $575 \times 3$  (current instance count) = 1,725 / 2 (final number of instances when scaled in) = 862.5 threads. This means autoscale would have to immediately scale-out again even after it scaled in, if the average thread count remains the same or even falls only a small amount. However, if it scaled out again, the whole process would repeat, leading to an infinite loop.
5. To avoid this situation (termed "flapping"), autoscale does not scale in at all. Instead, it skips and reevaluates the condition again the next time the service's job executes. This can confuse many people because autoscale wouldn't appear to work when the average thread count was 575.

Estimation during a scale-in is intended to avoid "flapping" situations, where scale-in and scale-out actions continually go back and forth. Keep this behavior in mind when you choose the same thresholds for scale-out and in.

We recommend choosing an adequate margin between the scale-out and in thresholds. As an example, consider the following better rule combination.

- Increase instances by 1 count when CPU%  $\geq 80$
- Decrease instances by 1 count when CPU%  $\leq 60$

In this case

1. Assume there are 2 instances to start with.
2. If the average CPU% across instances goes to 80, autoscale scales out adding a third instance.

3. Now assume that over time the CPU% falls to 60.

4. Autoscale's scale-in rule estimates the final state if it were to scale-in. For example,  $60 \times 3$  (current instance count) = 180 / 2 (final number of instances when scaled in) = 90. So autoscale does not scale-in because it would have to scale-out again immediately. Instead, it skips scaling in.

5. The next time autoscale checks, the CPU continues to fall to 50. It estimates again -  $50 \times 3$  instance = 150 / 2 instances = 75, which is below the scale-out threshold of 80, so it scales in successfully to 2 instances.

## Considerations for scaling when multiple rules are

## configured in a profile

There are cases where you may have to set multiple rules in a profile. The following set of autoscale rules are used by services when multiple rules are set.

On *scale-out*, autoscale runs if any rule is met. On *scale-in*, autoscale requires all rules to be met. To illustrate, assume that you have the following four autoscale rules:

- If CPU < 30 %, scale-in by 1
- If Memory < 50%, scale-in by 1
- If CPU > 75%, scale-out by 1
- If Memory > 75%, scale-out by 1

Then the following occurs:

- If CPU is 76% and Memory is 50%, we scale-out.
- If CPU is 50% and Memory is 76% we scale-out.

On the other hand, if CPU is 25% and memory is 51% autoscale does not scale-in. In order to scale-in, CPU must be 29% and Memory 49%.

## Always select a safe default instance count

The default instance count is important as autoscale scales your service to that count when metrics are not available. Therefore, select a default instance count that's safe for your workloads.

## Configure autoscale notifications

Autoscale will post to the Activity Log if any of the following conditions occur:

- Autoscale issues a scale operation
- Autoscale service successfully completes a scale action
- Autoscale service fails to take a scale action.
- Metrics are not available for autoscale service to make a scale decision.
- Metrics are available (recovery) again to make a scale decision.

You can also use an Activity Log alert to monitor the health of the autoscale engine. In addition to using activity log alerts, you can also configure email or webhook notifications to get notified for successful scale actions via the notifications tab on the autoscale setting.

42

## Knowledge check

### Multiple choice

*Which of these statements best describes autoscaling?*

- ⑥ Autoscaling requires an administrator to actively monitor the workload on a system.
- ⑥ Autoscaling is a scale out/scale in solution.
- ⑥ Scaling up/scale down provides better availability than autoscaling.

### Multiple choice

*Which of these scenarios is a suitable candidate for autoscaling?*

- ⑥ The number of users requiring access to an application varies according to a regular schedule.  
For example, more users use the system on a Friday than other days of the week.

- ⑥ The system is subject to a sudden influx of requests that grinds your system to a halt.
- ⑥ Your organization is running a promotion and expects to see increased traffic to their web site for the next couple of weeks.

## Multiple choice

*There are multiple rules in an autoscale profile. Which of the following scale operations will run if any of the rule conditions are met?*

- ⑥ scale-out
- ⑥ scale-in
- ⑥ scale-out/in

## Summary

In this module, you learned how to:

- Identify scenarios for which autoscaling is an appropriate solution
- Create autoscaling rules for a web app
- Monitor the effects of autoscaling

43

# Explore Azure App Service deployment slots Introduction

The deployment slot functionality in App Service is a powerful tool that enables you to preview, manage, test, and deploy your different development environments.

## Learning objectives

After completing this module, you'll be able to:

- Describe the benefits of using deployment slots
- Understand how slot swapping operates in App Service
- Perform manual swaps and enable auto swap
- Route traffic manually and automatically

## Prerequisites

- Experience using the Azure portal to create and manage App Service web

## apps Explore staging environments

When you deploy your web app, web app on Linux, mobile back end, or API app to Azure App Service, you can use a separate deployment slot instead of the default production slot when you're running in the **Standard**, **Premium**, or **Isolated** App Service plan tier. Deployment slots are live apps with their own host names. App content and configurations elements can be swapped between two deployment slots, including the production slot.

Deploying your application to a non-production slot has the following benefits:

- You can validate app changes in a staging deployment slot before swapping it with the production slot.



- Deploying an app to a slot first and swapping it into production makes sure that all instances of the slot are warmed up before being swapped into production. This eliminates downtime when you deploy your app. The traffic redirection is seamless, and no requests are dropped because of swap operations. You can automate this entire workflow by configuring auto swap when pre-swap validation isn't needed.
- After a swap, the slot with previously staged app now has the previous production app. If the changes swapped into the production slot aren't as you expect, you can perform the same swap immediately to get your "last known good site" back.

Each App Service plan tier supports a different number of deployment slots. There's no additional charge for using deployment slots. To find out the number of slots your app's tier supports, visit **App Service limits**<sup>2</sup>.

To scale your app to a different tier, make sure that the target tier supports the number of slots your app already uses. For example, if your app has more than five slots, you can't scale it down to the **Standard** tier, because the **Standard** tier supports only five deployment slots.

<sup>2</sup> <https://docs.microsoft.com/azure/azure-resource-manager/management/azure-subscription-service-limits#app-service-limits>

When you create a new slot the new deployment slot has no content, even if you clone the settings from a different slot. You can deploy to the slot from a different repository branch or a different repository.

## Examine slot swapping

When you swap slots (for example, from a staging slot to the production slot), App Service does the following to ensure that the target slot doesn't experience downtime:

1. Apply the following settings from the target slot (for example, the production slot) to all instances of the source slot:
  - Slot-specific app settings and connection strings, if applicable.
  - Continuous deployment settings, if enabled.
  - App Service authentication settings, if enabled.

Any of these cases trigger all instances in the source slot to restart. During **swap with preview**, this marks the end of the first phase. The swap operation is paused, and you can validate that the source slot works correctly with the target slot's settings.
2. Wait for every instance in the source slot to complete its restart. If any instance fails to restart, the swap operation reverts all changes to the source slot and stops the operation.
3. If local cache is enabled, trigger local cache initialization by making an HTTP request to the application root ("/") on each instance of the source slot. Wait until each instance returns any HTTP response. Local cache initialization causes another restart on each instance.
4. If auto swap is enabled with custom warm-up, trigger Application Initiation by making an HTTP request to the application root ("/") on each instance of the source slot.
  - If `applicationInitialization` isn't specified, trigger an HTTP request to the application root of the source slot on each instance.
  - If an instance returns any HTTP response, it's considered to be warmed up.
5. If all instances on the source slot are warmed up successfully, swap the two slots by switching the routing rules for the two slots. After this step, the target slot (for example, the production slot) has the app that's previously warmed up in the source slot.
6. Now that the source slot has the pre-swap app previously in the target slot, perform the same operation by applying all settings and restarting the instances.

At any point of the swap operation, all work of initializing the swapped apps happens on the source slot.

The target slot remains online while the source slot is being prepared and warmed up, regardless of where the swap succeeds or fails. To swap a staging slot with the production slot, make sure that the production slot is always the target slot. This way, the swap operation doesn't affect your production app.

When you clone configuration from another deployment slot, the cloned configuration is editable. Some configuration elements follow the content across a swap (not slot specific), whereas other configuration elements stay in the same slot after a swap (slot specific). The following table shows the settings that change when you swap slots.

Settings that are swapped	Settings that aren't swapped
General settings, such as framework version, 32/64-bit, web sockets	Publishing endpoints
App settings (can be configured to stick to a slot)	Custom domain names

45

Settings that are swapped	Settings that aren't swapped
Connection strings (can be configured to stick to a slot)	Non-public certificates and TLS/SSL settings
Handler mappings	Scale settings
Public certificates	WebJobs schedulers
WebJobs content	IP restrictions
Hybrid connections *	Always On
Virtual network integration *	Diagnostic log settings
Service endpoints *	Cross-origin resource sharing (CORS)
Azure Content Delivery Network *	

Features marked with an asterisk (\*) are planned to be unswapped.

**Note:** To make settings swappable, add the app setting

`WEBSITE_OVERRIDE_PRESERVE_DEFAULT_STICKY_SLOT_SETTINGS` in every slot of the app and set its value to `0` or `false`. These settings are either all swappable or not at all. You can't make just some settings swappable and not the others. Managed identities are never swapped and are not affected by this override app setting.

To configure an app setting or connection string to stick to a specific slot (not swapped), go to the Configuration page for that slot. Add or edit a setting, and then select **Deployment slot setting**. Selecting this check box tells App Service that the setting is not swappable.

## Swap deployment slots

You can swap deployment slots on your app's Deployment slots page and the Overview page. Before you swap an app from a deployment slot into production, make sure that production is your target slot and that all settings in the source slot are configured exactly as you want to have them in production.

## Manually swapping deployment slots

To swap deployment slots:

1. Go to your app's **Deployment slots** page and select **Swap**. The **Swap** dialog box shows settings in the selected source and target slots that will be changed.

2. Select the desired **Source** and **Target** slots. Usually, the target is the production slot. Also, select the **Source Changes** and **Target Changes** tabs and verify that the configuration changes are expected. When you're finished, you can swap the slots immediately by selecting **Swap**.  
  
To see how your target slot would run with the new settings before the swap actually happens, don't select **Swap**, but follow the instructions in *Swap with preview* below.
3. When you're finished, close the dialog box by selecting **Close**.

## Swap with preview (multi-phase swap)

Before you swap into production as the target slot, validate that the app runs with the swapped settings. The source slot is also warmed up before the swap completion, which is desirable for mission-critical applications.

When you perform a swap with preview, App Service performs the same swap operation but pauses after the first step. You can then verify the result on the staging slot before completing the swap.

If you cancel the swap, App Service reapplies configuration elements to the source slot.

46

To swap with preview:

1. Follow the steps above in Swap deployment slots but select **Perform swap with preview**. The dialog box shows you how the configuration in the source slot changes in phase 1, and how the source and target slot change in phase 2.
2. When you're ready to start the swap, select **Start Swap**.  
  
When phase 1 finishes, you're notified in the dialog box. Preview the swap in the source slot by going to `https://<app_name>-<source-slot-name>.azurewebsites.net`.
3. When you're ready to complete the pending swap, select **Complete Swap in Swap action** and select **Complete Swap**.  
  
To cancel a pending swap, select **Cancel Swap** instead.
4. When you're finished, close the dialog box by selecting **Close**.

## Configure auto swap

Auto swap streamlines Azure DevOps scenarios where you want to deploy your app continuously with zero cold starts and zero downtime for customers of the app. When auto swap is enabled from a slot into production, every time you push your code changes to that slot, App Service automatically swaps the app into production after it's warmed up in the source slot.

**Note:** Auto swap isn't currently supported in web apps on Linux.

To configure auto swap:

1. Go to your app's resource page and select the deployment slot you want to configure to auto swap. The setting is on the **Configuration > General settings** page.
2. Set **Auto swap enabled** to **On**. Then select the desired target slot for Auto swap deployment slot, and select **Save** on the command bar.
3. Execute a code push to the source slot. Auto swap happens after a short time, and the update is reflected at your target slot's URL.

## Specify custom warm-up

Some apps might require custom warm-up actions before the swap. The `applicationInitialization` configuration element in `web.config` lets you specify custom initialization actions. The swap operation waits for this custom warm-up to finish before swapping with the target slot. Here's a sample `web.config` fragment.

```
<system.webServer>
  <applicationInitialization>
    <add initializationPage="/" hostname="[app hostname]" /> <add
initializationPage="/Home/About" hostname="[app hostname]" />
  </applicationInitialization>
</system.webServer>
```

For more information on customizing the `applicationInitialization` element, see **Most common deployment slot swap failures and how to fix them**<sup>3</sup>.

<sup>3</sup> <https://ruslany.net/2017/11/most-common-deployment-slot-swap-failures-and-how-to-fix-them/>

You can also customize the warm-up behavior with one or both of the following app settings:

- `WEBSITE_SWAP_WARMUP_PING_PATH`: The path to ping to warm up your site. Add this app setting by specifying a custom path that begins with a slash as the value. An example is `/statuscheck`. The default value is `/`.
- `WEBSITE_SWAP_WARMUP_PING_STATUSES`: Valid HTTP response codes for the warm-up operation. Add this app setting with a comma-separated list of HTTP codes. An example is `200,202`. If the returned status code isn't in the list, the warmup and swap operations are stopped. By default, all response codes are valid.

## Roll back and monitor a swap

If any errors occur in the target slot (for example, the production slot) after a slot swap, restore the slots to their pre-swap states by swapping the same two slots immediately.

If the swap operation takes a long time to complete, you can get information on the swap operation in the activity log.

On your app's resource page in the portal, in the left pane, select **Activity log**.

A swap operation appears in the log query as `Swap Web App Slots`. You can expand it and select one of the suboperations or errors to see the details.

## Route traffic in App Service

By default, all client requests to the app's production URL (`http://<app_name>.azurewebsites.net`) are routed to the production slot. You can route a portion of the traffic to another slot. This feature is useful if you need user feedback for a new update, but you're not ready to release it to production.

## Route production traffic automatically

To route production traffic automatically:

1. Go to your app's resource page and select **Deployment slots**.
2. In the **Traffic %** column of the slot you want to route to, specify a percentage (between 0 and 100) to represent the amount of total traffic you want to route. Select **Save**.

After the setting is saved, the specified percentage of clients is randomly routed to the non-production slot.

After a client is automatically routed to a specific slot, it's "pinned" to that slot for the life of that client session. On the client browser, you can see which slot your session is pinned to by looking at the `x-ms-routing-name` cookie in your HTTP headers. A request that's routed to the "staging" slot has the cookie `x-ms-routing-name=staging`. A request that's routed to the production slot has the cookie `x-ms-routing-name=self`.

## Route production traffic manually

In addition to automatic traffic routing, App Service can route requests to a specific slot. This is useful when you want your users to be able to opt in to or opt out of your beta app. To route production traffic manually, you use the `x-ms-routing-name` query parameter.

To let users opt out of your beta app, for example, you can put this link on your webpage:

```
<a href="<webappname>.azurewebsites.net/?x-ms-routing-name=self">Go back to  
production app</a>
```

The string `x-ms-routing-name=self` specifies the production slot. After the client browser accesses the link, it's redirected to the production slot. Every subsequent request has the `x-ms-routing-name=self` cookie that pins the session to the production slot.

To let users opt in to your beta app, set the same query parameter to the name of the non-production slot. Here's an example:

```
<webappname>.azurewebsites.net/?x-ms-routing-name=staging
```

By default, new slots are given a routing rule of 0%, a default value is displayed in grey. When you explicitly set this value to 0% it is displayed in black, your users can access the staging slot manually by using the `x-ms-routing-name` query parameter. But they won't be routed to the slot automatically because the routing percentage is set to 0. This is an advanced scenario where you can "hide" your staging slot from the public while allowing internal teams to test changes on the slot.

## Knowledge check

### Multiple choice

*By default, all client requests to the app's production URL (`http://<app_name>.azurewebsites.net`) are routed to the production slot. One can automatically route a portion of the traffic to another slot. What is the default routing rule applied to new deployment slots?*

- ☐ 0%
- ☐ 10%
- ☐ 20%

### Multiple choice

*Some configuration elements follow the content across a swap (not slot specific), whereas other configuration elements stay in the same slot after a swap (slot specific). Which of the settings below are swapped?*

- ☐ Publishing endpoints
- ☐ WebJobs content
- ☐ WebJobs schedulers

## Summary

In this module, you learned how to:

- Describe the benefits of using deployment slots
- Understand how slot swapping operates in App Service

- Perform manual swaps and enable auto swap
- Route traffic manually and automatically

## Answers

### Multiple choice

Which of the following App Service plans supports only function apps?

- Ⓔ Dedicated
- Ⓔ Isolated
- Consumption

*Explanation*

*That's correct. The consumption tier is only available to function apps. It scales the functions dynamically depending on workload.*

### Multiple choice

Which of the following networking features of App Service can be used to control outbound network traffic?

- Ⓔ App-assigned address
- Hybrid Connections
- Ⓔ Service endpoints

*Explanation*

*That's correct. Hybrid Connections are an outbound network feature.*

### Multiple choice

In which of the app configuration settings categories below would you set the language and SDK version?

- Ⓔ Application settings
- Ⓔ Path mappings
- General settings

*Explanation*

*That's correct. This category is used to configure stack, platform, debugging, and incoming client certificate settings.*

### Multiple choice

Which of the following types of application logging is supported on the Linux platform?

- Ⓔ Web server logging
- Ⓔ Failed request tracing
- Deployment logging

*Explanation*

*That's correct. Deployment logging is supported on the Linux platform.*

### Multiple choice

Which of the following choices correctly lists the two parts of a feature flag?

- Ⓔ Name, App Settings
- Name, one or more filters

- ⑥ Feature manager, one or more filters

*Explanation*

*That's correct. Each feature flag has two parts: a name and a list of one or more filters that are used to evaluate if a feature's state is on.*

**Multiple choice**

Which of these statements best describes autoscaling?

- ⑥ Autoscaling requires an administrator to actively monitor the workload on a system.

■ Autoscaling is a scale out/scale in solution.

- ⑥ Scaling up/scale down provides better availability than autoscaling.

*Explanation*

*That's correct. The system can scale out when specified resource metrics indicate increasing usage, and scale in when these metrics drop.*

**Multiple choice**

Which of these scenarios is a suitable candidate for autoscaling?

■ The number of users requiring access to an application varies according to a regular schedule. For example, more users use the system on a Friday than other days of the week.

- ⑥ The system is subject to a sudden influx of requests that grinds your system to a halt.

- ⑥ Your organization is running a promotion and expects to see increased traffic to their web site for the next couple of weeks.

*Explanation*

*That's correct. Changes in application load that are predictable are good candidates for*

*autoscaling. Multiple choice*

There are multiple rules in an autoscale profile. Which of the following scale operations will run if any of the rule conditions are met?

■ scale-out

- ⑥ scale-in

- ⑥ scale-out/in

*Explanation*

*That's correct. Scale-out operations will trigger if any of the rule conditions are met.*

51

**Multiple choice**

By default, all client requests to the app's production URL

([http://<app\\_name>.azurewebsites.net](http://<app_name>.azurewebsites.net)) are routed to the production slot. One can automatically route a portion of the traffic to another slot. What is the default routing rule applied to new deployment slots?

■ 0%

- ⑥ 10%

- ⑥ 20%

*Explanation*

*That's correct. By default, new slots are given a routing rule of 0%.*

**Multiple choice**

Some configuration elements follow the content across a swap (not slot specific), whereas other configuration elements stay in the same slot after a swap (slot specific). Which of the settings below are swapped?

⑥ Publishing endpoints

■ WebJobs content

⑥ WebJobs schedulers

*Explanation*

*That's correct. WebJobs content are swapped.*

## Module 2 Implement Azure Functions

### Explore Azure Functions

#### Introduction

Azure Functions lets you develop serverless applications on Microsoft Azure. You can write just the code you need for the problem at hand, without worrying about a whole application or the infrastructure to run it.

After completing this module, you'll be able to:

- Explain functional differences between Azure Functions, Azure Logic Apps, and

WebJobs • Describe Azure Functions hosting plan options

- Describe how Azure Functions scale to meet business needs

#### Discover Azure Functions

Azure Functions are a great solution for processing data, integrating systems, working with the internet-of-things (IoT), and building simple APIs and microservices. Consider Functions for tasks like image or order processing, file maintenance, or for any tasks that you want to run on a schedule. Functions provides templates to get you started with key scenarios.

Azure Functions supports *triggers*, which are ways to start execution of your code, and *bindings*, which are ways to simplify coding for input and output data. There are other integration and automation services in Azure and they all can solve integration problems and automate business processes. They can all define input, actions, conditions, and output.

#### Compare Azure Functions and Azure Logic Apps

Both Functions and Logic Apps enable serverless workloads. Azure Functions is a serverless compute service, whereas Azure Logic Apps provides serverless workflows. Both can create complex orchestrations. An orchestration is a collection of functions or steps, called actions in Logic Apps, that are executed to accomplish a complex task.

54

For Azure Functions, you develop orchestrations by writing code and using the Durable Functions extension. For Logic Apps, you create orchestrations by using a GUI or editing configuration files.

You can mix and match services when you build an orchestration, calling functions from logic apps and calling logic apps from functions. The following table lists some of the key differences between these:

	Azure Functions	Logic Apps
<b>Development</b>	Code-first (imperative)	Designer-first (declarative)



<b>Connectivity</b>	About a dozen built-in binding types, write code for custom bindings	Large collection of connectors, Enterprise Integration Pack for B2B scenarios, build custom connectors
<b>Actions</b>	Each activity is an Azure function; write code for activity functions	Large collection of ready-made actions
<b>Monitoring</b>	Azure Application Insights	Azure portal, Azure Monitor logs
<b>Management</b>	REST API, Visual Studio	Azure portal, REST API, Power Shell, Visual Studio
<b>Execution context</b>	Can run locally or in the cloud	Supports run-anywhere scenarios

## Compare Functions and WebJobs

Like Azure Functions, Azure App Service WebJobs with the WebJobs SDK is a code-first integration service that is designed for developers. Both are built on Azure App Service and support features such as source control integration, authentication, and monitoring with Application Insights integration.

Azure Functions is built on the WebJobs SDK, so it shares many of the same event triggers and connections to other Azure services. Here are some factors to consider when you're choosing between Azure Functions and WebJobs with the WebJobs SDK:

	Functions	WebJobs with WebJobs SDK
Serverless app model with automatic scaling	Yes	No
Develop and test in browser	Yes	No
Pay-per-use pricing	Yes	No
Integration with Logic Apps	Yes	No
Trigger events	Timer Azure Storage queues and blobs Azure Service Bus queues and topics Azure Cosmos DB Azure Event Hubs HTTP/WebHook (GitHub Slack) Azure Event Grid	Timer Azure Storage queues and blobs Azure Service Bus queues and topics Azure Cosmos DB Azure Event Hubs File system

Azure Functions offers more developer productivity than Azure App Service WebJobs does. It also offers more options for programming languages, development environments, Azure service integration, and pricing. For most scenarios, it's the best choice.

## Compare Azure Functions hosting options

When you create a function app in Azure, you must choose a hosting plan for your app. There are three

basic hosting plans available for Azure Functions: Consumption plan, Functions Premium plan, and App service (Dedicated) plan. All hosting plans are generally available (GA) on both Linux and Windows virtual machines.

The hosting plan you choose dictates the following behaviors:

- How your function app is scaled.
- The resources available to each function app instance.
- Support for advanced functionality, such as Azure Virtual Network connectivity. The

following is a summary of the benefits of the three main hosting plans for Functions:

Plan	Benefits
<b>Consumption plan</b>	This is the default hosting plan. It scales automatically and you only pay for compute resources when your functions are running. Instances of the Functions host are dynamically added and removed based on the number of incoming events.
<b>Premium plan</b>	Automatically scales based on demand using pre-warmed workers which run applications with no delay after being idle, runs on more powerful instances, and connects to virtual networks.
<b>Dedicated plan</b>	Run your functions within an App Service plan at regular App Service plan rates. Best for long-running scenarios where Durable Functions can't be used.

There are two other hosting options which provide the highest amount of control and isolation in which to run your function apps.

Hosting option	Details
<b>ASE</b>	<b>App Service Environment (ASE)</b> ( <a href="https://docs.microsoft.com/azure/app-service/environment/intro">https://docs.microsoft.com/azure/app-service/environment/intro</a> ) is an App Service feature that provides a fully isolated and dedicated environment for securely running App Service apps at high scale.
<b>Kubernetes</b>	Kubernetes provides a fully isolated and dedicated environment running on top of the Kubernetes platform. For more information visit <b>Azure Functions on Kubernetes with KEDA</b> ( <a href="https://docs.microsoft.com/azure/azure-functions/functions-kubernetes-keda">https://docs.microsoft.com/azure/azure-functions/functions-kubernetes-keda</a> ).

## Always on

If you run on an Dedicated plan, you should enable the **Always on** setting so that your function app runs correctly. On an App Service plan, the functions runtime goes idle after a few minutes of inactivity, so only HTTP triggers will “wake up” your functions. Always on is available only on an App Service plan. On

## Storage account requirements

On any plan, a function app requires a general Azure Storage account, which supports Azure Blob, Queue, Files, and Table storage. This is because Functions relies on Azure Storage for operations such as managing triggers and logging function executions, but some storage accounts do not support queues and tables. These accounts, which include blob-only storage accounts (including premium storage) and general-purpose storage accounts with zone-redundant storage replication, are filtered-out from your existing **Storage Account** selections when you create a function app.

The same storage account used by your function app can also be used by your triggers and bindings to store your application data. However, for storage-intensive operations, you should use a separate storage account.

## Scale Azure Functions

In the Consumption and Premium plans, Azure Functions scales CPU and memory resources by adding additional instances of the Functions host. The number of instances is determined on the number of events that trigger a function.

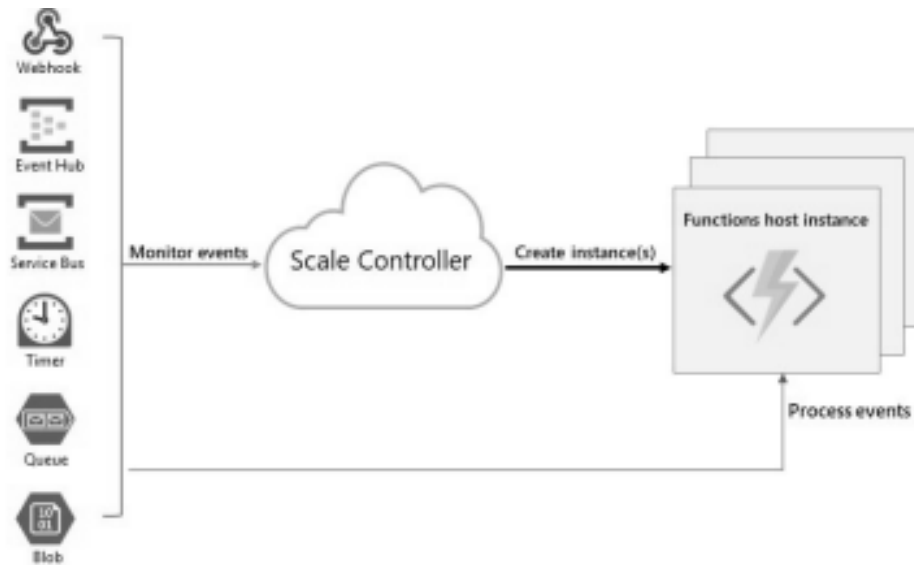
Each instance of the Functions host in the Consumption plan is limited to 1.5 GB of memory and one CPU. An instance of the host is the entire function app, meaning all functions within a function app share resource within an instance and scale at the same time. Function apps that share the same Consumption plan scale independently. In the Premium plan, the plan size determines the available memory and CPU for all apps in that plan on that instance.

Function code files are stored on Azure Files shares on the function's main storage account. When you delete the main storage account of the function app, the function code files are deleted and cannot be recovered.

## Runtime scaling

Azure Functions uses a component called the *scale controller* to monitor the rate of events and determine whether to scale out or scale in. The scale controller uses heuristics for each trigger type. For example, when you're using an Azure Queue storage trigger, it scales based on the queue length and the age of the oldest queue message.

The unit of scale for Azure Functions is the function app. When the function app is scaled out, additional resources are allocated to run multiple instances of the Azure Functions host. Conversely, as compute demand is reduced, the scale controller removes function host instances. The number of instances is eventually "scaled in" to zero when no functions are running within a function app.



**Note:** After your function app has been idle for a number of minutes, the platform may scale the number of instances on which your app runs down to zero. The next request has the added latency of scaling from zero to one. This latency is referred to as a *cold start*.

## Scaling behaviors

Scaling can vary on a number of factors, and scale differently based on the trigger and language selected. There are a few intricacies of scaling behaviors to be aware of:

- **Maximum instances:** A single function app only scales out to a maximum of 200 instances. A single instance may process more than one message or request at a time though, so there isn't a set limit on number of concurrent executions.
- **New instance rate:** For HTTP triggers, new instances are allocated, at most, once per second. For non-HTTP triggers, new instances are allocated, at most, once every 30 seconds.

## Limit scale out

You may wish to restrict the maximum number of instances an app used to scale out. This is most common for cases where a downstream component like a database has limited throughput. By default, Consumption plan functions scale out to as many as 200 instances, and Premium plan functions will scale out to as many as 100 instances. You can specify a lower maximum for a specific app by modifying the `functionAppScaleLimit` value. The `functionAppScaleLimit` can be set to 0 or null for unrestricted, or a valid value between 1 and the app maximum.

## Azure Functions scaling in an App service plan

Using an App Service plan, you can manually scale out by adding more VM instances. You can also enable autoscale, though autoscale will be slower than the elastic scale of the Premium plan.

58

## Knowledge check

### Multiple choice

Which of the following Azure Functions hosting plans is best when predictive scaling and costs are required? Ⓒ Functions Premium Plan

Ⓒ App service plan

## Multiple choice

*An organization wants to implement a serverless workflow to solve a business problem. One of the requirements is the solution needs to use a designer-first (declarative) development model. Which of the choices below meets the requirements?*

- ⑥ Azure Functions
- ⑥ Azure Logic Apps
- ⑥ WebJobs

## Summary

In this module, you learned how to:

- Explain functional differences between Azure Functions, Azure Logic Apps, and WebJobs
- Describe Azure Functions hosting plan options
- Describe how Azure Functions scale to meet business needs

59

# Develop Azure Functions

## Introduction

Functions share a few core technical concepts and components, regardless of the language or binding you use.

After completing this module, you'll be able to:

- Explain the key components of a function and how they are structured
- Create triggers and bindings to control when a function runs and where the output is directed
- Connect a function to services in Azure
- Create a function by using Visual Studio Code and the Azure Functions Core

## Tools Explore Azure Functions

## development

A function contains two important pieces - your code, which can be written in a variety of languages, and some config, the *function.json* file. For compiled languages, this config file is generated automatically from annotations in your code. For scripting languages, you must provide the config file yourself.

The *function.json* file defines the function's trigger, bindings, and other configuration settings. Every function has one and only one trigger. The runtime uses this config file to determine the events to monitor and how to pass data into and return data from a function execution. The following is an example *function.json* file.

```
{
  "disabled": false,
  "bindings": [
    // ... bindings here
    {
      "type": "bindingType",
```

```

    "direction": "in",
    "name": "myParamName",
    // ... more depending on binding
  }
]
}

```

The `bindings` property is where you configure both triggers and bindings. Each binding shares a few common settings and some settings which are specific to a particular type of binding. Every binding requires the following settings:

Property	Types	Comments
<code>type</code>	string	Name of binding. For example, <code>queueTrigger</code> .
<code>direction</code>	string	Indicates whether the binding is for receiving data into the function or sending data from the function. For example, <code>in</code> or <code>out</code> .

60

Property	Types	Comments
<code>name</code>	string	The name that is used for the bound data in the function. For example, <code>myQueue</code> .

## Function app

A function app provides an execution context in Azure in which your functions run. As such, it is the unit of deployment and management for your functions. A function app is comprised of one or more individual functions that are managed, deployed, and scaled together. All of the functions in a function app share the same pricing plan, deployment method, and runtime version. Think of a function app as a way to organize and collectively manage your functions.

**Note:** In Functions 2.x all functions in a function app must be authored in the same language. In previous versions of the Azure Functions runtime, this wasn't required.

## Folder structure

The code for all the functions in a specific function app is located in a root project folder that contains a host configuration file. The **host.json**<sup>1</sup> file contains runtime-specific configurations and is in the root folder of the function app. A *bin* folder contains packages and other library files that the function app requires. Specific folder structures required by the function app depend on language:

- **C# compiled (.csproj)**<sup>2</sup>
- **C# script (.csx)**<sup>3</sup>
- **F# script**<sup>4</sup>
- **Java**<sup>5</sup>
- **JavaScript**<sup>6</sup>
- **Python**<sup>7</sup>

## Local development environments

Functions makes it easy to use your favorite code editor and development tools to create and test functions on your local computer. Your local functions can connect to live Azure services, and you can debug them on your local computer using the full Functions runtime.

The way in which you develop functions on your local computer depends on your language and tooling preferences. See **Code and test Azure Functions locally**<sup>8</sup> for more information.

**Warning:** Do not mix local development with portal development in the same function app. When you create and publish functions from a local project, you should not try to maintain or modify project code in the portal.

<sup>1</sup> <https://docs.microsoft.com/azure/azure-functions/functions-host-json>

<sup>2</sup> <https://docs.microsoft.com/azure/azure-functions/functions-dotnet-class-library#functions-class-library-project>

<sup>3</sup> <https://docs.microsoft.com/azure/azure-functions/functions-reference-csharp#folder-structure>

<sup>4</sup> <https://docs.microsoft.com/azure/azure-functions/functions-reference-fsharp#folder-structure>

<sup>5</sup> <https://docs.microsoft.com/azure/azure-functions/functions-reference-java#folder-structure>

<sup>6</sup> <https://docs.microsoft.com/azure/azure-functions/functions-reference-node#folder-structure>

<sup>7</sup> <https://docs.microsoft.com/azure/azure-functions/functions-reference-python#folder-structure>

<sup>8</sup> <https://docs.microsoft.com/azure/azure-functions/functions-develop-local>

## Create triggers and bindings

Triggers are what cause a function to run. A trigger defines how a function is invoked and a function must have exactly one trigger. Triggers have associated data, which is often provided as the payload of the function.

Binding to a function is a way of declaratively connecting another resource to the function; bindings may be connected as *input bindings*, *output bindings*, or both. Data from bindings is provided to the function as parameters.

You can mix and match different bindings to suit your needs. Bindings are optional and a function might have one or multiple input and/or output bindings.

Triggers and bindings let you avoid hardcoding access to other services. Your function receives data (for example, the content of a queue message) in function parameters. You send data (for example, to create a queue message) by using the return value of the function.

## Trigger and binding definitions

Triggers and bindings are defined differently depending on the development language.

Language	Triggers and bindings are configured by...
C# class library	decorating methods and parameters with C# attributes
Java	decorating methods and parameters with Java annotations
JavaScript/PowerShell/Python/TypeScript	updating <i>function.json</i> schema

For languages that rely on *function.json*, the portal provides a UI for adding bindings in the **Integration** tab. You can also edit the file directly in the portal in the **Code + test** tab of your function.

In .NET and Java, the parameter type defines the data type for input data. For instance, use `string` to bind to the text of a queue trigger, a byte array to read as binary, and a custom type to de-serialize to an object. Since .NET class library functions and Java functions don't rely on *function.json* for binding defini

tions, they can't be created and edited in the portal. C# portal editing is based on C# script, which uses *function.json* instead of attributes.

For languages that are dynamically typed such as JavaScript, use the `dataType` property in the *function.json* file. For example, to read the content of an HTTP request in binary format, set `dataType` to `binary`:

```
{
  "dataType": "binary",
  "type": "httpTrigger",
  "name": "req",
  "direction": "in"
}
```

Other options for `dataType` are `stream` and `string`.

## Binding direction

All triggers and bindings have a `direction` property in the *function.json* file:

- For triggers, the direction is always `in`

- Input and output bindings use `in` and `out`
- Some bindings support a special direction `inout`. If you use `inout`, only the **Advanced editor** is available via the **Integrate** tab in the portal.

When you use attributes in a class library to configure triggers and bindings, the direction is provided in an attribute constructor or inferred from the parameter type.

## Azure Functions trigger and binding example

Suppose you want to write a new row to Azure Table storage whenever a new message appears in Azure Queue storage. This scenario can be implemented using an Azure Queue storage trigger and an Azure Table storage output binding.

Here's a *function.json* file for this scenario.

```
{
  "bindings": [
    {
      "type": "queueTrigger",
      "direction": "in",
      "name": "order",
      "queueName": "myqueue-items",
      "connection": "MY_STORAGE_ACCT_APP_SETTING"
    },
    {
      "type": "table",
      "direction": "out",
      "name": "$return",
      "tableName": "outTable",
      "connection": "MY_TABLE_STORAGE_ACCT_APP_SETTING"
    }
  ]
}
```

The first element in the `bindings` array is the Queue storage trigger. The `type` and `direction` properties identify the trigger. The `name` property identifies the function parameter that receives the



queue message content. The name of the queue to monitor is in `queueName`, and the connection string is in the app setting identified by `connection`.

The second element in the `bindings` array is the Azure Table Storage output binding. The `type` and `direction` properties identify the binding. The `name` property specifies how the function provides the new table row, in this case by using the function return value. The name of the table is in `tableName`, and the connection string is in the app setting identified by `connection`.

## C# script example

Here's C# script code that works with this trigger and binding. Notice that the name of the parameter that provides the queue message content is `order`; this name is required because the `name` property value in *function.json* is `order`.

```
#r "Newtonsoft.Json"

using Microsoft.Extensions.Logging;

using Newtonsoft.Json.Linq;

// From an incoming queue message that is a JSON object, add fields
// and write to Table storage
// The method return value creates a new row in Table
Storage public static Person Run(JObject order, ILogger log)
{
    return new Person() {
        PartitionKey = "Orders",
        RowKey = Guid.NewGuid().ToString(),
        Name = order["Name"].ToString(),
        MobileNumber = order["MobileNumber"].ToString() }; }

public class Person
{
    public string PartitionKey { get; set; }
    public string RowKey { get; set; }
    public string Name { get; set; }
    public string MobileNumber { get; set; }
}
```

63

## JavaScript example

The same *function.json* file can be used with a JavaScript function:

```
// From an incoming queue message that is a JSON object, add fields
// and write to Table Storage
// The second parameter to context.done is used as the value for the new
// row
module.exports = function (context, order) {
    order.PartitionKey = "Orders";
    order.RowKey = generateRandomId();

    context.done(null, order);
};

function generateRandomId() {
    return Math.random().toString(36).substring(2, 15) +
        Math.random().toString(36).substring(2, 15);
}
```

```
}
```

## Class library example

In a class library, the same trigger and binding information — queue and table names, storage accounts, function parameters for input and output — is provided by attributes instead of a *function.json* file. Here's an example:

```
public static class QueueTriggerTableOutput
{

    [FunctionName("QueueTriggerTableOutput")]
    [return: Table("outTable", Connection = "MY_TABLE_STORAGE_ACCT_APP_SETTING")]
    public static Person Run(
        [QueueTrigger("myqueue-items", Connection = "MY_STORAGE_ACCT_APP_SETTING")]JObject order,
        ILogger log)
    {
        return new Person() {
            PartitionKey = "Orders",
            RowKey = Guid.NewGuid().ToString(),
            Name = order["Name"].ToString(),
            MobileNumber = order["MobileNumber"].ToString() };
    }

    public class Person
    {
        public string PartitionKey { get; set; }
        public string RowKey { get; set; }
        public string Name { get; set; }
        public string MobileNumber { get; set; }
    }
}
```

64

## Additional resource

For more detailed examples of triggers and bindings please visit:

- [Azure Blob storage bindings for Azure Functions<sup>9</sup>](#)
- [Azure Cosmos DB bindings for Azure Functions 2.x<sup>10</sup>](#)
- [Timer trigger for Azure Functions<sup>11</sup>](#)
- [Azure Functions HTTP triggers and bindings<sup>12</sup>](#)

## Connect functions to Azure services

Your function project references connection information by name from its configuration provider. It does not directly accept the connection details, allowing them to be changed across environments. For example, a trigger definition might include a `connection` property. This might refer to a connection string, but you cannot set the connection string directly in a *function.json*. Instead, you would set `connection` to the name of an environment variable that contains the connection string.

The default configuration provider uses environment variables. These might be set by **Application Settings**<sup>13</sup> when running in the Azure Functions service, or from the **local settings file**<sup>14</sup> when developing locally.

## Connection values

When the connection name resolves to a single exact value, the runtime identifies the value as a *connection string*, which typically includes a secret. The details of a connection string are defined by the service to which you wish to connect.

However, a connection name can also refer to a collection of multiple configuration items. Environment variables can be treated as a collection by using a shared prefix that ends in double underscores `__`. The group can then be referenced by setting the connection name to this prefix.

For example, the `connection` property for a Azure Blob trigger definition might be `Storage1`. As long as there is no single string value configured with `Storage1` as its name, `Storage1__serviceUri` would be used for the `serviceUri` property of the connection. The connection properties are different for each service.

## Configure an identity-based connection

Some connections in Azure Functions are configured to use an identity instead of a secret. Support depends on the extension using the connection. In some cases, a connection string may still be required in Functions even though the service to which you are connecting supports identity-based connections.

**Note:** Identity-based connections are not supported with Durable Functions.

When hosted in the Azure Functions service, identity-based connections use a **managed identity**<sup>15</sup>. The system-assigned identity is used by default, although a user-assigned identity can be specified with the `credential` and `clientId` properties. When run in other contexts, such as local development, your developer identity is used instead, although this can be customized using alternative connection parameters.

## Grant permission to the identity

Whatever identity is being used must have permissions to perform the intended actions. This is typically done by assigning a role in Azure RBAC or specifying the identity in an access policy, depending on the service to which you are connecting.

**Important:** Some permissions might be exposed by the target service that are not necessary for all contexts. Where possible, adhere to the **principle of least privilege**, granting the identity only required privileges.

## Exercise: Create an Azure Function by using Visual Studio Code

In this exercise you'll learn how to create a simple C# function that responds to HTTP requests. After creating and testing the code locally in Visual Studio Code you will deploy to Azure.

## Prerequisites

Before you begin make sure you have the following requirements in place:

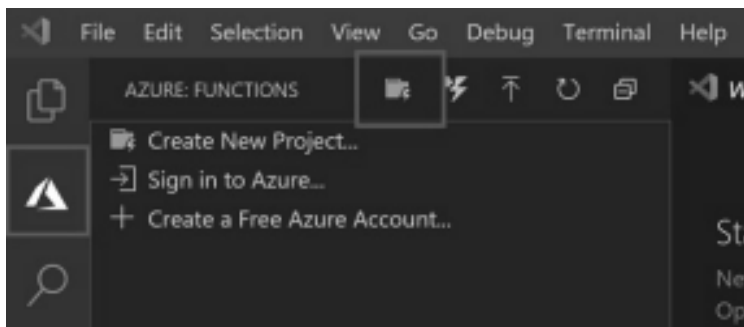
- An Azure account with an active subscription. If you don't already have one, you can sign up for a free trial at <https://azure.com/free>.

- The **Azure Functions Core Tools**<sup>16</sup> version 3.x.
- **Visual Studio Code**<sup>17</sup> on one of the **supported platforms**<sup>18</sup>.
- **.NET Core 3.1**<sup>19</sup> is the target framework for the steps below.
- The **C# extension**<sup>20</sup> for Visual Studio Code.
- The **Azure Functions extension**<sup>21</sup> for Visual Studio Code.

## Create your local project

In this section, you use Visual Studio Code to create a local Azure Functions project in C#. Later in this exercise, you'll publish your function code to Azure.

1. Choose the Azure icon in the Activity bar, then in the **Azure: Functions** area, select **Create new project...**



2. Choose a directory location for your project workspace and choose **Select**.

**Note:** Be sure to select a project folder that is outside of a workspace.

3. Provide the following information at the prompts:

- **Select a language:** Choose C#.
- **Select a .NET runtime:** Choose .NET Core 3.1
- **Select a template for your project's first function:** Choose HTTP trigger.
- **Provide a function name:** Type HttpExample.
- **Provide a namespace:** Type My.Functions.
- **Authorization level:** Choose Anonymous, which enables anyone to call your function endpoint.
- **Select how you would like to open your project:** Choose Add to workspace.

4. Using this information, Visual Studio Code generates an Azure Functions project with an HTTP trigger.

<sup>16</sup> <https://docs.microsoft.com/azure/azure-functions/functions-run-local#install-the-azure-functions-core-tools>

<sup>17</sup> <https://code.visualstudio.com/>

<sup>18</sup> [https://code.visualstudio.com/docs/supporting/requirements#\\_platforms](https://code.visualstudio.com/docs/supporting/requirements#_platforms)

<sup>19</sup> <https://dotnet.microsoft.com/download/dotnet/3.1>

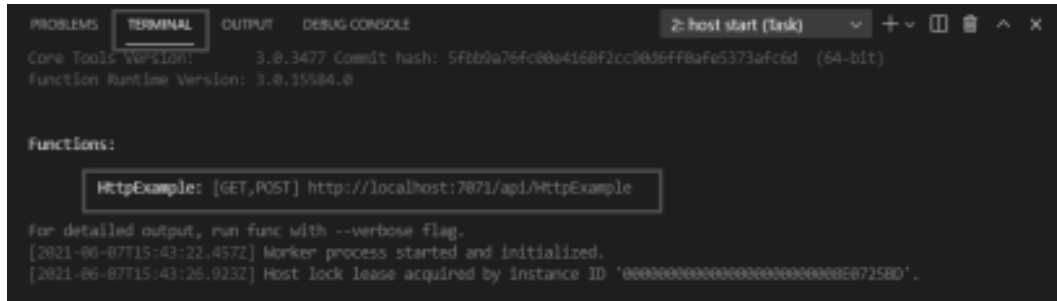
<sup>20</sup> <https://marketplace.visualstudio.com/items?itemName=ms-dotnettools.csharp>

<sup>21</sup> <https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-azurefunctions>

## Run the function locally

Visual Studio Code integrates with Azure Functions Core tools to let you run this project on your local development computer before you publish to Azure.

1. To call your function, press **F5** to start the function app project. Output from Core Tools is displayed in the **Terminal** panel. Your app starts in the **Terminal** panel. You can see the URL endpoint of your HTTP-triggered function running locally.

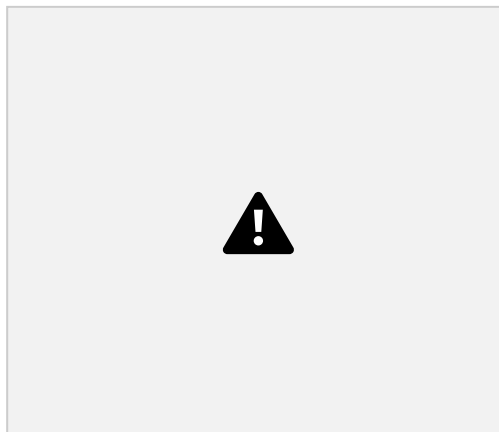


```
PROBLEMS  TERMINAL  OUTPUT  DEBUG CONSOLE
Core Tools Version:      3.0.3477 Commit hash: 5fbb9a76fc80a4168f2cc90d6ff8afe5373afc6d (64-bit)
Function Runtime Version: 3.0.15584.0

Functions:
  HttpExample: [GET,POST] http://localhost:7071/api/HttpExample

For detailed output, run func with --verbose flag.
[2021-06-07T15:43:22.457Z] Worker process started and initialized.
[2021-06-07T15:43:26.923Z] Host lock lease acquired by instance ID '00000000000000000000000000000000E8725BD'.
```

2. With Core Tools running, go to the **Azure: Functions** area. Under **Functions**, expand **Local Project** > **Functions**. Right-click the `HttpExample` function and choose **Execute Function Now....**



3. In **Enter request body** type the request message body value of `{ "name": "Azure" }`. Press **Enter** to send this request message to your function. When the function executes locally and returns a response, a notification is raised in Visual Studio Code. Information about the function execution is shown in **Terminal** panel.
4. Press **Ctrl + C** to stop Core Tools and disconnect the debugger.

After you've verified that the function runs correctly on your local computer, it's time to use Visual Studio Code to publish the project directly to Azure.

## Sign in to Azure

Before you can publish your app, you must sign in to Azure. If you're already signed in, go to the next section.

1. If you aren't already signed in, choose the Azure icon in the Activity bar, then in the **Azure: Functions** area, choose **Sign in to Azure....**