

Microsoft
Official
Course



AZ-400T00

Designing and
Implementing Microsoft
DevOps solutions

AZ-400T00

**Designing and Implementing
Microsoft DevOps solutions**

II Disclaimer

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>¹ are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

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

MICROSOFT LICENSE TERMS

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

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 acquire 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,

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 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 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,

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, not 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.

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

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.
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.**

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 dues 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 contrefaçon 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.

Revised April 2019



Contents

■	Module 0 Welcome	1
	Start Here	1
■	Module 1 Planning for DevOps	15
	Module Overview	15
	Transformation Planning	16
	Project Selection	28
	Team Structures	32
	Migrating to Azure DevOps	38
	Lab	41
	Module Review and Takeaways	42
■	Module 2 Getting Started with Source Control	45
	Module Overview	45
	What is Source Control	46
	Benefits of Source Control	50
	Types of Source Control Systems	52
	Introduction to Azure Repos	67
	Introduction to GitHub	68
	Migrating from Team Foundation Version Control (TFVC) to Git in Azure Repos	71
	Authenticating to Git in Azure Repos	74
	Lab	76
	Module Review and Takeaways	77
■	Module 3 Scaling Git for Enterprise DevOps	79
	Module Overview	79
	How to Structure Your Git Repo	80
	Git Branching Workflows	81
	Collaborating with Pull Requests in Azure Repos	99
	Why Care About GitHooks	106
	Fostering Inner Source	111
	Lab	118
	Module Review and Takeaways	119
■	Module 4 Consolidating Artifacts and Designing a Dependency Management Strategy	121
	Module Overview	121
	Packaging Dependencies	122

Package Management	125
Migrating and Consolidating Artifacts	135
Lab	136
Module Review and Takeaways	137
Module 5 Implementing Continuous Integration with Azure Pipelines	139
Module Overview	139
The Concept of Pipelines in DevOps Azure Pipelines	140
Azure Pipelines	142
Evaluate Use of Hosted vs Private Agents	146
Agent Pools	147
Pipelines and Concurrency	150
Azure DevOps and Open Source projects (Public Projects)	154
Azure Pipelines YAML vs Visual Designer	156
Continuous Integration Overview	158
Implementing a Build Strategy	163
Integration with Azure Pipelines	166
Integrate external source control with Azure Pipelines	182
Set Up Private Agents	183
Analyze and Integrate Docker Multi-Stage Builds	187
Labs	191
Module Review and Takeaways	192
Module 6 Managing Application Config and Secrets	197
Module Overview	197
Introduction to Security	198
Implement Secure and Compliant Development Proces	202
Rethinking Application Config Data	210
Manage Secrets, Tokens, and Certificates	213
Implement Tools for Managing Security and Compliance in a Pipeline	219
Lab	228
Module Review and Takeaways	229
Module 7 Managing Code Quality and Security Policies	231
Module Overview	231
Managing Code Quality	232
Managing Security Policies	239
Lab	242
Module Review and Takeaways	243
Module 8 Implementing a Container Build Strategy	245
Module Overview	245
Managing Code Quality	246
Lab	258
Module Review and Takeaways	259
Module 9 Manage Artifact Versioning, Security, and Compliance	261
Module Overview	261
Package Security	262
Open source software	266
Integrating license and vulnerability scans	270
Implement a versioning strategy	272
Lab	278
Module Review and Takeaways	279

Module 10 Design a Release Strategy	281
Module Overview	281
Introduction to Continuous Delivery	283
Release strategy recommendations	288
Building a High-Quality Release pipeline	322
Choosing a deployment pattern	326
Choosing the right release management tool	327
Module Review and Takeaways	334
Module 11 Set up a Release Management Workflow	337
Module Overview	337
Create a Release Pipeline	339
Provision and Configure Environments	368
Manage and Modularize Tasks and Templates	378
Integrate Secrets with the release pipeline	389
Configure Automated Integration and Functional Test Automation	398
Automate Inspection of Health	401
Labs	413
Module Review and Takeaways	421
Module 12 Implement an Appropriate Deployment Pattern	423
Module Overview	423
Introduction to Deployment Patterns	425
Implement Blue Green Deployment	429
Feature Toggles	442
Canary Releases	446
Dark Launching	452
AB Testing	453
Progressive Exposure Deployment	454
Lab	456
Module Review and Takeaways	457
Module 13 Implement Process for Routing System Feedback to Development Teams	459
Module Overview	459
Implement Tools to Track System Usage, Feature Usage, and Flow	461
Implement Routing for Mobile Application Crash Report Data	484
Develop Monitoring and Status Dashboards	489
Integrate and Configure Ticketing Systems	495
Lab	504
Module Review and Takeaways	505
Module 14 Infrastructure and Configuration Azure Tools	507
Module Overview	507
Infrastructure as Code and Configuration Management	508
Create Azure Resources using ARM Templates	518
Create Azure Resources using Azure CLI	537
Create Azure Resources by using Azure PowerShell	544
Desired State Configuration (DSC)	553
Azure Automation with DevOps	565
Additional Automation Tools	589
Lab	593
Module Review and Takeaways	594
Module 15 Azure Deployment Models and Services	601

Module Overview	601
Deployment Modules and Options	602
Azure Infrastructure-as-a-Service (IaaS) Services	605
Azure Platform-as-a-Service (PaaS) Services	620
Serverless and HPC Computer Services	640
Azure Service Fabric	650
Lab	668
Module Review and Takeaways	669
Module 16 Create and Manage Kubernetes Service Infrastructure	677
Module Overview	677
Azure Kubernetes Service (AKS)	679
Lab	692
Module Review and Takeaways	693
Module 17 Third Party Infrastructure as Code Tools available with Azure	695
Module Overview	695
Chef	696
Puppet	701
Ansible	704
Terraform	720
Labs	742
Module Review and Takeaways	743
Module 18 Implement Compliance and Security in your Infrastructure	749
Module Overview	749
Security and Compliance Principles with DevOps	751
Azure security Center	765
Lab	778
Module Review and Takeaways	779
Module 19 Recommend and Design System Feedback Mechanisms	787
Module Overview	787
The Inner Loop	789
Continuous Experimentation Mindset	795
Design Practices to Measure End-User Satisfaction	801
Design processes to capture and analyze user feedback	807
Design process to automate application analytics	818
Lab	822
Module Review and Takeaways	823
Module 20 Optimize Feedback Mechanisms	825
Module Overview	825
Site Reliability Engineering	826
Analyze telemetry to establish a baseline	828
Perform ongoing tuning to reduce meaningless or non-actionable alerts	835
Analyze alerts to establish a baseline	839
Blameless Retrospectives and a Just Culture	846
Module Review and Takeaways	849

Module 0 Welcome

Start Here

Azure DevOps Curriculum

Welcome to the **Designing and Implementing Microsoft DevOps Solutions** course. This course will help you prepare for the AZ-400, **Microsoft Azure DevOps Solutions¹** certification exam.

The DevOps certification exam is for DevOps professionals who combine people, process, and technologies to continuously deliver valuable products and services that meet end user needs and business objectives. DevOps professionals streamline delivery by optimizing practices, improving communications and collaboration, and creating automation. They design and implement strategies for application code and infrastructure that allow for continuous integration, continuous testing, continuous delivery, and continuous monitoring and feedback.

Exam candidates must be proficient with Agile practices. They must be familiar with both Azure administration and Azure development and experts in at least one of these areas. Azure DevOps professionals must be able to design and implement DevOps practices for version control, compliance, infrastructure as code, configuration management, build, release, and testing by using Azure technologies.

There are seven exam study areas.

AZ-400 Study Areas	Weights
Develop an Instrumentation Strategy	5-10%
Develop a Site Reliability Engineering (SRE) Strategy	5-10%
Develop a security and compliance plan	10-15%
Manage source control	10 - 15%
Facilitate communication and collaboration	10-15%
Define and implement continuous integration	20-25%
Define and implement a continuous delivery and release management strategy	10-15%

¹ <https://www.microsoft.com/en-us/learning/exam-AZ-400.aspx>

About this Course

Course Description

This course provides the knowledge and skills to design and implement DevOps processes and practices. Students will learn how to plan for DevOps, use source control, scale Git for an enterprise, consolidate artifacts, design a dependency management strategy, manage secrets, implement continuous integration, implement a container build strategy, design a release strategy, set up a release management workflow, implement a deployment pattern, and optimize feedback mechanisms.

Level: Advanced

Audience

Students in this course are interested in implementing DevOps processes or in passing the Microsoft Azure DevOps Solutions certification exam.

Pre-requisites

Fundamental knowledge about Azure, version control, Agile software development, and core software development principles. It would be helpful to have experience in an organization that delivers software.

Expected learning

After completing this course, students will be able to:

- Plan for the transformation with shared goals and timelines
- Select a project and identify project metrics and Key Performance Indicators (KPI's)
- Create a team and agile organizational structure
- Describe the benefits of using Source Control
- Migrate from TFVC to Git
- Scale Git for Enterprise DevOps
- Recommend artifact management tools and practices
- Abstract common packages to enable sharing and reuse
- Migrate and consolidate artifacts
- Migrate and integrate source control measures
- Manage application config and secrets
- Develop a project quality strategy
- Plan for secure development practices and compliance rules
- Implement and manage build infrastructure
- Explain why continuous integration matters
- Implement continuous integration using Azure DevOps
- Manage code quality including: technical debt SonarCloud, and other tooling solutions
- Manage security policies with open source and OWASP
- Implement a container strategy including how containers are different from virtual machines and how microservices use containers
- Implement containers using Docker

- Inspect open source software packages for security and license compliance to align with corporate standards
- Configure build pipeline to access package security and license rating
- Configure secure access to package feeds
- Inspect codebase to identify code dependencies that can be converted to packages
- Identify and recommend standardized package types and versions across the solution
- Refactor existing build pipelines to implement version strategy that publishes packages
- Manage security and compliance
- Differentiate between a release and a deployment
- Define the components of a release pipeline
- Explain things to consider when designing your release strategy
- Classify a release versus a release process and outline how to control the quality of both
- Describe the principle of release gates and how to deal with release notes and documentation
- Explain deployment patterns, both in the traditional sense and in the modern sense
- Choose a release management tool
- Explain the terminology used in Azure DevOps and other Release Management Tooling
- Describe what a Build and Release task is, what it can do, and some available deployment tasks
- Classify an Agent, Agent Queue, and Agent Pool
- Explain why you sometimes need multiple release jobs in one release pipeline
- Differentiate between multi-agent and multi-configuration release job
- Use release variables and stage variables in your release pipeline
- Deploy to an environment securely using a service connection
- Embed testing in the pipeline
- List the different ways to inspect the health of your pipeline and release by using alerts, service hooks, and reports
- Create a release gate
- Describe deployment patterns
- Implement Blue Green Deployment
- Implement Canary Release
- Implement Progressive Exposure Deployment
- Configure crash report integration for client applications
- Develop monitoring and status dashboards
- Implement routing for client application crash report data
- Implement tools to track system usage, feature usage, and flow
- Integrate and configure ticketing systems with development team's work management
- Apply infrastructure and configuration as code principles.

- Deploy and manage infrastructure using Microsoft automation technologies such as ARM templates, PowerShell, and Azure CLI
- Describe deployment models and services that are available with Azure
- Deploy and configure a Managed Kubernetes cluster
- Deploy and configure infrastructure using 3rd party tools and services with Azure, such as Chef, Puppet, Ansible, and Terraform
- Define an infrastructure and configuration strategy and appropriate toolset for a release pipeline and application infrastructure
- Implement compliance and security in your application infrastructure
- Design practices to measure end-user satisfaction
- Design processes to capture and analyze user feedback from external sources
- Design routing for client application crash report data
- Recommend monitoring tools and technologies
- Recommend system and feature usage tracking tools
- Analyze alerts to establish a baseline
- Analyze telemetry to establish a baseline
- Perform live site reviews and capture feedback for system outages
- Perform ongoing tuning to reduce meaningless or non-actionable alerts

Syllabus

This course includes content that will help you prepare for the Microsoft Azure DevOps Solution certification exam. Other content is included to ensure you have a complete picture of DevOps. The course content includes a mix of graphics, reference links, module review questions, and hands-on labs.

Module 1 – Planning for DevOps

- Lesson 1: Module Overview
- Lesson 2: Transformation Planning
- Lesson 3: Project Selection
- Lesson 4: Team Structures
- Lesson 5: Migrating to Azure DevOps
- Lesson 6: Lab
 - Agile Planning and Portfolio Management with Azure Boards
- Lesson 7: Module Review and Takeaways

Module 2 – Getting Started with Source Control

- Lesson 1: Module Overview
- Lesson 2: What is Source Control
- Lesson 3: Benefits of Source Control
- Lesson 4: Types of Source Control Systems

- Lesson 5: Introduction to Azure Repos
- Lesson 6: Introduction to GitHub
- Lesson 7: Migrating from Team Foundation Version Control (TFVC) to Git in Azure Repos
- Lesson 8: Authenticating to Git in Azure Repos
- Lesson 9: Lab
 - Version Controlling with Git
- Lesson 10: Module Review and Takeaways

Module 3 – Scaling Git for Enterprise DevOps

- Lesson 1: Module Overview
- Lesson 2: How to Structure Your Git Repo
- Lesson 3: Git Branching Workflows
- Lesson 4: Collaborating with Pull Requests in Azure Repos
- Lesson 5: Why Care About GitHooks
- Lesson 6: Fostering Inner Source
- Lesson 7: Lab
 - Code Review with Pull Requests
- Lesson 8: Module Review and Takeaways

Module 4 - Consolidating Artifacts and Designing a Dependency Management Strategy

- Lesson 1: Module Overview
- Lesson 2: Packaging Dependencies
- Lesson 3: Package Management
- Lesson 4: Migrating and Consolidating Artifacts
- Lesson 5: Lab
 - Updating Packages
- Lesson 5: Module Review and Takeaways

Module 5 - Implementing Continuous Integration with Azure Pipelines

- Lesson 1: Module Overview
- Lesson 2: The concept of pipelines in DevOps
- Lesson 3: Azure Pipelines
- Lesson 4: Evaluate use of Hosted vs Private Agents
- Lesson 5: Agent pools
- Lesson 6: Pipelines and concurrency
- Lesson 7: Azure DevOps and Open Source projects (Public Projects)
- Lesson 8: Azure Pipelines YAML vs Visual Designer

- Lesson 9: Continuous Integration Overview
- Lesson 10: Implementing a Build Strategy
- Lesson 11: Integration with Azure Pipelines
- Lesson 12: Integrate external source control with Azure Pipelines
- Lesson 13: Set Up Private Agents
- Lesson 14: Analyze and integrate Docker multi-stage builds
- Lesson 15: Labs

- Enabling Continuous Integration with Azure Pipelines
- Integrating External Source Control with Azure Pipelines
- Integrate Jenkins with Azure Pipelines
- Deploying a Multi-Container Application
- Lesson 16: Module Review and Takeaways

Module 6 - Managing Application Config & Secrets

- Lesson 1: Module Overview
- Lesson 2: Introduction to Security
- Lesson 3: Implement secure and compliant development process
- Lesson 4: Rethinking application config data
- Lesson 5: Manage secrets, tokens, and certificates
- Lesson 6: Implement tools for managing security and compliance in a pipeline
- Lesson 7: Lab
 - Integrating Azure Key Vault with Azure DevOps
- Lesson 8: Module Review and Takeaways

Module 7 - Managing Code Quality and Security Policies

- Lesson 1: Module Overview
- Lesson 2: Managing Code Quality
- Lesson 3: Managing Security Policies
- Lesson 4: Lab
 - Managing Technical Debt with Azure DevOps and SonarCloud
- Lesson 5: Module Review and Takeaways

Module 8 - Implementing a Container Build Strategy

- Lesson 1: Module Overview
- Lesson 2: Implementing a Container Build Strategy

- Lesson 3: Lab
 - Modernizing Existing ASP.NET Apps with Azure
- Lesson 4: Module Review and Takeaways

Module 9 - Manage Artifact versioning, security & compliance

- Lesson 1: Module Overview
- Lesson 2: Package security
- Lesson 3: Open source software
- Lesson 4: Integrating license and vulnerability scans
- Lesson 5: Implement a versioning strategy (git version)
- Lesson 6: Lab
 - Manage Open Source Security and License with WhiteSource
- Lesson 7: Module Review and Takeaways

Module 10 - Design a Release Strategy

- Lesson 1: Module Overview
- Lesson 2: Introduction to Continuous Delivery
- Lesson 3: Release strategy recommendations
- Lesson 4: Building a High-Quality Release pipeline
- Lesson 5: Choosing a deployment pattern
- Lesson 6: Choosing the right release management tool
- Lesson 7: Module Review and Takeaways

Module 11 - Set up a Release Management Workflow

- Lesson 1: Module Overview
- Lesson 2: Create a Release Pipeline
- Lesson 3: Provision and Configure Environments
- Lesson 4: Manage and Modularize Tasks and Templates
- Lesson 5: Integrate Secrets with the release pipeline
- Lesson 6: Configure Automated Integration and Functional Test Automation
- Lesson 7: Automate Inspection of Health
- Lesson 8: Labs
 - Configuring Pipelines as Code with YAML
 - Setting up secrets in the pipeline with Azure Key vault
 - Setting up and Running Functional Tests
 - Using Azure Monitor as release gate
 - Creating a release Dashboard

- Lesson 9: Module Review and Takeaways

Module 12 - Implement an appropriate deployment pattern

- Lesson 1: Module Overview
- Lesson 2: Introduction to Deployment Patterns
- Lesson 3: Implement Blue Green Deployment
- Lesson 4: Feature Toggles
- Lesson 5: Canary Releases
- Lesson 6: Dark Launching
- Lesson 7: AB Testing
- Lesson 8: Progressive Exposure Deployment
- Lesson 9: Lab
 - Feature Flag Management with LaunchDarkly and Azure DevOps
- Lesson 10: Module Review and Takeaways

Module 13 - Implement process for routing system feedback to development teams

- Lesson 1: Module Overview
- Lesson 2: Implement tools to track system usage, feature usage, and flow
- Lesson 3: Implement routing for mobile application crash report data
- Lesson 4: Develop monitoring and status dashboards
- Lesson 5: Integrate and configure ticketing systems
- Lesson 6: Lab
 - Monitoring Application Performance
- Lesson 6: Module Review and Takeaways

Module 14 - Infrastructure and Configuration Azure Tools

- Lesson 1: Module Overview
- Lesson 2: Infrastructure as Code and Configuration Management
- Lesson 3: Create Azure Resources using ARM Templates
- Lesson 4: Create Azure Resources using Azure CLI
- Lesson 5: Create Azure Resources by using Azure PowerShell
- Lesson 6: Desired State Configuration (DSC)
- Lesson 7: Azure Automation with DevOps
- Lesson 8: Additional Automation Tools
- Lesson 9: Lab
 - Azure Deployments using Resource Manager templates
- Lesson 10: Module Review and Takeaways

Module 15 - Azure Deployment Models and Services

- Lesson 1: Module Overview
- Lesson 2: Deployment Modules and Options
- Lesson 3: Azure Infrastructure-as-a-Service (IaaS) Services
- Lesson 4: Azure Platform-as-a-Service (PaaS) Services
- Lesson 5: Serverless and HPC Computer Services
- Lesson 6: Azure Service Fabric
- Lesson 7: Lab
 - Azure Automation - IaaS or PaaS deployment
- Lesson 8: Module Review and Takeaways

Module 16 - Create and Manage Kubernetes Service Infrastructure

- Lesson 1: Module Overview
- Lesson 2: Azure Kubernetes Service (AKS)
- Lesson 3: Lab
 - Deploy a multi-container application to Azure Kubernetes Service
- Lesson 4: Module Review and Takeaways

Module 17 - Third Party Infrastructure as Code Tools available with Azure

- Lesson 1: Module Overview
- Lesson 2: Chef
- Lesson 3: Puppet
- Lesson 4: Ansible
- Lesson 5: Terraform
- Lesson 6: Labs
 - Infrastructure as Code
 - Automating your infrastructure deployments in the Cloud with Terraform and Azure Pipelines
- Lesson 7: Module Review and Takeaways

Module 18 - Implement Compliance and Security in your Infrastructure

- Lesson 1: Module Overview
- Lesson 2: Security and Compliance Principles with DevOps
- Lesson 3: Azure security Center
- Lesson 4: Lab
 - Implement Security and Compliance in an Azure DevOps pipeline
- Lesson 5: Module Review and Takeaways

Module 19 - Recommend and design system feedback mechanisms

- Lesson 1: Module Overview
- Lesson 2: The inner loop
- Lesson 3: Continuous Experimentation mindset
- Lesson 4: Design practices to measure end-user satisfaction
- Lesson 5: Design processes to capture and analyze user feedback
- Lesson 6: Design process to automate application analytics
- Lesson 7: Lab
 - Integration between Azure DevOps and Teams
- Lesson 8: Module Review and Takeaways

Module 20 - Optimize Feedback Mechanisms

- Lesson 1: Module Overview
- Lesson 2: Site Reliability Engineering
- Lesson 3: Analyze telemetry to establish a baseline
- Lesson 4: Perform ongoing tuning to reduce meaningless or non-actionable alerts
- Lesson 5: Analyze alerts to establish a baseline
- Lesson 6: Blameless Retrospectives and a Just Culture
- Lesson 7: Module Review and Takeaways

Lab Environment Setup

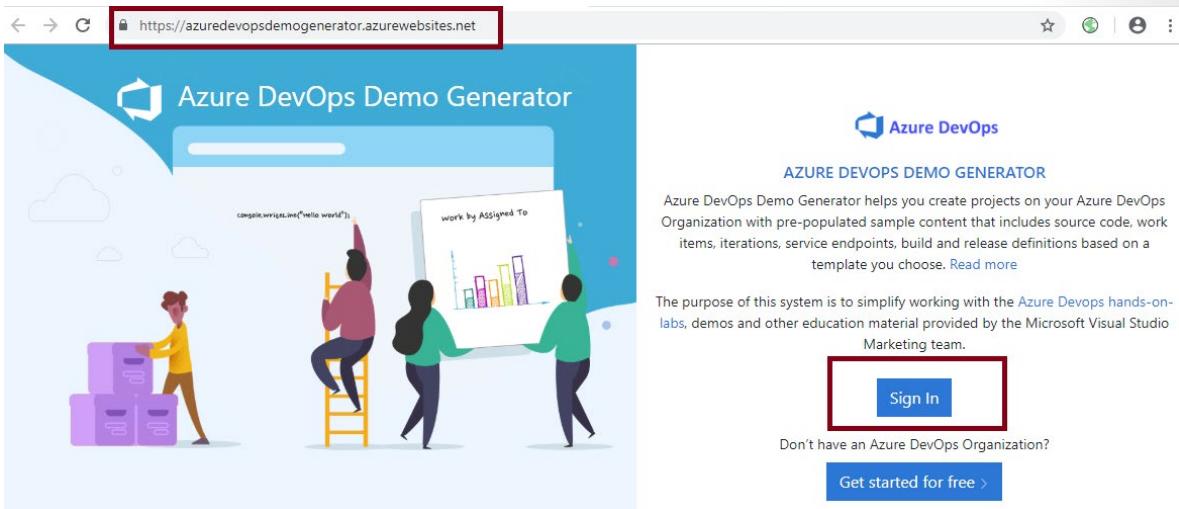
We highly recommend that you complete the assigned hands-on lab work. To do the hands-on labs, you will need to complete the following steps.

1. Sign up for a free **Azure DevOps account**². Use the Sign up for a free account button to create your account. If you already have an account proceed to the next step.
2. Sign up for free **Azure Account**³. If you already have an account proceed to the next step.
3. To make it easier to get set up for testing Azure DevOps, a **Azure DevOps Generator Demo**⁴ program has been created. Click the **Sign In** button and sign in with your Azure DevOps account.

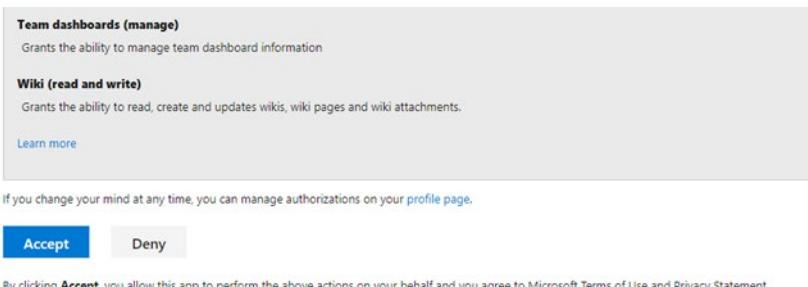
² <https://www.azuredevopslabs.com/>

³ <https://azure.microsoft.com/en-us/free/>

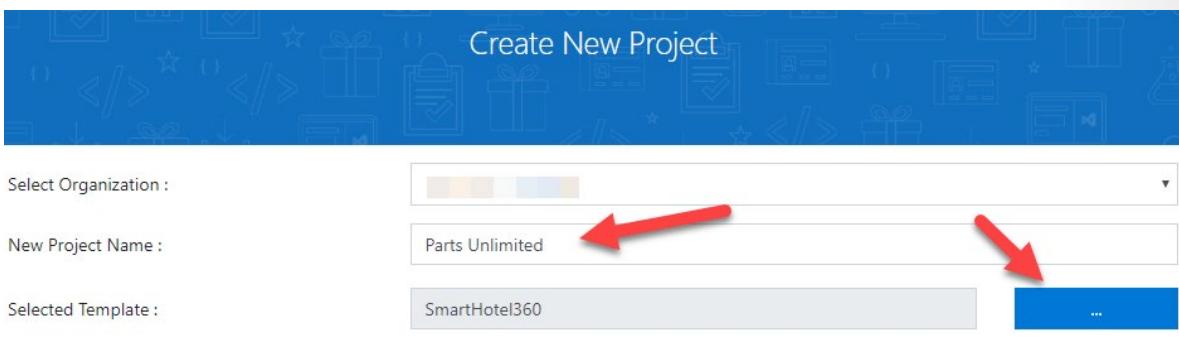
⁴ <https://azuredemogenerator.azurewebsites.net/>



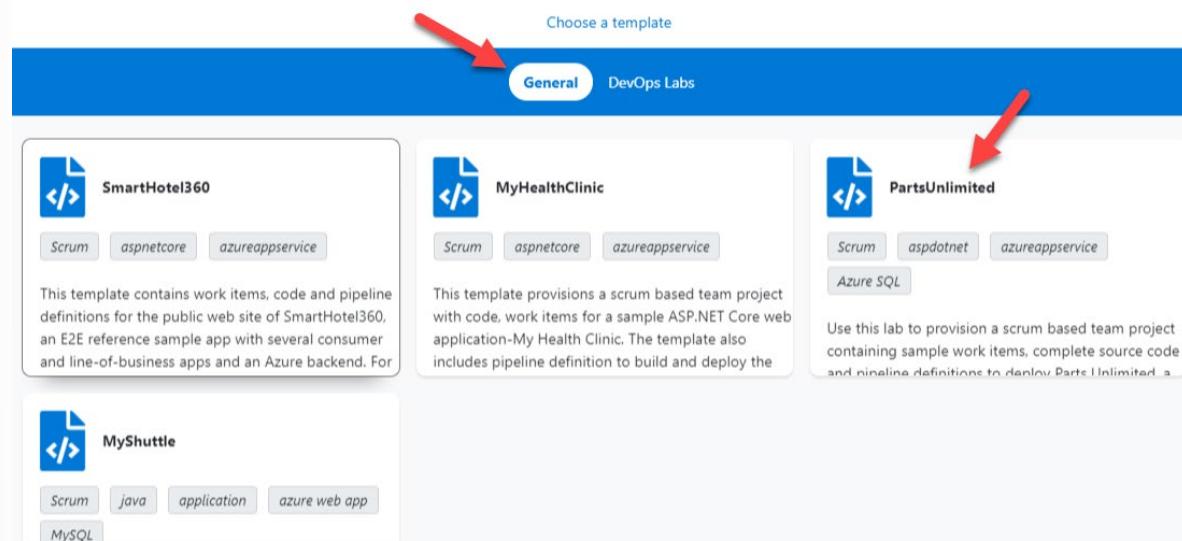
4. You will then be asked to confirm that the generator site can have permission to create objects in your Azure DevOps account.



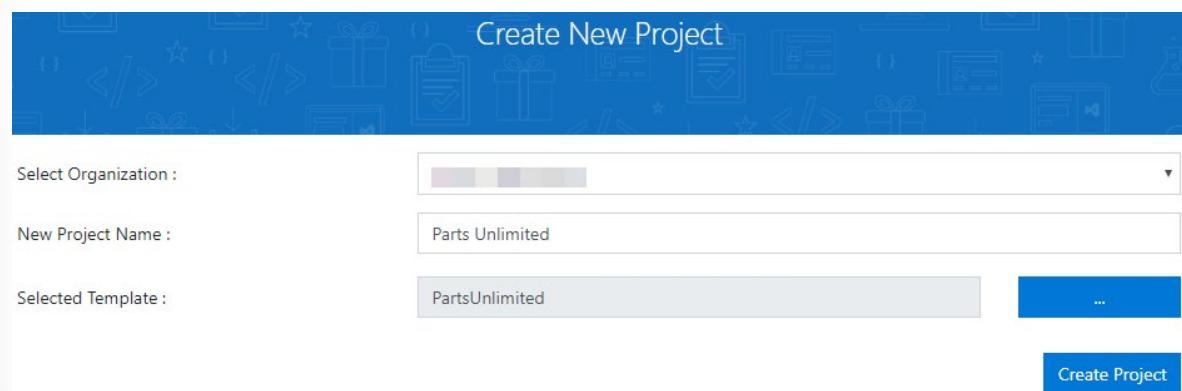
5. If you agree, click the **Accept** button and you should be greeted by the Create New Project screen:



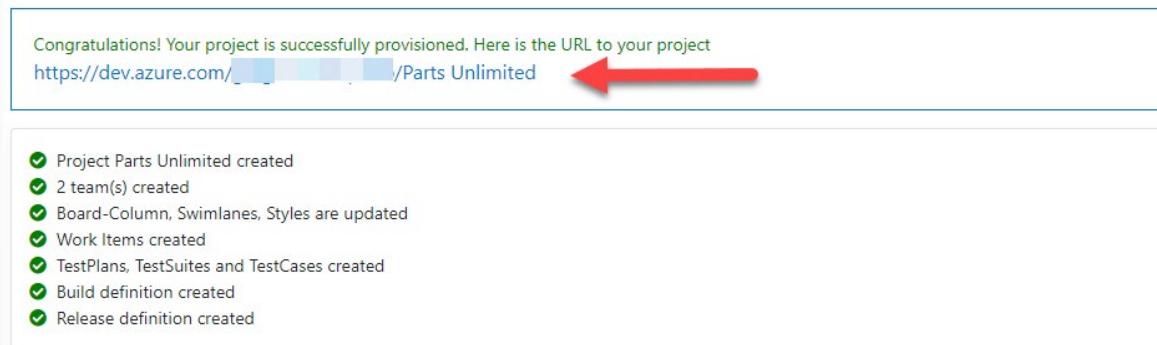
6. Select the appropriate organization (if you have more than one) and enter **Parts Unlimited** as the **New Project Name**, then click the ellipsis to view the available templates. These will change over time but you should see a screen similar to the following:



7. From the **General** tab, choose **PartsUnlimited**, then click **Select Template**.



8. Now that the Create New Project screen is completed, click **Create Project** to begin the project creation phase.



9. When the project is successfully completed, click the link provided to go to your team project within Azure DevOps.

-
- ✓ Note that because Azure DevOps was previously called VSTS (Visual Studio Team Services), some of the existing hands-on labs might refer to VSTS rather than Azure DevOps.

Module 1 Planning for DevOps

Module Overview

Module Overview

Plan before you act. This module will help you understand what DevOps is and how to plan for a DevOps transformation journey.

Learning Objectives

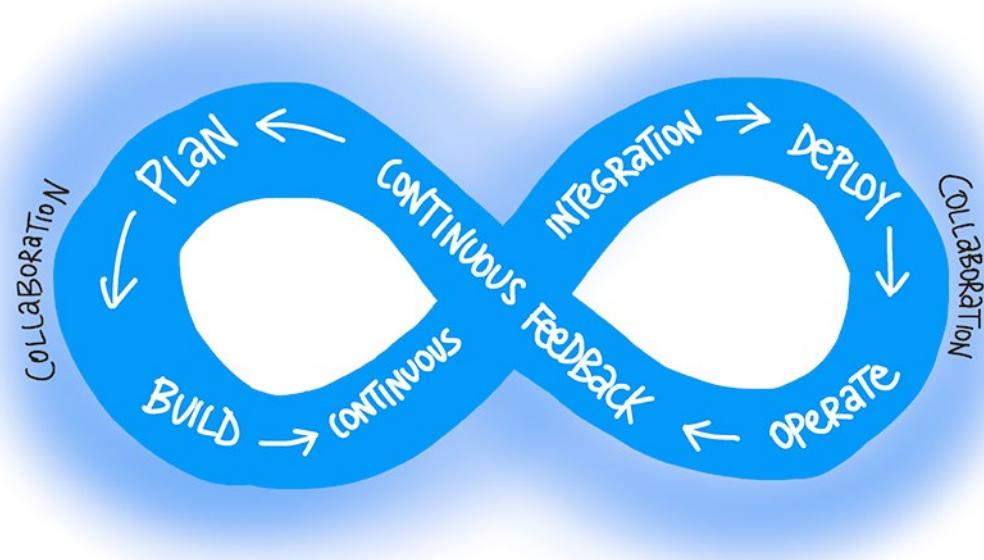
After completing this module, students will be able to:

- Plan for the transformation with shared goals and timelines
- Select a project and identify project metrics and Key Performance Indicators (KPI's)
- Create a team and agile organizational structure
- Design a tool integration strategy
- Design a license management strategy (e.g. Azure DevOps users)
- Design a strategy for end-to-end traceability from work items to working software
- Design an authentication and access strategy
- Design a strategy for integrating on-premises and cloud resources

Transformation Planning

What is DevOps

DevOps is the union of people, process, and products to enable continuous delivery of value to our end users. The contraction of "Dev" and "Ops" refers to replacing siloed Development and Operations to create multidisciplinary teams that now work together with shared and efficient practices and tools. Essential DevOps practices include agile planning, continuous integration, continuous delivery, and monitoring of applications. DevOps is a continuous journey.



Understand your Cycle Time

Let's start with a basic assumption about software development. We'll describe it with the OODA (Observe, Orient, Decide, Act) loop. Originally designed to keep fighter pilots from being shot out of the sky, the OODA loop is a good way to think about staying ahead of your competitors. You start with observation of business, market, needs, current user behavior, and available telemetry data. Then you orient with the enumeration of options for what you can deliver, perhaps with experiments. Next you decide what to pursue, and you act by delivering working software to real users. All of this occurs in some cycle time.

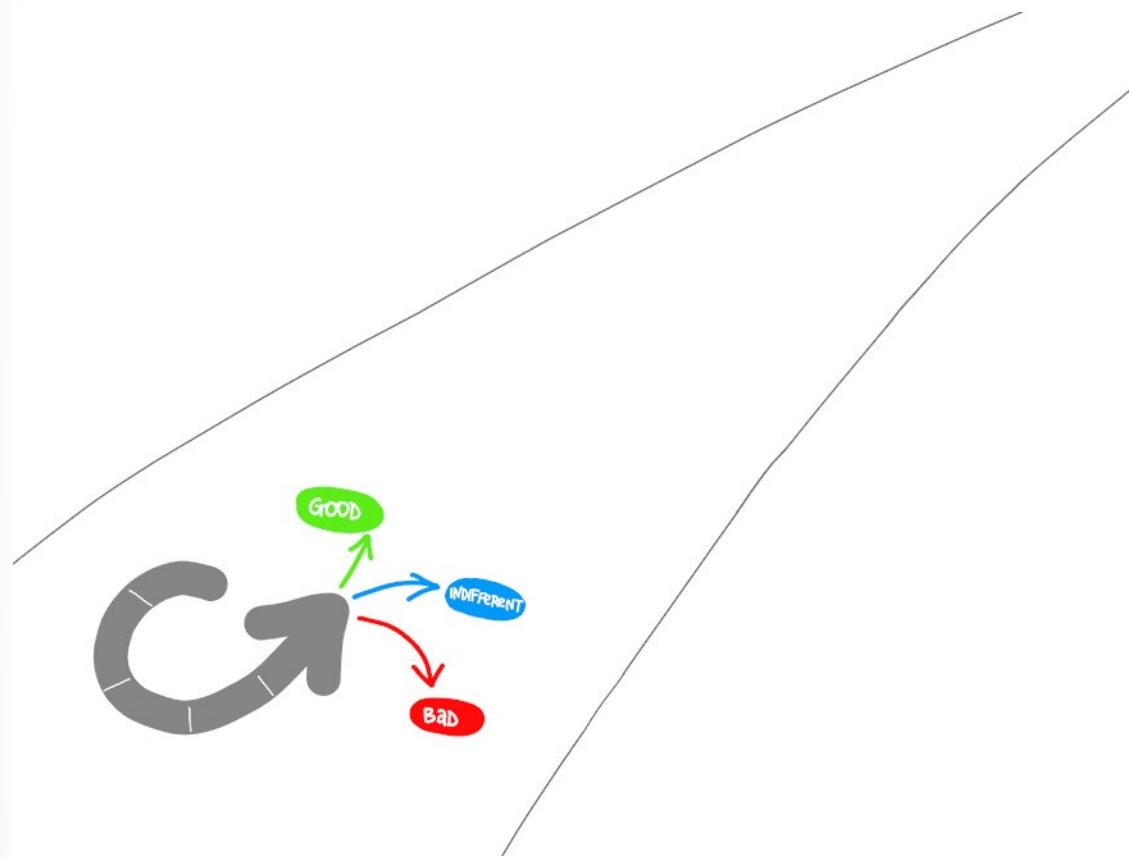


Become Data-Informed

Hopefully, you use data to inform what to do in your next cycle. Many experience reports tell us that roughly one-third of the deployments will have negative business results, roughly one third will have positive results, and one third will make no difference. Ideally, you would like to fail fast on those that don't advance the business and double down on those that support the business. Sometimes this is called pivot or persevere.

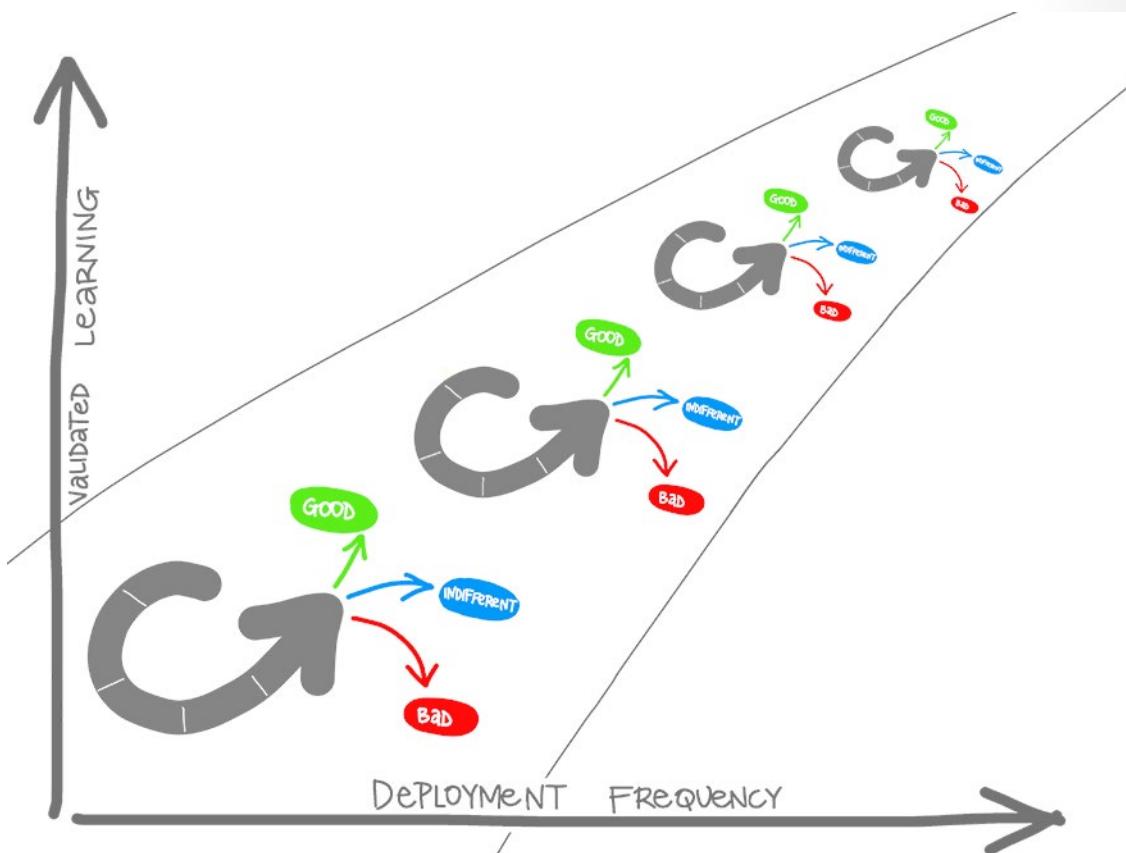
Strive for Validated Learning

How quickly you can fail fast or double down is determined by how long that loop takes, or in lean terms, by your cycle time. Your cycle time determines how quickly you can gather feedback to determine what happens in the next loop. The feedback that you gather with each cycle should be real, actionable data. This is called validated learning.



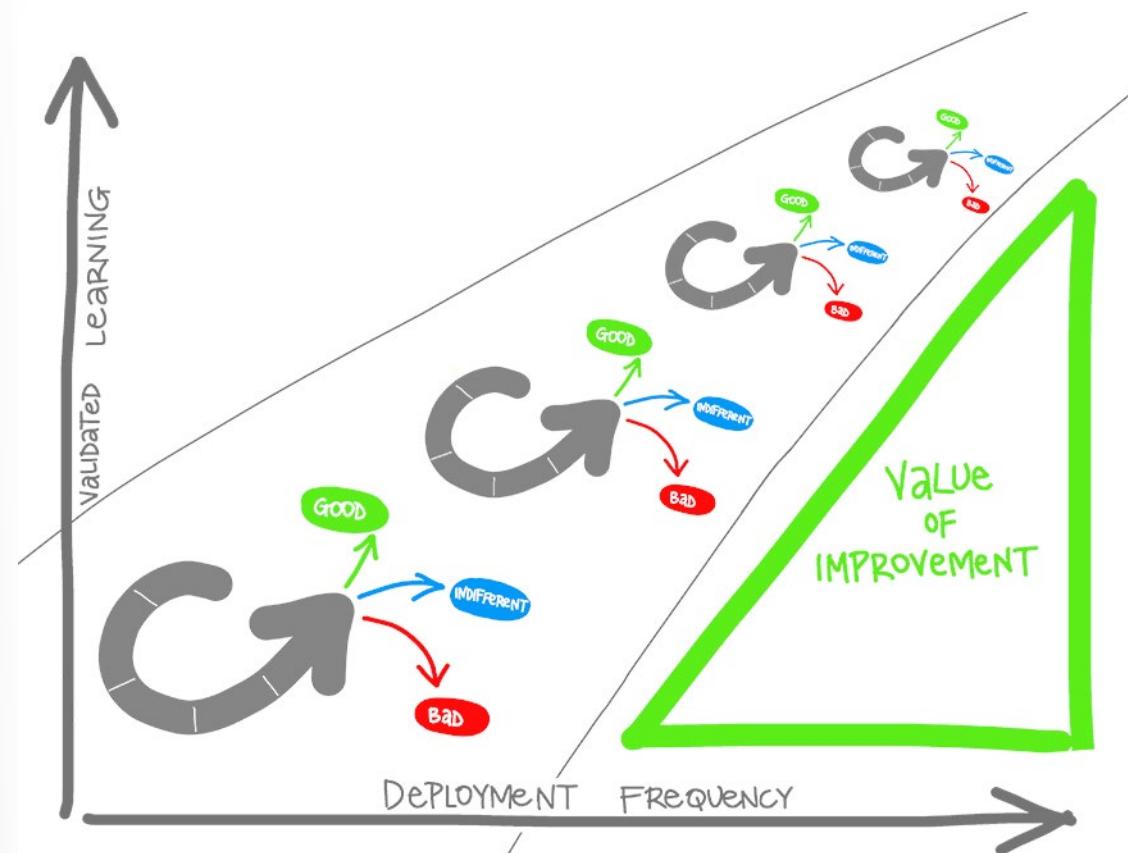
Shorten Your Cycle Time

When you adopt DevOps practices, you shorten your cycle time by working in smaller batches, using more automation, hardening your release pipeline, improving your telemetry, and deploying more frequently.



Optimize Validated Learning

The more frequently you deploy, the more you can experiment, the more opportunity you have to pivot or persevere, and to gain validated learning each cycle. This acceleration in validated learning is the value of improvement. Think of it as the sum of improvements that you achieve and the failures that you avoid.

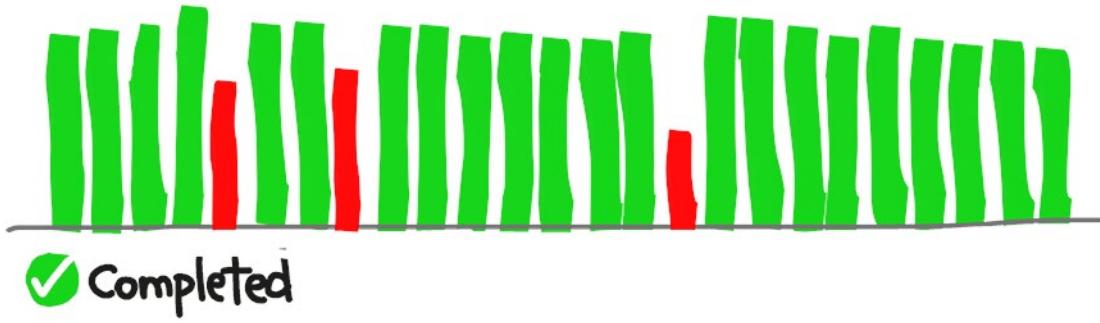


The DevOps Journey

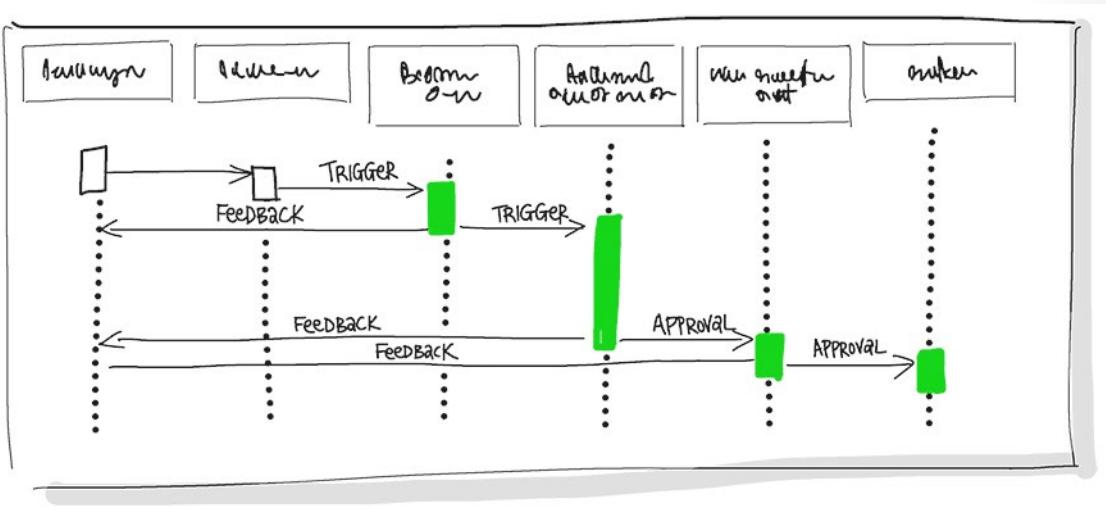
Remember, the goal is to shorten cycle time. Start with the release pipeline. How long does it take you to deploy a change of one line of code or configuration? Ultimately, that's the brake on your velocity.

1. Continuous Integration drives the ongoing merging and testing of code, which leads to finding defects early. Other benefits include less time wasted on fighting merge issues and rapid feedback for development teams.

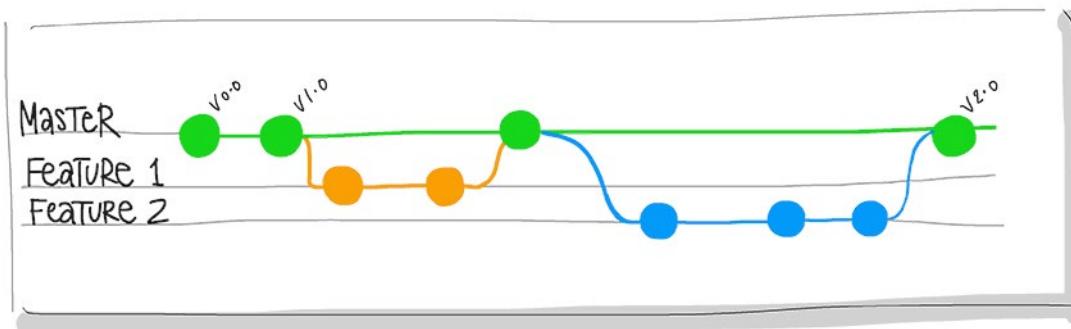
BUILD Succeeded



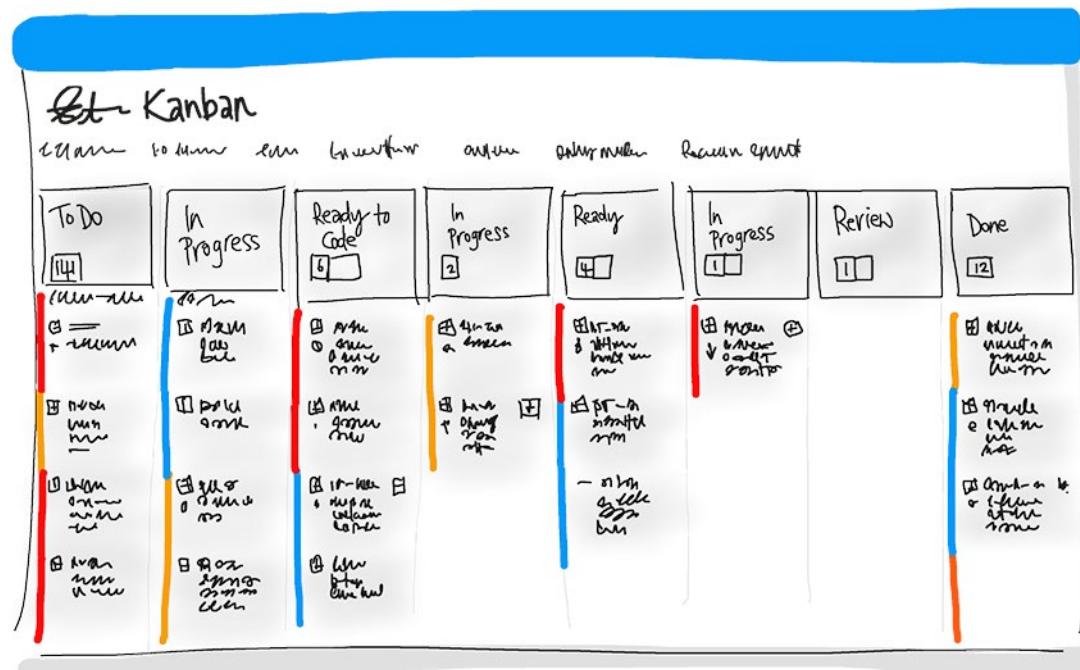
2. Continuous Delivery of software solutions to production and testing environments helps organizations quickly fix bugs and respond to ever-changing business requirements.



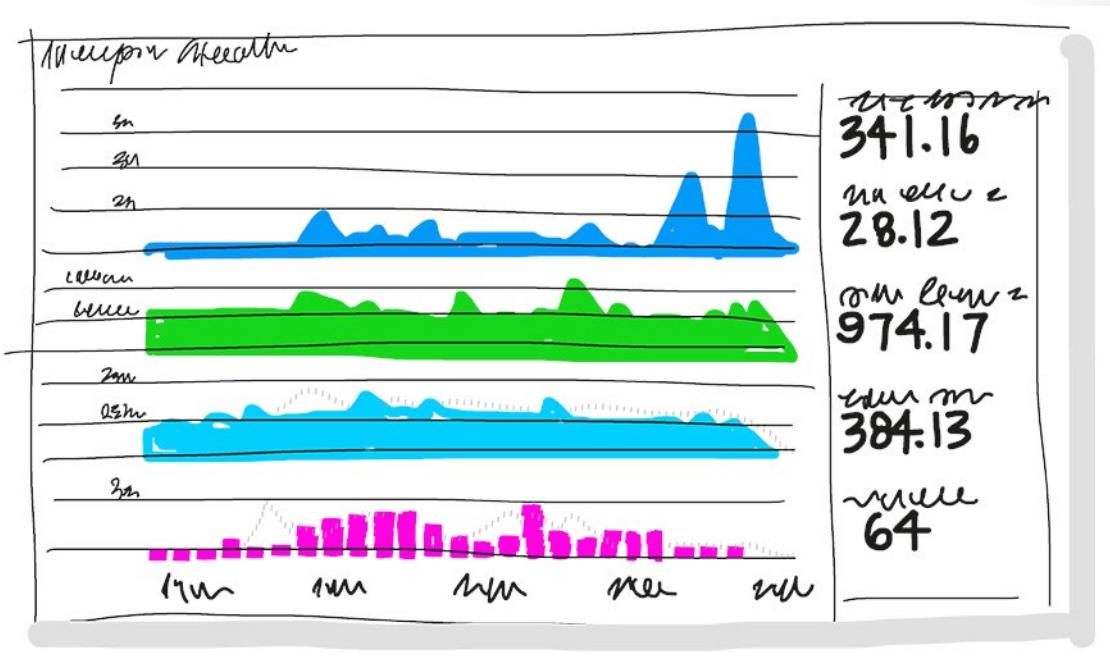
3. Version Control, Usually With Git, enables teams located anywhere in the world to communicate effectively during daily development activities as well as to integrate with software development tools for monitoring activities such as deployments.



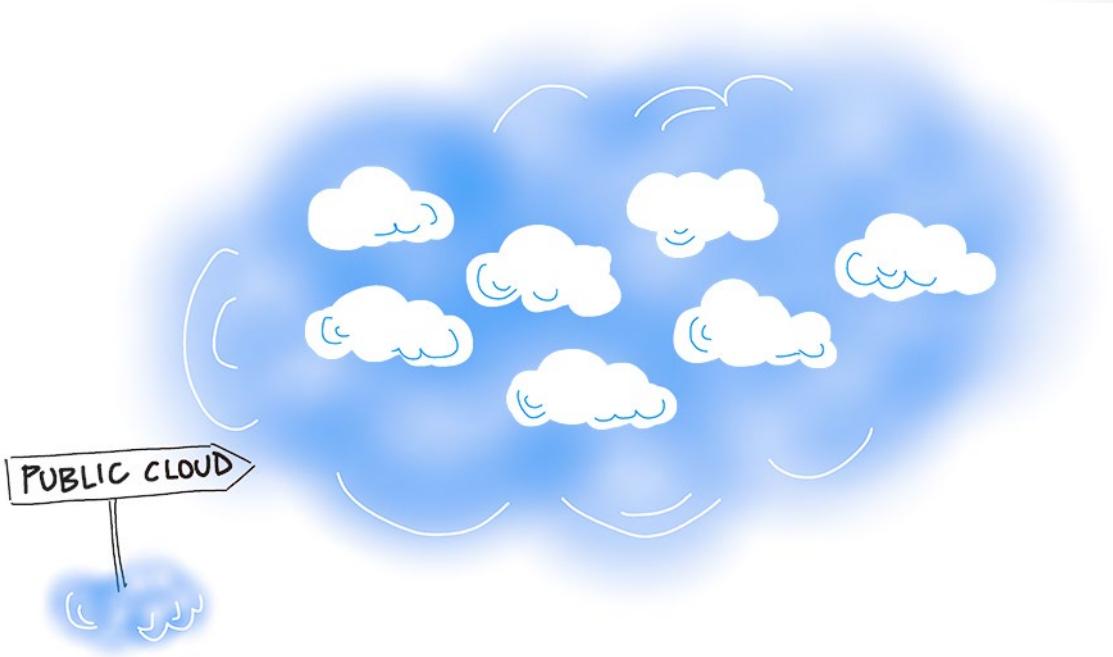
4. Agile planning and lean project management techniques are used to plan and isolate work into sprints, manage team capacity, and help teams quickly adapt to changing business needs. A DevOps Definition of Done is working software collecting telemetry against the intended business objectives.



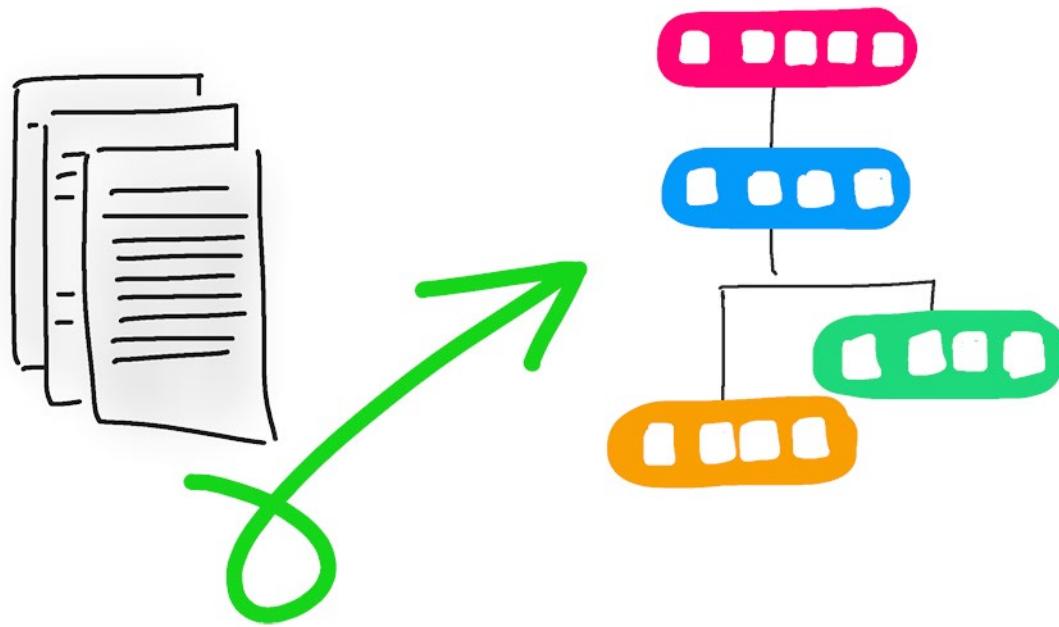
5. Monitoring and Logging of running applications including production environments for application health as well as customer usage, helps organizations form a hypothesis and quickly validate or disprove strategies. Rich data is captured and stored in various logging formats.



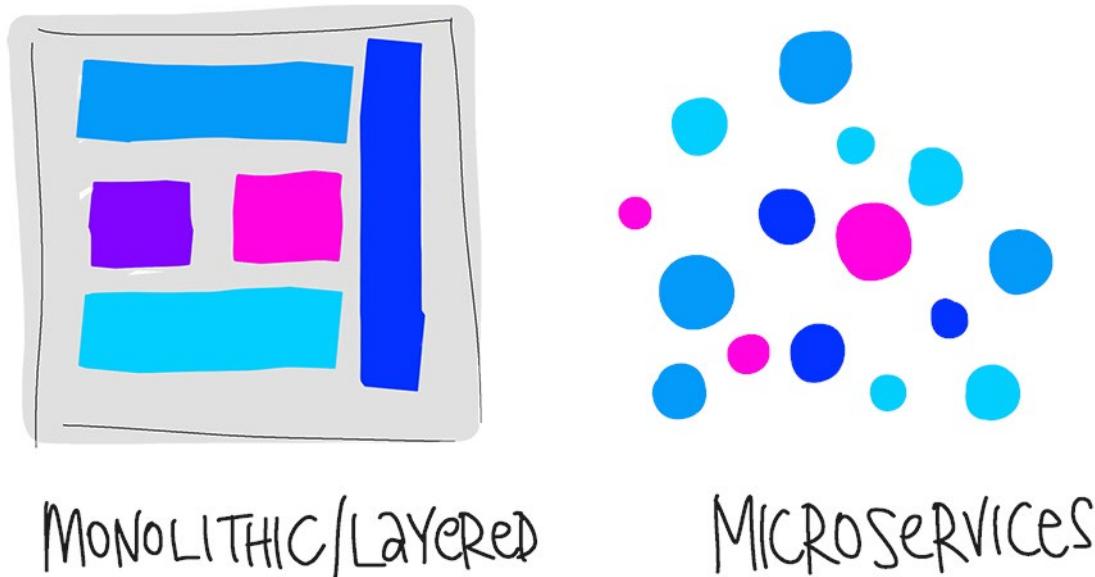
6. Public and Hybrid Clouds have made the impossible easy. The cloud has removed traditional bottlenecks and helped commoditize infrastructure. Whether you use Infrastructure as a Service (IaaS) to lift and shift your existing apps, or Platform as a Service (PaaS) to gain unprecedented productivity, the cloud gives you a datacenter without limits.



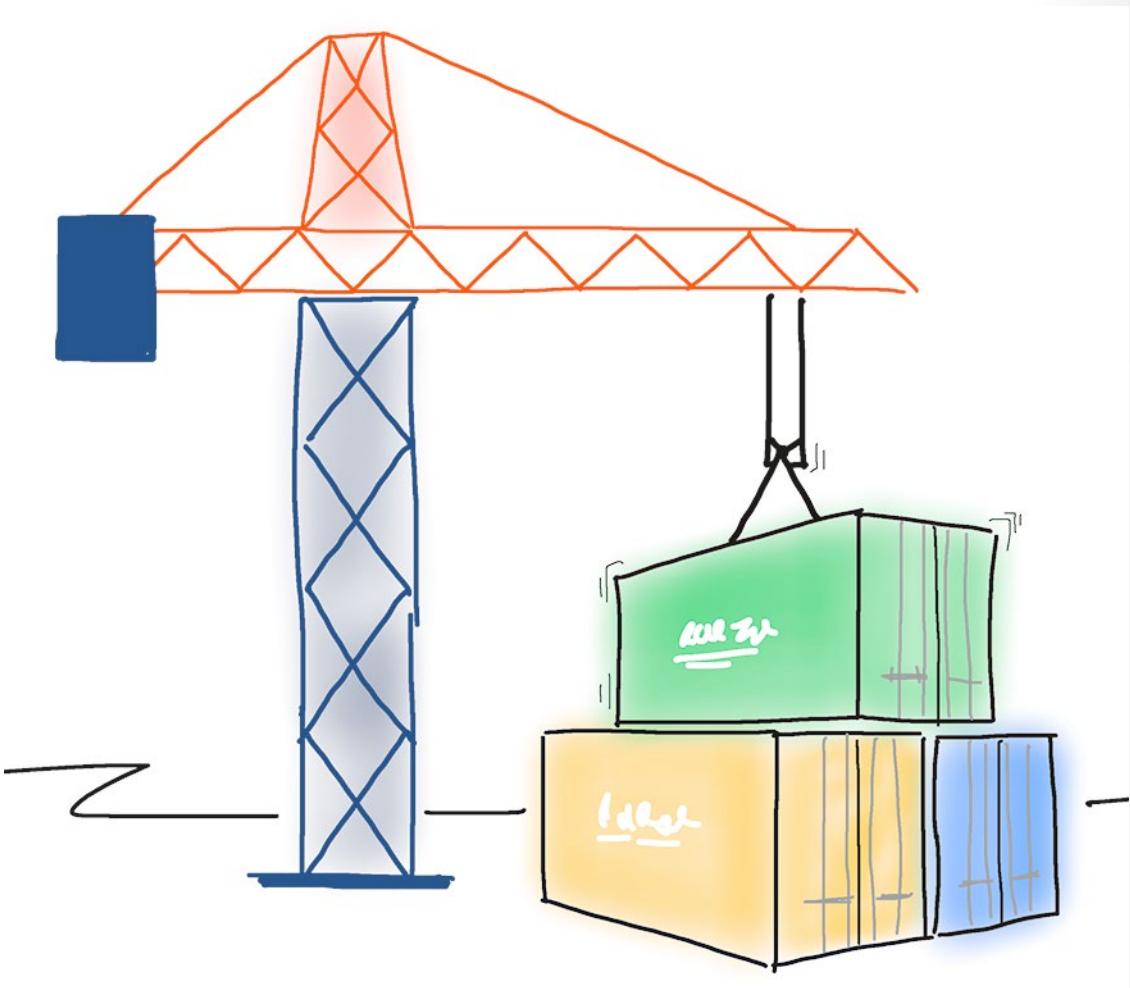
7. Infrastructure as Code (IaC) is a practice which enables the automation and validation of creation and teardown of environments to help with delivering secure and stable application hosting platforms.



8. Microservices architecture is leveraged to isolate business use cases into small reusable services that communicate via interface contracts. This architecture enables scalability and efficiency.

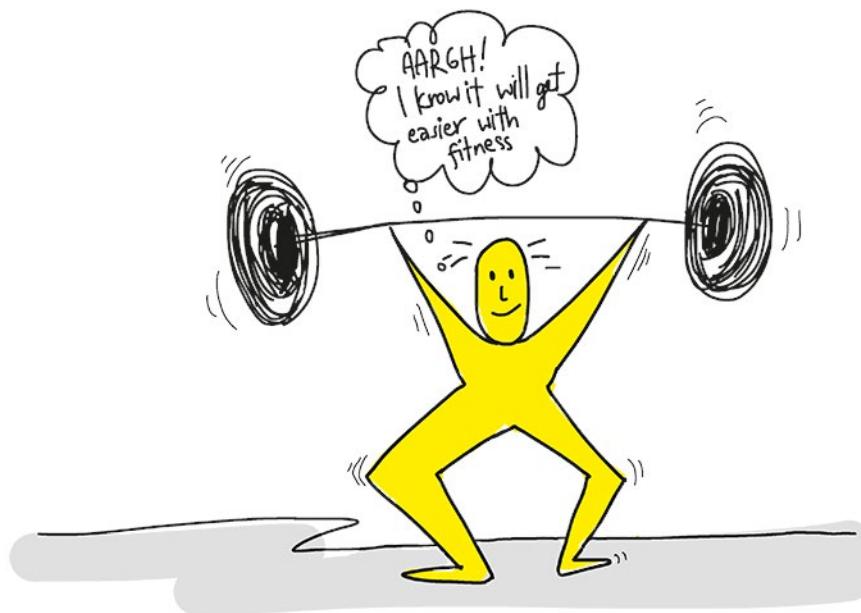


9. Containers are the next evolution in virtualization. They are much more lightweight than virtual machines, allow much faster hydration, and can be easily configured from files.



DevOps May Hurt at First

If it hurts, do it more often. Just like going to the gym, adopting new practices is likely to hurt at first. The more often you exercise the new practices, the easier they will become. And just like training at the gym, where you exercise large muscles before small muscles, adopt practices that have the greatest impact first and cross-train to develop synergy.



Separating Transformation Teams

Unless you are building an entirely new organization, one of the big challenges with any DevOps Transformation Project is that you'll be taking actions that conflict at least in some way with ongoing business states.

There are many aspects to this. The first challenge is the availability of staff. If the staff members, who were leading the transformation project, are also involved in existing day-to-day work within the organization, it will be difficult for them to focus on the transformation, at least in any meaningful way, particularly if their existing role directly impacts customer outcomes. We all know the desperate situations that involve customers will always win over a long-term project, like DevOps Transformations.

Another issue will be the way that the organization currently operates. Existing processes and procedures have been implemented to support current business outcomes. The disruption that is required for a true DevOps Transformation

will usually end up challenging those existing processes and procedures. Doing that is often very difficult. Dr. Vijay Govindarajan and Dr. Chris Trimble, in their book *Beyond the Idea: How to Execute Innovation*, have researched what's involved in allowing innovation to occur in organizations and noted that when this

is successful, it's often been in spite of the existing organizational processes. They concluded that it only works where a separate team is created to pursue the transformation. For DevOps transformations, the separate team should be made up of staff members, all of whom are focused on

and measured on the transformation outcomes, and not involved in the operational day-to-day work. The team might also include some external experts that can fill the knowledge gaps and help to advise on processes that are new to the existing staff members. Ideally the staff members who were recruited for this should already be well-regarded throughout the organization and as a group they should offer a broad knowledge base so they can think outside the box.

Defining Shared Goals

Most management courses would tell you that if you want to change something, you need to make sure that it's measurable. DevOps transformations are no different. The project needs to have a clearly-defined set of measurable outcomes.

These outcomes should include specific measurable targets like:

- Reduce the time spent on fixing bugs by 60%.
 - Reduce the time spent on unplanned work by 70%.
 - Reduce the out-of-hours work required by staff to no more than 10% of total working time.
 - Remove all direct patching of production systems.
- ✓ Note: One of the key aims of DevOps is to provide greater customer value, so outcomes should have a customer value focus.

Setting Timelines for Goals

Measurable goals also need to have timelines. While it's easy to set longer-term goals, it's also easy to put off work when you know it's not needed for a while. While overall projects should have timelines that span anywhere from a few months to a year or two in any DevOps transformation project, it's important to have a constant series of short-term goals. Every few weeks, the improvements made should be clear and measurable and ideally, also obvious to the organization and/or its customers. Clearly the timeline should not be too short. They should always be challenging yet achievable. A review should occur after each short-term goal to assist in planning the next one. There are several advantages of the shorter timelines. One key advantage is that it's easier to change plans or priorities when necessary. Another is that the reduced delay between doing work and getting feedback helps to ensure that the learnings and feedback are incorporated quickly. Finally, it's easier to keep organizational support when positive outcomes are apparent.

Project Selection

Greenfield and Brownfield projects Defined

The terms greenfield and brownfield have their origins in residential and industrial building projects. A greenfield project is one done on a green field, that is, undeveloped land. A brownfield project is one that was done on land that has been previously used for other purposes. Because of the land use that has previously occurred there could be challenges with reusing the land. Some of these would be obvious, like existing buildings, but could also be less obvious, like polluted soil.

Applied to Software or DevOps Projects

The same terms are routinely applied to software projects and commonly used to describe DevOps projects. On the surface it can seem that a greenfield DevOps project would be easier to manage and to achieve success. There was no existing code base, no existing team dynamics or politics, and possibly no existing, rigid processes. Because of this there's a common misconception that DevOps is really only for greenfield projects, and that it suits startups best. However, a large number of very successful brownfield DevOps projects have occurred. The beauty of these projects is that there's often already a large gap between the customer expectations and what is being delivered, and the teams involved may well realize that the status quo needs to change, because they've lived the challenges and the limitations associated with what they're currently doing.

Choosing Greenfield and Brownfield Projects

When starting a DevOps transformation, you might need to choose between Greenfield and Brownfield projects. There is a common misconception that DevOps suits Greenfield projects better than Brownfield projects, but this is not the case.

Greenfield Projects

A Greenfield project will always appear to be an easier starting point, because a blank slate offers the chance to implement everything the way that you want. You might also have a better chance of avoiding existing business processes that do not align with your project plans.

For example, if current IT policies do not allow the use of cloud-based infrastructure, this might be allowed for entirely new applications that are designed for that environment from scratch. As another example, you might be able to sidestep internal political issues that are well-entrenched.

Brownfield Projects

While Brownfield projects come with the baggage of existing code bases, existing teams, and often a great amount of technical debt, they can still be ideal projects for DevOps transformations.

When your teams are spending large percentages of their time just maintaining existing Brownfield applications, you have limited ability to work on new code. It's important to find a way to reduce that time, and to make software releases less risky. A DevOps transformation can provide that.

The existing team members will often have been worn down by the limitations of how they have been working in the past, and be keen to try to experiment with new ideas. These are often systems that the organizations will be currently depending upon, so it might also be easier to gain stronger management buy in for these projects because of the size of the potential benefits that could be derived. Management

might also have a stronger sense of urgency to point brownfield projects in an appropriate direction, when compared to greenfield projects that don't currently exist.

Take the First Step

Eventually the goal will be to evolve your entire organization. In looking to take the first step, many organizations start with a Greenfield Project and then move on from there.

Choosing Systems of Record vs Systems of Engagement

When selecting systems as candidates for starting a DevOps transformation, it's important to consider the types of systems that you operate.

Some researcher suggests that organizations often use Bimodal IT; a practice of managing two separate, coherent modes of IT delivery - one focused on stability and predictability, and the other on agility.

Systems of Record

Systems that are considered to be providing the truth about data elements are often called systems of record. These systems have historically evolved slowly and carefully. For example, it is crucial that a banking system accurately reflect your bank balance.

Systems of record emphasize accuracy and security.

Systems of Engagement

Many organizations have other systems that are more exploratory. These often use experimentation to solve new problems. Systems of engagement are ones that are modified regularly. Making changes quickly is prioritized over ensuring that the changes are right.

There is a perception that DevOps suit systems of engagement more than systems of record. But the lessons from high performing companies show that this just isn't the case. Sometimes, the criticality of doing things right with a system of record is an excuse for not implementing DevOps practices. Worse, given the way that applications are interconnected, an issue in a system of engagement might end up causing a problem in a system of record anyway. Both types of systems are important. While it might be easier to start with a system of engagement when first starting a DevOps Transformation, DevOps practices apply to both types of systems. The most significant outcomes often come from transforming systems of record.

Selecting Groups to Minimize Initial Resistance

Not all staff members within an organization will be receptive to the change that is required for a DevOps transformation. In discussions around continuous delivery, users are often categorized into three general buckets:

- **Canaries** who voluntarily test bleeding edge features as soon as they are available.
- **Early adopters** who voluntarily preview releases, considered more refined than the code that canary users are exposed to.
- **Users** who consume the products, after passing through canaries and early adopters.

While development and IT operations staff might generally be expected to be less conservative than users, their attitudes will also range from very conservative, to early adopters, and to those happy to work at the innovative edge.

Ideal DevOps team members

For a successful DevOps transformation, the aim is to find team members with the following characteristics:

- They already think there is a need to change.
- They have previously shown an ability to innovate.
- They are already well-respected within the organization.
- They have a broad knowledge of the organization and how it operates.
- Ideally, they already believe that DevOps practices are what is needed

Ideal target improvements

It is also important to roll out changes incrementally. There is an old saying in the industry that any successful large IT system was previously a successful small IT system. Large scale systems that are rolled out all at once, have a very poor record of success. Most fail, no matter how much support management has provided.

When starting, it is important to find an improvement goal that:

- Can be used to gain early wins.
- Is small enough to be achievable in a reasonable time-frame.
- Has benefits that are significant enough to be obvious to the organization.

This allows constant learning from rapid feedback, and the ability to recover from mistakes quickly.

✓ Note: The aim is to build a snowball effect where each new successful outcome adds to previous successful outcomes. This will maximize the buy-in from all those affected.

Identifying Project Metrics and KPIs

We spoke earlier about the importance of shared goals. As well as being agreed by team members, the goals needed to be specific, measurable, and time-bound. To ensure that these goals are measurable, it is important to establish (and agree upon) appropriate metrics and Key Performance Indicators (KPIs). While there is no specific list of metrics and KPIs that apply to all DevOps projects, the following are commonly used:

Faster Outcomes

- **Deployment Frequency.** Increasing the frequency of deployments is often a critical driver in DevOps projects.
- **Deployment Speed.** As well as increasing how often deployments happen, it's important to decrease the time that they take.
- **Deployment Size.** How many features, stories, and bug fixes are being deployed each time?
- **Lead Time.** How long does it take from starting on a work item, until it is deployed?

Efficiency

- **Server to Admin Ratio.** Are the projects reducing the number of administrators required for a given number of servers?
- **Staff Member to Customers Ratio.** Is it possible for less staff members to serve a given number of customers?
- **Application Usage.** How busy is the application?
- **Application Performance.** Is the application performance improving or dropping? (Based upon application metrics)?

Quality and Security

- **Deployment Failure Rates.** How often do deployments (and/or applications) fail?
- **Application Failure Rates.** How often do application failures occur, such as configuration failures, performance timeouts, etc?
- **Mean Time to Recover.** How quickly can you recover from a failure?
- **Bug Report Rates.** You don't want customers finding bugs in your code. Is the amount they are finding increasing or decreasing?
- **Test Pass Rates.** How well is your automated testing working?
- **Defect Escape Rate.** What percentage of defects are being found in production?
- **Availability.** What percentage of time is the application truly available for customers?
- **SLA Achievement.** Are you meeting your service level agreements (SLAs)?
- **Mean Time to Detection.** If there is a failure, how long does it take for it to be detected?

Culture

- **Employee Morale.** Are employees happy with the transformation and where the organization is heading? Are they still willing to respond to further changes? This can be very difficult to measure, but is often done by periodic, anonymous employee surveys.
- **Retention Rates.** Is the organization losing staff?
 - ✓ Note: It is important to choose metrics that focus on specific business outcomes and that achieve a return on investment and increased business value.

Team Structures

Agile Development Practices Defined

Waterfall

Traditional software development practices involve determining a problem to be solved, analyzing the requirements, building and testing the required code, and then delivering the outcome to users. This is often referred to as a waterfall approach. The waterfall Model follows a sequential order; a project development team only moves to the next phase of development or testing if the previous step is completed successfully. It's what an engineer would do when building a bridge or a building. So, it might seem appropriate for software projects as well. However, the waterfall methodology has some drawbacks. One, in particular, relates to the customer requirements. Even if a customer's requirements are defined very accurately at the start of a project, because these projects often take a long time, by delivery, the outcome may no longer match what the customer needs. There's a real challenge with the gathering of customer requirements in the first place. Even if you built exactly what the customer asked for, it'll often be different to what they need. Customers often don't know what they want until they see it or are unable to articulate what they need.

Agile

By comparison, Agile methodology emphasizes constantly adaptive planning and early delivery with continual improvement. Rather than restricting development to rigid specifications, it encourages rapid and flexible responses to change as they occur. In 2001, a group of highly regarded developers published a manifesto for Agile software development. They said that development needs to favor individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to changes over following a plan. Agile software development methods are based on releases and iterations. One release might consist of several iterations. Each iteration is similar to a very small independent project and after being estimated and prioritized, features, bug fixes and enhancements and refactoring width is assigned to a release, and then assigned again to a specific iteration within the release, generally on a priority basis. At the end of each iteration, they should be tested working code. In each iteration, the team must focus on the outcomes of the previous iteration and learn from that. An advantage of having teams focused on shorter term outcomes is that teams are also less likely to waste time over engineering features or allowing an unnecessary scope creep to occur. Agile software development helps teams keep focused on business outcomes.

Comparison of Waterfall and Agile Methodologies

Waterfall	Agile
divided into distinct phases	separates the project development lifecycle into sprints
can be quite rigid	known for flexibility
all project development phases, such as design, development, and test, are completed once	follows an iterative development approach, so each phase may appear more than once
define requirements at start of project with little change expected	requirements are expected to change and evolve
focus on completing the project	focus on meeting customer's demands

Principles of Agile Development

The **Agile Alliance**¹ says that its mission is to support people who explore and apply agile values, principles, and practices to make building software solutions more effective, humane, and sustainable.

They have published a **Manifesto for Agile Software Development**².

From that, they have distilled the **12 Principles Behind the Agile Manifesto**³.

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Creating Organization Structure for Agile Practices

For most organizations, reorganizing to be agile is very difficult. It requires a mind-shift and a culture-shift that challenges many existing policies and processes within the organization.

This isn't surprising because good governance in organizations, particularly in large organizations often ends up leading to a large number of quite rigid rules, operating structures, and methods. It also tends to avoid wide delegation of authority.

While most large organizations haven't moved to an agile structure, most are now experimenting with doing so. Their business environments are volatile and complex and they have seen the limitations of their current structures, particularly in regard to an inability to cope with change fast enough. They realize that it is common today for long-term established businesses and their industries, to be disrupted by startups.

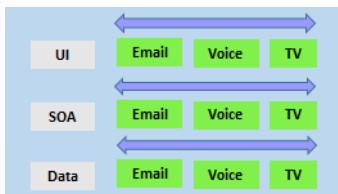
¹ <https://www.agilealliance.org/>

² <https://www.agilealliance.org/agile101/the-agile-manifesto/>

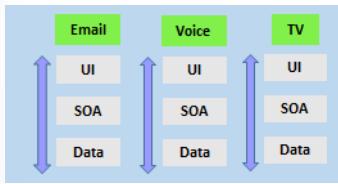
³ <https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/>

Horizontal vs vertical teams

Traditionally, horizontal team structures divide teams according to the software architecture. In this example, the teams have been divided into user interface, service-oriented architecture, and data teams:

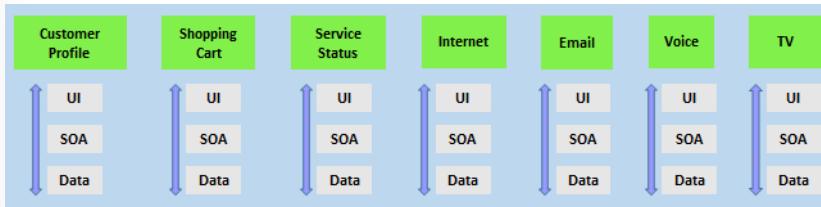


By comparison, vertical team structures span the architecture and are aligned with skill sets or disciplines:



Vertical teams have been shown to provide stronger outcomes in Agile projects. It's important that each product has a clearly-identified owner.

Another key benefit of the vertical team structure is that scaling can occur by adding teams. In this example, feature teams have been created rather than just project teams:



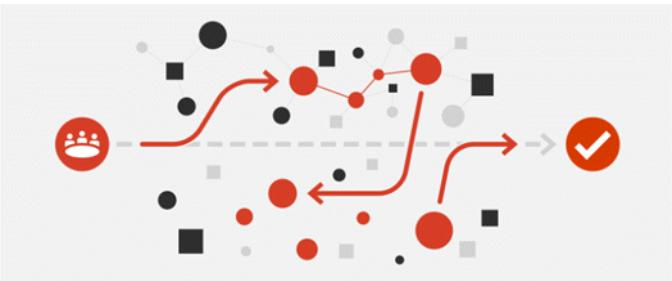
Mentoring Team Members on Agile Practices

While it's desirable to have formal Agile training for staff members, no matter how good any Agile course is, there's a world of difference between learning a concept within a few days and putting it into practice. When they first start an Agile transformation, many teams hire external coaches or mentors. Agile coaches help teams or individuals to adopt Agile methods or to improve the current methods and practices. They must be agents of change by helping people to understand how they work and encouraging them to adopt new methods. Agile coaches typically work with more than one team and try to remove any roadblocks from inside or outside the organization. This work requires a variety of skills, including coaching, mentoring, teaching, and facilitating. Agile coaches must be both trainers and consultants.

There is more than one type of Agile coach. Some coaches are technical experts who aim to show staff members how to apply specific concepts, like test-driven development and the implementation of continuous integration or deployment. These coaches might perform peer programming sessions with staff members. Other coaches are focused on Agile processes, determining requirements, and managing work activities. They might assist in how to run effective stand-up and review meetings. Some coaches may themselves act as scrum masters. They might mentor staff in how to fill these roles.

Over time, though, it's important for team members to develop an ability to mentor each other. Teams should aim to be self-organizing. Team members are often expected to learn as they work and to acquire skills from each other. To make this effective, though, the work itself needs to be done in a collaborative way, not by individuals working by themselves.

Enabling In-Team and Cross-Team Collaboration



Effective collaboration is critical for well-functioning Agile teams. Enabling this requires both cultural changes, cross-functional team collaboration, and tooling.

Cultural changes

Over recent decades, offices have often become open-spaces with few walls. Ironically, this can limit collaboration and ambient noise and distractions often also reduce productivity. Staff tend to work better when they have quiet comfortable working environments. Defined meeting times and locations lets staff choose when they want to interact with others.

Asynchronous communication should be encouraged but there should not be an expectation that all communications will be responded to urgently. Staff should be able to focus on their primary tasks without feeling like they are being left out of important decisions.

All meetings should have strict time-frames, and more importantly, have an agenda. If there is no agenda, there should be no meeting.

As it is becoming harder to find the required staff, great teams will be just as comfortable with remote or work-from-home workers as they are for those in the office. To make this successful though, collaboration via communication should become part of the organization's DNA.

Staff should be encouraged to communicate openly and frankly. Learning to deal with conflict is important for any team, as there will be disagreements at some point. Mediation skills training would be useful.

Cross-functional teams

It is clear that members of a team need to have good collaboration, it's also important to have great collaboration with wider teams, to bring people with different functional expertise together to work toward a common goal. Often, these will be people from different departments within an organization.

Faster and better innovation can occur in these cross-functional teams. People from different areas of the organization will have different views of the same problem, and they are more likely to come up with alternate solutions to problems or challenges. Existing entrenched ideas are more likely to be challenged.

Cross-functional teams can also minimize turf-wars within organizations. The more widely that a project appears to have ownership, the easier it will be for it to be widely accepted. Bringing cross-functional teams together also helps to spread knowledge across an organization.

Recognizing and rewarding collective behavior across cross-functional teams can also help to increase team cohesion.

Collaboration tooling

The following collaboration tools are commonly used by agile teams:

Teams (Microsoft)⁴. Is a group chat application from Microsoft. It provides a combined location with workplace chat, meetings, notes, and storage of file attachments. A user can be a member of many teams.

Slack⁵. Is a very commonly used tool for collaboration in Agile and DevOps teams. From a single interface, it provides a series of separate communication channels. These can be organized by project, team, or topic. Conversations are retained and are searchable. It is very easy to add both internal and external team members. Slack directly integrates with many third party tools like **GitHub**⁶ for source code and **DropBox**⁷ for document and file storage.

Skype (Microsoft)⁸. Some teams use Skype widely as it is a very commonly-deployed application. Because it is designed for general purpose communication, it is easy to enlist people from outside the team such as external stakeholders. Many will already have Skype installed. It does not currently offer more advanced team collaboration tooling.

Other common tools that include collaboration offerings include Google Hangouts, WebEx, GoToMeeting, FlowDock, Asana, ProofHub, RedBooth, Trello, DaPulse, and many others.

Selecting Tools and Processes for Agile Practices

While developing using agile methods doesn't require specific tooling, the use of tools can often enhance the outcomes achieved. It's important to realize, though, that the most important tool for agile development is the process itself. You should become familiar with the processes that you need to follow before you try to work out how to implement tools. Several categories of tools are commonly used.

Physical tools

Note that not all tools need to be digital tools. Many teams make extensive use of white boards for collaborating on ideas, index cards for recording stories, and sticky notes for moving tasks around. Even when digital tools are available, it might be more convenient to use these physical tools during stand up and other meetings.

Collaboration tools

These tools were discussed in the previous topic.

Project management tools

These tools usually include project planning and execution monitoring abilities (including how to respond to impediments), automation for stand up meetings, management and tracking of releases, and a way to

⁴ <https://products.office.com/en-us/microsoft-teams/group-chat-software>

⁵ <https://slack.com/>

⁶ <https://github.com/>

⁷ <https://dropbox.com/>

⁸ <https://www.skype.com/en/>

record and work with the outcomes of retrospectives. Many include Kanban boards and detailed sprint planning options.

Most of these tools will also provide detailed visualizations, often as a graphic dashboard that shows team progress against assigned goals and targets. Some tools also integrate directly with code repositories and CI/CD tools and add code-related metrics including quality metrics, along with direct support for code reviews.



As well as a complete CI/CD system, Azure Boards includes flexible Kanban boards, traceability through Backlogs, customizable dashboards, built-in scrum boards and integrates directly with code repositories. Code changes can be linked directly to tasks or bugs.

Apart from Azure Boards, other common tools include GitHub, Jira Agile, Trello, Active Collab, Agilo for Scrum, SpiraTeam, Icescrum, SprintGround, Gravity, Taiga, VersionOne, Agilean, Wrike, Axosoft, Assembla, PlanBox, Asana, Binfire, Proggio, VivifyScrum, and many others.

Screen recording tools

It might seem odd to add screen recording tools into this list but they are really helpful when working with remote team members, for recording bugs in action, and for building walkthroughs and tutorials that demonstrate actual or potential features.

There is a screen recorder built into Windows but other common ones include SnagIt, Camtasia, OBS, and Loom.

Migrating to Azure DevOps

What is Azure DevOps

Azure DevOps is a Software as a service (SaaS) platform from Microsoft that provides an end-to-end DevOps toolchain for developing and deploying software. It also integrates with most leading tools on the market and is a great option for orchestrating a DevOps toolchain.

What can Azure DevOps do?

Azure DevOps comprises a range of services covering the full development life-cycle.

- Azure Boards: agile planning, work item tracking, visualisation and reporting tool.
- Azure Pipelines: a language, platform and cloud agnostic CI/CD platform with support for containers or Kubernetes.
- Azure Repos: provides cloud-hosted private git repos.
- Azure Artifacts: provides integrated package management with support for Maven, npm, Python and NuGet package feeds from public or private sources.
- Azure Test Plans: provides an integrated planned and exploratory testing solution.

Azure DevOps can also be used to orchestrate third-party tools.

What if we are not a Microsoft / dotnet organization?

Azure Devops is not focussed at organizations that are end-to-end Microsoft or Windows.

Azure DevOps provides a platform that is:

- Flexible: you don't have to go 'all in' on Azure DevOps. It is possible to adopt each of the services independently and integrate them with your existing tool chain, most popular tools are supported.
- Cross Platform: designed to work with any platform (Linux, MacOS and Windows) or language (including Node.js, Python, Java, PHP, Ruby, C/C++, .Net, Android and iOS apps) Azure DevOps is not just aimed at organisations building and shipping on the Microsoft technology stack.
- Cloud Agnostic: continuous delivery is supported to AWS and GCP as well as to Azure.

Designing an Authorization and Access Strategy

Azure DevOps Services uses enterprise-grade authentication. You can use either a Microsoft account, GitHub account or Azure Active Directory (AAD) to protect and secure your data. Many client applications, such as Visual Studio or Azure DevOps, natively support this indication by other Microsoft Accounts or AAD. Eclipse can also support this if you install a Team Explorer Everywhere plug-in. When you need a non-Microsoft tool and like GIT, NuGet or Xcode to integrate directly with Azure DevOps Services and the tools don't directly support Microsoft account or AAD account for authentication, you can still use them by setting up personal access tokens. These tokens can be set up using GIT Credential managers or you can create them manually. Personal access tokens are also useful when you need to establish access in command line tools, or in tools and tasks in build pipelines and when calling REST-based APIs because you don't have a UI popping out to perform the authentication. When access is no longer required you can then just revoke the personal access token. Azure DevOps is pre-configured with default security groups. Default permissions are assigned to the default security groups. But you can also configure access at the organization level, the collection level, and at the project or object level. In the organization

settings in Azure DevOps, you can configure app access policies. Based on your security policies, you might allow alternate authentication methods, allow third party applications to access via OAuth, or even allow anonymous access to some projects. For even tighter control, you can set conditional access to Azure DevOps. This offers simple ways to help secure resources when using Azure Active Directory for authentication. Conditional access policies such as Multifactor Authentication can help to minimize the risk of compromised credentials. As part of a conditional access policy you might require security group membership, a location or network identity, a specific operating system, a managed device, or other criteria.

Migrating or Integrating Existing Work Management Tools

Azure DevOps can be integrated with a variety of existing work management tools. As an example, in the Visual Studio Marketplace, Microsoft offers **Trello integration tooling**⁹.

Migrating from other work management tools to Azure DevOps takes considerable planning. Most work management tools are highly configurable by the end user. This means that there might not be a tool available that will perform the migration without further configuration.

Jira

Jira is a commonly-used work management tool.

In the Visual Studio Marketplace, **Solidify**¹⁰ offers a tool for Jira to Azure DevOps migration. It does the migration in two phases. Jira issues are exported to files and then the files are imported to Azure DevOps.

If you decide to try to write the migration code yourself, the following blog post provides sample code that might help you to get started:

[Migrate your project from Jira to Azure DevOps](#)¹¹

Other applications

Third party organizations do offer commercial tooling to assist with migrating other work management tools like Aha, BugZilla, ClearQuest, and others to Azure DevOps.

Migrating or Integrating Existing Test Management Tools

Azure Test Plans are used to track manual testing for sprints and milestones. This allows you to track when that testing is complete.

Azure DevOps also has a Test & Feedback extension available in the Visual Studio Marketplace. The extension is used to help teams perform exploratory testing and provide feedback. All team members (developers, product owners, managers, UX or UI engineers, marketing teams, early adopters), and other stakeholders can use the extension to submit bugs or provide feedback.

Apache JMeter¹² is open source software written in Java and designed to load test functional behavior and measure performance.

⁹ <https://marketplace.visualstudio.com/items?itemName=ms-vsts.services-trello>

¹⁰ <https://marketplace.visualstudio.com/items?itemName=solidify-labs.jira-devops-migration>

¹¹ <http://www.azurefieldnotes.com/2018/10/01/migrate-your-project-from-jira-to-azure-devops/>

¹² <https://docs.microsoft.com/en-us/azure/devops/test/load-test/get-started-jmeter-test?view=vsts>

Pester¹³ is a tool that can be used to automate the testing of PowerShell code.

SoapUI¹⁴ is another testing framework for SOAP and REST testing.

If you are using Microsoft Test Manager you should plan to migrate to using Azure Test Plans instead.

For more information, see **Marketplace search for test management**¹⁵.

Designing a License Management Strategy

For the latest, most up-to-date pricing information, visit **Azure DevOps Pricing**¹⁶.

¹³ <https://marketplace.visualstudio.com/items?itemName=richardfennellBM.BM-VSTS-PesterRunner-Task>

¹⁴ <https://marketplace.visualstudio.com/items?itemName=AjeetChouksey.soapui>

¹⁵ <https://marketplace.visualstudio.com/search?term=test%20management&target=AzureDevOps&category>All%20categories&sortBy=Relevance>

¹⁶ <https://azure.microsoft.com/en-us/pricing/details/devops/azure-devops-services/>

Lab

Agile Planning and Portfolio Management with Azure Boards



In this lab, **Agile Planning and Portfolio Management with Azure Boards**¹⁷, you will learn about the agile planning and portfolio management tools and processes provided by Azure Boards and how they can help you quickly plan, manage, and track work across your entire team. You will explore the product backlog, sprint backlog, and task boards which can be used to track the flow of work during the course of an iteration. We will also take a look at how the tools have been enhanced in this release to scale for larger teams and organizations. Tasks include:

1. Working with teams, areas, and iterations.
2. Working with work items.
3. Managing sprints and capacity.
4. Customizing Kanban Boards.
5. Defining dashboards.
6. Customizing team processes.

✓ Note: You must have already completed the Lab Environment Setup in the Welcome section.

¹⁷ <https://www.azuredevopslabs.com/labs/azuredevops/agile/>

Module Review and Takeaways

Module Review Questions

Multiple choice

Which of the following would a system that manages inventory in a warehouse be considered?

- System of Record
- System of Engagement

Multiple choice

An Agile tool that is used to manage and visualize work by showing tasks moving from left to right across columns representing stages. What is this tool commonly called?

- Backlog
- Kanban Board

Multiple choice

In which Of the following would you find large amounts of technical debt?

- Greenfield project
- Brownfield project

Suggested Answer

As a project metric, what is Lead Time measuring?

Suggested Answer

What is a cross-functional team?

Answers

Multiple choice

Which of the following would a system that manages inventory in a warehouse be considered?

- System of Record
- System of Engagement

Explanation

Systems that are providing the truth about data elements are often called Systems of Record.

Multiple choice

An Agile tool that is used to manage and visualize work by showing tasks moving from left to right across columns representing stages. What is this tool commonly called?

- Backlog
- Kanban Board

Explanation

A Kanban Board lets you visualize the flow of work and constrain the amount of work in progress. Your Kanban board turns your backlog into an interactive signboard, providing a visual flow of work.

Multiple choice

In which Of the following would you find large amounts of technical debt?

- Greenfield project
- Brownfield project

Explanation

A Brownfield Project comes with the baggage of existing code bases, existing teams, and often a great amount of technical debt, they can still be ideal projects for DevOps transformations.

As a project metric, what is Lead Time measuring?

Lead Time Measuring is how long it takes from starting on a work item, until it is deployed.

What is a cross-functional team?

A team that brings people with different functional expertise, and often from different departments, together to work toward a common goal.

Module 2 Getting Started with Source Control

Module Overview

Module Overview

Source control is fundamental to DevOps. In this modern day you'll hardly find any resistance to the use of source control; however, there is some level of ambiguity around the differences in the two different types of source control systems and which type is better suited where.

Learning Objectives

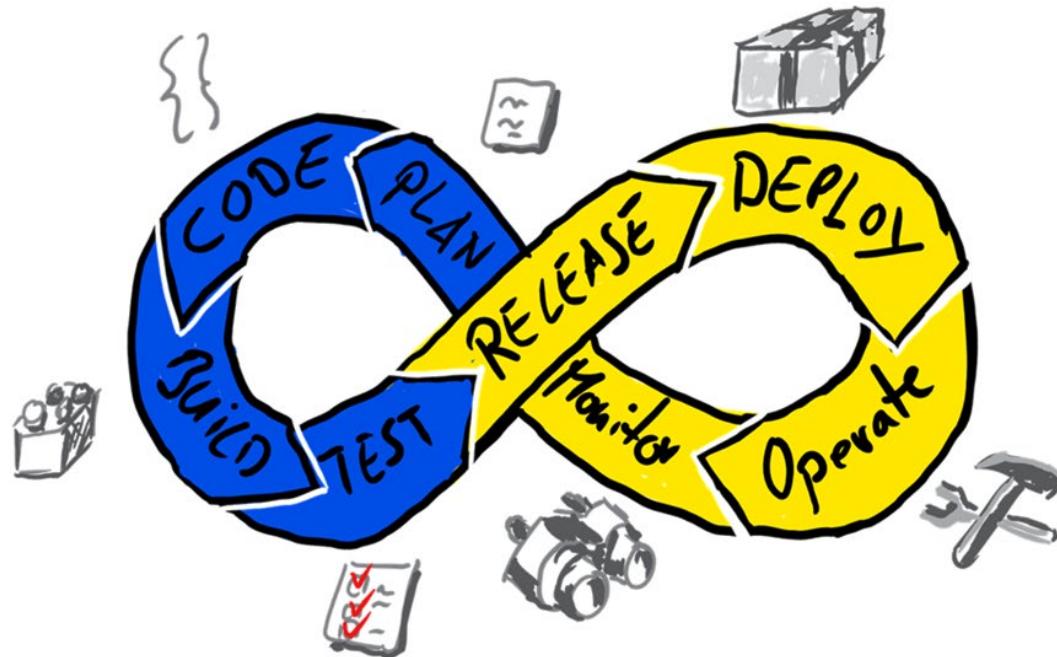
After completing this modules, students will be able to:

- Describe the benefits of using Source Control
- Describe Azure Repos and GitHub
- Migrate from TFVC to Git

What is Source Control

Introduction to Source Control

DevOps has been an emerging trend in the software development world for the past several years. While the term may be relatively new, it is really a convergence of a number of practices that have been evolving for decades. DevOps is a revolutionary way to release software quickly and efficiently while maintaining a high level of security. While it has proven to be the wave of the future, many still have yet to hop aboard the DevOps train.



Let's be honest, version control isn't exactly a topic you would bring up at a dinner party to spice up the conversation. Managing file versions and meticulously reviewing code for errors can be a dull subject. In fact, it rarely makes the headlines — even in software development news when there are far more exciting trends to cover like AI or the latest Surface device hitting the market.

But much like no one really talks about putting on clean socks in the morning, setting an alarm at night or looking both ways before crossing the street, version control is an essential every-day practice. Versioning is a common part of the developer's routine and if leveraged correctly, can save organizations enormous cost and resources. Though version control is today a common sense aspect of programming, it is important from time to time to look at why we do what we do and how versioning impacts the entire value stream at an organization.

Foundational Practices of DevOps

In an effort to eliminate the enigma that comes with any new business practice... the **2019 State of DevOps report**¹, highlights version control in almost all stages of DevOps evolution.

¹ <https://puppet.com/resources/report/state-of-devops-report>

Foundational practices and the 5 stages of DevOps evolution

	Defining practices* and associated practices	Practices that contribute to success
Stage 0	<ul style="list-style-type: none"> Monitoring and alerting are configurable by the team operating the service. Deployment patterns for building applications or services are reused. Testing patterns for building applications or services are reused. Teams contribute improvements to tooling provided by other teams. Configurations are managed by a configuration management tool. 	
Stage 1	<ul style="list-style-type: none"> Application development teams use version control. Teams deploy on a standard set of operating systems. 	<ul style="list-style-type: none"> Build on a standard set of technology. Put application configurations in version control. Test infrastructure changes before deploying to production. Source code is available to other teams.
Stage 2	<ul style="list-style-type: none"> Build on a standard set of technology. Teams deploy on a single standard operating system. 	<ul style="list-style-type: none"> Deployment patterns for building applications and services are reused. Rearchitect applications based on business needs. Put system configurations in version control.
Stage 3	<ul style="list-style-type: none"> Individuals can do work without manual approval from outside the team. Deployment patterns for building applications and services are reused. Infrastructure changes are tested before deploying to production. 	<ul style="list-style-type: none"> Individuals can make changes without significant wait times. Service changes can be made during business hours. Post-incident reviews occur and results are shared. Teams build on a standard set of technologies. Teams use continuous integration. Infrastructure teams use version control.
Stage 4	<ul style="list-style-type: none"> System configurations are automated. Provisioning is automated. Application configurations are in version control. Infrastructure teams use version control. 	<ul style="list-style-type: none"> Security policy configurations are automated. Resources made available via self-service.
Stage 5	<ul style="list-style-type: none"> Incident responses are automated. Resources available via self-service. Rearchitect applications based on business needs. Security teams are involved in technology design and deployment. 	<ul style="list-style-type: none"> Security policy configurations are automated. Application developers deploy testing environments on their own. Success metrics for projects are visible. Provisioning is automated.

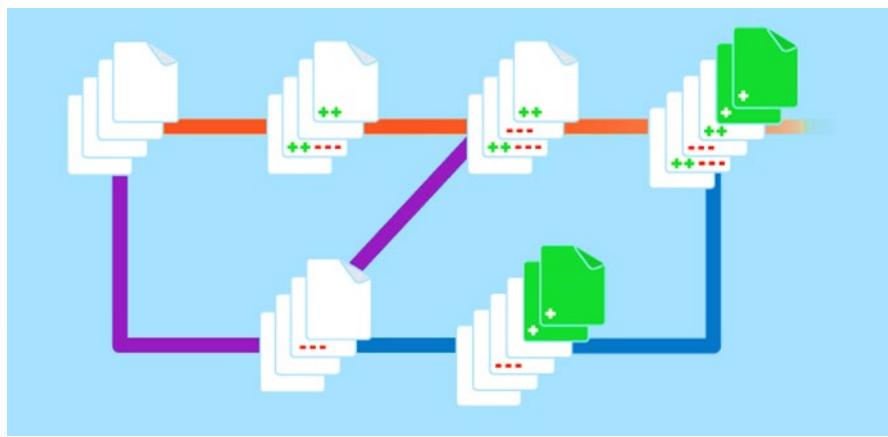
* The practices that define each stage are highlighted in bold font.

Also, it's helpful for non-developers at an organization to understand the fundamentals of a discipline that is so deeply rooted in the daily life of a software engineer — especially if those individuals are making decisions about which version control tools and platforms to use.

Version control is important for all software development projects and is particularly vital at large businesses and enterprises. Enterprises have many stakeholders, distributed teams, strict processes and workflows, silo'ed organizations and hierarchical organization. All of those characteristics represent coordination and integration challenges when it comes to merging and deploying code. Even more so in companies within highly-regulated industries such as in banking and healthcare, with many rules and regulations, need a practical way to ensure that all standards are being met appropriately and risk is mitigated.

What is Source Control

Source control (or version control) is the practice of tracking and managing changes to code. Source control management (SCM) systems provide a running history of code development and help to resolve conflicts when merging contributions from multiple sources. For most software teams, the source code is a repository of the invaluable knowledge and understanding about the problem domain that the developers have collected and refined through careful effort. Version control protects source code from both catastrophe and the casual degradation of human error and unintended consequences.



Without version control, you're tempted to keep multiple copies of code on your computer. This is dangerous—it's easy to change or delete a file in the wrong copy of code, potentially losing work. Version control systems solve this problem by managing all versions of your code but presenting you with a single version at a time. Before we dive into the details of source control, let's clarify some common goals for software teams...

Common Software Development Values

- **Reusability** – why do the same thing twice? Re-use of code is a common practice and makes building on existing assets simpler.
- **Traceability** – Audits are not just for fun, in many industries this is a legal matter. All activity must be traced and managers must be able to produce reports when needed. Traceability also makes debugging and identifying root cause easier. Additionally, this will help with feature re-use as developers can link requirements to implementation.
- **Manageability** – Can team leaders define and enforce workflows, review rules, create quality gates and enforce QA throughout the lifecycle?
- **Efficiency** – are we using the right resources for the job and are we minimizing time and efforts? This one is pretty self-explanatory.
- **Collaboration** – When teams work together quality tends to improve. We catch one another's mistakes and can build on each other's strengths.
- **Learning** – Organizations benefit when they invest in employees learning and growing. This is not only important for on-boarding new team members, but for the lifelong learning of seasoned members and the opportunity for workers to contribute not just to the bottom line but to the industry as a whole.

Tools and processes alone are not enough to accomplish the above and hence the adoption of Agile, Continuous Integration and DevOps. Believe it or not, all of these rely on a solid version control practice.

Version control is about keeping track of every change to software assets — tracking and managing the who, what and when. Version control is a first step needed to assure quality at the source, ensuring flow and pull value and focusing on process. All of these create value not just for the software teams, but ultimately for the customer.

Version control is a solution for managing and saving changes made to any manually created assets. It allows you to go back in time and easily roll back to previously working versions if changes are made to source code. Version control tools allow you to see who made changes, when and what exactly was

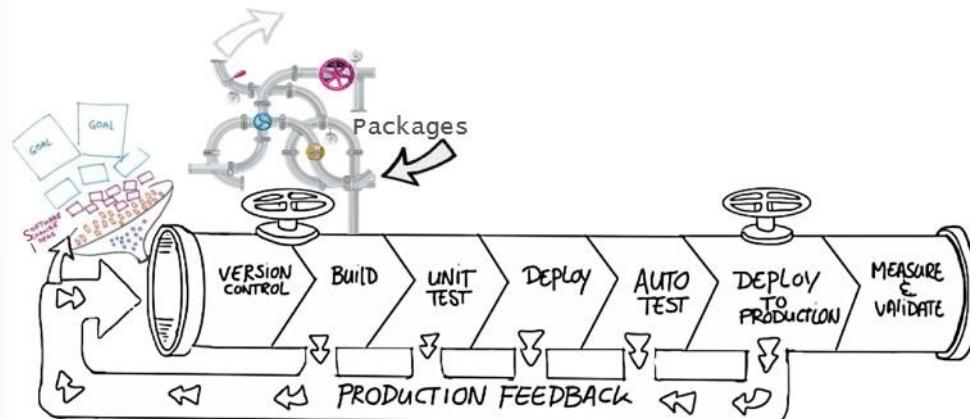
changed. Version control also makes experimenting easy and most importantly makes collaboration possible. Without version control, collaborating over source code would be a painful operation.

There are a number of perspectives on version control. For developers though, this is a daily enabler for work and collaboration to happen. It's part of the daily job, one of the most-used tools. For management, the key value of version control is in IP security, risk management and time-to-market speed through Continuous Delivery where version control is a fundamental enabler.

Benefits of Source Control

Benefits of Source control

"Code doesn't exist unless it's committed into source control... Source control is the fundamental enabler of continuous delivery."



Whether you are writing code professionally or personally, you should always version your code using a source control management system. Some of the advantages of using source control are,

- **Create workflows.** Version control workflows prevent the chaos of everyone using their own development process with different and incompatible tools. Version control systems provide process enforcement and permissions, so everyone stays on the same page.
- **Work with versions.** Every version has a description in the form of a comment. These descriptions help you follow changes in your code by version instead of by individual file changes. Code stored in versions can be viewed and restored from version control at any time as needed. This makes it easy to base new work off any version of code.
- **Collaboration.** Version control synchronizes versions and makes sure that your changes doesn't conflict with other changes from your team. Your team relies on version control to help resolve and prevent conflicts, even when people make changes at the same time.
- **Maintains history of changes.** Version control keeps a history of changes as your team saves new versions of your code. This history can be reviewed to find out who, why, and when changes were made. History gives you the confidence to experiment since you can roll back to a previous good version at any time. History lets you base work from any version of code, such as to fix a bug in a previous release.
- **Automate tasks.** Version control automation features save your team time and generate consistent results. Automate testing, code analysis and deployment when new versions are saved to version control.

Best Practices for Source Control

- **Make small changes.** In other words, commit early and commit often. Of course, be careful not to commit any unfinished work that could break the build.

- **Don't commit personal files.** These could include application settings or SSH keys. Often these are committed accidentally but cause problems later down the line when other team members are working on the same code.
- **Update often and right before pushing to avoid merge conflicts.**
- **Verify your code change before pushing it to a repository;** ensure it compiles and tests are passing.
- **Pay close attention to commit messages as these will tell you why a change was made.** Consider commit messages as a mini form of documentation for the change.
- **Link code changes to work items.** This will concretely link what was created to why it was created or changed by providing traceability across requirements and code changes.
- **No matter your background or preferences, be a team player and follow agreed conventions and workflows.** Consistency is important and helps ensure quality making it easier for team members to pick up where you left off, to review your code, to debug, etc.

Using version control of some kind is necessary for any organization and following the guidelines above can help developers avoid needless time spent fixing errors and mistakes. These practices also help organizations reap greater benefits from having a good version control system. In my next post, I'll provide some tips and best practices for organizations regarding version control. These tips will outline some ways that companies can ensure the workflow is optimal for ensuring quality and efficiency.

Types of Source Control Systems

Centralized Version Control



Strengths	Best used for
Easily scales for very large codebases	Large integrated codebases
Granular permission control	Audit & Access control down to file level
Permits monitoring of usage	Hard to merge file types
Allows exclusive file locking	

Centralized version control systems are based on the idea that there is a single “central” copy of your project somewhere (probably on a server), and programmers will check in (or commit) their changes to this central copy. “Committing” a change simply means recording the change in the central system. Other programmers can then see this change. They can also pull down the change, and the version control tool will automatically update the contents of any files that were changed. Most modern version control systems deal with “changesets,” which simply are a group of changes (possibly to many files) that should be treated as a cohesive whole. For example: a change to a C header file and the corresponding .c file should always be kept together. Programmers no longer have to keep many copies of files on their hard drives manually, because the version control tool can talk to the central copy and retrieve any version they need on the fly.

Some of the most common centralized version control systems you may have heard of or used are TFVC, CVS, Subversion (or SVN) and Perforce.

A Typical Centralized Version Control Workflow

When you’re working with a centralized version control system, your workflow for adding a new feature or fixing a bug in your project will usually look something like this:

- Get the latest changes other people have made from the central server.
- Make your changes, and make sure they work properly.
- Check in your changes to the central server, so other programmers can see them.

Distributed Version Control

Strengths	Best used for
Cross Platform support	Small & Modular codebases
An open source friendly code review model via pull requests	Evolving through open source
Complete offline support	Highly distributed teams
Portable history	Teams working across platforms
An enthusiastic growing user base	Green field codebases

In the past five years or so a new breed of tools has appeared: so-called “distributed” version control systems (DVCS for short). The three most popular of these are Mercurial, Git and Bazaar.

These systems do not necessarily rely on a central server to store all the versions of a project's files. Instead, every developer "clones" a copy of a repository and has the full history of the project on their own hard drive. This copy (or "clone") has all of the metadata of the original.

This method may sound wasteful, but in practice, it's not a problem. Most programming projects consist mostly of plain text files (and maybe a few images), and disk space is so cheap that storing many copies of a file doesn't create a noticeable dent in a hard drive's free space. Modern systems also compress the files to use even less space.

The act of getting new changes from a repository is usually called "pulling," and the act of moving your own changes to a repository is called "pushing". In both cases, you move changesets (changes to files groups as coherent wholes), not single-file diffs.

One common misconception about distributed version control systems is that there cannot be a central project repository. This is simply not true – there is nothing stopping you from saying "this copy of the project is the authoritative one." This means that instead of a central repository being required by the tools you use, it is now optional and purely a social issue.

Advantages Over Centralized Version Control

The act of cloning an entire repository gives distributed version control tools several advantages over centralized systems:

- Performing actions other than pushing and pulling changesets is extremely fast because the tool only needs to access the hard drive, not a remote server.
- Committing new changesets can be done locally without anyone else seeing them. Once you have a group of changesets ready, you can push all of them at once.
- Everything but pushing and pulling can be done without an internet connection. So you can work on a plane, and you won't be forced to commit several bugfixes as one big changeset.
- Since each programmer has a full copy of the project repository, they can share changes with one or two other people at a time if they want to get some feedback before showing the changes to everyone.

Disadvantages Compared to Centralized Version Control

There are almost no disadvantages to using a distributed version control system over a centralized one. Distributed systems do not prevent you from having a single "central" repository, they just provide more options on top of that.

There are only two major inherent disadvantages to using a distributed system:

- If your project contains many large, binary files that cannot be easily compressed, the space needed to store all versions of these files can accumulate quickly.
- If your project has a very long history (50,000 changesets or more), downloading the entire history can take an impractical amount of time and disk space.

Git and TFVC

Git (distributed)

Git is a distributed version control system. Each developer has a copy of the source repository on their dev machine. Developers can commit each set of changes on their dev machine and perform version con-

trol operations such as history and compare without a network connection. Branches are lightweight. When you need to switch contexts, you can create a private local branch. You can quickly switch from one branch to another to pivot among different variations of your codebase. Later, you can merge, publish, or dispose of the branch.

TFVC (centralized)

Team Foundation Version Control (TFVC) is a centralized version control system. Typically, team members have only one version of each file on their dev machines. Historical data is maintained only on the server. Branches are path-based and created on the server.

TFVC has two workflow models:

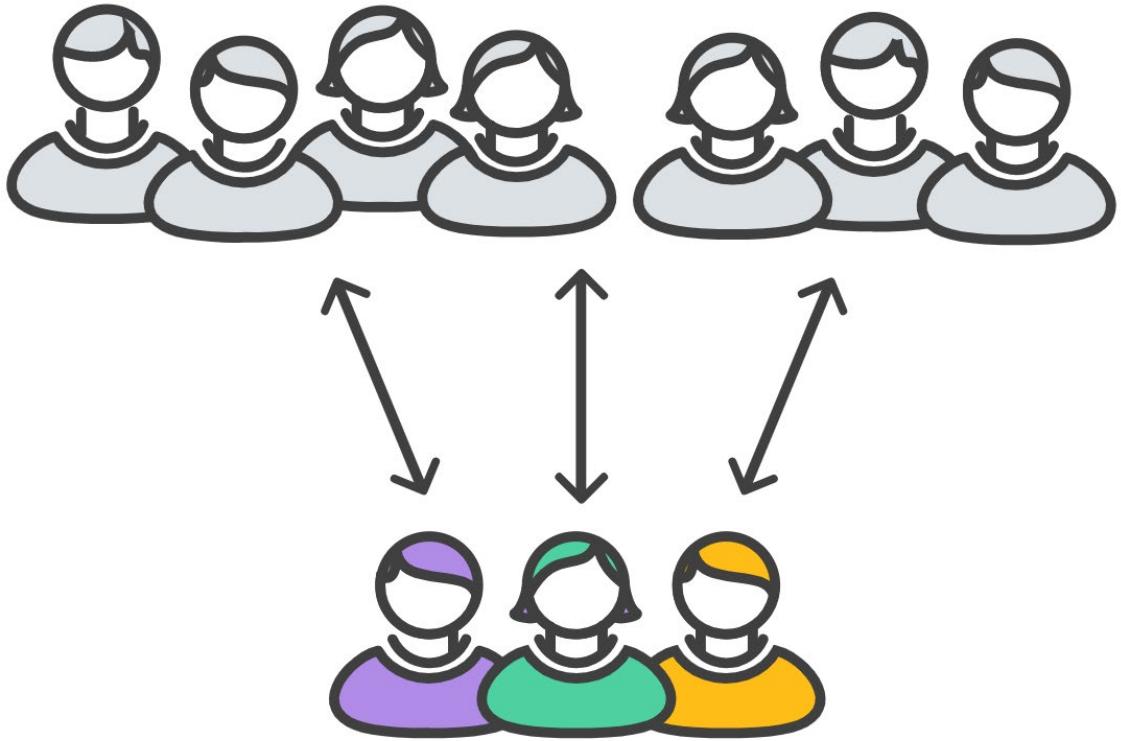
- **Server workspaces** - Before making changes, team members publicly check out files. Most operations require developers to be connected to the server. This system facilitates locking workflows. Other systems that work this way include Visual Source Safe, Perforce, and CVS. With server workspaces, you can scale up to very large codebases with millions of files per branch and large binary files.
- **Local workspaces** - Each team member takes a copy of the latest version of the codebase with them and works offline as needed. Developers check in their changes and resolve conflicts, as necessary. Another system that works this way is Subversion.

Why Git

Switching from a centralized version control system to Git changes the way your development team creates software. And, if you're a company that relies on its software for mission-critical applications, altering your development workflow impacts your entire business. Developers would gain the following benefits by moving to Git.

Community

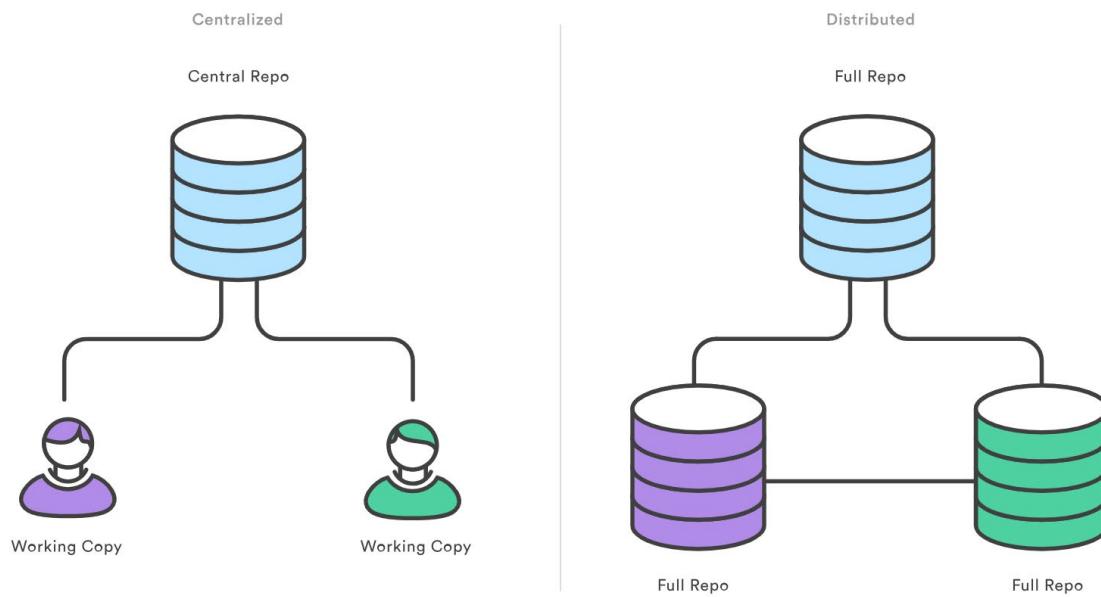
In many circles, Git has come to be the expected version control system for new projects. If your team is using Git, odds are you won't have to train new hires on your workflow, because they'll already be familiar with distributed development.



In addition, Git is very popular among open source projects. This means it's easy to leverage 3rd-party libraries and encourage others to fork your own open source code.

Distributed Development

In TFVC, each developer gets a working copy that points back to a single central repository. Git, however, is a distributed version control system. Instead of a working copy, each developer gets their own local repository, complete with a full history of commits.



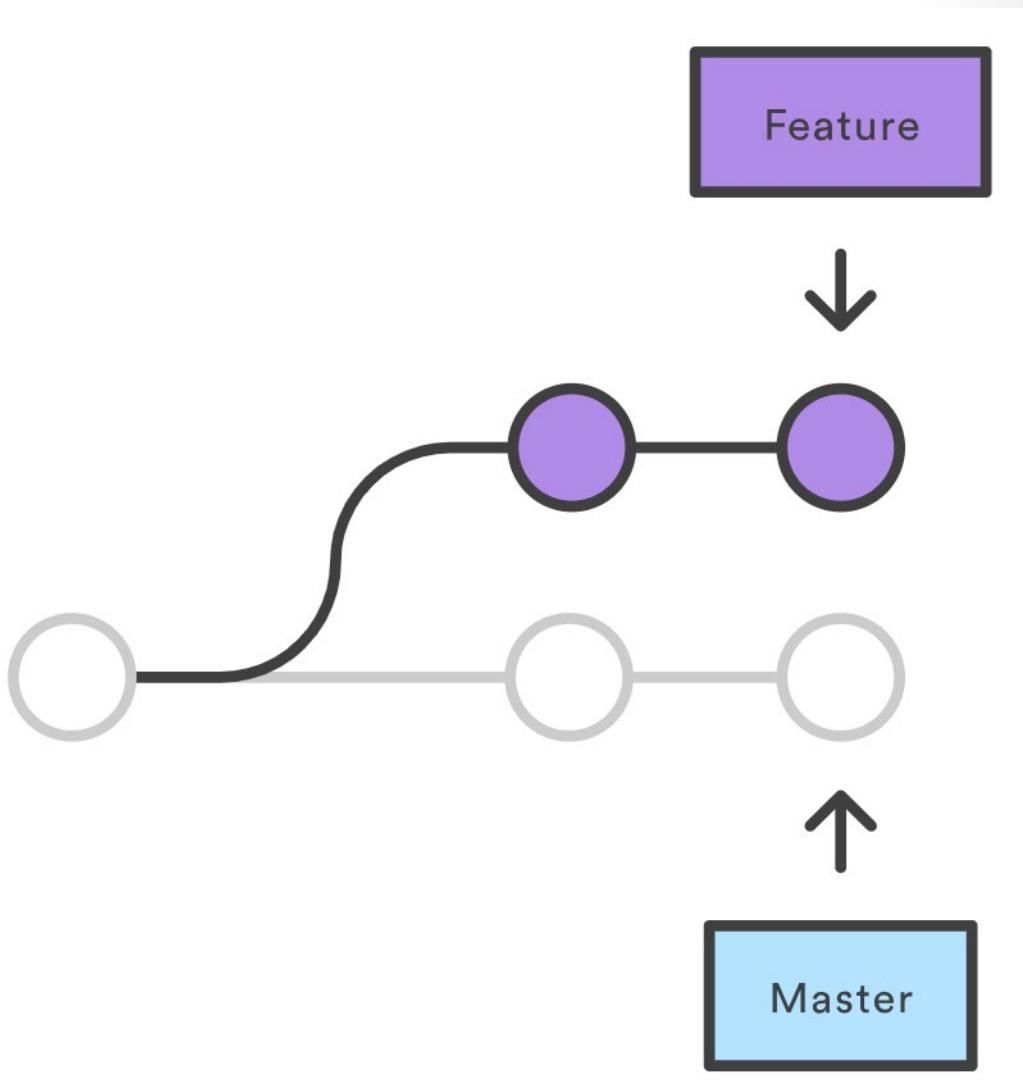
Having a full local history makes Git fast, since you don't need a network connection to create commits, inspect previous versions of a file, or perform diffs between commits.

Distributed development also makes it easier to scale your engineering team. If someone breaks the production branch in SVN, other developers can't check in their changes until it's fixed. With Git, this kind of blocking doesn't exist. Everybody can continue going about their business in their own local repositories.

And, similar to feature branches, distributed development creates a more reliable environment. Even if a developer obliterates their own repository, they can simply clone someone else's and start afresh.

Trunk-based Development

One of the biggest advantages of Git is its branching capabilities. Unlike centralized version control systems, Git branches are cheap and easy to merge.

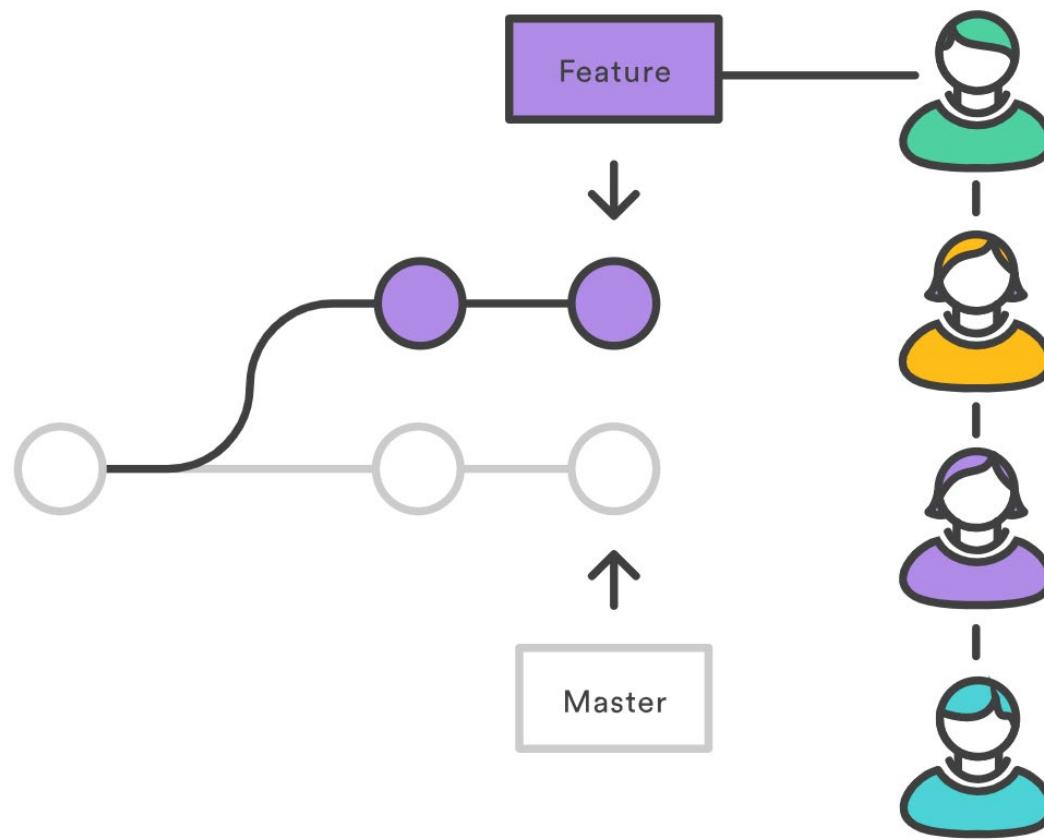


Trunk-based development provides an isolated environment for every change to your codebase. When a developer wants to start working on something—no matter how big or small—they create a new branch. This ensures that the master branch always contains production-quality code.

Using trunk-based development is not only more reliable than directly editing production code, but it also provides organizational benefits. They let you represent development work at the same granularity as your agile backlog. For example, you might implement a policy where each work item is addressed in its own feature branch.

Pull Requests

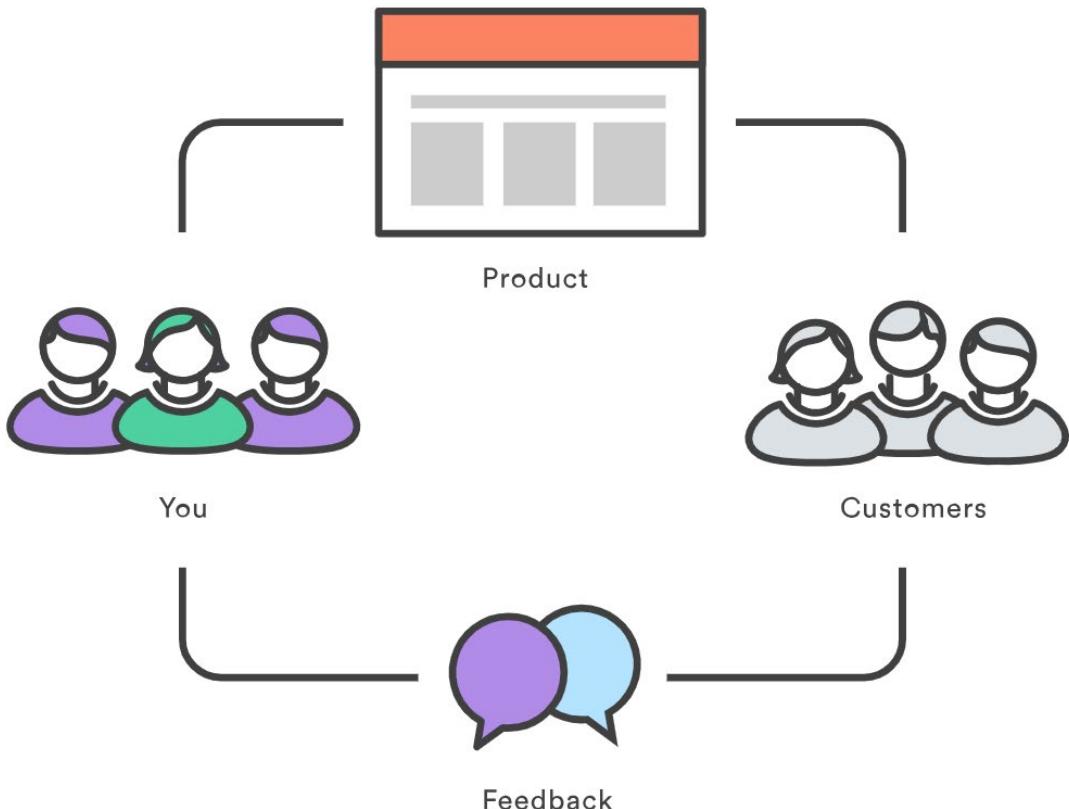
Many source code management tools such as Azure Repos enhance core Git functionality with pull requests. A pull request is a way to ask another developer to merge one of your branches into their repository. This not only makes it easier for project leads to keep track of changes, but also lets developers initiate discussions around their work before integrating it with the rest of the codebase.



Since they're essentially a comment thread attached to a feature branch, pull requests are extremely versatile. When a developer gets stuck with a hard problem, they can open a pull request to ask for help from the rest of the team. Alternatively, junior developers can be confident that they aren't destroying the entire project by treating pull requests as a formal code review.

Faster Release Cycle

The ultimate result of feature branches, distributed development, pull requests, and a stable community is a faster release cycle. These capabilities facilitate an agile workflow where developers are encouraged to share smaller changes more frequently. In turn, changes can get pushed down the deployment pipeline faster than the monolithic releases common with centralized version control systems.



As you might expect, Git works very well with continuous integration and continuous delivery environments. Git hooks allow you to run scripts when certain events occur inside of a repository, which lets you automate deployment to your heart's content. You can even build or deploy code from specific branches to different servers.

For example, you might want to configure Git to deploy the most recent commit from the develop branch to a test server whenever anyone merges a pull request into it. Combining this kind of build automation with peer review means you have the highest possible confidence in your code as it moves from development to staging to production.

Objections to using Git

There are five common objections I often hear to migrating to Git:

- I can overwrite history
- I have large files
- I have a very large repo
- I don't want to use GitHub
- There's a steep learning curve

Overwriting History

Git technically does allow you to overwrite history - but like any useful feature, if used incorrectly can cause conflicts. If your teams are careful, they should never have to overwrite history. And if you're synchronizing to Azure Repos, you can also add a security rule that prevents developers from overwriting history by using the explicit "Force Push" permissions. Every source control system works best when the developers using it understand how it works and which conventions work. While you can't overwrite history with TFVC, you can still overwrite code and do other painful things.

Large Files

Git works best with repos that are small and that do not contain large files (or binaries). Every time you (or your build machines) clone the repo, they get the entire repo with all its history from the first commit. This is great for most situations, but can be frustrating if you have large files. Binary files are even worse since Git just can't optimize how they are stored. That's why **Git LFS²** was created - this lets you separate large files out of your repos and still have all the benefits of versioning and comparing. Also, if you're used to storing compiled binaries in your source repos - stop! Use **Azure Artifacts³** or some other package management tool to store binaries you have source code for. However, teams that have large files (like 3D models or other assets) you can use Git LFS to keep your code repo slim and trim.

Large Repos

This used to be a blocker - but fortunately the engineers at Microsoft have been on a multi-year journey to convert all of Microsoft's source code to Git. The Windows team has a repo that's over 300GB in size, and they use Git for source control! How? They invented **Virtual File System (VFS) for Git⁴**. VFS for Git is a client plugin that lets Git think it has the entire repo - but only fetches files from the upstream repo when a file is touched. This means you can clone your giant repo in a few seconds, and only when you touch files does Git fetch them down locally. In this way, the Windows team is able to use Git even for their giant repo.

Git? GitHub?

There is a lot of confusion about Git vs GitHub. Git is the distributed source control system created by Linus Torvalds in 2005 for the Linux kernel. If you create a repo, you have a fully functioning Git repo on your local machine. However, to share that code, you need to pick a central place that developers can use to synchronize their repos - so if I want your changes, you'd push your changes to the central repo, and I'd pull them from there. We're still both working totally disconnected - but we're able to share our code via this push/pull model. GitHub is a cloud service for hosting these sorts of centralized repos - made famous mostly because it's free for open source projects (so you can host unlimited public repos). You don't have to use GitHub to use Git - though it's pretty much the de-facto platform for open source code. They do offer private repos too. You can also create Git repos in Team Foundation Server (TFS) from TFS 2015 to TFS 2019 (now renamed to Azure DevOps Server).

Learning Curve

There is a learning curve - if you've never used source control before you're probably better off when learning Git. I've found that users of centralized source control (TFVC or SubVersion) battle initially to

² <https://git-lfs.github.com/>

³ <https://azure.microsoft.com/en-us/services/devops/artifacts/>

⁴ <https://github.com/Microsoft/VFSForGit>

make the mental shift especially around branches and synchronizing. Once developers grok how Git branches work and get over the fact that they have to commit and then push, they have all the basics they need to be successful in Git.

Working with Git Locally

Git and Continuous Delivery is one of those delicious chocolate & peanut butter combinations we occasionally find in the software world, two great tastes that taste great together! Continuous Delivery of software demands a significant level of automation. It's hard to deliver continuously if you don't have a quality codebase. Git provides you with the building blocks to really take charge of quality in your codebase; it gives you the ability to automate most of the checks in your codebase even before committing the code into your repository. To fully appreciate the effectiveness of Git, you must first understand how to carry out basic operations on Git, such as clone, commit, push, and pull.

The natural question is, how do we get started with Git? One option is to go native with the command line or look for a code editor that supports Git natively. Visual Studio Code is a cross-platform open source code editor that provides a powerful developer tooling for hundreds of languages. To work in the open source, you need to embrace open source tools. In this recipe, we'll start off by setting up the development environment with Visual Studio Code, create a new Git repository, commit code changes locally, and then push changes to a remote repository on Azure DevOps Server.

Getting ready

In this tutorial, we'll learn how to initialize a Git repository locally, then we'll use the ASP.NET Core MVC project template to create a new project and version it in the local Git repository. We'll then use Visual Studio Code to interact with the Git repository to perform basic operations of commit, pull, and push. You'll need to set up your working environment with the following:

- .NET Core 3.1 SDK or later: [Download .NET⁵](https://dotnet.microsoft.com/download)
- Visual Studio Code: [Download Visual Studio Code⁶](https://code.visualstudio.com/Download)
- C# Visual Studio Code extension: [C# for Visual Studio Code⁷](https://marketplace.visualstudio.com/items?itemName=ms-dotnettools.csharp)
- Git: [Git - Downloads⁸](https://git-scm.com/downloads)
- Git for Windows (if you are using Windows): [Git for Windows⁹](https://gitforwindows.org/)

The Visual Studio Marketplace features several extensions for Visual Studio Code that you can install to enhance your experience of using Git:

- **Git Lens¹⁰:** This extension brings visualization for code history by leveraging Git blame annotations and code lens. The extension enables you to seamlessly navigate and explore the history of a file or branch. In addition to that the extension allows you to gain valuable insights via powerful comparison commands, and so much more.
- **Git History¹¹:** Brings visualization and interaction capabilities to view the Git log, file history, and compare branches or commits.

⁵ <https://dotnet.microsoft.com/download>

⁶ <https://code.visualstudio.com/Download>

⁷ <https://marketplace.visualstudio.com/items?itemName=ms-dotnettools.csharp>

⁸ <https://git-scm.com/downloads>

⁹ <https://gitforwindows.org/>

¹⁰ <https://marketplace.visualstudio.com/items?itemName=eamodio.gitlens>

¹¹ <https://marketplace.visualstudio.com/items?itemName=donjayamanne.githistory>

How to do it

1. Open the Command Prompt and create a new working folder:

```
mkdir myWebApp  
cd myWebApp
```

2. In myWebApp, initialize a new Git repository:

```
git init
```

3. Configure global settings for the name and email address to be used when committing in this Git repository:

```
git config --global user.name "Tarun Arora"  
git config --global user.email "tarun.arora@contoso.com"
```

If you are working behind an enterprise proxy, you can make your Git repository proxy-aware by adding the proxy details in the Git global configuration file. There are different variations of this command that will allow you to set up an HTTP/HTTPS proxy (with username/password) and optionally bypass SSL verification. Run the below command to configure a proxy in your global git config.

```
git config --global http.proxy  
http://proxyUsername:proxyPassword@proxy.server.com:port
```

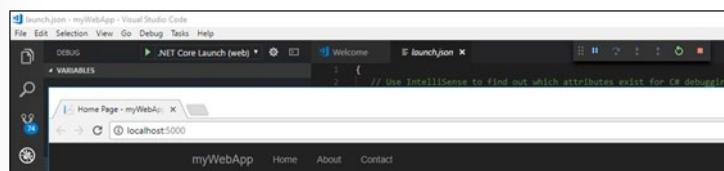
4. Create a new ASP.NET core application. The new command offers a collection of switches that can be used for language, authentication, and framework selection. More details can be found on **Microsoft docs**¹².

```
dotnet new mvc
```

Launch Visual Studio Code in the context of the current working folder:

```
code .
```

5. When the project opens up in Visual Studio Code, select Yes for the Required assets to build and debug are missing from 'myWebApp'. Add them? warning message. Select Restore for the There are unresolved dependencies info message. Hit F5 to debug the application, then myWebApp will load in the browser, as shown in the following screenshot:



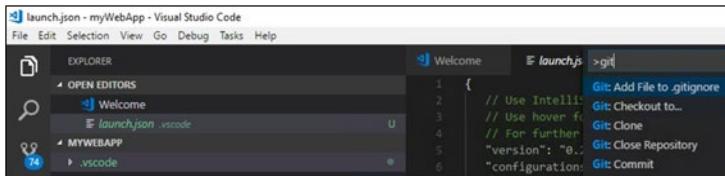
If you prefer to use the commandline you can run the following commands in the context of the git repository to run the web application.

```
dotnet build
```

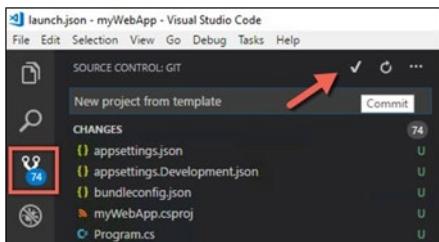
¹² <https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-new>

```
dotnet run
```

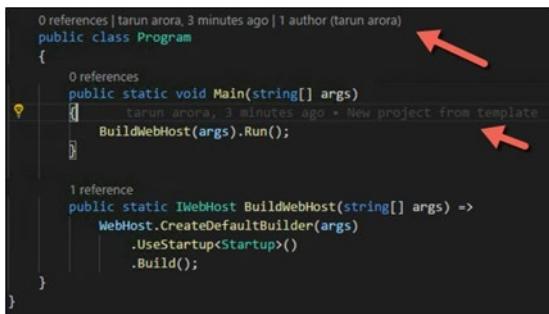
You'll notice the .vscode folder is added to your working folder. To avoid committing this folder into your Git repository, you can include this in the .gitignore file. With the .vscode folder selected, hit F1 to launch the command window in Visual Studio Code, type gitignore, and accept the option to include the selected folder in the .gitignore file:



- To stage and commit the newly-created myWebApp project to your Git repository from Visual Studio Code, navigate to the Git icon from the left panel. Add a commit comment and commit the changes by clicking the checkmark icon. This will stage and commit the changes in one operation:



Open Program.cs, you'll notice Git lens decorates the classes and functions with the commit history and also brings this information inline to every line of code:



- Now launch cmd in the context of the git repository and run `git branch --list`. This will show you that currently only `master` branch exists in this repository. Now run the following command to create a new branch called `feature-devops-home-page`

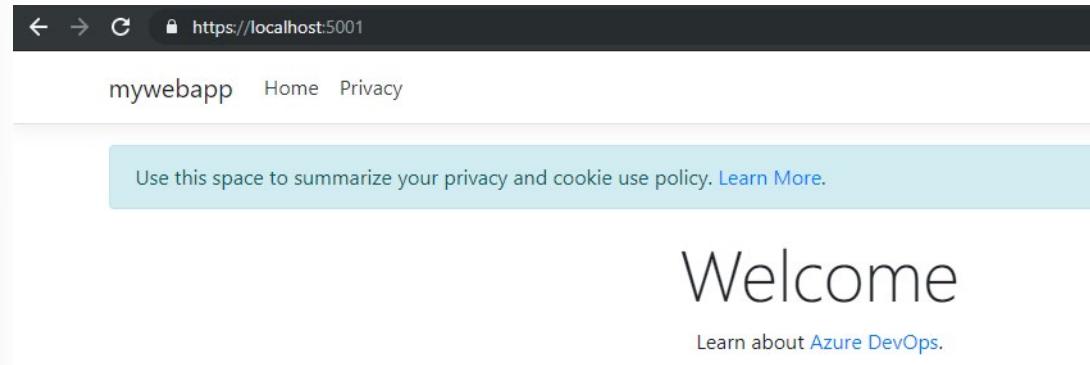
```
git branch feature-devops-home-page
git checkout feature-devops-home-page
git branch --list
```

With these commands, you have created a new branch, checked it out. The --list keyword shows you a list of all branches in your repository. The green colour represents the branch that's currently checked out.

8. Now navigate to the file ~\Views\Home\Index.cshtml and replace the contents with the text below...

```
@{  
    ViewData["Title"] = "Home Page";  
}  
  
<div class="text-center">  
    <h1 class="display-4">Welcome</h1>  
    <p>Learn about <a href="https://azure.microsoft.com/en-gb/services/devops/">Azure DevOps</a>.</p>  
</div>
```

9. Refresh the web app in the browser to see the changes...



10. In the context of the git repository execute the following commands... These commands will stage the changes in the branch and then commit them.

```
git status  
  
git add .  
  
git commit -m "updated welcome page"  
  
git status
```

11. In order to merge the changes from the feature-devops-home-page into master, run the following commands in the context of the git repository.

```
git checkout master  
  
git merge feature-devops-home-page
```

```
Updating 5d2441f..e9c9484
Fast-forward
 Views/Home/Index.cshtml | 4 +--- 
 1 file changed, 2 insertions(+), 2 deletions(-)
```

12. Run the below command to delete the feature branch

```
git branch --delete feature-devops-home-page
```

How it works

The easiest way to understand the outcome of the steps done earlier is to check the history of the operation. Let's have a look at how to do this...

1. In git, committing changes to a repository is a two step process. Upon running `add .` the changes are staged but not committed. Finally running `commit` promotes the staged changes into the repository.
2. To see the history of changes in the master branch run the command `git log -v`

```
commit e9c948427c1aa99e8aede67f6a2be206d148beaf
Author: Tarun Arora <tarun.arora@contoso.com>
Date:   Thu Jul 25 12:45:43 2019 +0100

    updated welcome page

commit 5d2441f0be4f1e4ca1f8f83b56dee31251367adc
Author: Tarun Arora <tarun.arora@contoso.com>
Date:   Thu Jul 25 12:07:55 2019 +0100

    project init
```

3. To investigate the actual changes in the commit, you can run the command `git log -p`

```
commit e9c948427c1aa99e8aede67f6a2be206d148beaf
Author: Tarun Arora <tarun.arora@contoso.com>
Date:   Thu Jul 25 12:45:43 2019 +0100

    updated welcome page

diff --git a/Views/Home/Index.cshtml b/Views/Home/Index.cshtml
index d2d19bd..6d8ad94 100644
--- a/Views/Home/Index.cshtml
+++ b/Views/Home/Index.cshtml
@@ -4,5 +4,5 @@
<div class="text-center">
    <h1 class="display-4">Welcome</h1>
-   <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">building Web apps with ASP.NET Core</a>.</p>
-</div>
+   <p>Learn about <a href="https://azure.microsoft.com/en-gb/services/devops/">Azure DevOps</a>.</p>
+</div>
\ No newline at end of file
```

There is more

Git makes it really easy to backout changes, following our example, if you wanted to take out the changes made to the welcome page, this can be done by hard resetting the master branch to a previous version of the commit using the command below...

```
git reset --hard 5d2441f0be4f1e4ca1f8f83b56dee31251367adc
```

Running the above command would reset the branch to the project init change, if you run `git log -v` you'll see that the changes done to the welcome page are completely removed from the repository.

Introduction to Azure Repos

What is Azure Repos

Azure Repos is a set of version control tools that you can use to manage your code. Whether your software project is large or small, using version control as soon as possible is a good idea.

Azure Repos provides two types of version control:

- Git: distributed version control
- Team Foundation Version Control (TFVC): centralized version control

What do I get with Azure Repos?

- Free private Git repositories, pull requests and code search: Get unlimited private Git repository hosting and support for TFVC that scales from a hobby project to the world's largest repository.
- Support for any Git client: Securely connect with and push code into your Git repos from any IDE, editor or Git client.
- Web hooks and API integration: Add validations and extensions from the marketplace or build your own using web hooks and REST APIs.
- Semantic code search: Quickly find what you're looking for with code-aware search that understands classes and variables.
- Collaborate to build better code: Perform more effective Git code reviews with threaded discussion and continuous integration for each change. Use forks to promote collaboration with inner source workflows.
- Automate with built-in CI/CD: Set up continuous integration/continuous delivery (CI/CD) to automatically trigger builds, tests and deployments with every completed pull request using Azure pipelines or your tools.
- Protect your code quality with branch policies: Keep code quality high by requiring code reviewer sign-off, successful builds and passing tests before pull requests can be merged. Customise your branch policies to maintain your team's high standards.
- Use with your favourite tools: Use Git and TFVC repositories on Azure Repos with your favourite editor and IDE.

For further reference on using git in Azure Repos refer to [Microsoft Docs¹³](#)

¹³ <https://docs.microsoft.com/en-us/azure/devops/repos/?view=azure-devops>

Introduction to GitHub

What is GitHub

GitHub is the largest open source community in the world. GitHub is owned by Microsoft. GitHub is a development platform inspired by the way you work. From open source to business, you can host and review code, manage projects, and build software alongside 40 million developers. GitHub is a Git repository hosting service, but it adds many of its own features. While Git is a command line tool, GitHub provides a Web-based graphical interface. It also provides access control and several collaboration features, such as a wikis and basic task management tools for every project. So what are the main benefits of using GitHub? To be honest, nearly every open-source project uses GitHub to manage their project. Using GitHub is free if your project is open source and includes a wiki and issue tracker that makes it easy to include more in-depth documentation and get feedback about your project.

What are some of the features offered by GitHub?

- Automate from code to cloud: Cycle your production code faster and simplify your workflow with GitHub Packages and built-in CI/CD using GitHub Actions.
- Automate your workflows: Build, test, deploy, and run CI/CD the way you want in the same place you manage code. Trigger Actions from any GitHub event to any available API. Build your own Actions in the language of your choice, or choose from thousands of workflows and Actions created by the community.
- Packages at home with their code: Use Actions to automatically publish new package versions to GitHub Packages. Install packages and images hosted on GitHub Packages or your preferred registry of record in your CI/CD workflows. It's always free for open source—and data transfer within Actions is unlimited for everyone.
- Securing software, together: GitHub play a role in securing the world's code—developers, maintainers, researchers, and security teams. On GitHub, development teams everywhere can work together to secure the world's software supply chain, from fork to finish.
 - Get alerts about vulnerabilities in your code: GitHub continuously scan security advisories for popular languages, and send security alerts to maintainers of affected repositories with details so they can remediate risks.
 - Automatically update vulnerabilities: GitHub monitors your project dependencies and automatically open pull requests to update dependencies to the minimum version that resolves known vulnerabilities.
 - Stay on top of CVEs: Stay up to date with the latest Common Vulnerabilities and Exposures (CVEs), and learn how they affect you with the GitHub Advisory Database.
 - Find vulnerabilities that other tools miss: CodeQL is the industry's leading semantic code analysis engine. GitHub's revolutionary approach treats code as data to identify security vulnerabilities faster.
 - Eliminate variants: Never make the same mistake twice. Proactive vulnerability scanning prevents vulnerabilities from ever reaching production.
 - Keep your tokens safe: Accidentally committed a token to a public repository? GitHub's got you. With support for 20 service providers GitHub takes steps to keep you safe.

- Seamless code review: Code review is the surest path to better code, and it's fundamental to how GitHub works. Built-in review tools make code review an essential part of your team's process.
 - Propose changes: Better code starts with a Pull Request, a living conversation about changes where you can talk through ideas, assign tasks, discuss details, and conduct reviews.
 - Request reviews: If you're on the other side of a review, you can request reviews from your peers to get the exact feedback you need.
 - See the difference: Reviews happen faster when you know exactly what's changed. Diffs compare versions of your source code side by side, highlighting the parts that are new, edited, or deleted.
 - Comment in context: Discussions happen in comment threads, right within your code. Bundle comments into one review, or reply to someone else's inline to start a conversation.
 - Give clear feedback: Your teammates shouldn't have to think too hard about what a thumbs up emoji means. Specify whether your comments are required changes or just a few suggestions.
 - Protect branches: Only merge the highest quality code. You can configure repositories to require status checks, reducing both human error and administrative overhead.
- All your code and documentation in one place: There are hundreds of millions of private, public, and open source repositories hosted on GitHub. Every repository is equipped with tools to help you host, version, and release code and documentation.
 - Code where you collaborate: Repositories keep code in one place and help your teams collaborate with the tools they love, even if you work with large files using Git LFS. With unlimited private repositories for individuals and teams, you can create or import as many projects as you'd like.
 - Documentation alongside your code: Host your documentation directly from your repositories with GitHub Pages. Use Jekyll as a static site generator and publish your Pages from the /docs folder on your master branch.
- Manage your ideas: Coordinate early, stay aligned, and get more done with GitHub's project management tools.
 - See your project's big picture: See everything happening in your project and choose where to focus your team's efforts with Projects, task boards that live right where they belong: close to your code.
 - Track and assign tasks: Issues help you identify, assign, and keep track of tasks within your team. You can open an Issue to track a bug, discuss an idea with an @mention, or start distributing work.
- The human side of software: Building software is as much about managing teams and communities as it is about code. Whether you're on a team of two or two thousand, GitHub's got the support your people need.
 - Manage and grow teams: Help people get organized with GitHub teams, level up access with administrative roles, and fine tune your permissions with nested teams.
 - Keep conversations on topic: Moderation tools, like issue and pull request locking, help your team stay focused on code. And if you maintain an open source project, user blocking reduces noises and ensures conversations are productive.
 - Set community guidelines: Set roles and expectations without starting from scratch. Customize common codes of conduct to create the perfect one for your project. Then choose a pre-written license right from your repository.

GitHub offers great learning resources for it's platform. You can find everything from git introduction training to deep dive on publishing static pages to GitHub and how to do DevOps on GitHub right [here¹⁴](#).

¹⁴ <https://lab.github.com/>

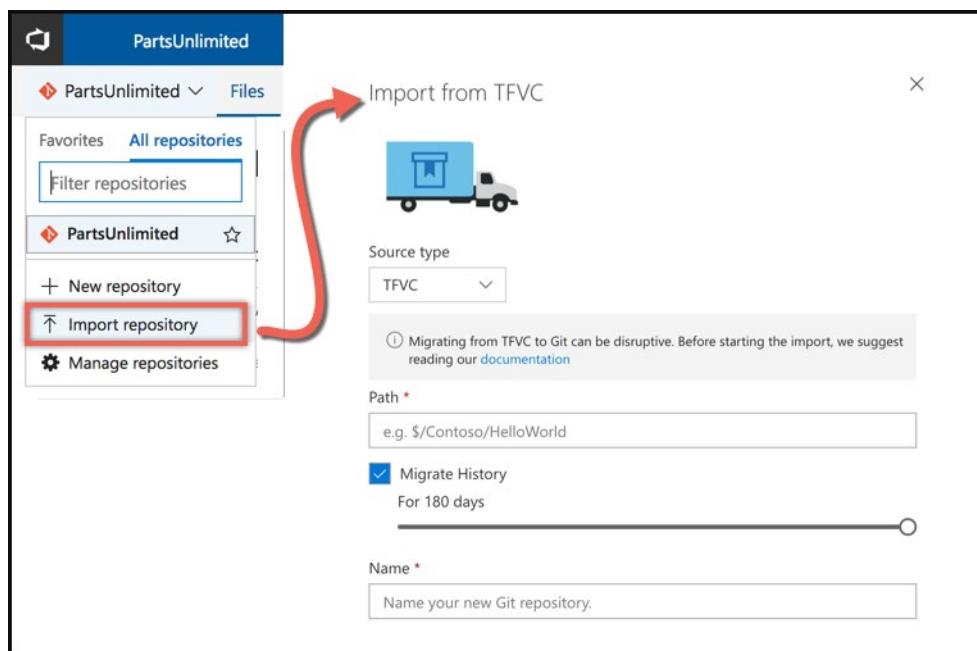
Migrating from Team Foundation Version Control (TFVC) to Git in Azure Repos

Single Branch Import

Migrating the Tip

Most teams wish they could reorganize their source control structure - typically the structure the team is using today was set up by a well-meaning developer a decade ago but it's not really optimal. Migrating to Git could be a good opportunity to restructure your repo. In this case, it probably doesn't make sense to migrate history anyway, since you're going to restructure the code (or break the code into multiple repos). The process is simple: create an empty Git repo (or multiple empty repos), then get-latest from TFS and copy/reorganize the code into the empty Git repos. Then just commit and push and you're there! Of course if you have shared code you need to create builds of the shared code to publish to a package feed and then consume those packages in downstream applications, but the Git part is really simple.

If you're on TFVC and you're in Azure DevOps, then you have the option of a simple single-branch import. Click on Import repository from the Azure Repos top level drop-down menu to open the dialog. Then enter the path to the branch you're migrating (yes, you can only choose one branch). Select if you want history or not (up to 180 days). Add in a name for the repo, and the import will be triggered.



Import repository also allows you to import a git repository, this is especially useful if you are looking to move your git repositories from GitHub or any other public or private hosting spaces into Azure Repos.

There are some limitations here (that apply only when migrating source type TFVC): a single branch and only 180 days of history. However, if you only care about one branch and you're already in Azure DevOps, then this is a very simple but effective way to do the migration.

Git TFS

What if you need to migrate more than a single branch and retain branch relationships? Or you're going to drag all the history with you? In that case, you're going to have to use Git-tfs. This is an open-source project that is build to synchronize Git and TFVC repos. But you can use it to do a once-off migration using git tfs clone. Git-tfs has the advantage that it can migrate multiple branches and will preserve the relationships so that you can merge branches in Git after you migrate. Be warned that it can take a while to do this conversion - especially for large repos or repos with long history. You can easily dry-run the migration locally, iron out any issues and then do it for real. There's lots of flexibility with this tool, so I highly recommend it.

If you're on Subversion, then you can use Git svn to import your Subversion repo in a similar manner to using Git-tfs.

Migrating from TFVC to Git Using Git-tfs

If Chocolatey is already installed on your computer, run choco install gittfs

Add the git-tfs folder path to your PATH. You could also set it temporary (the time of your current terminal session) using: set PATH=%PATH%;%cd%\GitTfs\bin\Debug

You need .NET 4.5.2 and maybe the 2012 or 2013 version of Team Explorer installed (or Visual Studio) depending on the version of TFS you want to target.

Clone the whole repository (wait for a while...) :

```
git tfs clone http://tfss:8080/tfs/DefaultCollection $/some_project
```

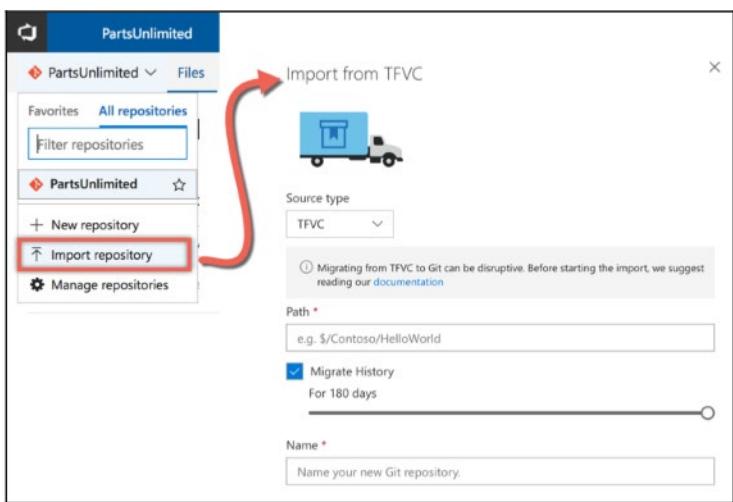
Some of the more advanced use cases of cloning the TFVC repository into git are **documented here¹⁵**.

```
cd some_project  
git log
```

Import Repository

To make it easier for you to switch from TFVC to Git, Azure DevOps server now provides an out-of-the-box migration workflow, called import repository. The import repository option can be reached from the code hub. This allows you to migrate a TFVC repository into Git with history. However, the tool only allows you to migrate up to 180 days' worth of history from your TFVC repository. Not being able to migrate the entire history from the TFVC repository may be a dealbreaker for some. The image below shows you how to get to the import repository dialogue, the image also shows the migrate history options available to you.

¹⁵ <https://github.com/git-tfs/git-tfs/blob/master/doc/commands/clone.md>



The import repository also allows you to import a Git repository, which is especially useful if you are looking to move your Git repositories from GitHub or any other public or private hosting spaces into Azure DevOps Server.

Authenticating to Git in Azure Repos

Authenticating to Your Git Repos

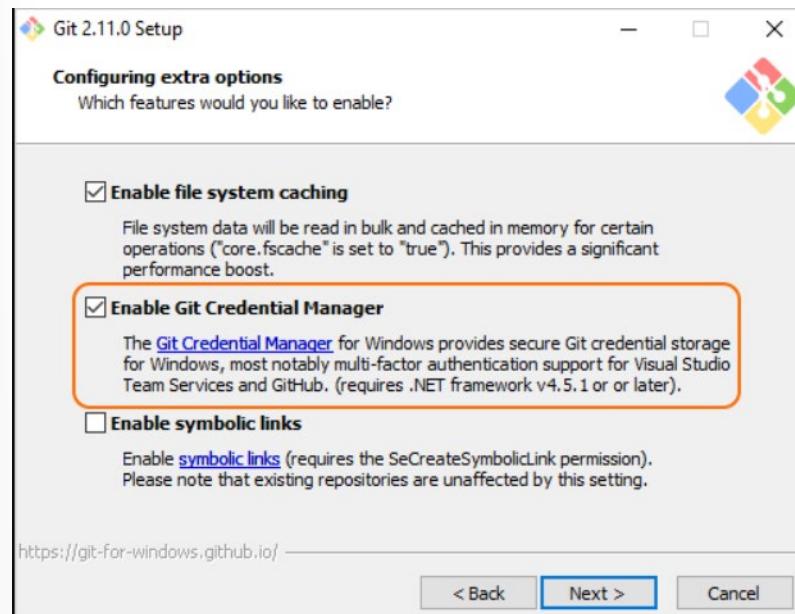
Git Credential Managers simplify authentication with your Azure DevOps Services/TFS Git repos. Credential Managers let you use the same credentials that you use for the Azure DevOps Services/TFS web portal and support multi-factor authentication through Microsoft Account (MSA) or Azure Active Directory (Azure AD). In addition to supporting multi-factor authentication with Azure DevOps Services, the credential managers also provide support two-factor authentication with GitHub repositories.

Azure DevOps Services provides IDE support for MSA and Azure AD authentication through Team Explorer in Visual Studio, IntelliJ and Android Studio with the Azure Repos Plugin for IntelliJ, and Eclipse (with the Team Explorer Everywhere plug-in). If your environment doesn't have an integration available, configure your IDE with a **Personal Access Token**¹⁶ or **SSH**¹⁷ to connect with your repos.

Install the Git Credential Manager

Windows

Download and run the latest Git for Windows installer, which includes the Git Credential Manager for Windows. Make sure to leave the Git Credential Manager installation option enabled when prompted.



macOS and Linux

Review the system and software requirements before installing the credential manager.

¹⁶ <https://docs.microsoft.com/en-us/azure/devops/organizations/accounts/use-personal-access-tokens-to-authenticate?view=azure-devops&tabs=preview-page>

¹⁷ <https://docs.microsoft.com/en-us/azure/devops/repos/git/use-ssh-keys-to-authenticate?view=azure-devops&tabs=current-page>

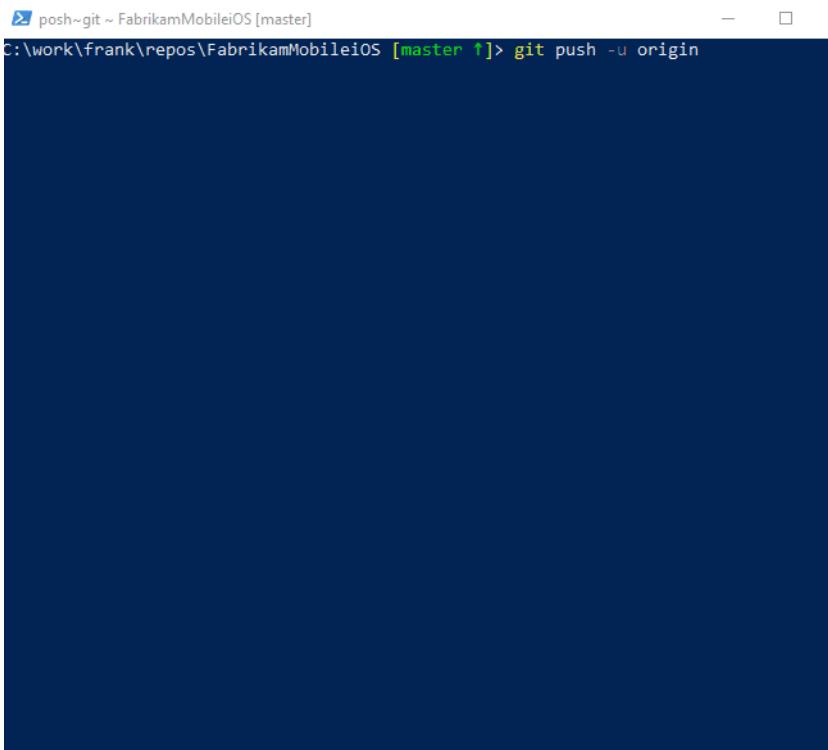
On macOS and Linux, there are several install options that use native package managers to install the credential manager. After installing the package for your platform, run the following command to configure Git to use the credential manager :

```
> git-credential-manager install
```

Demonstration - Git Credential Manager

Using the Git Credential Manager

When you connect to a Git repo in Azure Repos from your Git client for the first time, the credential manager prompts for your Microsoft Account or Azure Active Directory credentials. If your account has multi-factor authentication enabled, you are prompted to go through that experience as well.



A screenshot of a terminal window showing a command-line interface. The window title bar says "posh~git ~ FabrikamMobileiOS [master]". The command entered is "C:\work\frank\repos\FabrikamMobileiOS [master ✘]> git push -u origin". The rest of the terminal window is completely obscured by a large, solid dark blue rectangle.

Once authenticated, the credential manager creates and caches a personal access token for future connections to the repo. Git commands that connect to this account won't prompt for user credentials until the token expires or is revoked through Azure DevOps Services/TFS.

Lab

Version Controlling with Git

In this lab, **Version Controlling with Git in Azure Repos¹⁸**, you will learn how to establish a local Git repository, which can easily be synchronized with a centralized Git repository in Azure DevOps. In addition, you will learn about Git branching and merging support.

Activities to perform:

- Exercise 1: Configuring the lab environment
- Exercise 2: Cloning an existing repository
- Exercise 3: Saving work with commits
- Exercise 4: Reviewing history
- Exercise 5: Working with branches

¹⁸ <https://www.azuredevopslabs.com/labs/azuredevops/git/>

Module Review and Takeaways

Module Review Questions

Checkbox

What are some of the benefits of source control? Mark all that apply.

- reusability
- collaboration
- manageability
- efficiency
- accountability
- traceability
- automate tasks

Checkbox

What are the benefits of using Distributed Version Control? Mark all that apply.

- permits monitoring of usage
- complete offline support
- cross platform support
- allows exclusive file locking
- portable history

Checkbox

What are the benefits of using Centralized Version Control? Mark all that apply.

- easily scales for very large codebases
- an open source friendly code review model via pull requests
- granular permission control
- an enthusiastic growing user base

Suggested Answer

What is source control?

Answers

Checkbox

What are some of the benefits of source control? Mark all that apply.

- reusability
- collaboration
- manageability
- efficiency
- accountability
- traceability
- automate tasks

Explanation

Source control is the practice of tracking and managing changes to code. Benefits include: reusability, traceability, manageability, efficiency, collaboration, learning, create workflows, work with versions, collaboration, maintains history of changes, and automate tasks

Checkbox

What are the benefits of using Distributed Version Control? Mark all that apply.

- permits monitoring of usage
- complete offline support
- cross platform support
- allows exclusive file locking
- portable history

Checkbox

What are the benefits of using Centralized Version Control? Mark all that apply.

- easily scales for very large codebases
- an open source friendly code review model via pull requests
- granular permission control
- an enthusiastic growing user base

What is source control?

Source control is the practice of tracking and managing changes to code.

Module 3 Scaling Git for Enterprise DevOps

Module Overview

Module Overview

As a version control system, git is easy to get started with but difficult to master. While there is no one way to implement git in the right way, there are lots of techniques that can help you scale the implementation of git across the organization. Simple things like structuring your code into micro repos, selecting a lean branching & merging model and leveraging pull requests for code review can make your teams more productive.

Learning Objectives

After completing this module, students will be able to:

- Explain how to structure Git repos
- Describe Git branching workflows
- Leverage pull requests for collaboration and code reviews
- Leverage Git hooks for automation
- Use git to foster inner source across the organization

How to Structure Your Git Repo

Mono vs Multi Repos

A repository is simply a place where the history of your work is stored. It often lives in a .git subdirectory of your working copy. So what's the best way to organize your code repository? Software development teams start off with the best intentions to keep clear separation of concerns in both the software being developed and their code repositories. However, overtime it is not uncommon for the code repositories to be bloated with unrelated code and artifacts.

Mono-Repo or Multi-Repo?

There are two philosophies on how to organize your repos: Mono-Repo or Multi-Repo. Mono-repos are a source control pattern where all of the source code is kept in a single repository. This makes it super simple to give all of your employees access to everything in one shot. Just clone it down, and done. Multi-repos on the other hand, refers to organizing your projects each into their own separate repositories.

The fundamental difference between the mono-repo and multi-repo philosophies boils down to a difference about what will allow teams working together on a system to go fastest. The multi-repo view, in extreme form, is that if you let every sub-team live in its own repo, they have the flexibility to work in their area however they want, using whatever libraries, tools, development workflow, etc. will maximize their productivity. The cost, obviously, is that anything not developed within a given repo has to be consumed as if it was a third-party library or service, even if it was written by the person sitting one desk over. If you find a bug in, say, a library you use, you have to fix it in the appropriate repo, get a new artifact published, and then go back to your own repo to make the change to your code. In the other repo you have to deal with not only a different code base but potentially with a different libraries and tools or even a different workflow. Or maybe you just have to ask someone who owns that system to make the change for you and wait for them to get around to it.

The mono-repo view, on the other hand, is that that friction, especially when dealing with more complicated dependency graphs, is much more costly than multi-repo advocates recognize and that the productivity gains to be had by letting different teams go their own way aren't really all that significant: While it may be the case that some teams will find a locally optimal way of working, it is also likely that their gains will be offset by other teams choosing a sub-optimal way of working. By putting all your eggs in the one basket of the mono-repo you can then afford to invest in watching that basket carefully. Clearly the friction of having to make changes in other repos or, worse, having to wait for other teams to make changes for you, is largely avoided in a mono-repo because anyone can (and is in fact encouraged) to change anything. If you find a bug in a library, you can fix it and get on with your life with no more friction than if you had found a bug in your own code.

Multiple Repositories	Mono Repository
Clear ownership	Better developer testing
Better scale	Reduced code complexity
Narrow clones	Effective code reviews
-	Sharing of common components
-	Easy refactoring

Git Branching Workflows

Branching Workflow Types

What is a successful Git branch workflow?

When evaluating a workflow for your team, it's most important that you consider your team's culture. You want the workflow to enhance the effectiveness of your team and not be a burden that limits productivity. Some things to consider when evaluating a Git workflow are:

- Does this workflow scale with team size?
- Is it easy to undo mistakes and errors with this workflow?
- Does this workflow impose any new unnecessary cognitive overhead to the team?

Common branch workflows

Most popular Git workflows will have some sort of centralized repo that individual developers will push and pull from. Below is a list of some popular Git workflows that we'll go into more details in the next section. These extended workflows offer more specialized patterns in regard to managing branches for feature development, hot fixes, and eventual release.

Trunk-based Development

Trunk-based development is a logical extension of Centralized Workflow. The core idea behind the Feature Branch Workflow is that all feature development should take place in a dedicated branch instead of the master branch. This encapsulation makes it easy for multiple developers to work on a particular feature without disturbing the main codebase. It also means the master branch should never contain broken code, which is a huge advantage for continuous integration environments.

Gitflow Workflow

The Gitflow Workflow was first published in a highly regarded 2010 blog post from **Vincent Driessen at nvie¹**. The Gitflow Workflow defines a strict branching model designed around the project release. This workflow doesn't add any new concepts or commands beyond what's required for the Feature Branch Workflow. Instead, it assigns very specific roles to different branches and defines how and when they should interact.

Forking Workflow

The Forking Workflow is fundamentally different than the other workflows discussed in this tutorial. Instead of using a single server-side repository to act as the "central" codebase, it gives every developer a server-side repository. This means that each contributor has not one, but two Git repositories: a private local one and a public server-side one.

¹ <https://nvie.com/posts/a-successful-git-branching-model/>

Trunk-Based Development

The core idea behind the Feature Branch Workflow is that all feature development should take place in a dedicated branch instead of the master branch. This encapsulation makes it easy for multiple developers to work on a particular feature without disturbing the main codebase. It also means the master branch will never contain broken code, which is a huge advantage for continuous integration environments.

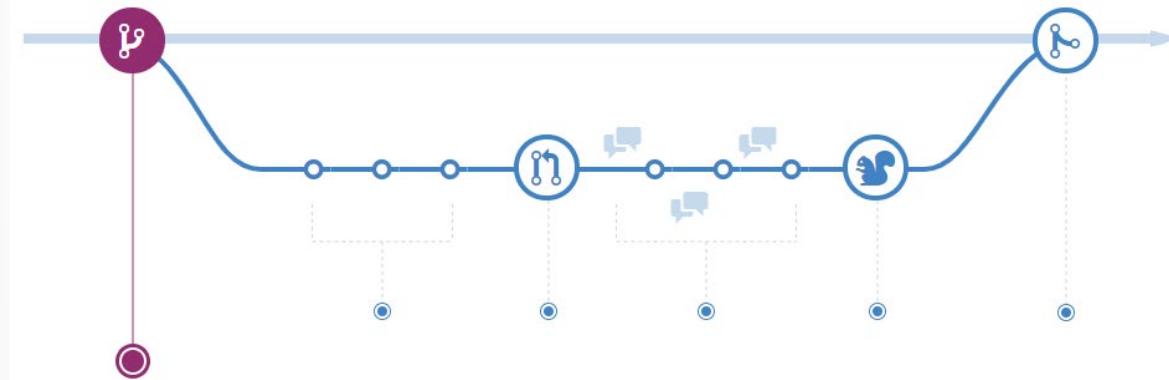
Encapsulating feature development also makes it possible to leverage pull requests, which are a way to initiate discussions around a branch. They give other developers the opportunity to sign off on a feature before it gets integrated into the official project. Or, if you get stuck in the middle of a feature, you can open a pull request asking for suggestions from your colleagues. The point is, pull requests make it incredibly easy for your team to comment on each other's work.

In addition, feature branches can (and should) be pushed to the central repository. This makes it possible to share a feature with other developers without touching any official code. Since master is the only "special" branch, storing several feature branches on the central repository doesn't pose any problems. Of course, this is also a convenient way to back up everybody's local commits.

How it works

The trunk-based development Workflow assumes a central repository, and master represents the official project history. Instead of committing directly on their local master branch, developers create a new branch every time they start work on a new feature. Feature branches should have descriptive names, like new-banner-images or bug-91. The idea is to give a clear, highly-focused purpose to each branch. Git makes no technical distinction between the master branch and feature branches, so developers can edit, stage, and commit changes to a feature branch.

Create a branch



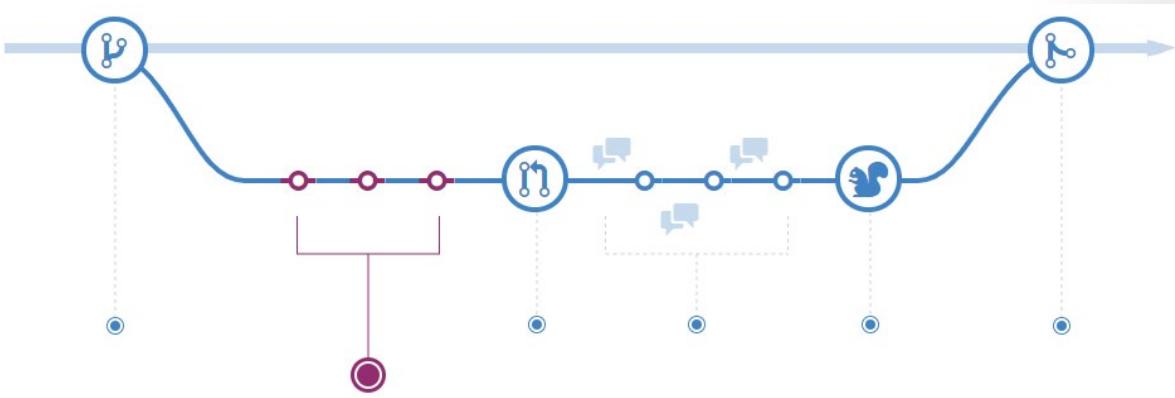
When you're working on a project, you're going to have a bunch of different features or ideas in progress at any given time – some of which are ready to go, and others which are not. Branching exists to help you manage this workflow.

When you create a branch in your project, you're creating an environment where you can try out new ideas. Changes you make on a branch don't affect the master branch, so you're free to experiment and commit changes, safe in the knowledge that your branch won't be merged until it's ready to be reviewed by someone you're collaborating with.

Branching is a core concept in Git, and the entire branch flow is based upon it. There's only one rule: anything in the master branch is always deployable.

Because of this, it's extremely important that your new branch is created off of master when working on a feature or a fix. Your branch name should be descriptive (e.g., refactor-authentication, user-content-cache-key, make-retina-avatars), so that others can see what is being worked on.

Add commits

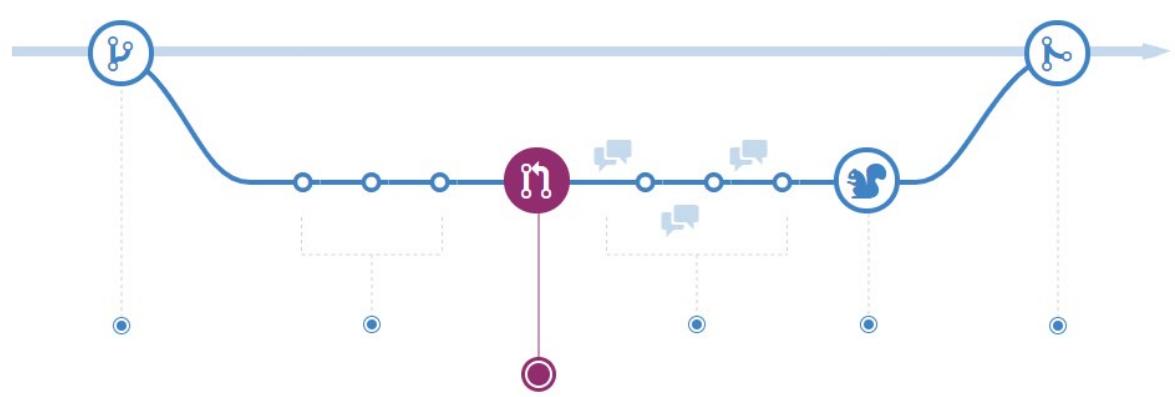


Once your branch has been created, it's time to start making changes. Whenever you add, edit, or delete a file, you're making a commit, and adding them to your branch. This process of adding commits keeps track of your progress as you work on a feature branch.

Commits also create a transparent history of your work that others can follow to understand what you've done and why. Each commit has an associated commit message, which is a description explaining why a particular change was made. Furthermore, each commit is considered a separate unit of change. This lets you roll back changes if a bug is found, or if you decide to head in a different direction.

Commit messages are important, especially since Git tracks your changes and then displays them as commits once they're pushed to the server. By writing clear commit messages, you can make it easier for other people to follow along and provide feedback.

Open a Pull Request



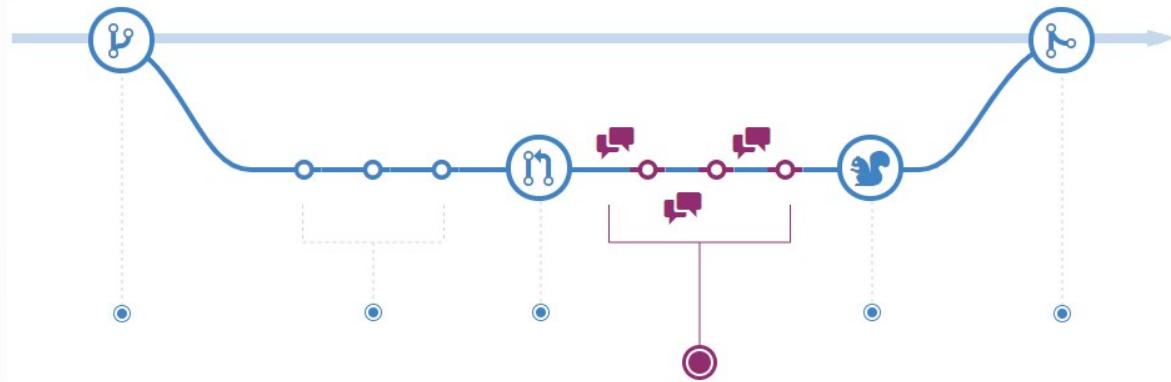
Pull Requests initiate discussion about your commits. Because they're tightly integrated with the underlying Git repository, anyone can see exactly what changes would be merged if they accept your request.

You can open a Pull Request at any point during the development process: when you have little or no code but want to share some screenshots or general ideas, when you're stuck and need help or advice, or when you're ready for someone to review your work. By using @mention system in your Pull Request

message, you can ask for feedback from specific people or teams, whether they're down the hall or ten time zones away.

Pull Requests are useful for contributing to projects and for managing changes to shared repositories. If you're using a Fork & Pull Model, Pull Requests provide a way to notify project maintainers about the changes you'd like them to consider. If you're using a Shared Repository Model, Pull Requests help start code review and conversation about proposed changes before they're merged into the master branch.

Discuss and review your code

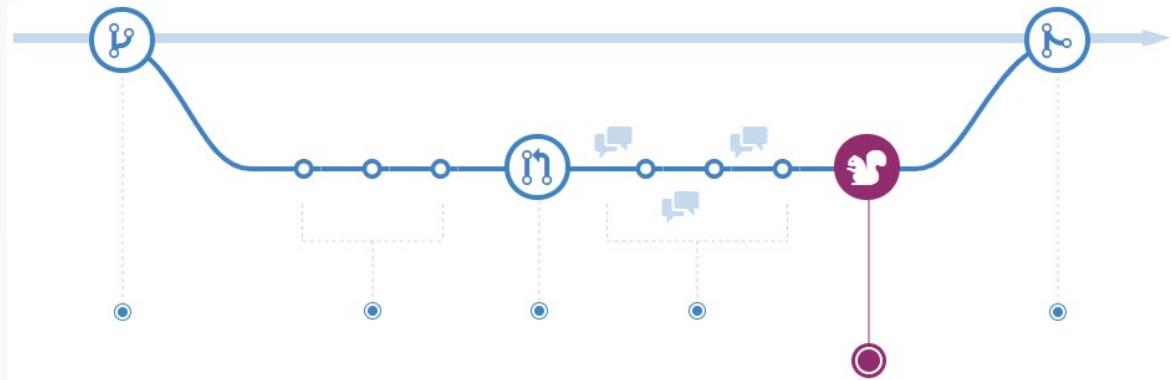


Once a Pull Request has been opened, the person or team reviewing your changes may have questions or comments. Perhaps the coding style doesn't match project guidelines, the change is missing unit tests, or maybe everything looks great and props are in order. Pull Requests are designed to encourage and capture this type of conversation.

You can also continue to push to your branch in light of discussion and feedback about your commits. If someone comments that you forgot to do something or if there is a bug in the code, you can fix it in your branch and push up the change. Git will show your new commits and any additional feedback you may receive in the unified Pull Request view.

Pull Request comments are written in Markdown, so you can embed images and emoji, use pre-formatted text blocks, and other lightweight formatting.

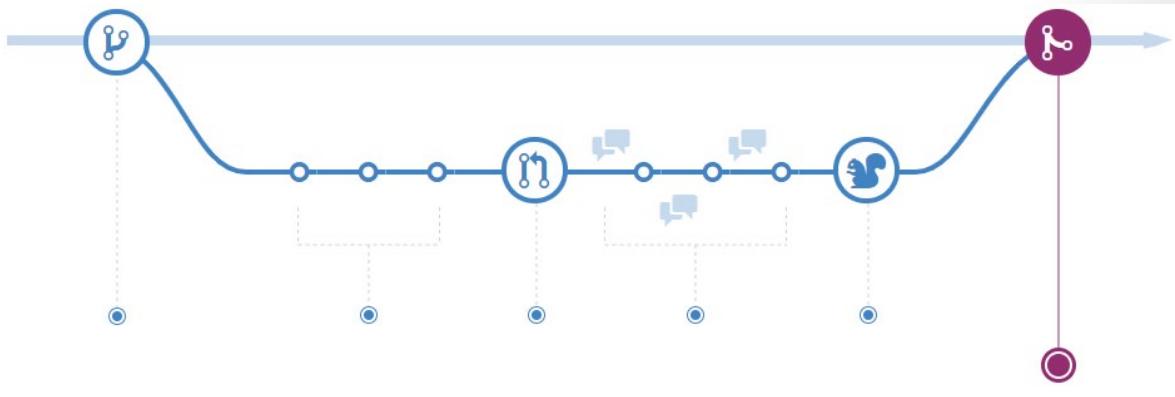
Deploy



With Git, you can deploy from a branch for final testing in an environment before merging to master.

Once your pull request has been reviewed and the branch passes your tests, you can deploy your changes to verify them . If your branch causes issues, you can roll it back by deploying the existing master.

Merge



Now that your changes have been verified, it is time to merge your code into the master branch.

Once merged, Pull Requests preserve a record of the historical changes to your code. Because they're searchable, they let anyone go back in time to understand why and how a decision was made.

By incorporating certain keywords into the text of your Pull Request, you can associate issues with code. When your Pull Request is merged, the related issues can also closed.

This workflow helps organize and track branches that are focused on business domain feature sets. Other Git workflows like the Git Forking Workflow and the Gitflow Workflow are repo focused and can leverage the Git Feature Branch Workflow to manage their branching models.

Git-Branching model for continuous delivery

The purpose of writing code is to ship enhancements to your software. A branching model that introduces too much process overhead does not help in increasing the speed with which you can get changes out to customers. It is therefore important to come up with a branching model that gives you enough padding to not ship poor-quality changes but at the same time not introduce too many processes to slow you down. The internet is full of branching strategies for Git; while there is no right or wrong, a perfect branching strategy is one that works for your team! In this recipe, we'll learn how to use the combination of feature branches and pull requests to always have a ready-to-ship master branch and how to sync bug fixes fixed in fix of fail branches back into master to avoid regression.

Getting ready

Let's cover the principles of what is being proposed:

- The master branch:
 - The master branch is the only way to release anything to production.
 - The master branch should always be in a ready-to-release state.
 - Protect the master branch with branch policies.
 - Any changes to the master branch flow through pull requests only.
 - Tag all releases in the master branch with Git tags.

- The feature branch:
 - Use feature branches for all new features and bug fixes.
 - Use feature flags to manage long-running feature branches.
 - Changes from feature branches to the master only flow through pull requests.
 - Name your feature to reflect their purpose.

List of branches

```
features/feature-area/feature-name  
users/username/description  
users/username/workitem  
bugfix/description  
features/feature-name  
features/feature-area/feature-name  
hotfix/description
```

- Pull requests:
 - Review and merge code with pull requests.
 - Automate what you inspect and validate as part of pull requests.
 - Track pull request completion duration and set goals to reduce the time it takes.

In this recipe, we'll be using the myWebApp created in the Pull Request for code review using branch policies recipe. If you haven't already, follow that recipe to lock down the master branch using branch policies. In this recipe, we'll also be using two very popular extensions from the marketplace:

- Azure DevOps CLI (<https://marketplace.visualstudio.com/items?itemName=ms-vsts.cli>): This is a new command-line experience for Azure DevOps (AzDo) and Azure DevOps Server (AzDos), designed to seamlessly integrate with Git, CI pipelines, and Agile tools. With the Azure DevOps CLI, you can contribute to your projects without ever leaving the command line. CLI runs on Windows, Linux, and Mac.
- Git Pull Request Merge Conflict (<https://marketplace.visualstudio.com/items?itemName=ms-devlabs.conflicts-tab>): This open source extension created by Microsoft DevLabs allows you to review and resolve pull request merge conflicts on the web. Before a Git pull request can complete, any conflicts with the target branch must be resolved. With this extension, you can resolve these conflicts on the web, as part of the pull request merge, instead of performing the merge and resolving conflicts in a local clone.

```
$AzureDevOpsServer = "https://dev.azure.com/Geeks"  
az devops login --token xxxxxxxx --instance $AzureDevOpsServer  
$prj = "PartsUnlimited"  
az devops code repo list --instance $i --project $prj --output table
```

vsts code repo list --instance \$i --project \$prj --output table			
ID	Name	Default Branch	Project
9b08d519-37a6-4aed-84e6-3a2712f742a9	MyWebApp	master	PartsUnlimited
b2c65132-d148-49e0-81aa-ce106fbf3747	PartsUnlimited		PartsUnlimited

The Azure DevOps CLI supports returning the results of the query in JSON, JSONC, table, and TSV. You can configure your preference by using the configure command.

How to do it

1. After you've cloned the master branch into a local repository, create a new feature branch, myFeature-1:

```
myWebApp> git checkout -b feature/myFeature-1  
Switched to a new branch 'feature/myFeature-1'
```

2. Run the Git branch command to see all the branches, the branch showing up with asterisk is the currently-checked-out branch:

```
myWebApp> git branch * feature/myFeature-1 maste
```

3. Make a change to the Program.cs file in the feature/myFeature-1 branch:

```
myWebApp> notepad Program.cs
```

4. Stage your changes and commit locally, then publish your branch to remote:

```
myWebApp> git status
```

```
On branch feature/myFeature-1 Changes not staged for commit: (use "git add <file>..." to update what will be committed) (use "git checkout -- <file>..." to discard changes in working directory) modified: Program.cs
```

```
myWebApp> git add .
```

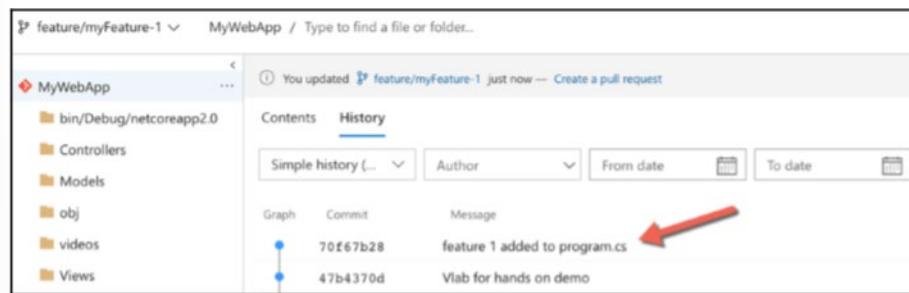
```
myWebApp> git commit -m "Feature 1 added to Program.cs"
```

```
[feature/myFeature-1 70f67b2] feature 1 added to program.cs 1 file changed,  
1 insertion(+)
```

```
myWebApp> git push -u origin feature/myFeature-1
```

```
Delta compression using up to 8 threads. Compressing objects: 100% (3/3),  
done. Writing objects: 100% (3/3), 348 bytes | 348.00 KiB/s, done. Total 3  
(delta 2), reused 0 (delta 0) remote: Analyzing objects... (3/3) (10 ms)  
remote: Storing packfile... done (44 ms) remote: Storing index... done (62  
ms) To http://dev.azure.com/Geeks/PartsUnlimited/_git/MyWebApp * [new  
branch] feature/myFeature-1 -> feature/myFeature-1 Branch feature/myFea-  
ture-1 set up to track remote branch feature/myFeature-1 from origin.
```

The remote shows the history of the changes:

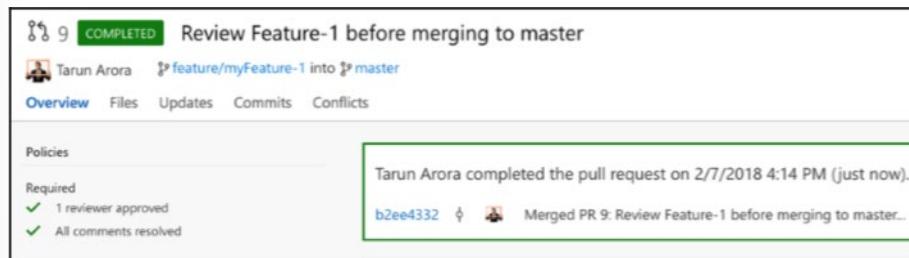


5. Create a new pull request (using the Azure DevOps CLI) to review the changes in the feature-1 branch:

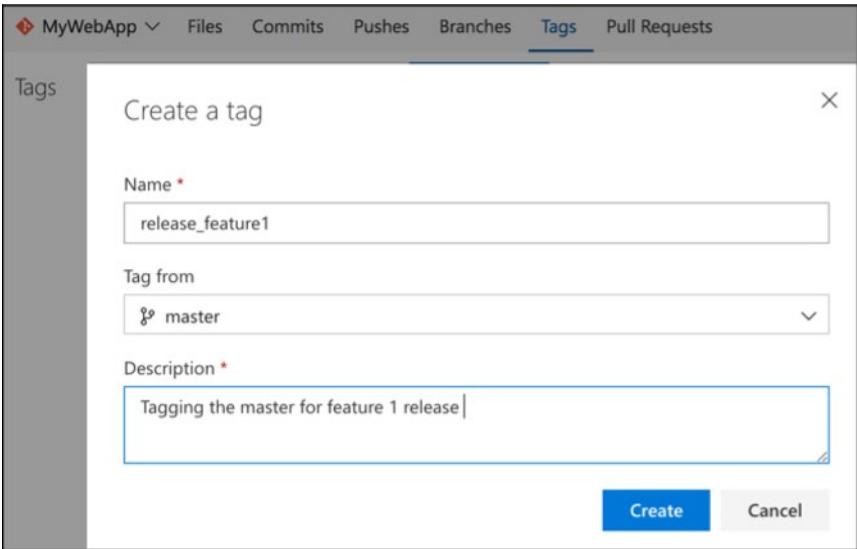
```
> az code pr create --title "Review Feature-1 before merging to master"  
--work-items 38 39 `  
-d "#Merge feature-1 to master" `  
-s feature/myFeature-1 -t master -r myWebApp -p  
$prj -i $i
```

Use the `--open` switch when raising the pull request to open the pull request in a web browser after it has been created. The `--deletesource-branch` switch can be used to delete the branch after the pull request is complete. Also consider using `--auto-complete` to complete automatically when all policies have passed and the source branch can be merged into the target branch.

The team jointly reviews the code changes and approves the pull request:



The master is ready to release, team tags master branch with the release number:



6. Start work on Feature 2. Create a branch on remote from the master branch and do the checkout locally:

```
myWebApp> git push origin origin:refs/heads/feature/myFeature-2
```

```
Total 0 (delta 0), reused 0 (delta 0) To https://dev.azure.com/Geeks/PartsUnlimited/_git/MyWebApp * [new branch] origin/HEAD -> refs/heads/feature/myFeature-2
```

```
myWebApp> git checkout feature/myFeature-2
```

```
Switched to a new branch 'feature/myFeature-2' Branch feature/myFeature-2 set up to track remote branch feature/myFeature-2 from origin
```

7. Modify Program.cs by changing the same line of code that was changed in feature-1

```
public class Program
{
    // Editing the same line (file from feature-2 branch) | <-- Red arrow
    public static void Main(string[] args)
    {
        BuildWebHost(args).Run();
    }

    public static IWebHost BuildWebHost(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>()
            .Build();
}
```

8. Commit the changes locally, push to the remote repository, and then raise a pull request to merge the changes from feature/myFeature-2 to the master branch:

```
> az code pr create --title "Review Feature-2 before merging to master"
--work-items 40 42
          -d "#Merge feature-2 to master" ` 
          -s feature/myFeature-2 -t master -r myWebApp -p
$prj -i $1
```

With the pull request in flight, a critical bug is reported in production against the feature-1 release. In order to investigate the issue, you need to debug against the version of code currently deployed in production. To investigate the issue, create a new fof branch using the release_feature1 tag:

```
myWebApp> git checkout -b fof/bug-1 release_feature1  
Switched to a new branch 'fof/bug-1'
```

9. Modify Program.cs by changing the same line of code that was changed in the feature-1 release:

```
// Editing this file from [feature-fof branch] ↗  
public static void Main(string[] args)  
{  
    BuildWebHost(args).Run();  
}  
  
public static IWebHost BuildWebHost(string[] args) =>  
    WebHost.CreateDefaultBuilder(args)  
        .UseStartup<Startup>()  
        .Build();
```

10. Stage and commit the changes locally, then push changes to the remote repository:

```
myWebApp> git add .
```

```
myWebApp> git commit -m "Adding FOF changes"
```

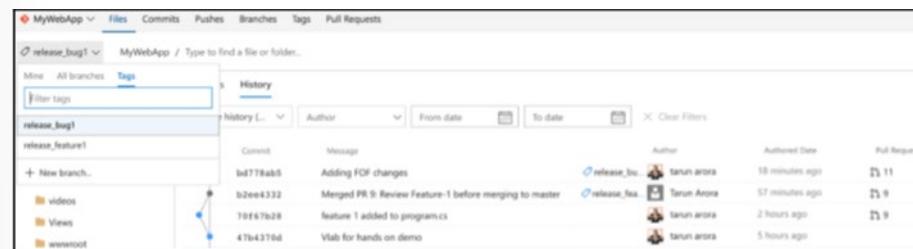
```
myWebApp> git push -u origin fof/bug-1
```

```
To https://dev.azure.com/Geeks/PartsUnlimited/_git/MyWebApp * [new branch]  
fof/bug-1 -> fof/bug-1 Branch fof/bug-1 set up to track remote branch fof/  
bug-1 from origin
```

11. Immediately after the changes have been rolled out to production, tag the fof\bug-1 branch with the release_bug-1 tag, then raise a pull request to merge the changes from fof/bug-1 back into the master:

```
> az code pr create --title "Review Bug-1 before merging to master" --work-  
items 100 `  
      -d "#Merge Bug-1 to master" `  
      -s fof/Bug-1 -t master -r myWebApp -p  
$prj -i $i
```

As part of the pull request, the branch is deleted, however, you can still reference the full history to that point using the tag:



Commit	Message	Author	Authored Date	Pull Request
bd778ab0	Adding FOF changes	release_bug-1	18 minutes ago	PR 11
b2ee4332	Merged PR 9: Review Feature-1 before merging to master	release_bug-1	17 minutes ago	PR 9
70f67828	feature 1 added to program.cs	tarun arora	2 hours ago	PR 9
47b4378d	Vital for hands on demo	tarun arora	5 hours ago	

With the critical bug fix out of the way, let's go back to the review of the feature-2 pull request. The branches page makes it clear that the feature/myFeature-2 branch is one change ahead of the master and two changes behind the master:

The screenshot shows the GitHub 'Branches' page for the 'MyWebApp' repository. The 'feature' branch is selected. It lists two commits: 'myFeature-2' (commit f8efdb74 by tarun arora, 44 minutes ago) and 'master' (commit fddee4165 by Tarun Arora, 14 minutes ago). The 'Behind | Ahead' column shows '2 | 1'.

If you tried to approve the pull request, you'll see an error message informing you of a merge conflict:

The screenshot shows the GitHub pull request review interface for 'feature-2' into 'master'. The 'Conflicts' tab is active, showing a conflict in 'Program.cs' where both branches have edited the same line. A red box highlights the conflict message: 'conflict prevents automatic merging'.

12. The Git Pull Request Merge Conflict resolution extension makes it possible to resolve merge conflicts right in the browser. Navigate to the conflicts tab and click on Program.cs to resolve the merge conflicts:

The screenshot shows the 'Program.cs' file in the 'Conflicts' tab of the Git Pull Request Merge Conflict resolution extension. A red arrow points to the 'Submit Merge' button at the top right. The code editor shows a conflict between 'feature-2' and 'master' branches, with specific lines highlighted in red, green, and yellow to indicate the source of each edit.

The user interface gives you the option to take the source version, target version, or add custom changes and review and submit the merge. With the changes merged, the pull request is completed.

How it works

In this recipe, we learned how the Git branching model gives you the flexibility to work on features in parallel by creating a branch for each feature. The pull request workflow allows you to review code changes using the branch policies. Git tags are a great way to record milestones, such as the version of code released; tags give you a way to create branches from tags. We were able to create a branch from a previous release tag to fix a critical bug in production. The branches view in the web portal makes it easy to identify branches that are ahead of the master, and forces a merge conflict if any ongoing pull requests try to merge to the master without first resolving the merge conflicts. A lean branching model such as this allows you to create short-lived branches and push quality changes to production faster.

GitFlow Branch Workflow

Gitflow Workflow is a Git workflow design that was first published and made popular by Vincent Driessen at nvie. The Gitflow Workflow defines a strict branching model designed around the project release. This provides a robust framework for managing larger projects.

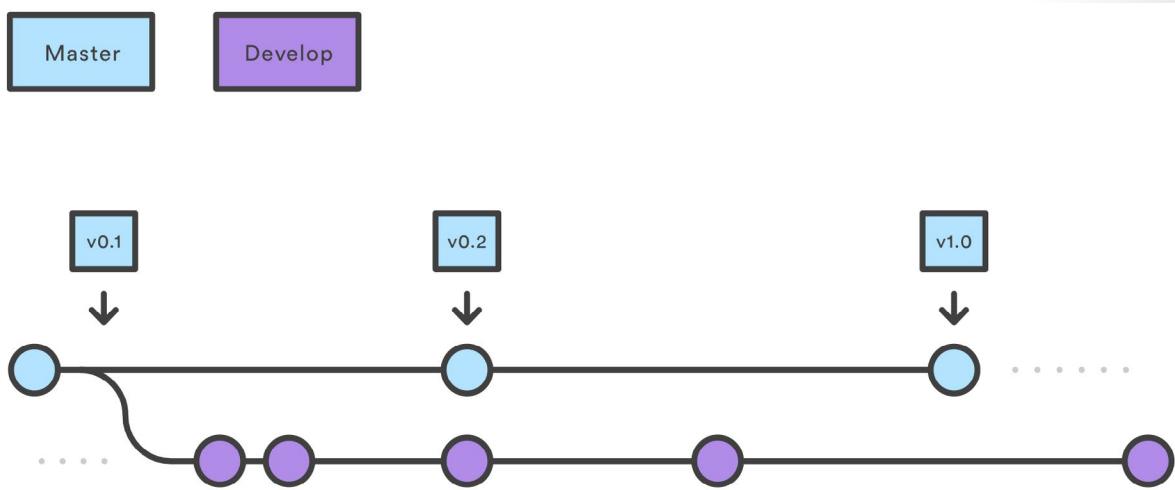
Gitflow is ideally suited for projects that have a scheduled release cycle. This workflow doesn't add any new concepts or commands beyond what's required for the Feature Branch Workflow. Instead, it assigns very specific roles to different branches and defines how and when they should interact. In addition to feature branches, it uses individual branches for preparing, maintaining, and recording releases. Of course, you also get to leverage all the benefits of the Feature Branch Workflow: pull requests, isolated experiments, and more efficient collaboration.

In addition to the abstract Gitflow Workflow idea, there is a more tangible git-flow toolset available which integrates with Git to provide specialized Gitflow Git command line tool extensions.

Getting Started

Gitflow is really just an abstract idea of a Git workflow. This means it dictates what kind of branches to set up and how to merge them together. We will touch on the purposes of the branches below. The git-flow toolset is an actual command line tool that has an installation process. The installation process for git-flow is straightforward. Packages for git-flow are available on multiple operating systems. On OSX systems, you can execute brew install git-flow. On windows you will need to download and install git-flow. After installing git-flow you can use it in your project by executing git flow init. Git-flow is a wrapper around Git. The git flow init command is an extension of the default git init command and doesn't change anything in your repository other than creating branches for you.

How it works



Develop and Master Branches

Instead of a single master branch, this workflow uses two branches to record the history of the project. The master branch stores the official release history, and the develop branch serves as an integration branch for features. It's also convenient to tag all commits in the master branch with a version number.

The first step is to complement the default master with a develop branch. A simple way to do this is for one developer to create an empty develop branch locally and push it to the server:

```
git branch develop
git push -u origin develop
```

This branch will contain the complete history of the project, whereas master will contain an abridged version. Other developers should now clone the central repository and create a tracking branch for develop.

When using the git-flow extension library, executing git flow init on an existing repo will create the develop branch:

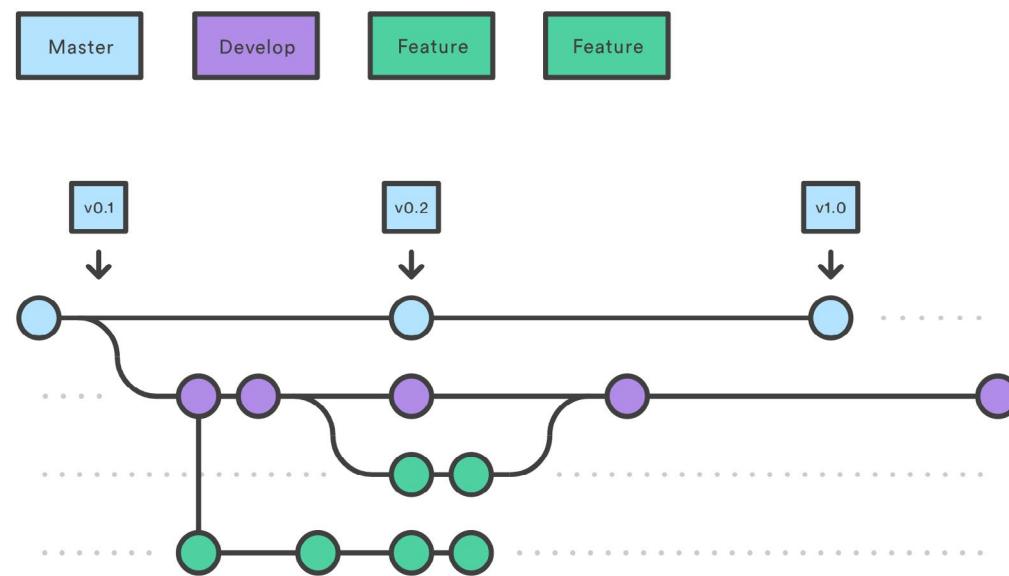
```
Initialized empty Git repository in ~/project/.git/
No branches exist yet. Base branches must be created now.
Branch name for production releases: [master]
Branch name for "next release" development: [develop]
```

How to name your supporting branch prefixes?
 Feature branches? [feature/]
 Release branches? [release/]
 Hotfix branches? [hotfix/]
 Support branches? [support/]
 Version tag prefix? []

```
$ git branch
* develop
  master
```

Feature Branches

Each new feature should reside in its own branch, which can be pushed to the central repository for backup/collaboration. But, instead of branching off of master, feature branches use develop as their parent branch. When a feature is complete, it gets merged back into develop. Features should never interact directly with master.



Note that feature branches combined with the develop branch is, for all intents and purposes, the Feature Branch Workflow. But, the Gitflow Workflow doesn't stop there.

Feature branches are generally created off to the latest develop branch.

Creating a feature branch

Without the git-flow extensions:

```
git checkout develop
git checkout -b feature_branch
```

When using the git-flow extension:

```
git flow feature start feature_branch
```

Continue your work and use Git like you normally would.

Finishing a feature branch

When you're done with the development work on the feature, the next step is to merge the feature_branch into develop.

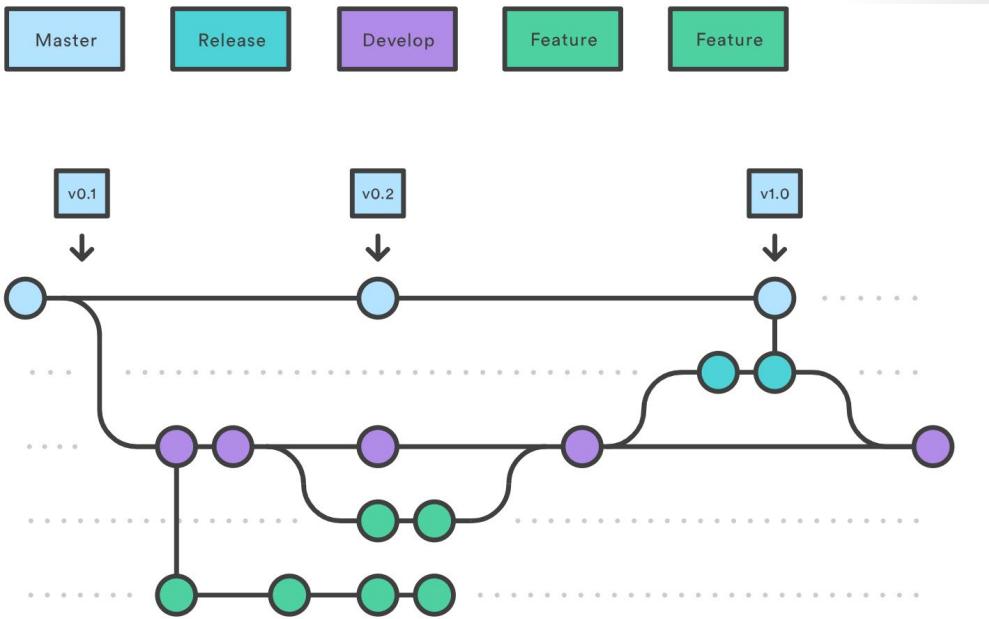
Without the git-flow extensions:

```
git checkout develop
git merge feature_branch
```

Using the git-flow extensions:

```
git flow feature finish feature_branch
```

Release Branches



Once develop has acquired enough features for a release (or a predetermined release date is approaching), you fork a release branch off of develop. Creating this branch starts the next release cycle, so no new features can be added after this point—only bug fixes, documentation generation, and other release-oriented tasks should go in this branch. Once it's ready to ship, the release branch gets merged into master and tagged with a version number. In addition, it should be merged back into develop, which may have progressed since the release was initiated.

Using a dedicated branch to prepare releases makes it possible for one team to polish the current release while another team continues working on features for the next release. It also creates well-defined phases of development (e.g., it's easy to say, "This week we're preparing for version 4.0," and to actually see it in the structure of the repository).

Making release branches is another straightforward branching operation. Like feature branches, release branches are based on the develop branch. A new release branch can be created using the following methods.

Without the git-flow extensions:

```
git checkout develop
git checkout -b release/0.1.0
`` cmd
```

When using the git-flow extensions:

```
`` cmd
$ git flow release start 0.1.0
Switched to a new branch 'release/0.1.0'
```

Once the release is ready to ship, it will get merged into master and develop, then the release branch will be deleted. It's important to merge back into develop because critical updates may have been added to the release branch and they need to be accessible to new features. If your organization stresses code review, this would be an ideal place for a pull request.

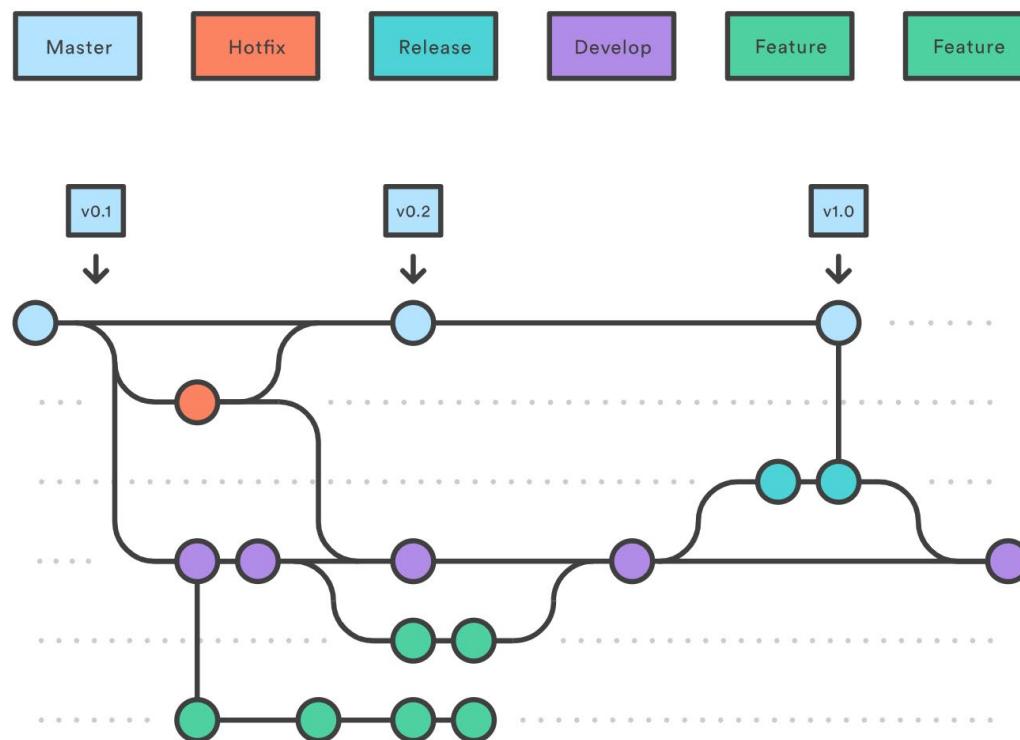
To finish a release branch, use the following methods:

Without the git-flow extensions:

```
git checkout develop  
git merge release/0.1.0
```

Or with the git-flow extension:

```
git checkout master  
git checkout merge release/0.1.0  
git flow release finish '0.1.0'
```



Maintenance or “hotfix” branches are used to quickly patch production releases. Hotfix branches are a lot like release branches and feature branches except they’re based on master instead of develop. This is the only branch that should fork directly off of master. As soon as the fix is complete, it should be merged into both master and develop (or the current release branch), and master should be tagged with an updated version number.

Having a dedicated line of development for bug fixes lets your team address issues without interrupting the rest of the workflow or waiting for the next release cycle. You can think of maintenance branches as ad hoc release branches that work directly with master. A hotfix branch can be created using the following methods:

Without the git-flow extensions:

```
git checkout master  
git checkout -b hotfix_branch
```

When using the git-flow extensions:

```
$ git flow hotfix start hotfix_branch
```

Similar to finishing a release branch, a hotfix branch gets merged into both master and develop.

```
git checkout master
git merge hotfix_branch
git checkout develop
git merge hotfix_branch
git branch -D hotfix_branch
$ git flow hotfix finish hotfix_branch
```

Some key takeaways to know about Gitflow are:

- The workflow is great for a release-based software workflow.
- Gitflow offers a dedicated channel for hotfixes to production.

The overall flow of Gitflow is:

- A develop branch is created from master
- A release branch is created from develop
- Feature branches are created from develop
- When a feature is complete it is merged into the develop branch
- When the release branch is done it is merged into develop and master
- If an issue in master is detected a hotfix branch is created from master
- Once the hotfix is complete it is merged to both develop and master

Forking Workflow

The Forking Workflow is fundamentally different than other popular Git workflows. Instead of using a single server-side repository to act as the “central” codebase, it gives every developer their own server-side repository. This means that each contributor has not one, but two Git repositories: a private local one and a public server-side one. The Forking Workflow is most often seen in public open source projects.

The main advantage of the Forking Workflow is that contributions can be integrated without the need for everybody to push to a single central repository. Developers push to their own server-side repositories, and only the project maintainer can push to the official repository. This allows the maintainer to accept commits from any developer without giving them write access to the official codebase.

The Forking Workflow typically follows a branching model based on the Gitflow Workflow. This means that complete feature branches will be purposed for merge into the original project maintainer’s repository. The result is a distributed workflow that provides a flexible way for large, organic teams (including untrusted third-parties) to collaborate securely. This also makes it an ideal workflow for open source projects.

How it works

As in the other Git workflows, the Forking Workflow begins with an official public repository stored on a server. But when a new developer wants to start working on the project, they do not directly clone the official repository.

Instead, they fork the official repository to create a copy of it on the server. This new copy serves as their personal public repository—no other developers are allowed to push to it, but they can pull changes from it (we'll see why this is important in a moment). After they have created their server-side copy, the developer performs a git clone to get a copy of it onto their local machine. This serves as their private development environment, just like in the other workflows.

When they're ready to publish a local commit, they push the commit to their own public repository—not the official one. Then, they file a pull request with the main repository, which lets the project maintainer know that an update is ready to be integrated. The pull request also serves as a convenient discussion thread if there are issues with the contributed code. The following is a step-by-step example of this workflow.

- A developer 'forks' an 'official' server-side repository. This creates their own server-side copy.
- The new server-side copy is cloned to their local system.
- A Git remote path for the 'official' repository is added to the local clone.
- A new local feature branch is created.
- The developer makes changes on the new branch.
- New commits are created for the changes.
- The branch gets pushed to the developer's own server-side copy.
- The developer opens a pull request from the new branch to the 'official' repository.
- The pull request gets approved for merge and is merged into the original server-side repository

To integrate the feature into the official codebase, the maintainer pulls the contributor's changes into their local repository, checks to make sure it doesn't break the project, merges it into their local master branch, then pushes the master branch to the official repository on the server. The contribution is now part of the project, and other developers should pull from the official repository to synchronize their local repositories.

It's important to understand that the notion of an "official" repository in the Forking Workflow is merely a convention. In fact, the only thing that makes the official repository so official is that it's the repository of the project maintainer.

Forking vs cloning

It's important to note that "forked" repositories and "forking" are not special operations. Forked repositories are created using the standard git clone command. Forked repositories are generally "server-side clones" and usually managed and hosted by a Git service provider such as Azure Repos. There is no unique Git command to create forked repositories. A clone operation is essentially a copy of a repository and its history.

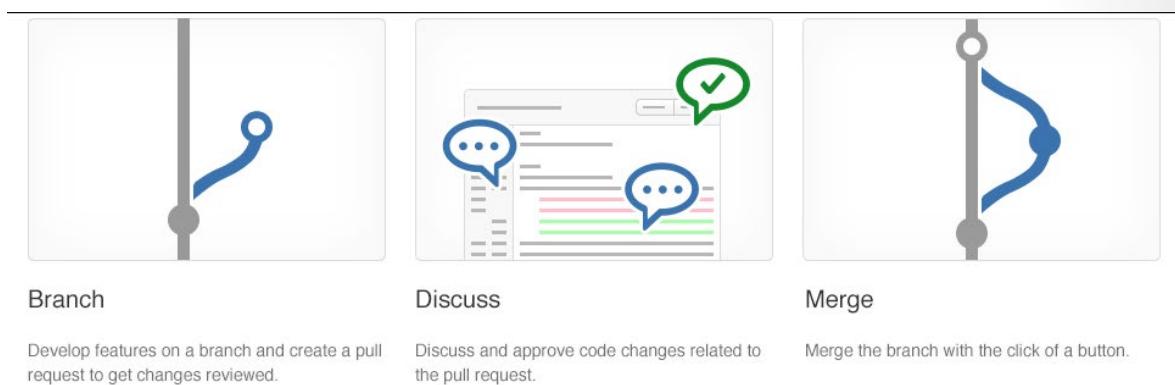
Collaborating with Pull Requests in Azure Repos

Collaborating with Pull Requests

Pull requests let you tell others about changes you've pushed to a GitHub repository. Once a pull request is sent, interested parties can review the set of changes, discuss potential modifications, and even push follow-up commits if necessary.

Pull Requests are commonly used by teams and organizations collaborating using the Shared Repository Model, where everyone shares a single repository and topic branches are used to develop features and isolate changes. Many open source projects on Github use pull requests to manage changes from contributors as they are useful in providing a way to notify project maintainers about changes one has made and in initiating code review and general discussion about a set of changes before being merged into the main branch.

Pull requests combine the review and merge of your code into a single collaborative process. Once you're done fixing a bug or new feature in a branch, create a new pull request. Add the members of the team to the pull request so they can review and vote on your changes. Use pull requests to review works in progress and get early feedback on changes. There's no commitment to merge the changes as the owner can abandon the pull request at any time.



Get your code reviewed

The code review done in a pull request isn't just to find obvious bugs—that's what your tests are for. A good code review catches less obvious problems that could lead to costly issues later. Code reviews help protect your team from bad merges and broken builds that sap your team's productivity. The review catches these problems before the merge, protecting your important branches from unwanted changes.

Cross-pollinate expertise and spread problem solving strategies by using a wide range of reviewers in your code reviews. Diffusing skills and knowledge makes your team stronger and more resilient.

Give great feedback

High quality reviews start with high quality feedback. The keys to great feedback in a pull request are:

- Have the right people review the pull request
- Make sure that reviewers know what the code does
- Give actionable, constructive feedback

- Reply to comments in a timely manner

When assigning reviewers to your pull request, make sure you select the right set of reviewers. You want reviewers that will know how your code works, but try to also include developers working in other areas so they can share their ideas. Provide a clear description of your changes and provide a build of your code that has your fix or feature running in it. Reviewers should make an effort to provide feedback on changes they don't agree with. Identify the issue and give a specific suggestions on what you would do differently. This feedback has clear intent and is easy for the owner of the pull request to understand. The pull request owner should reply to the comments, accepting the suggestion or explaining why the suggested change isn't ideal. Sometimes a suggestion is good, but the changes are outside the scope of the pull request. Take these suggestions and create new work items and feature branches separate from the pull request to make those changes.

Protect branches with policies

There are a few critical branches in your repo that the team relies on always being in good shape, such as your master branch. Require pull requests to make any changes on these branches. Developers pushing changes directly to the protected branches will have their pushes rejected.

Add additional conditions to your pull requests to enforce a higher level of code quality in your key branches. A clean build of the merged code and approval from multiple reviewers are some extra requirements you can set to protect your key branches.

Demonstration Azure Repos Collaborating with Pull Requests

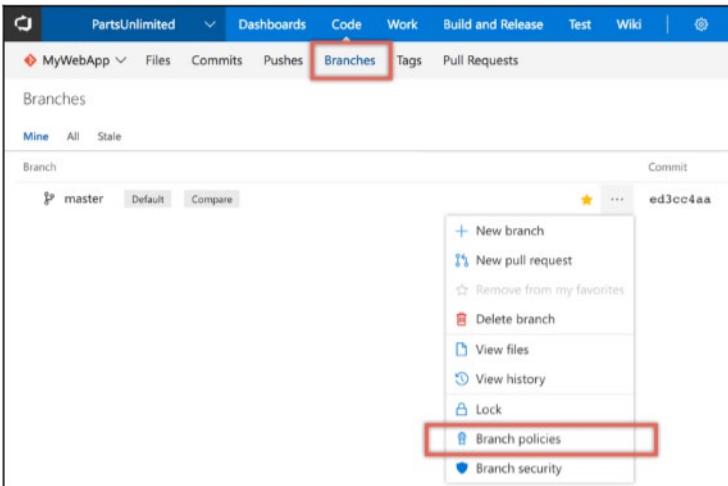
Code issues that are found sooner are both easier and cheaper to fix, therefore development teams strive to push code quality checks as far left into the development process as possible. As the name suggests, branch policies give you a set of out-of-the-box policies that can be applied to the branches on the server. Any changes being pushed to the server branches need to comply with these policies before the changes can be accepted. Policies are a great way to enforce your team's code quality and change-management standards. In this recipe, you'll learn how to configure branch policies on your master branch.

Getting ready

The out-of-the-box branch policies include several policies, such as build validation and enforcing a merge strategy. In this recipe, we'll only focus on the branch policies needed to set up a code-review workflow.

How to do it

1. Open the branches view for the myWebApp Git repository in the parts unlimited team portal. Select the master branch, and from the pull-down context menu choose Branch policies:

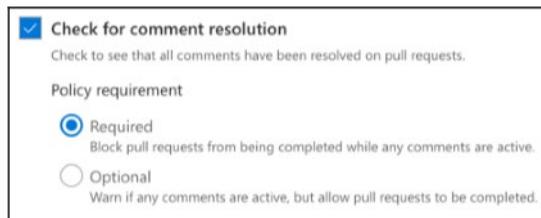


2. In the policies view, check the option to protect this branch:

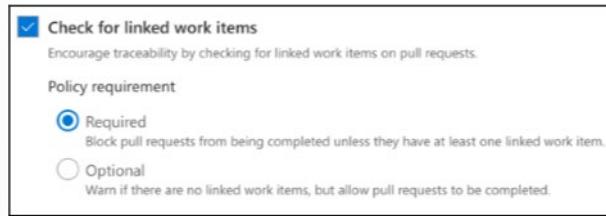
3. This presents the out-of-the-box policies. Check this option to select a minimum number of reviewers. Set the minimum number of reviewers to 1 and check the option to reset the code reviewer's votes when there are new changes:

The Allow users to approve their own changes option allows the submitter to self-approve their changes. This is OK for mature teams, where branch policies are used as a reminder for the checks that need to be performed by the individual.

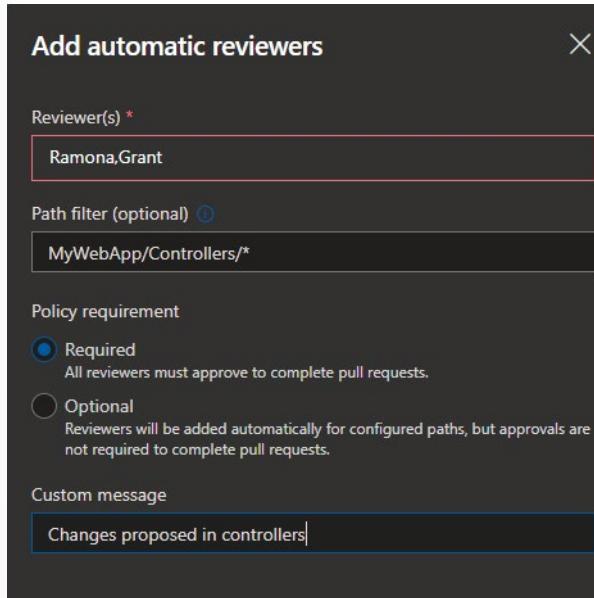
4. Use the review policy in conjunction with the comment-resolution policy. This allows you to enforce that the code review comments are resolved before the changes are accepted. The requester can take the feedback from the comment and create a new work item and resolve the changes, this at least guarantees that code review comments aren't just lost with the acceptance of the code into the master branch:



5. A code change in the team project is instigated by a requirement, if the work item that triggered the work isn't linked to the change, it becomes hard to understand why the changes were made over time. This is especially useful when reviewing the history of changes. Configure the Check for linked work items policy to block changes that don't have a work item linked to them:

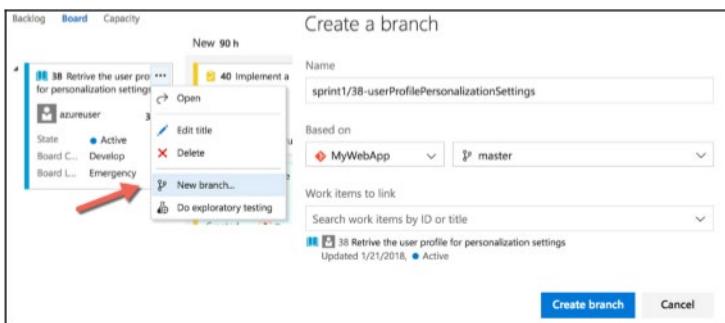


6. Select the option to automatically add code reviewers when a pull request is raised. You can map which reviewers are added based on the area of the code being changed:



How it works

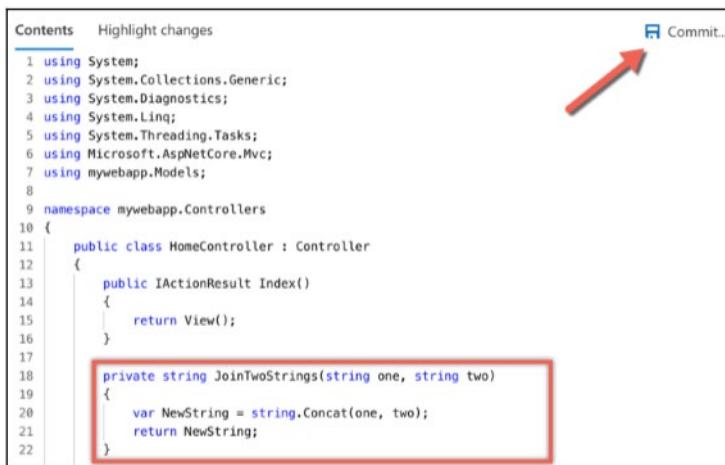
With the branch policies in place, the master branch is now fully protected. The only way to push changes to the master branch is by first making the changes in another branch and then raising a pull request to trigger the change-acceptance workflow. From one of the existing user stories in the work item hub, choose to create a new branch. By creating a new branch from a work item, that work item automatically gets linked to the branch, you can optionally include more than one work item with a branch as part of the create workflow:



Prefix in the name when creating the branch to make a folder for the branch to go in. In the preceding example, the branch will go in the folder. This is a great way to organise branches in busy environments.

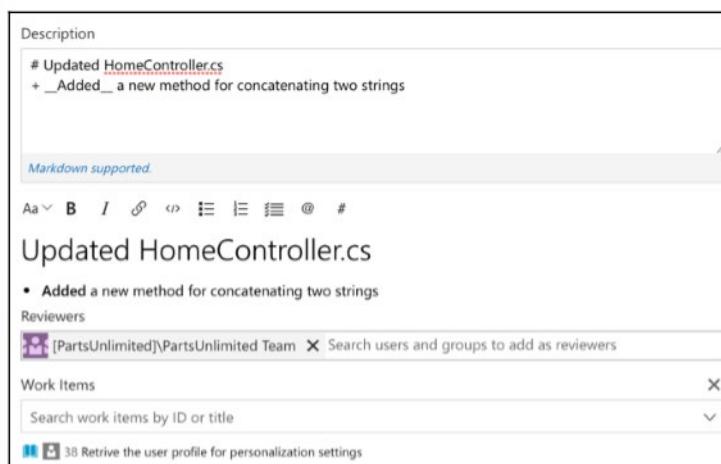
With the newly-created branch selected in the web portal, edit the `HomeController.cs` file to include the following code snippet and commit the changes to the branch. In the image below you'll see that after editing the file, you can directly commit the changes by clicking the commit button.

The file path control in team portal supports search. Start typing the file path to see all files in your Git repository under that directory starting with these letters show up in the file path search results dropdown.

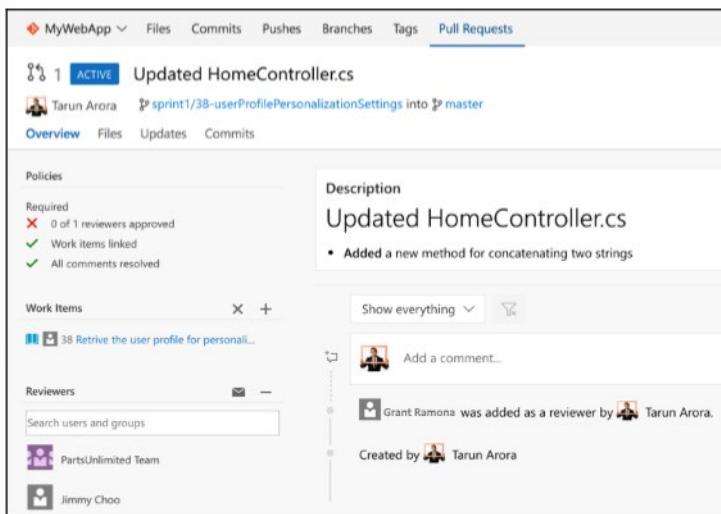


The code editor in web portal has several new features in Azure DevOps Server 2018, such as support for bracket matching and toggle white space. You can load the command palette by pressing . Among many other new options, you can now toggle the file using a file mini-map, collapse and expand, as well as other standard operations.

To push these changes from the new branch into the master branch, create a pull request from the pull request view. Select the new branch as the source and the master as the target branch. The new pull request form supports markdown, so you can add the description using the markdown syntax. The description window also supports @ mentions and # to link work items:



The pull request is created; the overview page summarizes the changes and the status of the policies. The Files tab shows you a list of changes along with the difference between the previous and the current versions. Any updates pushed to the code files will show up in the updates tab, and a list of all the commits is shown under the Commits tab:



Open the Files tab: this view supports code comments at the line level, file level, and overall. The comments support both @ for mentions and # to link work items, and the text supports markdown syntax:

The code comments are persisted in the pull request workflow; the code comments support multiple iterations of reviews and work well with nested responses. The reviewer policy allows for a code review workflow as part of the change acceptance. This is a great way for the team to collaborate on any code changes being pushed into the master branch. When the required number of reviewers approve the pull request, it can be completed. You can also mark the pull request to auto-complete after your review, this auto-completes the pull requests once all the policies have been successfully compiled to.

There's more

Have you ever been in a state where a branch has been accidentally deleted? It can be difficult to figure out what happened... Azure DevOps Server now supports searching for deleted branches. This helps you understand who deleted it and when, the interface also allows you to recreate the branch if you wish.

To cut out the noise from the search results, deleted branches are only shown if you search for them by their exact name. To search for a deleted branch, enter the full branch name into the branch search box. It will return any existing branches that match that text. You will also see an option to search for an exact match in the list of deleted branches. If a match is found, you will see who deleted it and when. You can also restore the branch. Restoring the branch will re-create it at the commit to which is last pointed. However, it will not restore policies and permissions.

Why Care About GitHooks

Why Care about GitHooks

Continuous delivery demands a significant level of automation... You can't be continuously delivering if you don't have a quality codebase. This is where git fares so well, it gives you the ability to automate most of the checks in your code base even before committing the code into your local repository let alone the remote.

GitHooks

GitHooks are a mechanism that allows arbitrary code to be run before, or after, certain Git lifecycle events occur. For example, one could have a hook into the commit-msg event to validate that the commit message structure follows the recommended format. The hooks can be any sort of executable code, including shell, PowerShell, Python, or any other scripts. Or they may be a binary executable. Anything goes! The only criteria is that hooks must be stored in the .git/hooks folder in the repo root, and that they must be named to match the corresponding events (as of Git 2.x):

- applypatch-msg
- pre-applypatch
- post-applypatch
- pre-commit
- prepare-commit-msg
- commit-msg
- post-commit
- pre-rebase
- post-checkout
- post-merge
- pre-receive
- update
- post-receive
- post-update
- pre-auto-gc
- post-rewrite
- pre-push

Practical use cases for using GitHooks

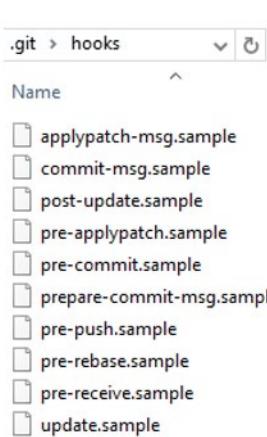
Since GitHooks simply execute the scripts on the specific event type they are called on, you can do pretty much anything with GitHooks. Some examples of where you can use hooks to enforce policies, ensure consistency, and control your environment...

- In Enforcing preconditions for merging
- Verifying work Item Id association in your commit message

- Preventing you & your team from committing faulty code
- Sending notifications to your team's chat room (Teams, Slack, HipChat, etc)

So where do I start?

Let's start by exploring client side GitHooks... Navigate to repo.git\hooks directory, you'll find that there a bunch of samples, but they are disabled by default. For instance, if you open that folder you'll find a file called pre-commit.sample. To enable it, just rename it to pre-commit by removing the .sample extension and make the script executable. When you attempt to commit using git commit, the script is found and executed. If your pre-commit script exits with a 0 (zero), you commit successfully, otherwise the commit fails.



Automation through Source Control...

"Using #Git hooks is like having little robot minions to carry out your every wish..."

GitHooks. Oh Windows!

Now if you are on Windows, simply renaming the file won't work. Git will fail to find shell in the designated path as specified in the script. The problem was lurking in the first line of the script, the shebang declaration:

```
#!/bin/sh
```

On Unix-like OS's, the #! tells the program loader that this is a script to be interpreted, and /bin/sh is the path to the interpreter you want to use, sh in this case. Windows is definitely not a Unix-like OS. Git for Windows supports Bash commands and shell scripts via Cygwin. By default, what does it find when it looks for sh.exe at /bin/sh? Yup, nothing; nothing at all. Fix it by providing the path to the sh executable on your system. I'm using the 64-bit version of Git for Windows, so my shebang line looks like this.

```
#!C:/Program\ Files/Git/usr/bin/sh.exe
```

PreCommit GitHook to scan commit for keywords

How can GitHooks help with security? You can invoke a script at pre-commit using GitHooks to scan the increment of code being committed into your local repository for specific keywords. Replace the code in this pre-commit shell file with the below code.

```
#!C:/Program\ Files/Git/usr/bin/sh.exe
matches=$(git diff-index --patch HEAD | grep '^+' | grep -Pi 'password|key-
```

```
word2|keyword3')
if [ ! -z "$matches" ]
then
    cat <<\EOT
Error: Words from the blacklist were present in the diff:
EOT
    echo $matches
    exit 1
fi
```

Of course, you don't have to build the full key word scan list in this script, you can branch off to a different file by referring it here that you could simply encrypt or scramble if you wanted to.

The repo.git\hooks folder is not committed into source control, so you may ask how do you share the goodness of the automated scripts you create with the team? The good news is that from Git version 2.9 you now have the ability to map githooks to a folder that can be committed into source control, you could do that by simply updating the global settings configuration for your git repository...

```
git config --global core.hooksPath '~/.githooks'
```

If you ever need to overwrite the GitHooks you have set up on the client side, you could do so by using the no-verify switch.

```
git commit --no-verify
```

Server side GitHooks with Azure Repos

So far we have looked at the client side GitHooks on Windows, Azure Repos also exposes server side hooks. Azure DevOps uses the same mechanism itself to create Pull requests... You can read more about it at [Server hooks event reference²](#).

Interested in learning more about GitHooks, check out another useful usecase for applying GitHooks at [GitHooks with PowerShell on Windows to automate source control operations³](#).

GitHooks in Action

Ryan Hellyer accidentally leaked his Amazon AWS access keys to GitHub and woke up to a \$6,000 bill the next morning. Wouldn't you just expect a source control as clever as Git to stop you from making such a blunder? Well, in case you didn't know, you can put Git Hooks to work to address not just this but many similar scenarios. In the spirit of pushing quality left into the development process, you want to enable developers to identify and catch code quality issues when they are developing the code locally in their repository, even before raising the pull request to trigger the branch policies. Git hooks allow you to run custom scripts whenever certain important events occur in the Git life cycle, such as committing, merging, and pushing. Git ships with a number of sample hook scripts in the repo.git\hooks directory. Since Git snares simply execute the contents on the particular occasion type they are approached, you can do practically anything with Git snares. Here are a few instances of where you can utilize snares to uphold arrangements, guarantee consistency, and control your environment:

- Enforcing preconditions for merging

² <https://docs.microsoft.com/en-gb/azure/devops/service-hooks/events?view=vsts#code-pushed>

³ <https://www.visualstudiologeeks.com/DevOps/UsingPowerShellForGitHooksWithVstsGitOnWindows>

- Verifying work Item ID association in your commit message Preventing you and your team from committing faulty code
- Sending notifications to your team's chatroom (Teams, Slack, HipChat)

In this recipe, we'll look at using the pre-commit Git hook to scan the commit for keywords from a predefined list to block the commit if it contains any of these keywords.

Getting ready

Let's start by exploring client-side Git hooks. Navigate to the `repo.git\hooks` directory – you'll find that there a bunch of samples, but they are disabled by default. For instance, if you open that folder, you'll find a file called `precommit.sample`. To enable it, just rename it to `pre-commit` by removing the `.sample` extension and make the script executable. When you attempt to commit using `git commit`, the script is found and executed. If your pre-commit script exits with a 0 (zero), you commit successfully, otherwise, the commit fails.

If you are using Windows, simply renaming the file won't work. Git will fail to find the shell in the designated path as specified in the script. The problem is lurking in the first line of the script, the shebang declaration:

```
#!/bin/sh
```

On Unix-like OSes, the `#!` tells the program loader that this is a script to be interpreted, and `/bin/sh` is the path to the interpreter you want to use, `sh` in this case. Windows is definitely not a Unix-like OS. Git for Windows supports Bash commands and shell scripts via Cygwin. By default, what does it find when it looks for `sh.exe` at `/bin/sh`? Yup, nothing; nothing at all. Fix it by providing the path to the `sh` executable on your system. I'm using the 64-bit version of Git for Windows, so my shebang line looks like this:

```
#!C:/Program\ Files/Git/usr/bin/sh.exe
```

How to do it

Let's go back to the example we started with, how could have Git hooks stopped Ryan Hellyer from accidentally leaking his Amazon AWS access keys to GitHub? You can invoke a script at pre-commit using Git hooks to scan the increment of code being committed into your local repository for specific keywords:

1. Replace the code in this pre-commit shell file with the following code

```
#!C:/Program\ Files/Git/usr/bin/sh.exe matches=$(git diff-index --patch HEAD | grep '^+' | grep -Pi 'password|keyword2|keyword3') if [ ! -z "$matches" ] then cat <<\EOT Error: Words from the blacklist were present in the diff: EOT echo $matches exit 1 fi
```

You don't have to build the full keyword scan list in this script, you can branch off to a different file by referring it here that you could simply encrypt or scramble if you wanted to.

How it works

In the script, Git `diff-index` is used to identify the code increment being committed. This increment is then compared against the list of specified keywords. If any matches are found, an error is raised to block the

commit; the script returns an error message with the list of matches. In this case, the pre-commit script doesn't return 0 (zero), which means the commit fails.

There's more

The repo.git\hooks folder is not committed into source control, so you may wonder how you share the goodness of the automated scripts you create with the team. The good news is that, from Git version 2.9, you now have the ability to map Git hooks to a folder that can be committed into source control. You could do that by simply updating the global settings configuration for your Git repository:

```
git config --global core.hooksPath '~/.githooks'
```

If you ever need to overwrite the Git hooks you have set up on the client side, you can do so by using the no-verify switch:

```
git commit --no-verify
```

Fostering Inner Source

Fostering Inner Source

The fork-based pull request workflow is popular with open source projects, since it allows anybody to contribute to a project. You don't need to be an existing contributor or have write access to a project to offer up your changes. This workflow isn't just for open source: forks also help support Inner Source workflows within your company.

Before forks, you've always been able to contribute to a project using Pull Requests. The workflow is simple enough: push a new branch up to your repository and then open a pull request to get a code review from your team and have Azure Repos evaluate your branch policies. When your code is approved, you can click one button to merge your pull request into master and deploy. This workflow is great for working on your projects with your team. But what if you notice a simple bug in a different project within your company and you want to fix it yourself? What if you want to add a feature to a project that you use but another team develops? That's where forks come in; forks are at the heart of **Inner Source practices**.

Inner Source – sometimes called “internal open source” – brings all the benefits of open source software development inside your firewall. It opens up your software development processes so that your developers can easily collaborate on projects across your company, using the same processes that are popular throughout the open source software communities. But it keeps your code safe and secure within your organization.

Microsoft uses the Inner Source approach heavily. As part of the efforts to standardize on one engineering system throughout the company – backed by Azure Repos – Microsoft has also opened up the source code to all our projects to everyone within the company.

Before the move to Inner Source, Microsoft was “siloed”: only engineers working on Windows could read the Windows source code. Only developers working on Office could look at the Office source code. So if you were an engineer working on Visual Studio and you thought that you had found a bug in Windows or Office – or you wanted to add a new feature – you were simply out of luck. But by moving to offer Inner Source throughout the company, powered by Azure Repos, it's easy to fork a repository to contribute back. As an individual making the change you don't need write access to the original repository, just the ability to read it and create a fork.

Implementing the Fork Workflow

As discussed in the fork workflow, a fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. Most commonly, forks are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own idea. A fork is a complete copy of a repository, including all files, commits, and (optionally) branches. Forks are a great way to support an Inner Source workflow: you can create a fork to suggest changes to a project when you don't have permissions to write to the original project directly. Once you're ready to share those changes, it's easy to contribute them back using pull requests.

What's in a fork

A fork starts with all the contents of its upstream (original) repository. When you create a fork, you can choose whether to include all branches or limit to only the default branch. None of the permissions, policies, or build pipelines are applied. The new fork acts as if someone cloned the original repository, then pushed to a new, empty repository. After a fork has been created, new files, folders, and branches are not shared between the repositories unless a Pull Request (PR) carries them along.

Sharing code between forks

You can create PRs in either direction: from fork to upstream, or upstream to fork. The most common direction will be from fork to upstream. The destination repository's permissions, policies, builds, and work items will apply to the PR.

Choosing between branches and forks

For a very small team (2-5 developers), we recommend working in a single repo. Everyone should work in a topic branches, and master should be protected with branch policies. As your team grows larger, you may find yourself outgrowing this arrangement and prefer to switch to a forking workflow.

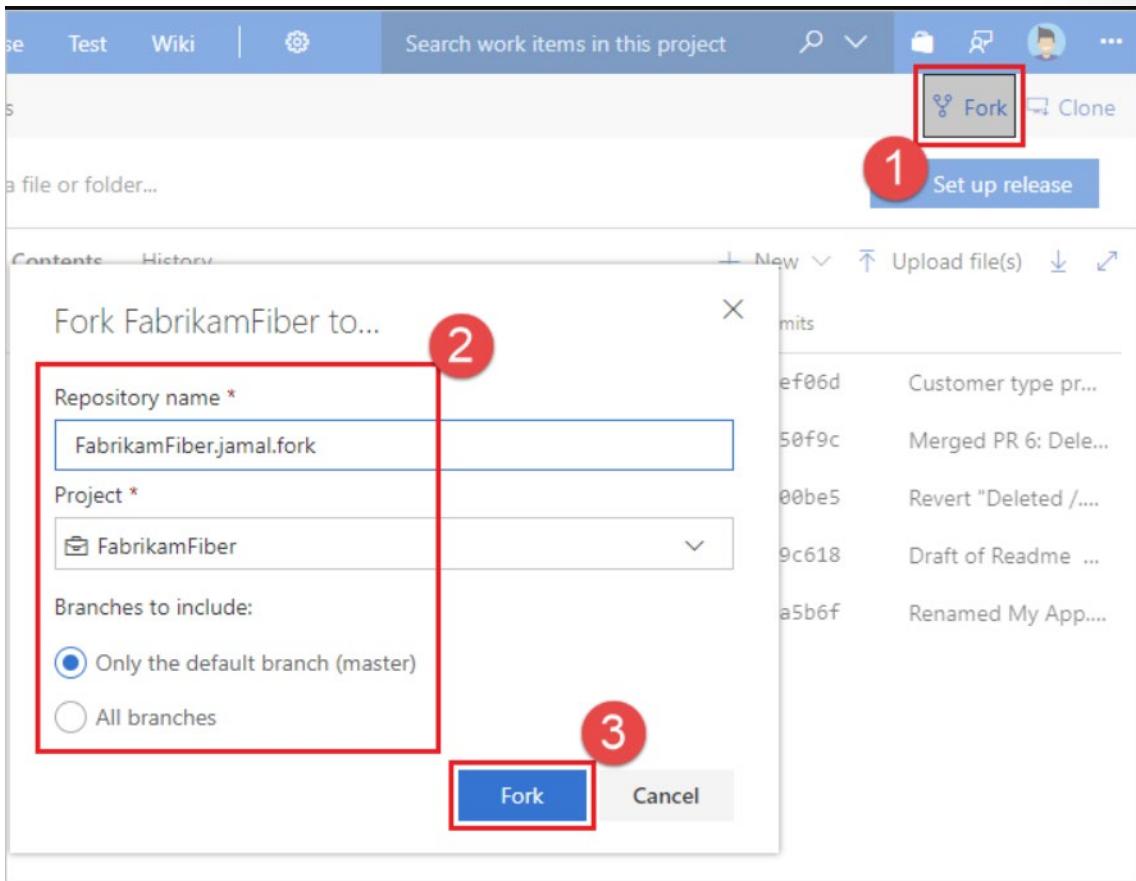
If your repository has a large number of casual or infrequent committers (similar to an open source project), we recommend the forking workflow. Typically only core contributors to your project have direct commit rights into your repository. You should ask collaborators from outside this core set of people to work from a fork of the repository. This will isolate their changes from yours until you've had a chance to vet the work.

The forking workflow

- Create a fork
- Clone it locally
- Make your changes locally and push them to a branch
- Create and complete a PR to upstream
- Sync your fork to the latest from upstream

Create the fork

1. Navigate to the repository to fork, and choose Fork.
2. Specify a name, and choose the project where you want the fork to be created. If the repository contains a lot of topic branches, we recommend you fork only the default branch.
3. Choose Fork to create the fork.



Note - You must have the Create Repository permission in your chosen project to create a fork. We recommend you create a dedicated project for forks where all contributors have the Create Repository permission. For an example of granting this permission, see Set Git repository permissions.

Clone your fork locally

Once your fork is ready, clone it using the command line or an IDE like Visual Studio. The fork will be your origin remote.

For convenience, after cloning you'll want to add the upstream repository (where you forked from) as a remote named upstream.

```
git remote add upstream {upstream_url}
```

Make and push changes

It's possible to work directly in master - after all, this fork is your personal copy of the repo. We recommend you still work in a topic branch, though. This allows you to maintain multiple, independent workstreams simultaneously. Also, it reduces confusion later when you want to sync changes into your fork.

Make and commit your changes as you normally would. When you're done with the changes, push them to origin (your fork).

Create and complete a PR

Open a pull request from your fork to the upstream. All the policies, required reviewers, and builds will be applied in the upstream repo. Once all policies are satisfied, the PR can be completed and the changes become a permanent part of the upstream repo.

The screenshot shows a pull request creation interface in a web-based Git repository. The top navigation bar includes links for Dashboards, Code, ..., Search work items in this project, and a dropdown for the repository. Below the navigation is a sub-navigation bar with links for FabrikamFiber.jamal.fork, Files, Commits, Pushes, Branches, Tags, and Pull Requests, where 'Pull Requests' is underlined. A large button labeled 'New Pull Request' is visible. The main form area has fields for the source branch ('FabrikamFiber.jamal.fork'), target branch ('users/jamal/date-fix'), destination repository ('FabrikamFiber'), and destination branch ('master'). The 'Title' field contains 'Added a new option to the settings page'. The 'Description' section contains two paragraphs of text and a list of checkboxes for delivery preferences. Below the description is a rich text editor toolbar. The 'Reviewers' section lists '[FabrikamFiber]\FabrikamFiber Team' and a search bar for adding more reviewers. The 'Work Items' section is partially visible at the bottom.

Important - Anyone with the Read permission can open a PR to upstream. If a PR build pipeline is configured, the build will run against the code introduced in the fork.

Sync your fork to latest

When you've gotten your PR accepted into upstream, you'll want to make sure your fork reflects the latest state of the repo. We recommend rebasing on upstream's master branch (assuming master is the main development branch).

```
git fetch upstream master  
git rebase upstream/master
```

```
git push origin
```

The forking workflow lets you isolate changes from the main repository until you're ready to integrate them. When you're ready, integrating code is as easy as completing a pull request.

Inner Source with Forks

People fork repositories when they want to change the code in a repository they don't have write access to. Clearly if you don't have write access, you really aren't part of the team contributing to that repository, so why would you want to modify the code repository? In our line of work, we tend to look for technical reasons to improve something. You may find a better way of implementing the solution or may simply want to enhance the functionality by contributing to or improving an existing feature. Personally, I fork repositories in the following situations:

- I want to make a change.
- I think the project is interesting and may want to use it in the future.
- I want to use some or all of the code in that repository as a starting point for my own project.

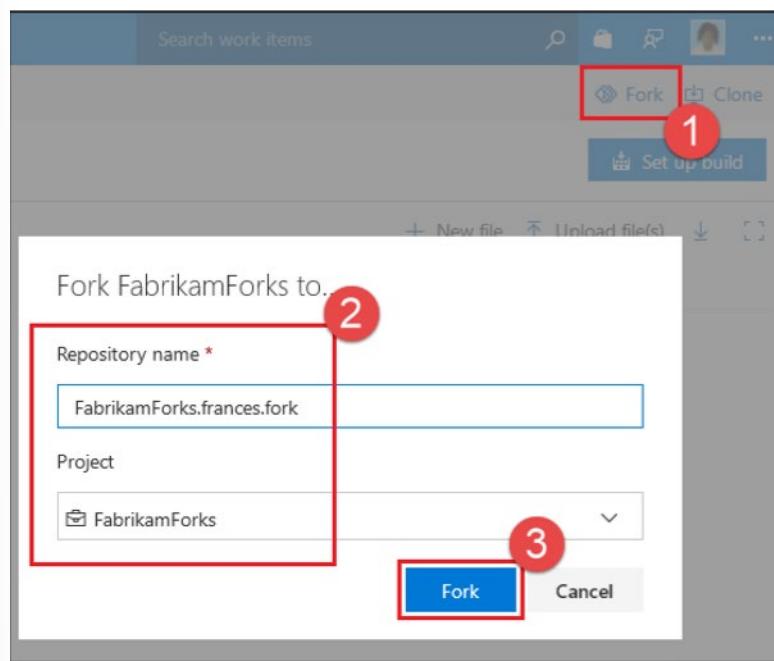
Software teams are encouraged to contribute to all projects internally, not just their own software projects. Forks are a great way to foster a culture of inner open source. Forks are a recent addition to the Azure DevOps Server-hosted Git repositories. In this recipe, we'll learn how to fork an existing repository and contribute changes back upstream via a pull request.

Getting ready

A fork starts with all the contents of its upstream (original) repository. When you create a fork in the Azure DevOps Server, you can choose whether to include all branches or limit to only the default branch. A fork doesn't copy the permissions, policies, or build definitions of the repository being forked. After a fork has been created, the newly-created files, folders, and branches are not shared between the repositories unless you start a pull request. Pull requests are supported in either direction: from fork to upstream, or upstream to fork. The most common direction for a pull request will be from fork to upstream.

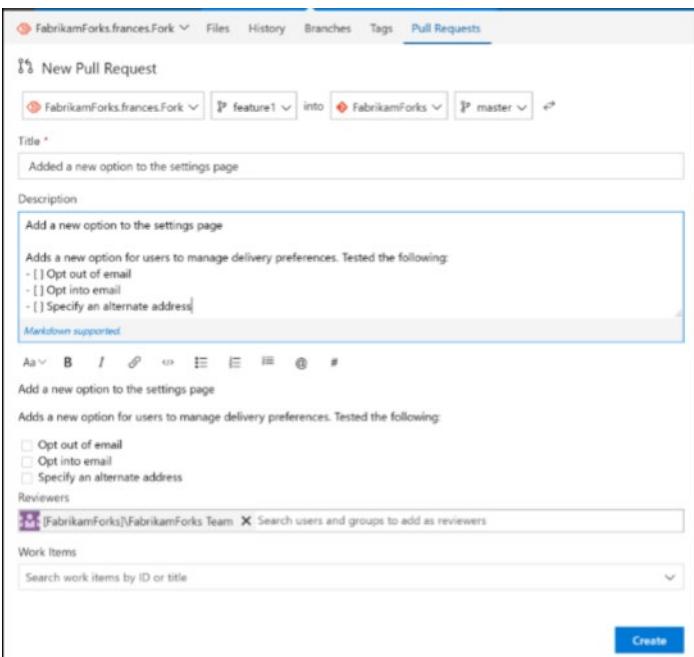
How to do it

1. Choose the Fork button (1), and then select the project where you want the fork to be created (2). Give your fork a name and choose the Fork button (3).



2. Once your fork is ready, clone it using the command line or an IDE, such as Visual Studio. The fork will be your origin remote. For convenience, you'll want to add the upstream repository (where you forked from) as a remote named upstream. On the command line, type the following:

```
git remote add upstream {upstream_url}
```
3. It's possible to work directly in the master – after all, this fork is your personal copy of the repo. We recommend you still work in a topic branch, though. This allows you to maintain multiple independent workstreams simultaneously. Also, it reduces confusion later when you want to sync changes into your fork. Make and commit your changes as you normally would. When you're done with the changes, push them to origin (your fork).
4. Open a pull request from your fork to the upstream. All the policies, required reviewers, and builds will be applied in the upstream repo. Once all the policies are satisfied, the PR can be completed and the changes become a permanent part of the upstream repo:



- When your PR is accepted into upstream, you'll want to make sure your fork reflects the latest state of the repo. We recommend rebasing on the upstream's master branch (assuming the master is the main development branch). On the command line, run the following:

```
git fetch upstream master
git rebase upstream/master
git push origin
```

How it works

The forking workflow lets you isolate changes from the main repository until you're ready to integrate them. When you're ready, integrating code is as easy as completing a pull request.

For more information, see:

- Clone an Existing Git repo⁴**
- Azure Repos Git Tutorial⁵**

⁴ <https://docs.microsoft.com/en-us/azure/devops/repos/git/clone?view=azure-devops&tabs=visual-studio>

⁵ <https://docs.microsoft.com/en-us/azure/devops/repos/git/gitworkflow?view=azure-devops>

Lab

Code Review with Pull Requests

In this lab, **Version Controlling with Git in Azure Repos**⁶, you will learn how to establish a local Git repository, which can easily be synchronized with a centralized Git repository in Azure DevOps. In addition, you will learn about Git branching and merging support.

- Exercise 6: Managing branches from Azure DevOps
- Exercise 7: Managing repositories

⁶ <https://www.azuredevopslabs.com/labs/azuredevops/git/>

Module Review and Takeaways

Module Review Questions

Multiple choice

Which repository type offers the benefits of reduced code complexity, effective code reviews, and sharing of common components?

- Multiple-repo
- Mono-repo

Checkbox

What are the three types of branching? Select three.

- Trunk-based development
- Toggle workflow
- Gitflow branching
- Forking workflow
- Straight branching

Suggested answer

What are GitHooks?

Suggested answer

What are some best practices when working with files in Git? What do you suggest for working with large files?

Answers

Multiple choice

Which repository type offers the benefits of reduced code complexity, effective code reviews, and sharing of common components?

Multiple-repo

Mono-repo

Checkbox

What are the three types of branching? Select three.

Trunk-based development

Toggle workflow

Gitflow branching

Forking workflow

Straight branching

What are GitHooks?

A mechanism that allows arbitrary code to be run before, or after, certain Git lifecycle events occur. Use GitHooks to enforce policies, ensure consistency, and control your environment. Can be either client-side or server-side.

What are some best practices when working with files in Git? What do you suggest for working with large files?

Best practices: use a package management system for DLLs, library files, and other dependent files, don't commit the binaries, logs, tracing output or diagnostic data from your builds, don't commit large, frequently updated binary assets, and use diffable plain text formats, such as JSON, for configuration information. For large files, use Git LFS.

Module 4 Consolidating Artifacts and Designing a Dependency Management Strategy

Module Overview

Module Overview

In this module, we will talk about managing dependencies in software development. We are going to cover what dependencies are and how to identify them in your codebase. Then you will learn how to package these dependencies and manage the packages in package feeds. Finally, you are going to learn about versioning strategies.

We will look at dependency management as a concept in software and why it is needed. We are going to look at dependency management strategies and how you can identify components in your source code and change these to dependencies.

Learning Objectives

After completing this module, students will be able to:

- Recommend artifact management tools and practices
- Abstract common packages to enable sharing and reuse
- Migrate and consolidate artifacts
- Migrate and integrate source control measures

Packaging Dependencies

What is dependency management?

Before we can understand dependency management, we will need to first get introduced to the concepts of dependencies.

Dependencies in software

Modern software development involves complex projects and solutions. Projects have dependencies on other projects and solutions are not single pieces of software. The solutions and software built consists of multiple parts and components and are often reused.

As codebases are expanding and evolving it needs to be componentized to be maintainable. A team that is writing software will not write every piece of code by itself, but leverage existing code written by other teams or companies and open source code that is readily available. Each component can have its own maintainers, speed of change and distribution, giving both the creators and consumers of the components autonomy.

A software engineer will need to identify the components that make up parts of the solution and decide whether to write the implementation or include an existing component. The latter approach introduces a dependency on other components.

Why is dependency management needed?

It is essential that the software dependencies that are introduced in a project and solution can be properly declared and resolved. You need to be able to manage the overall composition of project code and the included dependencies. Without proper dependency management it will be hard to keep the components in the solution controlled.

Management of dependencies allows a software engineer and team to be more efficient working with dependencies. With all dependencies being managed it is also possible to stay in control of the dependencies that are consumed, enabling governance and security scanning for use of packages with known vulnerabilities or exploits.

Elements of a dependency management strategy

There are a number of aspects for a dependency management strategy.

- **Standardization**

Managing dependencies benefits from a standardized way of declaring and resolving them in your codebase. Standardization allows a repeatable, predictable process and usage that can be automated as well.

- **Package formats and sources**

The distribution of dependencies can be performed by a packaging method suited for the type of dependency in your solution. Each dependency is packaged using its applicable format and stored in a centralized source. Your dependency management strategy should include the selection of package formats and corresponding sources where to store and retrieve packages.

- **Versioning**

Just like your own code and components, the dependencies in your solution usually evolve over time. While your codebase grows and changes, you need to take into account the changes in your depend-

encies as well. This requires a versioning mechanism for the dependencies so you can be selective of the particular version of a dependency you want to use.

Identifying dependencies

It starts with identifying the dependencies in your codebase and deciding which dependencies will be formalized.

Your software project and its solution probably already use dependencies. It is very common to use libraries and frameworks that are not written by yourself. Additionally, your existing codebase might have internal dependencies that are not treated as such. For example, take a piece of code that implements certain business domain model. It might be included as source code in your project, and also consumed in other projects and teams. You need to look into your codebase to identify pieces of code that can be considered dependencies to also treat them as such. This requires changes to how you organize your code and build the solution. It will bring your components.

Source and package componentization

Current development practices already have the notion of componentization. There are two ways of componentization commonly used.

1. Source componentization

The first way of componentization is focussed on source code. It refers to splitting up the source code in the codebase in separate parts and organizing it around the identified components. It works as long as the source code is not shared outside of the project. Once the components need to be shared, it requires distributing the source code or the produced binary artefacts that are created from it.

2. Package componentization

The second way uses packages. Distributing of software components is performed by means of packages as a formal way of wrapping and handling the components. A shift to packages adds characteristics needed for proper dependency management, like tracking and versioning of packages in your solution.

See also [Collaborate more and build faster with packages¹](#).

Decompose your system

Before you can change your codebase into separate components to prepare for finding dependencies that can be taken out of your system, you will need to get better insights in your code and solution. This allows you to decompose your system to individual components and dependencies.

The goal is to reduce the size of your own codebase and system, making it more efficient to build and manageable in the end. You achieve this by removing certain components of your solution. These are going to be centralized, reused and maintained independently. You will remove those components and externalizing them from your solution at the expense of introducing dependencies on other components.

This process of finding and externalizing components is effectively creating dependencies. It may require some refactoring, such as creating new solution artifacts for code organization, or code changes to cater for the unchanged code to take a dependency on an (external) component. You might need to introduce some code design patterns to isolate and include the componentized code. Examples of patterns are abstraction by interfaces, dependency injection and inversion of control.

Decomposing could also mean that you will replace your own implementation of reusable code with an available open source or commercial component.

¹ <https://docs.microsoft.com/en-us/azure/devops/artifacts/collaborate-with-packages?view=vsts>

Scanning your codebase for dependencies

There are a number of ways to identify the dependencies in your codebase. These include scanning your code for patterns and reuse, as well as analyzing how the solution is composed of individual modules and components.

- **Duplicate code**

When certain pieces of code appear in several places it is a good indication that this code can be reused. Keep in mind that code duplication is not necessarily a bad practice. However, if the code can be made available in a properly reusable way, it does have benefits over copying code and have to manage that. The first step to isolate these pieces of duplicate code is to centralize them in the codebase and componentize them in the appropriate way for the type of code.

- **High cohesion and low coupling**

A second approach is to find code that might define components in your solution. You will look for code elements that have a high cohesion to each other, and low coupling with other parts of code. This could be a certain object model with business logic, or code that is related because of its responsibility, such as a set of helper or utility code or perhaps a basis for other code to be built upon.

- **Individual lifecycle**

Related to the high cohesion you can look for parts of the code that have a similar lifecycle and can be deployed and released individually. If such code can be maintained by a team separate from the codebase that it is currently in, it is an indication that it could be a component outside of the solution.

- **Stable parts**

Some parts of your codebase might have a slow rate of change. That particular code is stable and is not altered often. You can check your code repository to find the code with a low change frequency.

- **Independent code and components**

Whenever code and components are independent and unrelated to other parts of the system, it can potentially be isolated to a separate component and dependency.

You can use a variety of tools to assist you in scanning and examining your codebase. These range from tools that scan for duplicate code, draw solution dependency graphs, to tools that can compute metrics for coupling and cohesion.

Package Management

Packages

Packages are used to define the components you rely and depend upon in your software solution. They provide a way to store those components in a well defined format with metadata to describe it.

What is a package?

A package is a formalized way of creating a distributable unit of software artifacts that can be consumed from another software solution. The package describes the content it contains and usually provides additional metadata. This additional information uniquely identifies the individual packages and to be self-descriptive. It helps to better store packages in centralized locations and consume the contents of the package in a predictable manner. In addition, it enables tooling to manage the packages in the software solution.

Types of packages

Packages can be used for a variety of components. The type of components you want to use in your codebase differ for the different parts and layers of the solution you are creating. These range from frontend components, such as JavaScript code files, to backend components like .NET assemblies or Java components, complete self-contained solutions or reusable files in general.

Over the past years the packaging formats have changed and evolved. At the moment there are a couple of de facto standard formats for packages.

Types of packages

Packages can be used for a variety of components. The type of components you want to use in your codebase differ for the different parts and layers of the solution you are creating. These range from frontend components, such as JavaScript code files, to backend components like .NET assemblies or Java components, complete self-contained solutions or reusable files in general.

Over the past years the packaging formats have changed and evolved. At the moment there are a couple of de facto standard formats for packages.

- **NuGet**

NuGet packages (pronounced “nugget”) is a standard used for .NET code artifacts. This includes .NET assemblies and related files, tooling and sometimes only metadata. NuGet defines the way packages are created, stored and consumed. A NuGet package is essentially a compressed folder structure with files in ZIP format and has the .nupkg extension.

See also [An introduction to NuGet²](#)

- **NPM**

An NPM package is used for JavaScript development. It originates from node.js development where it is the default packaging format. A NPM package is a file or folder that contains JavaScript files and a package.json file describing the metadata of the package. For node.js the package usually contains one or more modules that can be loaded once the package is consumed.

See also [About packages and modules³](#)

² <https://docs.microsoft.com/en-us/nuget/what-is-nuget>

³ <https://docs.npmjs.com/about-packages-and-modules>

- **Maven**

Maven is used for Java based projects. Each package has It has a Project Object Model file describing the metadata of the project and is the basic unit for defining a package and working with it.

- **PyPi**

The Python Package Index, abbreviated as PyPI and also known as the Cheese Shop, is the official third-party software repository for Python.

- **Docker**

Docker packages are called images and contain complete and self-contained deployments of components. Most commonly a Docker image represents a software component that can be hosted and executed by itself, without any dependencies on other images. Docker images are layered and might be dependent on other images as their basis. Such images are referred to as base images.

Package feeds

Packages should be stored in a centralized place for distribution and consumption by others to take dependencies on the components it contains. The centralized storage for packages is most commonly called a package feed. There are other names in use, such as repository or registry. We will refer to all of these as packages feeds, unless it is necessary to use the specific name for clarity.

Each package type has its own type of feed. Put another way, one feed typically contains one type of packages. There are NuGet feeds, NPM feeds, Maven repositories, PyPi feed and Docker registries.

Package feeds offer versioned storage of packages. A certain package can exist in multiple versions in the feed, catering for consumption of a specific version.

Private and public feeds

The package feeds are centralized and available for many different consumers. Depending on the package, its purpose and origin, it might be generally available or to a select audience. Typically open source projects for applications, libraries and frameworks are shared to everyone and publically available. The feeds can be exposed in public or private to distinguish in visibility. Public feeds can be consumed by anyone.

There might be reasons why you do not want your packages to be available publically. This could be because it contains intellectual property or does not make sense to share with other software developers. Components that are developed for internal use might be available only to the project, team or company that developed it.

In such cases you can still use packages for dependency management and choose to store the package in a private package feed.

Private feeds can only be consumed by those who are allowed access.

Package feed managers

Each of the package types has a corresponding manager that takes care of one or more of the following aspects of package management:

- Installation and removal of local packages
- Pushing packages to a remote package feed
- Consuming packages from a remote package feed
- Searching feeds for packages

The package manager have cross-platform command-line interface (CLI) tools to manage the local packages and feeds that host the packages. This CLI tooling is part of a local install on a development machine.

Choosing tools

The command-line nature of the tooling offers the ability to include it in scripts to automate the package management. Ideally, one should be able to use the tooling in build and release pipelines for component creating, publishing and consuming packages from feeds.

Additionally, developer tooling can have integrated support for working with package managers, providing an user interface for the raw tooling. Examples of such tooling are Visual Studio 2017, Visual Studio Code and Eclipse.

Package sources

The various package types have a standard source that is commonly used for public use. It is a go-to place for developers to find and consume publically available components as software dependencies. These sources are package feeds. We will discuss the public and private package sources available to give you a starting point for finding the most relevant feeds.

Public

In general you will find that publically available package sources are free to use. Sometimes they have a licensing or payment model for consuming individual packages or the feed itself.

These public sources can also be used to store packages you have created as part of your project. It does not have to be open source, although it is in most cases. Public and free package sources that offer feeds at no expense will usually require that you make the packages you store publically available as well.

Package type	Package source	URL
NuGet	NuGet Gallery	https://nuget.org
NPM	NPMjs	https://npmjs.org
Maven	Maven	https://search.maven.org
Docker	Docker Hub	https://hub.docker.com
Python	Python Package Index	https://pypi.org

The table above does not contain an extensive list of all public sources available. There are other public package sources for each of the types.

Private

Private feeds can be used in cases where packages should be available to a select audience.

The main difference between public and private feeds is the need for authentication. Public feeds can be anonymously accessible and optionally authenticated. Private feeds can be accessed only when authenticated.

There are two options for private feeds:

1. **Self-hosting**

Some of the package managers are also able to host a feed. Using on-premises or private cloud resources one can host the required solution to offer a private feed.

2. SaaS services

A variety of third party vendors and cloud providers offer software-as-a-service feeds that can be kept privately. This typically requires a consumption fee or cloud subscription.

The following table contains a non-exhaustive list of self-hosting options and SaaS offerings to privately host package feeds for each of the types covered.

Package type	Self-hosted private feed	SaaS private feed
NuGet	NuGet server	Azure Artifacts, MyGet
NPM	Sinopia, cnpmjs.org, Verdaccio	NPMjs.org, MyGet, Azure Artifacts
Maven	Nexus, Artifactory, Archivia	Azure Artifacts, Bintray, JitPack
Docker	Portus, Quay, Harbor	Docker Hub, Azure Container Registry, Amazon Elastic Container Registry
Python	PyPI Server	Gemfury

Consuming Packages

Each software project that consumes packages to include the required dependencies will need to use the package manager and one or more packages sources. The package manager will take care of downloading the individual packages from the sources and install them locally on the development machine or build server.

The developer flow will follow this general pattern:

1. Identify a required dependency in your codebase
2. Find a component that satisfies the requirements for the project
3. Search the package sources for a package offering a correct version of the component
4. Install the package into the codebase and development machine
5. Create the software implementation that uses the new components from the package.

The package manager tooling will facilitate searching and installing the components in the packages. How this is performed varies for the different package types. Refer to the documentation of the package manager for instructions on consuming packages from feeds.

To get started you will need to specify the package source to be used. Package managers will have a default source defined that refers to the standard package feed for its type. Alternative feeds will need to be configured to allow consuming the packages they offer.

Upstream sources

Part of the package management involves keeping track of the various sources. It is possible to refer to multiple sources from a single software solution. However, when combining private and public sources, the order of resolution of the sources becomes important.

One way to specify multiple packages sources is by choosing a primary source and specifying an upstream source. The package manager will evaluate the primary source first and switch to the upstream source when the package is not found there. The upstream source might be one of the official public sources or a private source. The upstream source could refer to another upstream source itself, creating a chain of sources.

A typical scenario is to use a private package source referring to an public upstream source for one of the

official feeds. This effectively enhances the packages in the upstream source with packages from the private feed, avoiding the need to publish private packages in a public feed.

A source that has an upstream source defined may download and cache the packages that were requested it does not contain itself. The source will include these downloaded packages and starts to act as a cache for the upstream source. It also offers the ability to keep track of any packages from the external upstream source.

An upstream source can be a way to avoid direct access of developer and build machines to external sources. The private feed uses the upstream source as a proxy to the otherwise external source. It will be your feed manager and private source that have the communication to the outside. Only privileged roles can add upstream sources to a private feed.

See also **Upstream sources⁴**.

Packages graph

A feed can have one or more upstream sources, which might be internal or external. Each of these can have additional upstream sources, creating a package graph of source. Such a graph can offer many possibilities for layering and indirection of origins of packages. This might fit well with multiple teams taking care of packages for frameworks and other base libraries.

The downside is that package graphs can become complex when not properly understood or designed. It is important to understand how you can create a proper package graph.

See also **Constructing a complete package graph⁵**.

Azure Artifacts

Previously you learned about packaging dependencies and the various packaging formats, feeds, sources and package managers. Now, you will learn more about package management and how to create a feed and publish packages to it. During this module NuGet and Azure Artifacts are used as an example of a package format and a particular type of package feed and source.

Microsoft Azure DevOps provides various features for application lifecycle management, including work item tracking, source code repositories, build and release pipelines and artifact management.

The artifact management is called Azure Artifacts and was previously known as Package management. It offers public and private feeds for software packages of various types.

Types of packages supported

Azure Artifacts currently supports feeds that can store 5 different package types:

1. NuGet packages
2. NPM packages
3. Maven
4. Universal packages
5. Python

⁴ <https://docs.microsoft.com/en-us/azure/devops/artifacts/concepts/upstream-sources>

⁵ <https://docs.microsoft.com/en-us/azure/devops/artifacts/concepts/package-graph>

Previously, we discussed the package types for NuGet, NPM, Maven and Python. Universal packages are an Azure Artifacts specific package type. In essence it is a versioned package containing multiple files and folders.

A single Azure Artifacts feed can contain any combination of such packages. You can connect to the feed using the package managers and the corresponding tooling for the package types. For Maven packages this can also be the Gradle build tool.

Selecting package sources

When creating your solution you will decide which packages you want to consume to offer the components you are dependent on. The next step is to determine the sources for these packages. The main choice is selecting public and private feeds or a combination of these.

Publicly available packages can usually be found in the public package sources. This would be nuget.org, npmjs and pypi.org. Your solution can select these sources if it only consumes packages available there.

Whenever your solution also has private packages, that are not or cannot be available on public sources, you will need to use a private feed.

In the previous module you learned that public package sources can be upstream sources to private feeds. Azure Artifacts allows its feeds to specify one or more upstream sources, which can be public or other private feeds.

Publishing packages

As software is developed and components are written, you will most likely also produce components as dependencies that can be packaged for reuse. Discussed previously was guidance to find components that can be isolated into dependencies. These components need to be managed and packaged. After that they can be published to a feed, allowing others to consume the packages and use the components it contains.

Creating a feed

The first step is to create a feed where the packages can be stored. In Azure Artifacts you can create multiple feeds, which are always private. During creation you can specify the name, visibility and whether to prepopulate the default public upstream sources for NuGet, NPM and Python packages.

What are feeds in Azure Artifacts?

Most package management system provide endpoints where you can request packages to install in your applications. Those endpoints are called feeds. In Azure Artifacts, you can have multiple feeds in your projects and you can make them available to only users authorized in your project or for your entire organization. Each feed can contain any types of packages, even mixed type, but it is recommended that you create one feed per type you want to support in your organization, this way it's clear what the feed contains. Each feed can contain one or more upstream and can manage its own security.

Controlling access

The Azure Artifacts feed you created is always private and not available publically. You need access to it by authenticating to Azure Artifacts with an account that has access to Azure DevOps and a team project.

By default a feed will be available to all registered users in Azure DevOps. You can select it to be visible only to the team project where the feed is created. Whichever option is chosen, you can change the permissions for a feed from the settings dialog.

Push packages to a feed

Once you have authenticated to Azure DevOps you are allowed to pull and push packages to the package feed, provided you have permissions to do so.

Pushing packages is done with the tooling for the package manager. Each of the package managers and tooling have different syntax for pushing.

To manually push a NuGet package you would use the NuGet.exe command-line tool. For a package called MyDemoPackage the command would resemble this:

```
nuget.exe push -Source {NuGet package source URL} -ApiKey YourKey YourPack-
age\YourPackage.nupkg
```

Updating packages

Packages might need to be updated during their lifetime. Technically, updating a package is performed by pushing a new version of the package to the feed. The package feed manager takes care of properly storing the updated package amongst the existing packages in the feed.

Please note that updating packages requires a versioning strategy. This will be covered later.

Demonstration Creating a Package Feed

Prerequisite: Open an existing Azure DevOps project

Steps to create a package feed

1. Go to dev.azure.com and open your team project

- Open Azure Artifacts
- Click on the button **New feed**
- Enter a name: PartsUnlimited
- Click **Create**

The feed is now created.

2. Go to **Settings** and click **Feed Settings**

You are in the tab Feed Details

3. Set a description: Packages for the PartsUnlimited Web application

- Click **Save**
- Open the Permissions tab

Under permissions you will find which roles are available under which users and groups and you will learn about this in a next module.

4. Open the Views tab

We will talk about this later in the module.

5. Open the **Upstream sources** tab

You can manage and add your upstream sources here.

6. Go back to the root of the feed by clicking on **PartsUnlimited**

- Click **Connect to Feed**

This dialog shows info about how packages that are stored in this feed can be consumed from various tools. You will learn more about consuming packages in a next demonstration.

Demonstration Pushing a Package

Prerequisite: Make sourcecode for PartsUnlimited available in your Azure DevOps repo

Steps to create a NuGet package

1. Go to Visual Studio and open your PartsUnlimited project

Here is a ASP.NET MVC application, where security related parts have been isolated into a separate .NET standard project. Isolating gives you the option to create a package for this code and publish it to a feed. .NET standard has support for creating Nuget packages.

2. Right-click PartsUnlimited.Security project and select **Properties**

Under the tab Package you can set the version and package id.

3. Build the project PartsUnlimited.Security

4. Go to Command prompt, use command `dir bin\debug`

Here you see the .nupkg file for PartsUnlimited.Security

5. Open your team project in dev.azure.com and go to Artifacts

- Click **Connect to feed**

- Follow the dialog instructions

- Copy the command for "Add this feed" by clicking the **Copy** icon.

- Switch back to your commandline

- Look at the existing NuGet sources with command: `nuget sources`

You see two NuGet sources available now.

- Paste and run the copied instructions

- Look at the existing NuGet sources again with command: `nuget sources`

You see a third NuGet source available.

Steps to publish the package

1. Go back to the dialog instructions

2. Copy the command for "Push a Package" by clicking the **Copy** icon.

3. Switch back to your commandline and paste the copied instructions

* Change the folder and name `my_package.nupkg` to `bin\debug\PartsUnlimited.Security1.0.0.nupkg`

* Run the command

We have published the package to the feed and is pushed successfully.

4. Check if the package is available in Azure DevOps Artifacts
 - * Close the dialog instructions
 - * Refresh the Artifacts page

You see the successfully published package.

Demonstration Promoting a Package

Prerequisite: Have access to an existing Azure DevOps project and the connected package feed from the previous demo

Steps to demonstrate the views that exist on package feeds in Azure Artifacts

1. Go to dev.azure.com and open your team project
2. Open **Artifacts** and select the feed **PartsUnlimited.Security**
3. Go to **Settings** and click **Feed Settings**
4. Open the **Views** tab. By default there will be three views. Local: includes all packages in the feed and all cached from upstream sources. Prerelease and Release. In the **Default view** column is a checkmark behind Local. This is the default view that will always be used.

Steps to use the release view instead

1. Open Visual Studio and open NuGet Package Manager
2. Click the settings wheel and check the source address for the PartsUnlimited feed. *If you want to use a different view than the local view, you need to include that in the Source url of your feed, by adding @Release.*
3. Add @Release to the source url .../PartsUnlimited@Release/nuget/... and click **Update**
4. **Refresh** the Browse tab. You will see there are **No packages found** in the Release feed. Before any packages will appear, you need to promote them.

Steps to promote packages to particular views

1. Go back to your feed in Azure Artifacts
2. Click on the created NuGet Package **PartsUnlimited.Security**
3. Click **Promote**
4. Select the feed you want to use, in this case **PartsUnlimited@Release** and Promote.
5. Go back to the **Overview**. If we look again at our package in the feed, you will notice there is now a Release tag associated with this particular package.
6. Go back to Visual Studio, **NuGet Package Manager**
7. **Refresh** the Browse tab. You will see that your version is promoted to this view.

8. Select **PartsUnlimited.Security** and click **Update** and **OK**. The latest version of this package is now used.

Migrating and Consolidating Artifacts

Identifying Existing Artifact Repositories

An **artifact** is a deployable component of your application. Azure Pipelines can work with a wide variety of artifact's sources and repositories. When you're creating a release pipeline, you need to link the required artifact sources. Often, this will represent the output of a build pipeline from a continuous integration system like Azure pipelines, Jenkins, or TeamCity. The artifacts that you produce might be stored in source control, like Git or Team Foundation version control. But you might also be using package management tools when you get repositories. When you need to create a release though, you need to specify which version of the artifacts are required. By default, the release pipeline will choose the latest version of the artifacts. But you might not want that. For example, you might need to choose a specific branch, a specific build version, or perhaps you need to specify tags. Azure artifacts is one of the services that's part of Azure DevOps. Using it can eliminate the need to manage file shares or host private package service. It lets you share code easily by letting you store Maven, npm, or NuGet packages together, cloud hosted, indexed, and matched. Now, while we can do so, there's also no need to store your binaries in Git. You can store them directly using universal packages. This is also a great way to protect your packages. Azure artifacts provides universal artifact management from Maven, npm, and NuGet. As well as sharing packages though, you can then easily access all of your artifacts in builds and releases because it integrates naturally with Azure pipelines and it's CI/CD tooling, along with versioning and testing.

Migrating and Integrating Artifact Repositories

While you can continue to work with your existing artifact repositories in their current locations when using Azure Artifacts, there are advantages to migrating them.

NuGet and Other Packages

Azure Artifacts provides hosted NuGet feeds as a service. By using this service, you can often eliminate the dependencies on on-premises resources such as file shares and locally hosted instances of NuGet. Server. The feeds can also be consumed by any Continuous Integration system that supports authenticated NuGet feeds.

Walkthroughs

For details on how to integrate NuGet, npm, Maven, Python, and Universal Feeds, see the following walkthroughs:

Get started with NuGet packages in Azure DevOps Services and TFS⁶

Use npm to store JavaScript packages in Azure DevOps Services or TFS⁷

Get started with Maven packages in Azure DevOps Services and TFS⁸

Get started with Python packages in Azure Artifacts⁹

Publish and then download a Universal Package¹⁰

⁶ <https://docs.microsoft.com/en-us/azure/devops/artifacts/get-started-nuget?view=vsts&tabs=new-nav>

⁷ <https://docs.microsoft.com/en-us/azure/devops/artifacts/get-started-npm?view=vsts&tabs=new-nav%2Cwindows>

⁸ <https://docs.microsoft.com/en-us/azure/devops/artifacts/get-started-maven?view=vsts&tabs=new-nav>

⁹ <https://docs.microsoft.com/en-us/azure/devops/artifacts/quickstarts/python-packages?view=vsts&tabs=new-nav>

¹⁰ <https://docs.microsoft.com/en-us/azure/devops/artifacts/quickstarts/universal-packages?view=vsts&tabs=azuredevops>

Lab

Lab: Updating Packages

In this lab, **Package Management with Azure Artifacts¹¹**, you will:

- Create a package feed
- Connect to the feed
- Create a NuGet package and publish it to the feed
- Import the new NuGet package into an existing project
- Update a NuGet package in the feed

¹¹ <https://www.azuredevopslabs.com/labs/azuredevops/packagemanagement>

Module Review and Takeaways

Module Review Questions

Multiple choice

If you are creating a feed that will allow yourself and those that you invite to publish, what visibility should you choose?

- public
- private

Multiple choice

Can you create a package feed for Maven in Azure Artifacts?

- Yes
- No

Multiple choice

What type of package should you use for Machine learning training data and models?

- NuGet
- NPM
- Maven
- Universal
- Python

Suggested answer

If an existing package is found to be broken or buggy, how should it be fixed?

Suggested answer

What is meant by saying that a package should be immutable?

Answers

Multiple choice

If you are creating a feed that will allow yourself and those that you invite to publish, what visibility should you choose?

- public
- private

Multiple choice

Can you create a package feed for Maven in Azure Artifacts?

- Yes
- No

Multiple choice

What type of package should you use for Machine learning training data and models?

- NuGet
- NPM
- Maven
- Universal
- Python

If an existing package is found to be broken or buggy, how should it be fixed?

Publish a new version

What is meant by saying that a package should be immutable?

A published version should never be changed, only replaced by a later version

Module 5 Implementing Continuous Integration with Azure Pipelines

Module Overview

Module Overview

Continuous Integration is one of the key pillars of DevOps. Once you have your code in a version control system you need an automated way of integrating the code on an ongoing basis. Azure Pipelines is a fully featured cross platform CI and CD service. It works with your preferred Git provider and can deploy to most major cloud services, which include Azure services. Azure DevOps offers a comprehensive Pipelines offering.

Learning Objectives

After completing this module, students will be able to:

- Implement and manage build infrastructure
- Explain why continuous integration matters
- Implement continuous integration using Azure DevOps

The Concept of Pipelines in DevOps Azure Pipelines

The Concept of Pipelines in DevOps

The business demands continuous delivery of value, and that value is created only when a product is delivered to a satisfied customer. It's not created when one silo in the process is completed. This demands that you reset focus from silos to an end-to-end flow of value.

The core idea is to create a repeatable, reliable and incrementally improving process for taking software from concept to customer. The goal of is to enable a constant flow of changes into production via an automated software production line. Think of this as a pipeline...

The pipeline breaks down the software delivery process into stages. Each stage is aimed at verifying the quality of new features from a different angle to validate the new functionality and prevent errors from affecting your users. The pipeline should provide feedback to the team and visibility into the flow of changes to everyone involved in delivering the new feature(s).

A delivery pipeline enables the flow of smaller changes more frequently, with a focus on flow. Your teams can concentrate on optimizing the delivery of changes that bring quantifiable value to the business. This approach leads teams to continuously monitor and learn where they are encountering obstacles, resolve those issues, and gradually improve the flow of the pipeline. As the process continues, the feedback loop provides new insights into new issues and obstacles to be resolved. The pipeline is the focus of your continuous improvement loop.

A typical pipeline will include the following stages: build automation and continuous integration; test automation; and deployment automation.

Build automation and Continuous Integration

The pipeline starts by building the binaries to create the deliverables that will be passed to the subsequent stages. New features implemented by the developers are integrated into the central code base on a continuous basis, built and unit tested. This is the most direct feedback cycle that informs the development team about the health of their application code.

Test Automation

Throughout this stage, the new version of an application is rigorously tested to ensure that it meets all desired system qualities. It is important that all relevant aspects — whether functionality, security, performance or compliance — are verified by the pipeline. The stage may involve different types of automated or (initially, at least) manual activities.

Deployment Automation

A deployment is required every time the application is installed in an environment for testing, but the most critical moment for deployment automation is rollout time. Since the preceding stages have verified the overall quality of the system, this is a low-risk step. The deployment can be staged, with the new version being initially released to a subset of the production environment and monitored before being completely rolled out. The deployment is automated, allowing for the reliable delivery of new functionality to users within minutes, if needed.

Your Pipeline Needs Platform Provisioning and Configuration Management

The deployment pipeline is supported by platform provisioning and system configuration management, which allow teams to create, maintain and tear down complete environments automatically or at the push of a button.

Automated platform provisioning ensures that your candidate applications are deployed to, and tests carried out against, correctly configured and reproducible environments. It also facilitates horizontal scalability and allows the business to try out new products in a sandbox environment at any time.

Orchestrating it all: Release and Pipeline Orchestration

The multiple stages in a deployment pipeline involve different groups of people collaborating and supervising the release of the new version of your application. Release and pipeline orchestration provides a top-level view of the entire pipeline, allowing you to define and control the stages and gain insight into the overall software delivery process.

By carrying out value stream mappings on your releases, you can highlight any remaining inefficiencies and hot spots, and pinpoint opportunities to improve your pipeline.

These automated pipelines need infrastructure to run on, the efficiency of this infrastructure will have a direct impact on the effectiveness of the pipeline.

Azure Pipelines

Azure Pipelines

Azure Pipelines is a cloud service that you can use to automatically build and test your code project and make it available to other users. It works with just about any language or project type. Azure Pipelines combines continuous integration (CI) and continuous delivery (CD) to constantly and consistently test and build your code and ship it to any target.

Does Azure Pipelines work with my language and tools?

Azure Pipelines is a fully featured cross platform CI and CD service. It works with your preferred Git provider and can deploy to most major cloud services, which include Azure services.

Languages

You can use many languages with Azure Pipelines, such as Python, Java, PHP, Ruby, C#, and Go.

Version control systems

Before you use continuous integration and continuous delivery practices for your applications, you must have your source code in a version control system. Azure Pipelines integrates with GitHub, GitLab, Azure Repos, Bitbucket, and Subversion.

Application types

You can use Azure Pipelines with most application types, such as Java, JavaScript, Python, .NET, PHP, Go, XCode, and C++.

Deployment targets

Use Azure Pipelines to deploy your code to multiple targets. Targets include container registries, virtual machines, Azure services, or any on-premises or cloud target such as Microsoft Azure, Google Cloud, or Amazon cloud services.

Package formats

To produce packages that can be consumed by others, you can publish NuGet, npm, or Maven packages to the built-in package management repository in Azure Pipelines. You also can use any other package management repository of your choice.

Why should I use CI and CD and Azure Pipelines?

Implementing CI and CD pipelines helps to ensure consistent and quality code that's readily available to users.

Azure Pipelines is a quick, easy, and safe way to automate building your projects and making them available to users.

Use CI and CD for your project

Continuous integration is used to automate tests and builds for your project. CI helps to catch bugs or issues early in the development cycle, when they're easier and faster to fix. Items known as artifacts are produced from CI systems. They're used by the continuous delivery release pipelines to drive automatic deployments.

Continuous delivery is used to automatically deploy and test code in multiple stages to help drive quality. Continuous integration systems produce deployable artifacts, which includes infrastructure and apps. Automated release pipelines consume these artifacts to release new versions and fixes to the target of your choice.

Continuous integration (CI)	Continuous delivery (CD)
Increase code coverage.	Automatically deploy code to production.
Build faster by splitting test and build runs.	Ensure deployment targets have latest code.
Automatically ensure you don't ship broken code.	Use tested code from CI process.
Run tests continually.	

Use Azure Pipelines for CI and CD

There are several reasons to use Azure Pipelines for your CI and CD solution. You can use it to:

- Work with any language or platform.
- Deploy to different types of targets at the same time.
- Integrate with Azure deployments.
- Build on Windows, Linux, or Mac machines.
- Integrate with GitHub.
- Work with open-source projects.

Azure Key Terms

Understanding the basic terms and parts of Azure Pipelines helps you further explore how it can help you deliver better code more efficiently and reliably.

Agent

When your build or deployment runs, the system begins one or more jobs. An agent is installable software that runs one build or deployment job at a time.

Artifact

An artifact is a collection of files or packages published by a build. Artifacts are made available to subsequent tasks, such as distribution or deployment.

Build

A build represents one execution of a pipeline. It collects the logs associated with running the steps and the results of running tests.

Continuous delivery

Continuous delivery (CD) is a process by which code is built, tested, and deployed to one or more test and production stages. Deploying and testing in multiple stages helps drive quality. Continuous integration systems produce deployable artifacts, which includes infrastructure and apps. Automated release pipelines consume these artifacts to release new versions and fixes to existing systems. Monitoring and alerting systems run constantly to drive visibility into the entire CD process. This process ensures that errors are caught often and early.

Continuous integration

Continuous integration (CI) is the practice used by development teams to simplify the testing and building of code. CI helps to catch bugs or problems early in the development cycle, which makes them easier and faster to fix. Automated tests and builds are run as part of the CI process. The process can run on a set schedule, whenever code is pushed, or both. Items known as artifacts are produced from CI systems. They're used by the continuous delivery release pipelines to drive automatic deployments.

Deployment target

A deployment target is a virtual machine, container, web app, or any service that's used to host the application being developed. A pipeline might deploy the app to one or more deployment targets after build is completed and tests are run.

Job

A build contains one or more jobs. Each job runs on an agent. A job represents an execution boundary of a set of steps. All of the steps run together on the same agent. For example, you might build two configurations - x86 and x64. In this case, you have one build and two jobs.

Pipeline

A pipeline defines the continuous integration and deployment process for your app. It's made up of steps called tasks. It can be thought of as a script that defines how your test, build, and deployment steps are run.

Release

When you use the visual designer, you create a release pipeline in addition to a build pipeline. A release is the term used to describe one execution of a release pipeline. It's made up of deployments to multiple stages.

Task

A task is the building block of a pipeline. For example, a build pipeline might consist of build tasks and test tasks. A release pipeline consists of deployment tasks. Each task runs a specific job in the pipeline.

Trigger

A trigger is something that's set up to tell the pipeline when to run. You can configure a pipeline to run upon a push to a repository, at scheduled times, or upon the completion of another build. All of these actions are known as triggers.

Evaluate Use of Hosted vs Private Agents

Microsoft vs Self-Hosted Agents

To build your code or deploy your software you need at least one agent. As you add more code and people, you'll eventually need more. When your build or deployment runs, the system begins one or more jobs. An agent is installable software that runs one build or deployment job at a time.

Microsoft-hosted agents

If your pipelines are in Azure Pipelines, then you've got a convenient option to build and deploy using a Microsoft-hosted agent. With Microsoft-hosted agents, maintenance and upgrades are taken care of for you. Each time you run a pipeline, you get a fresh virtual machine. The virtual machine is discarded after one use.

For many teams this is the simplest way to build and deploy. You can try it first and see if it works for your build or deployment. If not, you can use a self-hosted agent.

Self-hosted agents

An agent that you set up and manage on your own to run build and deployment jobs is a self-hosted agent. You can use self-hosted agents in Azure Pipelines. Self-hosted agents give you more control to install dependent software needed for your builds and deployments.

You can install the agent on Linux, macOS, or Windows machines. You can also install an agent on a Linux Docker container. After you've installed the agent on a machine, you can install any other software on that machine as required by your build or deployment jobs.

Agent Pools

Agent Pools

Instead of managing each agent individually, you organize agents into agent pools. An agent pool defines the sharing boundary for all agents in that pool. In Azure Pipelines, agent pools are scoped to the Azure DevOps organization; so you can share an agent pool across projects.

A project agent pool provides access to an organization agent pool. When you create a build or release pipeline, you specify which pool it uses. Pools are scoped to your project so you can only use them across build and release pipelines within a project.

To share an agent pool with multiple projects, in each of those projects, you create a project agent pool pointing to an organization agent pool. While multiple pools across projects can use the same organization agent pool, multiple pools within a project cannot use the same organization agent pool. Also, each project agent pool can use only one organization agent pool.

Default Agent Pools

The following organization agent pools are provided by default:

- Default pool: Use it to register **self-hosted agents**¹ that you've set up.
- Hosted VS2019 pool (Azure Pipelines only): The Hosted VS2019 pool is another built-in pool in Azure Pipelines. Machines in this pool have Visual Studio 2019 installed on Windows Server 2019 operating system.
- Hosted VS2017 pool (Azure Pipelines only): The Hosted VS2017 pool is another built-in pool in Azure Pipelines. Machines in this pool have Visual Studio 2017 installed on Windows Server 2016 operating system.
- Hosted Ubuntu 2004 pool (Azure Pipelines only)
- Hosted Ubuntu 1804 pool (Azure Pipelines only)
- Hosted Ubuntu 1604 pool (Azure Pipelines only): Enables you to build and release on Linux machines without having to configure a self-hosted Linux agent. Agents in this pool do not run in a container, but the Docker tools are available for you to use if you want to run **container jobs**².
- Hosted macOS 1014 pool (Azure Pipelines only)
- Hosted macOS 1015 pool (Azure Pipelines only)

Each of these Microsoft-hosted organization agent pools is exposed to each project through a corresponding project agent pool. By default, all contributors in a project are members of the User role on each hosted pool. This allows every contributor in a project to author and run build and release pipelines using Microsoft-hosted pools.

Pools are used to run jobs. Learn about **specifying pools for jobs**³.

Note: For the most up-to-date list of Agent Pools and the complete list of software installed on these machines, see **Microsoft-hosted agents**⁴.

¹ <https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/agents?view=vsts>

² <https://docs.microsoft.com/en-us/azure/devops/pipelines/process/container-phases?view=vsts&tabs=yaml>

³ <https://docs.microsoft.com/en-us/azure/devops/pipelines/process/phases?view=vsts&tabs=yaml>

⁴ <https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/hosted?view=azure-devops&tabs=yaml>

Typical Situations for Agent Pools

If you've got a lot of agents intended for different teams or purposes, you might want to create additional pools as explained below.

Creating agent pools

Here are some typical situations when you might want to create agent pools:

- You're a member of a project and you want to use a set of machines owned by your team for running build and deployment jobs. First, make sure you're a member of a group in All Pools with the Administrator role. Next create a New project agent pool in your project settings and select the option to Create a new organization agent pool. As a result, both an organization and project-level agent pool will be created. Finally install and configure agents to be part of that agent pool.
- You're a member of the infrastructure team and would like to set up a pool of agents for use in all projects. First make sure you're a member of a group in All Pools with the Administrator role. Next create a New organization agent pool in your admin settings and select the option to Auto-provision corresponding project agent pools in all projects while creating the pool. This setting ensures all projects have a pool pointing to the organization agent pool. The system creates a pool for existing projects, and in the future it will do so whenever a new project is created. Finally install and configure agents to be part of that agent pool.
- You want to share a set of agent machines with multiple projects, but not all of them. First create a project agent pool in one of the projects and select the option to Create a new organization agent pool while creating that pool. Next, go to each of the other projects, and create a pool in each of them while selecting the option to Use an existing organization agent pool. Finally, install and configure agents to be part of the shared agent pool.

Security of agent pools

Understanding how security works for agent pools helps you control sharing and use of agents.

Azure Pipelines

In Azure Pipelines, roles are defined on each agent pool, and membership in these roles governs what operations you can perform on an agent pool.

Role on an organization agent pool	Purpose
Reader	Members of this role can view the organization agent pool as well as agents. You typically use this to add operators that are responsible for monitoring the agents and their health.
Service Account	Members of this role can use the organization agent pool to create a project agent pool in a project. If you follow the guidelines above for creating new project agent pools, you typically do not have to add any members here.

Role on an organization agent pool	Purpose
Administrator	In addition to all the above permissions, members of this role can register or unregister agents from the organization agent pool. They can also refer to the organization agent pool when creating a project agent pool in a project. Finally, they can also manage membership for all roles of the organization agent pool. The user that created the organization agent pool is automatically added to the Administrator role for that pool.

The All agent pools node in the Agent Pools tab is used to control the security of all organization agent pools. Role memberships for individual organization agent pools are automatically inherited from those of the 'All agent pools' node.

Roles are also defined on each organization agent pool, and memberships in these roles govern what operations you can perform on an agent pool.

Role on a project agent pool	Purpose
Reader	Members of this role can view the project agent pool. You typically use this to add operators that are responsible for monitoring the build and deployment jobs in that project agent pool.
User	Members of this role can use the project agent pool when authoring build or release pipelines.
Administrator	In addition to all the above operations, members of this role can manage membership for all roles of the project agent pool. The user that created the pool is automatically added to the Administrator role for that pool.

The All agent pools node in the Agent pools tab is used to control the security of all project agent pools in a project. Role memberships for individual project agent pools are automatically inherited from those of the 'All agent pools' node. By default, the following groups are added to the Administrator role of 'All agent pools': Build Administrators, Release Administrators, Project Administrators.

Pipelines and Concurrency

Microsoft-Hosted vs Self-Hosted

One Parallel job in Azure Pipeline will let you run a single Build or Release job at any given time. This rule is true whether you run the job on Microsoft hosted or self hosted agents.

Microsoft-hosted CI/CD

If you want to run your builds and releases on machines that Microsoft manages, use Microsoft-hosted parallel jobs. Your jobs run on the pool of hosted agents.

Microsoft provides a free tier of service by default for every organization:

- Public project: 10 free Microsoft-hosted parallel jobs that can run for up to 360 minutes (6 hours) each time, with no overall time limit per month.
- Private project: One free parallel job that can run for up to 30 minutes each time, until you've used 1,800 minutes (30 hours) per month.

When the free tier is no longer sufficient:

- Public project: Contact Microsoft to get your free tier limits increased.
- Private project: You can pay for additional capacity per parallel job. Paid parallel jobs remove the monthly time limit and allow you to run each job for up to 360 minutes (6 hours). Buy Microsoft-hosted parallel jobs.

Self-hosted CI/CD

If you want Azure Pipelines to orchestrate your builds and releases, but use your own machines to run them, use self-hosted parallel jobs. You start by deploying agents on your machines. You can register any number of these self-hosted agents in your organization. Microsoft charges based on the number of jobs you want to run at a time, not the number of agents registered.

Microsoft provide a free tier of service by default in every organization:

- Public project: 10 free self-hosted parallel jobs.
- Private project: One self-hosted parallel job. Additionally, for each active Visual Studio + Enterprise subscriber who is a member of your organization, you get one additional self-hosted parallel job.

When the free tier is no longer sufficient:

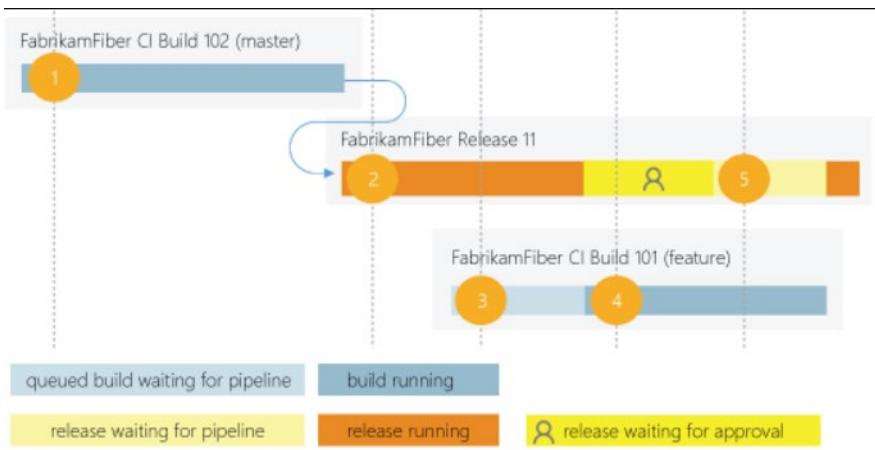
- Public project: Contact us to get your free tier limits increased.
- Private project: You can pay for additional capacity per parallel job. Buy self-hosted parallel jobs.

There are no time limits on self-hosted jobs.

Parallel jobs

How a parallel job is consumed by a build or release

Consider an organization that has only one Microsoft-hosted parallel job. This job allows users in that organization to collectively run only one build or release job at a time. When additional jobs are triggered, they are queued and will wait for the previous job to finish.



A release consumes a parallel job only when it's being actively deployed to a stage. While the release is waiting for an approval or a manual intervention, it does not consume a parallel job.

Simple example of parallel jobs

- FabrikamFiber CI Build 102 (master branch) starts first.
- Deployment of FabrikamFiber Release 11 is triggered by completion of FabrikamFiber CI Build 102.
- FabrikamFiber CI Build 101 (feature branch) is triggered. The build can't start yet because Release 11's deployment is active. So the build stays queued.
- Release 11 waits for approvals. Fabrikam CI Build 101 starts because a release that's waiting for approvals does not consume a parallel job.
- Release 11 is approved. It resumes only after Fabrikam CI Build 101 is completed.

Relationship between jobs and parallel jobs

The term job can refer to multiple concepts, and its meaning depends on the context:

- When you define a build or release, you can define it as a collection of jobs. When a build or release runs, you can run multiple jobs as part of that build or release.
- Each job consumes a parallel job that runs on an agent. When there aren't enough parallel jobs available for your organization, then the jobs are queued up and run one after the other.

When you run a server job or deploy to a deployment group, you don't consume any parallel jobs.

Estimating Parallel jobs

Determine how many parallel jobs you need

You can begin by seeing if the free tier offered in your organization is enough for your teams. When you've reached the 1,800-minute per month limit for the free tier of Microsoft-hosted parallel jobs, you can start by buying one parallel job to remove this monthly time limit before deciding to purchase more.

As the number of queued builds and releases exceeds the number of parallel jobs you have, your build and release queues will grow longer. When you find the queue delays are too long, you can purchase additional parallel jobs as needed.

Simple estimate

A simple rule of thumb: Estimate that you'll need one parallel job for every four to five users in your organization.

Detailed estimate

In the following scenarios, you might need multiple parallel jobs:

- If you have multiple teams, and if each of them require a CI build, you'll likely need a parallel job for each team.
- If your CI build trigger applies to multiple branches, you'll likely need a parallel job for each active branch.
- If you develop multiple applications by using one organization or server, you'll likely need additional parallel jobs: one to deploy each application at the same time.

View available parallel jobs

Browse to Organization settings > Pipelines > Retention and parallel jobs > Parallel jobs

Location of parallel jobs in organization settings

URL example: https://{{your_organization}}/_admin/_buildQueue?a=resourceLimits

View the maximum number of parallel jobs that are available in your organization.

Select View in-progress jobs to display all the builds and releases that are actively consuming an available parallel job or that are queued waiting for a parallel job to be available.

The screenshot shows the 'Parallel jobs' tab selected in the 'Retention and parallel jobs' section of the Azure Organization Settings. On the left, there's a sidebar with navigation links: General, Work, Pipelines (which is expanded), Agent pools, Deployment pools, Retention and parallel jobs (which is selected and highlighted in blue), and OAuth configurations. The main area displays two sections: Microsoft-hosted and Self-hosted. Under Microsoft-hosted, it shows a 'Free tier' with 1 parallel job up to 1800 mins/mo, currently 0/1800 minutes are consumed, and a 'Purchase parallel jobs' button. Under Self-hosted, it shows 2 parallel jobs, with details for Free parallel jobs (1), Visual Studio Enterprise subscribers (1), and Monthly purchases (0 Change).

Microsoft-hosted	Free tier
View in-progress jobs	1 parallel job up to 1800 mins/mo
Currently 0/1800 minutes are consumed	Purchase parallel jobs

Self-hosted	Parallel jobs
View in-progress jobs	2
Free parallel jobs	1
Visual Studio Enterprise subscribers	1
Monthly purchases	0 Change

Sharing of parallel jobs across projects in a collection

Parallel jobs are purchased at the organization level, and they are shared by all projects in an organization. Currently, there isn't a way to partition or dedicate parallel job capacity to a specific project or agent pool. For example:

- You purchase two parallel jobs in your organization.
- You queue two builds in the first project, and both the parallel jobs are consumed.
- You queue a build in the second project. That build won't start until one of the builds in your first project is completed.

Azure DevOps and Open Source projects (Public Projects)

Azure DevOps and Open Source projects

Azure DevOps offers a suite of DevOps capabilities to developers including Source control, Agile planning, Build, Release, Test and more. But to use Azure DevOps features require the user to first login using a Microsoft or GitHub Account. This however blocks a lot of interesting scenarios where you would want to share your code and artifacts publically or simply provide a wiki library or build status page for unauthenticated users....

With public projects, users will be able to mark a Azure DevOps Team Project as public. This will enable anonymous users to be able to view the contents of that project in a read-only state enabling collaboration with anonymous (un-authenticated) users that wasn't possible before. Anonymous users will largely see the same views as authenticated users, with non-public functionality such as settings, or actions (such as queue build) hidden or disabled.

Public versus private projects

Projects in Azure DevOps provide a repository for source code and a place for a group of developers and teams to plan, track progress, and collaborate on building software solutions. One or more projects can be defined within an organization in Azure DevOps.

Users that aren't signed into the service have read-only access to public projects on Azure DevOps. Private projects, on the other hand, require users to be granted access to the project and signed in to access the services.

Supported services

Non-members of a public project will have read-only access to a limited set of services, specifically:

- Browse the code base, download code, view commits, branches, and pull requests
- View and filter work items
- View a project page or dashboard
- View the project Wiki
- Perform semantic search of the code or work items

For additional information, see **Differences and limitations for non-members of a public project⁵**.

A practical example: .NET Core CLI

Supporting open source development is one of the most compelling scenarios for public projects. A good example is the .NET Core CLI. Their source is hosted on GitHub and they use Azure DevOps for their CI builds. However, if you click on the build badges in their readme, unless you were one of the maintainers of that project, you will not be able to see the build results. Since this is an open source project, everybody should be able to view the full results so they can see why a build failed and maybe even send a pull request to help fix it.

⁵ <https://docs.microsoft.com/en-us/azure/devops/organizations/public/feature-differences?view=azure-devops>

Thanks to public projects capabilities, the team will be able to enable just that experience and everyone in the community will have access to the same build results, regardless of if they are a maintainer on the project or not.

How do I qualify for the free tier of Azure Pipelines for public projects?

Microsoft will automatically apply the free tier limits for public projects if you meet both of these conditions:

- Your pipeline is part of an Azure Pipelines public project.
- Your pipeline builds a public repository from GitHub or from the same public project in your Azure DevOps organization.

Are there limits on who can use Azure Pipelines?

You can have as many users as you want when you're using Azure Pipelines. There is no per-user charge for using Azure Pipelines. Users with both basic and stakeholder access can author as many builds and releases as they want.

Are there any limits on the number of builds and release pipelines that I can create?

No. You can create hundreds or even thousands of definitions for no charge. You can register any number of self-hosted agents for no charge.

As a Visual Studio Enterprise subscriber, do I get additional parallel jobs for TFS and Azure Pipelines?

Yes. Visual Studio Enterprise subscribers get one parallel job in Team Foundation Server 2017 or later and one self-hosted parallel job in each Azure DevOps Services organization where they are a member.

What about the option to pay for hosted agents by the minute?

Some of our earlier customers are still on a per-minute plan for the hosted agents. In this plan, you pay \$0.05/minute for the first 20 hours after the free tier, and \$0.01/minute after 20 hours. Because of the following limitations in this plan, you might want to consider moving to the parallel jobs model:

When you're using the per-minute plan, you can run only one job at a time.

If you run builds for more than 14 paid hours in a month, the per-minute plan might be less cost-effective than the parallel jobs model.

Azure Pipelines YAML vs Visual Designer

Azure Pipelines and YAML

Mirroring the rise of interest in infrastructure as code, there has been a considerable interest in defining pipelines as code. However, pipeline as code doesn't simply mean executing a script that's stored in source control. Codified pipelines use their own programming model to simplify the setup and maximize reuse. A typical microservice architecture will require many deployment pipelines that are for the most part identical. It is tedious to craft these pipelines via a user interface or SDK. Having the ability to define the pipeline along with the code helps apply all principles of code sharing, reuse, templatization and code reviews.

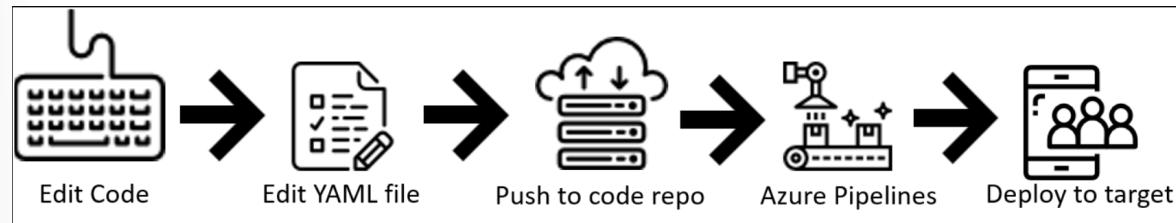
Azure DevOps offers you both experiences, you can either use YAML to define your pipelines or use the visual designer to do the same. You will however find that there are more product level investments being made to enhance the YAML pipeline experience.

When you use YAML, you define your pipeline mostly in code (a YAML file) alongside the rest of the code for your app. When you use the visual designer, you define a build pipeline to build and test your code, and then to publish artifacts. You also define a release pipeline to consume and deploy those artifacts to deployment targets.

Use Azure Pipelines with YAML

You can configure your pipelines in a YAML file that exists alongside your code.

- Configure Azure Pipelines to use your Git repo.
- Edit your azure-pipelines.yml file to define your build.
- Push your code to your version control repository. This action kicks off the default trigger to build and deploy and then monitor the results.
- Your code is now updated, built, tested, and packaged. It can be deployed to any target.



Benefits of using YAML

- The pipeline is versioned with your code and follows the same branching structure. You get validation of your changes through code reviews in pull requests and branch build policies.
- Every branch you use can modify the build policy by modifying the azure-pipelines.yml file.
- A change to the build process might cause a break or result in an unexpected outcome. Because the change is in version control with the rest of your codebase, you can more easily identify the issue.

If you think the YAML workflow is best for you, create your first pipeline by using **YAML**⁶.

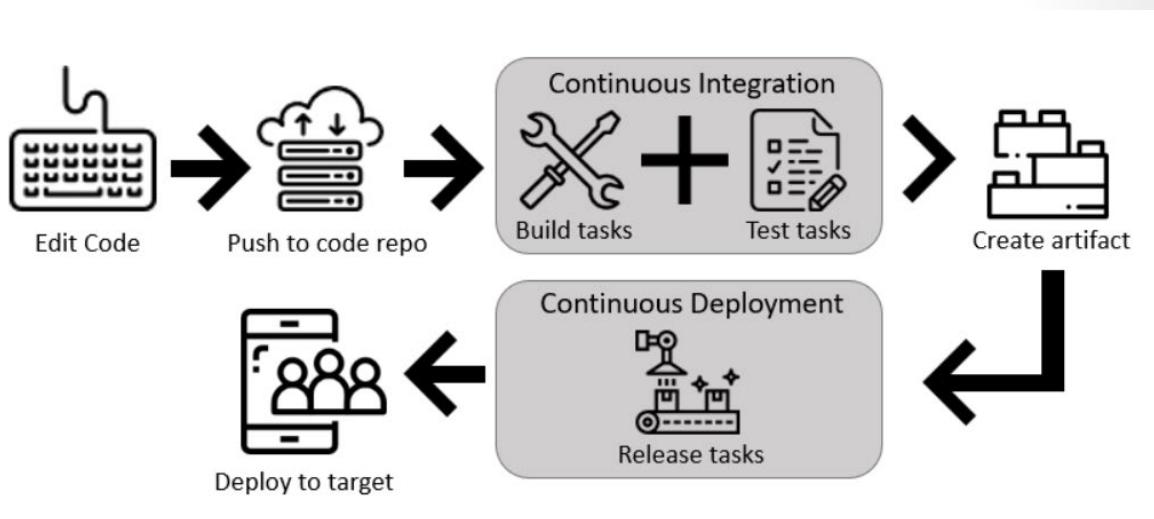
⁶ <https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started-yaml?view=vsts>

Azure Pipelines and Visual Designer

You can create and configure your build and release pipelines in the Azure DevOps web portal with the visual designer.

Configure Azure Pipelines to use your Git repo.

1. Use the Azure Pipelines visual designer to create and configure your build and release pipelines.
2. Push your code to your version control repository. This action triggers your pipeline and runs tasks such as building or testing code.
3. The build creates an artifact that's used by the rest of your pipeline to run tasks such as deploying to staging or production.
4. Your code is now updated, built, tested, and packaged. It can be deployed to any target.



Benefits of using the visual designer

The visual designer is great for users who are new to the world of continuous integration (CI) and continuous delivery (CD).

- The visual representation of the pipelines makes it easier to get started.
- The visual designer is located in the same hub as the build results. This location makes it easier to switch back and forth and make changes.

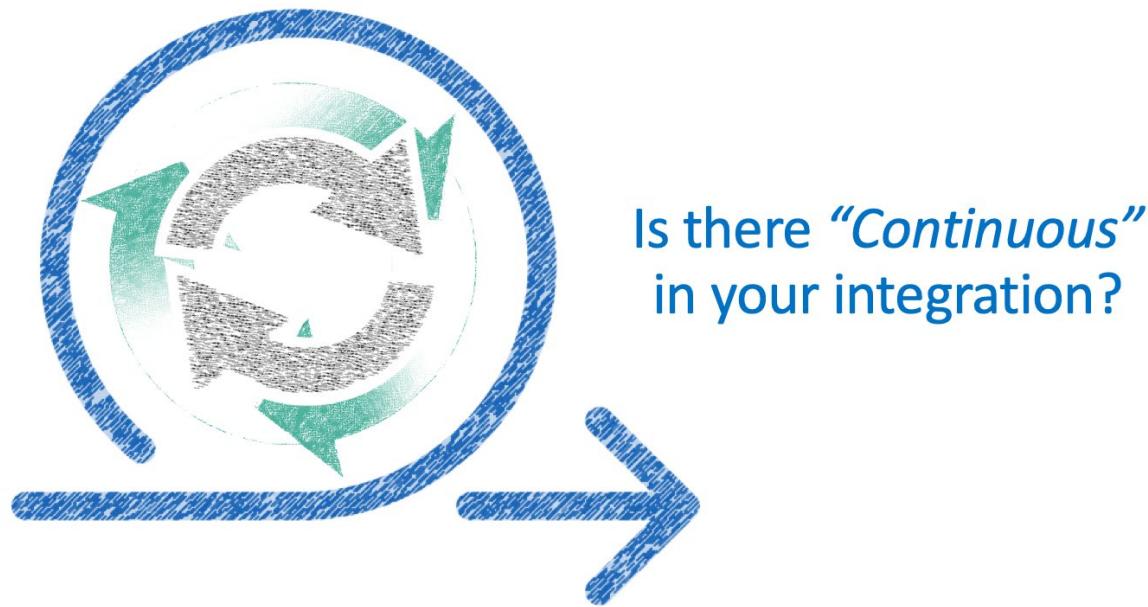
If you think the designer workflow is best for you, create your first pipeline by using the **visual designer**⁷.

⁷ <https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started-designer?view=vsts&tabs=new-nav>

Continuous Integration Overview

Introduction to Continuous Integration

Continuous Integration (CI) is the process of automating the build and testing of code every time a team member commits changes to version control. CI encourages developers to share their code and unit tests by merging their changes into a shared version control repository after every small task completion. Committing code triggers an automated build system to grab the latest code from the shared repository and to build, test, and validate the full master branch (also known as the trunk or main).



The idea is to minimize the cost of integration by making it an early consideration. Developers can discover conflicts at the boundaries between new and existing code early, while conflicts are still relatively easy to reconcile. Once the conflict is resolved, work can continue with confidence that the new code honors the requirements of the existing codebase.

Integrating code frequently does not, by itself, offer any guarantees about the quality of the new code or functionality. In many organizations, integration is costly because manual processes are used to ensure that the code meets standards, does not introduce bugs, and does not break existing functionality. Frequent integration can create friction when the level of automation does not match the amount quality assurance measures in place.

To address this friction within the integration process, in practice, continuous integration relies on robust test suites and an automated system to run those tests. When a developer merges code into the main repository, automated processes kick off a build of the new code. Afterwards, test suites are run against the new build to check whether any integration problems were introduced. If either the build or the test phase fails, the team is alerted so that they can work to fix the build.

The end goal of continuous integration is to make integration a simple, repeatable process that is part of the everyday development workflow in order to reduce integration costs and respond to defects early. Working to make sure the system is robust, automated, and fast while cultivating a team culture that encourages frequent iteration and responsiveness to build issues is fundamental to the success of the strategy.

The Four Pillars of Continuous Integration

Continuous Integration relies on four key elements for successful implementation: a Version Control System, Package Management System, Continuous Integration System, and an Automated Build Process.

A **Version Control System** manages changes to your source code over time.

- [Git⁸](#)
- [Apache Subversion⁹](#)
- [Team Foundation Version Control¹⁰](#)

A **Package Management System** is used to install, uninstall and manage software packages.

- [NuGet¹¹](#)
- [Node Package Manager¹²](#)
- [Chocolatey¹³](#)
- [HomeBrew¹⁴](#)
- [RPM¹⁵](#)

A **Continuous Integration System** merges all developer working copies to a shared mainline several times a day.

- [Azure DevOps¹⁶](#)
- [TeamCity¹⁷](#)
- [Jenkins¹⁸](#)

An **Automated Build Process** creates a software build including compiling, packaging, and running automated tests.

- [Apache Ant¹⁹](#)
- [NAnt²⁰](#)
- [Gradle²¹](#)

✓ Note: For each element, your team needs to select the specific platforms and tools they will use and you must ensure that you have established each pillar before proceeding.

Benefits of Continuous Integration

Continuous Integration (CI) provides many benefits to the development process, including:

- Improving code quality based on rapid feedback

⁸ <https://git-scm.com/>

⁹ <https://subversion.apache.org/>

¹⁰ <https://docs.microsoft.com/en-us/azure/devops/repos/tfvc/overview?view=vsts>

¹¹ <https://www.nuget.org/>

¹² <https://www.npmjs.com/>

¹³ <https://chocolatey.org/>

¹⁴ <https://brew.sh/>

¹⁵ <http://rpm.org/>

¹⁶ <https://azure.microsoft.com/en-us/services/devops>

¹⁷ <https://www.jetbrains.com/teamcity/>

¹⁸ <https://jenkins.io/>

¹⁹ <http://ant.apache.org/>

²⁰ <http://nant.sourceforge.net/>

²¹ <https://gradle.org/>

- Triggering automated testing for every code change
- Reducing build times for rapid feedback and early detection of problems (risk reduction)
- Better managing technical debt and conducting code analysis
- Reducing long, difficult, and bug-inducing merges
- Increasing confidence in codebase health long before production deployment

Key Benefit: Rapid Feedback for Code Quality

Possibly the most important benefit of continuous integration is rapid feedback to the developer. If the developer commits something and it breaks the code, he or she will know that almost immediately from the build, unit tests, and through other metrics. If successful integration is happening across the team, the developer is also going to know if their code change breaks something that another team member did to a different part of the code base. This process removes very long, difficult, and drawn out bug-inducing merges, which allows organizations to deliver in a very fast cadence.

Continuous integration also enables tracking metrics to assess code quality over time, such as unit test pass rates, code that breaks frequently, code coverage trends, and code analysis. It can be used to provide information on what has been changed between builds for traceability benefits, as well as for introducing evidence of what teams do in order to have a global view of build results.

For more information, you can see [What is Continuous Integration²²](#).

Discussion - CI Implementation Challenges

Have you tried to implement continuous integration in your organization? Were you successful? If you were successful, what lessons did you learn? If you were not successful, what were the challenges?

Build Number Formatting and Status

You may have noticed that in some demos, the build number was just an integer, yet in other demos, there is a formatted value that was based upon the date. This is one of the items that can be set in the **Build Options**.

Build Number Formatting

The example shown below is from the build options that were configured by the ASP.NET Web Application build template:

²² <https://docs.microsoft.com/en-us/azure/devops/learn/what-is-continuous-integration>

The screenshot shows the 'Build properties' section of the Azure DevOps pipeline configuration. The 'Build number format' is set to \$(date:yyyyMMdd)\$(rev:r), which generates build numbers like 20230714.1.

In this case, the date has been retrieved as a system variable, then formatted via yyyyMMdd, and the revision is then appended.

Build Status

While we have been manually queuing each build, we will see in the next lesson that builds can be automatically triggered. This is a key capability required for continuous integration. But there are times that we might not want the build to run, even if it is triggered. This can be controlled with these settings:

- The new build request is processing
- Enabled - queue and start builds when eligible agent(s) available
- Paused - queue new builds but do not start
- Disabled - do not queue new builds

Note that you can use the **Paused** setting to allow new builds to queue but to then hold off starting them.

For more information, see [Build Pipeline Options²³](#).

Build Authorizations Timeouts and Badges

Authorization and Timeouts

You can configure properties for the build job as shown below:

²³ <https://docs.microsoft.com/en-us/azure/devops/pipelines/build/options?view=vsts&tabs=yaml>

Build job

Define build job authorization and timeout settings

Build job authorization scope [\(i\)](#)

Project collection

Build job timeout in minutes [\(i\)](#)

60

Build job cancel timeout in minutes [\(i\)](#)

5

The authorization scope determines whether the build job is limited to accessing resources in the current project, or if it can access resources in other projects in the project collection.

The build job timeout determines how long the job can execute before being automatically canceled. A value of zero (or leaving the text box empty) specifies that there is no limit.

The build job cancel timeout determines how long the server will wait for a build job to respond to a cancellation request.

Badges

Some development teams like to show the state of the build on an external monitor or website. These settings provide a link to the image to use for that. Here is an example Azure Pipelines badge that has Succeeded.:



For more information, see [Build Pipeline Options²⁴](#).

²⁴ <https://docs.microsoft.com/en-us/azure/devops/pipelines/build/options?view=vsts&tabs=yaml>

Implementing a Build Strategy

Configuring Agent Demands

Not all agents are the same. We've seen that they can be based on different operating systems, but they can also have different dependencies installed. To describe this, every agent has a set of capabilities configured as name-value pairs. The capabilities such as machine name and type of operating system that are automatically discovered, are referred to as **system capabilities**. The ones that you define are called **user capabilities**.

On the Agent Pools page (at the Organization level), when you select an agent, there is a tab for **Capabilities**. You can use it to see the available capabilities for an agent, and to configure user capabilities. For the self-hosted agent that I configured in the earlier demo, you can see the capabilities on that tab:

Capability name	Capability value
Agent.ComputerName	GREGP50
Agent.HomeDirectory	C:\agent
Agent.Name	GREGP50
Agent.OS	Windows_NT
Agent.OSArchitecture	X64
Agent.OSVersion	10.0.17134
Agent.Version	2.141.2
ALLUSERSPROFILE	C:\ProgramData
APPDATA	C:\Users\Greg\AppData\Roaming
AzurePS	5.7.0
bower	C:\Users\Greg\AppData\Roaming\npm\bower.cmd
Cmd	C:\WINDOWS\system32\cmd.exe
CommonProgramFiles	C:\Program Files\Common Files
CommonProgramFiles(x86)	C:\Program Files (x86)\Common Files
CommonProgramW6432	C:\Program Files\Common Files
COMPUTERNAME	GREGP50

When you configure a build pipeline, as well as the agent pool to use, on the **Options** tab, you can specify certain demands that the agent must meet.

The screenshot shows the 'Build job' configuration section of an Azure Pipeline. It includes fields for 'Project collection' (set to 'Project collection'), 'Build job timeout in minutes' (set to 60), 'Build job cancel timeout in minutes' (set to 5), and a 'Demands' section where 'HasPaymentService' is specified with the condition 'exists'. A 'Add' button is also visible.

In the above image, the HasPaymentService is required in the collection of capabilities. As well as an **exists** condition, you can choose that a capability **equals** a specific value.

For more information, see [Capabilities²⁵](#).

Implementing Multi-Agent Builds

You can use multiple build agents to support multiple build machines, either to distribute the load, to run builds in parallel, or to use different agent capabilities. As an example, components of an application might require different incompatible versions of a library or dependency.

Multiple Jobs in a Pipeline

Adding multiple jobs to a pipeline lets you:

- Break your pipeline into sections that need different agent pools, or self-hosted agents
- Publish artifacts in one job and consume them in one or more subsequent jobs
- Build faster by running multiple jobs in parallel
- Enable conditional execution of tasks

To add another agent job to an existing pipeline, click on the ellipsis and choose as shown in this image:

²⁵ <https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/agents?view=vsts#capabilities>

... > Parts Unlimited-ASP.NET-CI

The screenshot shows the Azure DevOps Pipeline editor for the project "Parts Unlimited-ASP.NET-CI". At the top, there are tabs for Tasks, Variables, Triggers, Options, Retention, History, Save & queue, Discard, and Summary. Below the tabs, the pipeline is defined with a "Get sources" task and an "Agent job 1" task. A context menu is open at the end of the pipeline, indicated by a red arrow, with options to "Add an agent job" and "Add an agentless job". A link to "Learn more about jobs" is also visible.

Parallel Jobs

At the Organization level, you can configure the number of parallel jobs that are made available.

The screenshot shows the "Organization Settings" page. On the left, there is a sidebar with links: General, Overview, Projects, Policy, Users, Security, Notifications, Extensions, Usage, Boards, Process, Pipelines, Agent pools, Deployment pools, Retention and parallel jobs (which has a red arrow pointing to it), and OAuth configurations. The "Parallel jobs" tab is selected under the "Pipelines" section. The main content area shows "Private projects" and "Public projects" sections, each with Microsoft-hosted and Self-hosted options, along with their respective parallel job counts and purchase links.

Parallel jobs

The free tier allows for one parallel job of up to 1800 minutes per month. The self-hosted agents has higher levels.

- ✓ Note: You can define a build as a collection of jobs, rather than as a single job. Each job consumes one of these parallel jobs that runs on an agent. If there aren't enough parallel jobs available for your organization, the jobs will be queued and run sequentially.

Discussion - Build-Related Tooling

Azure DevOps can be integrated with a wide range of existing tooling that is used for builds or associated with builds. Which build-related tools do you currently use? What do you like or don't like about the tools?

Integration with Azure Pipelines

Anatomy of a pipeline

Azure Pipelines can automatically build and validate every pull request and commit to your Azure Repos Git repository. Azure Pipelines can be used with Azure DevOps public projects as well as Azure DevOps private projects. In future sections of this training we'll also learn how to use Azure Repos with external code repositories such as GitHub. Let's start off by creating a hello world yaml pipeline...

Hello World

Let's start off slowly and create a simple pipeline that echos "Hello World!" to the console. No technical course is complete without a Hello World example.

```
name: 1.0$(Rev:.r)

# simplified trigger (implied branch)
trigger:
- master

# equivalent trigger
# trigger:
#   branches:
#     include:
#       - master

variables:
  name: martin

pool:
  vmImage: ubuntu-latest

jobs:
- job: helloworld
  steps:
    - script: echo "Hello, $(name)"
```

Most pipelines will have these components:

- Name – though often this is skipped (if it is skipped, a date-based name is generated automatically)
- Trigger – more on triggers later, but without an explicit trigger, there's an implicit "trigger on every commit to any path from any branch in this repo"
- Variables – these are "inline" variables (more on other types of variables later)
- Job – every pipeline must have at least one job
- Pool – you configure which pool (queue) the job must run on
- Checkout – the "checkout: self" tells the job which repository (or repositories if there are multiple checkouts) to checkout for this job

- Steps – the actual tasks that need to be executed: in this case a “script” task (script is an alias) that can run inline scripts

Name

The variable name is a bit misleading, since the name is really the build number format. If you do not explicitly set a name format, you'll get an integer number. This is a monotonically increasing number for runs triggered off this pipeline, starting at 1. This number is stored in Azure DevOps. You can make use of this number by referencing \$(Rev).

To make a date-based number, you can use the format \$(Date:yyyy-mm-dd-HH-mm) to get a build number like 2020-01-16-19-22. To get a semantic number like 1.0.x, you can use something like 1.0\$(Rev:.r)

Triggers

If there is no explicit triggers section, then it is implied that any commit to any path in any branch will trigger this pipeline to run. You can be more explicit though using filters such as branches and/or paths.

Let's consider this trigger:

```
trigger:
  branches:
    include:
      - master
```

This trigger is configured to queue the pipeline only when there is a commit to the master branch. What about triggering for any branch except master? You guessed it: use exclude instead of include:

```
trigger:
  branches:
    exclude:
      - master
```

TIP: You can get the name of the branch from the variables Build.SourceBranch (for the full name like refs/heads/master) or Build.SourceBranchName (for the short name like master).

What about a trigger for any branch with a name that starts with topic/ and only if the change is in the webapp folder?

```
trigger:
  branches:
    include:
      - feature/*
  paths:
    include:
      - webapp/**
```

Of course, you can mix includes and excludes if you really need to. You can also filter on tags.

TIP: Don't forget one overlooked trigger: none. If you never want your pipeline to trigger automatically, then you can use none. This is useful if you want to create a pipeline that is only manually triggered.

There are other triggers for other events such as:

- Pull Requests (PRs), which can also filter on branches and paths
- Schedules, which allow you to specify cron expressions for scheduling pipeline runs
- Pipelines, which allow you to trigger pipelines when other pipelines complete, allowing pipeline chaining

You can find all the documentation on triggers [here²⁶](#).

Jobs

A job is a set of steps that are executed by an agent in a queue (or pool). Jobs are atomic – that is, they are executed wholly on a single agent. You can configure the same job to run on multiple agents at the same time, but even in this case the entire set of steps in the job are run on every agent. If you need some steps to run on one agent and some on another, you'll need two jobs.

A job has the following attributes besides its name:

1. displayName – a friendly name
2. dependsOn - a way to specify dependencies and ordering of multiple jobs
3. condition – a binary expression: if this evaluates to true, the job runs; if false, the job is skipped
4. strategy - used to control how jobs are parallelized
5. continueOnError - to specify if the remainder of the pipeline should continue or not if this job fails
6. pool – the name of the pool (queue) to run this job on
7. workspace - managing the source workspace
8. container - for specifying a container image to execute the job in - more on this later
9. variables – variables scoped to this job
10. steps – the set of steps to execute
11. timeoutInMinutes and cancelTimeoutInMinutes for controlling timeouts
12. services - sidecar services that you can spin up

Dependencies

You can define dependencies between jobs using the `dependsOn` property. This lets you specify sequences and fan-out and fan-in scenarios. If you do not explicitly define a dependency, a sequential dependency is implied. If you truly want jobs to run in parallel, you need to specify `dependsOn: none`.

Let's look at a few examples. Consider this pipeline:

```
jobs:  
  - job: A  
    steps:  
      # steps omitted for brevity  
  
  - job: B  
    steps:
```

²⁶ <https://docs.microsoft.com/en-us/azure/devops/pipelines/build/triggers?view=azure-devops&tabs=yaml>

```
# steps ommited for brevity
```

Because no dependsOn was specified, the jobs will run sequentially: first A and then B.

To have both jobs run in parallel, we just add dependsOn: none to job B:

```
jobs:
- job: A
  steps:
    # steps ommited for brevity

- job: B
  dependsOn: none
  steps:
    # steps ommited for brevity
```

If we want to fan out and fan in, we can do that too:

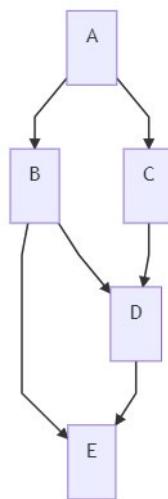
```
jobs:
- job: A
  steps:
    - script: echo 'job A'

- job: B
  dependsOn: A
  steps:
    - script: echo 'job B'

- job: C
  dependsOn: A
  steps:
    - script: echo 'job C'

- job: D
  dependsOn:
    - B
    - C
  steps:
    - script: echo 'job D'

- job: E
  dependsOn:
    - B
    - D
  steps:
    - script: echo 'job E'
```



Checkout

Classic builds implicitly checkout any repository artifacts, but pipelines require you to be more explicit using the `checkout` keyword:

- Jobs check out the repo they are contained in automatically unless you specify `checkout: none`.
- Deployment jobs do not automatically check out the repo, so you'll need to specify `checkout: self` for deployment jobs if you want to get access to files in the YAML file's repo.

Download

Downloading artifacts requires you to use the `download` keyword. Downloads also work the opposite way for jobs and deployment jobs:

- Jobs do not download anything unless you explicitly define a download
- Deployment jobs implicitly perform a download: current which downloads any pipeline artifacts that have been created in the current pipeline. To prevent this, you must specify `download: none`.

Resources

What if your job requires source code in another repository? You'll need to use resources. Resources let you reference:

1. other repositories
2. pipelines
3. builds (classic builds)
4. containers (for container jobs)
5. packages

To reference code in another repo, specify that repo in the resources section and then reference it via its alias in the checkout step:

```
resources:  
  repositories:
```

```
- repository: appcode
  type: git
  name: otherRepo

steps:
- checkout: appcode
```

Steps are Tasks

Steps are the actual “things” that execute, in the order that they are specified in the job. Each step is a task: there are out of the box (OOB) tasks that come with Azure DevOps, many of which have aliases, and there are tasks that get installed to your Azure DevOps account via the marketplace.

Creating custom tasks is beyond the scope of this chapter, but you can see how to create your own custom tasks [here](#)²⁷.

Variables

It would be tough to achieve any sort of sophistication in your pipelines without variables. There are several types of variables, though this classification is partly mine and pipelines don’t distinguish between these types. However, I’ve found it useful to categorize pipeline variables to help teams understand some of the nuances that occur when dealing with them.

Every variable is really a key:value pair. The key is the name of the variable, and it has a value.

To dereference a variable, simply wrap the key in \$. Let’s consider this simple example:

```
variables:
  name: martin
steps:
- script: echo "Hello, $(name)!"
```

This will write Hello, martin! to the log.

Pipeline Structure

A pipeline is one or more stages that describe a CI/CD process. Stages are the major divisions in a pipeline. The stages “Build this app,” “Run these tests,” and “Deploy to preproduction” are good examples.

A stage is one or more jobs, which are units of work assignable to the same machine. You can arrange both stages and jobs into dependency graphs. Examples include “Run this stage before that one” and “This job depends on the output of that job.”

A job is a linear series of steps. Steps can be tasks, scripts, or references to external templates.

²⁷ <https://docs.microsoft.com/en-us/azure/devops/extend/develop/add-build-task?view=azure-devops>

This hierarchy is reflected in the structure of a YAML file like:

- Pipeline

- Stage A

- Job 1

- Step 1.1

- Step 1.2

- ...

- Job 2

- Step 2.1

- Step 2.2

- ...

- Stage B

- ...

Simple pipelines don't require all of these levels. For example, in a single-job build you can omit the containers for stages and jobs because there are only steps. And because many options shown in this article aren't required and have good defaults, your YAML definitions are unlikely to include all of them.

Pipeline

The schema for a pipeline...

```
name: string    # build numbering format
resources:
  pipelines: [ pipelineResource ]
  containers: [ containerResource ]
  repositories: [ repositoryResource ]
variables: # several syntaxes
trigger: trigger
pr: pr
stages: [ stage | templateReference ]
```

If you have a single stage, you can omit the stages keyword and directly specify the jobs keyword:

```
# ... other pipeline-level keywords
jobs: [ job | templateReference ]
```

If you have a single stage and a single job, you can omit the stages and jobs keywords and directly specify the steps keyword:

```
# ... other pipeline-level keywords
steps: [ script | bash | pwsh | powershell | checkout | task | templateRef-
```

```
erence ]
```

Stage

A stage is a collection of related jobs. By default, stages run sequentially. Each stage starts only after the preceding stage is complete.

Use approval checks to manually control when a stage should run. These checks are commonly used to control deployments to production environments.

Checks are a mechanism available to the resource owner. They control when a stage in a pipeline consumes a resource. As an owner of a resource like an environment, you can define checks that are required before a stage that consumes the resource can start.

This example runs three stages, one after another. The middle stage runs two jobs in parallel.

```
stages:
- stage: Build
  jobs:
    - job: BuildJob
      steps:
        - script: echo Building!
- stage: Test
  jobs:
    - job: TestOnWindows
      steps:
        - script: echo Testing on Windows!
    - job: TestOnLinux
      steps:
        - script: echo Testing on Linux!
- stage: Deploy
  jobs:
    - job: Deploy
      steps:
        - script: echo Deploying the code!
```

Job

A job is a collection of steps run by an agent or on a server. Jobs can run conditionally and might depend on earlier jobs.

```
jobs:
- job: MyJob
  displayName: My First Job
  continueOnError: true
  workspace:
    clean: outputs
  steps:
    - script: echo My first job
```

Steps

A step is a linear sequence of operations that make up a job. Each step runs in its own process on an agent and has access to the pipeline workspace on a local hard drive. This behavior means environment variables aren't preserved between steps but file system changes are.

```
steps:  
- script: echo This runs in the default shell on any machine  
- bash: |  
    echo This multiline script always runs in Bash.  
    echo Even on Windows machines!  
- pwsh: |  
    Write-Host "This multiline script always runs in PowerShell Core."  
    Write-Host "Even on non-Windows machines!"
```

Tasks

Tasks are the building blocks of a pipeline. There's a catalog of tasks available to choose from.

```
steps:  
- task: VSBuild@1  
  displayName: Build  
  timeoutInMinutes: 120  
  inputs:  
    solution: '**\*.sln'
```

Templates

Template references

You can export reusable sections of your pipeline to a separate file. These separate files are known as templates. Azure Pipelines supports four kinds of templates:

Azure Pipelines supports four kinds of templates:

- Stage
- Job
- Step
- Variable

You can also use templates to control what is allowed in a pipeline and to define how parameters can be used.

- Parameter

Templates themselves can include other templates. Azure Pipelines supports a maximum of 50 unique template files in a single pipeline.

Stage templates

You can define a set of stages in one file and use it multiple times in other files.

In this example, a stage is repeated twice for two different testing regimes. The stage itself is specified only once.

```
# File: stages/test.yml

parameters:
  name: ''
  testFile: ''

stages:
- stage: Test_${{ parameters.name }}
  jobs:
  - job: ${{ parameters.name }}_Windows
    pool:
      vmImage: vs2017-win2016
    steps:
    - script: npm install
    - script: npm test -- --file=${{ parameters.testFile }}
  - job: ${{ parameters.name }}_Mac
    pool:
      vmImage: macos-10.14
    steps:
    - script: npm install
    - script: npm test -- --file=${{ parameters.testFile }}
```

Templated pipeline

```
# File: azure-pipelines.yml

stages:
- template: stages/test.yml # Template reference
  parameters:
    name: Mini
    testFile: tests/minisuite.js

- template: stages/test.yml # Template reference
  parameters:
    name: Full
    testFile: tests/fullsuite.js
```

Job templates

You can define a set of jobs in one file and use it multiple times in other files.

In this example, a single job is repeated on three platforms. The job itself is specified only once.

```
# File: jobs/build.yml

parameters:
  name: ''
  pool: ''
  sign: false
```

```
jobs:
- job: ${{ parameters.name }}
  pool: ${{ parameters.pool }}
  steps:
  - script: npm install
  - script: npm test
  - ${{ if eq(parameters.sign, 'true') }}:
    - script: sign

# File: azure-pipelines.yml

jobs:
- template: jobs/build.yml # Template reference
  parameters:
    name: macOS
    pool:
      vmImage: 'macOS-10.14'

- template: jobs/build.yml # Template reference
  parameters:
    name: Linux
    pool:
      vmImage: 'ubuntu-16.04'

- template: jobs/build.yml # Template reference
  parameters:
    name: Windows
    pool:
      vmImage: 'vs2017-win2016'
    sign: true # Extra step on Windows only
```

Step templates

You can define a set of steps in one file and use it multiple times in another file.

```
# File: steps/build.yml

steps:
- script: npm install
- script: npm test

# File: azure-pipelines.yml

jobs:
- job: macos
  pool:
    vmImage: 'macOS-10.14'
  steps:
    - template: steps/build.yml # Template reference
```

```
- job: Linux
  pool:
    vmImage: 'ubuntu-16.04'
  steps:
    - template: steps/build.yml # Template reference

- job: Windows
  pool:
    vmImage: 'vs2017-win2016'
  steps:
    - template: steps/build.yml # Template reference
    - script: sign           # Extra step on Windows only
```

Variable templates

You can define a set of variables in one file and use it multiple times in other files.

In this example, a set of variables is repeated across multiple pipelines. The variables are specified only once.

```
# File: variables/build.yml
variables:
- name: vmImage
  value: vs2017-win2016
- name: arch
  value: x64
- name: config
  value: debug

# File: component-x-pipeline.yml
variables:
- template: variables/build.yml # Template reference
pool:
  vmImage: ${{ variables.vmImage }}
steps:
- script: build x ${{ variables.arch }} ${{ variables.config }}

# File: component-y-pipeline.yml
variables:
- template: variables/build.yml # Template reference
pool:
  vmImage: ${{ variables.vmImage }}
steps:
- script: build y ${{ variables.arch }} ${{ variables.config }}
```

Resources

A resource is any external service that is consumed as part of your pipeline. An example of a resource is another CI/CD pipeline that produces:

- Artifacts like Azure Pipelines or Jenkins.
- Code repositories like GitHub, Azure Repos, or Git.
- Container-image registries like Azure Container Registry or Docker hub.

Resources in YAML represent sources of pipelines, containers, repositories, and types. For more information on Resources, [see here²⁸](#).

General Schema

```
resources:  
  pipelines: [ pipeline ]  
  repositories: [ repository ]  
  containers: [ container ]
```

Pipeline resource

If you have an Azure pipeline that produces artifacts, your pipeline can consume the artifacts by using the pipeline keyword to define a pipeline resource.

```
resources:  
  pipelines:  
    - pipeline: MyAppA  
      source: MyCIPipelineA  
    - pipeline: MyAppB  
      source: MyCIPipelineB  
      trigger: true  
    - pipeline: MyAppC  
      project: DevOpsProject  
      source: MyCIPipelineC  
      branch: releases/M159  
      version: 20190718.2  
      trigger:  
        branches:  
          include:  
            - master  
            - releases/*  
          exclude:  
            - users/*
```

²⁸ <https://docs.microsoft.com/en-us/azure/devops/pipelines/process/resources?view=azure-devops&tabs=schema>

Container resource

Container jobs let you isolate your tools and dependencies inside a container. The agent launches an instance of your specified container then runs steps inside it. The container keyword lets you specify your container images.

Service containers run alongside a job to provide various dependencies like databases.

```
resources:
  containers:
    - container: linux
      image: ubuntu:16.04
    - container: windows
      image: myprivate.azurecr.io/windowsservercore:1803
      endpoint: my_acr_connection
    - container: my_service
      image: my_service:tag
      ports:
        - 8080:80 # bind container port 80 to 8080 on the host machine
        - 6379 # bind container port 6379 to a random available port on the
host machine
      volumes:
        - /src/dir:/dst/dir # mount /src/dir on the host into /dst/dir in the
container
```

Repository resource

If your pipeline has templates in another repository, or if you want to use multi-repo checkout with a repository that requires a service connection, you must let the system know about that repository. The repository keyword lets you specify an external repository.

```
resources:
  repositories:
    - repository: common
      type: github
      name: Contoso/CommonTools
      endpoint: MyContosoServiceConnection
```

Checkout multiple repositories in your pipeline

It is not uncommon to have micro git repositories providing utilities that are used in multiple pipelines within your project. Pipelines often rely on multiple repositories. You can have different repositories with source, tools, scripts, or other items that you need to build your code. By using multiple checkout steps in your pipeline, you can fetch and check out other repositories in addition to the one you use to store your YAML pipeline. Previously Azure Pipelines has not offered support for using multiple code repositories in a single pipeline. It has been possible to work around this by using artifacts or directly cloning other repositories via script within a pipeline (this leaves access management and security down to you). Repositories are now a first class citizen within Azure Pipelines. This enables some interesting use cases such as the ability to checkout specific parts of a repository, checkout multiple repositories. There is also a use case for not checking out any repository in the pipeline. This can be useful in cases where you are setting up a pipeline to perform a job that has no dependency on any repository.

Specify multiple repositories

Repositories can be specified as a repository resource, or inline with the checkout step. Supported repositories are Azure Repos Git (git), GitHub (github), and BitBucket Cloud (bitbucket).

The following combinations of checkout steps are supported.

- If there are no checkout steps, the default behavior is as if checkout: self were the first step.
- If there is a single checkout: none step, no repositories are synced or checked out.
- If there is a single checkout: self step, the current repository is checked out.
- If there is a single checkout step that isn't self or none, that repository is checked out instead of self.
- If there are multiple checkout steps, each designated repository is checked out to a folder named after the repository, unless a different path is specified in the checkout step. To check out self as one of the repositories, use checkout: self as one of the checkout steps.

Repository Resource - How to do it?

You must use a repository resource if your repository type requires a service connection or other extended resources field. You may use a repository resource even if your repository type doesn't require a service connection, for example if you have a repository resource defined already for templates in a different repository.

In the following example, three repositories are declared as repository resources, and then these repositories are checked out along with the current self repository that contains the pipeline YAML.

```
rresources:  
  repositories:  
    - repository: MyGitHubRepo # The name used to reference this repository  
      in the checkout step  
        type: github  
        endpoint: MyGitHubServiceConnection  
        name: MyGitHubOrgOrUser/MyGitHubRepo  
    - repository: MyBitBucketRepo  
      type: bitbucket  
      endpoint: MyBitBucketServiceConnection  
      name: MyBitBucketOrgOrUser/MyBitBucketRepo  
    - repository: MyAzureReposGitRepository  
      type: git  
      name: MyProject/MyAzureReposGitRepo  
  
trigger:  
  - master  
  
pool:  
  vmImage: 'ubuntu-latest'  
  
steps:  
  - checkout: self  
  - checkout: MyGitHubRepo  
  - checkout: MyBitBucketRepo  
  - checkout: MyAzureReposGitRepository
```

```
- script: dir $(Build.SourcesDirectory)
```

If the self repository is named CurrentRepo, the script command produces the following output: CurrentRepo MyAzureReposGitRepo MyBitBucketRepo MyGitHubRepo. In this example, the names of the repositories are used for the folders, because no path is specified in the checkout step.

Inline - How to do it?

If your repository doesn't require a service connection, you can declare it inline with your checkout step.

steps:

```
- checkout: git://MyProject/MyRepo # Azure Repos Git repository in the same organization
```

The default branch is checked out unless you designate a specific ref.

If you are using inline syntax, designate the ref by appending @ref. For example:

```
- checkout: git://MyProject/MyRepo@features/tools # checks out the features/tools branch  
- checkout: git://MyProject/MyRepo@refs/heads/features/tools # also checks out the features/tools branch  
- checkout: git://MyProject/MyRepo@refs/tags/MyTag # checks out the commit referenced by MyTag.
```

Integrate external source control with Azure Pipelines

Source Control Types supported by Azure Pipelines

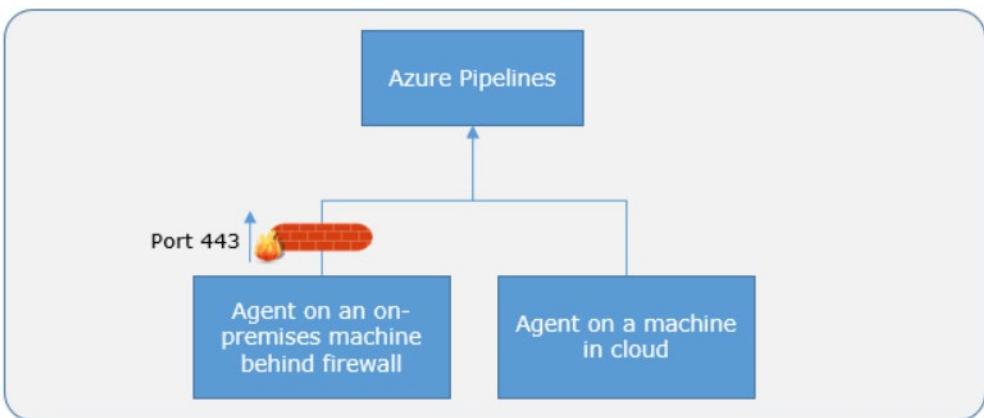
As previously discussed Azure Pipelines offers both Yaml based pipelines and the classic editor. The table below shows the repository types supported by both.

Repository type	Azure Pipelines (YAML)	Azure Pipelines (classic editor)
Azure Repos Git	Yes	Yes
Azure Repos TFVC	No	Yes
Bitbucket Cloud	Yes	Yes
Other Git (generic)	No	Yes
GitHub	Yes	Yes
GitHub Enterprise Server	Yes	Yes
Subversion	No	Yes

Set Up Private Agents

Communication with Azure Pipelines

The agent communicates with Azure Pipelines to determine which job it needs to run, and to report the logs and job status. This communication is always initiated by the agent. All the messages from the agent to Azure Pipelines over HTTPS, depending on how you configure the agent. This pull model allows the agent to be configured in different topologies as shown below.



Here is a common communication pattern between the agent and Azure Pipelines.

The user registers an agent with Azure Pipelines by adding it to an agent pool. You need to be an agent pool administrator to register an agent in that agent pool. The identity of agent pool administrator is needed only at the time of registration and is not persisted on the agent, nor is used in any further communication between the agent and Azure Pipelines. Once the registration is complete, the agent downloads a listener OAuth token and uses it to listen to the job queue.

Periodically, the agent checks to see if a new job request has been posted for it in the job queue in Azure Pipelines. When a job is available, the agent downloads the job as well as a job-specific OAuth token. This token is generated by Azure Pipelines for the scoped identity specified in the pipeline. That token is short lived and is used by the agent to access resources (e.g., source code) or modify resources (e.g., upload test results) on Azure Pipelines within that job.

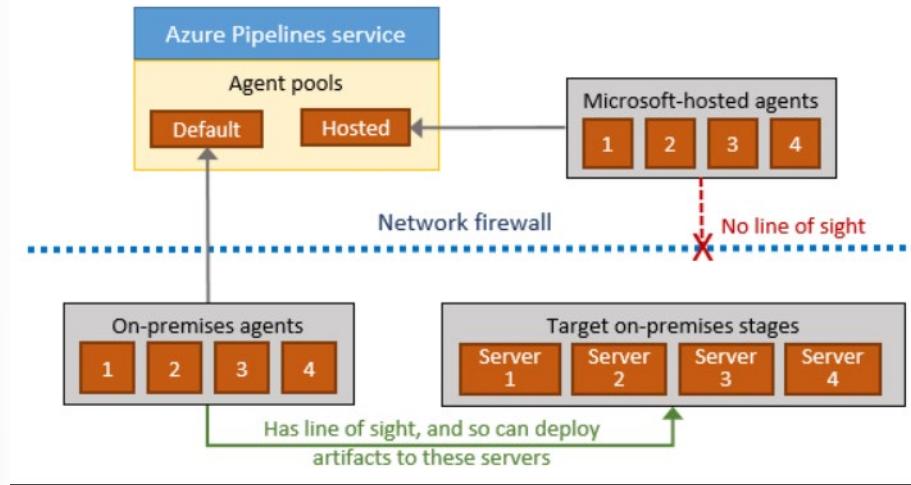
Once the job is completed, the agent discards the job-specific OAuth token and goes back to checking if there is a new job request using the listener OAuth token.

The payload of the messages exchanged between the agent and Azure Pipelines are secured using asymmetric encryption. Each agent has a public-private key pair, and the public key is exchanged with the server during registration. The server uses the public key to encrypt the payload of the job before sending it to the agent. The agent decrypts the job content using its private key. This is how secrets stored in build pipelines, release pipelines, or variable groups are secured as they are exchanged with the agent.

Communication to Deploy to Target Servers

When you use the agent to deploy artifacts to a set of servers, it must have “line of sight” connectivity to those servers. The Microsoft-hosted agent pools, by default, have connectivity to Azure websites and servers running in Azure.

If your on-premises environments do not have connectivity to a Microsoft-hosted agent pool (which is typically the case due to intermediate firewalls), you'll need to manually configure a self-hosted agent on on-premises computer(s). The agents must have connectivity to the target on-premises environments, and access to the Internet to connect to Azure Pipelines or Team Foundation Server, as shown in the following schematic.



Other Considerations

Authentication

To register an agent, you need to be a member of the administrator role in the agent pool. The identity of agent pool administrator is needed only at the time of registration and is not persisted on the agent, and is not used in any subsequent communication between the agent and Azure Pipelines. In addition, you must be a local administrator on the server in order to configure the agent. Your agent can authenticate to Azure Pipelines or TFS using one of the following methods:

Personal Access Token (PAT)

Generate and use a PAT to connect an agent with Azure Pipelines. PAT is the only scheme that works with Azure Pipelines. Also, as explained above, this PAT is used only at the time of registering the agent, and not for subsequent communication.

Interactive vs. service

You can run your agent as either a service or an interactive process. Whether you run an agent as a service or interactively, you can choose which account you use to run the agent. Note that this is different from the credentials that you use when you register the agent with Azure Pipelines. The choice of agent account depends solely on the needs of the tasks running in your build and deployment jobs.

For example, to run tasks that use Windows authentication to access an external service, you must run the agent using an account that has access to that service. However, if you are running UI tests such as Selenium or Coded UI tests that require a browser, the browser is launched in the context of the agent account.

After you've configured the agent, we recommend you first try it in interactive mode to make sure it works. Then, for production use, we recommend you run the agent in one of the following modes so that

it reliably remains in a running state. These modes also ensure that the agent starts automatically if the machine is restarted.

As a service. You can leverage the service manager of the operating system to manage the lifecycle of the agent. In addition, the experience for auto-upgrading the agent is better when it is run as a service.

As an interactive process with auto-logon enabled. In some cases, you might need to run the agent interactively for production use - such as to run UI tests. When the agent is configured to run in this mode, the screen saver is also disabled. Some domain policies may prevent you from enabling auto-logon or disabling the screen saver. In such cases, you may need to seek an exemption from the domain policy, or run the agent on a workgroup computer where the domain policies do not apply.

Note: There are security risks when you enable automatic logon or disable the screen saver because you enable other users to walk up to the computer and use the account that automatically logs on. If you configure the agent to run in this way, you must ensure the computer is physically protected; for example, located in a secure facility. If you use Remote Desktop to access the computer on which an agent is running with auto-logon, simply closing the Remote Desktop causes the computer to be locked and any UI tests that run on this agent may fail. To avoid this, use the tscon command to disconnect from Remote Desktop. For example:

```
%windir%\System32\tscon.exe 1 /dest:console
```

Agent version and upgrades

Microsoft updates the agent software every few weeks in Azure Pipelines. The agent version is indicated in the format {major}.{minor}. For instance, if the agent version is 2.1, then the major version is 2 and the minor version is 1. When a newer version of the agent is only different in minor version, it is automatically upgraded by Azure Pipelines. This upgrade happens when one of the tasks requires a newer version of the agent.

If you run the agent interactively, or if there is a newer major version of the agent available, then you have to manually upgrade the agents. You can do this easily from the agent pools tab under your project collection or organization.

You can view the version of an agent by navigating to Agent pools and selecting the Capabilities tab for the desired agent.

```
Azure Pipelines: <a href="https://dev.azure.com/{your_organization}/_admin/_AgentPool" title="" target="_blank" data-generated=''>https://dev.azure.com/{your_organization}/_admin/_AgentPool</a>
```

Question and Answer

Do self-hosted agents have any performance advantages over Microsoft-hosted agents?

In many cases, yes. Specifically:

If you use a self-hosted agent you can run incremental builds. For example, you define a CI build pipeline that does not clean the repo and does not perform a clean build, your builds will typically run faster. When you use a Microsoft-hosted agent, you don't get these benefits because the agent is destroyed after the build or release pipeline is completed.

A Microsoft-hosted agent can take longer to start your build. While it often takes just a few seconds for your job to be assigned to a Microsoft-hosted agent, it can sometimes take several minutes for an agent to be allocated depending on the load on our system.

Can I install multiple self-hosted agents on the same machine?

Yes. This approach can work well for agents that run jobs that don't consume a lot of shared resources. For example, you could try it for agents that run releases that mostly orchestrate deployments and don't do a lot of work on the agent itself.

You might find that in other cases you don't gain much efficiency by running multiple agents on the same machine. For example, it might not be worthwhile for agents that run builds that consume a lot of disk and I/O resources.

You might also run into problems if parallel build jobs are using the same singleton tool deployment, such as npm packages. For example, one build might update a dependency while another build is in the middle of using it, which could cause unreliable results and errors.

Further instructions on how to set up private agents can be found at:

- **Self-hosted Windows agents²⁹**
- **Run a self-hosted agent behind a web proxy³⁰**

²⁹ <https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/v2-windows?view=vsts>

³⁰ <https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/proxy?view=vsts&tabs=windows>

Analyze and Integrate Docker Multi-Stage Builds

Builder Patterns

With compiled runtimes like Go, Java and .NET, you'll want to first compile your code before having a binary that can be run. The components required to compile your code are not required to run your code. And the SDKs can be quite big, not to mention any potential attack surface area. A workaround which is informally called the builder pattern involves using two Docker images - one to perform a build and another to ship the results of the first build without the penalty of the build-chain and tooling in the first image.

An example of the builder pattern:

- Derive from a dotnet base image with the whole runtime/SDK (Dockerfile.build)
- Add source code
- Produce a statically-linked binary
- Copy the static binary from the image to the host (docker create, docker cp)
- Derive from SCRATCH or some other light-weight image (Dockerfile)
- Add the binary back in
- Push a tiny image to the Docker Hub

This normally meant having two separate Dockerfiles and a shell script to orchestrate all of the 7 steps above. Additionally the challenge with building on the host, including hosted build agents is we must first have a build agent with everything we need, including the specific versions. If your dev shop has any history of .NET Apps, you'll likely have multiple versions to maintain. Which means you have complex agents to deal with the complexities.

Multi-stage Builds

What are multi-stage builds?

Multi-stage builds give the benefits of the builder pattern without the hassle of maintaining three separate files... Lets take a look at a multi-stage dockerfile...

```
FROM microsoft/aspnetcore:2.0 AS base
WORKDIR /app
EXPOSE 80

FROM microsoft/aspnetcore-build:2.0 AS builder
WORKDIR /src
COPY *.sln .
COPY Web/Web.csproj Web/
RUN dotnet restore
COPY .. .
WORKDIR /src/Web
RUN dotnet build -c Release -o /app

FROM builder AS publish
RUN dotnet publish -c Release -o /app
```

```
FROM base AS production
WORKDIR /app
COPY --from=publish /app .
ENTRYPOINT ["dotnet", "Web.dll"]
```

At first, it simply looks like several dockerfiles stitched together. Multi-stage Dockerfiles can be layered or inherited. When you look closer, there are a couple of key things to realize.

Notice the 3rd stage

```
FROM builder AS publish
```

builder isn't an image pulled from a registry. It's the image we defined in stage 2, where we named the result of our build (sdk) image "builder". Docker build will create a named image we can later reference.

We can also copy the output from one image to another. This is the real power to compile our code with one base sdk image ([microsoft/aspnetcore-build](#)) , while creating a production image, based on an optimized runtime image. ([microsoft/aspnetcore](#)). Notice the line

```
COPY --from=publish /app .
```

This takes the /app directory from the publish image, and copies it to the working directory of the production image.

Breakdown Of Stages

The first stage provides the base of our optimized runtime image. Notice it derives from [microsoft/aspnetcore](#). This is where we'd specify additional production configurations, such as registry configurations, MSexec of additional components,... Any of those environment configurations you would hand off to your ops folks to prepare the VM.

The second stage is our build environment. [microsoft/aspnetcore-build](#) This includes everything we need to compile our code. From here, we have compiled binaries we can publish, or test. More on testing in a moment.

The 3rd stage derives from our builder. It takes the compiled output and "publishes" them, in .NET terms. Publishing simply means take all the output required to deploy your "app/service/component" and place it in a single directory. This would include your compiled binaries, graphics (images), javascript, etc.

The 4th stage is taking the published output, and placing it in the optimized image we defined in the first stage.

Why Is Publish Separate From Build?

You'll likely want to run unit tests to verify your compiled code, or the aggregate of the compiled code from multiple developers being merged together, continues to function as expected. To run unit tests, you could place the following stage between builder and publish.

```
FROM builder AS test
WORKDIR /src/Web.test
RUN dotnet test
```

If your tests fail, the build will cease to continue.

Why Is Base First?

You could argue this is simply the logical flow. We first define the base runtime image. Get the compiled output ready, and place it in the base image. However, it's more practical. While debugging your applications under Visual Studio Container Tools, VS will debug your code directly in the base image. When you hit F5, Visual Studio will compile the code on your dev machine. It will then volume mount the output to the built runtime image; the first stage. This way you can test any configurations you've made to your production image, such as registry configurations or otherwise.

When `docker build --target base` is executed, docker starts processing the dockerfile from the beginning, through the stage (target) defined. Since base is the first stage, we take the shortest path, making the F5 experience as fast as possible. If base was after compilation (builder), you'd have to wait for all the subsequent steps to complete. One of the perf optimizations we make with VS Container Tools is to take advantage of the Visual Studio compilations on your dev machine.

Multiple Projects and Solutions

The multi-stage dockerfile on the previous page is based on a Visual Studio solution. The full example can be found in this [github repo³¹](#) representing a Visual Studio solution with a Web and API project. The additional unit tests are under the AddingUnitTests branch.

The challenge with solutions is they represent a collection of projects. We often think of dockerfiles specific to a single image. While true, that single image may be the result of multiple "projects".

Consider the common pattern to develop shared dlls that may represent your data access layer, your logging component, your business logic, an authentication library, or a shipping calculation. The Web or API project may each reference these project(s). They each need to take the compiled output from those project and place them in the optimized image. This isn't to say we're building yet another monolithic application. There will certainly be additional services, such as checkout, authentication, profile management, communicating with the telco switch. But there's a balance. Microservices doesn't mean every shared piece of code is its own service.

If we look at the solution, we'll notice a few key aspects:

- Each project, which will represent a final docker image, has its own multi-stage dockerfile
- Shared component projects that are referenced by other resulting docker images do not have dockerfiles
- Each dockerfile assumes its context is the solution directory. This gives us the ability to copy in other projects
- There's a `docker-compose.yml` in the root of the solution. This gives us a single file to build multiple images, as well as specify build parameters, such as the image name:tag

```
Multi.sln
docker-compose.yml
[Api]
Dockerfile
[Web]
Dockerfile
```

We can now build the solution with a single docker command. We'll use `docker-compose` as our compose file has our image names as well as the individual build definitions

³¹ <https://github.com/SteveLasker/AspNetCoreMultiProject>

```
version: '3'

services:
  web:
    image: stevelas.azurecr.io/samples/multiproject/web
    build:
      context: .
      dockerfile: Web/Dockerfile

  api:
    image: stevelas.azurecr.io/samples/multiproject/api
    build:
      context: .
      dockerfile: Api/Dockerfile
```

Opening a command or powershell window, open the root directory of the solution:

```
PS> cd C:\Users\stevelas\Documents\GitHub\SteveLasker\AspNetCoreMultiProject
PS> docker-compose build
```

Elements of the output contain the following:

```
Building web
Step 1/17 : FROM microsoft/aspnetcore:2.0 AS base
Step 4/17 : FROM microsoft/aspnetcore-build:2.0 AS builder
Step 12/17 : FROM builder AS publish
Step 14/17 : FROM base AS production
Successfully tagged stevelas.azurecr.io/samples/multiproject/web:latest
Building api
Step 1/17 : FROM microsoft/aspnetcore:2.0 AS base
Step 4/17 : FROM microsoft/aspnetcore-build:2.0 AS builder
Step 6/17 : COPY *.sln .
Step 7/17 : COPY Api/Api.csproj Api/
Step 8/17 : RUN dotnet restore
Step 11/17 : RUN dotnet build -c Release -o /app
Step 12/17 : FROM builder AS publish
Step 13/17 : RUN dotnet publish -c Release -o /app
Step 14/17 : FROM base AS production
Step 16/17 : COPY --from=publish /app .
Successfully tagged stevelas.azurecr.io/samples/multiproject/api:latest
```

Coming Into Port

With multi-stage dockerfiles, we can now encapsulate our entire build process. By setting the context to our solution root, we can build multiple images, or build and aggregate shared components into images. By including our build environment in our multi-stage dockerfile, the development team owns the requirements to build their code, helping the CI/CD team to maintain a cattle farm without having to maintain individual build environments.

Labs

Enabling Continuous Integration with Azure Pipelines

In this hands-on lab, **Enabling Continuous Integration with Azure Pipelines**³², you will learn how to configure continuous integration with Azure Pipelines.

You will perform the following tasks:

- Creating a basic build pipeline from a template
- Tracking and reviewing a build
- Invoking a continuous integration build

✓ Note: You must have already completed the Lab Environment Setup in the Welcome section.

Integrating External Source Control with Azure Pipelines

In this lab, **Integrate Your GitHub Projects With Azure Pipelines**³³, you'll see how easy it is to set up Azure Pipelines with your GitHub projects and how you can start seeing benefits immediately.

- Install Azure Pipelines from the GitHub Marketplace.
- Integrate a GitHub project with an Azure DevOps pipeline.
- Track pull requests through the pipeline.

³² <https://www.azuredevopslabs.com/labs/azuredevops/continuousintegration/>

³³ <https://www.azuredevopslabs.com/labs/azuredevops/github-integration/>

Module Review and Takeaways

Module Review Questions

Checkbox

What are some advantages of Azure pipelines? Mark all that apply.

- Work with any language or platform - Python, Java, PHP, Ruby, C#, and Go
- work with open-source projects
- deploy to different types of targets at the same time
- integrate with Azure deployments - container registries, virtual machines, Azure services, or any on-premises or cloud target (Microsoft Azure, Google Cloud, or Amazon cloud services)
- build on Windows, Linux, or Mac machines
- integrate with GitHub

Suggested answer

What is a pipeline and why is it used?

Suggested answer

What are the two types of agents and how are they different?

Suggested answer

What is an agent pool and why would you use it?

Suggested answer

Name two ways to configure your Azure pipelines.

Suggested answer

Name the four pillars of continuous integration.

Suggested answer

You want to take your build server offline to make a configuration change. You want it to complete any build that it is currently processing, but you want to queue any new build requests. What should you do?

Suggested answer

You want to set a maximum time that builds can run for. Builds should not run for more than 5 minutes. What configuration change should you make?

Answers

Checkbox

What are some advantages of Azure pipelines? Mark all that apply.

- Work with any language or platform - Python, Java, PHP, Ruby, C#, and Go
- work with open-source projects
- deploy to different types of targets at the same time
- integrate with Azure deployments - container registries, virtual machines, Azure services, or any on-premises or cloud target (Microsoft Azure, Google Cloud, or Amazon cloud services)
- build on Windows, Linux, or Mac machines
- integrate with GitHub

What is a pipeline and why is it used?

A pipeline enables a constant flow of changes into production via an automated software production line. Pipelines create a repeatable, reliable and incrementally improving process for taking software from concept to customer.

What are the two types of agents and how are they different?

Automatically take care of maintenance and upgrades. Each time you run a pipeline, you get a fresh virtual machine. The virtual machine is discarded after one use. Self-hosted agents – You take care of maintenance and upgrades. Give you more control to install dependent software needed. You can install the agent on Linux, macOS, Windows machines, or even in a Linux Docker container.

What is an agent pool and why would you use it?

You can organize agents into agent pools. An agent pool defines the sharing boundary. In Azure Pipelines, agent pools are scoped to the Azure DevOps organization; so you can share an agent pool across projects.

Name two ways to configure your Azure pipelines.

Visual Designer & YAML file

Name the four pillars of continuous integration.

Continuous Integration relies on four key elements for successful implementation: a Version Control System, Package Management System, Continuous Integration System, and an Automated Build Process.

You want to take your build server offline to make a configuration change. You want it to complete any build that it is currently processing, but you want to queue any new build requests. What should you do?

You should pause the build. A paused build will not start new builds and will queue any new build requests.

You want to set a maximum time that builds can run for. Builds should not run for more than 5 minutes. What configuration change should you make?

You should change the build job timeout setting to 5 minutes. A blank value means unlimited.

Module 6 Managing Application Config and Secrets

Module Overview

Module Overview

Gone are the days of tossing a build over the wall and hoping that it works in production. Now development and operations are joined together as one in DevOps. DevOps accelerates the velocity with which products are deployed to customers. However, the catch with DevOps is that it moves fast, and security must move faster to keep up and make an impact. When products were built under the waterfall process, the release cycle was measured in years, so security process could take almost as long as it wanted. Face it, DevOps is here to stay, and it is not getting any slower. Application security must speed up to keep pace with the speed of business. Security automation is king under DevOps.

Learning Objectives

After completing this module, students will be able to:

- Manage application config and secrets
- Implement Tools for Managing security and compliance in pipeline

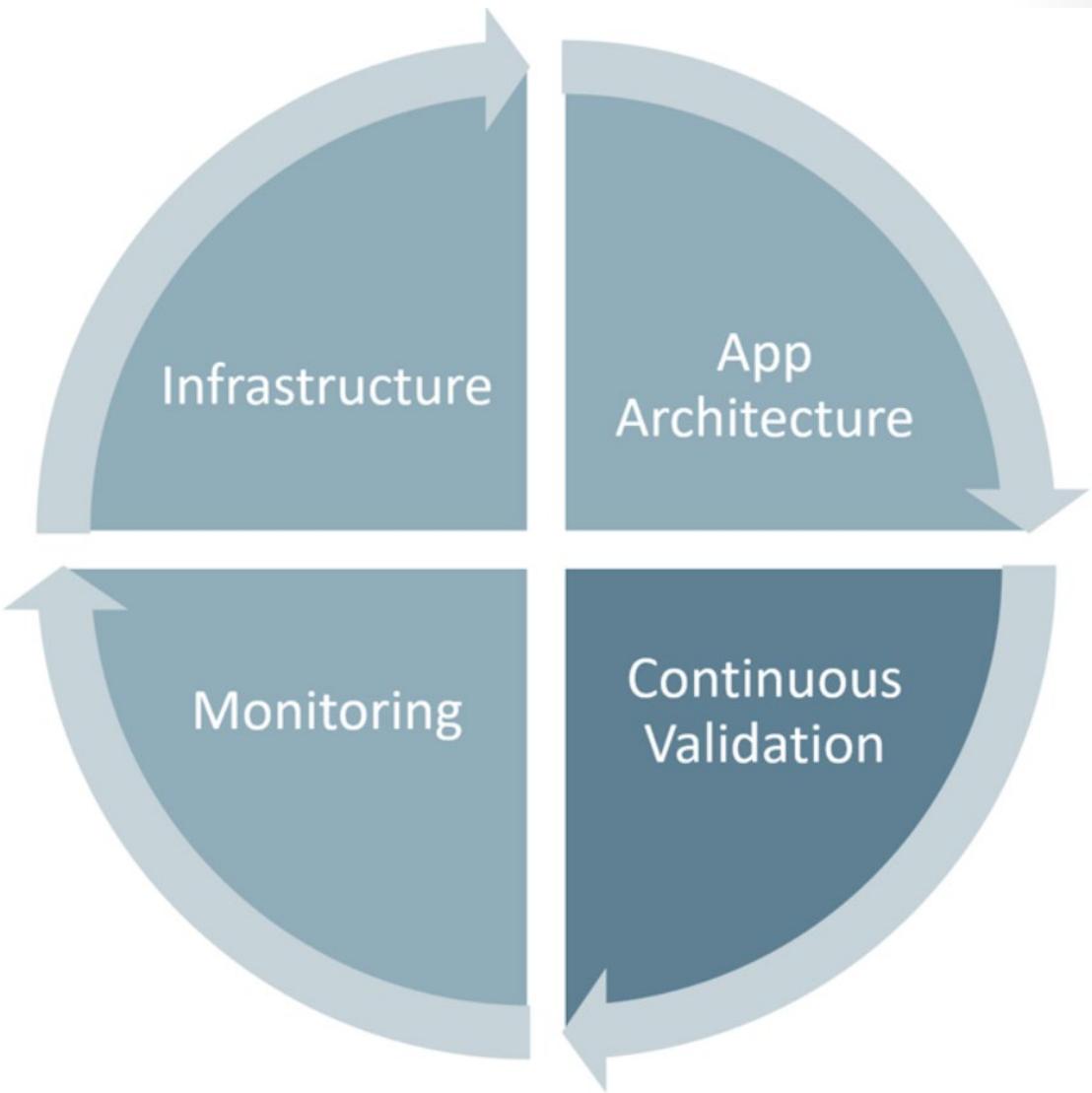
Introduction to Security

Introduction to Security

While DevOps way of working enables development teams to deploy applications faster, going faster over a cliff doesn't really help! Thanks to the cloud, DevOps teams have access to unprecedented infrastructure and scale. But that also means they can be approached by some of the most nefarious actors on the internet, as they risk the security of their business with every application deployment. Perimeter-class security is no longer viable in such a distributed environment, so now companies need to adapt a more micro-level security across application and infrastructure and have multiple lines of defence.

With continuous integration and continuous delivery, how do you ensure your applications are secure and stay secure? How can you find and fix security issues early in the process? This begins with practices commonly referred to as DevSecOps. DevSecOps incorporates the security team and their capabilities into your DevOps practices making security a responsibility of everyone on the team.

Security needs to shift from an afterthought to being evaluated at every step of the process. Securing applications is a continuous process that encompasses secure infrastructure, designing an architecture with layered security, continuous security validation, and monitoring for attacks.



Security is everyone's responsibility and needs to be looked at holistically across the application life cycle. In this module we'll discuss practical examples using real code for automating security tasks. We'll also see how continuous integration and deployment pipelines can accelerate the speed of security teams and improve collaboration with software development teams.

SQL Injection Attack

SQL Injection (SQLi) is a type of an injection attack that makes it possible to execute malicious SQL statements. These statements control a database server behind a web application. Attackers can use SQL Injection vulnerabilities to bypass application security measures. They can go around authentication and authorization of a web page or web application and retrieve the content of the entire SQL database. They can also use SQL Injection to add, modify, and delete records in the database.

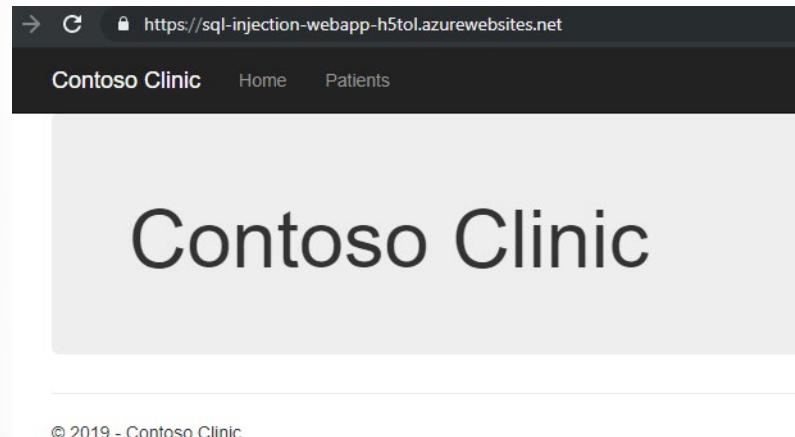
An SQL Injection vulnerability may affect any website or web application that uses an SQL database such as MySQL, Oracle, SQL Server, or others. Criminals may use it to gain unauthorized access to your sensitive data: customer information, personal data, trade secrets, intellectual property, and more. SQL Injection attacks are one of the oldest, most prevalent, and most dangerous web application vulnerabili-

ties. The OWASP organization (Open Web Application Security Project) lists injections in their OWASP Top 10 2017 document as the number one threat to web application security.

In this tutorial we'll simulate sql injection attack...

Getting Started

- Use the **SQL Injection ARM template here¹** to provision a web app and a sql database with known sql injection vulnerability
- Ensure you can browse to the 'Contoso Clinic' web app provisioned in your sql injection resource group



How it works

1. Navigate to the Patients view and in the search box type " ' " and hit enter. You'll see an error page with SQL exception indicating that the search box is feeding the text into a SQL statement

Server Error in '/' Application.

*Incorrect syntax near ''%'' OR [LastName] LIKE '%'.
Unclosed quotation mark after the character string ''%''.*

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.Data.SqlClient.SqlException: Incorrect syntax near ''%'' OR [LastName] LIKE '%'.
Unclosed quotation mark after the character string ''%''.

Source Error:

An unhandled exception was generated during the execution of the current web request. Information regarding the origin and location of the exception can be identified using the exception stack trace below.

Stack Trace:

[SqlException (0x80131904): Incorrect syntax near ''%'' OR [LastName] LIKE '%'.
Unclosed quotation mark after the character string ''%''.]

The helpful error message is enough to guess that the text in the search box is being appended into the sql statement.

2. Next try passing SQL statement 'AND FirstName = 'Kim'-- in the search box. You'll see that the results in the list below are filtered down to only show the entry with firstname Kim

¹ <https://azure.microsoft.com/en-us/resources/templates/101-sql-injection-attack-prevention/>

Patients

'AND FirstName = 'Kim'--				<input type="button" value="Search"/>	SQL Hints	
SSN	FirstName	LastName	MiddleName	StreetAddress	City	ZipCode
990-00-6818	Kim	Abercrombie		Tanger Factory	Branch	55056

© 2019 - Contoso Clinic

3. You can try to order the list by SSN by using this statement in the search box 'order by SSN--

Patients

<input type="text" value="order by SSN--"/>				<input type="button" value="Search"/>	SQL Hints
SSN	FirstName	LastName	MiddleName	StreetAddress	City
002-47-6040	Jean	Handley	P.	259826 Russell Rd. South	Kent
003-23-9305	Jeanie	Glenn	R.	9909 W. Ventura Boulevard	Camarillo

4. Now for the finale run this drop statement to drop the table that holds the information being displayed in this page... 'AND 1=1; Drop Table Patients --. Once the operation is complete, try and load the page. You'll see that the view errors out with an exception indicating that the dbo.patients table cannot be found

Invalid object name 'dbo.Patients'.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.Data.SqlClient.SqlException: Invalid object name 'dbo.Patients'.

Source Error:

An unhandled exception was generated during the execution of the current web request. Information regarding the origin and location of the exception can be identified using the exception stack trace below.

Stack Trace:

```
[SqlException (0x80131904): Invalid object name 'dbo.Patients'.]
```

There's more

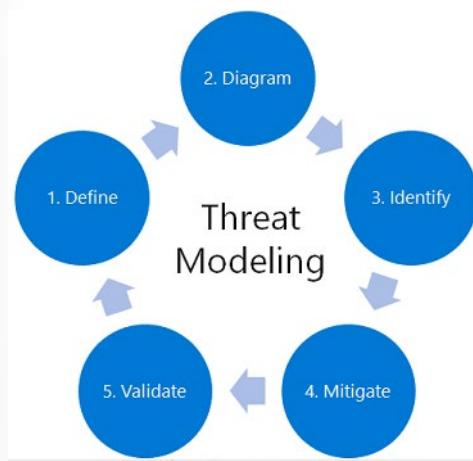
The Azure security centre team has other [playbooks](#)² you can look at to learn how vulnerabilities are exploited to trigger a virus attack and a DDoS attack.

² <https://azure.microsoft.com/en-gb/blog/enhance-your-devsecops-practices-with-azure-security-center-s-newest-playbooks/>

Implement Secure and Compliant Development Proces

Threat Modeling

Threat modeling is a core element of the Microsoft Security Development Lifecycle (SDL). It's an engineering technique you can use to help you identify threats, attacks, vulnerabilities, and countermeasures that could affect your application. You can use threat modeling to shape your application's design, meet your company's security objectives, and reduce risk. The tool with non-security experts in mind, making threat modeling easier for all developers by providing clear guidance on creating and analyzing threat models.



There are five major threat modeling steps:

- Defining security requirements.
- Creating an application diagram.
- Identifying threats.
- Mitigating threats.
- Validating that threats have been mitigated.

Threat modeling should be part of your routine development lifecycle, enabling you to progressively refine your threat model and further reduce risk.

Microsoft Threat Modeling Tool

The Microsoft Threat Modeling Tool makes threat modeling easier for all developers through a standard notation for visualizing system components, data flows, and security boundaries. It also helps threat modelers identify classes of threats they should consider based on the structure of their software design. The tool has been designed with non-security experts in mind, making threat modeling easier for all developers by providing clear guidance on creating and analyzing threat models.

The Threat Modeling Tool enables any developer or software architect to:

- Communicate about the security design of their systems.
- Analyze those designs for potential security issues using a proven methodology.

- Suggest and manage mitigations for security issues.

For more information, you can see:

- **Threat Modeling Tool feature overview³**
- **Microsoft Threat Modeling Tool⁴**

Demonstration Threat Modeling

Security cannot be a separate department in a silo. Security has to be part of DevOps, together they are called DevSecOps. The biggest weakness is not knowing the weakness in your solution. To re-mediate this, Microsoft has created a threat modelling tool, that helps you understand potential security vulnerabilities in your solution.

The Threat Modelling Tool is a core element of the Microsoft Security Development Life cycle (SDL). It allows software architects to identify and mitigate potential security issues early, when they are relatively easy and cost-effective to resolve. As a result, it greatly reduces the total cost of development. The tool has been designed with non-security experts in mind, making threat modelling easier for all developers by providing clear guidance on creating and analysing threat models.

The tool enables anyone to:

- Communicate about the security design of their systems
- Analyse those designs for potential security issues using a proven methodology
- Suggest and manage mitigation's for security issues

In this tutorial, we'll see how easy is it to use Threat Modelling tool to see potential vulnerabilities in your infrastructure solution that one should be thinking about when provisioning and deploying the Azure resources and the application solution into the solution...

Getting Started

- Download and install the **Threat Modelling tool⁵**

How to do it

1. Launch the Microsoft Threat Modelling Tool and choose the option to Create a Model...

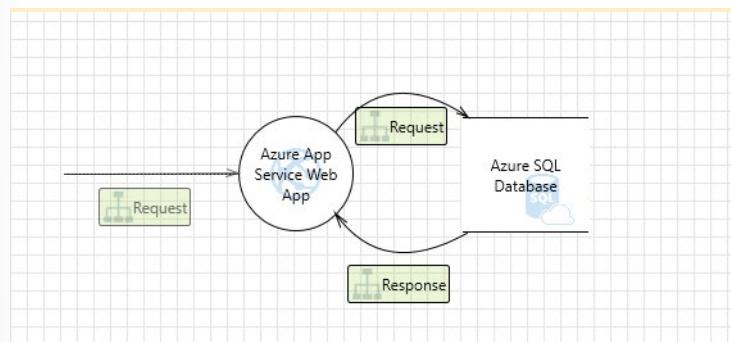
³ <https://docs.microsoft.com/en-us/azure/security/azure-security-threat-modeling-tool-feature-overview>

⁴ <https://blogs.msdn.microsoft.com/secdevblog/2018/09/12/microsoft-threat-modeling-tool-ga-release/>

⁵ <https://aka.ms/threatmodelingtool>



2. From the right panel search and add Azure App Service Web App, Azure SQL Database, link them up to show a request and response flow as demonstrated below...

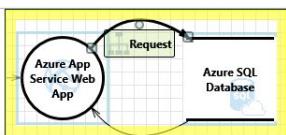


3. From the toolbar menu select View -> Analysis view, the analysis view will show you a full list of threats categorised by severity.

Short Description	Description	Interaction	Possible Mitigation(s)
A user subject gains increased capability or privilege by t...	Due to poorly configured account policies, adversary can...	Request	When possible use Azure Active Directory Authentication for connecting to SQL Database. Restrict access to Azure SQL Database instances by configuring server-level and database-level security settings. Clients connecting to an Azure SQL Database instance using a connection string should ensure...
A user subject gains increased capability or privilege by t...	An adversary can gain unauthorized access to Azure SQL...	Request	It is recommended to rotate user account passwords (e.g. those used in connection strings) regularly. Enable SQL Vulnerability Assessment to gain visibility into the security posture of your Azure SQL Database instances.
Information disclosure happens when the information contain...	An adversary can read confidential data due to weak auth...	Request	Enable Transparent Data Encryption (TDE) on Azure SQL Database instances to have data encrypted at rest. It is recommended to review permission and role assignments to ensure the users are granted...
Information disclosure happens when the information contain...	An adversary having access to the storage contains confi...	Request	Enable auditing on Azure SQL Database instances to track and log database events. After configuration, it is recommended to rotate user account passwords (e.g. those used in connection strings) regularly. Enable SQL Vulnerability Assessment to gain visibility into the security posture of your Azure SQL Database instances.
A user subject gains increased capability or privilege by t...	A compromised identity may permit more privileges, allowing...	Request	It is recommended to review permission and role assignments to ensure the users are granted...
Repudiation threats involve an adversary denying that someo...	An adversary can deny actions performed on Azure SQL Datab...	Request	Enable auditing on Azure SQL Database instances to track and log database events. After configuration, it is recommended to rotate user account passwords (e.g. those used in connection strings) regularly. Enable SQL Vulnerability Assessment to gain visibility into the security posture of your Azure SQL Database instances.
A user subject gains increased capability or privilege by t...	An adversary can gain long term, persistent access to Azure...	Request	It is recommended to rotate user account passwords (e.g. those used in connection strings) regularly. Enable SQL Vulnerability Assessment to gain visibility into the security posture of your Azure SQL Database instances.
A user subject gains increased capability or privilege by t...	An adversary may abuse weak Azure SQL Database authentication...	Request	Enable SQL Vulnerability Assessment to gain visibility into the security posture of your Azure SQL Database instances.
Denial of Service happens when the process or a datastor...	An adversary may block access to the application or data stor...	Response	Network level denial of service mitigations are automatically enabled as part of the Azure platform. Store secrets in secret storage solutions where possible, and rotate secrets on a regular schedule.
A user subject gains increased capability or privilege by t...	An adversary may gain long term persistent access to Azure...	Response	Restrict access to Azure App Service to selected networks (e.g. IP whitelisting, VNET integration).
A user subject gains increased capability or privilege by t...	An adversary may gain unauthorized access to Azure SQL Datab...	Response	

4. To generate a full report of the threats, from the toolbar menu select Reports -> Create full report, select a location to save the report.

A full report is generated with details of the threat along with the SLDC phase it applies to as well as possible mitigation and links to more details...



1. An adversary can gain unauthorized access to Azure SQL database due to weak account policy [State: Not Started] [Priority: High]

Category: Elevation of Privileges

Description: Due to poorly configured account policies, adversary can launch brute force attacks on Azure SQL Database

Justification: <no mitigation provided>

Possible When possible use Azure Active Directory Authentication for connecting to SQL Database. Refer: <https://aka.ms/tmt-th10a> Ensure that least-privileged accounts are used to

Mitigation(s): connect to Database server. Refer: <https://aka.ms/tmt-th10b> and <https://aka.ms/tmt-th10c>

SDL Phase: Implementation

2. An adversary can gain unauthorized access to Azure SQL DB instances due to weak network security configuration. [State: Not Started] [Priority: High]

Category: Elevation of Privileges

Description: An adversary can gain unauthorized access to Azure SQL DB instances due to weak network security configuration.

Justification: <no mitigation provided>

Possible Restrict access to Azure SQL Database instances by configuring server-level and database-level firewall rules to permit connections from selected networks (e.g. a virtual

Mitigation(s): network or a custom set of IP addresses) where possible. Refer:<https://aka.ms/tmt-th143>

SDL Phase: Implementation

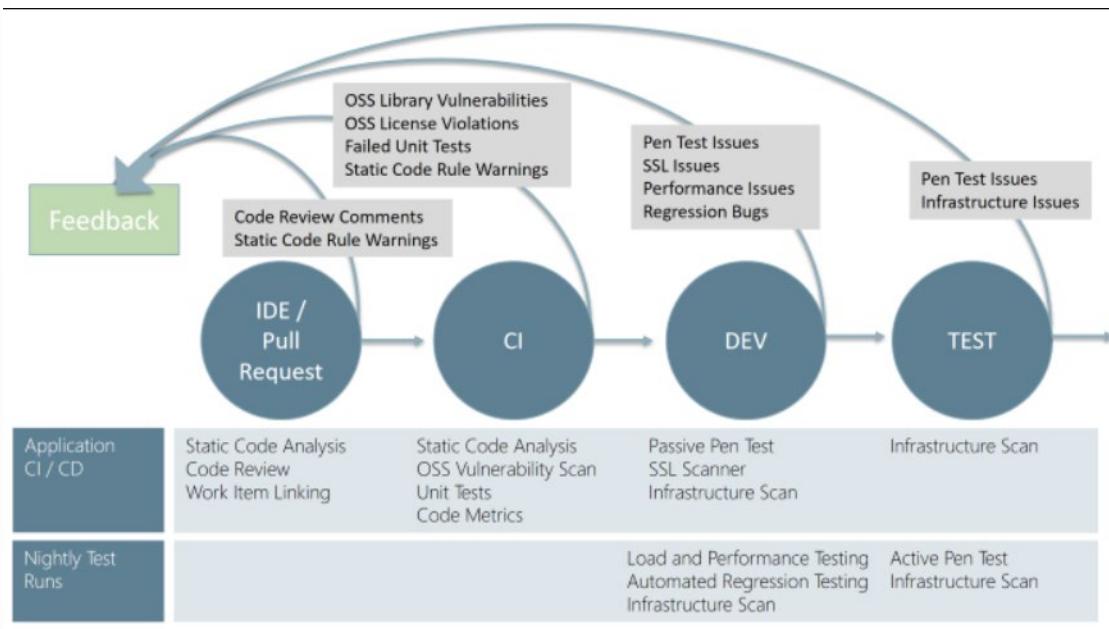
There's more

You can find a full list of threats used in the threat modelling tool [here⁶](#)

Key Validation Points

Continuous security validation should be added at each step from development through production to help ensure the application is always secure. The goal of this approach is to switch the conversation with the security team from approving each release to approving the CI/CD process and having the ability to monitor and audit the process at any time. When building greenfield applications, the diagram below highlights the key validation points in the CI/CD pipeline. Depending on your platform and where your application is at in its lifecycle, you may need to consider implementing the tools gradually. Especially if your product is mature and you haven't previously run any security validation against your site or application.

⁶ <https://docs.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats>



IDE / Pull Request

Validation in the CI/CD begins before the developer commits his or her code. Static code analysis tools in the IDE provide the first line of defense to help ensure that security vulnerabilities are not introduced into the CI/CD process. The process for committing code into a central repository should have controls to help prevent security vulnerabilities from being introduced. Using Git source control in Azure DevOps with branch policies provides a gated commit experience that can provide this validation. By enabling branch policies on the shared branch, a pull request is required to initiate the merge process and ensure that all defined controls are being executed. The pull request should require a code review, which is the one manual but important check for identifying new issues being introduced into your code. Along with this manual check, commits should be linked to work items for auditing why the code change was made and require a continuous integration (CI) build process to succeed before the push can be completed.

CI (Continuous Integration)

The CI build should be executed as part of the pull request (PR-CI) process and once the merge is complete. Typically, the primary difference between the two runs is that the PR-CI process doesn't need to do any of the packaging/staging that is done in the CI build. These CI builds should run static code analysis tests to ensure that the code is following all rules for both maintenance and security. Several tools can be used for this, such as one of the following:

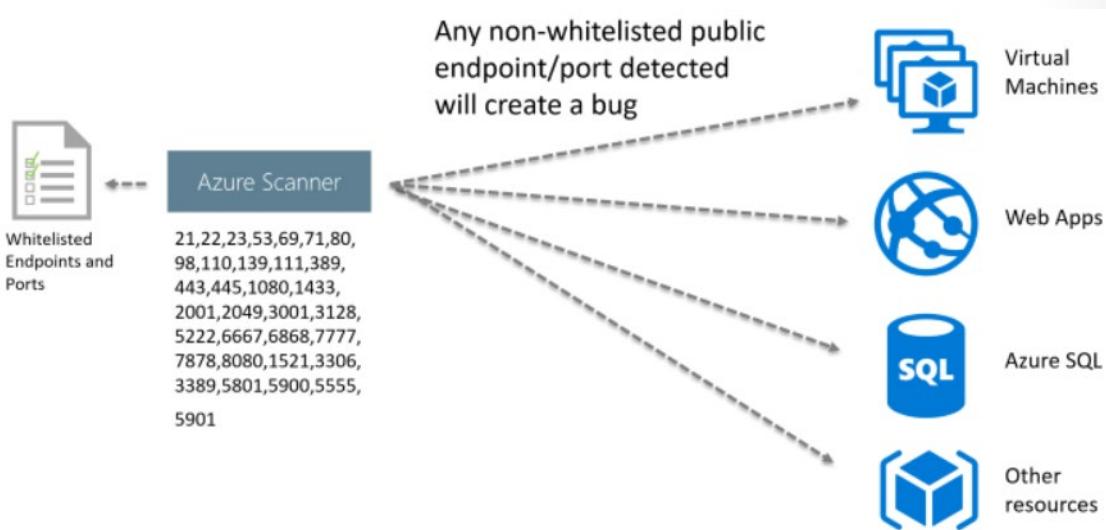
- SonarQube
- Visual Studio Code Analysis and the Roslyn Security Analyzers
- Checkmarx - A Static Application Security Testing (SAST) tool
- BinSkim - A binary static analysis tool that provides security and correctness results for Windows portable executables
- and many more

Many of the tools seamlessly integrate into the Azure Pipelines build process. Visit the Visual Studio Marketplace for more information on the integration capabilities of these tools.

In addition to code quality being verified with the CI build, two other tedious or ignored validations are scanning 3rd party packages for vulnerabilities and OSS license usage. Often when we ask about 3rd party package vulnerabilities and the licenses, the response is fear or uncertainty. Those organizations that are trying to manage 3rd party packages vulnerabilities and/or OSS licenses, explain that their process for doing so is tedious and manual. Fortunately, there are a couple of tools by WhiteSource Software that can make this identification process almost instantaneous. The tool runs through each build and reports all of the vulnerabilities and the licenses of the 3rd party packages. WhiteSource Bolt is a new option, which includes a 6-month license with your Visual Studio Subscription. Bolt provides a report of these items but doesn't include the advanced management and alerting capabilities that the full product offers. With new vulnerabilities being regularly discovered, your build reports could change even though your code doesn't. Checkmarx includes a similar WhiteSource Bolt integration so there could be some overlap between the two tools. See **Manage your open source usage and security as reported by your CI/CD pipeline⁷** for more information about WhiteSource and the Azure Pipelines integration.

Infrastructure Vulnerabilities

In addition to validating the application, the infrastructure should also be validated to check for any vulnerabilities. When using the public cloud such as Azure, deploying the application and shared infrastructure is easy, so it is important to validate that everything has been done securely. Azure includes many tools to help report and prevent these vulnerabilities including Security Center and Azure Policies. Also, we have set up a scanner that can ensure any public endpoints and ports have been whitelisted or else it will raise an infrastructure issue. This is run as part of the Network pipeline to provide immediate verification, but it also needs to be executed each night to ensure that there aren't any resources publicly exposed that should not be.



For more information, see also **Secure DevOps Kit (AzSK) CICD Extensions for Azure⁸**.

Application Deployment to Dev and Test

Once your code quality is verified, and the application is deployed to a lower environment like development or QA, the process should verify that there are not any security vulnerabilities in the running application. This can be accomplished by executing automated penetration test against the running

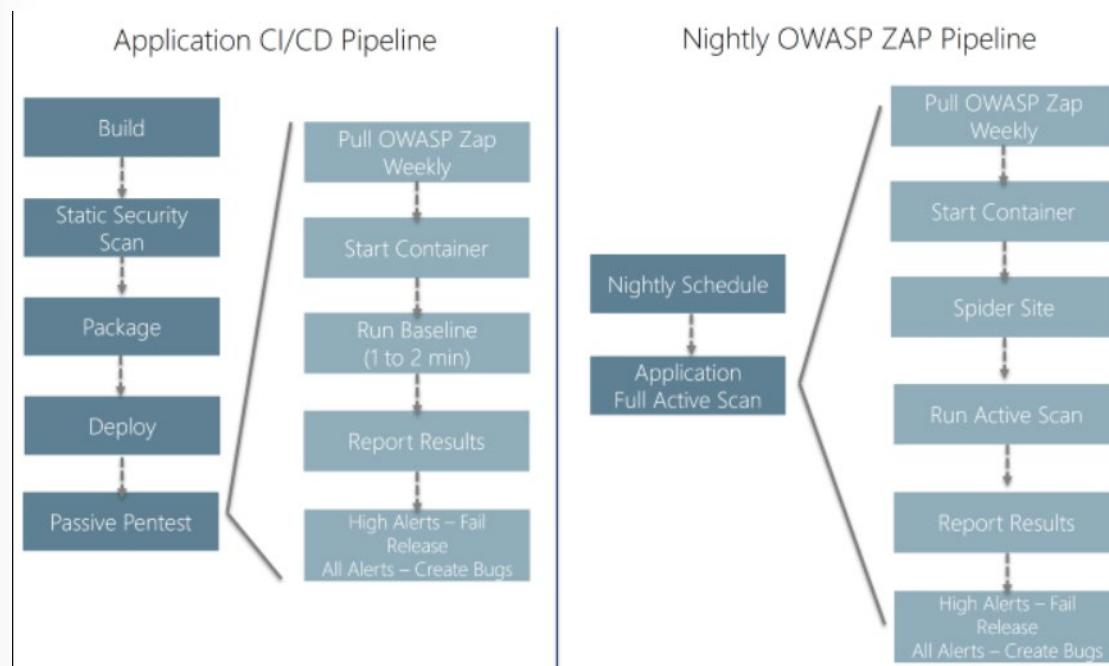
⁷ <https://blogs.msdn.microsoft.com/visualstudioalmrangers/2017/06/08/manage-your-open-source-usage-and-security-as-reported-by-your-cicd-pipeline/>

⁸ <https://marketplace.visualstudio.com/items?itemName=azsdktm.AzSDK-task>

application to scan it for vulnerabilities. There are different levels of tests that are categorized into passive tests and active tests. Passive tests scan the target site as is but don't try to manipulate the requests to expose additional vulnerabilities. These can run fast and are usually a good candidate for a CI process that you want to complete in a few minutes. Whereas the Active Scan can be used to simulate many techniques that hackers commonly use to attack websites. These tests can also be referred to dynamic or fuzz tests because the tests are often trying a large number of different combinations to see how the site reacts to verify that it doesn't reveal any information. These tests can run for much longer, and typically you don't want to cut these off at any particular time. These are better executed nightly as part of a separate Azure DevOps release.

One tool to consider for penetration testing is **OWASP ZAP**. OWASP is a worldwide not-for-profit organization dedicated to helping improve the quality of software. ZAP is a free penetration testing tool for beginners to professionals. ZAP includes an API and a weekly docker container image that can be integrated into your deployment process. Refer to the [oswa zap vsts extension⁹](#) repo for details on how to set up the integration. Here we're going to explain the benefits of including this into your process.

The application CI/CD pipeline should run within a few minutes, so you don't want to include any long-running processes. The baseline scan is designed to identify vulnerabilities within a couple of minutes making it a good option for the application CI/CD pipeline. The Nightly OWASP ZAP can spider the website and run the full Active Scan to evaluate the most combinations of possible vulnerabilities. OWASP ZAP can be installed on any machine in your network, but we like to use the OWASP Zap/Weekly docker container within Azure Container Services. This allows for the latest updates to the image and also allows being able to spin up multiple instances of the image so several applications within an enterprise can be scanned at the same time. The following figure outlines the steps for both the Application CI/CD pipeline and the longer running Nightly OWASP ZAP pipeline.



⁹ <https://github.com/deliveron/owasp-zap-vsts-extension>

Results and Bugs

Once the scans have completed, the Azure Pipelines release is updated with a report that includes the results and bugs are created in the team's backlog. Resolved bugs will close if the vulnerability has been fixed and move back into in-progress if the vulnerability still exists.

The benefit of using this is that the vulnerabilities are created as bugs that provide actionable work that can be tracked and measured. False positives can be suppressed using OWASP ZAP's context file, so only vulnerabilities that are true vulnerabilities are surfaced.

The screenshot shows the Azure Pipelines Board view for a release named "Release-40". The board has three columns: "Backlog" (empty), "New 61.5 h" (containing 6 items), and "Active 11 h" (empty). The "New" column contains the following items:

ID	Description	Type	Status
207810	Incomplete or No Cache-control and Pragma HTTP Header Set	Bug	New
207811	Cookie No HttpOnly Flag	Bug	New
207812	Cookie Without Secure Flag	Bug	New
207813	Web Browser XSS Protection Not Enabled	Bug	New
207814	X-Content-Type-Options Header Missing	Bug	New
207815	X-Frame-Options Header Not Set	Bug	New

Below the board, a summary section shows 6 total tests, 0% pass percentage, and 0s run duration. A "Test" section lists the following findings:

- 0/6 Passed - OWASP ZAP Security Tests
- 6 Failed:
 - Incomplete or No Cache-control and Pragma HTTP Header Set
 - Cookie No HttpOnly Flag
 - Cookie Without Secure Flag
 - Web Browser XSS Protection Not Enabled
 - X-Content-Type-Options Header Missing
 - X-Frame-Options Header Not Set

Even with continuous security validation running against every change to help ensure new vulnerabilities are not introduced, hackers are continuously changing their approaches, and new vulnerabilities are being discovered. Good monitoring tools allow you to help detect, prevent, and remediate issues discovered while your application is running in production. Azure provides a number of tools that provide detection, prevention, and alerting using rules, such as **OWASP Top 10¹⁰** and now even using machine learning to detect anomalies and unusual behavior to help identify attackers.

Minimize security vulnerabilities by taking a holistic and layered approach to security including secure infrastructure, application architecture, continuous validation, and monitoring. DevSecOps practices enable your entire team to incorporate these security capabilities throughout the entire lifecycle of your application. Establishing continuous security validation into your CI/CD pipeline can allow your application to stay secure while you are improving the deployment frequency to meet needs of your business to stay ahead of the competition.

¹⁰ <https://owasp.org/www-project-top-ten/>

Rethinking Application Config Data

Rethinking Application Config Data

Configuration information in files

The majority of application runtime environments include configuration information that's held in files deployed with the application. In some cases, it's possible to edit these files to change the application behavior after it's been deployed. However, changes to the configuration require the application be redeployed, often resulting in unacceptable downtime and other administrative overhead.

Local configuration files also limit the configuration to a single application, but sometimes it would be useful to share configuration settings across multiple applications. Examples include database connection strings, UI theme information, or the URLs of queues and storage used by a related set of applications. It's challenging to manage changes to local configurations across multiple running instances of the application, especially in a cloud-hosted scenario. It can result in instances using different configuration settings while the update is being deployed. In addition, updates to applications and components might require changes to configuration schemas. Many configuration systems don't support different versions of configuration information.

Example

It's 2:00 AM, Adam is done making all changes to his super awesome code piece, the tests are all running fine, he hit commit -> push -> all commits pushed successfully to git. Happily, he drives back home. Ten mins later he gets a call from the SecurityOps engineer, – Adam did you push the Secret Key to our public repo?

YIKES!! that blah.config file Adam thinks, how could I have forgotten to include that in .gitignore, the nightmare has already begun

We can surely try to blame Adam here for committing the sin of checking in sensitive secrets and not following the recommended practices of managing configuration files, but the bigger question is that if the underlying toolchain had abstracted out the configuration management from the developer, this fiasco would have never happened!

History

The virus was injected a long time ago...

Since the early days of .NET, there has been the notion of app.config and web.config files which provide a playground for developers to make their code flexible by moving common configuration into these files. When used effectively, these files are proven to be worthy of dynamic configuration changes. However a lot of time we see the misuse of what goes into these files. A common culprit is how samples and documentation have been written, most samples out in the web would usually leverage these config files for storing key elements such as ConnectionStrings, and even password. The values might be obfuscated but what we are telling developers is that "hey, this is a great place to push your secrets!". So, in a world where we are preaching using configuration files, we can't blame the developer for not managing the governance of it. Don't get me wrong; I am not challenging the use of Configuration here, it is an absolute need of any good implementation, I am instead debating the use of multiple json, XML, yaml files in maintaining configuration settings. Configs are great for ensuring the flexibility of the application, config files, however, in my opinion, are a pain to manage especially across environments.

A ray of hope: The DevOps movement

In recent years, we have seen a shift around following some great practices around effective DevOps and some great tools (Chef, Puppet) for managing Configuration for different languages. While these have helped to inject values during CI/CD pipeline and greatly simplified the management of configuration, the blah.config concept has not completely moved away. Frameworks like ASP.NET Core support the notion of appSettings.json across environments, the framework has made it very effective to use these across environments through interfaces like IHostingEnvironment and IConfiguration but we can do better.

Separation of Concerns

One of the key reasons we would want to move the configuration away from source control is to delineate responsibilities. Let's define some roles to elaborate this, none of these are new concepts but rather a high-level summary:

- **Configuration Custodian:** Responsible for generating and maintaining the life cycle of configuration values, these include CRUD on keys, ensuring the security of secrets, regeneration of keys and tokens, defining configuration settings such as Log levels for each environment. This role can be owned by operation engineers and security engineering while injecting configuration files through proper DevOps processes and CI/CD implementation. Note that they do not define the actual configuration but are custodians of their management.
- **Configuration Consumer:** Responsible for defining the schema (loose term) for the configuration that needs to be in place and then consuming the configuration values in the application or library code. This is the Dev. And Test teams, they should not be concerned about what the value of keys are rather what the capability of the key is, for example, a developer may need different ConnectionString in the application but does not need to know the actual value across different environments.
- **Configuration Store:** The underlying store that is leveraged to store the configuration, while this can be a simple file, but in a distributed application, this needs to be a reliable store that can work across environments. The store is responsible for persisting values that modify the behavior of the application per environment but are not sensitive and does not require any encryption or HSM modules.
- **Secret Store:** While you can store configuration and secrets together, it violates our separation of concern principle, so the recommendation is to leverage a separate store for persisting secrets. This allows a secure channel for sensitive configuration data such as ConnectionStrings, enables the operations team to have Credentials, Certificate, Token in one repository and minimizes the security risk in case the Configuration Store gets compromised.

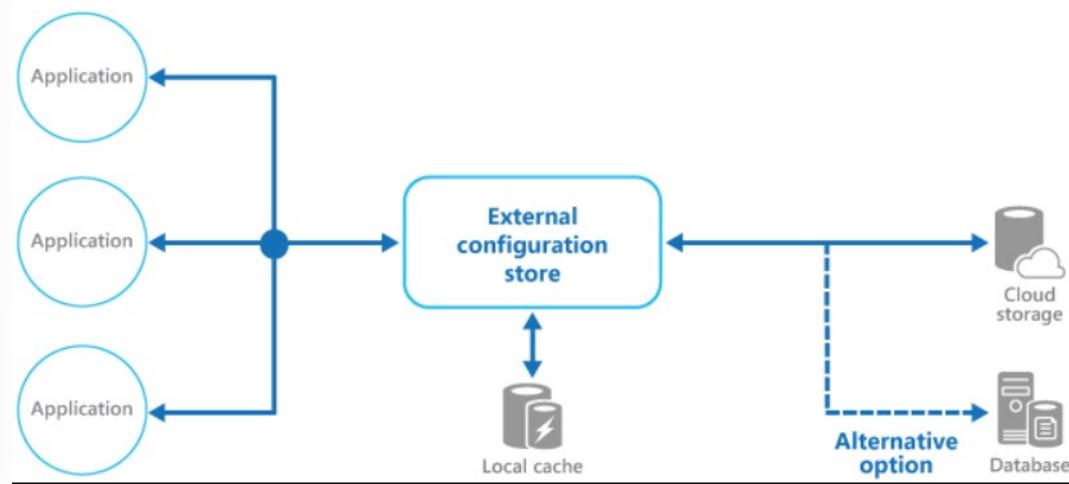
External Configuration Store pattern

Store the configuration information in external storage, and provide an interface that can be used to quickly and efficiently read and update configuration settings. The type of external store depends on the hosting and runtime environment of the application. In a cloud-hosted scenario it's typically a cloud-based storage service, but could be a hosted database or other system.

The backing store you choose for configuration information should have an interface that provides consistent and easy-to-use access. It should expose the information in a correctly typed and structured format. The implementation might also need to authorize users' access in order to protect configuration data, and be flexible enough to allow storage of multiple versions of the configuration (such as development, staging, or production, including multiple release versions of each one).

Many built-in configuration systems read the data when the application starts up, and cache the data in memory to provide fast access and minimize the impact on application performance. Depending on the

type of backing store used, and the latency of this store, it might be helpful to implement a caching mechanism within the external configuration store. For more information, see the Caching Guidance. The figure illustrates an overview of the External Configuration Store pattern with optional local cache.



This pattern is useful for:

- Configuration settings that are shared between multiple applications and application instances, or where a standard configuration must be enforced across multiple applications and application instances.
- A standard configuration system that doesn't support all of the required configuration settings, such as storing images or complex data types.
- As a complementary store for some of the settings for applications, perhaps allowing applications to override some or all of the centrally-stored settings.
- As a way to simplify administration of multiple applications, and optionally for monitoring use of configuration settings by logging some or all types of access to the configuration store.

Integrating Azure Key Vault with Azure Pipeline

Applications contain many secrets, such as connection strings, passwords, certificates, and tokens, which if leaked to unauthorized users can lead to a severe security breach. This can result in serious damage to the reputation of the organization and in compliance issues with different governing bodies. Azure Key Vault allows you to manage your organization's secrets and certificates in a centralized repository. The secrets and keys are further protected by Hardware Security Modules (HSMs). It also provides versioning of secrets, full traceability, and efficient permission management with access policies.

For more information on Azure Key Vault, visit [What is Azure Key Vault¹¹](https://docs.microsoft.com/en-us/azure/key-vault/key-vault-overview).

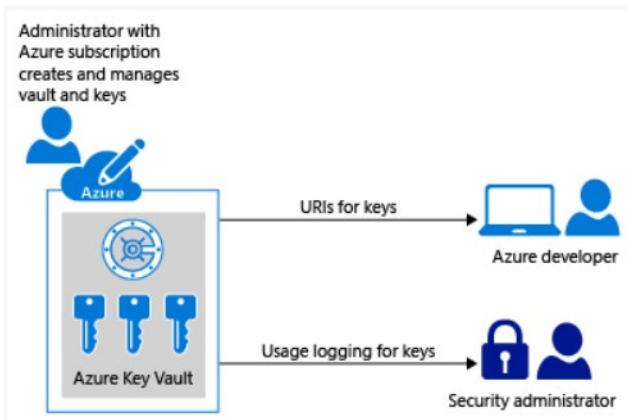
¹¹ <https://docs.microsoft.com/en-us/azure/key-vault/key-vault-overview>

Manage Secrets, Tokens, and Certificates

Manage Secrets, Tokens & Certificates

Azure Key Vault helps solve the following problems:

- **Secrets Management** - Azure Key Vault can be used to Securely store and tightly control access to tokens, passwords, certificates, API keys, and other secrets.
- **Key Management** - Azure Key Vault can also be used as a Key Management solution. Azure Key Vault makes it easy to create and control the encryption keys used to encrypt your data.
- **Certificate Management** - Azure Key Vault is also a service that lets you easily provision, manage, and deploy public and private Secure Sockets Layer/Transport Layer Security (SSL/TLS) certificates for use with Azure and your internal connected resources.
- **Store secrets backed by Hardware Security Modules** - The secrets and keys can be protected either by software or FIPS 140-2 Level 2 validates HSMs.



Why use Azure Key Vault?

##Centralize application secrets

Centralizing storage of application secrets in Azure Key Vault allows you to control their distribution. Key Vault greatly reduces the chances that secrets may be accidentally leaked. When using Key Vault, application developers no longer need to store security information in their application. This eliminates the need to make this information part of the code. For example, an application may need to connect to a database. Instead of storing the connection string in the app codes, store it securely in Key Vault.

Your applications can securely access the information they need by using URIs that allow them to retrieve specific versions of a secret after the application's key or secret is stored in Azure Key Vault. This happens without having to write custom code to protect any of the secret information.

Securely store secrets and keys

Secrets and keys are safeguarded by Azure, using industry-standard algorithms, key lengths, and hardware security modules (HSMs). The HSMs used are Federal Information Processing Standards (FIPS) 140-2 Level 2 validated.

Access to a key vault requires proper authentication and authorization before a caller (user or application) can get access. Authentication establishes the identity of the caller, while authorization determines the operations that they are allowed to perform.

Authentication is done via Azure Active Directory. Authorization may be done via role-based access control (RBAC) or Key Vault access policy. RBAC is used when dealing with the management of the vaults and key vault access policy is used when attempting to access data stored in a vault.

Azure Key Vaults may be either software- or hardware-HSM protected. For situations where you require added assurance you can import or generate keys in hardware security modules (HSMs) that never leave the HSM boundary. Microsoft uses Thales hardware security modules. You can use Thales tools to move a key from your HSM to Azure Key Vault.

Finally, Azure Key Vault is designed so that Microsoft does not see or extract your data.

Monitor access and use

Once you have created a couple of Key Vaults, you will want to monitor how and when your keys and secrets are being accessed. You can do this by enabling logging for Key Vault. You can configure Azure Key Vault to:

- Archive to a storage account.
- Stream to an event hub.
- Send the logs to Log Analytics.

You have control over your logs and you may secure them by restricting access and you may also delete logs that you no longer need.

Simplified administration of application secret

When storing valuable data, you must take several steps. Security information must be secured, it must follow a lifecycle, it must be highly available. Azure Key Vault simplifies a lot of this by:

- Removing the need for in-house knowledge of Hardware Security Modules
- Scaling up on short notice to meet your organization's usage spikes.
- Replicating the contents of your Key Vault within a region and to a secondary region. This ensures high availability and takes away the need of any action from the administrator to trigger the fail over.
- Providing standard Azure administration options via the portal, Azure CLI and PowerShell.
- Automating certain tasks on certificates that you purchase from Public CAs, such as enrollment and renewal.

In addition, Azure Key Vaults allow you to segregate application secrets. Applications may access only the vault that they are allowed to access, and they be limited to only perform specific operations. You can create an Azure Key Vault per application and restrict the secrets stored in a Key Vault to a specific application and team of developers.

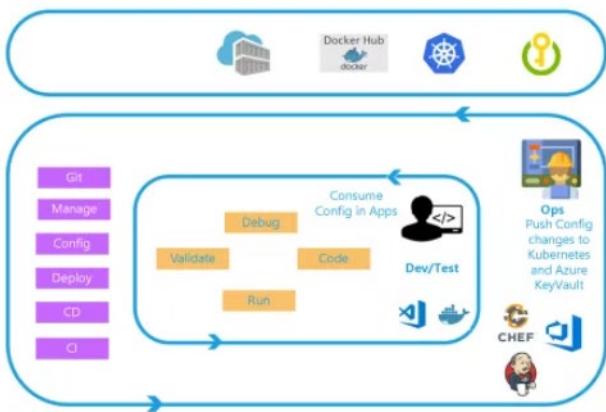
Integrate with other Azure services

As a secure store in Azure, Key Vault has been used to simplify scenarios like Azure Disk Encryption, the always encrypted functionality in SQL server and Azure SQL Database, Azure web apps. Key Vault itself can integrate with storage accounts, event hubs and log analytics.

DevOps Inner and Outer Loop

While you can store configuration and secrets together, it violates our separation of concern principle, so the recommendation is to leverage a separate store for persisting secrets. This allows a secure channel for sensitive configuration data such as ConnectionStrings, enables the operations team to have Credentials, Certificate, Token in one repository and minimizes the security risk in case the Configuration Store gets compromised.

The below diagram shows how these roles play together in a DevOps Inner loop and Outer loop. The inner loop is focused on the developer teams iterating over their solution development; they consume the configuration published by the outer loop. The Ops Engineer govern the Configuration management and push changes into Azure KeyVault and Kubernetes that are further isolated per environment.



Kubernetes and Azure Key Vault

So we have talked a lot about governance elements and the need to move out of configuration files, lets now use some of the magic available in Kubernetes and Azure Key Vault to implement this. In particular, we would be using the following capabilities of these technologies:

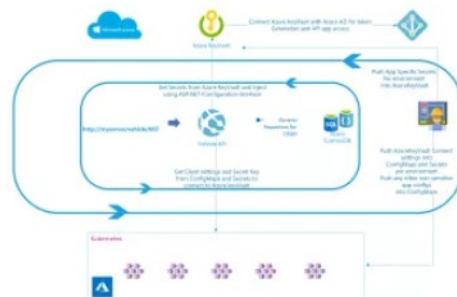
- Azure Key Vault Secret store
- Kubernetes ConfigMaps
- Kubernetes Secrets

Why not just use Kubernetes you may ask?

That's a valid question since Kubernetes supports both a ConfigMap store and Secret store. Remember our principle around the separation of concerns and how we can ensure enhanced security by separating out configuration from secrets. Having secrets in the same cluster as the configuration store can make them prone to higher risks. An additional benefit is Maintainability. Azure Key Vault gives the ability to provide a distributed "Secure Store as a Service" option that provides not just secret management but also Certificates and Key management as part of the service. The SecOps engineers can lock down access to the store without need of cluster admins permissions which allows for clear delineation of responsibilities and access.

The Scenario

We will be building a Vehicle microservice which provides CRUD operations for sending vehicle data to a CosmosDB document store. The sample micro-service needs to interact with the Configuration stores to get values such as connectionstring, database name, collection name, etc. We interact with Azure Key Vault for this purpose. Additionally, the application needs the Authentication token for Azure Key Vault itself, these details along with other Configuration will be stored in Kubernetes.



Mapping the above diagram to our roles and responsibilities earlier:

The Ops engineer/scripts are the Configuration Custodian and they are the only ones who work in the outer loop to manage all the configuration. They would have CI/CD scripts that would inject these configurations or use popular framework tools to enable the insertion during the build process.

The Vehicle API is the ASP.NET Core 2.0 application and is the Configuration Consumer here; the consumer is interested in getting the values without really worrying about what the value is and which environment it belongs to. The ASP.NET Core framework provides excellent support for this through its Configuration extensibility support. You can add as many providers as you like and they can be bound an IConfiguration object which provides access to all the configuration. In the below code snippet, we provide the configuration to be picked up from environment variables instead of a configuration file. The ASP.NET Core 2.0 framework also supports extensions to include Azure Key Vault as a configuration provider, and under the hood, the Azure Key Vault client allows for secure access to the values required by the application.

```
// add the environment variables to config
config.AddEnvironmentVariables();

// add azure key vault configuration using environment variables
var buildConfig = config.Build();

// get the key vault uri
var vaultUri = buildConfig["kvuri"].Replace("{vault-name}", buildConfig["vault"]);
// setup KeyVault store for getting configuration values
config.AddAzureKeyVault(vaultUri, buildConfig["clientId"], buildConfig["clientSecret"]);
```

AzureKeyVault is the Secret Store for all the secrets that are application specific. It allows for the creation of these secrets and also managing the lifecycle of them. It is recommended that you have a separate Azure KeyVault per environment to ensure isolation. The following command can be used to add a new configuration into KeyVault:

```
#Get a list of existing secrets
az keyvault secret list --vault-name -o table

#add a new secret to keyvault
az keyvault secret set -n MySecret --value MyValue --description "my custom
value" --vault-name
```

Kubernetes is the Configuration Store and serves multiple purposes here:

1. Creation of the ConfigMaps and Secret: Since we are injecting the Azure KeyVault authentication information using Kubernetes, the Ops engineer will provide these values using two constructs provided in the Kubernetes infrastructure. ConfigMaps and Secrets. The following shows how to add config maps and secrets from the Kubernetes command line:

```
kubectl create configmap vault --from-literal=vault=
kubectl create configmap kvuri --from-literal=kvuri=https://{{vault-name}}.{{vault.azure.net}}
kubectl create configmap clientid --from-literal=clientId=
kubectl create secret generic clientsecret --from-literal=clientSecret=
```

The clientsecret is the only piece of secure information we store in Kubernetes, all the application specific secrets are stored in Azure KeyVault. This is comparatively safer since the above scripts do not need to go in the same git repo. (so we don't check them in by mistake) and can be managed separately. We still control the expiry of this secret using Azure KeyVault, so the Security engineer still has full control over access and permissions.

1. Injecting Values into the Container: During runtime, Kubernetes will automatically push the above values as environment variables for the deployed containers, so the system does not need to worry about loading them from a configuration file. The Kubernetes configuration for the deployment looks like below. As you would notice, we only provide a reference to the ConfigMaps and Secret that have been created instead of punching in the actual values.

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: vehicle-api-deploy #name for the deployment
  labels:
    app: vehicle-api #label that will be used to map the service, this tag
    is very important
spec:
  replicas: 1
  selector:
    matchLabels:
      app: vehicle-api #label that will be used to map the service, this tag
      is very important
  template:
    metadata:
      labels:
        app: vehicle-api #label that will be used to map the service, this tag
        is very important
    spec:
```

```
containers:
  - name: vehicleapi #name for the container configuration
    image: <yourdockerhub>/<youdockerimage>:<youdockertagversion> #
    **CHANGE THIS: the tag for the container to be deployed
    imagePullPolicy: Always #getting latest image on each deployment
    ports:
      - containerPort: 80 #map to port 80 in the docker container
    env: #set environment variables for the docker container using
    configMaps and Secret Keys
      - name: clientId
        valueFrom:
          configMapKeyRef:
            name: clientid
            key: clientId
      - name: kvuri
        valueFrom:
          configMapKeyRef:
            name: kvuri
            key: kvuri
      - name: vault
        valueFrom:
          configMapKeyRef:
            name: vault
            key: vault
      - name: clientsecret
        valueFrom:
          secretKeyRef:
            name: clientsecret
            key: clientSecret
    imagePullSecrets: #secret to get details of private repo, disable
this if using public docker repo
      - name: regsecret
```

Implement Tools for Managing Security and Compliance in a Pipeline

SonarCloud

Technical debt can be classified as the measure between the codebase's current state and an optimal state. Technical debt saps productivity by making code hard to understand, easy to break, and difficult to validate, in turn creating unplanned work, ultimately blocking progress. Technical debt is inevitable! It starts small and grows over time through rushed changes, lack of context, and lack of discipline. Organizations often find that more than 50% of their capacity is sapped by technical debt. The hardest part of fixing technical debt is knowing where to start. SonarQube is an open source platform that is the de facto solution for understanding and managing technical debt. In this recipe, we'll learn how to leverage SonarQube in a build pipeline to identify technical debt.

Getting ready

SonarQube is an open platform to manage code quality. Originally famous in the Java community, SonarQube now supports over 20 programming languages. The joint investments made by Microsoft and SonarSource make SonarQube easier to integrate in Pipelines and better at analyzing .NET-based applications. You can read more about the capabilities offered by SonarQube here: <https://www.sonarqube.org/>. SonarSource, the company behind SonarQube offers a hosted SonarQube environment called as SonarCloud.

Implement Continuous Security Validation

Today developers don't hesitate to use components that are available in public package sources (such as npm or NuGet). With the aim of faster delivery and better productivity, using open source software (OSS) components is encouraged across many organisations. However, as the dependency on these third-party OSS components increases, the risk of security vulnerabilities or hidden license requirements also increases compliance issues.

For a business, this is critical, as issues related to compliance, liabilities and customer personally identifiable information (PII) can cause a great deal of privacy and security concerns. Identifying such issues early on in the release cycle gives you an advanced warning and allows you enough time to fix the issues. There are many tools such as WhiteSource, Veracode, and Checkmarx that are available, can scan for these vulnerabilities within the build and release pipelines.

Securing Infrastructure with AzSk

With the adoption of infrastructure as code and development teams taking on more of the Ops activities, organisations using Azure for enterprise hosting are practically at risk of breach with every new release they roll out to these environments... In this tutorial we'll walk through AzSDK Security Verification Tests that can be setup in the CI/CD pipeline using Azure DevOps to automate the inspection of your infrastructure in Azure and block releases that can potentially compromise your infrastructure.

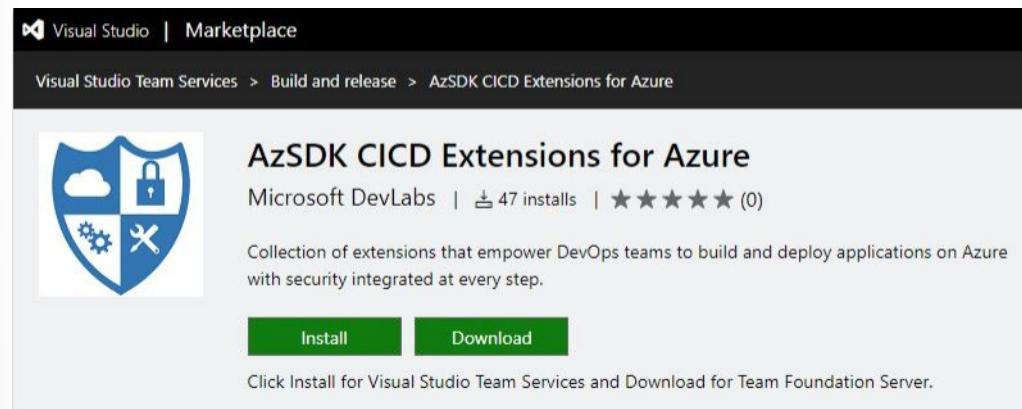
The DevOps way of working coupled with the use of cloud technologies has practically removed the biggest barriers from the software delivery life-cycle... Development teams don't care about security as much as they should. It's hard to blame the development teams though, the tools used by security are not easy to understand... The reports are long and hard to interpret... The approach of compliance driven security doesn't practically fit into the fast pace of software delivery we are used to today.

DevSecOps helps bring a fresh perspective by introducing a culture of making everyone accountable for security, using automation to move the process of inspection left and overall looking at security with a 360 lens.

Anyone doing cloud at scale is likely to have multiple Azure subscriptions with hundreds of resource groups in Azure dedicated to the application in question. Resource Groups comprising of resources such as Web App's, Azure Functions, Blob Storage, Redis Cache, App Insights, Service Bus, SQL warehouse among some other Azure resource types. The ratio of security consultants to the number of releases makes it practically impossible for security to inspect the changes to these resource types to guarantee compliance of best practices.

Getting started

- Start by installing the AzSDK extension from the Azure DevOps [marketplace](#)¹²



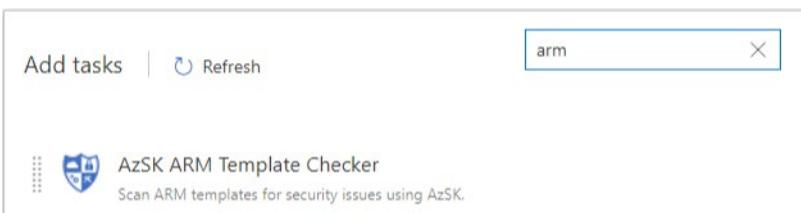
How to do it

The AzSk extension gives you the ability to both scan ARM templates to identify shortcoming in your ARM templates and also gives you the ability to scan actual Azure resources provisioned in an Azure subscription for compliance to best practices. We'll cover both in this tutorial.

Scanning Azure Resource Manager (ARM) Templates for best practices

- Create a build definition that is mapped to the git repository where you store your ARM templates
- Next add the AzSK ARM Template Checker task to the build pipeline

¹² <https://marketplace.visualstudio.com/items?itemName=azsdkts.AzSDK-task>



3. Next, configure the task - To start scanning, you just need to provide the root folder where you have ARM templates (and optionally mark it to scan recursively).

A screenshot of the Azure DevOps pipeline editor. A new task, 'Scan for security issues in ARM templates' (AzSK ARM Template Checker), has been added to the 'Phase 1' section. The task is configured with the following settings:

- Display name: Scan for security issues in ARM templates
- ARM template file path or folder path: src/health
- Recurse: Checked
- Skip Controls From File: Unchecked
- Exclude Files: Unchecked
- Do not auto-update AzSK: Unchecked

The pipeline also shows a 'Get sources' step and a 'Process' step.

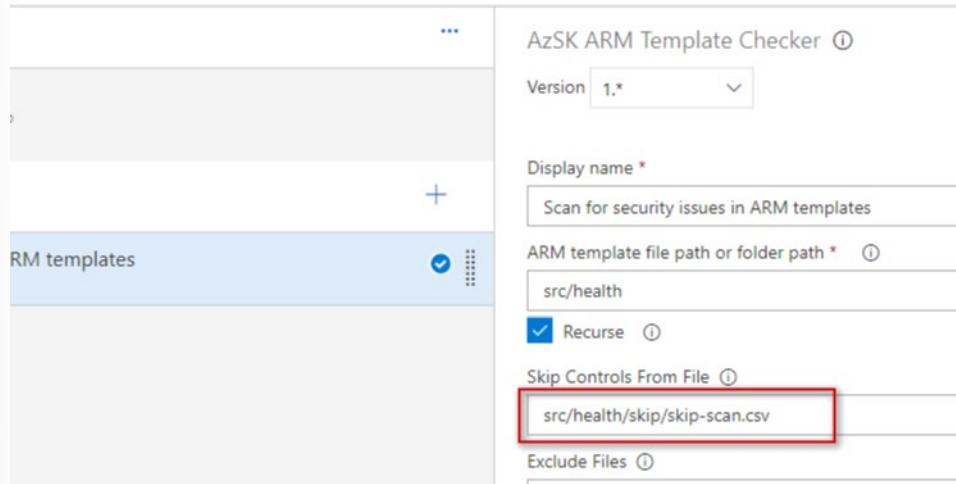
4. Run the build and let the build complete. You will see glimpse of the scan in the build output itself.

```
0.6529638Z Failed: [Azure_AppService_Config_Disable_Web_Sockets]
0.6529841Z Failed: [Azure_AppService_BCDR_Use_AlwaysOn]
0.6530042Z Failed: [Azure_AppService_Deploy_Use_Latest_Version]
0.6530272Z Failed: [Azure_AppService_Audit_Enable_Logging_and_Monitoring]
0.6530496Z Failed: [Azure_AppService_Audit_Enable_Logging_and_Monitoring]
0.6530704Z Failed: [Azure_AppService_Audit_Enable_Logging_and_Monitoring]
0.6530941Z Failed: [Azure_AppService_DP_Dont_Allow_HTTP_Access]
0.6531149Z Failed: [Azure_AppService_AuthN_Use_AAD_for_Client_AuthN]
0.6531354Z Failed: [Azure_AppService_AuthN_Use_AAD_for_Client_AuthN]
0.6531589Z -----
0.6531963Z Summary Total Failed
0.6532129Z -----
0.6532319Z High     8     8
0.6532486Z Medium   12    12
0.6532654Z Low      2     2
0.6532958Z -----
0.6533123Z Total    22    22
0.6533287Z -----
```

5. Once the build completes, you will find that task has attached all the results to the build log. You will be surprised to see the issues you find in your ARM templates.

FeatureName	Status	Supporter	Severity	PropertyPath	LineNum	CurrentValue	ExpectedValue	Resource	Description
AppService Failed	Microsoft.	Medium	Not found	-1	\$sku.cap>resources	62 "[paramet\$sku.cap>resources	30 greaterThan resources	App Service must be deployed on a minimum of two instances to ensure availability	
AppService Failed	Microsoft.	High	Not found	-1	\$property['False']	30 remoteDebugging	30 resources	30 Remote debugging must be turned off for App Service	
AppService Failed	Microsoft.	Medium	Not found	-1	\$property['False']	30 webSockets	30 resources	30 Web Sockets should be disabled for App Service	
AppService Failed	Microsoft.	Medium	Not found	-1	\$property['True']	30 alwaysOn	30 resources	30 'Always On' should be configured for App Service	
AppService Failed	Microsoft.	Low	Not found	-1	\$propertyAllowV4	30 .NETFrameworkVersion	30 resources	30 The latest version of .NET framework version should be used for App Service	
AppService Failed	Microsoft.	Medium	Not found	-1	\$property['True']	30 auditingMonitoring	30 resources	30 Auditing and Monitoring must be enabled for App Service	
AppService Failed	Microsoft.	Medium	Not found	-1	\$property['True']	30 auditingMonitoring	30 resources	30 Auditing and Monitoring must be enabled for App Service	
AppService Failed	Microsoft.	High	Not found	-1	\$property['True']	30 httpsOnly	30 resources	30 App Service must only be accessible over HTTPS	
AppService Failed	Microsoft.	High	Not found	-1	\$property['True']	30 aadAuth	30 credentials	30 App Service must authenticate users using Azure Active Directory backed credentials	
AppService Failed	Microsoft.	High	Not found	-1	\$propertyNonNull	30 credentials	30 resources	30 App Service must authenticate users using Azure Active Directory backed credentials	
Storage Failed	Microsoft.	Medium	Not found	-1	\$property['True']	95 httpsProtocol	95 resources	95 HTTPS protocol must be used for accessing Storage Account resources	
AppService Failed	Microsoft.	Medium	resources	62 "[paramet\$sku.cap>resources	49 availability	49 resources	49 resources	49 App Service must be deployed on a minimum of two instances to ensure availability	
AppService Failed	Microsoft.	High	Not found	-1	\$property['False']	49 remoteDebugging	49 resources	49 Remote debugging must be turned off for App Service	
AppService Failed	Microsoft.	Medium	Not found	-1	\$property['False']	49 webSockets	49 resources	49 Web Sockets should be disabled for App Service	
AppService Failed	Microsoft.	Medium	Not found	-1	\$property['True']	49 alwaysOn	49 resources	49 'Always On' should be configured for App Service	

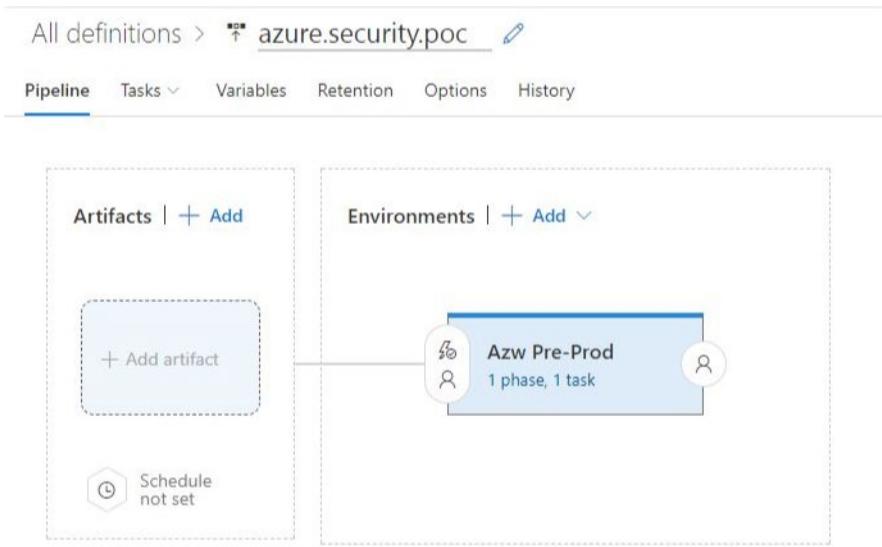
6. Occasionally you will find issues which you have decided as safe to ignore. The task allows you to skip such failures from the scan so that you will not get alerted as a security issue or cause in build failures. The way you configure this is simple - Use the generated csv file and keep only entries you need to ignore from the scan and commit to your repository as a csv file. Then specify this file in the task as below.



The task currently scans App Service, Storage, SQL, CDN, Traffic Manager, Document DB, Redis Cache, and Data Lake services only.

Scanning Azure Resources for best practices

- Create a new release Azure pipeline, add an environment

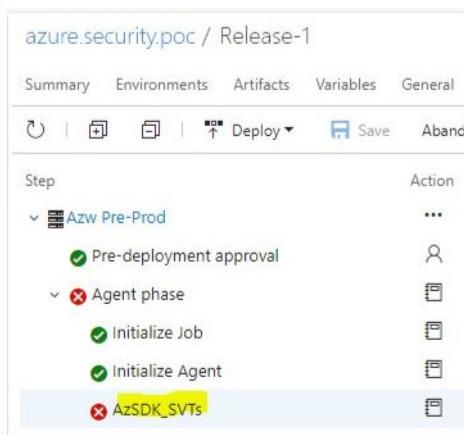


- In the newly added environment add the azSdk task and configure the Azure DevOps SPN for the Azure subscription, specify the name of the Azure resource group you wish to inspect as well as the Azure subscription id the resource group resides in.

The screenshot shows the 'Tasks' tab for the 'Azw Pre-Prod' deployment process. The tasks listed are 'Agent phase' and 'AzSDK_SVTs'. The 'AzSDK_SVTs' task is selected, and its configuration pane is displayed on the right. The configuration fields include:

- Display name: AzSDK_SVTs
- AzureRM Subscription: PreProd
- Select Parameter Set: ResourceGroupName
- ResourceGroup Names: azsu-rg-preprod-[REDACTED]
- Subscription ID: [REDACTED]
- Enable OMS Logging

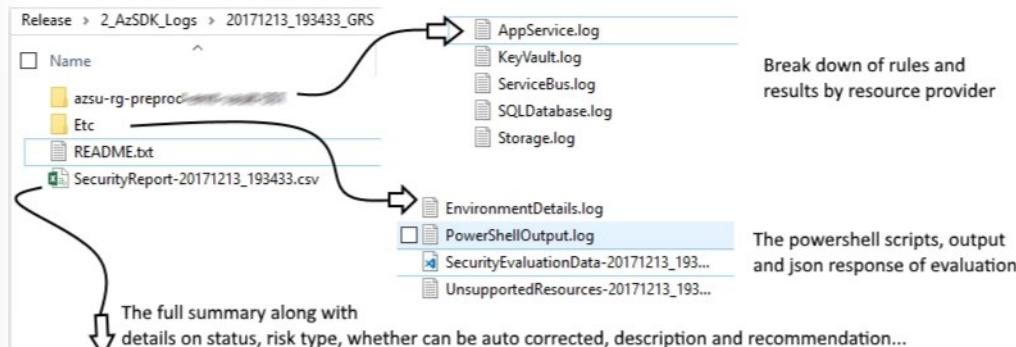
- Run the release pipeline and wait for the release to complete... The default security policies are evaluated, you have the option of customizing or creating your subset... If the setup isn't compliant the release pipeline will fail by default...



Now review the detailed logs from the inspection...

ControlID	Status	FeatureName	ResourceType	Criticality	ControlsSupported	Description	Recommendation
Azure_Storage_AuthN_Dont_Allow_Anonymous	Passed	Storage	azsu-1azsub	High	Yes	The Access Type for containers must not be set to 'Run comm	
Azure_Storage_Audit_Issue_Alert_AuthN_Req	Failed	Storage	azsu-1azsub	High	Yes	Alert rules must be configured for tracking anonym	Run comm
Azure_Storage_Deploy_Use_Geo_Redundant	Passed	Storage	azsu-1azsub	High	Yes	Use geo-redundant Storage Accounts	Note: Ena
Azure_Storage_DP_Encrypt_At_Rest_Blob	Passed	Storage	azsu-1azsub	High	Yes	Sensitive data in Storage Blob must be encrypted	Run comm
Azure_Storage_Audit_AuthN_Requests	Failed	Storage	azsu-1azsub	Medium	Yes	Storage Account must be configured to log and mo	Run comm
Azure_Storage_DP_Encrypt_In_Transit	Failed	Storage	azsu-1azsub	High	Yes	HTTPS protocol must be used for accessing Storage	Run comm
Azure_Storage_AuthZ_Use_IP_ACL	Manual	Storage	azsu-1azsub	Medium	No	Use IP-restrictions in SAS tokens to only permit ac	Restrict st
Azure_Storage_AuthZ_Clients_Use_SAS	Manual	Storage	azsu-1azsub	High	No	End user/client apps should access Storage Accou	Do not us
Azure_Storage_DP_Rotate_Keys	Manual	Storage	azsu-1azsub	Medium	No	Storage Account keys must be rotated periodically	Rotate Stc
Azure_Storage_AuthZ_Allow_Limited_Access_to_Ser	Manual	Storage	azsu-1azsub	High	No	Use Stored Access Policies with least privileges ne	Create a S
Azure_KeyVault_AuthN_Use_Cert_Auth_for_Apps	Error	KeyVault	azsu-1azsu-	High	No	Azure Active Directory applications, which have ac	Remove a

The logs give you the full picture! The extension gives you a criticality rating and also tells you if the issue can be auto fixed, a description and recommendation for the specific line item as well...



ControlID	Status	FeatureName	ResourceType	Criticality	ControlsSupported	Description	Recommendation
Azure_Storage_AuthN_Dont_Allow_Anonymous	Passed	Storage	azsu-1azsub	High	Yes	The Access Type for containers must not be set to 'Run comm	
Azure_Storage_Audit_Issue_Alert_AuthN_Req	Failed	Storage	azsu-1azsub	High	Yes	Alert rules must be configured for tracking anonym	Run comm
Azure_Storage_Deploy_Use_Geo_Redundant	Passed	Storage	azsu-1azsub	High	Yes	Use geo-redundant Storage Accounts	Note: Ena
Azure_Storage_DP_Encrypt_At_Rest_Blob	Passed	Storage	azsu-1azsub	High	Yes	Sensitive data in Storage Blob must be encrypted	Run comm
Azure_Storage_Audit_AuthN_Requests	Failed	Storage	azsu-1azsub	Medium	Yes	Storage Account must be configured to log and mo	Run comm
Azure_Storage_DP_Encrypt_In_Transit	Failed	Storage	azsu-1azsub	High	Yes	HTTPS protocol must be used for accessing Storage	Run comm
Azure_Storage_AuthZ_Use_IP_ACL	Manual	Storage	azsu-1azsub	Medium	No	Use IP-restrictions in SAS tokens to only permit ac	Restrict st
Azure_Storage_AuthZ_Clients_Use_SAS	Manual	Storage	azsu-1azsub	High	No	End user/client apps should access Storage Accou	Do not us
Azure_Storage_DP_Rotate_Keys	Manual	Storage	azsu-1azsub	Medium	No	Storage Account keys must be rotated periodically	Rotate Stc
Azure_Storage_AuthZ_Allow_Limited_Access_to_Ser	Manual	Storage	azsu-1azsub	High	No	Use Stored Access Policies with least privileges ne	Create a S
Azure_KeyVault_AuthN_Use_Cert_Auth_for_Apps	Error	KeyVault	azsu-1azsu-	High	No	Azure Active Directory applications, which have ac	Remove a

Summary

Pushing Security left does not mean using the old ways of security compliance checks earlier in the life-cycle, instead it is an opportunity to leverage the DevOps mindset to innovate and automate the security inspection to allow development teams to release features with confidence.

Azure Policy and Governance

If you don't have standards in IT, things can get out of control and there's no change in cloud. In fact, there's more of a need for governance than ever, if you don't have it in cloud it could cause allot of issues, excessive costs, issues supporting and a bad cloud experience to name a few. Azure Policy is essentially designed to help set those standards on Azure use (using policies), making sure it's used in the right way highlighting issues (using compliance) and help fix those issues (using remediation).

You can manually create policies (to check Azure resources are configured in a certain way) or you can use the growing number of pre-built policies. In this tutorial we'll look at a simple Azure Policy to define allowed Azure regions for resource provisioning.

How to do it

1. Launch the Azure Portal and create a new resource group called az-rg-labs-azurepolicy-001
2. From within Azure Portal, open Policy, from the left pane click on Definitions.

The screenshot shows the Azure Policy Definitions page. On the left, a navigation sidebar lists sections: Overview, Getting started, Join Preview, Compliance, Remediation, Authoring (with Assignments and Definitions), Blueprints (with Blueprints (preview)), and Resources (with Resource Graph (preview)). The main area displays the following metrics:

- Overall resource compliance: 100%
- Non-compliant initiatives: 0 out of 0
- Non-compliant policies: 0 out of 0
- Non-compliant resources: 0 out of 0

Below these metrics is a table header with columns: NAME, SCOPE, COMPLIANCE STATE, COMPLIANCE, and NON-COMPLIANT RES... A link "View all" is provided under the NAME column. At the bottom, a section titled "ASSIGNMENTS BY COMPLIANCE (LAST 7 DAYS)" is shown.

Definitions are effectively the rules you want to impose. You can use the built in policies, duplicate and edit them, or create your own from various templates like those on [GitHub](#)¹³

¹³ <https://github.com/Azure/azure-policy>

The screenshot shows the 'Policy - Definitions' blade in the Azure portal. On the left, there's a navigation menu with options like Overview, Getting started, Join Preview, Compliance, Remediation, Authoring, Assignments, and Definitions. The 'Definitions' option is selected. The main area has a search bar and filters for Scope (Microsoft Azure S...), Definition type (All definition types), Type (All types), Category (All categories), and a search field. A table lists three policy definitions:

NAME	DEFINITION LOCATION	POLICIES	TYPE	DEFINITIO...	CATEGORY
Audit resource location matches reso...			Built-in	Policy	General
Allowed locations			Built-in	Policy	General
Allowed locations for resource groups			Built-in	Policy	General

3. You can see the definition is a JSON file that needs a list of allowed locations and will cause a deny. You could duplicate this definition and add more checks if needed but we'll just assign it. Click Assign.

The screenshot shows the 'Allowed locations' policy definition in JSON format. The code is as follows:

```
19 },
20 "policyRule": {
21   "if": {
22     "allOf": [
23       {
24         "field": "location",
25         "notIn": "[parameters('listOfAllowedLocations')]"
26       },
27       {
28         "field": "location",
29         "notEquals": "global"
30       },
31       {
32         "field": "type",
33         "notEquals": "Microsoft.AzureActiveDirectory/b2cDirectories"
34       }
35     ]
36   },
37   "then": {
38     "effect": "deny"
39   }
}
```

4. When assigning the policy, a scope needs to be selected... There are 3 options, as listed below, choose Resource Group.
- Management Group
 - Subscription
 - Resource Group
5. When assigning the policy, in the basics section change the assignment name and add a description.
6. To test this policy, in the resource group az-rg-labs-azurepolicy-001 try to create a resource with location other than the allowed location.

The screenshot shows the Azure portal interface for creating a new virtual machine. On the left, there's a navigation bar with 'Dashboard > New > Create a virtual machine'. The main area is titled 'Create a virtual machine' and shows a 'Validation failed' message in a red box: 'Validation failed. Click here to view details.' Below this, there are tabs for 'Basics', 'Disks', 'Networking', 'Management', 'Guest config', 'Tags', and 'Review + create'. Under 'PRODUCT DETAILS', it says 'Standard DS1 v2 by Microsoft' with a price of '0.0797 GBP/hr'. There are links for 'Subscription credits apply', 'Terms of use', 'Privacy policy', and 'Pricing for other VM sizes'. The 'Review + create' tab is selected. On the right, there's an 'Errors' panel with a 'Summary' tab selected. It displays an error message: 'Resource <resourceId> was disallowed by policy. (Code: RequestDisallowedByPolicy) Policy: Allowed locations'. There are also 'Raw Error' and 'Error Details' tabs, along with troubleshooting options like 'Check Usage + Quota' and 'New Support Request'.

There's more

Policies in Azure are a great way to scale enterprise policies for provisioning infrastructure across your estate. You can now learn more about policies [here](#)¹⁴. Policies are part of Azure governance which now also supports [blue prints](#)¹⁵ and [Resource Graphs](#)¹⁶.

¹⁴ <https://docs.microsoft.com/en-us/azure/governance/policy/overview>

¹⁵ <https://azure.microsoft.com/en-gb/services/blueprints/>

¹⁶ <https://azure.microsoft.com/en-gb/features/resource-graph/>

Lab

Integrating Azure Key Vault with Azure DevOps

In this lab, **Integrating Azure KeyVault with Azure DevOps**¹⁷, we'll cover:

- Create a key vault, from the Azure portal, to store a MySQL server password
- Configure permissions to let a service principal to read the value
- Retrieve the password in an Azure pipeline and passed on to subsequent tasks

¹⁷ <https://www.azuredevopslabs.com/labs/vstsextend/azurekeyvault/>

Module Review and Takeaways

Module Review Questions

Suggested answer

What is OWASP ZAP and how can it be used?

Suggested answer

What are the five stages of threat modeling?

Suggested answer

Why would you use WhiteSource Bolt?

Suggested answer

What is the Azure Key Vault and why would use it?

Answers

What is OWASP ZAP and how can it be used?

OWASP ZAP can be used for penetration testing. Testing can be active or passive. Conduct a quick baseline scan to identify vulnerabilities. Conduct nightly more intensive scans.

What are the five stages of threat modeling?

Define security requirements. Create an application diagram. Identify threats. Mitigate threats. Validate that threats have been mitigated.

Why would you use WhiteSource Bolt?

Use WhiteSource Bolt to automatically detect alerts on vulnerable open source components, outdated libraries, and license compliance issues in your code.

What is the Azure Key Vault and why would use it?

Azure Key Vault is a cloud key management service which allows you to create, import, store & maintain keys and secrets used by your cloud applications. The applications have no direct access to the keys, which helps improving the security & control over the stored keys & secrets. Use the Key Vault to centralize application and configuration secrets, securely store secrets and keys, and monitor access and use.

Module 7 Managing Code Quality and Security Policies

Module Overview

Module Overview

Technical Debt refers to the trade-off between decisions that make something easy in the short term and the ones that make it maintainable in the long term. Companies constantly need to trade off between solving the immediate, pressing problems and fixing long-term issues. Both code quality and security are mostly overlooked by software development teams as not their problem to solve! Part of the solution to this problem is to create a quality-focused culture that encourages shared responsibility and ownership for both code quality and security compliance. Azure DevOps has great tooling and ecosystem to improve code quality and apply automated security checks. After completing this module, students will be able to:

Learning Objectives

After completing this module, students will be able to:

- Manage code quality including: technical debt SonarCloud, and other tooling solutions
- Manage security policies with open source and OWASP

Managing Code Quality

Code Quality Defined

The quality of code shouldn't be measured subjectively. A developer writing code would rate the quality of their code high, but that's not a great way to measure code quality. Different teams may use different definitions, based on context. Code that is considered high quality may mean one thing for an automotive developer. And it may mean another for a web application developer. The quality of the code is important, as it impacts the overall software quality.

A study on "Software Defect Origins and Removal Methods" found that individual programmers are less than 50% efficient at finding bugs in their own software. And most forms of testing are only 35% efficient. This makes it difficult to determine quality.

There are five key traits to measure for higher quality.

Reliability

Reliability measures the probability that a system will run without failure over a specific period of operation. It relates to the number of defects and availability of the software.

Number of defects can be measured by running a static analysis tool. Software availability can be measured using the mean time between failures (MTBF). Low defect counts are especially important for developing a reliable codebase.

Maintainability

Maintainability measures how easily software can be maintained. It relates to the size, consistency, structure, and complexity of the codebase. And ensuring maintainable source code relies on a number of factors, such as testability and understandability.

You can't use a single metric to ensure maintainability. Some metrics you may consider to improve maintainability are the number of stylistic warnings and Halstead complexity measures. Both automation and human reviewers are essential for developing maintainable codebases.

Testability

Testability measures how well the software supports testing efforts. It relies on how well you can control, observe, isolate, and automate testing, among other factors.

Testability can be measured based on how many test cases you need to find potential faults in the system. Size and complexity of the software can impact testability. So, applying methods at the code level — such as cyclomatic complexity — can help you improve the testability of the component.

Portability

Portability measures how usable the same software is in different environments. It relates to platform independency.

There isn't a specific measure of portability. But there are several ways you can ensure portable code. It's important to regularly test code on different platforms, rather than waiting until the end of development. It's also a good idea to set your compiler warning levels as high as possible — and use at least two compilers. Enforcing a coding standard also helps with portability.

Reusability

Reusability measures whether existing assets — such as code — can be used again. Assets are more easily reused if they have characteristics such as modularity or loose coupling.

Reusability can be measured by the number of interdependencies. Running a static analyzer can help you identify these interdependencies.

Quality Metrics

While there are various quality metrics, a few of the most important ones are listed below.

Defect Metrics

The number of defects — and severity of those defects — are important metrics of overall quality.

Complexity Metrics

Complexity metrics can help in measuring quality. Cyclomatic complexity measures of the number of linearly independent paths through a program's source code. Another way to understand quality is through calculating Halstead complexity measures. These measure:

- Program vocabulary
- Program length
- Calculated program length
- Volume
- Difficulty
- Effort

Code analysis tools can be used to check for considerations such as security, performance, interoperability, language usage, globalization, and should be part of every developer's toolbox and software build process. Regularly running a static code analysis tool and reading its output is a great way to improve as a developer because the things caught by the software rules can often teach you something.

Measuring and Managing Quality Metrics

One of the promises of DevOps is to deliver software both faster and with higher quality. Previously, these two metrics have been almost opposites. The faster you went, the lower the quality. The higher the quality, the longer it took. But DevOps processes can help you to find problems earlier, and this usually means that they take less time to fix.

Common quality-related metrics

We've previously talked about some general project metrics and KPIs. The following is a list of metrics that directly relate to the quality of both the code being produced, and of the build and deployment processes.

- **Failed Builds Percentage** - Overall, what percentage of builds are failing?
- **Failed Deployments Percentage** - Overall, what percentage of deployments are failing?
- **Ticket Volume** - What is the overall volume of customer and/or bug tickets?

- **Bug Bounce Percentage** - What percentage of customer or bug tickets are being re-opened?
- **Unplanned Work Percentage** - What percentage of the overall work being performed is unplanned?

Technical Debt Defined

Technical debt is a term that describes the future cost that will be incurred by choosing an easy solution today instead of using better practices because they would take longer to complete.

The term technical debt was chosen for its comparison to financial debt. It's common for people in financial debt to make decisions that seem appropriate or the only option at the time, but in so doing, interest accrues. The more interest that accrues, the harder it is for them in the future and the less options that are available to them later. With financial debt, soon interest accrues on interest, creating a snowball effect. Similarly, technical debt can build up to the point where developers are spending almost all their time sorting out problems and doing rework, either planned or unplanned, rather than adding value.

So, how does this happen? The most common excuse is tight deadlines. When developers are forced to create code quickly, they'll often take shortcuts. As an example, instead of refactoring a method to include new functionality, let's just copy to create a new version of it. Then I only test my new code and can avoid the level of testing that might be required if I change the original method because it's used by other parts of the code. The problem is, now I have two copies of the same code that I need to modify in the future instead of one, and I run the risk of the logic diverging. There are many causes. For example, there might simply be a lack of technical skills and maturity among the developers or no clear product ownership or direction. The organization might not have coding standards at all. So, the developers didn't even know what they should be producing. The developers might not have clear requirements to target. Well, they might be subject to last minute requirement changes. Necessary refactoring work might be delayed. There might not be any code quality testing, manual or automated. In the end, it just makes it harder and harder to deliver value to customers in a reasonable time frame and at a reasonable cost. Technical debt is one of the main reasons that projects fail to meet their deadlines.

Sources and Impacts of Technical Debt

Over time, it accrues in much the same way that monetary debt does. Common sources of technical debt are:

- Lack of coding style and standards.
- Lack of or poor design of unit test cases.
- Ignoring or not understanding object orient design principles.
- Monolithic classes and code libraries.
- Poorly envisioned use of technology, architecture and approach. (Forgetting that all attributes of the system, affecting maintenance, user experience, scalability, and others, need to be considered).
- Over-engineering code (adding or creating code that is not needed, adding custom code when existing libraries are sufficient, or creating layers or components that are not needed).
- Insufficient comments and documentation.
- Not writing self-documenting code (including class, method and variable names that are descriptive or indicate intent).
- Taking shortcuts to meet deadlines.
- Leaving dead code in place.

- ✓ Note: Over time, technical debt must be paid back. Otherwise, the team's ability to fix issues, and to implement new features and enhancements will take longer and longer, and eventually become cost-prohibitive.

Using Automated Testing to Measure Technical Debt

We have seen that technical debt adds a set of problems during development and makes it much more difficult to add additional customer value.

Having technical debt in a project saps productivity, frustrates development teams, makes code both hard to understand and fragile, increases the time to make changes, and to validate those changes. Unplanned work frequently gets in the way of planned work.

Longer term, it also saps the organization's strength. Technical debt tends to creep up on an organization. It starts small, and grows over time. Every time a quick hack is made, or testing is circumvented because changes needed to be rushed through, the problem grows worse and worse. Support costs get higher and higher, and invariably, a serious issue arises.

Eventually, the organization cannot respond to its customers' needs in a timely and cost efficient way.

Automated Measurement for Monitoring

One key way to minimize the constant acquisition of technical debt, is to use automated testing and assessment.

In the demos that follow, we'll take a look at one of the common tools that is used to assess the debt: SonarCloud. (The original on-premises version was SonarQube).

There are other tools available and we will discuss a few of them. Later, in the next hands-on lab, you will see how to configure your Azure Pipelines to use SonarCloud, how to understand the analysis results, and finally how to configure quality profiles to control the rule sets that are used by SonarCloud when analyzing your projects.

For more information, see [SonarCloud¹](#).

Discussion - Code Quality Tooling

#Discussion - Code Quality Tooling

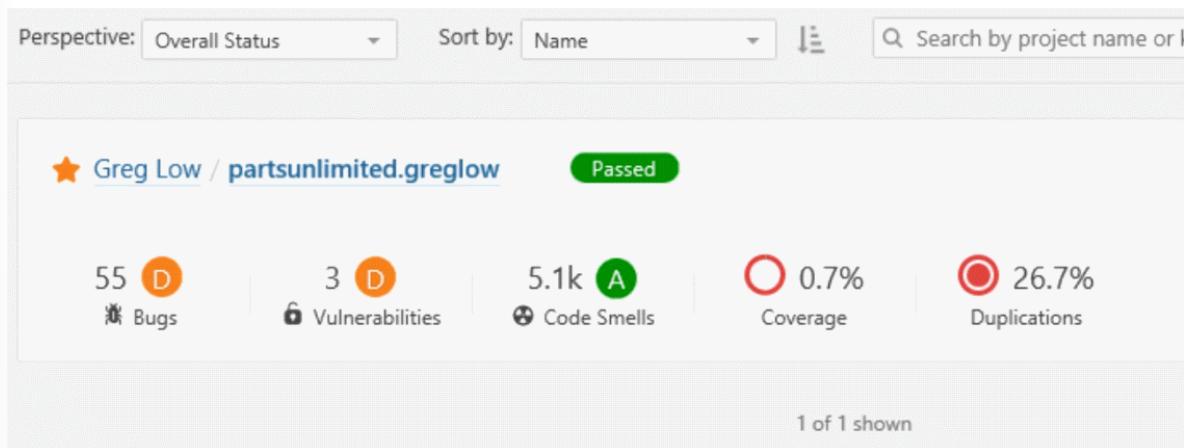
Azure DevOps can be integrated with a wide range of existing tooling that is used for checking code quality during builds. Which code quality tools do you currently use (if any)? What do you like or don't like about the tools?

Measuring and Managing Technical Debt

It is important to integrate the assessment and measurement of technical debt and of code quality overall, as part of your Continuous Integration and Deployment pipelines in Azure DevOps.

In the Continuous Integration course in this series, we showed how to add support for SonarCloud into an Azure DevOps pipeline. After it is added and a build performed, you can see an analysis of your code:

¹ <https://sonarcloud.io/about>



If you drill into the issues, you can then see what the issues are, along with suggested remedies, and estimates of the time required to apply a remedy.

PartsUnlimitedWebsite / App_Start\BundleConfig.cs

- Add a 'protected' constructor or the 'static' keyword to the class declaration. ...
Code Smell ▾ Major ▾ Open ▾ Not assigned ▾ 10min effort [Comment](#)
- Refactor your code not to use hardcoded absolute paths or URIs. ...
Code Smell ▾ Minor ▾ Open ▾ Not assigned ▾ 20min effort [Comment](#)
- Refactor your code not to use hardcoded absolute paths or URIs. ...
Code Smell ▾ Minor ▾ Open ▾ Not assigned ▾ 20min effort [Comment](#)

Integrating Other Code Quality Tools

There are many tools that can be used to assess aspects of your code quality and technical debt.

NDepend

For .NET developers, a common tool is NDepend.

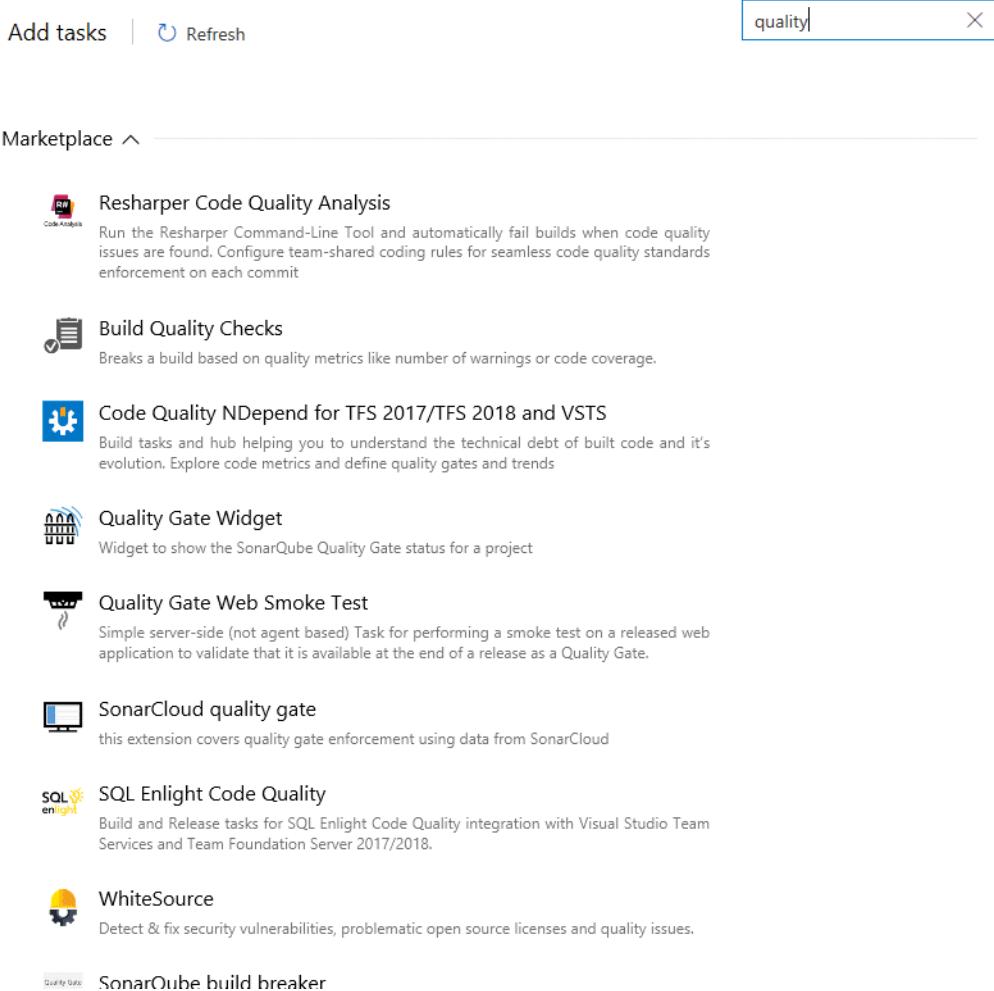
NDepend is a Visual Studio extension that assesses the amount of technical debt that a developer has added during a recent development period, typically in the last hour. With this information, the developer might be able to make the required corrections before ever committing the code. NDepend lets you create code rules that are expressed as C# LINQ queries but it has a large number of built-in rules that detect a wide range of code smells.

Resharper Code Quality Analysis

Resharper can make a code quality analysis from the command-line, and can be set to automatically fail builds when code quality issues are found. Rules can be configured for enforcement across teams.

Search in Azure DevOps

To find other code quality tools that are easy to integrate with Azure DevOps, search for the word **Quality** when adding a task into your build pipeline.



The screenshot shows the Azure DevOps Marketplace search results for the term "quality". The search bar at the top contains "quality". Below it, a list of extensions is displayed:

- Resharper Code Quality Analysis** (CodeAnalysis icon): Run the Resharper Command-Line Tool and automatically fail builds when code quality issues are found. Configure team-shared coding rules for seamless code quality standards enforcement on each commit.
- Build Quality Checks** (document icon): Breaks a build based on quality metrics like number of warnings or code coverage.
- Code Quality NDepend for TFS 2017/TFS 2018 and VSTS** (gear icon): Build tasks and hub helping you to understand the technical debt of built code and its evolution. Explore code metrics and define quality gates and trends.
- Quality Gate Widget** (bar chart icon): Widget to show the SonarQube Quality Gate status for a project.
- Quality Gate Web Smoke Test** (monitor icon): Simple server-side (not agent based) Task for performing a smoke test on a released web application to validate that it is available at the end of a release as a Quality Gate.
- SonarCloud quality gate** (monitor icon): this extension covers quality gate enforcement using data from SonarCloud
- SQL Enlight Code Quality** (SQL icon): Build and Release tasks for SQL Enlight Code Quality integration with Visual Studio Team Services and Team Foundation Server 2017/2018.
- WhiteSource** (globe icon): Detect & fix security vulnerabilities, problematic open source licenses and quality issues.
- SonarQube build breaker** (Quality Gate icon)

For more information, you can see:

- [NDepend²](https://www.ndepend.com)
- [Visual Studio marketplace³](https://marketplace.visualstudio.com/items?itemName=ndepend.ndependextension&targetId=2ec491f3-0a97-4e53-bfef-20bf80c7e1ea)
- [Resharper Code Quality Analysis⁴](https://marketplace.visualstudio.com/items?itemName=alanwales.resharper-code-analysis)

Planning Effective Code Reviews

Most developers would agree that code reviews can improve the quality of the applications they produce, but only if the process for the code reviews is effective.

² <https://www.ndepend.com>

³ <https://marketplace.visualstudio.com/items?itemName=ndepend.ndependextension&targetId=2ec491f3-0a97-4e53-bfef-20bf80c7e1ea>

⁴ <https://marketplace.visualstudio.com/items?itemName=alanwales.resharper-code-analysis>

It's important, up front, to agree that everyone is trying to achieve better code quality. Achieving code quality can seem challenging because there is no one single best way to write any piece of code, at least code with any complexity.

Everyone wants to do good work and to be proud of what they create. This means that it's easy for developers to become over-protective of their code. The organizational culture must let all involved feel that the code reviews are more like mentoring sessions where ideas about how to improve code are shared, than interrogation sessions where the aim is to identify problems and blame the author.

The knowledge sharing that can occur in mentoring-style sessions can be one of the most important outcomes of the code review process. It often happens best in small groups (perhaps even just two people), rather than in large team meetings. And it's important to highlight what has been done well, not just what needs to be improved.

Developers will often learn more in effective code review sessions than they will in any type of formal training. Reviewing code should be seen as an opportunity for all involved to learn, not just as a chore that must be completed as part of a formal process.

It's easy to see two or more people working on a problem and think that one person could have completed the task by themselves. That's a superficial view of the longer-term outcomes. Team management needs to understand that improving the code quality reduces the cost of code, not increases it. Team leaders need to establish and foster an appropriate culture across their teams.

Managing Security Policies

Inspecting and Validating Code Bases for Compliance

Security for applications is extremely important today. Every day, news services worldwide seem to carry stories about some company systems that have been breached. More importantly, private company and customer data has been disclosed. This has been happening for a long time. In many cases, it wasn't visible to the public. Often, private information was disclosed, and yet the people affected were not even notified. Governments across the world are frequently enacting legislation to require information about breaches to become public and to require notifications to those affected.

So, what are the issues? Clearly, we need to protect information from being disclosed to people that should not have access to it. But more importantly, we need to ensure that the information isn't altered or destroyed when it shouldn't be, and we need to make sure it is destroyed when it's supposed to be. We need to make sure we properly authenticate who is accessing the data and that they have the correct permissions to do so. Through historical or archival data or logs, we need to be able to find evidence when something has gone wrong.

There are many aspects to building and deploying secure applications.

- First, there is a general knowledge problem. Many developers and other staff assume they understand security, but they don't. Cybersecurity is a constantly evolving discipline. A program of ongoing education and training is essential.
- Second, we need to ensure that the code is created correctly and securely implements the required features, and we need to make sure that the features were designed with security in mind in the first place.
- Third, we need to ensure that the application complies with the rules and regulations that it is required to meet. We need to test for this while building the code and retest periodically even after deployment.

It's commonly accepted that security isn't something you can just add to an application or a system later. Secure development must be part of every stage of the software development life cycle. This is even more important for critical applications and those that process sensitive or highly confidential information. Application security concepts haven't been a focus for developers in the past. Apart from the education and training issues, it's because their organizations have emphasized fast development of features. With the introduction of DevOps practices however, security testing is much easier to integrate. Rather than being a task performed by security specialists, security testing should be part of the day-to-day delivery processes. Overall, when the time for rework is taken into account, adding security to your DevOps practices can reduce the overall time taken to develop quality software.

Planning to Implement OWASP Secure Coding Practices

The starting point for secure development is to use secure coding practices. The **Open Web Application Security Project (OWASP)**⁵ is a global charitable organization focused on improving the security of software. OWASP's stated mission is to make software security visible, so that individuals and organizations are able to make informed decisions. They offer impartial and practical advice.

⁵ <http://owasp.org>

OWASP regularly publish a set of Secure Coding Practices. Their guidelines currently cover advice in the following areas:

- Input Validation
- Output Encoding
- Authentication and Password Management
- Session Management
- Access Control
- Cryptographic Practices
- Error Handling and Logging
- Data Protection
- Communication Security
- System Configuration
- Database Security
- File Management
- Memory Management
- General Coding Practices

To learn about common vulnerabilities, and to see how they appear in applications, OWASP also publishes an intentionally vulnerable web application called **The Juice Shop Tool Project**⁶. It includes vulnerabilities from all of the **OWASP Top 10**⁷.

In 2002, Microsoft underwent a company-wide re-education and review phase to focus on producing secure application code. The book, **Writing Secure Code by David LeBlanc, Michael Howard**⁸, was written by two of the people involved and provides detailed advice on how to write secure code.

For more information, you can see:

- **The OWASP foundation**⁹
- **OWASP Secure Coding Practices Quick Reference Guide**¹⁰
- **OWASP Code Review guide**¹¹
- **OWASP Top Ten**¹²

Inspecting and Validating Code Bases for Compliance

Automated Code Security Analysis

There are many tools for automating code security analysis. They include the following:

⁶ https://www.owasp.org/index.php/OWASP_Juice_Shop_Project

⁷ https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

⁸ <https://www.booktopia.com.au/ebooks/writing-secure-code-david-leblanc/prod2370006179962.html>

⁹ <http://owasp.org>

¹⁰ https://www.owasp.org/images/0/08/OWASP SCP_Quick_Reference_Guide_v2.pdf

¹¹ https://www.owasp.org/images/2/2e/OWASP_Code_Review_Guide-V1_1.pdf

¹² https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Micro Focus Fortify¹³ provides build tasks that can be used in Azure DevOps continuous integration builds to identify vulnerabilities in source code. It offers two styles of analysis.

- **Fortify Static Code Analyzer** (SCA) searches for violations of security-specific coding rules and guidelines. It works in a variety of languages.
- **Fortify on Demand** is a service for checking application security. The outcomes of an SCA scan are audited by a team of security experts, including the use of Fortify WebInspect for automated dynamic scanning.

Checkmarx CxSAST¹⁴ is a solution for Static Source Code Analysis (SAST) and Open Source Analysis (OSA) designed for identifying, tracking and fixing technical and logical security flaws.

It is designed to integrated into Azure DevOps pipelines and allows for early detection and mitigation of crucial security flaws. To improve performance, it is capable of incremental scanning (ie: just checking the code recently altered or introduced).

BinSkim¹⁵ is a static analysis tool that scans binary files. BinSkim replaces an earlier Microsoft tool called BinScope. In particular, it checks that the executable produced (ie: a Windows PE formatted file) has opted into all of the binary mitigations offered by the Windows Platform, including:

- SafeSEH is enabled for safe exception handling
- ASLR is enabled so that memory is not laid out in a predictable fashion
- Stack Protection is enabled to prevent overflow

OWASP Zed Attack Proxy Scan. Also known as **OWASP ZAP** Scan is an open-source web application security scanner that is intended for users with all levels of security knowledge. It can be used by professional penetration testers.

Open Source Licensing Challenges

For more information, see **Common Vulnerabilities and Exposures**¹⁶.

Discussion - Security Policy Tooling

Azure DevOps can be integrated with a wide range of existing tooling that is used for checking security policy during builds.

Which security policy tools do you currently use (if any)? What do you like or don't like about them?

¹³ <https://marketplace.visualstudio.com/items?itemName=fortifyvsts.hpe-security-fortify-vsts>

¹⁴ <https://marketplace.visualstudio.com/items?itemName=checkmarx.cxsast>

¹⁵ <https://blogs.msdn.microsoft.com/secdevblog/2016/08/17/introducing-binskim/>

¹⁶ <https://cve.mitre.org/about/>

Lab

Managing Technical Debt with Azure DevOps and SonarCloud

In this hands-on lab, **Managing Technical Debt with Azure DevOps and SonarCloud**¹⁷, you will learn how to manage and report on technical debt using SonarCloud integration with Azure DevOps.

You will perform the following tasks:

- Integrate SonarCloud with Azure DevOps and run an analysis
- Analyze the results
- Configure a quality profile to control the rule set used for analyzing your project

 Note: You must have already completed the Lab Environment Setup in the Welcome section.

¹⁷ <https://www.azuredevopslabs.com/labs/azuredevops/sonarcloud/>

Module Review and Takeaways

Module Review Questions

Suggested answer

You want to run a penetration test against your application. Which tool could you use?

Suggested answer

What is code smells? Give an example of a code smell.

Suggested answer

You are using Azure Repos for your application source code repository. You want to create an audit of open source libraries that you have used. Which tool could you use?

Suggested answer

Name three attributes of high quality code.

Suggested answer

You are using Azure Repos for your application source code repository. You want to perform code quality checks. Which tool could you use?

Answers

You want to run a penetration test against your application. Which tool could you use?

OWASP ZAP. OWASP ZAP is designed to run penetration testing against applications. Bolt is used to analyze open source library usage. The two Sonar products are for code quality and code coverage analysis.

What is code smells? Give an example of a code smell.

Code smells are characteristics in your code that could possibly be a problem. Code smells hint at deeper problems in the design or implementation of the code. For example, code that works but contains many literal values or duplicated code.

You are using Azure Repos for your application source code repository. You want to create an audit of open source libraries that you have used. Which tool could you use?

WhiteSource Bolt is used to analyze open source library usage. OWASP ZAP is designed to run penetration testing against applications. The two Sonar products are for code quality and code coverage analysis.

Name three attributes of high quality code.

High quality code should have well-defined interfaces. It should be clear and easy to read so self-documenting is desirable, as is short (not long) method bodies.

You are using Azure Repos for your application source code repository. You want to perform code quality checks. Which tool could you use?

SonarCloud is the cloud-based version of the original SonarQube, and would be best for working with code in Azure Repos.

Module 8 Implementing a Container Build Strategy

Module Overview

Module Overview

Containers are the third model of compute, after bare metal and virtual machines – and containers are here to stay. Docker gives you a simple platform for running apps in containers, old and new apps on Windows and Linux, and that simplicity is a powerful enabler for all aspects of modern IT. Containers aren't only faster and easier to use than VMs; they also make far more efficient use of computing hardware.

Learning Objectives

After completing this module, students will be able to:

- Implement a container strategy including how containers are different from virtual machines and how microservices use containers
- Implement containers using Docker

Managing Code Quality

Overview of Containers

If you're a programmer or techie, chances are you've at least heard of Docker: a helpful tool for packing, shipping, and running applications within "containers." It'd be hard not to, with all the attention it's getting these days — from developers and system admins alike. Just to reiterate, there is a difference between containers and docker. A container is a thing that runs a little program package, while Docker is the container runtime and orchestrator.

What are containers and why do you need them?

Containers are a solution to the problem of how to get software to run reliably when moved from one computing environment to another. This could be from a developer's laptop to a test environment, from a staging environment into production, and perhaps from a physical machine in a data center to a virtual machine in a private or public cloud.

Problems arise when the supporting software environment is not identical. Let's take an example, say you are going to develop using Python 3, but when it gets deployed to production it's going to run on Python 2.7, this is likely to cause several issues. It's just not limited to software environment, you are likely to encounter issues in production if there are differences in the networking stack between the two environments.

How do containers solve this problem?

Put simply, a container consists of an entire runtime environment: an application, plus all its dependencies, libraries and other binaries, and configuration files needed to run it, bundled into one package. By containerizing the application platform and its dependencies, differences in OS distributions and underlying infrastructure are abstracted away.

What's the difference between containers and virtualization?

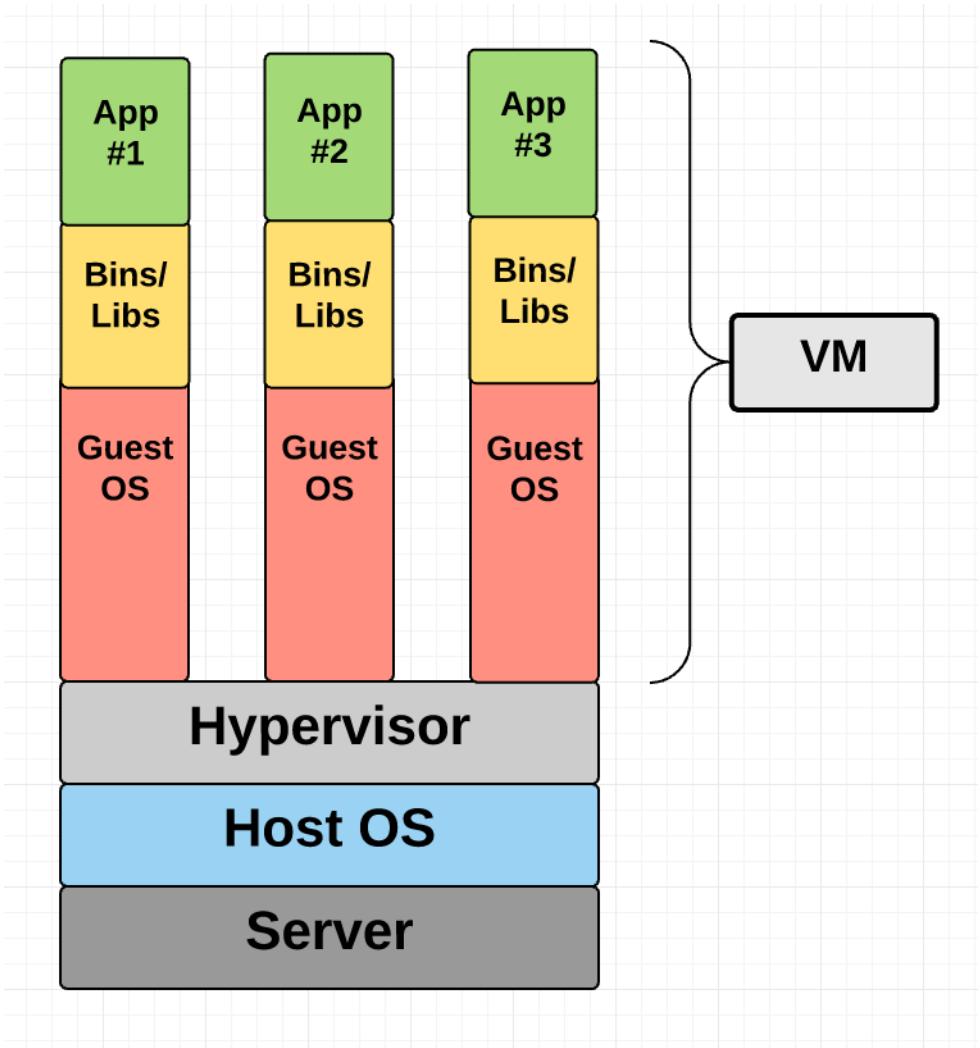
Containers and VMs are similar in their goals: to isolate an application and its dependencies into a self-contained unit that can run anywhere.

Moreover, containers and VMs remove the need for physical hardware, allowing for more efficient use of computing resources, both in terms of energy consumption and cost effectiveness.

The main difference between containers and VMs is in their architectural approach. Let's take a closer look.

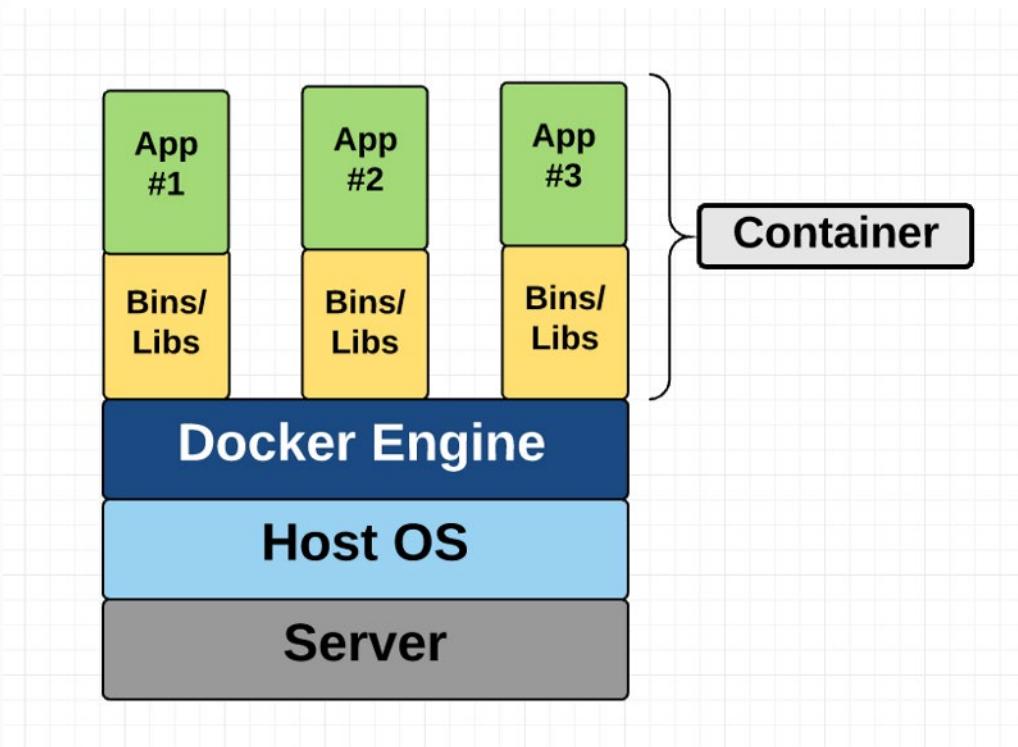
Virtual Machines

A VM is essentially an emulation of a real computer that executes programs like a real computer. VMs run on top of a physical machine using a "hypervisor". As you can see in the diagram, VMs package up the virtual hardware, a kernel (i.e. OS) and user space for each new VM.



Container

Unlike a VM which provides hardware virtualization, a container provides operating-system-level virtualization by abstracting the “user space”. This diagram shows you that containers package up just the user space, and not the kernel or virtual hardware like a VM does. Each container gets its own isolated user space to allow multiple containers to run on a single host machine. We can see that all the operating system level architecture is being shared across containers. The only parts that are created from scratch are the bins and libs. This is what makes containers so lightweight.



What other benefits do containers offer?

A container may be only tens of megabytes in size, whereas a virtual machine with its own entire operating system may be several gigabytes in size. Because of this, a single server can host far more containers than virtual machines.

Another major benefit is that virtual machines may take several minutes to boot up their operating systems and begin running the applications they host, while containerized applications can be started almost instantly. That means containers can be instantiated in a “just in time” fashion when they are needed and can disappear when they are no longer required, freeing up resources on their hosts.

A third benefit is that containerization allows for greater modularity. Rather than run an entire complex application inside a single container, the application can be split into modules (such as the database, the application front end, and so on). This is the so-called microservices approach. Applications built in this way are easier to manage because each module is relatively simple, and changes can be made to modules without having to rebuild the entire application. Because containers are so lightweight, individual modules (or microservices) can be instantiated only when they are needed and are available almost immediately.

Discussion - Containers vs Virtual Machines

In your development environment, do you currently use virtualization of any type? Do you prefer to deploy containers or entire virtual machines?

Docker Containers and Development



Docker is a software containerization platform with a common toolset, packaging model, and deployment mechanism, which greatly simplifies containerization and distribution of applications that can be run anywhere. This ubiquitous technology not only simplifies management by offering the same management commands against any host, it also creates a unique opportunity for seamless DevOps.

From a developer's desktop to a testing machine, to a set of production machines, a Docker image can be created that will deploy identically across any environment in seconds. This is a massive and growing ecosystem of applications packaged in Docker containers, with DockerHub, the public containerized-application registry that Docker maintains, currently publishing more than 180,000 applications in the public community repository. Additionally, to guarantee the packaging format remains universal, Docker organized the Open Container Initiative (OCI), aiming to ensure container packaging remains an open and foundation-led format.

As an example of the power of containers, a SQL Server Linux instance can be deployed using a Docker image in seconds.

For more information, see:

- **Docker Ebook, Docker for the Virtualization Admin¹**
- **Mark Russinovich blog post on Containers: Docker, Windows, and Trends²**

¹ <https://goto.docker.com/docker-for-the-virtualization-admin.html>

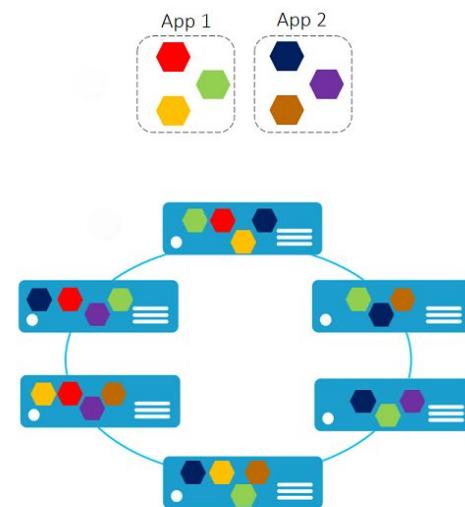
² <https://azure.microsoft.com/en-us/blog/containers-docker-windows-and-trends/>

Microservices and containers

Monolithic application approach



Microservices application approach



The most immediately lucrative use for containers has been focused on simplifying DevOps with easy developer-to-test-to-production flows for services deployed in the cloud or on-premises. But there is another fast-growing scenario where containers are becoming very compelling.

Microservices is an approach to application development where every part of the application is deployed as a fully self-contained component, called a microservice, that can be individually scaled and updated.

Example Scenario

Imagine that you are part of a software house that produces a large monolithic financial management application that you are migrating to a series of microservices. The existing application would include the code to update the general ledger for each transaction, and it would have this code in many places throughout the application. If the schema of the general ledger transactions table is modified, this would require changes throughout the application.

By comparison, the application could be modified to make a notification that a transaction has occurred. Any microservice that is interested in the transactions could subscribe. In particular, a separate general ledger microservice could subscribe to the transaction notifications, and then perform the general ledger related functionality. If the schema of the table that holds the general ledger transactions is modified, only the general ledger microservice should need to be updated.

If a particular client organization wants to run the application and not use the general ledger, that service could just be disabled. No other changes to the code would be required.

Scale

In a dev/test environment on a single system, while you might have a single instance of each microservice, in production you might scale out to different numbers of instances across a cluster of servers depending on their resource demands as customer request levels rise and fall. If different teams produce them, the teams can also independently update them. Microservices is not a new approach to program-

ming, nor is it tied explicitly to containers, but the benefits of Docker containers are magnified when applied to a complex microservice-based application. Agility means that a microservice can quickly scale out to meet increased load, the namespace and resource isolation of containers prevents one microservice instance from interfering with others, and use of the Docker packaging format and APIs unlocks the Docker ecosystem for the microservice developer and application operator. With a good microservice architecture, customers can solve the management, deployment, orchestration and patching needs of a container-based service with reduced risk of availability loss while maintaining high agility.

Azure Container-Related Services

Azure provides a wide range of services that help you to work with containers. These are the key services that are involved:

Azure Container Instances (ACI)³

Running your workloads in Azure Container Instances (ACI), allows you to focus on creating your applications rather than focusing on provisioning and management of the infrastructure that will run the applications.

ACIs are simple and fast to deploy, and when you are using them, you gain the security of hypervisor isolation for each container group. This ensures that your containers aren't sharing an operating system kernel with other containers.

Azure Kubernetes Service (AKS)⁴

Kubernetes has quickly become the de facto standard for container orchestration. This services lets you easily deploy and manage Kubernetes, to scale and run applications, while maintaining strong overall security.

This service started life as Azure Container Services (ACS) and at release supported Docker Swarm and Mesos/Mesosphere DC/OS for managing orchestrations. These original ACS workloads are still supported in Azure but Kubernetes support was added. This quickly became so popular that Microsoft changed the acronym for Azure Container Services to AKS, and later changed the name of the service to Azure Kubernetes Service (also AKS).

Azure Container Registry (ACR)⁵

This service lets you store and manage container images in a central registry. It provide you with a Docker private registry as a first-class Azure resource.

All types of container deployments, including DC/OS, Docker Swarm, Kubernetes are supported and the registry is integrated with other Azure services such as the App Service, Batch, Service Fabric and others.

Importantly, this allows your DevOps team to manage the configuration of apps without being tied to the configuration of the target hosting environment.

Azure Service Fabric⁶

Azure Service Fabric allows you to build and operate always-on, scalable, distributed apps. It simplifies the development of microservice-based applications and their life cycle management including rolling updates with rollback, partitioning, and placement constraints. It can host and orchestrate containers, including stateful containers.

Azure App Service⁷

³ <https://azure.microsoft.com/en-us/services/container-instances/>

⁴ <https://azure.microsoft.com/en-us/services/kubernetes-service/>

⁵ <https://azure.microsoft.com/en-us/services/container-registry/>

⁶ <https://azure.microsoft.com/en-us/services/service-fabric/>

⁷ <https://azure.microsoft.com/en-us/services/app-service/>

Azure Web Apps provides a managed service for both Windows and Linux based web applications, and provides the ability to deploy and run containerized applications for both platforms. It provides options for auto-scaling and load balancing and is easy to integrate with Azure DevOps.

Dockerfile Core Concepts

Dockerfiles are text files that contain the commands needed by **docker build** to assemble an image.

Here is an example of a very basic Dockerfile:

```
FROM ubuntu
LABEL maintainer="greglow@contoso.com"
ADD appsetup /
RUN /bin/bash -c 'source $HOME/.bashrc; \
echo $HOME'
CMD ["echo", "Hello World from within the container"]
```

Note the following in this example.

The first line refers to the parent image that this new image will be based upon. Generally, all images will be based off another existing image. In this case, the Ubuntu image would be retrieved from either a local cache or from DockerHub. An image that doesn't have a parent is called a **base** image. In that rare case, the FROM line can be omitted, or **FROM scratch** can be used instead.

The second line indicates the email address of the person who maintains this file. Previously, there was a MAINTAINER command but that has been deprecated and replaced by a label.

The third line adds a file into the root folder of the image. It can also add an executable.

The fourth and fifth lines are part of a RUN command. Note the use of the backslash to continue the fourth line onto the fifth line, for readability. This is equivalent to having written this instead:

```
RUN /bin/bash -c 'source $HOME/.bashrc; echo $HOME'
```

The RUN command is run when the image is being created by **docker build**. It is generally used to configure items within the image.

By comparison, the last line represents a command that will be executed when a new container is created from the image ie: it is run after container creation.

For more information, you can see:

[Dockerfile reference⁸](#)

Multiple Stage Builds

It's important when building images, to keep the image size as small as possible. Every additional instruction that is added to the Dockerfile adds what is referred to as a **layer**. Each layer often contains artifacts that are not needed in the final image and should be cleaned up before moving to the next layer. Doing this has been tricky.

Docker 17.05 added a new feature that allowed the creation of multi-stage builds. This helps with being able to optimize the files, improves their readability, and makes them easier to maintain.

Here is an example of a multi-stage file that was created by Visual Studio 2017:

⁸ <https://docs.docker.com/engine/reference/builder/>

```
FROM microsoft/dotnet:2.1-aspnetcore-runtime AS base
WORKDIR /app
EXPOSE 6636
EXPOSE 44320

FROM microsoft/dotnet:2.1-sdk AS build
WORKDIR /src
COPY ["WebApplication1/WebApplication1.csproj", "WebApplication1/"]
RUN dotnet restore "WebApplication1/WebApplication1.csproj"
COPY ..
WORKDIR "/src/WebApplication1"
RUN dotnet build "WebApplication1.csproj" -c Release -o /app

FROM build AS publish
RUN dotnet publish "WebApplication1.csproj" -c Release -o /app

FROM base AS final
WORKDIR /app
COPY --from=publish /app .
ENTRYPOINT ["dotnet", "WebApplication1.dll"]
```

With multi-stage builds, you use multiple FROM statements in your Dockerfile. Each FROM instruction starts a new stage. The stages are numbered in order, starting with stage 0. To make the file easier to maintain without needing to constantly change numbers that reference, note how each stage has been named (or aliased) by using an **AS** clause.

Each FROM instruction can have a different parent (ie: base). This allows the developer to control what is copied from one stage to another, and avoids the need for intermediate images.

Another advantage of named stages is that they are easier to refer to in external commands. For example, not all stages need to be built each time. You can see that in the following Docker CLI command:

```
$ docker build --target publish -t gregsimages/popkorn:latest .
```

The **--target** option tells **docker build** that it needs to create an image up to the target of publish, which was one of the named stages.

Considerations for Multiple Stage Builds

Adopt Container Modularity

Try to avoid creating overly complex container images that couple together a number of applications. Instead, use multiple containers and try to keep each container to a single purpose. The website and the database for a web application should likely be in separate containers.

There are always exceptions to any rule but breaking up application components into separate containers increases the chances of being able to reuse containers. It also makes it more likely that you could scale the application. For example, in the web application mentioned, you might want to add replicas of the website container but not for the database container.

Avoid Unnecessary Packages

To help with minimizing image sizes, it is also important avoid including packages that you suspect might be needed but aren't yet sure if they are needed. Only include them when they are needed.

Choose an Appropriate Base

While optimizing the contents of your Dockerfiles is important, it is also very important to choose the appropriate parent (base) image. Start with an image that only contains packages that are required.

Avoid Including Application Data

While application data can be stored in the container, doing this will make your images larger. You should consider using **docker volume** support to maintain the isolation of your application and its data. Volumes are persistent storage mechanisms that exist outside the lifespan of a container.

For more information, see [Use multiple stage builds⁹](#).

Demonstration - Create an Azure Container Registry

What is Azure Container Registry?

Azure Container Registry is a managed, private Docker registry service based on the open-source Docker Registry 2.0. You can use Azure container registries to store and manage your private Docker container images and related artifacts. Azure container registries can be used with your existing container development and deployment pipelines. Azure Container Registry task in Azure DevOps can be used to build container images in Azure. These tasks can be used to build images on demand and fully automate build workflows with triggers such as source code commits and base image updates.

Create an Azure Container Registry

The Azure Portal offers a great experience for creating a new container registry. In this section, we'll go through the steps of creating a new registry via the Azure CLI.

Create a resource group

Create a resource group with the `az group create` command. An Azure resource group is a logical container into which Azure resources are deployed and managed. The following example creates a resource group named `myResourceGroup` in the `eastus` location.

```
az group create --name myResourceGroup --location eastus
```

Create a container registry

The following example will create a basic registry called `myaz400containerregistry` in the resource group we created in the previous step.

⁹ <https://docs.docker.com/develop/develop-images/multistage-build/>

```
az acr create --resource-group myResourceGroup --name myaz400containerregistry --sku Basic
```

The response from this command, returns the `loginServer` which has the fully qualified url of the registry.

```
{
  "adminUserEnabled": false,
  "creationDate": "2020-03-08T22:32:13.175925+00:00",
  "id": "/subscriptions/00000000-0000-0000-0000-000000000000/resourceGroups/myResourceGroup/providers/Microsoft.ContainerRegistry/registries/myaz400containerregistry",
  "location": "eastus",
  "loginServer": "myaz400containerregistry.azurecr.io",
  "name": "myaz400containerregistry",
  "provisioningState": "Succeeded",
  "resourceGroup": "myResourceGroup",
  "sku": {
    "name": "Basic",
    "tier": "Basic"
  },
  "status": null,
  "storageAccount": null,
  "tags": {},
  "type": "Microsoft.ContainerRegistry/registries"
}
```

Log in to registry

Before pushing and pulling container images, you must log in to the registry. To do so, use the `az acr login` command.

```
az acr login --name <acrName>
```

Push image to registry

To push an image to an Azure Container registry, you must first have an image. If you don't yet have any local container images, run the following docker pull command to pull an existing image from Docker Hub. For this example, pull the `hello-world` image.

```
docker pull hello-world
```

Before you can push an image to your registry, you must tag it with the fully qualified name of your ACR login server. The login server name is in the format '`registry-name`.azurecr.io (all lowercase), for example, `myaz400containerregistry.azurecr.io`.

```
docker tag hello-world <acrLoginServer>/hello-world:v1
```

Finally, use docker push to push the image to the ACR instance. Replace acrLoginServer with the login server name of your ACR instance. This example creates the hello-world repository, containing the hello-world:v1 image.

```
docker push <acrLoginServer>/hello-world:v1
```

After pushing the image to your container registry, remove the hello-world:v1 image from your local Docker environment.

```
docker rmi <acrLoginServer>/hello-world:v1
```

List container images

The following example lists the repositories in your registry:

```
az acr repository list --name <acrName> --output table
```

Run image from registry

You can pull and run the hello-world:v1 container image from your container registry by using docker run:

```
docker run <acrLoginServer>/hello-world:v1
```

Clean up resources

When no longer needed, you can use the az group delete command to remove the resource group, the container registry, and the container images stored there.

```
az group delete --name myResourceGroup
```

Demonstration - Add Docker Support to an Existing Application

Add Docker Support to an Existing Application

Visual Studio 2017 and later versions support building, debugging, and running containerized ASP.NET Core apps targeting .NET Core. Both Windows and Linux containers are supported.

To containerize an ASP.NET Core project, the project must target .NET Core. Both Linux and Windows containers are supported.

When adding Docker support to a project, choose either a Windows or a Linux container. The Docker host must be running the same container type. To change the container type in the running Docker instance, right-click the System Tray's Docker icon and choose Switch to Windows containers... or Switch to Linux containers....

Follow the steps here on Microsoft Docs for a **walkthrough on how to add docker support to an existing application¹⁰**.

¹⁰ <https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/docker/visual-studio-tools-for-docker?view=aspnetcore-3.1>

Lab

Modernizing Your Existing ASP.NET Apps with Azure

In this hands-on lab, **Modernizing your existing ASP.NET Apps with Azure**¹¹, you will learn how to modernize an existing ASP.NET application with migration to Docker images managed by the Azure Container Registry.

You will perform the following tasks:

- Migrate the LocalDB to SQL Server in Azure
- Using the Docker tools in Visual Studio 2017, add Docker support for the application
- Publish Docker Images to Azure Container Registry (ACR)
- Push the new Docker images from ACR to Azure Container Instances (ACI)

¹¹ <https://www.azuredevopslabs.com/labs/vstsextend/aspmmodernize/>

Module Review and Takeaways

Module Review Questions

Suggested answer

You are reviewing an existing Dockerfile. How would you know if it's a multi-stage Dockerfile?

Suggested answer

You are designing a multi-stage Dockerfile. How can one stage refer to another stage within the Dockerfile?

Suggested answer

What is the line continuation character in Dockerfiles?

Suggested answer

You are using Azure to manage your containers. Which container orchestration styles are supported?

Suggested answer

When the Open Container Initiative defined a standard container image file format, which format did they choose as a starting point?

Answers

You are reviewing an existing Dockerfile. How would you know if it's a multi-stage Dockerfile?

Multi-stage Docker files are characterized by containing more than one starting point provided as FROM instructions.

You are designing a multi-stage Dockerfile. How can one stage refer to another stage within the Dockerfile?

The FROM clause in a multi-stage Dockerfile can contain an alias via an AS clause. The stages can refer to each other by number or by the alias names.

What is the line continuation character in Dockerfiles?

Lines can be broken and continued on the next line of a Dockerfile by using the backslash character.

You are using Azure to manage your containers. Which container orchestration styles are supported?

Swarm, DC/OS, and AKS are supported.

When the Open Container Initiative defined a standard container image file format, which format did they choose as a starting point?

The OCI used the Docker format as a starting point.

Module 9 Manage Artifact Versioning, Security, and Compliance

Module Overview

Module Overview

Welcome to this module about managing artifact versioning, security, and compliance. In this module, we will talk about how you can secure your packages and feeds and check security requirements on the packages used in developing your software solutions. Also we will cover how to make sure the packages used are compliant to the standard and requirements that exist in your organization from a licensing and security vulnerability perspective.

Learning objectives

At the end of this module, students will be able to:

- Inspect open source software packages for security and license compliance to align with corporate standards
- Configure build pipeline to access package security and license rating
- Configure secure access to package feeds
- Inspect codebase to identify code dependencies that can be converted to packages
- Identify and recommend standardized package types and versions across the solution
- Refactor existing build pipelines to implement version strategy that publishes packages
- Manage security and compliance

Package Security

Package Feeds

Package feeds are a trusted source of packages. The packages that are offered will be consumed by other code bases and used to build software that needs to be secure. Imagine what would happen if a package feed would offer malicious components in its packages. Each consumer would be affected when installing the packages onto its development machine or build server. This also happens at any other device that will run the end product, as the malicious components will be executed as part of the code. Usually the code runs with high privileges, giving a substantial security risk if any of the packages cannot be trusted and might contain unsafe code.

Therefore, it is essential that package feeds are secured for access by authorized accounts, so only verified and trusted packages are stored there. No one should be able to push packages to a feed without the proper role and permissions. This prevents others from pushing malicious packages. It still assumes that the persons who are allowed to push packages will only add safe and secure packages. Especially in the open source world this is performed by the community. A package source can further guard its feed with the use of security and vulnerability scan tooling. Additionally, consumers of packages can use similar tooling to perform the scans themselves.

Another aspect of security for package feeds is about public or private availability of the packages. The feeds of public sources are usually available for anonymous consumption. Private feeds on the other hand have restricted access most of the time. This applies to consumption and publishing of packages. Private feeds will allow only users in specific roles or teams access to its packages.

Package compliance

Nowadays companies have obligations towards their customers and employees to make sure that the services they offer with software and IT are compliant to rules and regulations. In part the rules and regulations come from governments, certification and standards institutes. They might also be self imposed rules and regulations from the company or organization itself. This could include rules about how open source is used, which license types are allowed and a package version policy.

When development teams are empowered to make their own choices for the use of packages it becomes very important that they are choosing the right type of packages. In this case that would imply that the packages are allowed from a licensing perspective and follow the chosen ruleset to be compliant with the applicable policies. It involves practices, guidelines, hosting, additional work and introduction of tooling that will help to make sure that compliance is part of the software development process when it comes to producing and consuming packages.

The compliance of packages should be guaranteed and provable. The software development processes should take this into account in an integral fashion. We will look at open source software and licensing in the next chapters, and see how we can leverage Azure DevOps and other tools and services to implement a policy for security and compliance for closed and open source alike.

Securing access to package feeds

Package feeds need to have secure access for a variety of reasons. The access should involve allowing:

- **Restricted access for consumption**

Whenever a package feed and its packages should only be consumed by a certain audience, it is required to restrict access to it. Only those allowed access will be able to consume the packages from the feed.

- **Restricted access for publishing**

Secure access is required to restrict who can publish so feeds and their packages cannot be modified by unauthorized or untrusted persons and accounts.

Roles

Azure Artifacts has four different roles for package feeds. These are incremental in the permissions they give.

The roles are in incremental order:

- Reader: Can list and restore (or install) packages from the feed
- Collaborator: Is able to save packages from upstream sources
- Contributor: Can push and unlist packages in the feed
- Owner: has all available permissions for a package feed

When creating an Azure Artifacts feed, the Project Collection Build Service is given contributor rights by default. This organization-wide build identity in Azure Pipelines is able to access the feeds it needs when running tasks. If you changed the build identity to be at project level, you will need also give that identity permissions to access the feed.

Any contributors to the team project are also contributors to the feed. Project Collection Administrators and administrators of the team project, plus the creator of the feed are automatically made owners of the feed. The roles for these users and groups can be changed or removed.

Permissions

The feeds in Azure Artifacts require permission to the various features it offers. The list of permissions consists of increasing privileged operations.

The list of privileges is as follows:

Permission	Reader	Collaborator	Contributor	Owner
List and restore/install packages	✓	✓	✓	✓
Save packages from upstream sources		✓	✓	✓
Push packages			✓	✓
Unlist/deprecate packages			✓	✓
Delete/unpublish package				✓
Edit feed permissions				✓
Rename and delete feed				✓

For each permission you can assign users, teams and groups to a specific role, giving the permissions corresponding to that role. You need to have the Owner role to be able to do so. Once an account has access to the feed from the permission to list and restore packages it is considered a Feed user.

DevOpsCertificationFeed > Feed settings

Feed details Permissions Views Upstream sources | + Add users/groups Delete ...

Filter by User/Group

User/Group	Role
Alex Thissen	Owner
[xpirit]\Project Collection Administrators	Owner
[DevOpsCertification-Course-MS]\Project Administrators	Owner
Project Collection Build Service (xspirit)	Contributor
[DevOpsCertification-Course-MS]\Contributors	Contributor

Just like permissions and roles for the feed itself, there are additional permissions for access to the individual views. Any feed user has access to all the views, whether the default views of @Local, @Release or @Prerelease, or newly created ones. During creation of a feed you can choose whether the feed is visible to people in your Azure DevOps organization or only specific people.

Visibility - Who can use your feed

-  People in xspirit - Members of your organization can view the packages in your feed
-  Specific people - Only people you give access to will be able to view this feed

See also:

[Secure and share packages using feed permissions¹](#)

Credentials

Azure DevOps users will authenticate against Azure Active Directory when accessing the Azure DevOps portal. After being successfully authenticated, they will not have to provide any credentials to Azure Artifacts itself. The roles for the user, based on its identity, or team and group membership, are for authorization. When access is allowed, the user can simply navigate to the Azure Artifacts section of the team project.

The authentication from Azure Pipelines to Azure Artifacts feeds is taken care of transparently. It will be based upon the roles and its permissions for the build identity. The previous section on Roles covered some details on the required roles for the build identity.

The authentication from inside Azure DevOps does not need any credentials for accessing feeds by itself. However, when accessing secured feeds outside Azure Artifacts, such as other package sources, you must

¹ <https://docs.microsoft.com/en-us/azure/devops/artifacts/feeds/feed-permissions>

likely have to provide credentials to authenticate to the feed manager. Each package type has its own way of handling the credentials and providing access upon authentication. The command-line tooling will provide support in the authentication process.

For the build tasks in Azure Pipelines, you will provide the credentials via a Service connection.

Open source software

How software is built

Let's look at using open-source software in building software.

Using open-source software

Packages contain components that are built from source code. Open source code is publicly available for inspection, reuse and contribution. Most commonly open source projects indicate how the sources can be used and distributed afterwards. A license agreement accompanies the source code and specifies what can and cannot be done.

In the previous module we covered how modern software is built by using components. These components are created in part by the team that is writing the entire software solution. Some dependencies are on components created and made available by other teams, third party companies and the community. The packages that contain the components are a formalized way for distribution.

On average the software solution being built is around 80% based on components that exist and are maintained outside of the project. The rest of the solution consists of your code with business logic and specifics for the functional requirements, plus "glue" code that binds together the components and your code.

The components can be a commercial offering or free of charge. A considerable part of the publicly available and free components are community efforts to offer reusable components for everyone to use and build software. The persons creating and maintaining these components often also make the source code available. This is considered to be open-source code as opposed to closed source. Closed source means that the source code is not available, even though components are available.

What is open-source software?

Wikipedia defines open-source software as follows:

"Open-source software (OSS) is a type of computer software in which source code is released under a license in which the copyright holder grants users the rights to study, change, and distribute the software to anyone and for any purpose."

The related open source software development is a collaborative form of software development, involving multiple contributors. Together they create and maintain software and source code using open sources. The use of open-source software is widely adopted now.

Microsoft itself has also embraced open-source software in its own software and the development platforms they offer. The .NET platforms, such as the original .NET Framework and even more so .NET Core, use several components that are created by the open source community and not Microsoft itself. In ASP.NET and ASP.NET Core, many of the frontend development libraries are open-source components, such as jQuery, Angular and React. Instead of creating new components themselves, the teams at Microsoft are leveraging the open-source components and take a dependency on them. The teams are also contributing and investing to the open source components and projects, joining in on the collaborative effort.

Besides adopting external open-source software Microsoft has also made significant parts of their software available as open-source. .NET is a perfect example of how Microsoft has changed its posture towards open-source. It has made the codebase for the .NET Framework and .NET Core available, as well as many other components. The .NET Foundation aims to advocate the needs, evangelize the benefits of the .NET platform, and promote the use of .NET open source for developers.

For more information, see the [.NET foundation website²](#).

Challenge to corporates

All in all, modern software development, including the Microsoft developer platform and ecosystem implies the use of open-source components. This has implications for companies that build software, either commercially or for internal use. The inclusion of software components that are not build by the companies themselves, means that there is no full control over the sources.

Others being responsible for the source code that is used in components used within a company, means that you have to accept the risks involved with it. The source code could:

- **Be of low quality**

This would impact maintainability, reliability and performance of the overall solution

- **Have no active maintenance**

The code would not evolve over time, or be alterable without making a copy of the source code, effectively forking away from the origin.

- **Contain malicious code**

The entire system that includes and uses the code will be compromised. Potentially the entire company's IT and infrastructure is affected.

- **Have security vulnerabilities**

The security of a software system is as good as its weakest part. Using source code with vulnerabilities makes the entire system susceptible to attack by hackers and misuse.

- **Have unfavorable licensing restrictions**

The effect of a license can affect the entire solution that uses the open-source software.

The companies will have to make a trade-off: its developers want to be able to use open-source software components, allowing them to speed up development and use modern frameworks, libraries and practices. On the other hand, giving the developers and projects the freedom to include open-source software should not put the company at risk. The challenges to the company are in finding a way to keep the developers empowered and free to choose technology to use, while making sure the risks for the company are managed as well as possible.

Other challenges comes from companies that offer open-source software to the public. These challenges include having a business model around the open-source, when to publish open-source code and how to deal with community contributions. The fact that your source code is open, doesn't imply that anyone can make changes to it. There can be contributions from community collaboration, but a company does not necessarily have to accept it. This is referred to as closed open-source. Suggestions for change are welcome, but the maintainers are the one that carry out the actual changes.

Licenses and vulnerabilities

Open-source software and the related source code is accompanied by a license agreement. The license describes the way the source code and the components built from it can be used and how any software created with it should deal with it.

According to the open source definition of OpenSource.org a license should not:

- Discriminate against persons or groups
- Discriminate against fields of endeavour

² <http://www.dotnetfoundation.org>

- Be specific to a product
- Restrict other software
- and more - See the **Open Source Definition³**

To cover the exact terms of a license several types exist. Each type has its own specifics and implications, which we will cover in the next part.

Even though open-source software is generally developed by multiple contributors from the community, it does not guarantee that the software is secure and without vulnerabilities. Chances are they are discovered by being inspected by multiple reviewers, but the discovery might not be immediate or before being consumed by others.

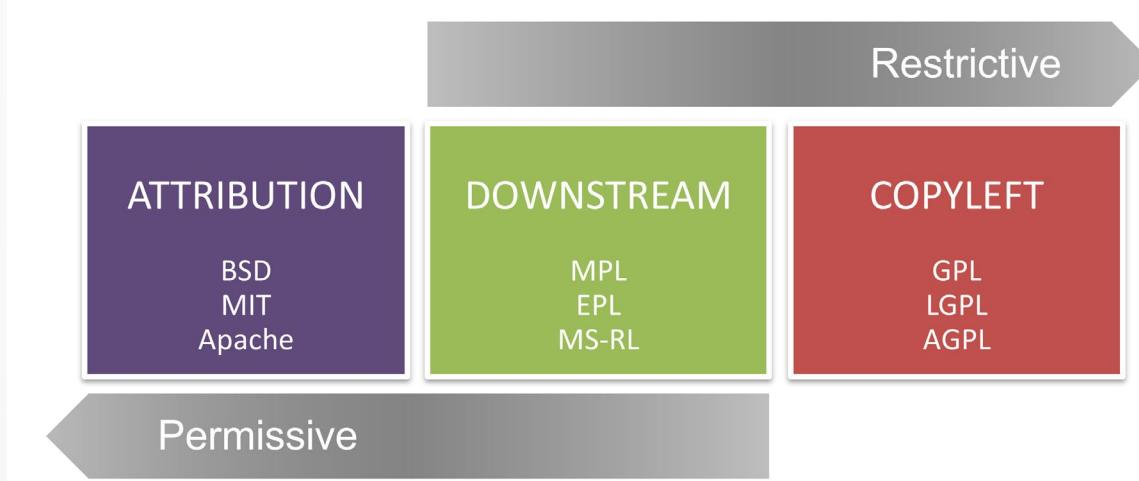
Since the source code is open-source, people with malicious intent can also inspect the code for vulnerabilities and exploit these when possible. In that regard, it is both a blessing and a curse that open-source software has source code available for others.

Open-source package licensing

In the current and previous module we have talked about software components from the perspective of packages. Packages are the formalized ways to distribute software components. The licensing types and concerns about vulnerabilities extend to the packages, as these contain the components.

Types of licenses

There are multiple licenses used in open-source and they are different in nature. The license spectrum is a chart that shows licenses from the perspective of the developer and the implications of use for downstream requirements that are imposed on the overall solution and source code.



On the left side there are the “attribution” licenses. They are permissive in nature and allow practically every type of use by the software that consumes it. An example is building commercially available software including the components or source code under this license. The only restriction is that the original attribution to the authors remains included in the source code or as part of the downstream use of the new software.

³ <http://opensource.org/osd>

The right side of the spectrum shows the "copyleft" licenses. These licenses are considered viral in nature, as the use of the source code and its components, and distribution of the complete software, implies that all source code using it should follow the same license form. The viral nature is that the use of the software covered under this license type forces you to forward the same license for all work with or on the original software.

The middle of the spectrum shows the "downstream" or "weak copyleft" licenses. It also requires that when the covered code is distributed, it must do so under the same license terms. Unlike the copyleft licenses this does not extend to improvements or additions to the covered code.

License implications and rating

Any form of software that uses code or components must take into account the license type that is covered. For companies it becomes important to know the types of licenses for components and packages that developers choose to use. When these include even one viral license, it requires that all software must be using the same license. Any intellectual property you might have, must be made public and open-source according to the terms of the viral license type. This has tremendous impact on the source code of the project and consequently for the company that produces it.

License rating

Licenses can be rated by the impact that they have. When a package has a certain type of license, the use of the package implies keeping to the requirements of the package. The impact the license has on the downstream use of the code, components and packages can be rated as High, Medium and Low, depending on the copy-left, downstream or attribution nature of the license type.

For compliancy reasons, a high license rating can be considered a risk for compliancy, intellectual property and exclusive rights.

Package security

The use of components creates a software supply chain. The resultant product is a composition of all its parts and components. This applies to the security level of the solution as well. Therefore, similar to license types it is important to know how secure the components being used are. If one of the components used is not secure, then the entire solution isn't either. We will talk more on package security and vulnerabilities in the next chapter.

Integrating license and vulnerability scans

Artifact repositories

Let's learn about artifact repositories and their use, plus tooling to help in identifying security vulnerabilities and license and compliancy issues.

Looking at package management and the challenges of keeping control of licenses and vulnerabilities, every feed and its packages that is consumed is a potential risk. Normally, the build tooling will interact with internal and potentially external package feeds. Upstream sources can be a way to avoid direct access to external feeds.

If each developer, team and build process would be able to connect directly to external feeds, it is hard to perform package management from the perspective of licensing and security vulnerabilities.

Artifact repositories are used to store software artifacts in a centralized, internal place or flow, much like package feeds. The repository becomes the single source of packages for the entire organization, facilitating an overview of the packages in use by each of the consumers.

Tools for assessing package security and license rating

There are several tools available from third parties to help in assessing the security and license rating of software packages that are in use.

One approach by these tools is to provide the centralized artifact repository as discussed in the previous section. Scanning can be done at any particular time, inspecting the packages that are part of the repository.

The second approach is to use tooling that scans the packages as they are used in a build pipeline. During the build process, the tool can scan the packages by the particular build, giving instantaneous feedback on the packages in use.

Inspect packages in delivery pipeline

There is tooling available to perform security scans on packages, components and source code while running a delivery pipeline. Often such tooling will use the build artifacts during the build process and perform scans.

The tooling can take either of the approaches of working on a local artifact repository or the intermediary build output. Some examples for each are products like:

Tool	Type
Artifactory	Artifact repository
SonarQube	Static code analysis tool
WhiteSource (Bolt)	Build scanning

Configure pipeline

The configuration of the scanning for license types and security vulnerability in the pipeline is done by using appropriate build tasks in your DevOps tooling. For Azure DevOps these are build pipeline tasks.

WhiteSource is a third party product that offers both a paid and free version to use in Azure Pipelines. The tool uses the local build artifacts on the build server and runs directly from there. It will scan for the

various package types used in the build and analyze those found. This requires external connectivity. The results of the analysis are returned in the build results as part of the step for the task.

Demonstration Whitesource Bolt

To correctly interpret the results of scanning tools, you need to be aware of some aspects:

- **False positives**

It is important to verify the findings to be real positives in terms of the scan results. The tooling is an automated way to scan and might be misinterpreting certain vulnerabilities. In the triaging of the finding in the scan results, you should be aware that some findings might not be correct. Such results are called *false positives*, established by human interpretation and expertise. One must not declare a result a false positive too easily. On the other hand, scan results are not guaranteed to be 100% accurate.

- **Security bug bar**

Most likely there will be many security vulnerabilities detected. Some of these *false positives*, but still a lot of findings. Quite often there are more findings than can be handled or mitigated, given a certain amount of time and money. In such cases it is important that there is a security bug bar, indicating the level of vulnerabilities that must be fixed before the security risks are acceptable enough to take the software into production. The bug bar makes sure that it is clear what must be taken care of, and what might be done if there is time and resources left.

The WhiteSource Bolt tooling gives aggregated reporting that can be found under the Pipelines section. It shows three different sections:

- **Security vulnerabilities**

The results from the scan are usually distinguishing between various levels of severity. The severity is a good candidate for establishing the bug bar. For example, the security bug bar could be set to not allow any findings on High or Medium vulnerability level before releasing software to production.

- **License risks and compliance**

A list of license types that are used. The higher the risk level of the licenses in use that are identified, the bigger chance of compliance issues.

- **Outdated libraries**

Libraries that are not using the latest version. Although these do not pose a security risk (yet), they might cause maintenance problems and introduce other risks for the software that uses these libraries, components and packages they come in.

The results of the scan tooling will be the basis for selecting what work remains to be done before software is considered stable and done. By setting a security bug bar in the Definition of Done and specifying the allowed license ratings, one can use the reports from the scans to find the work for the development team.

Implement a versioning strategy

Introduction to versioning

Software changes over time. The requirements for the software do not stay the same. The functionality it offers and how it is used will grow, change and adopt based on feedback. The hosting of an application might evolve as well, with new operating systems, new frameworks and versions thereof. The original implementation might contain flaws and bugs. Whatever reason for change, it is unlikely that software is stable and doesn't need to change. Since the software you build takes dependencies on other components, the same holds true for the components and packages you build or use while building your software.

In order to keep track of which piece of software is currently being used, correct versioning becomes essential to maintaining a codebase. Versioning is also relevant for dependency management, as it relates to the versions of the packages and the components within. Each dependency is clearly identified by its name and version. It allows keeping track of the exact packages being used. Each of the packages has its own lifecycle and rate of change.

Immutable packages

As packages get new versions, your codebase can choose when to use a new version of the packages it consumes. It does so by specifying the specific version of the package it requires. This implies that packages themselves should always have a new version when they change. Whenever a package is published to a feed it should not be allowed to change any more. If it were, it would be at the risk of introducing potential breaking changes to the code. In essence, a published package is considered to be immutable. Replacing or updating an existing version of a package is not allowed. Most of the package feeds do not allow operations that would change an existing version. Regardless of the size of the change a package can only be updated by the introduction of a new version. The new version should indicate the type of change and impact it might have.

See also [Key concepts for Azure Artifacts⁴](#).

Versioning of artifacts

It is proper software development practices to indicate changes to code with the introduction of an increased version number. However small or large a change, it requires a new version. A component and its package can have independent versions and versioning schemes.

The versioning scheme can differ per package type. Typically, it uses a scheme that can indicate the type of change that is made. Most commonly this involves three types of changes:

- **Major change**

Major indicates that the package and its contents have changed significantly. It often occurs at the introduction of a complete new version of the package. This can be at a redesign of the component. Major changes are not guaranteed to be compatible and usually have breaking changes from older versions. Major changes might require a substantial amount of work to adopt the consuming codebase to the new version.

- **Minor change**

Minor indicates that the package and its contents have substantial changes made, but are a smaller

⁴ <https://docs.microsoft.com/en-us/azure/devops/artifacts/artifacts-key-concepts#immutability>

increment than a major change. These changes can be backward compatible from the previous version, although they are not guaranteed to be.

- **Patch**

A patch or revision is used to indicate that a flaw, bug or malfunctioning part of the component has been fixed. Normally, this is a backward compatible version compared to the previous version.

How artifacts are versioned technically varies per package type. Each type has its own way of indicating the version in metadata. The corresponding package manager is able to inspect the version information. The tooling can query the package feed for packages and the available versions.

Additionally, a package type might have its own conventions for versioning as well as a particular versioning scheme.

See also **Publish to NuGet feeds⁵**

Semantic versioning

One of the predominant ways of versioning is the use of semantic versions. It is not a standard per se, but does offer a consistent way of expressing intent and semantics of a certain version. It describes a version in terms of its backward compatibility to previous versions.

Semantic versioning uses a three part version number and an additional label. The version has the form of Major.Minor.Patch, corresponding to the three types of changes covered in the previous section. Examples of versions using the semantic versioning scheme are 1.0.0 and 3.7.129. These versions do not have any labels.

For prerelease versions it is customary to use a label after the regular version number. A label is a textual suffix separated by a hyphen from the rest of the version number. The label itself can be any text describing the nature of the prerelease. Examples of these are rc1, beta27 and alpha, forming version numbers like 1.0.0-rc1 as a prerelease for the upcoming 1.0.0 version.

Prereleases are a common way to prepare for the release of the label-less version of the package. Early adopters can take a dependency on a prerelease version to build using the new package. In general it is not a good idea to use prerelease version of packages and their components for released software.

It is good to anticipate on the impact of the new components by creating a separate branch in the codebase and use the prerelease version of the package. Changes are that there will be incompatible changes from a prerelease to the final version.

See also **Semantic Versioning 2.0.0⁶**.

Release views

When building packages from a pipeline, the package needs to have a version before the package can be consumed and tested. Only after testing is the quality of the package known. Since package versions cannot and should not be changed, it becomes challenging to choose a certain version beforehand.

Azure Artifacts recognizes a quality level of packages in its feeds and the difference between prerelease and release versions. It offers different views on the list of packages and their versions, separating these based on their quality level. It fits well with the use of semantic versioning of the packages for predictability of the intent of a particular version, but is additional metadata from the Azure Artifacts feed called a descriptor.

⁵ <https://docs.microsoft.com/en-us/azure/devops/pipelines/artifacts/nuget#package-versioning>

⁶ <https://semver.org/>

Feeds in Azure Artifacts have three different views by default. These view are added at the moment a new feed is created. The three views are:

- **Release**

The @Release view contains all packages that are considered official releases.

- **Prerelease**

The @Prerelease view contains all packages that have a label in their version number.

- **Local**

The @Local view contains all release and prerelease packages as well as the packages downloaded from upstream sources.

Using views

You can use views to offer help consumers of a package feed to filter between released and unreleased versions of packages. Essentially, it allows a consumer to make a conscious decision to choose from released packages, or opt-in to prereleases of a certain quality level.

By default the @Local view is used to offer the list of available packages. The format for this URI is:

```
https://pkgs.dev.azure.com/{yourteamproject}/_packaging/{feedname}/nuget/v3/index.json
```

When consuming a package feed by its URI endpoint, the address can have the requested view included. For a specific view, the URI includes the name of the view, which changes to be:

```
https://pkgs.dev.azure.com/{yourteamproject}/_packaging/{feedname}@{Viewname}/nuget/v3/index.json
```

The tooling will show and use the packages from the specified view automatically.

Tooling may offer an option to select prerelease versions, such as shown in this Visual Studio 2017 NuGet dialog. This does not relate or refer to the @Prerelease view of a feed. Instead, it relies on the presence of prerelease labels of semantic versioning to include or exclude packages in the search results.

See also:

- [Views on Azure DevOps Services feeds⁷](#)
- [Communicate package quality with prerelease and release views⁸](#)

Promoting packages

Azure Artifacts has the notion of promoting packages to views as a means to indicate that a version is of a certain quality level. By selectively promoting packages you can plan when packages have a certain quality and are ready to be released and supported by the consumers.

You can promote packages to one of the available views as the quality indicator. The two views Release and Prerelease might be sufficient, but you can create more views when you want finer grained quality levels if necessary, such as alpha and beta.

Packages will always show in the Local view, but only in a particular view after being promoted to it. Depending on the URL used to connect to the feed, the available packages will be listed.

⁷ <https://docs.microsoft.com/en-us/azure/devops/artifacts/concepts/views>

⁸ <https://docs.microsoft.com/en-us/azure/devops/artifacts/feeds/views>

Upstream sources will only be evaluated when using the @Local view of the feed. After they have been downloaded and cached in the @Local view, you can see and resolve the packages in other views after they have promoted to it.

It is up to you to decide how and when to promote packages to a specific view. This process can be automated by using a Azure Pipelines task as part of the build pipeline.

Packages that have been promoted to a view will not be deleted based on the retention policies.

Demonstration Promoting a Package

Prerequisite: Have access to an existing Azure DevOps project and the connected package feed from the previous demo

Steps to demonstrate the views that exist on package feeds in Azure Artifacts

1. Go to dev.azure.com and open your team project.
2. Open **Artifacts** and select the feed **PartsUnlimited.Security**.
3. Go to **Settings** and click **Feed Settings**.
4. Open the **Views** tab. By default there will be three views. Local: includes all packages in the feed and all cached from upstream sources. Prerelease and Release. In the **Default view** column is a checkmark behind Local. This is the default view that will always be used.

Steps to use the release view instead

1. Open Visual Studio and open NuGet Package Manager.
2. Click the settings wheel and check the source address for the PartsUnlimited feed. If you want to use a different view than the local view, you need to include that in the Source url of your feed, by adding `@Release`.
3. Add `@Release` to the source url `./PartsUnlimited@Release/nuget/..` and click **Update**.
4. **Refresh** the Browse tab. You will see there are **No packages found** in the Release feed. Before any packages will appear, you need to promote them.

Steps to promote packages to particular views

1. Go back to your feed in Azure Artifacts.
2. Click on the created NuGet Package **PartsUnlimited.Security**.
3. Click **Promote**.
4. Select the feed you want to use, in this case **PartsUnlimited@Release** and Promote.
5. Go back to the **Overview**. If we look again at our package in the feed, you will notice there is now a Release tag associated with this particular package.
6. Go back to Visual Studio, **NuGet Package Manager**.
7. **Refresh** the Browse tab. You will see that your version is promoted to this view.
8. Select **PartsUnlimited.Security** and click **Update** and **OK**. The latest version of this package is now used.

Best practices for versioning

There are a number of best practices to effectively use versions, views and promotion flow for your versioning strategy. Here are a couple of suggestions:

- Have a documented versioning strategy
- Adopt SemVer 2.0 for your versioning scheme
- Each repository should have a unique feed
- When creating a package, also publish it to the unique feed

It is good to adopt a best practices yourself and share these in your development teams. It can be made part of the Definition of Done for work items that relate to publishing and consuming packages from feeds.

See also [Best practices for using Azure Artifacts⁹](#).

Demonstration Pushing from the Pipeline

Prerequisite: In your Azure DevOps PartsUnlimited project, prepare two builds pipelines (used in previous demos)

1. Build pipeline **PartsUnlimited Security Package**.

- **.NET Core** build task (Restore, build, push)
- enable CI trigger

2. Build pipeline **PartsUnlimited E2E**.

- **ASP.net** web application type
- NuGet restore task

Steps to build and push the Nuget package

1. Edit the build pipeline **PartsUnlimited Security Package**.

- dotnet restore
 - dotnet build
 - dotnet push
-
- Use the command `nuget push` and specify path `**/*.*nupkg`
 - Select target feed PartsUnlimited

2. Start a new build and select agent pool.

The build has succeeded, but you will see the final step **dotnet push** fails. The reason for this failure can be found in the log information.

3. Open the log information.

⁹ <https://docs.microsoft.com/en-us/azure/devops/artifacts/concepts/best-practices>

It shows the feed already contains the **PartsUnlimited.Security 1.0.0**. We go back to the Visual Studio project to see what is happening.

4. Open the source code for the PartsUnlimited package in Visual Studio in a separate solution.

- Open the **Project Properties**
- Go to the package tab

Look at Package version, we see that the version is still 1.0.0. Packages are immutable. As soon as a package is published to a feed there can never be another package with the exact same version. We need to upgrade the version to a new one that uses the major, minor and the changed patch version.

5. Change the patch version: 1 . 0 . 1. Make a small edit to the code for illustrative purposes, and check in the new code.
6. Change the **exception type** check in, commit and push the new code. We go back to the Azure Devops pipeline. Since there is a trigger on the code we just changed, the build will automatically start.
7. Go back to build pipeline for **PartsUnlimited Security Package**.

As you see the build is triggered and completed successfully. Including the push to the NuGet feed. Since there was no version conflict, we were able to successfully push the new version to the feed.

8. Open **Artifacts** and show the feed.

Since there was a successful build for the entire web application, you can see that the PartsUnlimited feed now also includes all the downloaded upstream packages from the NuGet.org source.

9. Scroll down and click on the **PartsUnlimited.Security 1.0.1** package. By clicking on it we can inspect the details and the versions.
10. Edit the build pipeline **PartsUnlimited E2E**.

11. Click **NuGet restore**.

There is a second pipeline that builds the complete web application. It is an ASP.net web application build. The NuGet restore task is configured to also consume packages from the PartsUnlimited feed. Because PartsUnlimited has an upstream source for NuGet.org we do not have to **Use packaged from NuGet.org** explicitly. You can uncheck this box.

Lab

Manage Open Source Security and License with WhiteSource

In this lab, **Managing Open-source security and license with WhiteSource¹⁰**, you will use WhiteSource Bolt with Azure DevOps to automatically detect alerts on vulnerable open source components, outdated libraries, and license compliance issues in your code. You will be using WebGoat, a deliberately insecure web application, maintained by OWASP designed to teach web application security lessons. You will learn how to:

- Detect and remedy vulnerable open source components.
 - Generate comprehensive open source inventory reports per project or build.
 - Enforce open source license compliance, including dependencies' licenses.
 - Identify outdated open source libraries with recommendations to update.
- ✓ Note: You must have already completed the prerequisite labs in the Welcome section.

¹⁰ <https://www.azuredevopslabs.com/labs/vstsextend/WhiteSource/>

Module Review and Takeaways

Module Review Questions

Suggested answer

What issues are often associated with the use of open source libraries?

Suggested answer

How can an open source library cause licensing issues if it is free to download?

Suggested answer

What is the minimum feed permission that will allow you to list available packages and to install them?

Suggested answer

What is open source software?

Suggested answer

How can you restrict which files are uploaded in a universal package feed?

Answers

What issues are often associated with the use of open source libraries?

Bugs, security vulnerabilities, and licensing issues

How can an open source library cause licensing issues if it is free to download?

Each library has usage restrictions as part of the licensing. These restrictions might not be compatible with your intended application use.

What is the minimum feed permission that will allow you to list available packages and to install them?

Reader

What is open source software?

A type of software where users of code are permitted to study, change, and distribute the software. The open source license type can limit the actions (such as sale provisions) that can be taken.

How can you restrict which files are uploaded in a universal package feed?

By using an .artifactignore file

Module 10 Design a Release Strategy

Module Overview

Module Overview

Welcome to this module about designing a release strategy. In this module, we will talk about Continuous Delivery in general. In this introduction, we will cover the basics. I'll explain the concepts of Continuous Delivery, Continuous Integration and Continuous Deployment and also the relation to DevOps, and we will discuss why you would need Continuous Delivery and Continuous Deployment. After that, we will talk about releases and deployments and the differences between those two.

Once we have covered these general topics, we will talk about release strategies and artifact sources, and walk through some considerations when choosing and defining those. We will also discuss the considerations for setting up deployment stages and your delivery and deployment cadence, and lastly about setting up your release approvals.

After that, we will cover some ground to create a high-quality release pipeline and talk about the quality of your release process and the quality of a release and difference between those two. We will take a look at how to visualize your release process quality and how to control your release using release gates as a mechanism. Finally, we will look at how to deal with release notes and documentation.

After these introductions, we will take a brief look at deployment patterns. We will cover modern deployment patterns like canary releases, but we will also take a quick look at the traditional deployment patterns, like DTAP environments.

Finally, we take a look at choosing the right release management tool. There are a lot of tools out there. We will cover the components that you need to take a look at if you are going to choose the right release management tool product or company.

Learning objectives

At the end of this module, students will be able to:

- Differentiate between a release and a deployment
- Define the components of a release pipeline

- Explain things to consider when designing your release strategy
- Classify a release versus a release process, and outline how to control the quality of both
- Describe the principle of release gates and how to deal with release notes and documentation
- Explain deployment patterns, both in the traditional sense and in the modern sense
- Choose a release management tool

Introduction to Continuous Delivery

Traditional IT

We will talk about Continuous Delivery and why this is something you should consider.

We will first talk about Continuous Delivery in general. What is Continuous Delivery, what does it mean, what problems does it solve, and why should you care?

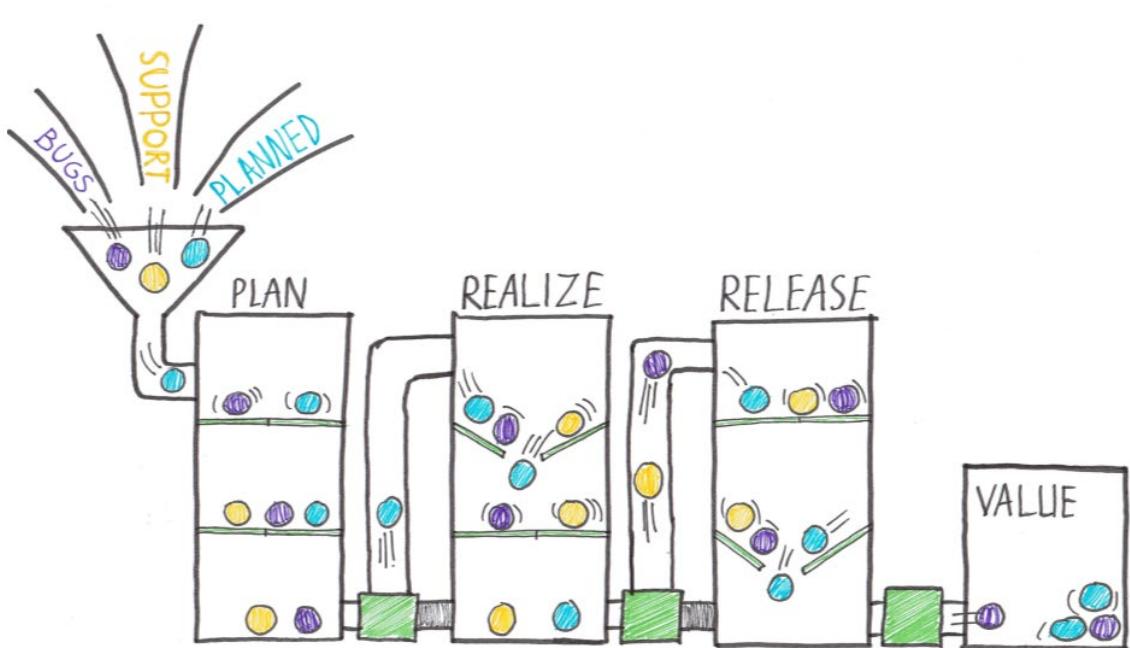
A few years ago, IT was a facilitating department. IT was there to support the business users, and because time had proven that developed software had bad quality by default, software changes were a risk. The resolution for this "quality problem" was to keep changes under strict control. The department that became responsible for controlling the changes became the IT(-Pro) department. In the past but also today, the IT(-Pro) department is responsible for the stability of the systems, while the development department is responsible for creating new value.

This split brings many companies in a difficult situation. Development departments are motivated to deliver value as soon as possible to keep their customers happy. On the other hand, IT is motivated to change nothing, because change is a risk and they are responsible for eliminating the risks and keeping everything stable. And what do we get out of this? Long release cycles.

Silo-Based Development

Long release cycles, a lot of testing, code freezes, night and weekend work and a lot of people involved, ensure that everything works. But the more we change, the more risk it entails, and we are back at the beginning. On many occasions resulting in yet another document or process that should be followed.

This is what I call silo-based development.

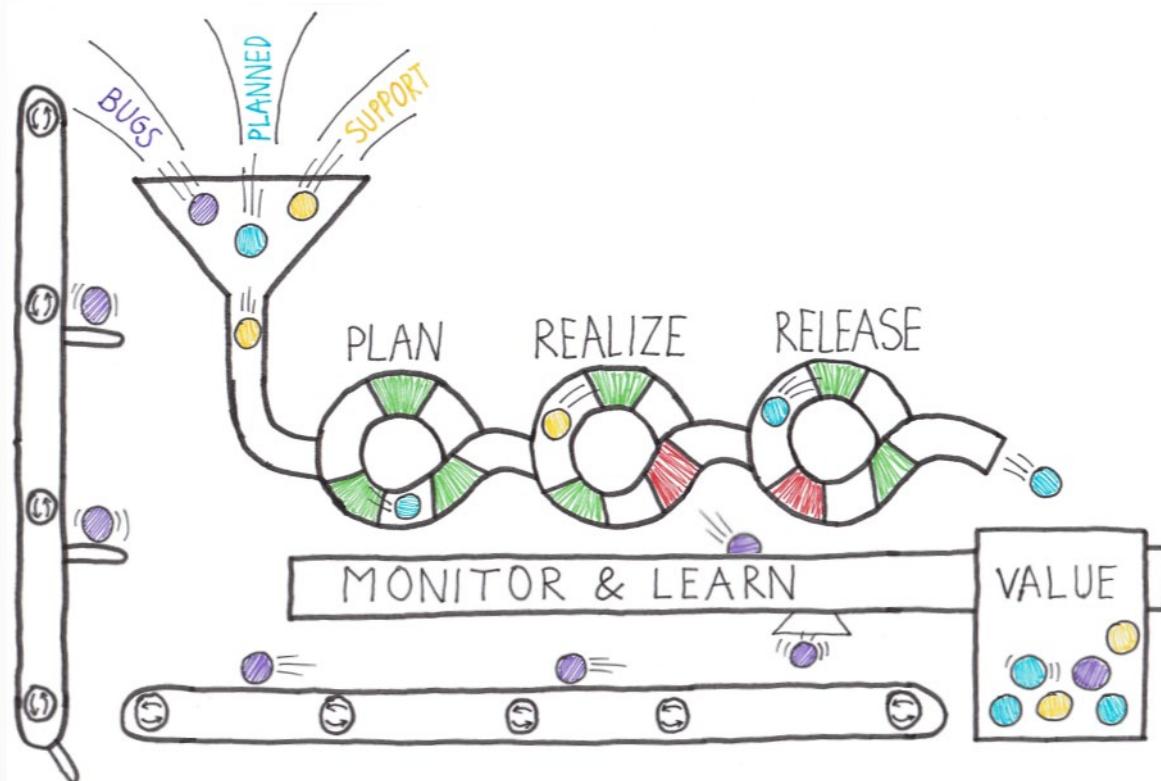


If we look at this picture of a traditional, silo-based value stream, we see Bugs and Unplanned work, necessary updates or support work and planned (value adding) work, all added to the backlog of the

teams. When everything is planned and the first "gate" can be opened, everything drops to the next phase. All the work, and thus all the value moves in piles to the next phase. It moves from Plan phase to a Realize phase where all the work is developed, tested and documented, and from here, it moves to the release phase. All the value is released at the same time. As a result, the release takes a long time.

Moving to Continuous Delivery

But times have changed, and we need to deal with a new normal. Our customers demand working software, and they wanted it yesterday. If we cannot deliver, they go to a competitor. And competition is fierce. With the internet, we always have global competition. With competitors on our whole stack but also with competitors that deliver a best of breed tool for one aspect of the software we built. We need to deliver fast, and the product we make must be good. And we should do this with our software production being cheap and quality being high. To achieve this, we need something like Continuous Delivery.



We need to move towards a situation where the value is not piled up and released all at once, but where value flows through a pipeline. Just like in the picture, a piece of work is a marble. And only one piece of work can flow through the pipeline at once. So work has to be prioritized in the right way. As you can see the pipeline has green and red outlets. These are the feedback loops or quality gates that we want to have in place.

A feedback loop can be different things:

- A unit test to validate the code
- An automated build to validate the sources
- An automated test on a Test environment
- Some monitor on a server
- Usage instrumentation in the code

If one of the feedback loops is red, the marble cannot pass the outlet and it will end up in the Monitor and Learn tray. This is where the learning happens. The problem is analyzed and solved so that the next time a marble passes the outlet, it is green.

Every single piece of work flows through the pipeline until it ends up in the tray of value. The more that is automated the faster value flows through the pipeline.

Companies want to move toward Continuous Delivery. They see the value. They hear their customers. Companies want to deliver their products as fast as possible. Quality should be higher. The move to production should be faster. Technical Debt should be lower.

A great way to improve your software development practices was the introduction of Agile and Scrum. Last year around 80% of all companies claimed that they adopted Scrum as a software development practice. By using Scrum, many teams can produce a working piece of software after a sprint of maybe 2 or 3 weeks. But producing working software is not the same as delivering working software. The result is that all "done" increments are waiting to be delivered in the next release, which is coming in a few months.

What we see now, is that Agile teams within a non-agile company are stuck in a delivery funnel. The bottleneck is no longer the production of working software, but the problem has become the delivery of working software. The finished product is waiting to be delivered to the customers to get business value, but this does not happen. Continuous Delivery needs to solve this problem.

What is Continuous Delivery?

Now that we know a bit more about the necessity to move towards Continuous Delivery, it is good to spend some time on the definition of Continuous Delivery and how this relates to DevOps.

Continuous Delivery (CD) is a set of processes, tools and techniques for the rapid, reliable and continuous development and delivery of software.

This means that Continuous Delivery goes beyond the release of software through a pipeline. The pipeline is a crucial component and also the main focus of this course, but Continuous Delivery is more.

To explain this a bit more, look at the eight principles of Continuous Delivery:

1. The process for releasing/deploying software must be repeatable and reliable.
2. Automate everything!
3. If something is difficult or painful, do it more often.
4. Keep everything in source control.
5. Done means "released."
6. Build quality in!
7. Everybody has responsibility for the release process.
8. Improve continuously.

If we want to fulfill these 8 principles, we can see that an automated pipeline does not suffice. In order to deploy more often, we need to reconsider our software architecture (monoliths are hard to deploy), our testing strategy (manual tests do not scale very well), our organization (separated business and IT departments do not work smoothly), and so forth.

This course will focus on the release management part of Continuous Delivery, but be aware of the other changes you might encounter. Find the next bottleneck in your process, solve it and learn from it, and repeat this forever.

How does this all relate to DevOps? If we look at the definition of DevOps from Donovan Brown:

"DevOps is the union of people, process, and products to enable Continuous Delivery of value to our end users."

Looking at this definition, Continuous Delivery is an enabler for DevOps. DevOps focuses on organizations and bringing people together to Build and Run their software products.

Continuous Delivery is a practice. Being able to deliver software on-demand. Not necessarily a 1000 times a day. Deploying every code change to production is what we call Continuous Deployment.

To be able to do this we need automation, we need a strategy, and we need pipelines. And this is what we will cover in the rest of this module.

Releases and Deployments

One of the essential steps in moving software more quickly to production is by changing the way we deliver the software to production. In our industry, it is widespread that we have teams that need to do overtime in the weekend to install and release new software. This is mainly caused by the fact that we have two parts of the release process bolted together. The moment we deploy the new software, we also release new features to the end users.

The best way to move your software to production safely while maintaining stability is by separating these two concerns. So we separate deployments from our release. This can also be phrased as separating your functional release from your technical release (deployment).

What is a release and what is a deployment

When we talk about releases and deployments, we see that commonly used tools deal a bit differently with the terminology as we did in the previous chapter. To make sure you both understand the concepts and the technical implementation in many tools, you need to know how tool vendors define the difference between a release and a deployment.

A release is a package or container that holds a versioned set of artifacts specified in a release pipeline in your CI/CD process. It includes a snapshot of all the information required to carry out all the tasks and actions in the release pipeline, such as the stages (or environments), the tasks for each one, the values of task parameters and variables, and the release policies such as triggers, approvers, and release queuing options. There can be multiple releases from one release pipeline (or release process).

Deployment is the action of running the tasks for one stage, which results in a tested and deployed application, and other additional activities that are specified for that stage. Initiating a release starts each deployment based on the settings and policies defined in the original release pipeline. There can be multiple deployments of each release even for one stage. When a deployment of a release fails for a stage, you can redeploy the same release to that stage.

See also [Releases in Azure Pipelines](#)¹.

Separating technical and functional release

When we want to get started with separating the technical and functional release, we need to start with our software itself. The software needs to be built in such a way that new functionality or features can be hidden from end-users while it is running.

A common way to do this, is the use of Feature Toggles. The simplest form of a Feature Toggle is an if statement that either executes or does not execute a certain piece of code. By making the if-statement

¹ <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/releases?view=vsts>

configurable you can implement the Feature Toggle. We will talk about Feature Toggles in Module 3 in more detail.

See also: [Explore how to progressively expose your features in production for some or all users²](#).

Once we have prepared our software, we need to make sure that the installation will not expose any new or changed functionality to the end user.

When the software has been deployed, we need to watch how the system behaves. Does it act the same as it did in the past?

If it is clear that the system is stable and operates the same as it did before, we can decide to flip a switch. This might reveal one or more features to the end user, or change a set of routines that are part of the system.

The whole idea of separating deployment from release (exposing features with a switch) is compelling and something we want to incorporate in our Continuous Delivery practice. It helps us with more stable releases and better ways to roll back when we run into issues when we have a new feature that produces problems.

We switch it off again and then create a hotfix. By separating deployment from the release of a feature, you create the opportunity to deploy any time of the day, since the new software will not affect the system that already works.

Discussion

What are your bottlenecks?

Discuss the need for Continuous Delivery in your organization and what blocks the implementation.

Topics you might want to discuss are:

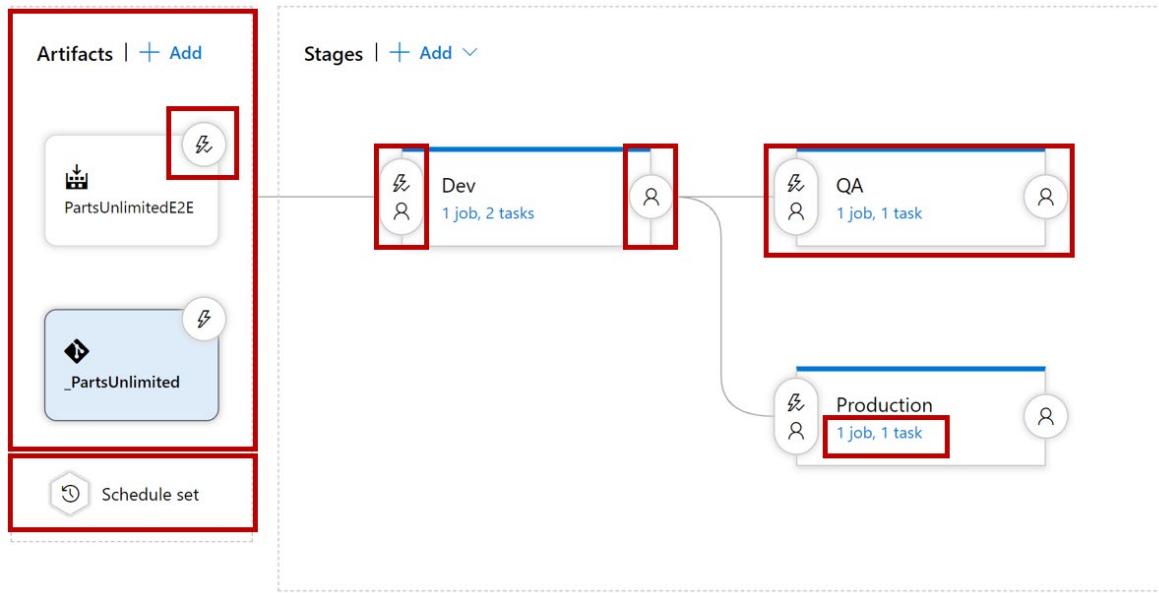
- Does your organization need Continuous Delivery?
- Do you use Agile/Scrum?
 - Is everybody involved or only the Dev departments?
 - Can you deploy your application multiple times per day? Why or why not?
 - What is the main bottleneck for Continuous Delivery in your organization?
 - The Organization
 - Application Architecture
 - Skills
 - Tooling
 - Tests
 - other things?

² <https://docs.microsoft.com/en-us/azure/devops/articles/phase-features-with-feature-flags?view=vsts>

Release strategy recommendations

Introduction and Overview

Different components make up a release pipeline. In short, a release pipeline takes an artifact and moves this artifact through different stages, so it can eventually be installed on a production environment.



Let us quickly walk through all the components step by step.

The first component in a release pipeline is an artifact. Artifacts can come from different sources. The most common source is a package from a build pipeline. Another commonly seen artifact source is for example source control. Furthermore, a release pipeline has a trigger: the mechanism that starts a new release.

A trigger can be:

- A manual trigger, where people start to release by hand
- A scheduled trigger, where a release is triggered based on a specific time, or
- A continuous deployment trigger, where another event triggers a release. For example, a completed build.

Another vital component of a release pipeline are stages or sometimes called environments. This is where the artifact will be eventually installed. For example, the artifact contains the compiled website, and this will be installed on the web server or somewhere in the cloud. You can have many stages (environments), and part of the release strategy is to find out what the appropriate combination of stages is.

Another component of a release pipeline is approval. In many cases, people want to sign off a release before it is installed on the environment. In more mature organizations, this manual approval process can be replaced by an automatic process that checks the quality before the components move on to the next stage.

Finally, we have the tasks within the various stages. The tasks are the steps that need to be executed to install, configure, and validate the installed artifact.

In this part of the module, we will walk through all the components of the release pipeline in detail and talk about what to consider for each component.

The components that make up the release pipeline or process are used to create a release. There is a difference between a release and the release pipeline or process.

The release pipeline is the blueprint through which releases are done. We will cover more of this when discussing the quality of releases and releases processes.

See also **Release pipelines**³.

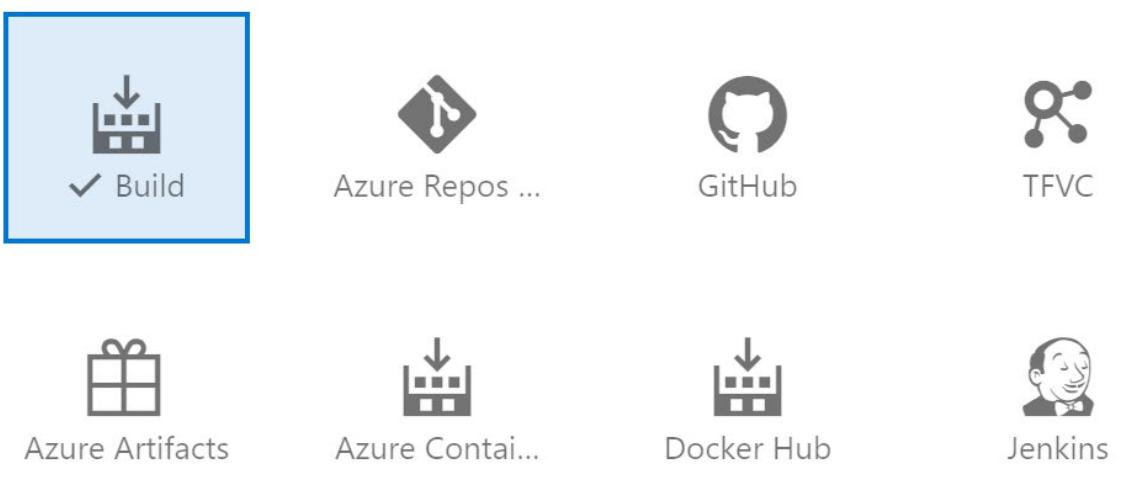
Artifacts and Artifact sources

What is an artifact? An artifact is a deployable component of your application. These components can then be deployed to one or more environments. In general, the idea about build and release pipelines and Continuous Delivery is to build once and deploy many times. This means that an artifact will be deployed to multiple environments. To achieve this, this implies that the artifact is a stable package. The only thing that you want to change when you deploy an artifact to a new environment is the configuration. The contents of the package should never change. This is what we call **immutability**⁴. We should be 100% sure that the package that what we build, the artifact, remains unchanged.

How do we get an artifact? There are different ways to create and retrieve artifacts, and not every method is appropriate for every situation.

Add an artifact

Source type



The most common and most used way to get an artifact within the release pipeline is to use a build artifact. The build pipeline compiles, tests, and eventually produces an immutable package, which is stored in a secure place (storage account, database etc.).

The release pipeline then uses a secure connection to this secured place to get the build artifact and perform additional actions to deploy this to an environment. The big advantage of using a build artifact is

³ <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/?view=vsts>

⁴ <https://docs.microsoft.com/en-us/azure/devops/artifacts/artifacts-key-concepts?view=vsts>

that the build produces a versioned artifact. The artifact is linked to the build and gives us automatic traceability. We can always find the sources that produced this artifact.

Another possible artifact source is version control. We can directly link our version control to our release pipeline. The release is then related to a specific commit in our version control system. With that, we can also see which version of a file or script is eventually installed. In this case, the version does not come from the build, but from version control. A consideration for choosing a version control artifact instead of a build artifact can be that you only want to deploy one specific file. If no additional actions are required before this file is used in the release pipeline, it does not make sense to create a versioned package containing one that file. Helper scripts that perform actions to support the release process (clean up, rename, string actions) are typically good candidates to get from version control.

Another possibility of an artifact source can be a network share containing a set of files. However, you should be aware of the possible risk. The risk is that you are not 100% sure that the package that you are going to deploy is the same package that was put on the network share. If other people can access the network share as well, the package might be compromised. For that reason, this option will not be sufficient to prove integrity in a regulated environment (banks, insurance companies).

Last but not least, container registries are a rising star when it comes to artifact sources. Container registries are versioned repositories where container artifacts are stored. By pushing a versioned container to the content repository, and consuming that same version within the release pipeline, it has more or less have the same advantages as using a build artifact stored in a safe location.

Considerations for choosing the right artifact source

When you use a release pipeline to deploy your packages to production you need to have some traceability. That means you want to know where the package that you are deploying originates from. It is essential to understand that the sources that you built and checked into your version control are precisely the same as the sources that you are going to deploy to the various environments that are going to be used by your customers. Primarily when you work in a regulated environment like a bank or an insurance company, auditors ask you to provide traceability to sources that you deployed to prove the integrity of the package.

Another crucial aspect of your artifacts is auditability. You need to know who changed that line of code and who triggered the build that produces the artifact that is being deployed.

A useful mechanism to make sure you can provide the right traceability and auditability is using immutable packages. That is not something that you can buy, but something that you need to implement yourself. By using a build pipeline that produces a package which is stored in a location that cannot be accessed by humans, you ensure the sources are unchanged throughout the whole release process. This is an essential concept of release pipelines.

You identify an immutable package by giving it a version so that you can refer to it in a later stage. Versioning strategy is a complex concept and is not in the scope of this module, but by having a unique identification number or label attached to the package, and making sure that this number or label cannot be changed or modified afterwards, you ensure traceability and auditability from source code to production.

Read more about **Semantic Versioning⁵**.

⁵ <https://semver.org/>

Choosing the right artifact source is tightly related to the requirements you have regarding traceability and auditability. If you need an immutable package (containing multiple files) that can never be changed and be traced, a build artifact is the best choice. If it is one file, you can directly link to source control.

You can also point at a disk or network share, but this implies some risk concerning auditability and immutability. Can you ensure the package never changed?

See also [Release artifacts and artifact sources⁶](#).

Demonstration Selecting an Artifact Source

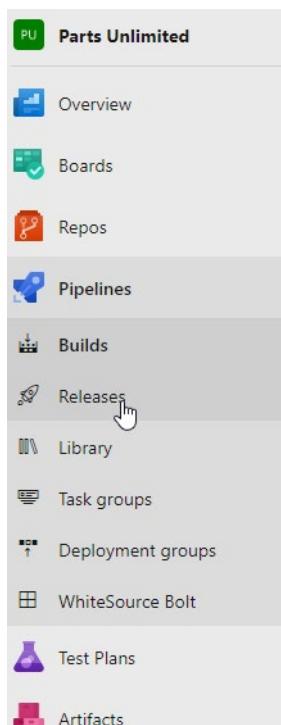
In this demonstration, you will investigate Artifact Sources.

Note: Before starting this demonstration, make sure you have performed the steps in the prerequisites section.

Steps

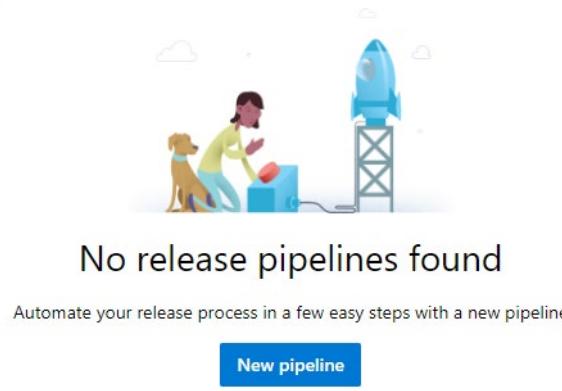
Let's take a look at how to work with one or more artifact sources in the release pipeline.

1. In the Azure DevOps environment, open the **Parts Unlimited** project, then from the main menu, click **Pipelines**, then click **Releases**.



2. In the main screen area, click **New pipeline**.

⁶ <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/artifacts?view=vsts>



3. In the **Select a template** pane, note the available templates, but then click the **Empty job** option at the top. This is because we are going to focus on selecting an artifact source.
4. In the **Artifacts** section, click **+Add an artifact**.
5. Note the available options in the **Add an artifact** pane, and click the option to see **more artifact types**, so that you can see all the available artifact types:

Add an artifact

Source type

✓ Build	Azure Repos ...	GitHub	TFVC
Azure Artifacts	GitHub Relea...	Azure Contai...	Docker Hub
Jenkins			

Show less ^

While we're in this section, let's briefly look at the available options.

6. Click **Build** and note the parameters required. This option is used to retrieve artifacts from an Azure DevOps Build pipeline. Using it requires a project name, and a build pipeline name. (Note that projects can have multiple build pipelines). This is the option that we will use shortly.

Project * ⓘ

Parts Unlimited

Source (build pipeline) * ⓘ

ⓘ This setting is required.

7. Click **Azure Repository** and note the parameters required. It requires a project name and also asks you to select the source repository.

Project * ⓘ

ⓘ This setting is required.

Source (repository) * ⓘ

ⓘ This setting is required.

8. Click **GitHub** and note the parameters required. The **Service** is a connection to the GitHub repository. It can be authorized by either OAuth or by using a GitHub personal access token. You also need to select the source repository.

Service * | Manage ⓘ

ⓘ This setting is required.

Source (repository) * ⓘ

ⓘ This setting is required.

9. Click **TFVC** and note the parameters required. It also requires a project name and also asks you to select the source repository.

Project * ⓘ

ⓘ This setting is required.

Source (repository) * ⓘ

ⓘ This setting is required.

Note: A release pipeline can have more than one set of artifacts as input. A common example is a situation where as well as your project source, you also need to consume a package from a feed.

10. Click **Azure Artifacts** and note the parameters required. It requires you to identify the feed, package type, and package.

Feed *

① This setting is required.

Package type

NuGet

Package * ⓘ

① This setting is required.

11. Click **GitHub Release** and note the parameters required. It requires a service connection and the source repository.

Service connection * | Manage ⓘ

① This setting is required.

Source (repository) * ⓘ

① This setting is required.

Note: we will discuss service connections later in the course.

12. Click **Azure Container Registry** and note the parameters required. Again, this requires a secure service connection, along with details of the Azure Resource Group that the container registry is located in. This allows you to provide all your Docker containers directly into your release pipeline.

Service connection * | Manage ⓘ

① This setting is required.

Resource Group * ⓘ

① This setting is required.

Azure Container Registry * ⓘ

① This setting is required.

Repository * ⓘ

① This setting is required.

13. Click **Docker Hub** and note the parameters required. This option would be useful if your containers are stored in Docker Hub rather than in an Azure Container Registry. After choosing a secure service connection, you need to select the namespace and the repository

Service connection * | Manage ↗

① This setting is required.

Namespaces * ⓘ

① This setting is required.

Repository * ⓘ

① This setting is required.

14. Last but not least, click **Jenkins** and note the parameters required. You do not need to get all your artifacts from Azure. You can retrieve them from a Jenkins build. So if you have a Jenkins Server in your infrastructure, you can use the build artifacts from there, directly in your Azure DevOps pipelines.

Service connection * | Manage ↗

① This setting is required.

Jenkins Job * ⓘ

① This setting is required.

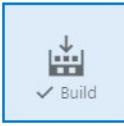
Configuring the Build Artifact

Let's return to adding our Build output as the artifact source.

15. Click the **Build** source type again. Note that the Project should show the current project. From the **Source (build pipeline)** drop down list, select **Parts Unlimited-ASP.NET-CI**. Take note of the default values for the other options, and then click **Add**.

Add an artifact

Source type

 Build  Azure Repos ...  GitHub  TFVC

[5 more artifact types ▾](#)

Project * [\(i\)](#)

Parts Unlimited

Source (build pipeline) * [\(i\)](#)

Parts Unlimited-ASP.NET-CI

Default version * [\(i\)](#)

Latest

Source alias * [\(i\)](#)

_Parts Unlimited-ASP.NET-CI

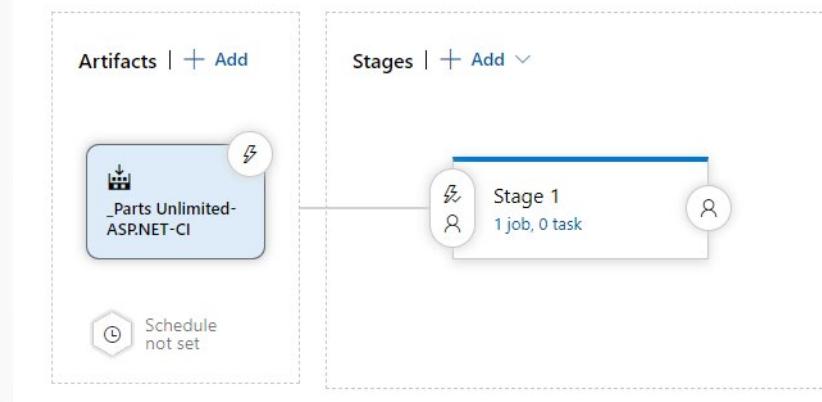
(i) The artifacts published by each version will be available for deployment in release pipelines. The latest successful build of Parts Unlimited-ASP.NET-CI published the following artifacts: drop.

Add

We have now added the artifacts that we will need for later walkthroughs.

All pipelines > New release pipeline

Pipeline Tasks ▾ Variables Retention Options History



16. To save the work, click **Save**, then in the Save dialog box, click **OK**.

Deployment Stages

A stage or deployment stage is a logical and independent entity that represents where you want to deploy a release generated from a release pipeline. Sometimes a stage is called an environment. For

example Test or Production. But it does not necessarily reflect the lifecycle of a product. It can represent any physical or real stage that you need. For example, the deployment in a stage may be to a collection of servers, a cloud, or multiple clouds. In fact, you can even use a stage to represent shipping the software to an app store, or the manufacturing process of a boxed product, or a way to group a cohort of users for a specific version of an application.

You must be able to deploy to a stage independently of other stages in the pipeline. There should be no dependency between stages in your pipeline. For example, your pipeline might consist of two stages A and B, and your pipeline could deploy Release 2 to A and Release 1 to B. If you make any assumptions in B about the existence of a certain release in A, the two stages are not independent.

Here are some suggestions and examples for stages:

- Dev, QA, Prod - As new builds are produced, they can be deployed to Dev. They can then be promoted to QA, and finally to Prod. At any time, each of these stages may have a different release (set of build artifacts) deployed to them. This is a good example of the use of stages in a release pipeline.
- Customer adoption rings (for example, early adopter ring, frequent adopter ring, late adopter ring)
 - You typically want to deploy new or beta releases to your early adopters more often than to other users. Therefore, you are likely to have different releases in each of these rings. This is a good example of the use of stages in a pipeline.
- Database and web tiers of an application - These should be modeled as a single stage because you want the two to be in sync. If you model these as separate stages, you risk deploying one build to the database stage and a different build to the web tier stage.
- Staging and production slots of a web site - There is clearly an interdependence between these two slots. You do not want the production slot to be deployed independently of the build version currently deployed to the staging slot. Therefore, you must model the deployment to both the staging and production slots as a single stage.
- Multiple geographic sites with the same application - In this example, you want to deploy your website to many geographically distributed sites around the globe and you want all of them to be the same version. You want to deploy the new version of your application to a staging slot in all the sites, test it, and - if all of them pass - swap all the staging slots to production slots. In this case, given the interdependence between the sites, you cannot model each site as a different stage. Instead, you must model this as a single stage with parallel deployment to multiple sites (typically by using jobs).
- Multiple test stages to test the same application - Having one or more release pipelines, each with multiple stages intended to run test automation for a build, is a common practice. This is fine if each of the stages deploys the build independently, and then runs tests. However, if you set up the first stage to deploy the build, and subsequent stages to test the same shared deployment, you risk overriding the shared stage with a newer build while testing of the previous builds is still in progress.

Considerations for setting up Deployment Stages

Now that we know what stage is and have some idea of what is commonly used as a stage, we need to review a few considerations around choosing the right stages. In most cases, organizations prefer the traditional approach when it comes to setting up the staging strategy. With the traditional method, you can think of setting up an environment for *Dev*, for *Test*, and *Production*. On many occasions, there is even an additional stage, *Staging* or *Acceptance*.

Even when companies use a cloud strategy, they still apply the general sense of environments. When applications are deployed to a cloud instead of a server in a data centre, you have the possibility to

rethink your strategy around stages. For example, a stage is not necessarily a long-lived entity. When we talk about Continuous Delivery, where we might deploy our application multiple times a day, we may assume that the application is also tested every time the application is deployed. The question that we need to ask ourselves is, do we want to test in an environment that is already in use, or do we want a testing environment that is clean from the start?

On many occasions both scenarios are valid. Sometimes you want to start from scratch, and sometimes you want to know what happens if you refresh the environment. In a DevOps world, we see infrastructure as just another piece of software (Infrastructure as Code). Using Cloud technology combined with Infrastructure as Code gives us new possibilities when it comes to environments. We are not limited to a fixed number of environments anymore. Instead, we can spin up environments on demand. When we want to test something, we spin up a new environment, deploy our code and run our tests. When we are done, we can clean up the environment. Traditional labels for environments, therefore, do not apply anymore. Let's take Test as an example. Maybe we have different test environments, one for load testing, one for integration testing, one for system testing and one for functional testing. The sky is the limit!

Depending on the needs of the organization and the DevOps teams, the number of stages and the purpose of stages vary. Some organizations stick to the DTAP (Dev, Test, Acceptance, Production) where others deploy directly to production with temporary stages in between.

Important things to consider are there for

- Is your stage long lived or short lived?
- What is the purpose of this specific stage?
- Who is going to use it?
- Is your target application overwriting an existing one would always be a fresh install?
- Do you need a new stage for bug fixes?
- Do you need an isolated environment with encrypted data or disconnected from a network?
- Can you afford downtime?
- Who is the owner of the stage? Who can apply changes?

These and maybe other considerations need to play a crucial role in defining the number of stages and the purpose of stages.

Everything you need to know about Deployment stages in combination with Azure DevOps you can find on the [Microsoft Docs⁷](#)

Discussion

What deployment stages would you define for your organization?

Discuss what deployment stages you recognize in your organization.

Consider the following things:

- Is your stage long lived or short lived?
- What is the purpose of this specific stage?
- Who is going to use it?

⁷ <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/environments?view=vsts>

- Is your target application overwriting an existing one would always be a fresh install?
- Do you need a new stage for bug fixes?
- Do you need an isolated environment with encrypted data or disconnected from a network?
- Can you afford downtime?
- Who is the owner of the stage? Who can apply changes?

Demonstration Setting Up Stages

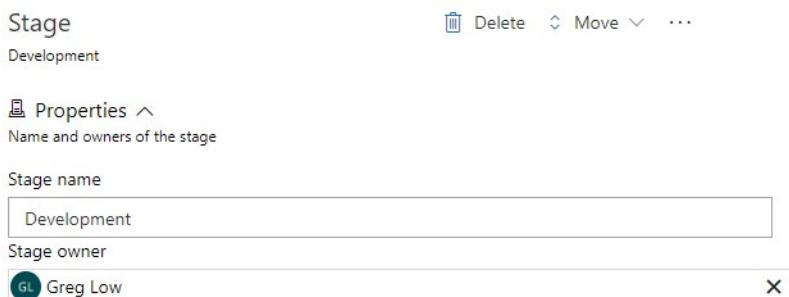
In this demonstration, you will investigate Stages.

Note: Before starting this walkthrough, make sure you have performed the steps in the prerequisites section and the previous walkthrough.

Steps

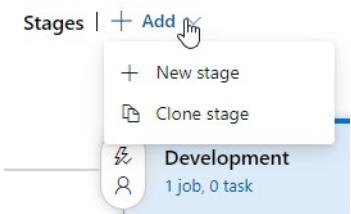
Let's now take a look at the other section in the release pipeline that we have created: Stages.

1. Click on **Stage 1** and in the Stage properties pane, set **Stage name** to **Development** and close the pane.



Note: stages can be based on templates. For example, you might be deploying a web application using node.js or Python. For this walkthrough, that won't matter because we are just focussing on defining a strategy.

2. To add a second stage, click **+Add** in the Stages section and note the available options. You have a choice to create a new stage, or to clone an existing stage. Cloning a stage can be very helpful in minimizing the number of parameters that need to be configured. But for now, just click **New stage**.



3. When the **Select a template** pane appears, scroll down to see the available templates. For now, we don't need any of these, so just click **Empty job** at the top, then in the Stage properties pane, set **Stage name** to **Test**, then close the pane.

Select a template Or start with an [Empty job](#)

Featured

Azure App Service deployment

Deploy your application to Azure App Service. Choose from Web App on Windows, Linux, containers, Function Apps, or WebJobs.

Deploy a Java app to Azure App Service

Deploy a Java application to an Azure Web App.

Deploy a Node.js app to Azure App Service

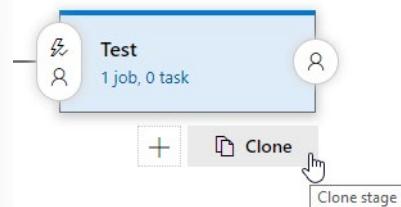
Deploy a Node.js application to an Azure Web App.

Deploy a PHP app to Azure App Service and Azure Database for MySQL

Deploy a PHP application to an Azure Web App and database to Azure Database for MySQL.

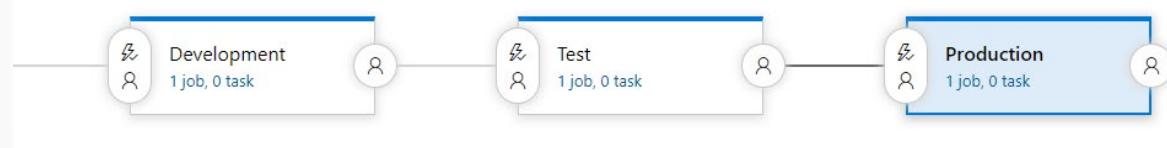
Deploy a Python app to Azure App Service and

4. Hover over the **Test** stage and notice that two icons appear below. These are the same options that were available in the menu drop down that we used before. Click the **Clone** icon to clone the stage to a new stage.



5. Click on the **Copy of Test** stage and in the stage properties pane, set **Stage name** to **Production** and close the pane.

Stages | [+ Add](#) ▾



We have now defined a very traditional deployment strategy. Each of the stages contains a set of tasks, and we will look at those tasks later in the course.

*Note: The same artifact sources move through each of the stages.

The lightning bolt icon on each stage shows that we can set a trigger as a predeployment condition. The person icon on both ends of a stage, show that we can have pre and post deployment approvers.

Concurrent stages

You'll notice that at present we have all the stages one after each other in a sequence. It is also possible to have concurrent stages. Let's see an example.

6. Click the **Test** stage, and on the stage properties pane, set **Stage name** to **Test Team A** and close the pane.
7. Hover over the **Test Team A** stage, and click the **Clone** icon that appears, to create a new cloned stage.
8. Click the **Copy of Test Team A** stage, and on the stage properties pane, set **Stage name** to **Test Team B** and close the pane.
9. Click the **Pre-deployment conditions** icon (i.e. the lightning bolt) on **Test Team B** to open the pre-deployment settings.



10. In the Pre-deployment conditions pane, note that the stage can be triggered in three different ways:

Pre-deployment conditions
Test Team B

Triggers ^
Define the trigger that will start deployment to this stage

Select trigger ⓘ

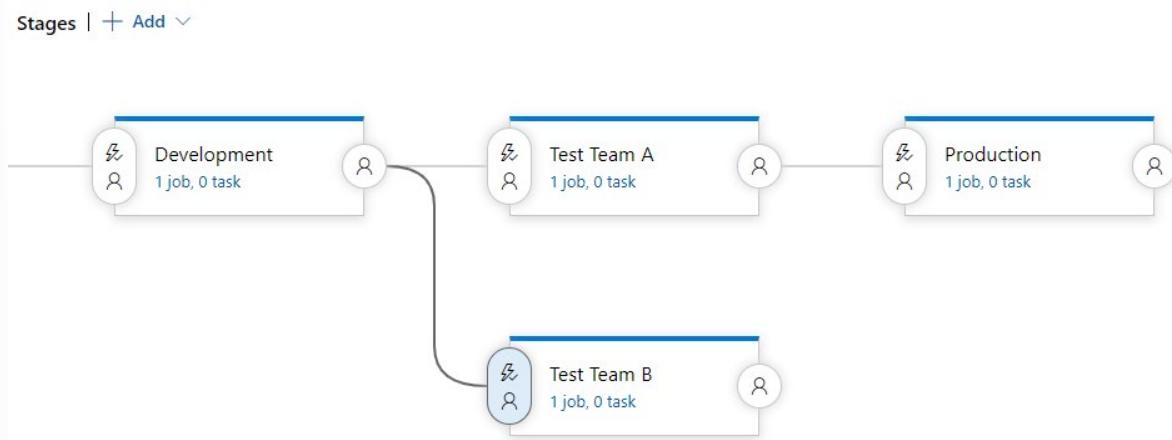
- After release
- After stage
- Manual only

Stages ⓘ

✓ Test Team A

The stage can immediately follow Release. (That is how the Development stage is currently configured). It can require manual triggering. Or, more commonly, it can follow another stage. At present, it is following **Test Team A** but that's not what we want.

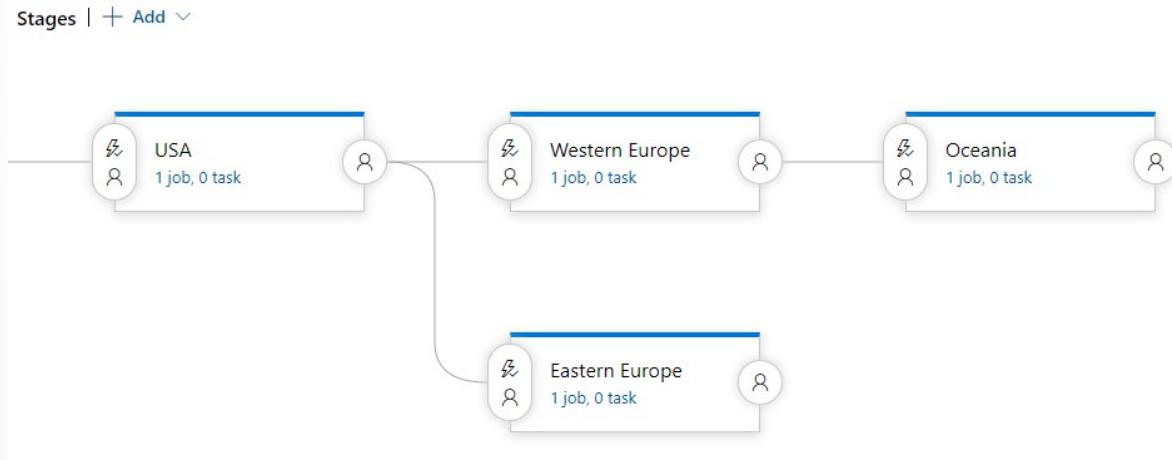
11. From the **Stages** drop down list, chose **Development** and uncheck **Test Team A**, then close the pane. We now have two concurrent Test stages.



Stage vs Environment

You may have wondered why these items are called **Stages** and not **Environments**.

In the current configuration, we are in fact using them for different environments. But this is not always the case. Here is a deployment strategy based upon regions instead:



Azure DevOps pipelines are very configurable and support a wide variety of deployment strategies. The name **Stages** is a better fit than **Environment** even though the stages can be used for environments.

For now, let's give the pipeline a better name and save the work.

12. At the top of the screen, hover over the **New release pipeline** name and when a pencil appears, click it to edit the name. Type **Release to all environments** as the name and hit enter or click elsewhere on the screen.



13. For now, save the environment based release pipeline that you have created by clicking **Save**, then in the Save dialog box, click **OK**.

Delivery and Deployment Cadence, Schedules and Triggers

When you have a clear view of your different stages, you need to think about when you want to deploy to these stages. Then there is a difference between a release and a deployment, as we discussed earlier. The moment you create the release (the package that holds a versioned set of artifacts), can differ from the moment you deploy to an environment.

But both the release and stages make use of triggers. There are three types of triggers we recognize.

Continuous deployment Trigger

You can set up this trigger on your release pipeline. Once you do that, every time a build completes, your release pipeline will trigger, and a new release will be created.

Scheduled Triggers

This speaks for itself, but what it allows you to, is to set up time-based manner to start a new release. For example every night at 3:00 AM or at 12:00 PM. You can have one or multiple schedules per day, but it will always run on this specific time.

Manual trigger

With a manual trigger, a person or system triggers the release based on a specific event. When it is a person, it probably uses some UI to start a new release. When it is an automated process most likely, some event will occur, and by using the automation engine, which is usually part of the release management tool, you can trigger the release from another system.

As we mentioned in the introduction, Continuous Delivery is not only about deploying multiple times a day, it is about being able to deploy on demand. When we define our cadence, questions that we should ask ourselves are:

- Do we want to deploy our application?
- Do we want to deploy multiple times a day
- Can we deploy to a stage? Is it used?

For example, when a tester is testing an application during the day might not want to deploy a new version of the app during the test phase.

Another example, when your application incurs downtime, you do not want to deploy when users are using the application.

The frequency of deployment, or cadence, differs from stage to stage. A typical scenario that we often see is that continuous deployment happens to the development stage. Every new change ends up there once it is completed and builds. Deploying to the next phase does not always occur multiple times a day but only during the night.

When you are designing your release strategy, choose your triggers carefully and think about the required release cadence.

Some things we need to take into consideration are

- What is your target environment?
 - Is it used by one team or is it used by multiple teams?
-
- If a single team uses it, you can deploy frequently. Otherwise, you need to be a bit more careful.
 - Who are the users? Do they want a new version multiple times a day?
 - How long does it take to deploy?
 - Is there downtime? What happens to performance? Are users impacted?

Some tools make a difference between a release and deployment. This is what we talked about in the introduction. You have to realise that a Trigger for the release pipeline only creates a new release. In most cases, you also need to set up triggers for the various stages as well to start deployments. For example, you can set up an automatic deployment to the first stage after the creation of a release. And, after that, when the deployment to the first stage is successful, start the deployment to the next stage(s).

For more information, see also:

- [Release triggers⁸](#)
- [Stage Triggers⁹](#)

Demonstration Selecting Delivery and Deployment Cadence

In this demonstration, you will investigate Delivery Cadence.

- ✓ Note: Before starting this walkthrough, make sure you have performed the steps in the prerequisites section and the previous walkthroughs.

Steps

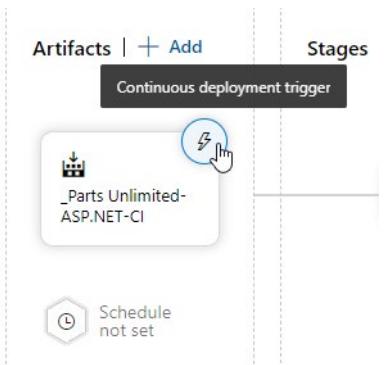
Let's now take a look at when our release pipeline is used to create deployments. Mostly, this will involve the use of triggers.

When we refer to a deployment, we are referring to each individual stage, and each stage can have its own set of triggers that determine when the deployment occurs.

1. Click the lightning bolt on the **_Parts Unlimited-ASP.NET-CI** artifact.

⁸ <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/triggers?view=vsts>

⁹ <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/triggers?view=vsts#env-triggers>



2. In the Continuous deployment trigger pane, click the **Disabled** option to enable continuous deployment. It will then say **Enabled**.

Continuous deployment trigger

Build: _Parts Unlimited-ASP.NET-CI



Enabled

Creates a release every time a new build is available.

Build branch filters (i)

No filters added.



Pull request trigger

Build: _Parts Unlimited-ASP.NET-CI



Disabled

(i) Enabling this will create a release every time a selected artifact is available as part of a pull request workflow

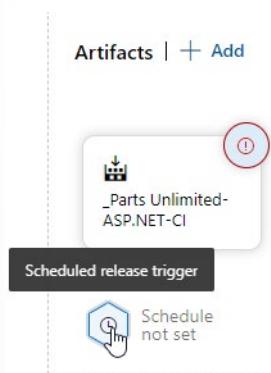
Once this is selected, every time that a build completes, a deployment of the release pipeline will start.

- ✓ Note: You can filter which branches affect this, so for example you could choose the master branch or a particular feature branch.

Scheduled Deployments

You might not want to have a deployment commence every time a build completes. That might be very disruptive to testers downstream if it was happening too often. Instead, it might make sense to set up a deployment schedule.

3. Click on the **Scheduled release trigger** icon to open its settings.



4. In the Scheduled release trigger pane, click the **Disabled** option to enable scheduled release. It will then say **Enabled** and additional options will appear.

Scheduled release trigger

Define schedules to trigger releases

Enabled

Create a new release at the specified times

⌚ Mon through Fri at 3:00 ⌈

Mon Tue Wed Thu Fri Sat Sun

03h ⌄ 00m ⌄ (UTC) Coordinated Universal Time ⌄

Only schedule releases if the source or pipeline has changed

+ Add a new time

You can see in the screenshot above that a deployment using the release pipeline would now occur each weekday at 3AM. This might be convenient when you for example, share a stage with testers who work during the day. You don't want to constantly deploy new versions to that stage while they're working. This setting would create a clean fresh environment for them at 3AM each weekday.

- ✓ Note: The default timezone is UTC. You can change this to suit your local timezone as this might be easier to work with when creating schedules.

5. For now, we don't need a scheduled deployment, so click the **Enabled** button again to disable the scheduled release trigger and close the pane.

Pre-deployment Triggers

6. Click the lightning bolt on the **Development** stage to open the pre-deployment conditions.

Pre-deployment conditions

Development

Triggers 

Define the trigger that will start deployment to this stage

Select trigger 

 After release  After stage  Manual only

Artifact filters 

Disabled

Schedule 

Disabled

Pull request deployment 

Disabled

Pre-deployment approvals 

Select the users who can approve or reject deployments to this stage

Disabled

Gates 

Define gates to evaluate before the deployment. [Learn more](#)

Disabled

Deployment queue settings 

Define behavior when multiple releases are queued for deployment

✓ Note: Both artifact filters and a schedule can be set at the pre-deployment for each stage rather than just at the artifact configuration level.

Deployment to any stage doesn't happen automatically unless you have chosen to allow that.

Considerations for Release Approvals

In this part, we will talk about release approvals and release gates.

First, we take a look at manual approvals. This does not always resonate well when we talk about Continuous Delivery, because Continuous Delivery implies that you deliver multiple times a day.

As we have described in the introduction, Continuous Delivery is all about delivering on demand. But, as we discussed in the differences between release and deployment, delivery, or deployment, is only the technical part of the Continuous Delivery process. It is all about how you are technically able to install the software on an environment, but it does not say anything about the process that needs to be in place for a release.

Release approvals are not to control *how*, but control *if* you want to deliver multiple times a day.

Manual approvals also suit a significant need. Organizations that start with Continuous Delivery often lack a certain amount of trust. They do not dare to release without a manual approval. After a while, when they find that the approval does not add any value and the release always succeeds, the manual approval is often replaced by an automatic check.

Things to consider when you are setting up a release approval are:

- What do we want to achieve with the approval?
Is it an approval that we need for compliance reasons? For example. We need to adhere to the four-eyes principal to get out SOX compliance. Or, Is it an approval that we need to manage our dependencies. Or, is it an approval that needs to be in place purely because we need a sign off from an authority like Security Officers or Product Owners.
- Who needs to approve?
We need to know who needs to approve the release. Is it a product owner, Security officer, or just someone that is not the one that wrote the code. This is important because the approver is part of the process. He is the one that can delay the process if not available. So be aware that.
- When do you want to approve?
Another essential thing to consider is when to approve. This is a direct relationship with what happens after approval. Can you continue without approval? Or, is everything on hold until approval is given. By using scheduled deployments, you can separate approval from deployment.

Although manual approval is a great mechanism to control the release, it is not always useful. On many occasions, the check can be done in an earlier stage. For example, approving a change that has been made in Source Control.

Scheduled deployments already solve the dependency issue. You do not have to wait for a person in the middle of the night. But there is still a manual action involved. If you want to eliminate manual activities altogether, but still want to have control you start talking about automatic approvals or release gates.

Example:

In many organizations, there are so-called dependency meetings. This is a planning session where the release schedule of dependent components is discussed. Think of downtime of a database server or an update of an API. This takes a lot of time and effort, and the only thing that is needed is a signal if the release can proceed. Instead of having this meeting you can also create a mechanism where people press a button on a form when the release cannot advance. When the release starts, it checks the state of the gate by calling an API. If the "gate" is open, we can continue. Otherwise, we stop the release.

By using scripts and API's, you can create your own release gates instead of a manual approval. Or at least extending your manual approval. Other scenarios for automatic approvals are for example.

- Incident and issues management. Ensure the required status for work items, incidents, and issues. For example, ensure that deployment only occurs if no bugs exist.
- Notify users such as legal approval departments, auditors, or IT managers about a deployment by integrating with approval collaboration systems such as Microsoft Teams or Slack, and waiting for the approval to complete.
- Quality validation. Query metrics from tests on the build artifacts such as pass rate or code coverage and only deploy if they are within required thresholds.
- Security scan on artifacts. Ensure security scans such as anti-virus checking, code signing, and policy checking for build artifacts have completed. A gate might initiate the scan and wait for it to complete, or check for completion.
- User experience relative to baseline. Using product telemetry, ensure the user experience hasn't regressed from the baseline state. The experience level before the deployment could be considered a baseline.
- Change management. Wait for change management procedures in a system such as ServiceNow complete before the deployment occurs.

- Infrastructure health. Execute monitoring and validate the infrastructure against compliance rules after deployment, or wait for proper resource utilisation and a positive security report.

In short, approvals and gates give you additional control over the start and completion of the deployment pipeline. They can usually be set up as a pre-deployment and post-deployment condition, that can include waiting for users to approve or reject deployments manually, and checking with other automated systems until specific requirements are verified. In addition, you can configure a manual intervention to pause the deployment pipeline and prompt users to carry out manual tasks, then resume or reject the deployment.

To find out more about Release Approvals and Gates, check these documents.

- **Release approvals and gates overview**¹⁰
- **Release Approvals**¹¹
- **Release Gates**¹²

Demonstration Setting Up Manual Approvals

In this demonstration, you will investigate Manual Approval.

Note: Before starting this demonstration, make sure you have performed the steps in the prerequisites section and the previous walkthroughs.

Steps

Let's now take a look at when our release pipeline needs manual approval before deployment of a stage starts, or manual approval that the deployment of a stage completed as expected.

While DevOps is all about automation, manual approvals are still very useful. There are many scenarios where they are needed. For example, a product owner might want to sign off a release before it moves to production. Or the scrum team wants to make sure that no new software is deployed to the test environment before someone signs off on it, because they might need to find an appropriate time if it's constantly in use.

This can help to gain trust in the DevOps processes within the business.

Even if the process will later be automated, people might still want to have a level of manual control until they become comfortable with the processes. Explicit manual approvals can be a great way to achieve that.

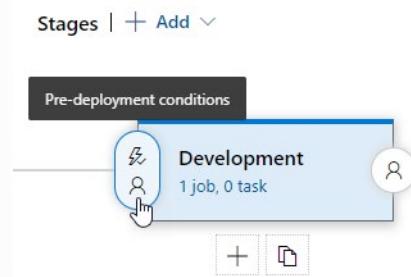
Let's try one.

1. Click the pre-deployment conditions icon for the **Development** stage to open the settings.

¹⁰ <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/approvals/approvals?view=vsts>

¹¹ <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/approvals/approvals?view=vsts>

¹² <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/approvals/gates?view=vsts>



2. Click the **Disabled** button in the Pre-deployment approvals section to enable it.

Pre-deployment approvals Enabled

Select the users who can approve or reject deployments to this stage

Approvers [\(i\)](#)

Search users and groups for approvers

Enter at least one approver.

Timeout [\(i\)](#)

30 Days

Approval policies

The user requesting a release or deployment should not approve it

Skip approval if the same approver approved the previous stage [\(i\)](#)

3. In the **Approvers** list, find your own name and select it. Then set the **Timeout** to **1 Days**.

Pre-deployment approvals Enabled

Select the users who can approve or reject deployments to this stage

Approvers [\(i\)](#)

Greg Low Search users and groups for approvers

Timeout [\(i\)](#)

1 Days

Note: Approvers is a list, not just a single value. If you add more than one person in the list, you can also choose if they need to approve in sequence, or if either or both approvals are needed.

4. Take note of the approver policy options that are available:

Approval policies

The user requesting a release or deployment should not approve it

Skip approval if the same approver approved the previous stage [\(i\)](#)

It is very common to not allow a user who requests a release or deployment to also approve it. In this case, we are the only approver so we will leave that unchecked.

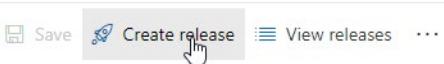
5. Close the Pre-deployment conditions pane and notice that a checkmark has appeared beside the person in the icon.



Test the Approval

Now it's time to see what happens when an approval is required.

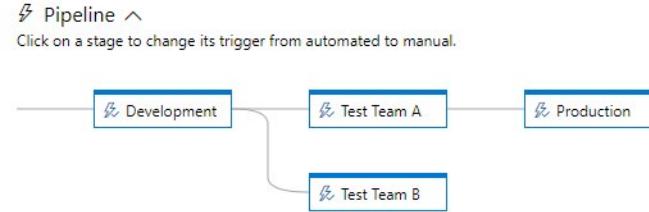
6. Click **Save** to save the work, and **OK** on the popup window.
7. Click **Create release** to start the release process.



8. In the **Create a new release** pane, note the available options, then click **Create**.

Create a new release X

Release to all environments



Click on a stage to change its trigger from automated to manual.

Stages for a trigger change from automated to manual. (i)



Artifacts (i)

Select the version for the artifact sources for this release

Source alias	Version
_Parts Unlimited-ASP.NET-CI	20190901.2

Release description



Create **Cancel**

9. In the upper left of the screen, you can see that a release has been created.

All pipelines >  Release to all environments



Pipeline Tasks Variables Retention Options History

10. At this point, an email should have been received, indicating that an approval is required.

The screenshot shows the Azure DevOps Release interface. At the top, there's a Microsoft logo and the text "Release to all environments > Release-1". Below that, a large bold title says "Deployment to Development is waiting for your approval". Underneath, it shows the pipeline "Parts Unlimited-ASP.NET-CI / 20190901.2" and the branch "master". It indicates that the release was "Requested for Greg Low". A blue button labeled "View approval" is visible. The main area is a "Summary" card with the following details:

Release description	("None")
Deployment trigger	automated: after release creation
Requested for	Greg Low
Attempt	1

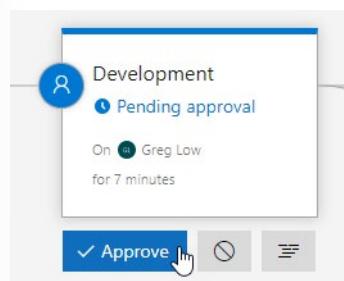
At this point, you could just click the link in the email, but instead, we'll navigate within Azure DevOps to see what's needed.

11. Click on the **Release 1 Created** link (or whatever number it is for you) in the area we looked at in Step 9 above. We are then taken to a screen that shows the status of the release.

The screenshot shows the Azure DevOps Release Pipeline interface. At the top, there are tabs for "Pipeline", "Variables", "History", and buttons for "Deploy", "Cancel", and "Approve multiple". The "Pipeline" tab is selected. On the left, under "Release", it says "Manually triggered by Greg Low 01/09/2019, 16:40". Under "Artifacts", it lists "Parts Unlimited-ASP.N... 20190901.2" and "master". On the right, under "Stages", there's a "Development" stage with a blue progress bar. A tooltip over the bar says "Pending approval" and "On 1 Greg Low for 4 minutes".

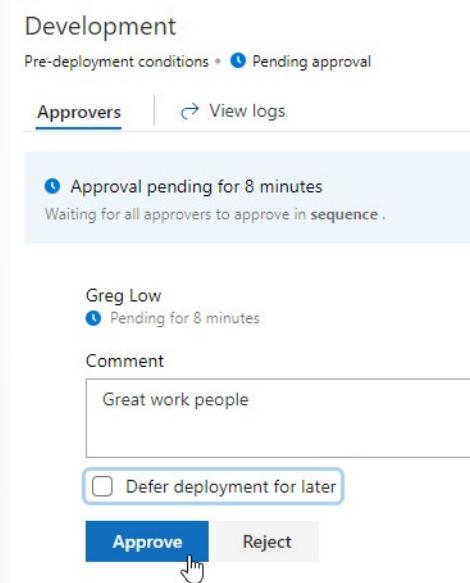
You can see that a release has been manually triggered and that the Development stage is waiting for an approval. As an approver, you can now perform that approval.

12. Hover over the **Development** stage and click the **Approve** icon that appears.

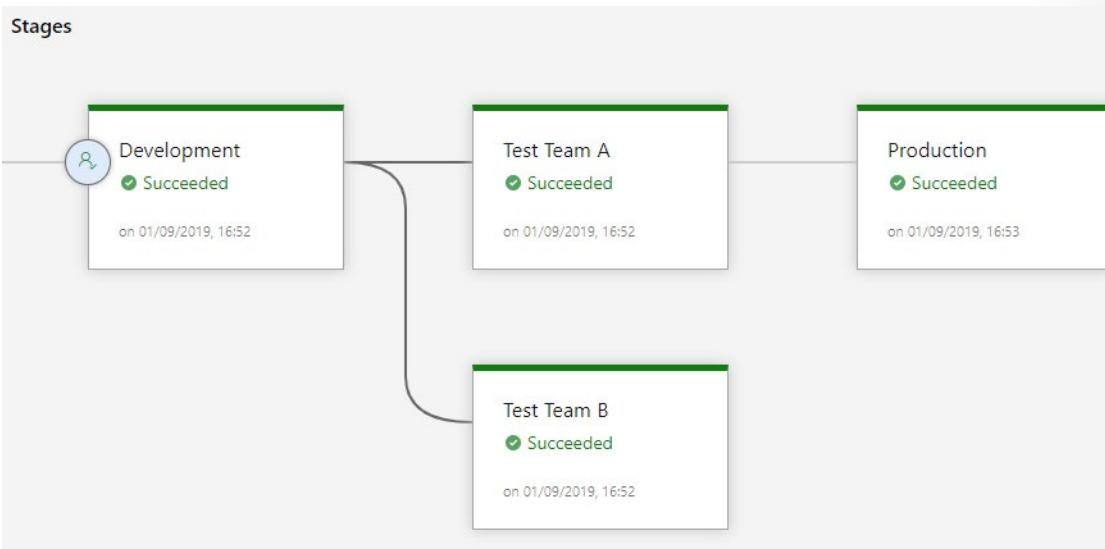


Note: Options to cancel the deployment or to view the logs are also provided at this point

13. In the Development approvals window, add a comment and click **Approve**.



The deployment stage will then continue. Watch as each stage proceeds and succeeds.



Demonstration Setting Up Release Gates

In this demonstration walkthrough, you will investigate Release Gates.

Note: Before starting this walkthrough, make sure you have performed the steps in the prerequisites section and the previous walkthroughs.

Steps

Let's now take a look at when our release pipeline needs to perform automated checks for issues like code quality, before continuing with the deployments. That automated approval phase is achieved by using Release Gates.

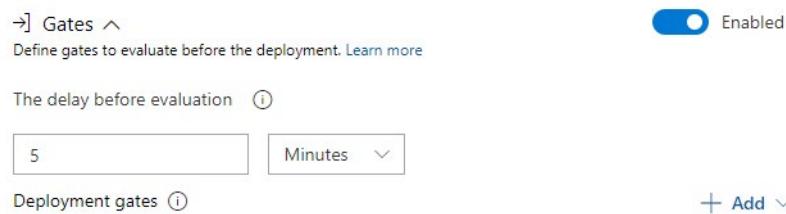
Let's take a look at configuring a release gate.

1. Click the lightning icon on the **Development** stage to open the pre-deployment conditions settings.

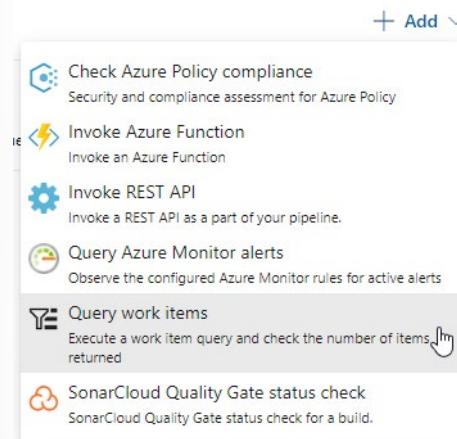
Stages | + Add ▾



2. In the Pre-deployment conditions pane, click the **Disabled** button beside **Gates** to enable them.



3. Click **+Add** to see the available types of gates, then click **Query work items**.



We will use the **Query work items** gate to check if there are any outstanding bugs that need to be dealt with. It does this by running a work item query. This is an example of what is commonly called a **Quality Gate**.

4. Set **Display name** to **No critical bugs allowed**, and from the **Query** drop down list, choose **Critical Bugs**. Leave the **Upper threshold** set to zero because we don't want to allow any bugs at all.

Deployment gates ⓘ

No critical bugs allowed

Enabled

Query work items ⓘ

Task version 0.*

Display name * No critical bugs allowed

Query * ⓘ Critical Bugs

Upper threshold * ⓘ 0

Advanced

Output Variables

- Click the drop down beside **Evaluation options** to see what can be configured. While 15 minutes is a reasonable value in production, for our testing, change **The time between re-evaluation of gates** to **5 Minutes**.

Evaluation options ▲

The time between re-evaluation of gates ⓘ

5 Minutes

Minimum duration for steady results after a successful gates evaluation ⓘ

0 Minutes

The timeout after which gates fail ⓘ

1 Days

Gates and approvals ⓘ

Before gates, ask for approvals

On successful gates, ask for approvals

Ignore gates outcome and ask for approvals

The release gate doesn't just fail or pass a single time. It can keep evaluating the status of the gate. It might fail the first time, but after re-evaluation, it might then pass if the underlying issue has been corrected.

- Close the pane and click **Save** and **OK** to save the work.
- Click **Create release** to start a new release, and in the **Create a new release** pane, click **Create**.

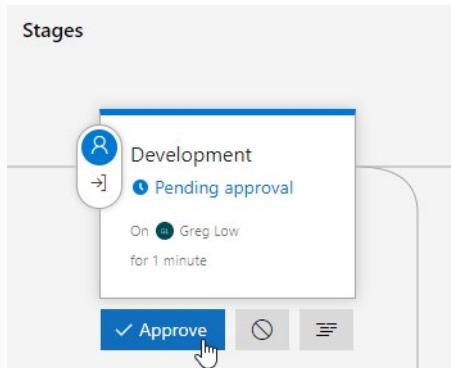


- Click on the release number to see how the release is proceeding.

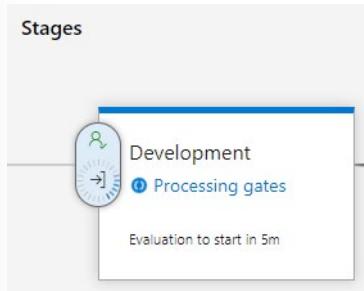
All pipelines >  Release to all environments



9. If it is waiting for approval, click **Approve** to allow it to continue, and in the **Development** pane, click **Approve**.



After a short while, you should see the release continuing and then entering the phase where it will process the gates.



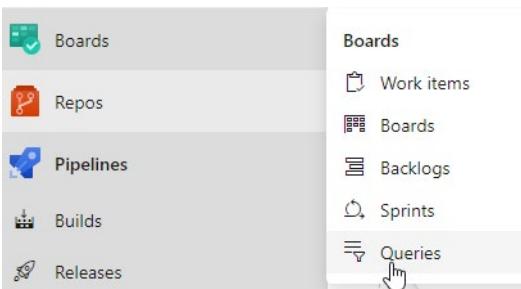
10. In the **Development** pane, click **Gates** to see the status of the release gates.

A screenshot of the Azure DevOps Development pane. The "Gates" tab is selected. It shows a message "Delay before evaluation is in progress" and "Gates evaluation will begin in 5m". Below this, there's a section for "Deployment gates \ samples" with a timestamp of "09:34" and a note "Next in 15m". At the bottom, there's a "No critical bugs allowed" gate with a red "X" icon and a note "Execute a work item query and check the num...".

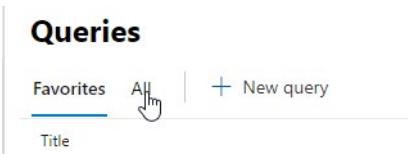
You will notice that the gate failed the first time it was checked. In fact, it will be stuck in the processing gates stage, as there is a critical bug. Let's look at that bug and resolve it.

11. Close the pane and click **Save** then **OK** to save the work.

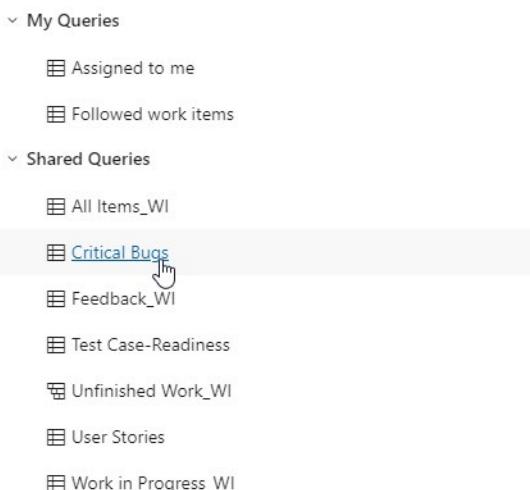
12. In the main menu, click **Boards** then click **Queries**.



13. In the **Queries** window, click **All** to see all the available queries.



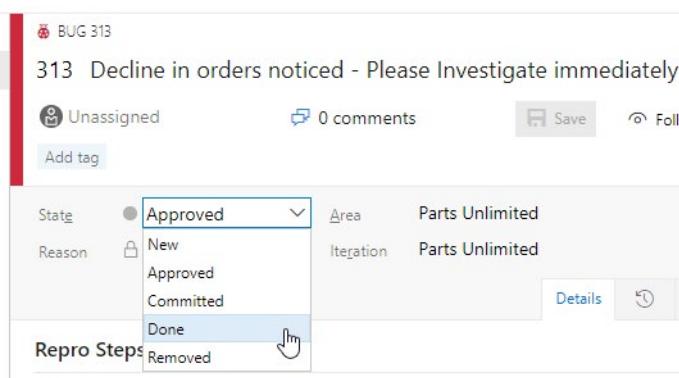
14. From the list of queries, click **Critical bugs**.



You will see that there is one critical bug that needs to be resolved.

ID	Work Item...	Title	Assigned To	State	Tags
313	Bug	Decline in orders noticed - Please Investigate immediately	...	Approved	

15. In the properties pane for the bug, change the **State** to **Done**, then click **Save**.



16. Click **Run query** to re-execute the work item query.

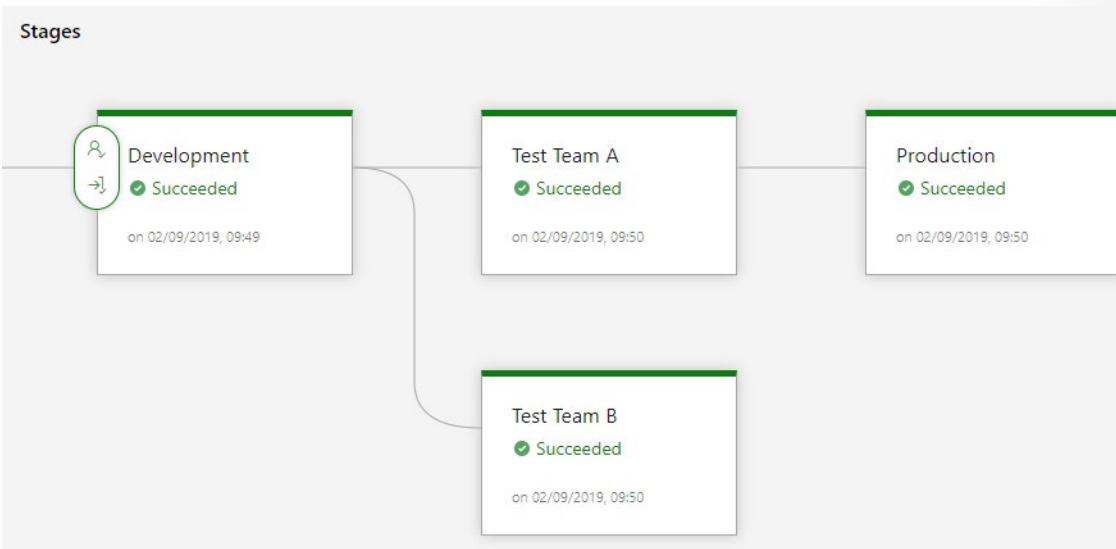
A screenshot of the 'Critical Bugs' shared query results page. The top navigation bar includes 'Queries > Shared Queries' and a search bar with '(R) Critical Bugs'. Below the navigation, there are tabs for 'Results' (which is selected), 'Editor', and 'Charts'. A 'Run query' button is highlighted with a mouse cursor. Other buttons include '+ New', 'Save query', and 'R'. The results table has columns for 'ID', 'Work Item...', and 'Title'. One row is shown: '313 Bug Decline in orders noticed - Please Investigate immediately'.

Note that there are now no critical bugs that will stop the release.

17. Return to the release by clicking **Pipelines** then **Releases** in the main menu, then clicking the name of the latest release.

A screenshot of the 'Releases' page in Azure DevOps. The top navigation bar includes 'Releases', 'Deployments', and 'Analytics'. Below the navigation, there are tabs for 'Releases' (selected), 'Deployments', and 'Analytics'. The main area shows two releases: 'Release-2' (status: 'GL', created: '20190512', branch: 'master') and 'Release-1' (status: 'GL', created: '20190901', branch: 'master').

18. When the release gate is checked next time, the release should continue and complete successfully.



Gates are a very powerful automated approval mechanism.

Clean up

To avoid excessive wait time in later walkthroughs, we'll disable the release gates.

19. In the main menu, click **Pipelines**, then click **Releases**, then click **Edit** to open the release pipeline editor.
20. Click the **Pre-deployment conditions** icon (i.e. the lightning bolt) on the **Development** task, and in the **Pre-deployment conditions** pane, click the switch beside **Gates** to disable release gates.
21. Click **Save**, then click **OK**.

Building a High-Quality Release pipeline

Release and release process

Before we dive into high-quality release pipelines, we need to consider the difference between a release and a release process. Or, when you talk about tooling, a release pipeline.

We start with defining a release process or release pipeline. The release pipeline contains all the steps that you walk through when you move your artifact, that comes from one of the artifact sources that we discussed earlier, through the stages or environments.

The stage can be a development stage, a test stage or a production stage or just a stage where a specific user can access the application. We discussed stages earlier in this module. Also part of your pipeline are the people that approve the release or the deployment to a specific stage, the triggers or the schedule on which the releases executes and the release gates, the automatic approvals of the process.

The release itself is something different. The release is an instance of the release pipeline. You can compare it with object inheritance. In Object Orientation, a class contains the blueprint or definition of an object. But the object itself is an instance of that blueprint.



How to measure quality of your release process

How do you measure the quality of your release process? The quality of your release process cannot be measured directly because it is a process. What you can measure is how good your process works. If your release process constantly changes, this might be an indication that there is something wrong in the process. If your releases constantly fail, and you constantly have to update your release process to make it work, might also be an indication that something is wrong with your release process.

Maybe something is wrong with the schedule on which your release runs, and you notice that your release always fails at a particular day or at a certain time. Or your release always fails after the deployment to another environment. This might be an indication that some things are maybe dependent or related.

What you can do to keep track of your release process quality, is creating visualisations about the quality of all the releases following that same release process or release pipeline. For example, adding a dashboard widget which shows you the status of every release.

Release Branch Runs - Default

Environments

	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	►
Sps.SelfTest	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	►
Sps.SelfHost	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	►
Tfs.SelfHost Set 1	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	►
Tfs.SelfHost Set 2	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✗ 98.69%	✓ 100%	✓ 100%	►
Tfs.SelfTest	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	►
Tfs.Deploy		✓ 100%	✓ 100%	✓ 100%		✓ 100%	✓ 100%	►
TfsOnPrem.SelfHost	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	►
TfsOnPrem.SelfTest	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	►

The release also has a quality aspect, but this is tightly related to the quality of the actual deployment and the package that has been deployed.

When we want to measure the quality of a release itself, we can perform all kinds of checks within the pipeline. Of course, you can execute all different types of tests like integration tests, load tests or even UI tests while running your pipeline and check the quality of the release that you are deploying.

Using a quality gate is also a perfect way to check the quality of your release. There are many different quality gates. For example, a gate that monitors to check if everything is healthy on your deployment targets, work item gates that verify the quality of your requirements process. You can add additional security and compliance checks. For example, do we comply with the 4-eyes principle, or do we have the proper traceability?

Using release gates as quality gate

A quality gate is the best way to enforce a quality policy in your organization. It is there to answer one question: can I deliver my application to production or not?

A quality gate is located before a stage that is dependent on the outcome of a previous stage. In the past, a quality gate was typically something that was monitored by a QA department. They had a number of documents or guidelines, and they verified if the software was of a good enough quality to move on to the next stage. When we think about Continuous Delivery, all manual processes are a potential bottleneck.

We need to reconsider the notion of quality gates and see how we can automate these checks as part of our release pipeline. By using automatic approval using a release gate, you can automate the approval and validate your company's policy before moving on.

Many quality gates can be considered.

- No new blocker issues
- Code coverage on new code greater than 80%
- No license violations
- No vulnerabilities in dependencies
- No new technical debt introduced

- Compliance checks
 - Are there work items linked to the release?
 - Is the release started by someone else as the code committer?
- Is the performance not affected after a new release?

Defining quality gates make the release process better, and you should always consider adding them.

Release Notes and Documentation

When you deploy a new release to a customer or install new software on your server, and you want to communicate what has been released to your customer, the usual way to do this is the use of release notes.

But where do the release notes come from? There are different ways where you can store your release notes, and I will walk through the various ways in this section of the course.

Document store

An often used way of storing release notes is by creating text files, or documents in some document store. This way, the release notes are stored together with other documents. The downside of this approach is that there is no direct connection between the release in the release management tool and the release notes that belong to this release.

Wiki

The most used way that is used at customers is to store the release notes in a Wiki. For example, Confluence from Atlassian, SharePoint Wiki, SlimWiki or the Wiki in Azure DevOps.

The release notes are created as a page in the wiki, and by using hyperlinks, relations can be associated with the build, the release and the artifacts.

In the code base

When you look at it, release notes belong strictly to the release. To the features you implemented and the code you wrote. In that case, the best option might be to store release notes as part of your code repository. The moment, the team completes a feature, they or the product owner also write the release notes and saves these alongside the code. This way it becomes living documentation because the notes change with the rest of the code.

In a work item

Another option is to store your release notes as part of your work items. Work items can be Bugs, Tasks, Product Backlog Items or User Stories. To save release notes in work items, you can create or use a separate field within the work item. In this field, you type the publicly available release notes that will be communicated to the customer. With a script or specific task in your build and release pipeline, you can then generate the release notes and store them as an artifact or publish them to an internal or external website.

The screenshot shows a Microsoft Azure DevOps feature card for item 344. The card has the following details:

- FEATURE 344**
- Title:** 344 Shopping Cart should be personalized
- Status:** Unassigned
- Comments:** 0 comments
- Tags:** Add tag
- State:** New
- Area:** Test demo
- Reason:** New feature
- Iteration:** Test demo
- Description:** The shopping cart should know the user
- Status:** Start Date
- Acceptance Criteria:** Click to add Acceptance Criteria
- Development:** Updated by rvanosnabrugge just now. Development hasn't started on this item.
- Release Notes:** Here can be the commercial release notes.
- Details:** Priority: 2, Effort, Business Value
- Related Work:** Add link. There are no links in this group.
- Discussion:** Add a comment. Use # to link a work item, ! to link a pull request, or @ to mention a person.
- Time Criticality:** Value area: Business

- **Generate Release Notes Build Task¹³**
- **WIKI Updater Tasks¹⁴**
- **Atlassian Confluence¹⁵**
- **Azure DevOps Wiki¹⁶**

There is a difference between functional and technical documentation. There is also a difference between documentation designing the product, mostly written up front and documentation describing the product afterwards, like manuals or help files. Storing technical documentation about your products, that is still in the design phase, is usually done on a document sharing portal, like SharePoint or Confluence.

A better and more modern way to store your documentation is to create a wiki. Wiki's do not contain Documents, Presentations or Spreadsheets but text files called Markdown Files. These markdowns can refer to pictures, can hold code samples and can be part of your code repository. Code repositories can deal very well with text files. Changes and history can be easily tracked by using the native code tools.

However, the most significant advantage of using a Wiki instead of documents is that a Wiki is accessible for everyone in your team. By giving the right permissions to all the team members, people can work together on the documentation instead of having to wait for each other when working on the same document.

Manuals or documentation that you release together with the product, should be treated as source code. When the product changes and new functionality is added, the documentation needs to change as well. You can store the documentation as part of your code repository, or you can create a new repository containing your documentation. In any case, the documentation should be treated the same way as your source code. Create a documentation artifact in your build pipeline, and deliver this artifact to the release pipeline. The release pipeline can then deploy the documentation to a site or just include it in the boxed product.

¹³ <https://marketplace.visualstudio.com/items?itemName=richardfennellBM.BM-VSTS-XplatGenerateReleaseNotes>

¹⁴ <https://marketplace.visualstudio.com/items?itemName=richardfennellBM.BM-VSTS-WIKIUpdater-Tasks>

¹⁵ <https://www.atlassian.com/software/confluence>

¹⁶ <https://azure.microsoft.com/en-us/services/devops/wiki/>

Choosing a deployment pattern

Deployment patterns

Choosing a deployment pattern

A deployment pattern is a way of how you choose to move your application to production. Traditionally the way to do it was to set up a *Development* environment, a *Test* environment and a *Production* environment and perform more or less the same steps on each environment. Nowadays we can have Infrastructure on-demand, and this opens up a new range of possibilities when it comes to deployment patterns. We are not limited to a fixed set of environments but have various options.

When we move towards Continuous Delivery, we also need to consider that deploying multiple times a day, can impact our users. They do not want to be surprised with a changed UI after a refresh or 5 minutes of downtime during a transaction. Modern deployment patterns built upon software architecture patterns. A good example is the use of Feature Toggles that enable you to hide specific functionality.

In this chapter, we will briefly discuss some deployment Patterns. In Module 3 we will cover them in more details, together with some demos and HandsOn exercises. The most important thing to realise is that not every application can follow these deployment patterns without changing the application and architecture.

Traditional Deployment Pattern

Dev, QA, Prod - As new builds are produced, they can be deployed to Dev. They can then be promoted to QA, and finally to Prod. At any time, each of these stages may have a different release (set of build artifacts) deployed to them. This is an excellent example of the use of stages in a release pipeline.

Modern Deployment Patterns

Blue-Green Deployment

The Blue-Green deployment is all about ensuring you have two production environments, as identical as possible. After deployment and tests, the environments are swapped.

Canary Release

A canary release is only pushed to a small number of users. Therefore its impact is relatively small should the new code prove to be buggy. Changes can be reversed quickly.

Dark Launching

Dark launching is the process of releasing production-ready features to a subset of your users before a full release. This enables you to decouple deployment from release, get real user feedback, test for bugs, and assess infrastructure performance.

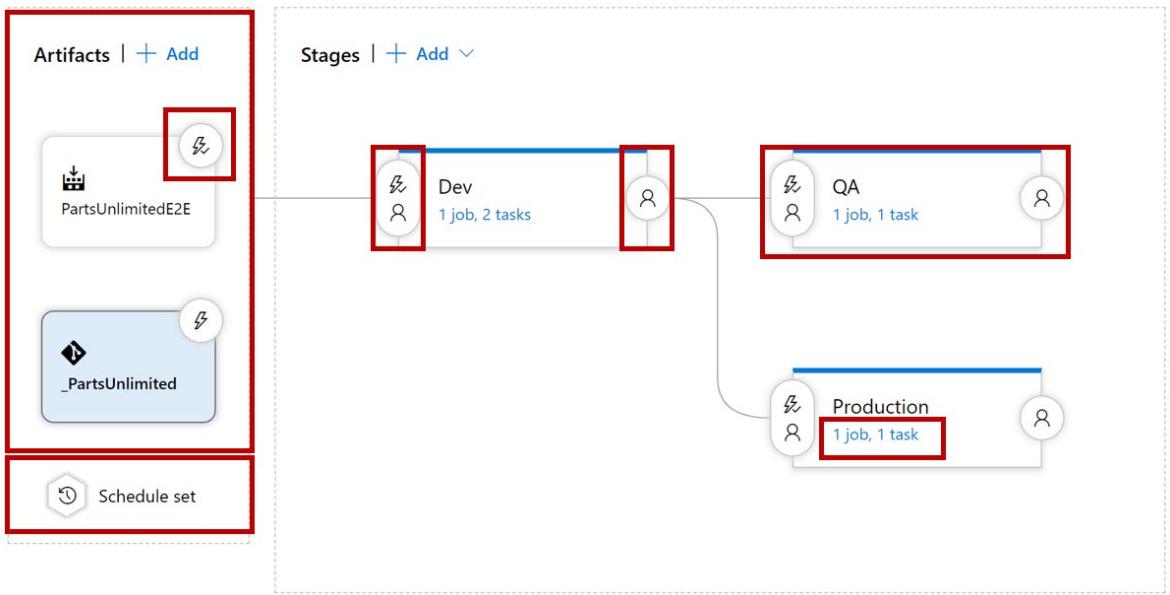
Progressive Exposure Deployment

In a progressive exposure deployment, you expose the new software to a subset of users, that you extend gradually over time. Impact (also called blast radius), is evaluated through observation, testing, analysis of telemetry, and user feedback.

Choosing the right release management tool

Overview Release Management Tools

A release pipeline consists of different components.



When choosing the right Release Management tool, you should look at the possibilities of all the different components and map them to the needs you have. There are many tools available in the marketplace from which we will discuss some in the next chapter. The most important thing to notice is that not every vendor or tool treats Release Management in the same manner.

The tools in the marketplace can be divided into 2 categories

- Tools that can do Build and Continuous Integration and Deployment
- Tools that can do Release Management

In many cases, companies only require the deployment part of Release Management. Deployment, or installing software can be done by all the build tools out there. Primarily because the technical part of the release is executing a script or running a program. Release Management that requires approvals, quality gates, and different stages, needs a different kind of tool that usually tightly integrate with the build and CI tools but are not the same thing.

Considerations for choosing

Previously, we have discussed all the components that are part of the release pipeline, and here we will briefly highlight the things you need to consider when choosing a Release Management Tool.

Artifacts and Artifact source

As we have seen in the previous chapter, artifacts can come from different sources. When you want to treat your artifact as a versioned package, it needs to be stored somewhere before your release pipeline consumes it. Considerations for choosing your tool can be:

- Which Source Control systems are supported?

- Can you have 1 or more artifact sources in your release pipeline? In other words, can you combine artifacts from different sources into one release?
- Does it integrate with your build server?
- Does it support other build servers?
- Does it support container registries?
- How do you secure the connection to the artifact source?
- Can you extend the artifact sources?

Triggers and Schedules

Triggers are an essential component in the pipeline. Triggers are required to start a release but, if you want to have multiple stages, also to start a deployment

Considerations for choosing your trigger can be:

- Does your system support Continuous Deployment triggers.
- Can you trigger releases from an API (for integration with other tools)
- Can you schedule releases
- Can you schedule and trigger each stage separately?

Approvals and gates

Starting a release and using scripts, executables or deployable artifacts does not make the difference between a Continuous Integration/Build tool and a Release Management tool. Adding a release approval workflow to the pipeline is the critical component that does make the difference.

Considerations When it comes to approvals:

- Do you need approvals for your release pipeline?
- Are the approvers part of your company? Do they need a tool license?
- Do you want to use manual or automatic approvals? Or both?
- Can you approve with an API (integration with other tools)
- Can you set up a workflow with approvers (optional and required)?
- Can you have different approvers per stage?
- Can you have more than one approver? Do you need them all to approve?
- What are the possibilities for automatic approval?
- Can you have a manual approval or step in your release pipeline?

Stages

Running a Continuous Integration pipeline that build and deploys your product is a very commonly used scenario.

But what if you want to deploy the same release to different environments? When choosing the right release management tool, you should consider the following things when it comes to stages (or environments)

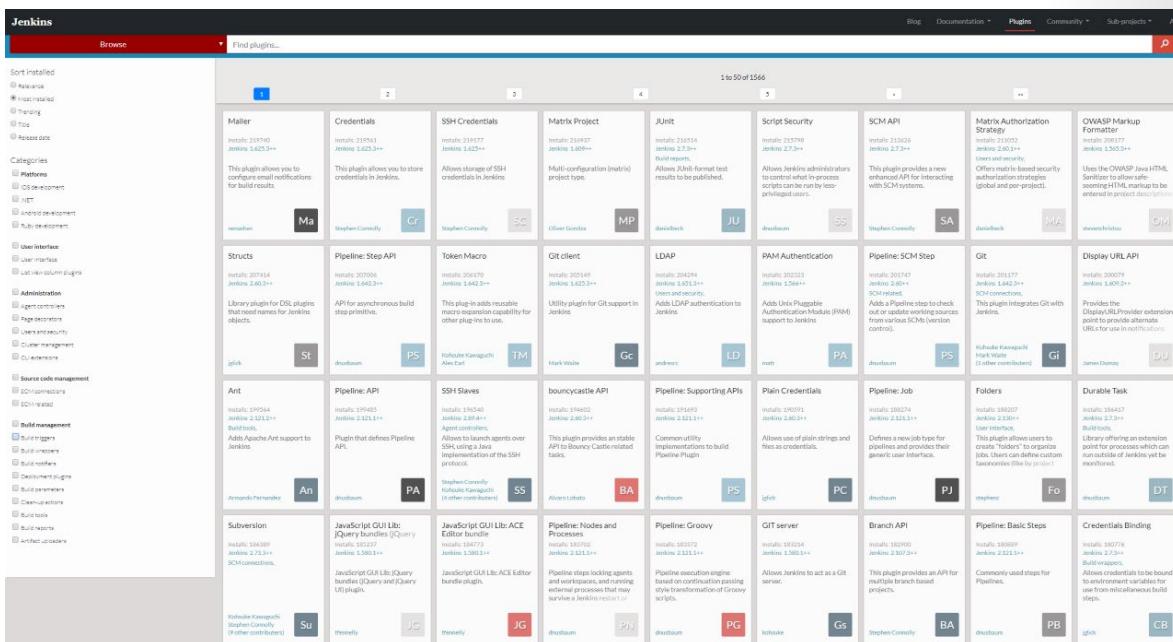
- Can you use the same artifact to deploy to different stages?

- Can you differ the configuration between the stages
- Can you have different steps for each stage?
- Can you follow the release between the stages?
- Can you track the artifacts / work items and source code between the stages?

Build and Release Tasks

Last but not least the work needs to be done within the pipeline. It is not only about the workflow and orchestration, the code needs to be deployed or installed. Things to consider when it comes to the execution of tasks

- How do you create your steps? Is it running a script (bat, shell, PowerShell cli) or are there specialised tasks?
- Can you create your own tasks?
- How do tasks authenticate to secure sources?
- Can tasks run on multiple platforms?
- Can tasks be easily reused
- Can you integrate with multiple environments? (Linux, Windows, Container Clusters, PaaS, Cloud)
- Can you control the tasks that are used in the pipeline



The screenshot shows the Visual Studio Marketplace interface, specifically the Azure DevOps extensions section. At the top, there are navigation tabs for Visual Studio, Visual Studio Code, Azure DevOps (which is highlighted in pink), Subscriptions, and links to sign in and publish extensions. Below the tabs is a search bar with the placeholder "Search Azure DevOps extensions". To the right of the search bar are filters for "Showing: All categories", "Hosted On: Any", "Price: Any", and "Sort By: Downloads". The main content area displays a grid of 864 extension cards, each with a thumbnail, title, developer, download count, description, rating, and status (e.g., FREE or FREE TRIAL). Some extensions shown include Code Search, Test & Feedback, VSTS Open in Excel, Azure Artifacts, Folder Management, Slack Integration, Analytics, Work Item Visualization, Microsoft Teams Integration, SonarQube, Delivery Plans, IIS Web App Deployment, Test Manager, Team Calendar, HockeyApp, CatLight, Replace Tokens, Docker Integration, Octopus Deploy Integration, Test Case Explorer, Release Management, Branch Visualization, AWS Tools for Microsoft, and Estimate.

Traceability, Auditability and Security

One of the most essential things in enterprises and companies that need to adhere to compliance frameworks is Traceability, Auditability and Security. Although this is not explicitly related to a release pipeline, it is an important part.

When it comes to compliance 3 principles are fundamental:

- 4-eyes principle
- Is the deployed artifact reviewed by at least one other person.
- Is the person that deploys another person as the one that writes the code

- Traceability
 - Can we see where the released software originates from (which code)
 - Can we see the requirements that led to this change
 - Can we follow the requirements through the code, build and release
- Auditability
 - Can we see who, when and why the release process changed
 - Can we see who, when and why a new release has been deployed

Security is vital in this. When people can do everything, including deleting evidence, this is not ok. Setting up the right roles, permissions and authorisation are important to protect your system and your pipeline.

When looking at an appropriate Release Management tool, you can consider

- Does it integrate with your company's Active Directory?
- Can you set up roles and permissions?
- Is there change history of the release pipeline itself?
- Can you ensure the artifact did not change during the release?
- Can you link requirements to the release?
- Can you link source code changes to the release pipeline?
- Can you enforce approval or 4-eyes principle?
- Can you see release history and the people who triggered the release?

Release Management Tools

The following tools are mainstream in the current ecosystem. You will find links to the product websites where you can explore the product and see if it fits the needs as we described in the previous chapter.

- What Artifacts and Artifact source does the tool support
- What Triggers and Schedules?
- Does the tool support Approvals and gates
- Can you define multiple stages?
- How does the Build and Release Tasks work?
- How does the tool deal with Traceability, Auditability and Security
- What is the Extensibility model?

Per tool is indicated if it is part of a bigger suite. Integration with a bigger suite gives you a lot of advantages regarding traceability, security and auditability. A lot of integration is already there out of the box.

Jenkins

The leading open source automation server, Jenkins provides hundreds of plugins to support building, deploying and automating any project.

- On-prem system. Offered as SaaS by third-party

- No part of a bigger suite
- Industry standard, especially in the full stack space
- Integrates with almost every source control system
- Rich ecosystem of plugins
- CI/Build tool with deployment possibilities.
- No release management capabilities

Links

- [Jenkins¹⁷](#)
- [Tutorial: Jenkins CI/CD to deploy an ASP.NET Core application to Azure Web App service¹⁸](#)
- [Azure Friday - Jenkins CI/CD with Service Fabric¹⁹](#)

Circle CI

CircleCI's continuous integration and delivery platform help software teams rapidly release code with confidence by automating the build, test, and deploy process. CircleCI offers a modern software development platform that lets teams ramp quickly, scale easily, and build confidently every day.

- CircleCI is a cloud-based system or an on-prem system
- Rest API — you have access to projects, build and artifacts
- The result of the build is going to be an artifact.
- Integration with GitHub and BitBucket
- Integrates with various clouds
- Not part of a bigger suite
- Not fully customizable

Links

- [circleci/²⁰](#)
- [How to get started on CircleCI 2.0: CircleCI 2.0 Demo²¹](#)

Azure DevOps Pipelines

Azure Pipelines helps you implement a build, test, and deployment pipeline for any app. Tutorials, references, and other documentation show you how to configure and manage continuous integration and Continuous Delivery (CI/CD) for the app and platform of your choice.

- Hosted on Azure as a SaaS in multiple regions and available as an on-prem product
- Complete Rest API for everything around Build and Release Management

¹⁷ <https://jenkins.io/>

¹⁸ <https://cloudblogs.microsoft.com/opensource/2018/09/21/configure-jenkins-cicd-pipeline-deploy-asp-net-core-application/>

¹⁹ <https://www.youtube.com/watch?v=5RYmoolZqS4>

²⁰ <https://circleci.com/>

²¹ <https://www.youtube.com/watch?v=KhjwnTD4oec>

- Integration with many build and source control systems (Github, Jenkins, Azure Repos, Bitbucket, Team Foundation Version Control, etc.)
- Cross Platform support, all languages and platforms
- Rich marketplace with extra plugins, build tasks and release tasks and dashboard widgets
- Part of the Azure DevOps suite. Tightly integrated
- Fully customizable
- Manual approvals and Release Quality Gates supported
- Integrated with (Azure) Active Directory
- Extensive roles and permissions

Links

- [Azure Pipelines²²](#)
- [Building and Deploying your Code with Azure Pipelines²³](#)

GitLab Pipelines

GitLab helps teams automate the release and delivery of their applications to enable them to shorten the delivery lifecycle, streamline manual processes and accelerate team velocity. With Continuous Delivery (CD), built into the pipeline, deployment can be automated to multiple environments like staging and production, and support advanced features such as canary deployments. Because the configuration and definition of the application are version controlled and managed, it is easy to configure and deploy your application on demand.

[GitLab²⁴](#)

Atlassian Bamboo

Bamboo is a continuous integration (CI) server that can be used to automate the release management for a software application, creating a Continuous Delivery pipeline.

[Atlassian Bamboo²⁵](#)

XL Deploy/XL Release

XL Release is an end-to-end pipeline orchestration tool for Continuous Delivery and DevOps teams. It handles automated tasks, manual tasks, and complex dependencies and release trains. And XL Release is designed to integrate with your change and release management tools.

[xl-release - XebiaLabs²⁶](#)

²² <https://azure.microsoft.com/en-us/services/devops/pipelines/>

²³ <https://www.youtube.com/watch?v=NuYDAs3kNV8>

²⁴ <https://about.gitlab.com/stages-devops-lifecycle/release/>

²⁵ <https://www.atlassian.com/software/bamboo/features>

²⁶ <https://xebialabs.com/products/xl-release/>

Module Review and Takeaways

Module Review Questions

Multiple choice

Would adding a feature flag increase or decrease the cyclomatic complexity of the code?

- Increase
- Decrease

Multiple choice

Would adding a feature flag increase or decrease technical debt?

- Increase
- Decrease

Suggested answer

You plan to slowly increase the traffic to a newer version of your site. What type of deployment pattern is this?

Suggested answer

When you want to change an immutable object of any type, what do you do?

Suggested answer

What can you use to prevent a deployment in Azure DevOps when a security testing tool finds a compliance problem?

Answers

Multiple choice

Would adding a feature flag increase or decrease the cyclomatic complexity of the code?

- Increase
- Decrease

Multiple choice

Would adding a feature flag increase or decrease technical debt?

- Increase
- Decrease

You plan to slowly increase the traffic to a newer version of your site. What type of deployment pattern is this?

Blue-green

When you want to change an immutable object of any type, what do you do?

You make a new one and (possibly) remove the old one

What can you use to prevent a deployment in Azure DevOps when a security testing tool finds a compliance problem?

Release gate

Module 11 Set up a Release Management Workflow

Module Overview

Module Overview

Continuous Delivery is much more about enabling teams within your organization. Enable them to deliver the software on demand. Making it possible that you can press a button at any time of the day, and still have a good product means a number of things. It says that the code needs to be high quality, the build needs to be fully automated and tested, and the deployment of the software needs to be fully automated and tested as well.

Now we need to dive a little bit further into the release management tooling. We will include a lot of things coming from Azure pipelines. A part of the Azure DevOps suite. Azure DevOps is an integrated solution for implementing DevOps and Continuous Delivery in your organization. We will cover some specifics of Azure pipelines, but this does not mean they do not apply for other products available in the marketplace. Many of the other tools share the same concepts and only differ in naming.

Release pipelines

A release pipeline, in its simplest form, is nothing more than the execution of a number of steps. In this module, we will dive a little bit further into the details of one specific stage. The steps that need to be executed and the mechanism that you need to execute the steps within the pipeline.

In this module, we will talk about agent and agent pools that you might need to execute your release pipeline. We will look at variables for the release pipeline and the various stages.

After that, we dive into the tasks that you can use to execute your deployment. Do you want to use script files or do you want to use specific tasks that can perform one job outstanding? For example, the marketplaces of both Azure DevOps and Jenkins have a lot of tasks in the store that you can use to make your life a lot easier.

We will talk about secrets and secret management in your pipeline. A fundamental part to secure your not only your assets but also the process of releasing your software. At the end of the module, we will talk about alerting mechanisms. How to report on your software, how to report on your quality and how

to get notified by using service hooks. Finally, we will dive a little bit further into automatic approvals using automated release gates.

Learning objectives

After completing this module, students will be able to:

- Explain the terminology used in Azure DevOps and other Release Management Tooling
- Describe what a Build and Release task is, what it can do, and some available deployment tasks
- Classify an Agent, Agent Queue, and Agent Pool
- Explain why you sometimes need multiple release jobs in one release pipeline
- Differentiate between multi-agent and multi-configuration release job
- Use release variables and stage variables in your release pipeline
- Deploy to an environment securely using a service connection
- Embed testing in the pipeline
- List the different ways to inspect the health of your pipeline and release by using alerts, service hooks, and reports
- Create a release gate

Create a Release Pipeline

Release pipelines Classic interface or YAML

Azure DevOps Release Pipelines

Azure DevOps has extended support for pipelines as code (also referred to as YAML pipelines) for continuous deployment and started to introduce various release management capabilities into pipelines as code. The existing UI based release management solution in Azure DevOps is referred to as classic release. In the table below you'll find a list of capabilities and their availability in YAML pipelines vs classic build and release pipelines.

Feature	YAML	Classic Build	Classic Release	Notes
Agents	Yes	Yes	Yes	Specifies a required resource on which the pipeline runs.
Approvals	Yes	No	Yes	Defines a set of validations required prior to completing a deployment stage.
Artifacts	Yes	Yes	Yes	Supports publishing or consuming different package types.
Caching	Yes	Yes	No	Reduces build time by allowing outputs or downloaded dependencies from one run to be reused in later runs. In Preview, available with Azure Pipelines only.
Conditions	Yes	Yes	Yes	Specifies conditions to be met prior to running a job.
Container jobs	Yes	No	No	Specifies jobs to run in a container.
Demands	Yes	Yes	Yes	Ensures pipeline requirements are met before running a pipeline stage. Requires self-hosted agents.

Feature	YAML	Classic Build	Classic Release	Notes
Dependencies	Yes	Yes	Yes	Specifies a requirement that must be met in order to run the next job or stage.
Deployment groups	Yes	No	Yes	Defines a logical set of deployment target machines.
Deployment group jobs	No	No	Yes	Specifies a job to release to a deployment group.
Deployment jobs	Yes	No	No	Defines the deployment steps. Requires Multi-stage pipelines experience.
Environment	Yes	No	No	Represents a collection of resources targeted for deployment. Available with Azure Pipelines only.
Gates	No	No	Yes	Supports automatic collection and evaluation of external health signals prior to completing a release stage. Available with Azure Pipelines only.
Jobs	Yes	Yes	Yes	Defines the execution sequence of a set of steps.
Service connections	Yes	Yes	Yes	Enables a connection to a remote service that is required to execute tasks in a job.
Service containers	Yes	No	No	Enables you to manage the lifecycle of a containerized service.

Feature	YAML	Classic Build	Classic Release	Notes
Stages	Yes	No	Yes	Organizes jobs within a pipeline.
Task groups	No	Yes	Yes	Encapsulates a sequence of tasks into a single reusable task. If using YAML, see templates.
Tasks	Yes	Yes	Yes	Defines the building blocks that make up a pipeline.
Templates	Yes	No	No	Defines reusable content, logic, and parameters.
Triggers	Yes	Yes	Yes	Defines the event that causes a pipeline to run.
Variables	Yes	Yes	Yes	Represents a value to be replaced by data to pass to the pipeline.
Variable groups	Yes	Yes	Yes	Use to store values that you want to control and make available across multiple pipelines.

Definitions and Glossary

We have covered the concepts of a release pipeline. Many terms and definitions are used. The hardest part is that these terms vary from tool to tool. In this part, you find a list of names and definitions and synonyms.

In this list, you find the term that is used in Azure DevOps.

Term	Description	Synonym
Stage	an isolated and independent target for deployment	Environment
Job	A phase in the release pipeline that can run simultaneously with other phases on different Operating Systems	Phases
Agent	The program that runs the build or release	
Build & Release Task	Tasks are units of executable code used to perform designated actions in a specified order.	Action, Plugin, App

Term	Description	Synonym
Release pipeline	The process that runs when deploying an artifact. Including triggers, approvals and gates	release process, pipeline, release definition
CI/CD	Continuous Integration / Continuous Deployment	
Release gate	An automated check that approves the continuation	Quality Gate, Automatic Approval
Service Connection	A secure connection to an environment or service	Service Endpoint

Build and Release Tasks

A build and release platform requires the ability to execute any number of repeatable actions during the build process. Tasks are units of executable code used to perform designated actions in a specified order.

Add tasks | Refresh Search

All Build **Utility** Test Package Deploy Tool Marketplace

- Download Secure File**
Download a secure file to a temporary location on the build or release agent
- Extract Files**
Extract a variety of archive and compression files such as .7z, .rar, .tar.gz, and .zip.
- FTP Upload**
FTP Upload
- GitHub Release**
Create, edit, or delete a GitHub release.
- Install Apple Certificate**
Install an Apple certificate required to build on a macOS agent
- Install Apple Provisioning Profile**
Install an Apple provisioning profile required to build on a macOS agent
- Install SSH Key**
Install an SSH key prior to a build or release

Add

Add steps to specify what you want to build, the tests that you want to run, and all of the other steps needed to complete the build process. There are steps for building, testing, running utilities, packaging, and deploying.

If a task is not available, you can find a lot of community tasks in the marketplace. Jenkins, Azure DevOps and Atlassian have an extensive marketplace where additional tasks can be found.

Links

For more information, see also:

- [Task types & usage¹](#)
- [Tasks for Azure²](#)
- [Atlassian marketplace³](#)
- [Jenkins Plugins⁴](#)
- [Azure DevOps Marketplace⁵](#)

Important Deployment Tasks

The choice of deployment tasks depends on the target environment to which you want to deploy your software.

If you want to deploy a mobile app, you can create a package and upload it with a script or use a task that directly deploys to the Apple Store or Google Play store.

If you want to deploy to a server running on premises, then you might want to use a PowerShell or command line script to install the software. Or, if you're going to use a specialised task, you can use the web deployment task, that deploys MS deploy packages directly on an IIS

You can also use an SSH Task and use, Shell tasks to execute the necessary actions on the target server.

When you want to deploy to Azure or another cloud, you can use command line tools specialised for that cloud, or some specialised tasks that you can use to install your resources.

When you want to deploy to a container cluster, you can run a docker command in a command line task, or you can use specific docker tasks. They do a lot of the plumbing and secure connections to the container cluster and container registry.

In the following section, a list of out of the box tasks is listed so you can see what is available for you to deploy. Regardless of the environment that you want to target.

¹ <https://docs.microsoft.com/en-us/azure/devops/pipelines/process/tasks?view=azure-devops&tabs=yaml>

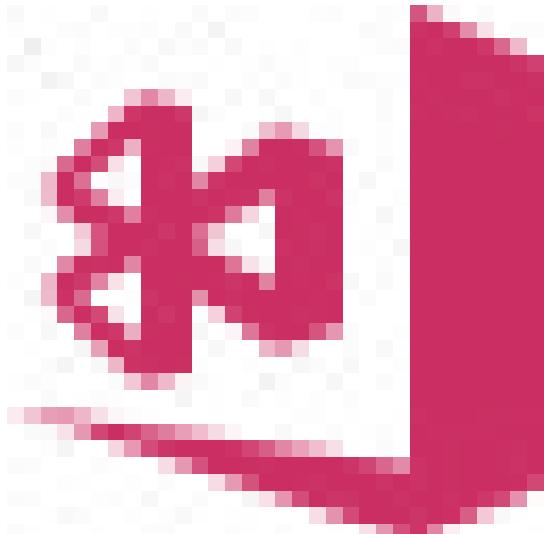
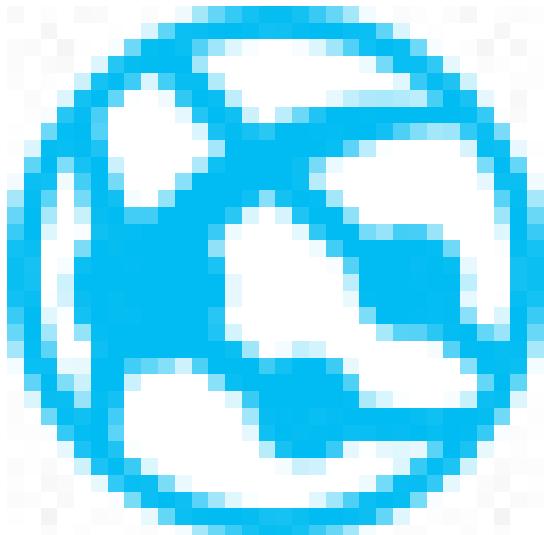
² <https://github.com/microsoft/azure-pipelines-tasks>

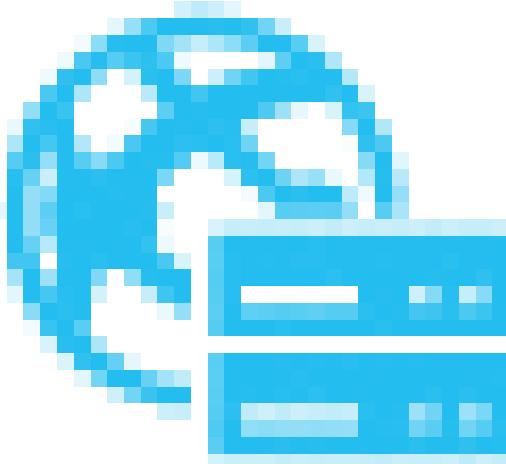
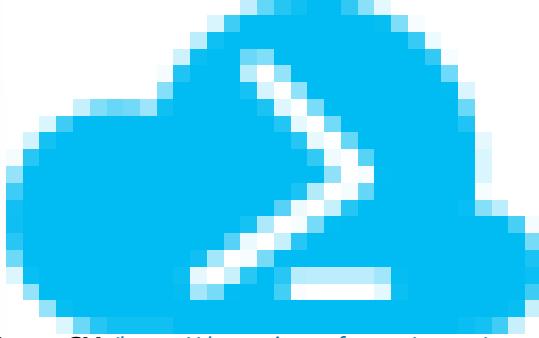
³ <https://marketplace.atlassian.com/add-ons/app/bamboo/trending>

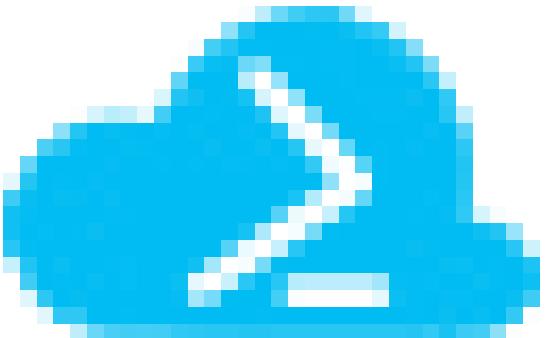
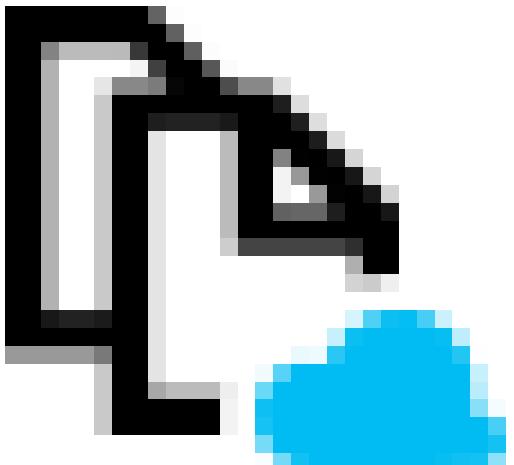
⁴ <https://plugins.jenkins.io/>

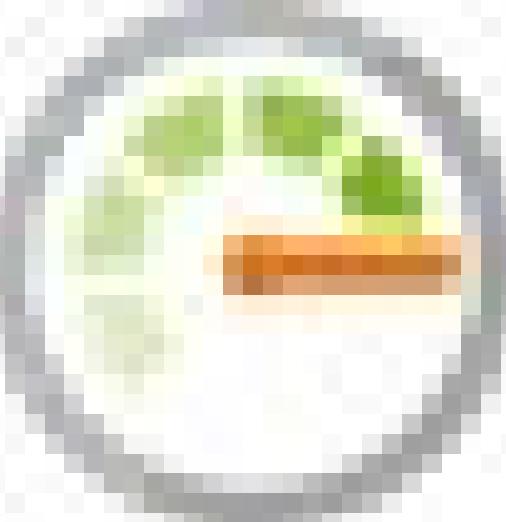
⁵ <https://marketplace.visualstudio.com/>

Deploy

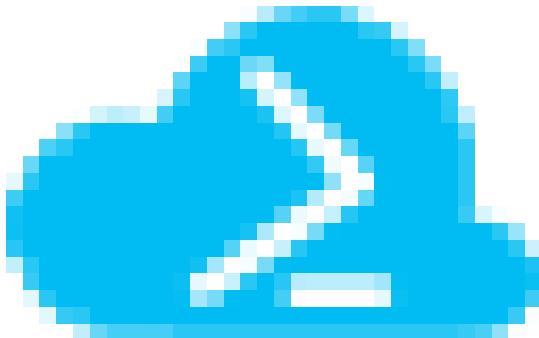
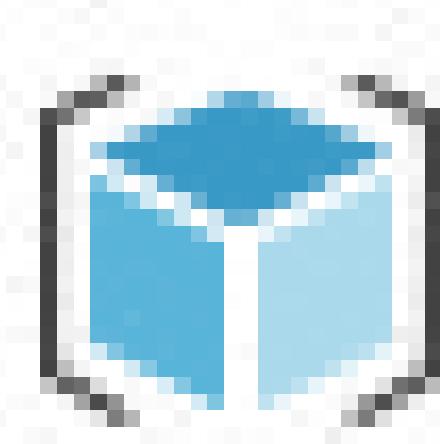
	Task
	Distribute app builds to testers and users via App Center
	Update Azure App Service using Web Deploy / Kudu REST APIs

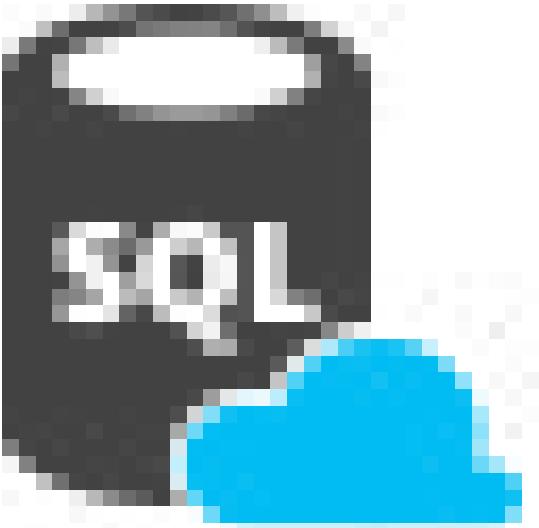
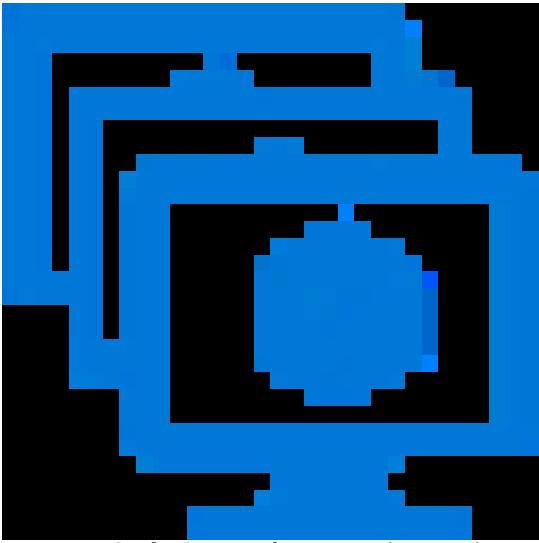
	Task
	Start, Stop, Restart or Slot swap for an Azure App Service
Azure App Service Manage (https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-app-service-manage?view=vsts)	Run a shell or batch script containing Azure CLI commands against an Azure subscription
	
Azure CLI (https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-cli?view=vsts)	

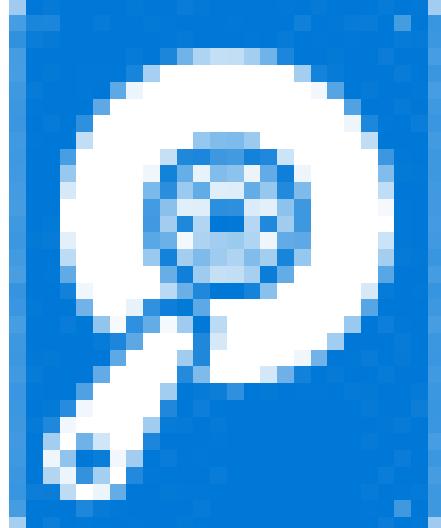
	Task
	Deploy an Azure Cloud Service
	Azure Cloud PowerShell Deployment (https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-cloud-powershell-deployment?view=vsts)
	Copy files to Azure blob or VM(s)
	Azure File Copy (https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-file-copy?view=vsts)

Task	
 Azure Key Vault (https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-key-vault?view=vsts)	Incorporate secrets from an Azure Key Vault into a release pipeline
 Azure Monitor Alerts (https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-monitor-alerts?view=vsts)	Configure alerts on available metrics for an Azure resource

	Task
 Azure MySQL Deployment (https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-mysql-deployment?view=vsts)	Run your scripts and make changes to your Azure DB for MySQL.
 Azure Policy Check Gate (https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-policy-check-gate?view=vsts)	Security and compliance assessment with Azure policies on resources that belong to the resource group and Azure subscription.

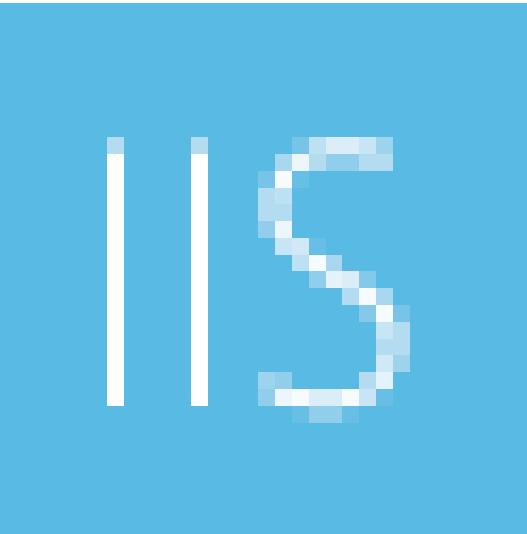
	Task
 A pixelated blue icon of a cloud with white highlights, representing Azure.	Run a PowerShell script within an Azure environment
Azure PowerShell (https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-powershell?view=vsts)  A pixelated blue icon of a cloud with a thick black border, representing Azure.	Deploy, start, stop, delete Azure Resource Groups

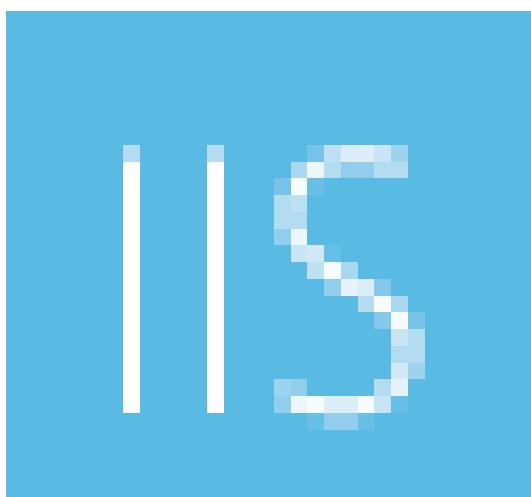
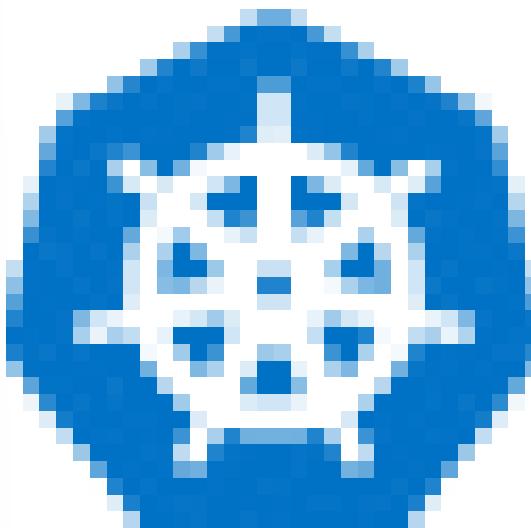
	Task
	Deploy an Azure SQL database using DACPAC or run scripts using SQLCMD
	Deploy a virtual machine scale set image.

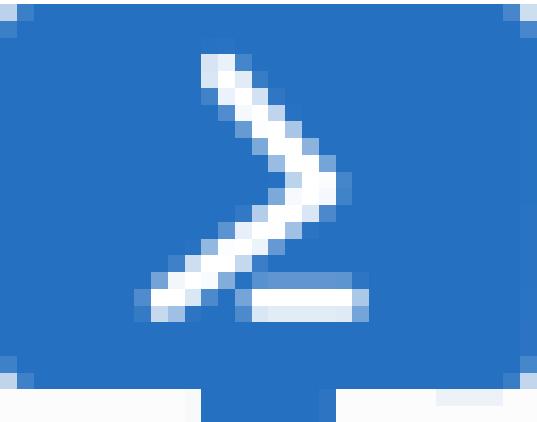
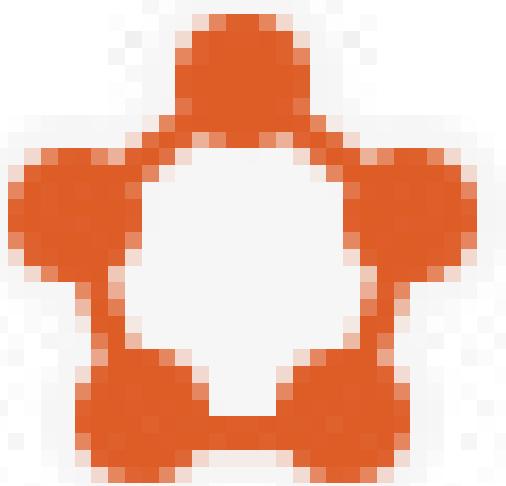
	Task
 Build Machine Image (Packer) (https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/packer-build?view=vsts)	Build a machine image using Packer.
 Chef (https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/chef?view=vsts)	Deploy to Chef environments by editing environment attributes

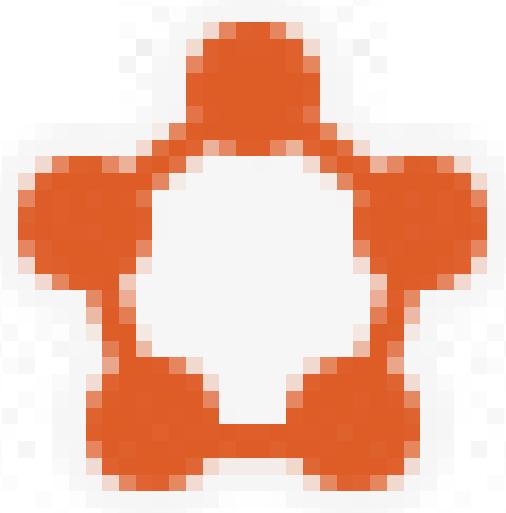
	Task
 Chef Knife (https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/chef-knife?view=vsts)	Run Scripts with knife commands on your chef workstation
 Copy Files Over SSH (https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/copy-files-over-ssh?view=vsts)	Copy files from source folder to target folder on a remote machine over SSH

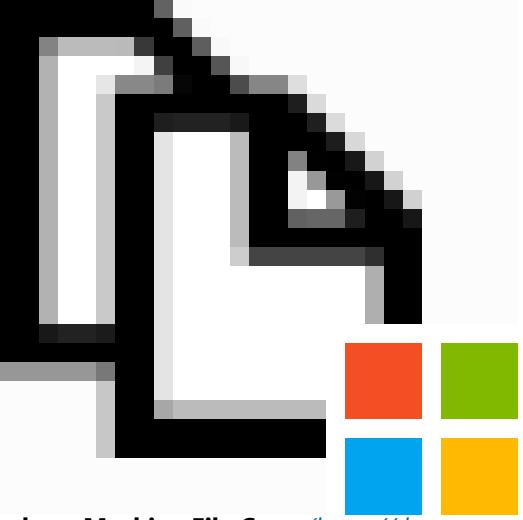
	Task
Docker	Build, tag, push, or run Docker images, or run a Docker command. Task can be used with Docker or Azure Container registry
Docker Compose	Build, push or run multi-container Docker applications.

	Task
 The Helm logo consists of the word "HELM" in a bold, sans-serif font, with each letter composed of a grid of blue and grey squares. The letters are arranged in three rows: a top row with two letters, a middle row with three letters, and a bottom row with two letters. Helm Deploy (https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/helm-deploy?view=vsts)	Deploy, configure, update your Kubernetes cluster in Azure Container Service by running helm commands.
 The IIS logo icon features a stylized "IIS" text where the letters are formed by a series of vertical bars and horizontal lines, all set against a solid blue background. IIS Web App Deploy (https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/iis-web-app-deployment-on-machine-group?view=vsts)	Deploy a website or web app to a machine group using WebDeploy

	Task
 IIS Web App Manage (https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/iis-web-app-management-on-machine-group?view=vsts)	Create or update a website, web app, virtual directory, or application pool on a machine group
 Kubernetes (https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/kubernetes?view=vsts)	Deploy, configure, update your Kubernetes cluster in Azure Container Service by running kubectl commands.

	Task
 PowerShell on Target Machines (https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/powershell-on-target-machines?view=vsts)	Execute PowerShell scripts on remote machine(s)
 Service Fabric Application Deployment (https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/service-fabric-deploy?view=vsts)	Deploy a Service Fabric application to a cluster

	Task
	Deploy a Service Fabric application to a cluster using a compose file
Service Fabric Compose Deploy (https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/service-fabric-compose-deploy?view=vsts) 	Run shell commands or a script on a remote machine using SSH
SSH (https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/ssh?view=vsts)	

	Task
	Copy files to remote machine(s)
	Deploy a SQL Server database using DACPAC or SQL scripts

Refer to **Build and release tasks**⁶ for a full list of tasks.

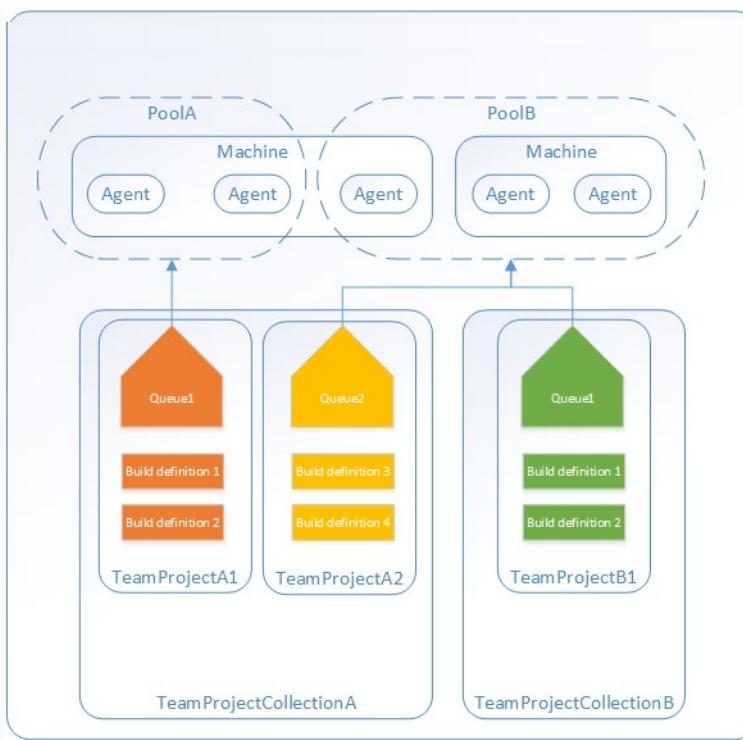
Agents, Agent Pools and Queues

A build agents is a running background service which listens for commands from the central server and starts and executes the actual build or release processes. It is installed and configured separately from the server (or saas solution).

⁶ <https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/index?view=vsts>

An agent can most of the times run on different Operating Systems and in some cases as a SaaS (hosted agent) solution.

Agents are organized into pools and provide access to the pools by using queues.



Agent pools

Agent pools are used to organize and define permission boundaries around your agents. In Azure DevOps, Pools are scoped to your organization. You can share your pool across multiple team project collections.

Agent Queues

An agent queue provides access to a pool of agents. When you create a build or release definition, you specify which queue it uses. Queues are scoped to your team project collection so that you can share them across build and release definitions in multiple team projects.

Build Agents and Server Options

Build agents are tasked with performing the builds. To build your code or deploy your software, you need at least one agent. As you add more code and add more people, you will eventually need more agents to form a pool.

Pipelines are based on build agents, which are implemented as a background service installed on a virtual machine. Build agents are then grouped into *agent pools*, which are supplied from an *agent queue*. This means that build pipelines do not interact with build agents directly, but instead place a request in an agent queue, which is owned by a team project. Each agent queue feeds a single agent pool, and multiple queues may feed the same agent. The pool then provides build agents based on availability and the build definition requirements.

Build agents come in two types:

Hosted agents. These agents exist within their own hosted pool and are maintained and upgraded by the vendor. Hosted agents have specific limitations and advantages:

- Hosted agents have no cost and are immediately available, and have most common software and libraries installed.
- Do not have an interactive mode.
- Do not allow administrative privilege or allow logon.

Hosted agents come in many flavours. Based on the needs of your product and the platform that you target you can choose an appropriate agent. The agent software is cross-platform and can run on any platform

New agent pool...

Manage organization agent pools

All agent pools

- Default (Default)
- Hosted (Hosted)
- Hosted macOS (Hosted macOS) ...
- Hosted Ubuntu 1604 (Hosted Ubuntu 1604)
- Hosted VS2017 (Hosted VS2017)
- Hosted Windows Container (Hosted Windows Container)

Private (or Custom) agents. Private agents are provisioned on private virtual machines (VMs) and are custom built to accommodate the project's needs.

System capabilities

System capabilities are name/value pairs that you can use to ensure that your build definition is run only by agents that meet the criteria that you specified. Environment variables automatically appear in the list. Some capabilities (such as frameworks) are also added automatically.

When a build is queued, the system sends the job only to agents that have the capabilities **demanded by the build definition⁷**.

⁷ <https://docs.microsoft.com/en-us/azure/devops/pipelines/build/options?view=vsts&tabs=yaml>

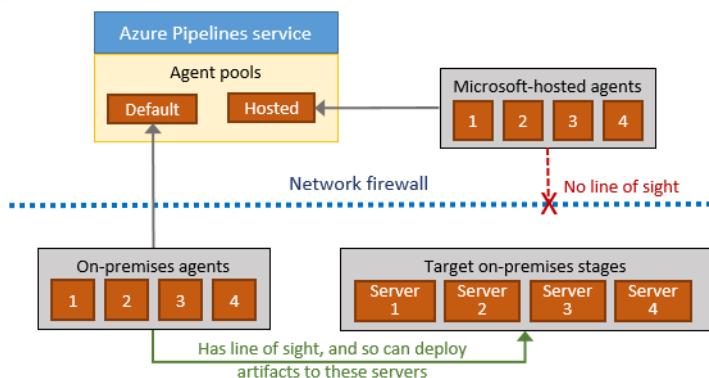
User capabilities

You can manually add capabilities (name/value pairs) that you know your agent has and that you want your build definition to be able to **demand**⁸.

Agents and Deployment Groups

Agents run on a separate server and install and execute actions on a deployment target. By having an agent installed in this way, you need to have permissions to access the target environment. The target environment needs to accept incoming traffic. When you use Hosted agents, running in the cloud, you cannot use these agents to deploy software on your on-premises server.

To overcome this issue, you can run agents in your local network that communicate with the central server.



If this also does not work, you need to fall back on the traditional approach of installing agents on the target environment. In Azure DevOps, this is called Deployment Groups. The agent on the target server is registered and do everything themselves. This means that they only need to have outbound access and is sometimes the only allowed option.

Learn more

For more information, see also:

- [Azure Pipelines agents⁹](#)
- [Provision deployment groups¹⁰](#)
- [Microsoft-hosted agents¹¹](#)
- [Agent pools¹²](#)
- [Self-hosted Linux agents¹³](#)
- [Deploy an Azure Pipeline agent in Windows¹⁴](#)
- [Deploy a build and release agent on macOS¹⁵](#)

⁸ <https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/v2-linux?view=vsts>

⁹ <https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/agents?view=vsts>

¹⁰ <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/deployment-groups/?view=vsts>

¹¹ <https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/hosted?view=vsts&tabs=yaml>

¹² <https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/pools-queues?view=vsts>

¹³ <https://docs.microsoft.com/en-us/vsts/build-release/actions/agents/v2-linux?view=vsts>

¹⁴ <https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/v2-windows?view=vsts>

¹⁵ <https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/v2-osx?view=vsts>

- Deploy a Azure Pipelines agent on Linux¹⁶

Release Agent Jobs Introduction

What is a Release Agent Job?

You can organize your build or release pipeline into jobs. Every build or deployment pipeline has at least one job.

A job is a series of tasks that run sequentially on the same target. This can be a Windows server, a Linux server, a container or a deployment group. A release job is executed by a build/release agent. This agent can only execute one job at the same time.

During the design of your job, you specify a series of tasks that you want to run on the same agent. At runtime (when either the build or release pipeline is triggered), each job is dispatched as one or more jobs to its target.

A scenario that speaks to the imagination, where Jobs play an essential role is the following.

Assume that you built an application, with a backend in .Net, a front end in Angular and a native IOS mobile App. This might be developed in 3 different source control repositories triggering three different builds, delivering three different artifacts.

The release pipeline brings the artifacts together and wants to deploy the backend, frontend, and Mobile App all together as part of 1 release. The deployment needs to take place on different agents. An IOS app needs to be built and distributed from a Mac, and the angular app is hosted on Linux so best deployed from a Linux machine. The backend might be deployed from a Windows machine.

Because you want all three deployments to be part of one pipeline, you can define multiple Release Jobs, which target the different agents, server or deployment groups.

By default, jobs run on the host machine where the agent is installed. This is convenient and typically well-suited for projects that are just beginning to adopt continuous integration (CI). Over time, you may find that you want more control over the stage where your tasks run.

¹⁶ <https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/v2-linux?view=vsts>

The screenshot shows the 'All pipelines > Release Jobs Picture' section of the Azure Pipelines interface. At the top, there are tabs: Pipeline, Tasks (which is selected), Variables, Retention, Options, and History. Below the tabs, there's a 'Development Deployment process' section with a 'macOS Job' card. This card includes a 'Run on agent' icon and a '+' button. Other cards listed include 'Publish to the App Store TestFlight track' (Apple App Store Release), 'Deployment group job' (Run on deployment group), 'PowerShell Script' (PowerShell), 'Deploy IIS Website/App:' (IIS Web App Deploy), 'Ubuntu Job' (Run on agent), and 'Release n/a to internal' (Google Play - Release). Each card has a '+' button to its right.

For more information, see [Jobs in Azure Pipelines and TFS¹⁷](#).

Using Release Jobs

You can have different Release Jobs. In this chapter, they are briefly covered

Jobs or Agent Jobs

A job is a series of tasks that run sequentially on the same target. At design time in your job, you specify a set of tasks that you want to run on a common target. At runtime (when either the build or release pipeline is triggered), each job is dispatched as one or more jobs to its target.

When the target is an agent, the tasks are run on the computer that hosts the agent. This agent can only execute one job at the same time.

For more information, see [Specify jobs in your pipeline¹⁸](#).

Server or Agentless Jobs

Tasks in a server or agentless job are orchestrated by and executed on the server (Azure Pipelines or TFS). A server job does not require an agent or any target computers. Only a few tasks, such as the Manual Intervention and Invoke REST API tasks, are supported in a server job at present.

¹⁷ <https://docs.microsoft.com/en-us/azure/devops/pipelines/process/phases?view=vsts&tabs=yaml>

¹⁸ <https://docs.microsoft.com/en-us/azure/devops/pipelines/process/phases?view=vsts&tabs=yaml>

Container Jobs

Containers offer a lightweight abstraction over the host operating system. You can select the exact versions of operating systems, tools, and dependencies that your build requires. When you specify a container in your pipeline, the agent will first fetch and start the container. Then, each step of the job will run inside the container.

For more information, see [Define container jobs \(YAML\)¹⁹](#).

Deployment Group Jobs

Deployment groups make it easy to define groups of target servers for deployment. Tasks that you define in a deployment group job run on some or all of the target servers, depending on the arguments you specify for the tasks and the job itself.

You can select specific sets of servers from a deployment group to receive the deployment by specifying the machine tags that you have defined for each server in the deployment group. You can also specify the proportion of the target servers that the pipeline should deploy to at the same time. This ensures that the app running on these servers is capable of handling requests while the deployment is taking place.

For more information, see [Deployment group jobs²⁰](#).

Multi-Configuration and Multi-Agent

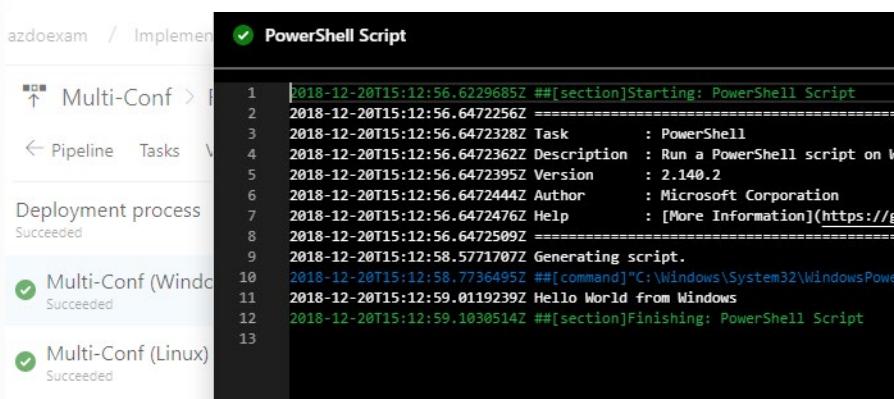
We talked about Jobs and how to make it possible to divide the work between different agents or servers. Sometimes you want to run the tasks of a pipeline multiple times but with a slightly different configuration, or split a large batch of work amongst multiple agents.

There are three different types of job you can run.

- **None:** Tasks will run on a single agent.
- **Multi-configuration:** Run the same set of tasks on multiple configurations as specified in the multipliers. Configurations will run in parallel, and each configuration will use a single agent. For example
 - Run the release once with configuration Setting A on WebApp A and setting B for WebApp B
 - Deploy to different geographic regions.
 - Multi-configuration testing: run a set of tests in parallel - once for each test configuration.

¹⁹ <https://docs.microsoft.com/en-us/azure/devops/pipelines/process/container-phases?view=vsts&tabs=yaml>

²⁰ <https://docs.microsoft.com/en-us/azure/devops/pipelines/process/deployment-group-phases?view=vsts&tabs=yaml>



The screenshot shows the Azure DevOps Pipeline interface. On the left, there's a navigation bar with 'azdoexam' and 'Implement'. Below it are sections for 'Deployment process' (Succeeded) and two stages: 'Multi-Conf (Windows)' and 'Multi-Conf (Linux)', both also marked as 'Succeeded'. The main area is titled 'PowerShell Script' and contains the following log output:

```
1 2018-12-20T15:12:56.6229685Z ##[section]Starting: PowerShell Script
2 2018-12-20T15:12:56.6472256Z =====
3 2018-12-20T15:12:56.6472395Z Task : PowerShell
4 2018-12-20T15:12:56.6472395Z Description : Run a PowerShell script on Windows
5 2018-12-20T15:12:56.6472444Z Version : 2.140.2
6 2018-12-20T15:12:56.6472444Z Author : Microsoft Corporation
7 2018-12-20T15:12:56.6472476Z Help : [More Information](https://go.microsoft.com/fwlink/?linkid=857607)
8 2018-12-20T15:12:56.6472509Z =====
9 2018-12-20T15:12:58.5771707Z Generating script.
10 2018-12-20T15:12:58.7736495Z ##[command]"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -NoProfile -InputFormat None -ExecutionPolicy Bypass -Command ".\d844a2e4-1a2c-4f3b-9a2a-1a2c4f3b9a2a.ps1"
11 2018-12-20T15:12:59.0119239Z Hello World from Windows
12 2018-12-20T15:12:59.1030514Z ##[section]Finishing: PowerShell Script
13
```

- **Multi-agent:** Run the same set of tasks on multiple agents using the specified number of agents. For example, you can run a broad suite of 1000 tests on a single agent. Or, you can use two agents and run 500 tests on each one in parallel.

For more information, see [Specify jobs in your pipeline²¹](#).

Discussion

How to use Release jobs

Do you see a purpose for Release Jobs in your pipeline? How would you set it up?

Topics you might want to consider are:

- Do you have artifacts from multiple sources?
- Do you want to run deployments on different servers simultaneously?
- Do you need multiple platforms?
- How long does your release take?
- Can you run your deployment in parallel or does it need to run in sequence?

Release Variables

Variables give you a convenient way to get critical bits of data into various parts of the pipeline. As the name suggests, the contents of a variable may change between releases, stages of jobs of your pipeline. The system predefines some variables, and you are free to add your own as well.

The most important thing you need to think about when using variables in the release pipeline is the scope of the variable. You can imagine that a variable containing the name of the target server may vary between a Development environment and a Test Environment.

Within the release pipeline, you can use variables in different scopes and different ways.

For more information, see [Release variables and debugging²²](#).

²¹ <https://docs.microsoft.com/en-us/azure/devops/pipelines/process/phases?view=vsts&tabs=designer#multi-configuration>

²² <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/variables?view=vsts&tabs=batch>

Predefined variables

When running your release pipeline, there are always variables that you need that come from the agent or context of the release pipeline. For example, the agent directory where the sources are downloaded, the build number or build id, the name of the agent or any other information. This information is usually accessible in pre-defined variables that you can use in your tasks.

Release pipeline variables

Choose a release pipeline variable when you need to use the same value across all the stages and tasks in the release pipeline, and you want to be able to change the value in a single place.

Stage variables

Share values across all of the tasks within one specific stage by using stage variables. Use a stage-level variable for values that vary from stage to stage (and are the same for all the tasks in a stage).

Variable groups

Share values across all of the definitions in a project by using variable groups. We will cover variable groups later in this module.

Normal and Secret variables

Because the tasks of the pipeline are executed on an agent, usually variable values are passed to the various tasks using environment variables. The task knows how to read this. You should be aware that a variable contains clear text and can be exposed on the target system. When you use the variable in the log output, you can also see the value of the variable. When the pipeline has finished, the values will be cleared.

You can mark a variable in the release pipeline as secret. This way the secret is hidden from the log output. This is especially useful when writing a password or other sensitive information

The screenshot shows the 'Variables' tab in the Azure DevOps interface. At the top, there are tabs for 'Variables', 'Retention', 'Options', and 'History'. Below the tabs is a search bar labeled 'Filter by keywords' and a 'Scope' dropdown set to 'Release'. To the right of the search bar are 'List' and 'Grid' buttons. On the far right, there is a 'Scope' dropdown menu with options: 'Release' (selected), 'Dev', 'Test', and 'Release'. The main area displays a table of variables:

Name	Value
Prefix	Demo
ServerName	DevServer
ServerName	TestServer
Password	*****

Each row in the table has a delete icon at the end. The 'ServerName' row under 'Test' scope and the 'Password' row both have a lock icon at the end.

Provision and Configure Environments

Provision and Configure Different Target Environments

The release pipeline deploys software to a target environment. But, it is not only the software that will be deployed with the release pipeline. If you are truly focusing on Continuous Delivery, Infrastructure as Code and spinning up infrastructure as part of your release pipeline is very important.

When we focus on the deployment of the infrastructure, we should first consider the differences between the target environments that we can deploy to.

- On-Premises servers
- Cloud servers or Infrastructure as a Service (IaaS). For example Virtual machines or networks.
- Platform as a Service (PaaS) and Functions as a Service (FaaS). For example Web apps or storage accounts.
- Clusters.

Let us dive a bit further into these different target environments.

On-Premises servers

In most cases, when you deploy to an on-premises server, the hardware and the operating system is already in place. The server is already there and ready. Sometimes empty but most of the times not. In this case, the release pipeline can focus on deploying the application only.

In some cases, you might want to start or stop a virtual machine (for example Hyper-V or VMWare). The scripts that you use to start or stop the on-premises servers should be part of your source control and be delivered to your release pipeline as a build artifact. Using a task in the release pipeline, you can run the script that starts or stops the servers.

When you want to take it one step further, and you want to configure the server as well. You should take a look at technologies like PowerShell Desired State Configuration(DSC), or use tools like Puppet and Chef. All these products will maintain your server and keep it in a particular state. When the server changes its state, they (Puppet, Chef, DSC) recover the changed configuration to the original configuration.

Integrating a tool like Puppet, Chef or Powershell DSC in to the release pipeline is no different from any other task you add.

Infrastructure as a service

When you use the cloud as your target environment things change a little bit. Some organizations did a lift and shift from their on-premises server to cloud servers. Then your deployment works the same as to an on-premises server. But when you use the cloud to provide you with Infrastructure as a Service (IaaS), you can leverage the power of the cloud, to start and create servers when you need them.

This is where Infrastructure as Code (IaC) starts playing a significant role. By creating a script or template, you can create a server or other infrastructural components like a SQL server, a network or an IP address. By defining a template or using a command line and save it in a script file, you can use that file in your release pipeline tasks to execute this on your target cloud. As part of your pipeline, the server (or another component) will be created. After that, you can execute the steps actually to deploy the software.

Technologies like Azure Resource Manager (ARM) or Terraform are great to create infrastructure on demand.

Platform as a Service

When you are moving from Infrastructure as a Service (IaaS) towards Platform as a Service (PaaS), you will get the infrastructure from the cloud that you are running on.

For example: In Azure, you can choose to create a Web application. The server, the hardware, the network, the public IP address, the storage account, and even the web server, is arranged by the cloud. The user only needs to take care of the web application that will run on this platform.

The only thing that you need to do is to provide the templates which instruct the cloud to create a WebApp. The same goes for Functions as a Service(FaaS or Serverless technologies. In Azure called Azure Functions and in AWS called AWS Lambda.

You only deploy your application, and the cloud takes care of the rest. However, you need to instruct the platform (the cloud) to create a placeholder where your application can be hosted. You can define this template in ARM or Terraform. You can use the Azure CLI or command line tools or in AWS use CloudFormation. In all cases, the infrastructure is defined in a script file and live alongside the application code in source control.

Clusters

Last but not least you can deploy your software to a cluster. A cluster is a group of servers that work together to host high-scale applications.

When you run a cluster as Infrastructure as a Service, you need to create and maintain the cluster. This means that you need to provide the templates to create a cluster. You also need to make sure that you roll out updates, bug fixes and patches to your cluster. This is comparable with Infrastructure as a Service.

When you use a hosted cluster, you should consider this as Platform as a Service. You instruct the cloud to create the cluster, and you deploy your software to the cluster. When you run a container cluster, you can use the container cluster technologies like Kubernetes or Docker Swarm.

Summary

Regardless of the technology, you choose to host your application, the creation, or at least configuration of your infrastructure should be part of your release pipeline and part of your source control repository. Infrastructure as Code is a fundamental part of Continuous Delivery and gives you the freedom to create servers and environments on demand.

Links

- [AWS Cloudformation²³](https://aws.amazon.com/cloudformation/)
- [Terraform²⁴](https://www.terraform.io/)
- [Powershell DSC²⁵](https://docs.microsoft.com/en-us/powershell/scripting/dsc/overview/overview?view=powershell-7)
- [AWS Lambda²⁶](https://docs.aws.amazon.com/lambda/latest/dg/welcome.html)

²³ <https://aws.amazon.com/cloudformation/>

²⁴ <https://www.terraform.io/>

²⁵ <https://docs.microsoft.com/en-us/powershell/scripting/dsc/overview/overview?view=powershell-7>

²⁶ <https://docs.aws.amazon.com/lambda/latest/dg/welcome.html>

- **Azure Functions**²⁷
- **Chef**²⁸
- **Puppet**²⁹
- **Azure Resource Manager /ARM**³⁰

Demonstration Setting Up Service Connections

In this demonstration, you will investigate Service Connections.

Note: Before starting this walkthrough, make sure you have performed the steps in the prerequisites section and the previous walkthroughs.

To follow along with this walkthrough, you will need to have an existing Azure subscription, that contains an existing storage account.

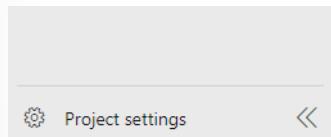
Steps

Let's now take a look at how a release pipeline can access resources that require a secure connection. In Azure DevOps, these are implemented by Service Connections.

You can set up a service connection to environments to create a secure and safe connection to the environment that you want to deploy to. Service connections are also used to get resources from other places in a secure manner. For example, you might need to get your source code from GitHub.

In this case, let's take a look at configuring a service connection to Azure.

1. From the main menu in the **Parts Unlimited** project, click **Project settings** at the bottom of the screen.



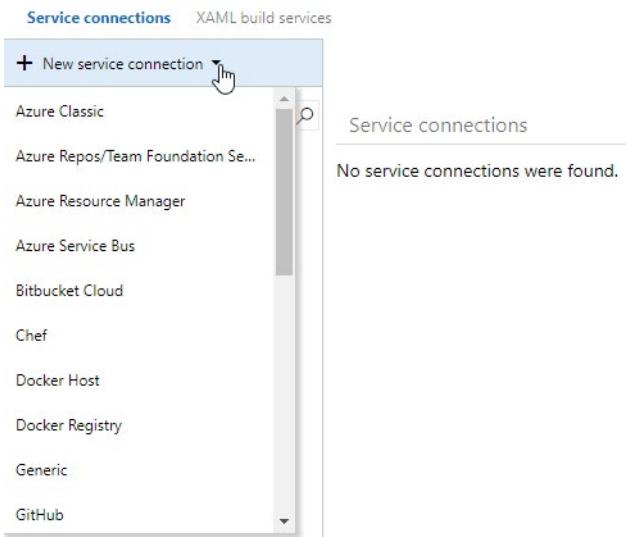
2. In the Project Settings pane, from the **Pipelines** section, click **Service connections**. Click the drop down beside **+New service connection**.

²⁷ <https://azure.microsoft.com/en-us/services/functions>

²⁸ <https://www.chef.io/chef/>

²⁹ <https://puppet.com/>

³⁰ <https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-overview>



As you can see, there are many types of service connections. You can create a connection to the Apple App Store or to the Docker Registry, to Bitbucket, or to Azure Service bus.

In this case, we want to deploy a new Azure resource, so we'll use the Azure Resource Manager option.

3. Click **Azure Resource Manager** to add a new service connection.

Add an Azure Resource Manager service connection

Service Principal Authentication Managed Identity Authentication

Connection name	
Scope level	Subscription
Subscription	()
Resource Group	

Subscriptions listed are from Azure Cloud

A new Azure service principal will be created and assigned with "Contributor" role, having access to all resources within the subscription. Optionally, you can select the Resource Group to which you want to limit access.

If your subscription is not listed above, or your organization is not backed by Azure Active Directory, or to specify an existing service principal, [use the full version of the service connection dialog.](#)

Allow all pipelines to use this connection.

OK Close

4. Set the **Connection name** to **ARM Service Connection**, click on an Azure **Subscription**, then select an existing **Resource Group**.

Note: You might be prompted to logon to Azure at this point. If so, logon first.

Add an Azure Resource Manager service connection

Service Principal Authentication Managed Identity Authentication

Connection name: ARM Service Connection

Scope level: Subscription

Subscription: Microsoft

Resource Group: SQLDEMO

Subscriptions listed are from Azure Cloud

A new Azure service principal will be created and assigned with "Contributor" role, having access to all resources within the subscription. Optionally, you can select the Resource Group to which you want to limit access.

If your subscription is not listed above, or your organization is not backed by Azure Active Directory, or to specify an existing service principal, use the [full version of the service connection dialog](#).

Allow all pipelines to use this connection.

OK **Close**

Notice that what we are actually creating is a **Service Principal**. We will be using the Service Principal as a means of authenticating to Azure. At the top of the window, there is also an option to set up Managed Identity Authentication instead.

The Service Principal is a type of service account that only has permissions in the specific subscription and resource group. This makes it a very safe way to connect from the pipeline.

5. Click **OK** to create it. It will then be shown in the list.

Service connections XAML build services

+ New service connection ▾

Filter connections...

ARM Service Connection

Service connection: ARM Service Connection

Details Roles Request history Policies

INFORMATION

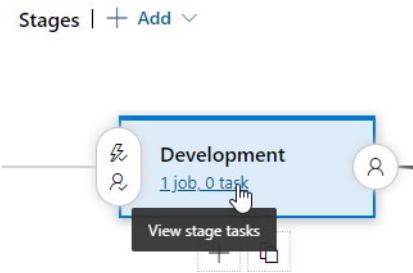
Type: Azure Resource Manager
Created by Greg Low
Connected to service using Service Principal

ACTIONS

List of actions that can be performed on this service connection:

Update service connection
Manage service connection roles
Manage Service Principal
Disconnect

6. In the main Parts Unlimited menu, click **Pipelines** then **Releases*, then **Edit** to see the release pipeline. Click the link to **View stage tasks**.



The current list of tasks is then shown. Because we started with an empty template, there are no tasks as yet. Each stage can execute many tasks.

All pipelines > **Release to all environments**

Pipeline Tasks Variables Retention Options History

Development
Deployment process

Agent job
Run on agent

7. Click the + sign to the right of **Agent job** to add a new task. Note the available list of task types.

Add tasks | Refresh

Search

All Build Utility Test Package Deploy Tool Marketplace

- .NET Core**
Build, test, package, or publish a dotnet application, or run a custom dotnet command
- Android signing**
Sign and align Android APK files
- Ant**
Build with Apache Ant
- App Center distribute**
Distribute app builds to testers and users via Visual Studio App Center

8. In the **Search** box, enter the word **storage** and note the list of storage-related tasks. These include standard tasks, and tasks available from the Marketplace.

The screenshot shows the Azure DevOps Marketplace search results for 'storage'. At the top, there are buttons for 'Add tasks' and 'Refresh', and a search bar with the text 'storage' and a magnifying glass icon. Below the search bar, a list of tasks is displayed:

- Azure file copy**: Copy files to Azure Blob Storage or virtual machines.
- Azure Storage**: Tasks to assist with the creation of storage accounts, containers therein and uploading of files.
- Azure Storage Container**: Task for creating azure storage container with a defined public access level inside an existing storage account.
- Manage Storage Account Release Tools**: Tools for managing creation Storage Accounts and its objects.
- Maven Cache**: Tasks for upload and download Maven cache from an Azure storage account.

We will use the Azure file copy task to copy one of our source files to a storage account container.

9. Hover over the **Azure file copy** task type, and click **Add** when it appears. The task will be added to the stage but requires further configuration.

The screenshot shows the Azure DevOps pipeline editor. The pipeline is named 'Development' and is part of a 'Deployment process'. It consists of two stages: 'Agent job' and 'File Copy'. The 'Agent job' stage contains a single task named 'File Copy'. A red warning icon with the text 'Some settings need attention' is displayed next to the task name.

10. Click the **File Copy** task to see the required settings.

Azure file copy ①

View YAML Remove

Task version 2.*

Display name *

File Copy

Source *

This setting is required.

Azure Connection Type

Azure Resource Manager

Azure Subscription *

Manage

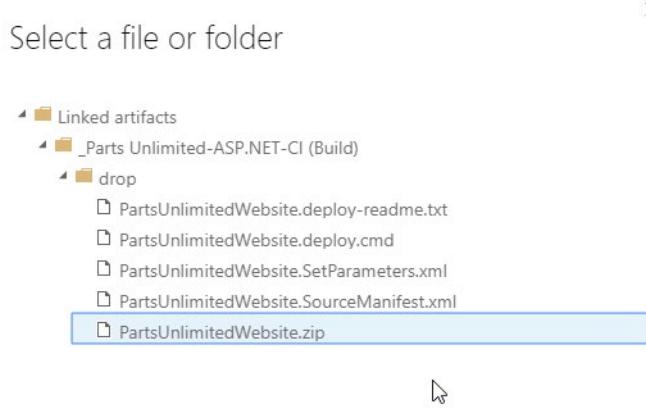
① This setting is required.

Destination Type *

RM Storage Account *

① This setting is required.

11. Set the **Display Name** to **Backup website zip file**, then click the ellipsis beside **Source** and locate the file as follows, then click **OK** to select it.



The artifacts published by each version will be available for deployment in release pipelines. The last successful version of **_Parts Unlimited-ASP.NET-CI (Build)** published the following artifacts: **drop**.

Location _Parts Unlimited-ASP.NET-CI/drop/PartsUnlimitedWebsite.zip

OK

Cancel

We then need to provide details of how to connect to the Azure subscription. The easiest and most secure way to do that is to use our new Service Connection.

12. From the **Azure Subscription** drop down list, find and select the **ARM Service Connection** that we created.

Azure Subscription * ⓘ | Manage ⓘ

Available Azure service connections

ARM Service Connection

13. From the **Destination Type** drop down list, select **Azure Blob**, and from the **RM Storage Account** and **Container Name**, select the storage account, and enter the name of the container, then click **Save** at the top of the screen and **OK**.

Azure Subscription * ⓘ | Manage ⓘ

ARM Service Connection

Scoped to resource group 'SQLDEMO'

Destination Type * ⓘ

Azure Blob

RM Storage Account * ⓘ

devopsoutput

Container Name * ⓘ

websitezipfileoutput

14. To test the task, click **Create release**, and in the **Create a new release** pane, click **Create**.

15. Click the new release to view the details.

All pipelines > Release to all environments

Release Release-3 has been created

Pipeline Tasks Variables Retention Options History

Development Deployment process

16. On the release page, approve the release so that it can continue.

17. Once the **Development** stage has completed, you should see the file in the Azure storage account.

Authentication method: Access key ([Switch to Azure AD User Account](#))
Location: websitezipfileoutput

Search blobs by prefix (case-sensitive) Sh

NAME	MODIFIED	ACCESS TIER	BLOB TYPE	SIZE
PartsUnlimitedWebsite.zip	9/2/2019, 12:04:39 PM	Hot (Inferred)	Block blob	9.55 MiB

A key advantage of using service connections is that this type of connection is managed in a single place within the project settings, and doesn't involve connection details spread throughout the pipeline tasks.

Manage and Modularize Tasks and Templates

Introduction

When you deploy multiple applications and have multiple builds and release pipelines, reusability comes into mind.

For example, you deploy to the same set of servers and use the same connection strings. You want to store the values in one place so you can easily update them.

The same goes for some actions you always want to perform together. For example, stop a website, running a script, start a website.

Sometimes you want to take this even one step further. You want to do some specialized actions, that are not available in the public marketplace. By writing a reusable task that you can use in your organization or even make public, is an excellent way to encapsulate logic and distribute this safely and securely.

Within the Azure DevOps suite, three important concepts enable reusability.

- Task Groups
- Variable Groups
- Custom Build and Release Tasks

Task Groups

A task group allows you to encapsulate a sequence of tasks, already defined in a build or a release pipeline, into a single reusable task that can be added to a build or release pipeline, just like any other task. You can choose to extract the parameters from the encapsulated tasks as configuration variables, and abstract the rest of the task information.

Task groups are a way to standardize and centrally manage deployment steps for all your applications. When you include a task group in your definitions, and then make a change centrally to the task group, the change is automatically reflected in all the definitions that use the task group. There is no need to change each one individually.

For more information, see [Task groups for builds and releases³¹](#).

Variable Groups

A variable group is used to store values that you want to make available across multiple builds and release pipelines.

Examples

- Store the username and password for a shared server
- Store a share connection string
- Store the geolocation of an application
- Store all settings for a specific application

For more information, see [Variable Groups for Azure Pipelines and TFS³²](#).

³¹ <https://docs.microsoft.com/en-us/azure/devops/pipelines/library/task-groups?view=vsts>

³² <https://docs.microsoft.com/en-us/azure/devops/pipelines/library/variable-groups?view=vsts>

Custom Tasks

Instead of using the out-of-the-box tasks, or using a command line or shell script, you can also use your custom build and release task. By creating your own tasks, the tasks are available for publicly or privately to everyone you share it with.

Creating your own task has significant advantages.

- You get access to variables that are otherwise not accessible,
- you can use and reuse secure endpoint to a target server
- you can safely and efficiently distribute across your whole organization
- users do not see implementation details.

For more information, see **Add a build or release task³³**.

Demonstration Creating and Managing Task Groups

In this demonstration, you will investigate Task Groups.

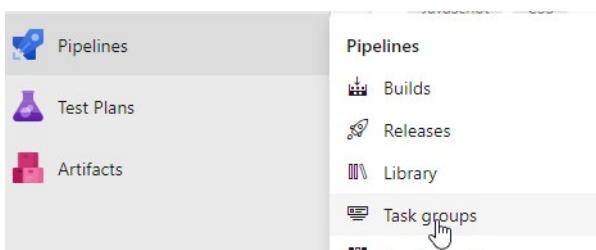
Note: Before starting this walkthrough, make sure you have performed the steps in the prerequisites section and the previous walkthroughs.

Steps

Let's now take a look at how a release pipeline can reuse groups of tasks.

It's common to want to reuse a group of tasks in more than one stage within a pipeline or in different pipelines.

1. In the main menu for the **Parts Unlimited** project, click **Pipelines** then click **Task groups**.



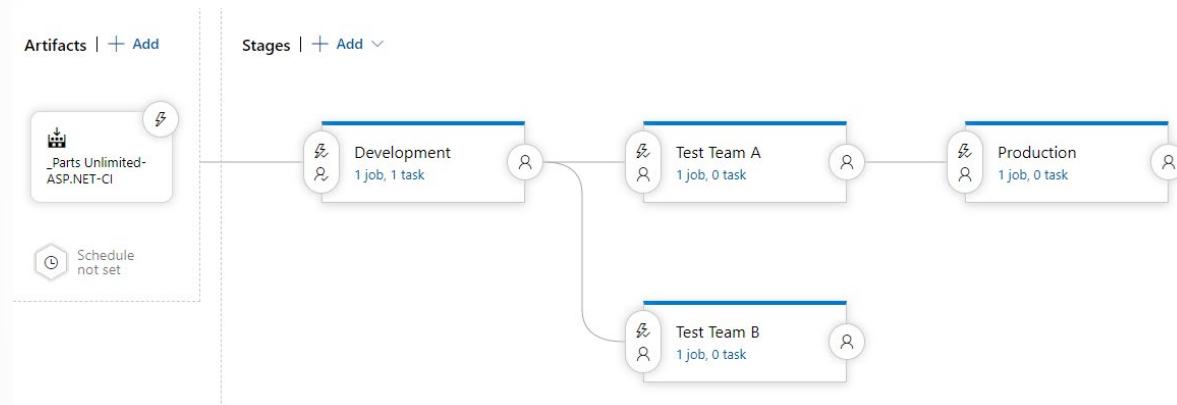
You will notice that you don't currently have any task groups defined.

[Task groups](#) | [Import](#) [Security](#)

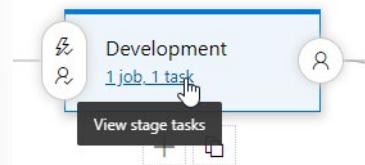
³³ <https://docs.microsoft.com/en-us/azure/devops/extend/develop/add-build-task?view=vsts>

There is an option to import task groups but the most common way to create a task group is directly within the release pipeline, so let's do that.

2. In the main menu, click **Pipelines** then click **Releases**, and click **Edit** to open the pipeline that we have been working on.



3. The **Development** stage currently has a single task. We will add another task to that stage. Click the **View stage tasks** link to open the stage editor.



You can see that there is currently one task.

A screenshot of the 'Development' stage editor. It shows a task named 'Backup website zip file' with an 'Azure file copy' action. A '+' sign is visible next to the task list, indicating where to add new tasks.

4. Click the + sign to the right of the **Agent job** line to add a new task. In the **Search** box, type **database**.

Add tasks | Refresh

database

- SQL Server database deploy**
Deploy a SQL Server database using DACPAC or SQL scripts
- Azure SQL Database deployment**
Deploy an Azure SQL Database using DACPAC or run scripts using SQLCMD
- MySQL database deploy**
Run scripts and make changes to a MySQL Database
- Azure Database for MySQL deployment**
Run your scripts and make changes to your Azure Database for MySQL

We will add a task to deploy an Azure SQL Database.

5. Hover over the **Azure SQL Database Deployment** option and click **Add**. Click the **Azure SQL DacpacTask** when it appears in the list, to open the settings pane.

Azure SQL Database deployment ⓘ

View YAML Remove

Task version 1.*

Display name *

Azure SQL DacpacTask

Azure Service Connection Type

Azure Resource Manager

Azure Subscription * ⓘ | Manage ↗

① This setting is required.

6. Set the **Display name** to **Deploy devopslog database**, and from the **Azure Subscriptions** drop down list, click **ARM Service Connection**.

Note: we are able to reuse our service connection here

Display name *

Deploy devopslog database

Azure Service Connection Type

Azure Resource Manager

Azure Subscription * ⓘ | Manage ↗

ARM Service Connection

① Scoped to resource group 'SQLDEMO'

7. In the **SQL Database** section, set a unique name for the SQL Server, set the **Database** to **devopslog**, set the **Login** to **devopsadmin**, and set any suitable password.

SQL Database ^

Authentication Type * ⓘ

SQL Server Authentication

Azure SQL Server * ⓘ

devopssqlserver.database.windows.net

Database * ⓘ

devopslog

Login * ⓘ

devopsadmin

Password * ⓘ

[REDACTED]

8. In the **Deployment Package** section, set the **Deploy type** to **Inline SQL Script**, set the **Inline SQL Script** to:

```
CREATE TABLE dbo.TrackingLog
(
    TrackingLogID int IDENTITY(1,1) PRIMARY KEY,
    TrackingDetails nvarchar(max)
);
```

Deployment Package ^

Deploy type *

Inline SQL Script

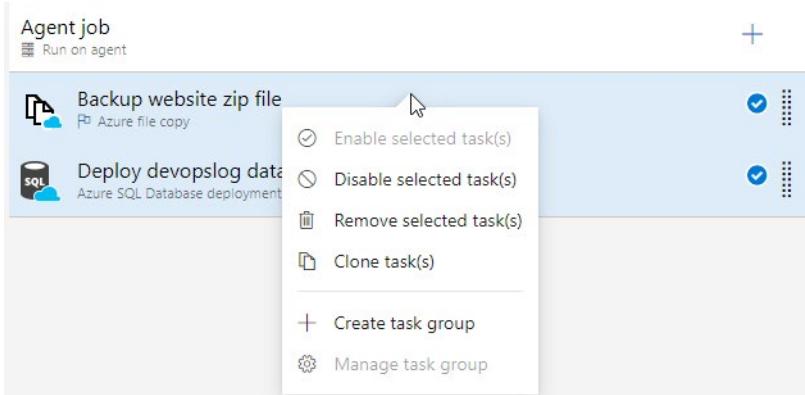
Inline SQL Script * ⓘ

CREATE TABLE dbo.TrackingLog
(
 TrackingLogID int IDENTITY(1,1) PRIMARY KEY,
 TrackingDetails nvarchar(max)
);

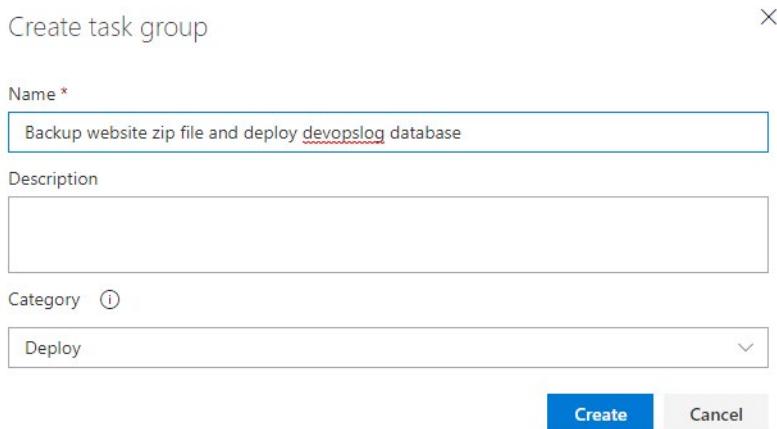
9. Click **Save** then **OK** to save the work.

Now that we have two tasks, let's use them to create a task group.

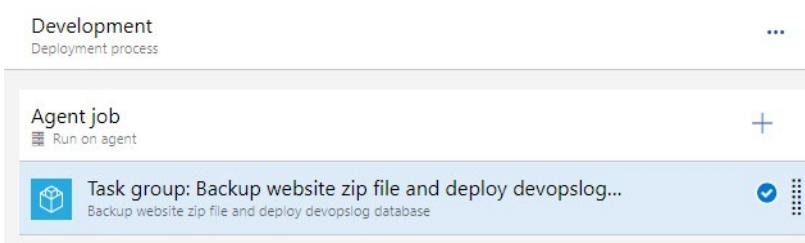
10. Click to select the **Backup website zip file** task and also select the **Deploy devopslog database** task, then right-click either task.



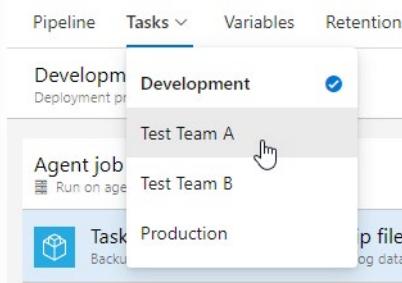
11. Click **Create task group**, then in the **Create task group** window, set **Name** to **Backup website zip file and deploy devopslog**. Click the **Category** drop down list to see the available options. Ensure that **Deploy** is selected, and click **Create**.



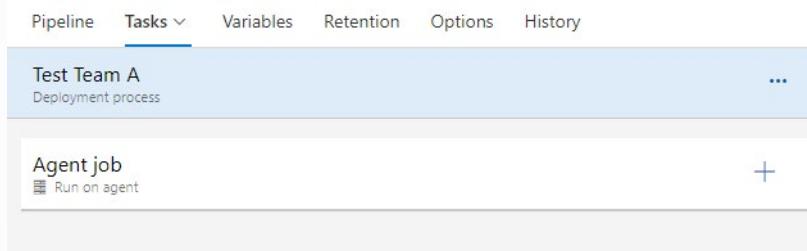
In the list of tasks, the individual tasks have now disappeared and the new task group appears instead.



12. From the **Task** drop down list, select the **Test Team A** stage.



There are currently no tasks in the stage.



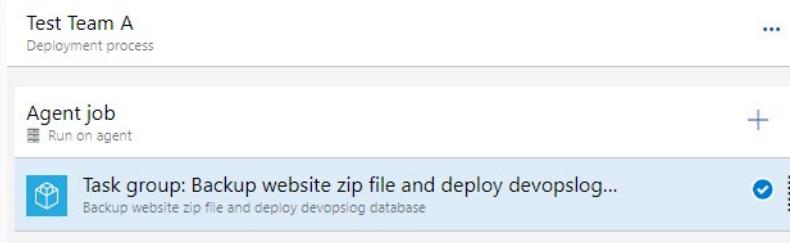
13. Click the + sign to the right of **Agent job** to add a new task. In the **Search** box, type **backup** and notice that the new task group appears like any other task.

Add tasks | Refresh

Search: backup

Backup website zip file and deploy devopslog database

14. Hover on the task group and click **Add** when it appears.



Task groups allow for each reuse of a set of tasks and limits the number of places where edits need to occur.

Walkthrough cleanup

15. Click **Remove** to remove the task group from the **Test Team A** stage.
16. From the **Tasks** drop down list, select the **Development** stage. Again click **Remove** to remove the task group from the **Development** stage.
17. Click **Save** then **OK**.

Demonstration Creating and Managing Variable Groups

In this demonstration, you will investigate Variable Groups.

Note: Before starting this walkthrough, make sure you have performed the steps in the prerequisites section and the previous walkthroughs.

Steps

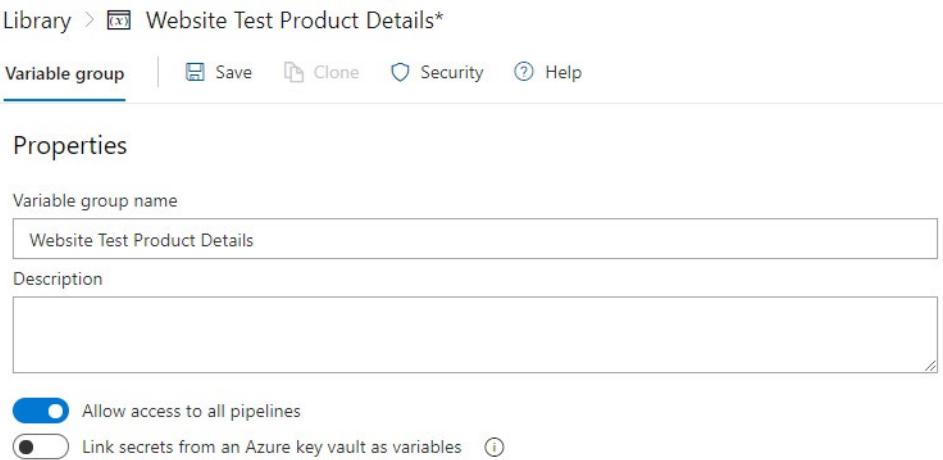
Let's now take a look at how a release pipeline can make use of predefined sets of variables, called Variable Groups.

Similar to the way we used task groups, variable groups provide a convenient way to avoid the need to redefine many variables when defining stages within pipelines, and even when working across multiple pipelines. Let's create a variable group and see how it can be used.

1. On the main menu for the **Parts Unlimited** project, click **Pipelines**, then click **Library**. There are currently no variable groups in the project.



2. Click **+ Variable group** to commence creating a variable group. Set **Variable group name** to **Website Test Product Details**.



3. In the **Variables** section, click **+Add**, then in **Name**, enter **ProductCode**, and in **Value**, enter **RED-POLOXL**.

Variables

Name ↑	Value	⋮
ProductCode	REDPOLOXL	
+ Add		

You can see an extra column that shows a lock. It allows you to have variable values that are locked and not displayed in the configuration screens. While this is often used for values like passwords, notice that there is an option to link secrets from an Azure key vault as variables. This would be a preferable option for variables that are providing credentials that need to be secured outside the project.

In this example, we are just providing details of a product that will be used in testing the website.

4. Add another variable called **Quantity** with a value of **12**.
5. Add another variable called **SalesUnit** with a value of **Each**.

Variables

Name ↑	Value	⋮
ProductCode	REDPOLOXL	
Quantity	12	
SalesUnit	Each	
+ Add		

6. Click **Save** to save the new variable group.



7. On the main menu, click **Pipelines**, then click **Releases**, then click **Edit** to return to editing the release pipeline that we have been working on. From the top menu, click **Variables**.

A screenshot of the 'Release to all environments' pipeline editor. At the top, there is a breadcrumb trail: 'All pipelines > Release to all environments'. Below the breadcrumb, there is a navigation bar with tabs: 'Pipeline' (underlined), 'Tasks', 'Variables' (with a hand cursor icon pointing at it), 'Retention', 'Options', and 'History'. Under the 'Variables' tab, there are two sections: 'Artifacts' (with '+ Add' button) and 'Stages' (with '+ Add' button). Both sections have dashed borders around them.

8. In the left-hand pane, click **Variable Groups**.

All pipelines >  Release to all environments

Pipeline Tasks Variables Retention Options History

Pipeline variables

Variable groups 

Predefined variables 

Filter by keywords

Name

Variable groups are linked to pipelines, rather than being directly added to them.

- Click **Link variable group**, then in the **Link variable group** pane, click to select the **Website Test Product Details** variable group (notice that it shows you how many variables are contained), then in the **Variable group scope**, select the **Development, Test Team A, and Test Team B** stages.

Link variable group 

Search

Website Test Product Details (3)

Variable group scope

Release

Stages

Development (+2) 

Link

We need the test product for development and during testing but we do not need it in production. If it was needed in all stages, we would have chosen **Release** for the Variable group scope instead.

10. Click **Link** to complete the link.

The screenshot shows the 'Variables' tab in the Azure DevOps interface. A variable group named 'Website Test Product Details' is selected. Below the table, there are two buttons: 'Link variable group' and 'Manage variable groups'.

Name	Value
Website Test Product Details (3)	Scopes: Development,Test Team A,Test Team B

The variables contained in the variable group are now available for use within all stages except Production, just the same way as any other variable.

Integrate Secrets with the release pipeline

Secrets are needed

When you deploy your applications to a target environment, there are almost always secrets involved.

- Secrets to access the target environment (servers, storage accounts)
- Secrets to access resources (connection strings, tokens, username/passwords)
- Secrets that your application uses (config files)

When your software is packaged as an artifact and moves through the different stages, secrets need to be inserted during the execution of the pipeline. Mainly because secrets (should) differ between stages and environments but, just as important, secrets should NEVER be part of your source control repository.

Secrets in your release pipelines

There are different ways to deal with secrets in your pipelines. We will walk through some options.

Using Service Connections

A Service Connection is a securely stored configuration of a connection to a server or service. By using this Service Connection, the pipeline tasks can execute securely against this endpoint. This way now credentials or secrets are needed as part of the release pipeline.

For more information, see [Service Connections³⁴](#).

Using secret variables

A straightforward and convenient way to add secrets to your pipeline is the use of secret variables. Earlier in this chapter we covered variables. By making the variable secret, it is hidden from view in all log files and unrecoverable.

Secret variables are often used to store secrets, connection strings and are used as replacement values in the pipeline. By creating placeholders in a target file, and replacing the placeholder with the real value in the pipeline, you create a secret free repository.

For more information, see [Define variables³⁵](#).

Storing secrets in a key vault

Another option of securing passwords and secrets is using a KeyVault. This is an excellent option because it allows you to keep the secrets outside of the pipeline and be able to retrieve them in another way (if you have the appropriate rights).

Retrieve with a variable group

To make use of this option within your build and release pipelines you need to create a variable group that refers to this KeyVault.

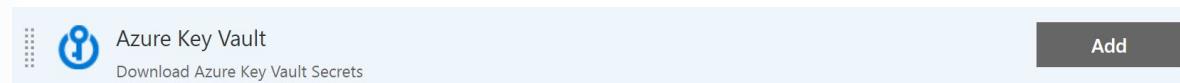
³⁴ <https://docs.microsoft.com/en-us/azure/devops/pipelines/library/service-endpoints?view=vsts>

³⁵ <https://docs.microsoft.com/en-us/azure/devops/pipelines/process/variables?view=vsts>

For more information, see [Add & use variable groups³⁶](#).

Accessing Keyvault from within the pipeline

You can also access Keyvault variables without a variable group by using the dedicated build task.



Demonstration Setting Up Pipeline Secrets

In this demonstration, you will investigate Pipeline Secrets.

Note: Before starting this walkthrough, make sure you have performed the steps in the prerequisites section and the previous walkthroughs.

Steps

Let's now take a look at how a release pipeline can make use of the secrets that it needs during deployment.

In the walkthrough on variable groups, you saw that variable groups can work in conjunction with Azure Key Vault.

1. In the Azure Portal, in the key vault (that was set up in the pre-requisites section) properties, in the **Secrets** section, create the following secrets using the **+Generate/Import** option with an **Upload option of Manual**.

A screenshot of the Azure Key Vault 'Secrets' page. The left sidebar shows 'walkthroughkeyvault - Secrets' and 'Key vault'. The main area shows a message: 'The secret 'database-password' has been successfully created.' Below this, there is a table with two rows:

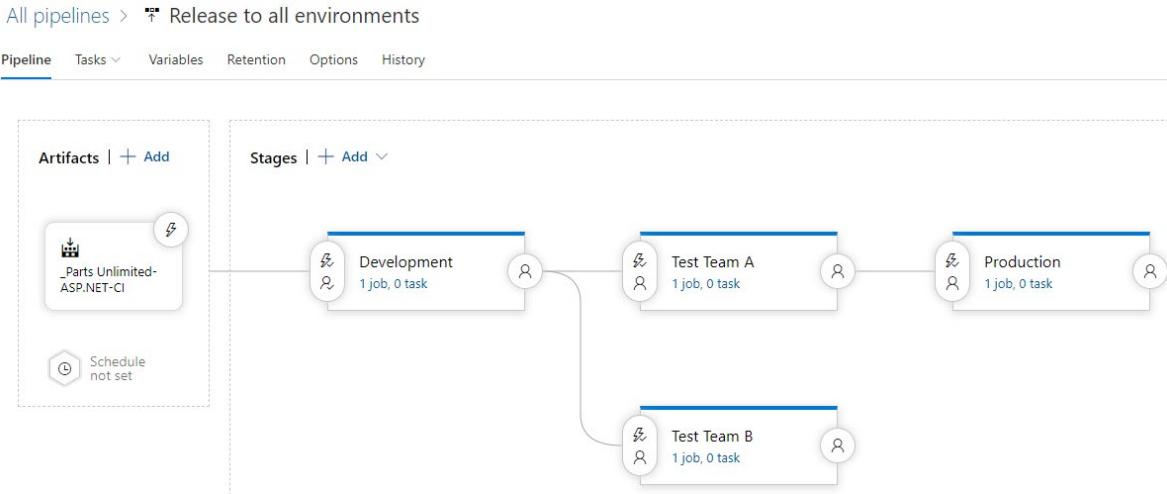
NAME	TYPE	STATUS
database-password		✓ Enabled
database-login		✓ Enabled

Adding secrets by using an Azure Key Vault task in a stage

There are several ways that secrets from an Azure Key Vault can be used. The first option is to add an Azure Key Vault task to a stage.

2. In Azure DevOps, in the main menu for the **Parts Unlimited** project, click **Pipelines**, then click **Releases**, then click **Edit** to open the editing screen for the release pipeline that was created in earlier walkthroughs.

³⁶ <https://docs.microsoft.com/en-us/vsts/build-release/concepts/library/variable-groups?view=vsts#link-secrets-from-an-azure-key-vault-as-variables>



- From the **Tasks** drop down list, click **Development** to open the tasks editor for the Development stage.

All pipelines > Release to all environments

Pipeline Tasks Variables Retention Options History

Development
Deployment process

Agent job +

No tasks have been defined as yet.

- Click the + sign to the right hand side of **Agent job** to add a new task. In the **Search** box, type **vault**.

Add tasks | ⟳ Refresh

🔍 vault ✖

⋮ **Azure Key Vault**
Download Azure Key Vault secrets

- Hover over the **Azure Key Vault** task and when **Add** appears, click it, then in the task list, click the **Azure Key Vault** task to open its settings.

The screenshot shows the 'Azure Key Vault' task configuration page. At the top right are 'View YAML' and 'Remove' buttons. Below them is a dropdown for 'Task version' set to '1.*'. The main configuration area includes fields for 'Display name' (set to 'Azure Key Vault'), 'Azure subscription' (selected 'Manage'), 'Key vault' (selected 'Manage'), and 'Secrets filter' (set to '*'). There are also 'Control Options' and 'Output Variables' sections at the bottom.

We can use the same service connection that was used in earlier walkthroughs.

- Set **Display name** to the name of your key vault, and from the **Azure subscription** drop down list, select **ARM Service Connection**. In the **Key vault** drop down list, select your key vault.

This screenshot shows the same configuration page as above, but with specific values entered: 'Display name' is 'walkthroughkeyvault', 'Azure subscription' is 'ARM Service Connection', 'Key vault' is 'walkthroughkeyvault', and 'Secrets filter' is '*'.

The **Secrets filter** can be used to define which secrets are required for the project. The default is to bring in all secrets in the vault. It will be more secure to restrict the secrets to just those that are needed.

- In the **Secrets filter**, enter **database-login, database-password**.

This screenshot shows the 'Secrets filter' field with the value 'database-login, database-password' entered.

- Click **Save** then **OK** to save your work.

The secrets can now be used like any other variables. You can reference them as \$(database-login) and \$(database-password) when configuring tasks.

Adding secrets by using variable groups

Another way to configure Azure Key Vault secrets is by linking them to variable groups.

9. Click **Remove** to remove the Azure Key Vault task, then click **Save** and **OK**.
10. In the main menu, click **Pipelines**, then **Library**. The existing variable group is shown.

The screenshot shows the 'Variable groups' section of the Library page. It displays a single item: 'Website Test Product Details' created 'an hour ago'. The interface includes tabs for 'Variable groups' and 'Secure files', and buttons for '+ Variable group', 'Security', and 'Help'.

11. Click **+Variable group** to add a new variable group.

The screenshot shows the 'Properties' page for a variable group named 'Database Credentials'. It includes fields for 'Variable group name' (set to 'Database Credentials'), 'Description' (empty), and two toggle buttons: 'Allow access to all pipelines' (selected) and 'Link secrets from an Azure key vault as variables' (not selected). Navigation links include 'Variable group', 'Save', 'Clone', 'Security', and 'Help'.

12. Click **Link secrets from an Azure key vault as variables**. In the **Azure subscription** drop down list, click **ARM Service Connection**, and from the **Key vault name** drop down list, select your key vault.

The screenshot shows the 'Key vault name' dropdown menu with 'walkthroughkeyvault' selected. Below it are 'Manage' and 'Authorize' buttons, and a note about authorizing the service connection.

ⓘ The specified Azure service connection needs to have "Get, List" secret management permissions on the selected key vault. Click "Authorize" to enable Azure Pipelines to set these permissions or manage secret permissions in the Azure portal.

Note the warning that appears. Any service principal that needs to list secrets or get their values, needs to have been permitted to do so, by creating an access policy. Azure DevOps is offering to configure this for you.

13. Click **Authorize** to create the required access policy.

Note: you will be required to log on to Azure to perform this action

The warning should then disappear.

Azure subscription * | Manage 🔍

ARM Service Connection ▾ ⏪

Scoped to resource group 'SQLDEMO'

Key vault name * Manage 🔍

walkthroughkeyvault ▾ ⏪

Variables

Delete	Secret name	Content type	Status	Expiration date
+ Add				

14. Click **+Add** to start to add the secrets.

Choose secrets

X

Choose secrets to be included in this variable group

Sel...	Secret name	Content type	Status	Expiration date
<input type="checkbox"/>	database-login		Enabled	Never
<input type="checkbox"/>	database-password		Enabled	Never

Ok Cancel

15. Click to select both database-login and database-password secrets, then click **Ok**.

The screenshot shows the 'Variables' section of the Azure Key Vault interface. At the top, there are dropdown menus for 'Azure subscription' (set to 'ARM Service Connection') and 'Key vault name' (set to 'walkthroughkeyvault'). Below these are two secrets listed in a table:

Secret name	Content type	Status	Expiration date
database-login		Enabled	Never
database-password		Enabled	Never

A blue '+ Add' button is located at the bottom left of the table.

16. Click **Save** to save the variable group.
17. In the main menu, click **Pipelines**, then **Releases**, then **Edit** to return to editing the release pipeline.
Click **Variables**, then **Variable groups**, then **Link variable group**.
18. In the **Link variable group** pane, click to select **Database Credentials**, then from the **Stages** drop down list, select **Production**.

The screenshot shows a user interface for linking a variable group. At the top left is a button labeled "Link variable group". To its right is a search bar with a magnifying glass icon and the word "Search". Below these are two sections: "Database Credentials (2)" and "Website Test Product Details (3)". The "Database Credentials" section is expanded, showing two entries: "database-login" and "database-password". The "Website Test Product Details" section is collapsed.

Variable group scope

Release

Stages

Production



Link

19. Click **Link** to link the variable group to the Production stage of the pipeline. Click the drop down beside **Database Credentials** so that you can see the variables that are now present.

Name	Value
Scopes: Production	
Database Credentials (2)	
database-login	*****
database-password	*****

[Link variable group](#) | [Manage variable groups](#)

20. Click **Save** then **OK**.

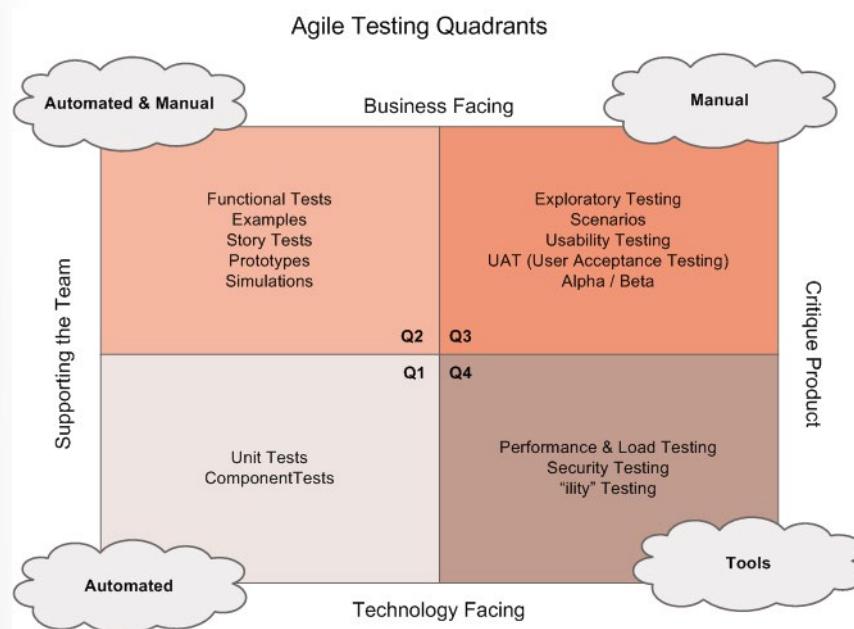
An important advantage of using variable groups to import Azure Key Vault secrets, over adding the Azure Key Vault task to a stage, is that they can be scoped to more than one stage in the pipeline, and linked to more than one pipeline.

Configure Automated Integration and Functional Test Automation

Introduction

The first thing that comes to mind when talking about Continuous Delivery is that everything needs to be automated. Otherwise, you cannot deploy multiple times a day. But how to deal with testing then? Many companies still have a broad suite of manual tests that need to run before delivering to production. Somehow these tests need to run every time a new release is created.

Instead of automating all your manual tests into automated UI tests, you need to rethink your testing strategy. As Lisa Crispin describes in her book Agile Testing, you can divide your tests into multiple categories.



>source: <http://lisacrispin.com/2011/11/08/using-the-agile-testing-quadrants/>

We can make four quadrants where each side of the square defines what we are targeting with our tests.

- Business facing, meaning the tests are more functional and most of the time executed by end users of the system or by specialized testers that know the problem domain very well.
- Supporting the Team, means it helps a development team to get constant feedback on the product so they can find bugs fast and deliver a product with quality build in
- Technology facing, means the tests are rather technical and non-meaningful to business people. They are typical tests written and executed by the developers in a development team.
- Critique Product, are tests that are there to validate the workings of a product on its functional and non-functional requirements.

Now we can place different test types we see in the different quadrants.

e.g., we can put functional tests, Story tests, prototypes and simulations in the first quadrant. These tests

are there to support the team in delivering the right functionality and are business facing since they are more functional.

In quadrant two we can place tests like exploratory tests, Usability tests, acceptance tests, etc.

In quadrant three we place tests like Unit tests, Component tests, and System or integration tests.

In quadrant four we place Performance tests, load tests, security tests, and any other non-functional requirements test.

Now if you look at these quadrants, you can see that specific tests are easy to automate or are automated by nature. These tests are in quadrant 3 and 4

Tests that are automatable but most of the time not automated by nature are the tests in quadrant 1

Tests that are the hardest to automate are in quadrant 2

What we also see is that the tests that cannot be automated or are hard to automate are tests that can be executed in an earlier phase and not after release. This is what we call shift-left where we move the testing process more towards the development cycle.

We need to automate as many tests as possible. And we need to test as soon as possible. A few of the principles we can use are:

- Tests should be written at the lowest level possible
- Write once, run anywhere including production system
- Product is designed for testability
- Test code is product code, only reliable tests survive
- Test ownership follows product ownership

By testing at the lowest level possible, you will find that you have a large number of tests that do not require infrastructure or applications to be deployed. For the tests that need an app or infrastructure, we can use the pipeline to execute them.

To execute tests within the pipeline, we can run scripts or use tools that execute certain types of tests. On many occasions, these are external tools that you execute from the pipeline, like Owasp ZAP, SpecFlow, or Selenium. In other occasions, you can use test functionality from a platform like Azure. For example Availability or Load Tests that are executed from within the cloud platform.

When you want to write your own automated tests, choose the language that resembles the language from your code. In most cases, the developers that write the application should also write the test, so it makes sense to use the same language. For example, write tests for your .Net application in .Net, and write tests for your Angular application in Angular.

To execute Unit Tests or other low-level tests that do not need a deployed application or infrastructure, the build and release agent can handle this. When you need to execute tests with a UI or other specialized functionality, you need to have a Test agent that can run the test and report the results back. Installation of the test agent then needs to be done up front, or as part of the execution of your pipeline.

Setting up Test Infrastructure

Installing the test agent

To execute tests from within the pipeline, you need an agent that can run these tests. An agent can run on another machine and can be installed as part of the pipeline.

For more information, see **Run Your Tests³⁷**.

Running in the cloud

Hosted agents run the test agent, which enable you to run tests "in the cloud."

For Load Testing, you can also run from the cloud. This enables you to create a large set of users and test your application without having to set up your infrastructure.

For more information, see **Changes to load test functionality in Visual Studio and cloud load testing in Azure DevOps³⁸**.

Setting up and Running Availability Tests

After you have deployed your web app or website to any server, you can set up tests to monitor its availability and responsiveness. It is useful to check if your application is still running and gives a healthy response.

Some applications have specific Health endpoints that can be checked by an automated process. The Health endpoint can be a simple HTTP status or a complex computation that uses and consumes crucial parts of your application.

For example, You can create a Health endpoint that queries the database. This way, you can check that your application is still accessible, but also the database connection is verified.

You can create your own framework to create availability tests (ping test) or use a platform that can do this for you. Azure has the functionality to create Availability tests. You can use these tests in the pipeline and as release gates.

In Azure, you can set up availability tests for any HTTP or HTTPS endpoint that is accessible from the public internet. You don't have to add anything to the website you're testing. It doesn't even have to be your site: you could test a REST API service on which you depend.

There are two types of availability tests:

- URL ping test: a simple test that you can create in the Azure portal. You can check an URL and check the response and status code of the response
- Multi-step web test. These are a number of HTTP calls that are executed in sequence.

For more information, see also:

- **Creating an Application Insights Web Test and Alert Programmatically³⁹**
- **Monitor the availability of any website⁴⁰**

³⁷ <https://docs.microsoft.com/en-us/azure/devops/pipelines/test/set-up-continuous-test-environments-builds?view=vsts>

³⁸ <https://docs.microsoft.com/en-us/azure/devops/test/load-test/overview?view=vsts>

³⁹ <https://azure.microsoft.com/nl-nl/blog/creating-a-web-test-alert-programmatically-with-application-insights/>

⁴⁰ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-monitor-web-app-availability>

Automate Inspection of Health

Automate Inspection of Health

Inspection of the release pipeline and release is something you should consider from the start. When you run multiple deployments a day, you want to stay informed. You want to know whether a release passed or failed, you want to know the quality of the release, you want to know details about the release and how it performed, you want to stop releases when you detect something suspicious, and you want to visualize some of these things on a dashboard.

There are a few different things you can do to stay informed about your release pipeline in an automated fashion. In the following chapters, we will dive a bit deeper on these.

Release gates

Release gates allow automatic collection of health signals from external services and then promote the release when all the signals are successful at the same time or stop the deployment on timeout. Typically, gates are used in connection with incident management, problem management, change management, monitoring, and external approval systems.

Events, subscriptions, and notifications

Events are raised when certain actions occur, like when a release is started or a build completed.

A notification subscription is associated with a supported event type. The subscription ensures you get notified when a certain event occurs.

Notifications are usually emails that you receive when an event occurs to which you are subscribed.

Service Hooks

Service hooks enable you to perform tasks on other services when events happen in your Azure DevOps Services projects. For example, create a card in Trello when a work item is created or send a push notification to your team's Slack when a build fails. Service hooks can also be used in custom apps and services as a more efficient way to drive activities when events happen in your projects.

Reporting

Reporting is the most static approach when it comes to inspection, but in many cases also the most evident. Creating a dashboard which shows the status of your build and releases combined with team specific information is in many cases a valuable asset to get insights.

Read more at [About dashboards, charts, reports, & widgets⁴¹](#).

Release gates

We talked about release gates earlier in the course. We can distinguish between 2 different types uses of release gates.

- As a quality gate. Check if the level of quality is still to decide whether or not the release can continue

⁴¹ <https://docs.microsoft.com/en-us/azure/devops/report/dashboards/overview?view=vsts>

- As an automatic approval. Check if certain things are done, before signing off to the next stage. This might in some cases be the same thing as a quality gate.

For example, In many organizations, there are so-called dependency meetings. This is a planning session where the release schedule of dependent components is discussed. Think of downtime of a database server or an update of an API. This takes a lot of time and effort, and the only thing that is needed is a signal if the release can proceed. Instead of having this meeting you can also create a mechanism where people press a button on a form when the release cannot advance. When the release starts, it checks the state of the gate by calling an API. If the "gate" is open, we can continue. Otherwise, we stop the release.

By using scripts and API's, you can create your own release gates instead of a manual approval. Or at least extending your manual approval. Other scenarios for automatic approvals are for example.

- Incident and issues management. Ensure the required status for work items, incidents, and issues. For example, ensure that deployment only occurs if no bugs exist.
- Notify users such as legal approval departments, auditors, or IT managers about a deployment by integrating with approval collaboration systems such as Microsoft Teams or Slack, and waiting for the approval to complete.
- Quality validation. Query metrics from tests on the build artifacts such as pass rate or code coverage and only deploy if they are within required thresholds.
- Security scan on artifacts. Ensure security scans such as anti-virus checking, code signing, and policy checking for build artifacts have completed. A gate might initiate the scan and wait for it to complete, or check for completion.
- User experience relative to baseline. Using product telemetry, ensure the user experience hasn't regressed from the baseline state. The experience level before the deployment could be considered a baseline.
- Change management. Wait for change management procedures in a system such as ServiceNow complete before the deployment occurs.
- Infrastructure health. Execute monitoring and validate the infrastructure against compliance rules after deployment, or wait for proper resource utilization and a positive security report.

In short, approvals and gates give you additional control over the start and completion of the deployment pipeline. They can usually be set up as a pre-deployment and post-deployment condition, that can include waiting for users to approve or reject deployments manually, and checking with other automated systems until specific requirements are verified. Besides, you can configure a manual intervention to pause the deployment pipeline and prompt users to carry out manual tasks, then resume or reject the deployment.

Release gates in Azure DevOps

A release gate is a concept that ships out of the Box in Azure DevOps. However, another tool can implement the same mechanism under a different name. When we take a look at the different types of release gate that ship out of the Box, you can already see that this can easily apply as well in any other tool.

You can configure a release gate before or after a stage and configure it in such a way that it evaluates the gate multiple times. For example, it can check the performance of your system, and when the gate fails, it can check again after a few minutes when the system warmed up.

For some examples around evaluation, read this carefully:

- **Gate evaluation flow examples⁴²**

Please take a look at this video to see how release gates work in practice. The layout of Azure DevOps differs a bit since there is a new, but the functionality stays the same.

- **Deploy quicker and safer with new greenlighting capabilities in VSTS⁴³**

To learn more about release gates, see also:

- **Release deployment control using gates⁴⁴**
- **Invoke Azure Function task⁴⁵**
- **Query Azure Monitor Alerts task⁴⁶**
- **Query Work Items task⁴⁷**
- **Publish To Azure Service Bus task⁴⁸**
- **Invoke REST API task⁴⁹**
- **Video: Taking control over your releases⁵⁰**

Events, Subscriptions, and Notifications

One of the first requests many people have when working in a system that performs asynchronous actions is to have the ability to get notifications or alerts. Why? Because they do not want to open the application, log in and see if things changed over and over again.

The ability to receive Alerts and notifications is a powerful mechanism to get notified about certain events in your system at the moment they happen.

For example, when a build takes a while to complete, probably you do not want to stare to the screen until it has finished. But, you want to know when it does.

Getting an email or another kind of notification instead is very powerful and convenient. Another example is a system that needs to be monitored. You want to get notified by the system in real time. By implementing a successful alert mechanism, you can use alerts to react to situations before anybody is bothered by it proactively.

However, When you define alerts, you need to be careful. When you get alerts for every single event that happens in the system, your mailbox will quickly be flooded with a lot of alerts. The more alerts you get that are not relevant, the higher the chance that people will never look at the alerts and notifications and will miss out on the important ones.

When defining alerts or notification, you need to think about the target audience. Who needs to react to the alerts? Do not send notifications to people for information only. They will stop looking at it very quickly.

⁴² <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/approvals/gates?view=vsts#gate-evaluation-flow-examples>

⁴³ <https://channel9.msdn.com/Events/Connect/2017/T181>

⁴⁴ <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/approvals/gates?view=vsts>

⁴⁵ <https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/utility/azure-function?view=vsts>

⁴⁶ <https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/utility/azure-monitor?view=vsts>

⁴⁷ <https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/utility/work-item-query?view=vsts>

⁴⁸ <https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/utility/publish-to-azure-service-bus?view=vsts>

⁴⁹ <https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/utility/http-rest-api?view=vsts>

⁵⁰ https://youtu.be/7WLcqwhTZ_4

Another thing to consider when defining alerts is the mechanism to deliver them. Do you want to send an email, or do you want to send a message in the Slack for your team? Or do you want to call another system to perform a particular action?

Within Azure DevOps, there are multiple ways to define your alerts. By using query and filter mechanisms, you can filter out specific alerts. For example, you only want to get notified for failed releases and not for successful ones.

Almost every action in the system raises an event to which you can subscribe to. A subscription is personal or for your whole team. When you have made a subscription, you can then select how you want the notification to be delivered.

For more information, see also:

- [About notifications⁵¹](#)
- [Events, subscriptions, and notifications⁵²](#)

Service Hooks

Service hooks enable you to perform tasks on other services when events happen in your Azure DevOps Services projects. For example, create a card in Trello when a work item is created or send a push notification to your team's mobile devices when a build fails. Service hooks can also be used in custom apps and services as a more efficient way to drive activities when events happen in your projects.

In Azure DevOps these Service Hooks are supported Out of the Box

Build and release	Collaborate	Customer support	Plan and track	Integrate
AppVeyor	Campfire	UserVoice	Trello	Azure Service Bus
Bamboo	Flowdock	Zendesk		Azure Storage
Jenkins	HipChat			Web Hooks
MyGet	Hubot			Zapier
Slack				

To learn more about service hooks and how to use and create them, read [Service Hooks in Azure DevOps⁵³](#).

Demonstration Setting Up Service Hooks to Monitor the Pipeline

In this demonstration, you will investigate Service Hooks.

Note: Before starting this walkthrough, make sure you have performed the steps in the prerequisites section and the previous walkthroughs.

Steps

Let's now take a look at how a release pipeline can communicate with other services by using service hooks.

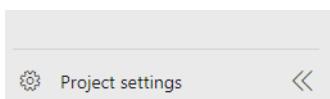
⁵¹ <https://docs.microsoft.com/en-us/azure/devops/notifications/index?view=vsts>

⁵² <https://docs.microsoft.com/en-us/azure/devops/notifications/concepts-events-and-notifications?view=vsts>

⁵³ <https://docs.microsoft.com/en-us/azure/devops/service-hooks/overview?view=vsts>

Azure DevOps can be integrated with a wide variety of other applications. It has built in support for many applications, and generic hooks for working with other applications. Let's take a look.

1. Below the main menu for the **Parts Unlimited** project, click **Project settings**.



2. In the **Project settings** menu, click **Service hooks**.

Project Settings

Parts Unlimited

- General
- Overview
- Teams
- Security
- Notifications
- Service hooks**
- Dashboards

Service Hooks

Integrate with your favorite services by notifying them when events happen in your project.

+ Create subscription

3. Click **+Create subscription**.

NEW SERVICE HOOKS SUBSCRIPTION x

Service

Select a service to integrate with. Discover more integrations

Service	Description
App Center	App Center allows you to automate the lifecycle of your iOS, Android, Windows, and macOS apps. Connect your repo and within minutes build in the cloud, test on thousands of real devices, distribute to beta testers and app stores, and monitor real-world usage with crash and analytics data, all in one place.
AppVeyor	AppVeyor
Azuqua	Azuqua
Azure App Service	Azure App Service
Azure Service Bus	Azure Service Bus
Azure Storage	Azure Storage
Bamboo	Bamboo
Campfire	Campfire
Flowdock	Flowdock
Grafana	Grafana
HipChat	HipChat
HockeyApp	HockeyApp
Jenkins	Jenkins
Microsoft Teams	Microsoft Teams

App Center

App Center allows you to automate the lifecycle of your iOS, Android, Windows, and macOS apps. Connect your repo and within minutes build in the cloud, test on thousands of real devices, distribute to beta testers and app stores, and monitor real-world usage with crash and analytics data, all in one place.

Supported events:

Work item updated

Supported actions:

Send notification

[Learn more about this service](#)

Previous Next Test Finish Cancel

By using service hooks, we can notify other applications that an event has occurred within Azure DevOps. We could also send a message to a team in **Microsoft Teams** or **Slack**. We could also trigger an action in **Bamboo** or **Jenkins**.

4. Scroll to the bottom of the list of applications and click on **Web Hooks**.

The screenshot shows a dialog box titled "NEW SERVICE HOOKS SUBSCRIPTION". On the left, a vertical list of services includes Gratana, HipChat, HockeyApp, Jenkins, Microsoft Teams, MyGet, Office 365, Slack, Trello, UserVoice, Web Hooks, Workplace Messaging Apps, Zapier, and Zendesk. The "Web Hooks" option is highlighted with a blue background. To the right of the list, detailed information about "Web Hooks" is displayed: "Provides event communication via HTTP", "Supported events: All events", "Supported actions: Post via HTTP", and a link to "Learn more about this service". At the bottom of the dialog are buttons for "Previous", "Next", "Test", "Finish", and "Cancel".

If the application that you want to communicate with isn't in the list of available application hooks, you can almost always use the **Web Hooks** option as a generic way to communicate. It allows you to make an HTTP POST when an event occurs. So, if for example, you wanted to call an Azure Function or an Azure Logic App, you could use this option.

To demonstrate the basic process for calling web hooks, we'll write a message into a queue in the Azure Storage account that we have been using.

5. From the list of available applications, click **Azure Storage**.

The screenshot shows a dialog box titled "NEW SERVICE HOOKS SUBSCRIPTION". On the left, a sidebar lists various services: App Center, AppVeyor, Azuqua, Azure App Service, Azure Service Bus, **Azure Storage**, Bamboo, Campfire, Flowdock, Grafana, HipChat, HockeyApp, Jenkins, and Microsoft Teams. The "Azure Storage" item is selected and highlighted with a blue background. To the right of the sidebar, detailed information about "Azure Storage" is displayed, including its definition, supported events (All events), and supported actions (Insert a message in a Storage Queue). Below this information is a link to "Learn more about this service". At the bottom of the dialog are buttons for "Previous", "Next", "Test", "Finish", and "Cancel".

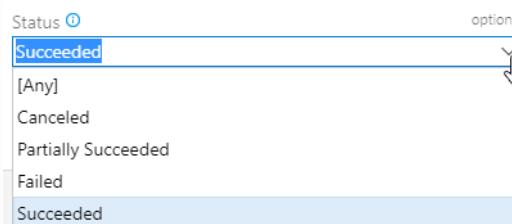
6. Click **Next**. In the **Trigger** page, we determine which event causes the service hook to be called. Click the drop down for **Trigger on this type of event** to see the available event types.

Trigger

Select an event to trigger on and configure any filters.

The screenshot shows a dropdown menu titled "Trigger on this type of event". The options listed are: Build completed, Build completed, Code pushed, Pull request commented on, Pull request created, Pull request merge attempted, Pull request updated, Release abandoned, and Release created. The first two items, "Build completed", are highlighted with a blue background, indicating they are currently selected.

7. Ensure that **Release deployment completed** is selected, then in the **Release pipeline name** select **Release to all environments**. For **Stage**, select **Production**. Drop down the list for **Status** and note the available options.



8. Ensure that **Succeeded** is selected, then click **Next**.

Trigger

Select an event to trigger on and configure any filters.

Trigger on this type of event

Release deployment completed

FILTERS

Release pipeline name optional

Release to all environments ✓

Stage name optional

Production ✓

Status optional

Succeeded ✓

Previous Next Test Finish Cancel

9. In the **Action** page, enter the name of your Azure storage account.

10. Open the Azure Portal, and from the settings for the storage account, in the **Access keys** section, copy the value for **Key**.

Home > SQLDEMO > devopsoutput - Access keys

devopsoutput - Access keys

Storage account

Search (Ctrl+ /)

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Data transfer

Events

Storage Explorer (preview)

Settings

Access keys

Use access keys to authenticate your applications when making requests to this Azure storage account. Store your access keys securely - for example, using Azure Key Vault - and don't share them. We recommend regenerating your access keys regularly. You are provided two access keys so that you can maintain connections using one key while regenerating the other.

When you regenerate your access keys, you must update any Azure resources and applications that access this storage account to use the new keys. This action will not interrupt access to disks from your virtual machines. [Learn more](#)

Storage account name

devopsoutput

key1

Key

ApxxJ9Dvs2dguErzh/vY6...

Connection string

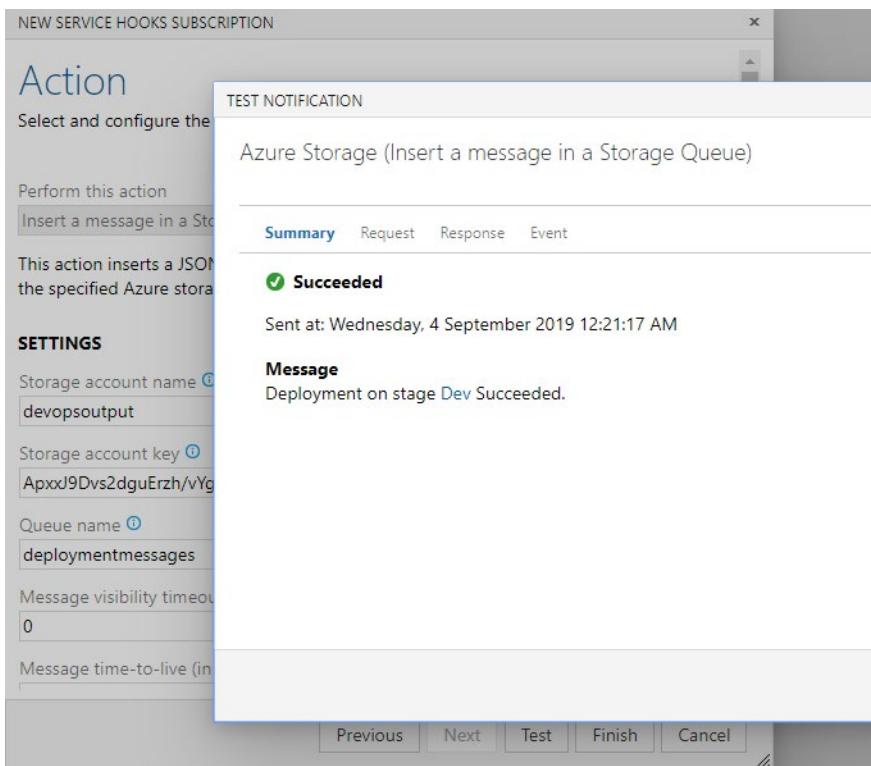
DefaultEndpointsProtocol=https;AccountName=devopsoutput;AccountKey=ApxxJ9Dvs2dguErzh/vY6...

11. Back in the **Action** page in Azure DevOps, paste in the key.

SETTINGS

Storage account name ⓘ	required
devopsoutput	✓
Storage account key ⓘ	required
ApxxJ9Dvs2dguErzh/vYgskz	✓

12. For **Queue name** enter **deploymentmessages**, then click **Test**.



13. Make sure that the test succeeded, then click **Close**, and on the **Action** page, click **Finish**.

Service Hooks

Integrate with your favorite services by notifying them when events happen in your project.

+		edit	refresh	x		History
Consumer ↑		Event ↑		Action		
Azure Storage	...	Release deployment c...	Release Release to all environm...	Insert a message in a S...	Account	

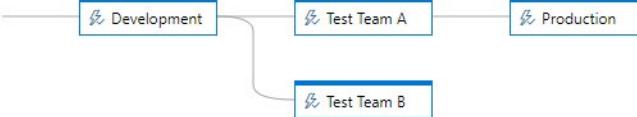
Create a release to test the service hook

Now that you have successfully added the service hook, it's time to test it.

14. From the main menu of the **Parts Unlimited** project, click **Pipelines**, then click **Releases**, then click **Create release**, and in the **Create a new release** pane, enter **Test the queue service hook** for **Release description**, and click **Create**.

Create a new release X

Release to all environments



Pipeline ^
Click on a stage to change its trigger from automated to manual.

Stages for a trigger change from automated to manual. i

Artifacts ^
Select the version for the artifact sources for this release

Source alias	Version
_Parts Unlimited-ASP.NET-CI	20190901.2

Release description

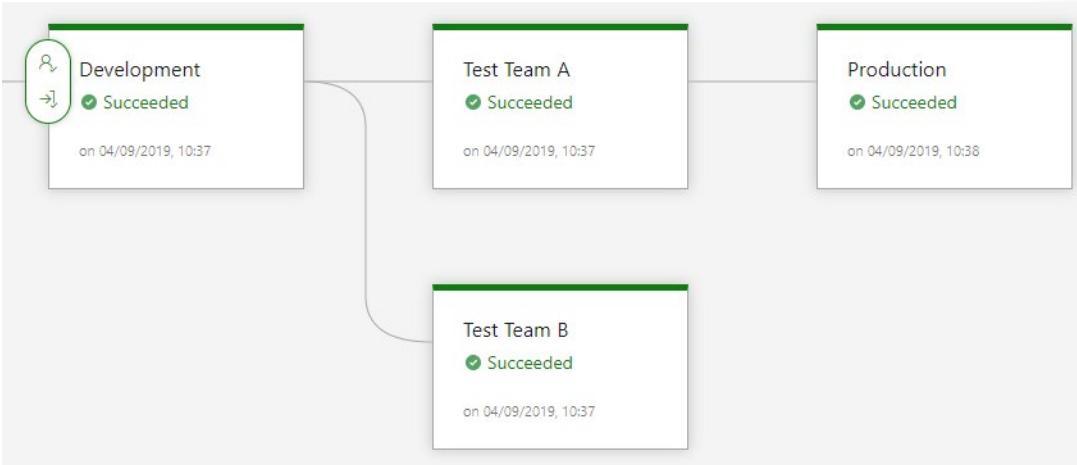
Test the queue service hook

Create Cancel

15. Click to view the release details.



16. If the release is waiting for approval, click to approve it and wait for the release to complete successfully.



Check the queue contents

17. In the **Azure Portal**, in the blade for the storage account, click **Queues** from the **Queue service** section.

QUEUE	URL
deploymentmessages	https://devopsoutput.queue.core.windows.net/deploymentmessages

18. Click to open the **deploymentmessages** queue.

ID	MESSAGE TEXT	INSERTION TIME
b00d5fc0-a8ed...	{"id":"53b18aab-0f53-4bc6-9394-2bb2283c438b","eventType":"ms.vss-rel...	9/4/2019, 10:21:17 AM
0ce1acee-2002-...	{"id":"48af54d0-a806-4744-9d1f-ca5f262bbcc5","eventType":"ms.vss-relea...	9/4/2019, 10:38:21 AM

Note: if you have run multiple releases, you might have multiple messages

19. Click the latest message (usually the bottom of the list) to open it and review the message properties, then close the **Message properties** pane.

Message properties

ID

0ce1acee-2002-4950-889d-9677518271d6

MESSAGE BODY

```
{"id":"48af54d0-a806-4744-9d1f-ca5f262bbcc5","eventType":"ms.vss-release.deployment-completed-event","publisherId":"rm","message": {"text":"Deployment of release Release-4 on stage Production succeeded.", "html":"Deployment on stage <a href='https://greglowdevopslab.visualstudio.com/Parts%20L_a=environment-summary&definitionId=2&definitionEnvironmentId=7'>Proc succeeded.", "markdown":"Deployment on stage [Production] (https://greglowdevopslab.visualstudio.com/Parts%20Unlimi\_a=environment-summary&definitionId=2&definitionEnvironmentId=7)", "detailedMessage": {"text": "Deployment of release Release-4 on stage Production succeeded. Time to deploy: 00:00:26.", "html": "Deployment on stage <a
```

You have successfully integrated this message queue with your Azure DevOps release pipeline.

Labs

Configuring Pipelines as Code with YAML

Overview

One of the key ideas of DevOps is infrastructure-as-code having the infrastructure for your delivery/deployment pipeline expressed in code—just as the products that flow it. Azure Pipelines allows you to define your pipelines using YAML (code). However, pipeline as code doesn't simply mean executing a script that's stored in source control. Codified pipelines use their own programming model to simplify the setup and maximize reuse. A typical microservice architecture will require many deployment pipelines that are for the most part identical. It is tedious to craft these pipelines via a user interface or SDK. Having the ability to define the pipeline along with the code helps apply all principles of code sharing, reuse, templatization and code reviews.

In this lab, **Configuring CI/CD Pipelines as Code with YAML in Azure DevOps⁵⁴**, we'll learn how to create build and deployment pipelines using YAML.

What's covered in this lab

In this lab, you will see:

- Adding a YAML build definition
- Adding continuous delivery to the YAML definition

Setting up secrets in the pipeline with Azure Key vault

Overview

You can use the Azure Key vault as a storage place for your secrets, keys, and certificates. You can benefit from this safe storage in your release pipelines by using the Azure Keyvault task or creating a variable group pointing to the Azure Keyvault.

What's covered in this lab?

This lab covers the configuration of the Azure Keyvault and the use of the secrets in the Azure Key vault in your release pipeline. We will show two different methods:

- Using the Azure KeyVault Build/Release task
- Using the Variable Group linked to an Azure Key vault.

Before you begin

Refer the **Azure DevOps Labs Getting Started⁵⁵** page to know the prerequisites for this lab.

⁵⁴ <https://www.azuredevopslabs.com/labs/azuredevops/yaml/>

⁵⁵ <https://azuredevopslabs.com/labs/vstsextend/Setup/>

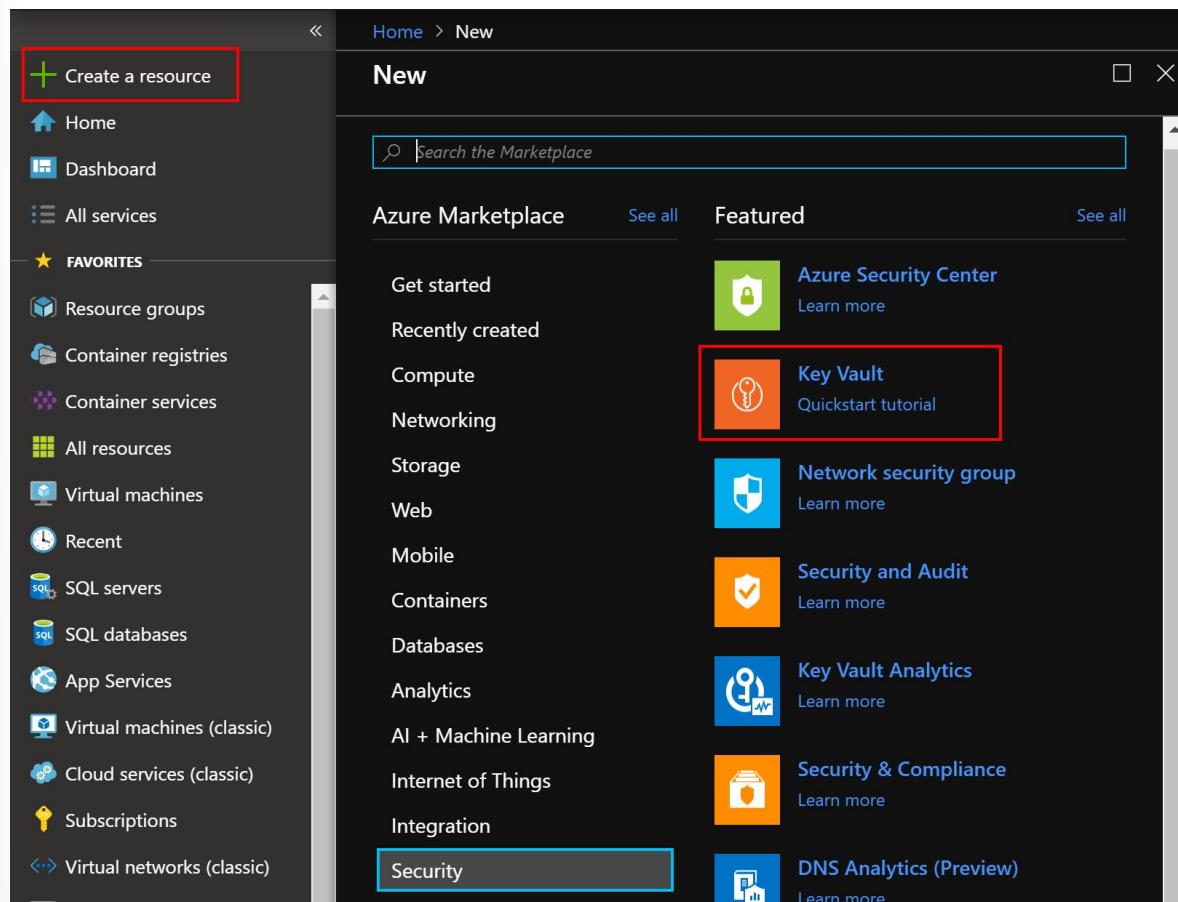
Click the **Azure DevOps Demo Generator⁵⁶** link and select the PartsUnlimited Demo project.

Setting up the Target Environment

You will create one resource group and an Azure Key vault. In the Azure Key vault, you will create two secrets.

You need to have an empty Azure DevOps project or the PartsUnlimited Demo project.

1. Go to Azure portal and click on +Create a resource and search and select Key Vault.



2. Provide a name for the Key vault, create new Resource Group or select existing one from the drop-down and click Create.
3. Once the deployment succeeds, navigate to your Resource Group to see the resources created.
4. In the Azure Key vault, navigate to **Secrets**, select **+Generate/Import**, and create a secret. Add the name **password**. The value is random and up to you and select the **create** button.

⁵⁶ <https://azuredemoprov.azurewebsites.net/Environment/CreateProject>

Exercise 1: Configure Release pipeline with Azure Key vault build/release task

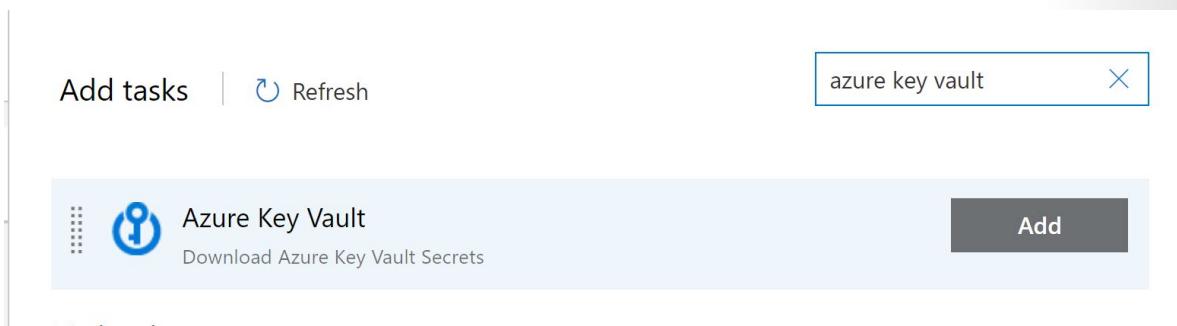
Setup Git Repo

1. Navigate to **Repos**, select the existing Repo and create a **New repository**. Call it secrets. **Initialize** the repository with a ReadMe.md file
2. Add a **New File** to the repository. Call it secrets.config.
3. Add the following lines of code to the file.

```
{  
    "username": "user123",  
    "password": "#{password}#"  
}
```

Setup the pipeline

1. Navigate to Releases under Pipelines and Create a **New Release Pipeline**. Select **Empty job**.
2. **Add an Artifact** and select the **secrets Git Repo**
3. Add tasks to **Stage 1**. Add a Task and select the **Azure Key Vault** Task.



4. Click the task, select the **Azure Subscription** and select the **Key vault**.

Azure Key Vault (i) Remove

Version 1.* ▾

Display name *

Azure Key Vault: mscd-keyvault

Azure subscription * (i) | Manage ▾

① Scoped to subscription 'Rene Azure Sponsorship'

Key vault * (i)

mscd-keyvault ▾ (↻)

Secrets filter * (i)

*

Control Options ▾

Output Variables ▾

5. Add a **Replace Tokens task**⁵⁷. If it is not yet installed, install it first.
6. Run the release. The # {password}# token has been replaced with the secret from the Azure Key vault.

Exercise 2: Configure Release pipeline with variable group

1. Navigate to **Library** under Pipelines and Create a **+Variable Group**. Name it **secret** and **Link secrets from an Azure Key vault**
2. Select the **Azure Subscription** and select the **Key vault**
3. **Add** the secret as variable and **Save** the variable group

Setup the pipeline

1. Navigate to Releases under Pipelines and Create a **New Release Pipeline**. Select **Empty job**.
2. **Add an Artifact** and select the **secrets Git Repo**
3. Add a **Replace Tokens task**⁵⁸. If it is not yet installed, install it first.
4. Go to **variables** and select **Variable Groups**. **Link variable group** to the **secret*** variable group

⁵⁷ <https://marketplace.visualstudio.com/items?itemName=qetza.replacetokens&targetId=4590486a-765d-48ea-a3f2-0772f9bcaefc>

⁵⁸ <https://marketplace.visualstudio.com/items?itemName=qetza.replacetokens&targetId=4590486a-765d-48ea-a3f2-0772f9bcaefc>

5. Run the release. The # {password} # token has been replaced with the secret from the Azure Key vault

Setting up and Running Functional Tests

Running a functional test in the pipeline

Selenium is a portable open source software-testing framework for web applications. It has the capability to operate on almost every operating system. It supports all modern browsers and multiple languages including .NET (C#), Java.

In this lab, **Automating Selenium Tests in Azure Pipelines**⁵⁹, you will learn how to execute selenium testcases on a C# web application, as part of the Azure DevOps Release pipeline.

Linking Test Cases to your pipeline

When you already have a large amount of manual tests in test cases, you can also start using the benefits of running the test cases as part of your pipeline. When you automate a test case, the automation bit can be linked to the test case.

By attaching automation to the test case, the results are stored in the test case but also as part of your release pipeline.

- **Associate automated tests with test cases**⁶⁰
- **Run automated tests from test plans**⁶¹

For additional information, see also:

- **Behavior-Driven Design with SpecFlow**⁶²
- **UI test with Selenium**⁶³
- **Getting Started with Selenium Testing in a Continuous Integration Pipeline with Visual Studio**⁶⁴

Using Azure Monitor as Release Gate

As you may be aware, a release pipeline specifies the end-to-end release process for an application to be deployed across a range of environments. Deployments to each environment are fully automated by using phases and tasks. Ideally, you do not want new updates to the applications to be exposed to all the users at the same time. It is a best practice to expose updates in a phased manner i.e. expose to a subset of users, monitor their usage and expose to other users based on the experience the initial set of users had.

Approvals and gates enable you to take control over the start and completion of the deployments in a release. With approvals, you can wait for users to manually approve or reject deployments. Using release gates, you can specify application health criteria that must be met before release is promoted to the next environment. Prior to or after any environment deployment, all the specified gates are automatically evaluated until they all pass or until they reach your defined timeout period and fail.

⁵⁹ <https://www.azuredevopslabs.com/labs/vstsextrnd/Selenium/>

⁶⁰ <https://docs.microsoft.com/en-us/azure/devops/test/associate-automated-test-with-test-case?view=vsts>

⁶¹ <https://docs.microsoft.com/en-us/azure/devops/test/run-automated-tests-from-test-hub?view=vsts>

⁶² <https://msdn.microsoft.com/en-us/magazine/dn296508.aspx>

⁶³ <https://docs.microsoft.com/en-us/azure/devops/pipelines/test/continuous-test-selenium?view=vsts>

⁶⁴ <https://blogs.msdn.microsoft.com/devops/2016/01/27/getting-started-with-selenium-testing-in-a-continuous-integration-pipeline-with-visual-studio/>

Gates can be added to an environment in the release definition from the pre-deployment conditions or the post-deployment conditions panel. Multiple gates can be added to the environment conditions to ensure all the inputs are successful for the release.

This lab covers the configuration of the deployment gates and details how to add the control to Azure pipelines. You will configure a release definition with two environments for an Azure Web App. You will deploy to the Canary environment only when there are no blocking bugs for the app and mark the Canary environment complete only when there are no active alerts in Azure Monitor (Application Insights).

Follow the instructions at [Controlling Deployments using Release Gates⁶⁵](#).

Creating a Release Dashboard

Overview

An essential part of setting up your release pipelines is able to have insights into them. By creating a Release dashboard that shows the relevant information, you can quickly see the status of your release and surrounding assets.

What's covered in this lab?

In this lab, we will cover the creation of a dashboard and some of the widgets you can use to gain insights. We will also take a look at the REST API to get information from Azure DevOps releases to use in your own applications or dashboards.

Before you begin

Refer to the [Azure DevOps Labs Getting Started⁶⁶](#) page to know the prerequisites for this lab.

Complete [Task 1 from the Azure DevOps Lab Prerequisites⁶⁷](#).

Setting up the Target Environment

You will run one build and one release to get some data in your project. Make sure you have an Azure account where you can deploy the application.

1. Navigate to **Pipelines****Releases** and configure the Service Connections of all stages in the **PartsUnlimitedE2E**. Set the deployment slot name of the Dev Stage to *Dev* and the QA stage to *Staging*
2. In your Azure DevOps Project Go to **Pipelines** and click **Builds** and **Queue** a new **PartsUnlimitedE2E** build.
3. The application is built and deployed automatically

Exercise 1: Create a Release Dashboard

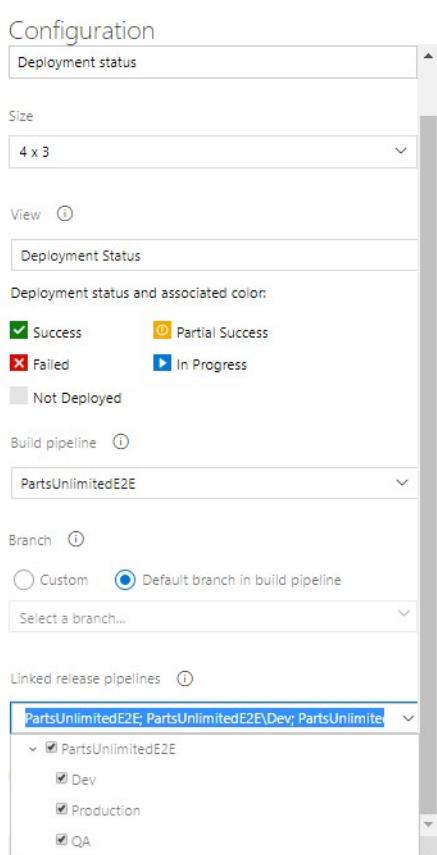
1. Navigate to **Dashboards** under Overview and Create a **+New Dashboard**. Name it **Releases**
2. Search the **Deployment Status** widget and add it to the dashboard

⁶⁵ <https://azuredevopslabs.com/labs/vstsextend/releaseagates/>

⁶⁶ <https://azuredevopslabs.com/labs/vstsextend/Setup/>

⁶⁷ <https://azuredevopslabs.com/labs/azuredevops/prereq/#task-1-configuring-the-parts-unlimited-team-project>

- Click the configure icon, Set the size to *4x3 *, select *PartsUnlimitedE2E* build and all the stages of the Linked Release Pipeline and **Save** the configuration.



- Search the **Release Pipeline Overview** widget and add it to the dashboard. Press the **Configure** icon and select the **PartsUnlimitedE2E** Release
- Search the **Build History** widget and add it to the dashboard. Press the **Configure** icon and select the **PartsUnlimitedE2E** Build
- Install the **Team Project Health widget**⁶⁸ from the marketplace
- Add the *Release Health Overview* and *Release Health Details* widget and configure them
- Add the *Build Health Overview* and *Build Health Details* widget and configure them

Exercise 2: Use the REST API to get Release Information

In this exercise, we will use the tool Postman to show the use of the REST API. You can use any tool of your choice to call the REST API.

You can download Postman at [Download Postman App](#)⁶⁹.

- Get a Personal Access Token by following the steps described at [Authenticate access with personal access tokens](#)⁷⁰.

⁶⁸ <https://marketplace.visualstudio.com/items?itemName=ms-devlabs.TeamProjectHealth>

⁶⁹ <https://www.getpostman.com/downloads/>

⁷⁰ <https://docs.microsoft.com/en-us/azure/devops/organizations/accounts/use-personal-access-tokens-to-authenticate?view=vsts>

2. Use the URL `https://vsrm.dev.azure.com/{organization}/{project}/_apis/release/definitions?api-version=5.1` to get all Release Definitions.
3. In the authorization tab, select Basic Authentication and paste the Personal Access token in the password field

The screenshot shows the Postman application interface. A GET request is being prepared to the URL `https://vsrm.dev.azure.com/azdoexam/_apis/release/definitions?api-version=5.0-preview.3`. The 'Authorization' tab is active, and 'Basic Auth' is selected. A note in the interface cautions about the security of sensitive data like Personal Access Tokens. The password field is redacted with dots. A 'Preview Request' button is visible at the bottom left.

4. Press **Send** and look at the results

You can find documentation about the Full REST API at [Azure DevOps Services REST API Reference⁷¹](#).

5. Try more calls

Links

For additional information, see also:

- [About dashboards, charts, reports, & widgets⁷²](#)
- [Widget catalog⁷³](#)

⁷¹ <https://docs.microsoft.com/en-us/rest/api/azure/devops/?view=azure-devops-rest-5.0>

⁷² <https://docs.microsoft.com/en-us/azure/devops/report/dashboards/overview?view=vsts>

⁷³ <https://docs.microsoft.com/en-us/azure/devops/report/dashboards/widget-catalog?view=vsts>

Module Review and Takeaways

Module Review Questions

Suggested answer

How many deployment jobs can be run concurrently by a single agent?

Suggested answer

What should you create to store values that you want to make available across multiple build and release pipelines?

Suggested answer

Which Azure-based tool can be used to safeguard cryptographic keys and other secrets used by cloud apps and services?

Suggested answer

How can you provision the agents for deployment groups in each of your VMs?

Suggested answer

How can you identify a default release variable?

Answers

How many deployment jobs can be run concurrently by a single agent?

One

What should you create to store values that you want to make available across multiple build and release pipelines?

Variable group

Which Azure-based tool can be used to safeguard cryptographic keys and other secrets used by cloud apps and services?

Azure Key Vault

How can you provision the agents for deployment groups in each of your VMs?

One of the following:

- * Run the script that is generated automatically when a deployment group is created*
- * Install the Azure Pipelines Agent Azure VM extension on each of the VMs*
- * Use the Azure Resource Group Deployment task in your release pipeline*

How can you identify a default release variable?

It has Release. as its prefix (eg: Release.Reason)

Module 12 Implement an Appropriate Deployment Pattern

Module Overview

Module Overview

This module is about implementing an appropriate deployment pattern.

Canary Releases

Canary releases enable you to expose a feature to a small subset of users to make sure your application continues to behave correctly in a production scenario.

Dark Launching

This has some overlap with Canary releases. But there is a difference. A dark launch is about launching a feature to a group of users without announcing it or without visibility of the feature. This way you can measure how users react on a feature and how the system behaves.

A/B Testing

Then we will briefly talk about the concept of A/B Testing. Although A/B Testing is not something that is required to implement when doing Continuous Delivery, it is something that can be enabled by Continuous Delivery. A/B Testing is running experiments on a production server, with the primary target to increase the usage of a particular page or feature. By exposing the feature to only a subset of your population, statistical analysis can be done on what is successful and not.

Progressive exposure deployment

We will finalize this module by discussing Progressive exposure deployment, or, ring based deployment. Ring based deployment is gradually rolling out a feature to new users or environments to measure impact and reduce risk carefully.

Learning objectives

After completing this module, students will be able to:

- Describe deployment patterns
- Implement Blue Green Deployment
- Implement Canary Release
- Implement Progressive Exposure Deployment

Introduction to Deployment Patterns

Continuous Delivery and Architecture

Continuous Delivery is more than release management. Continuous Delivery is all about the process, the people, and the tools that you need to make sure that you can deliver your software on demand. You need to recognize that deployment is only 1 step within the whole Continuous Delivery process. To be able to deploy on demand or multiple times a day, all the prerequisites need to be in place.

For example:

Testing Strategy

Your testing strategy should be in place. If you need to run a lot of manual tests to validate your software, this is a bottleneck to deliver on demand.

Coding practices

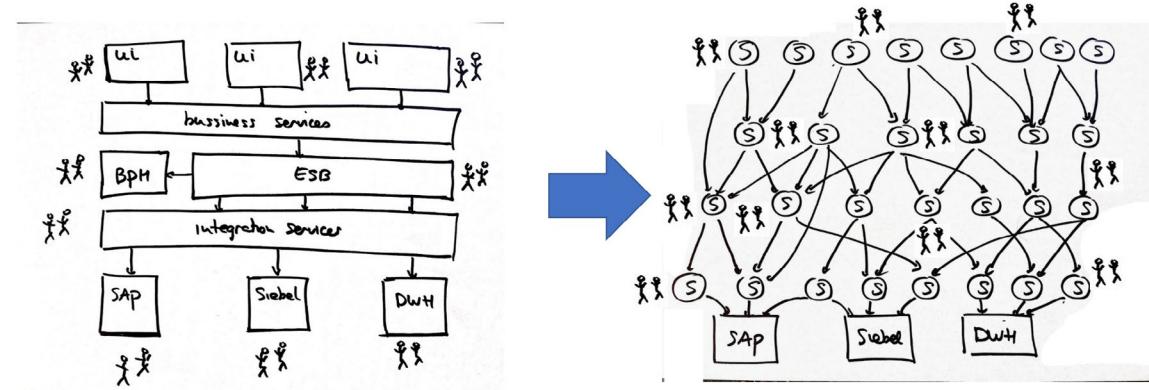
If your software is not written in a safe and maintainable manner, the chances are that you cannot maintain a high release cadence. When your software is complex because of a large amount of technical Debt, it is hard to quickly and reliably change the code. Writing high-quality software and high-quality tests are, therefore, an essential part of Continuous Delivery.

Architecture

The architecture of your application is always significant. But when implementing Continuous Delivery, it is maybe even more so. If your software is a monolith with a lot of tight coupling between the various components, it is difficult to deliver your software continuously. Every part that is changed might impact other parts that did not change. Automated tests can track a lot of these unexpected dependencies but it is still hard. There is also the time aspect when working with different teams. When Team A relies on a service of Team B, Team A cannot deliver until Team B is done. This introduces another constraint on delivery.

Continuous Delivery for large software products is hard. For smaller parts, it is easier. Therefore, breaking up your software into smaller, independent pieces, is in many cases a good solution. Nowadays, you frequently hear the term microservices. A microservice is an autonomous, independent deployable, and scalable software component. They are small, and they are focused on doing one thing very well, and they can run autonomously. If one micro service changes, it should not impact any other microservices within your landscape. By choosing a microservices architecture, you will create a landscape of services that can be developed, tested and deployed separately from each other.

Of course, this implies other risks and complexity. You need to create to keep track of interfaces and how they interact with each other. And you need to maintain multiple application lifecycles instead of one.



In a traditional application, we can often see a multi-layer architecture. One layer with the UI, a layer with the business logic and services and a layer with the data services. Sometimes there are dedicated teams for the UI and the backend. When something needs to change, it needs to change in all the layers.

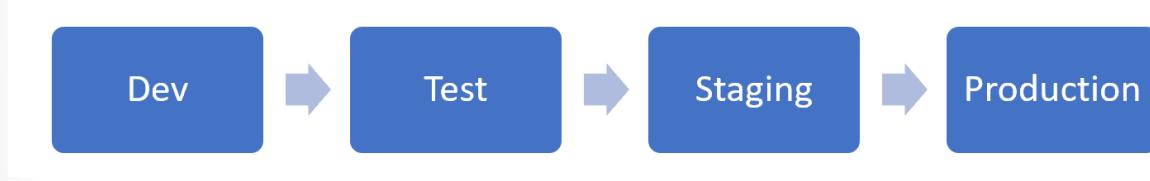
When moving towards a microservices architecture, all these layers are part of the same microservice. Only the microservice contains one specific function. The interaction between the microservices is done in an asynchronous matter. They do not call each other directly but make use of asynchronous mechanisms like queues or events

Each microservice has its own lifecycle and Continuous Delivery pipeline. If you built them correctly, you could deploy new versions of a microservice without impacting other parts of the system.

A microservice architecture is undoubtedly not a prerequisite for doing Continuous Delivery, but smaller software components certainly help in implementing a fully automated pipeline.

Deployment Patterns

When we have our prerequisites in place to be able to deliver our software continuously, we need to start thinking about a deployment pattern. Traditionally a deployment pattern was pretty straightforward.



The software was built, and when all features had been implemented, the software was deployed to an environment where a group of people could start using it.

The traditional or classical deployment pattern was moving your software to a development stage, a testing stage, maybe an acceptance or staging stage, and finally a production stage. The software moved as one piece through the stages. The production release was in most cases a Big Bang release, where users were confronted with a lot of changes at the same time. In spite of the different stages to test and validate, this approach still involves a lot of risks. By running all your tests and validation on non-production environments, it is hard to predict what happens when your production users start using it. Of course you can run load tests and availability tests, but in the end, there is no place like production.

End users always use your application differently. Unexpected events will happen in a data center, multiple events from multiple users will occur at the same time, triggering some code that has not been

tested in that way. To overcome this, we need to embrace the fact that some features can only be tested in production.

Testing in production sounds a bit scary, but that should not be the case. When we talked about separating our functional and technical release, we already saw that it is possible to deploy features without exposing them to all the users. When we take this concept, of feature toggling, and use it together with our deployment patterns, we can test our software in production.

For example

- Blue-Green deployments
- Canary releasing
- Dark launching
- A/B testing
- Progressive Exposure Deployment

We will talk about the different patterns in the next chapters.

With all deployment patterns, the main thing to consider is the creation of feedback loops. The only way to implement a deployment pattern successfully is to have monitoring in place. You need to know how your system behaves. Before a deployment and after a deployment, so you can see the differences and the impact.

Availability

You need to know details about your availability. Is your app available? What are the response times?

Performance

How does your application perform? What is the average CPU usage? And the average memory usage? Are there spikes in performance?

Usage

You need to know about your usage. How do people interact with your system? In what time frames do they interact with your system? Do they finish the flows or do they drop off?

Diagnostics

You need to have a lot of diagnostic logging and telemetry available. To see what happens when something goes wrong, or to see what happened before a performance spike, or a user drops off.

Discussion

A critical look at your architecture

Are your architecture and the current state of your software ready for Continuous Delivery?

Topics you might want to consider are:

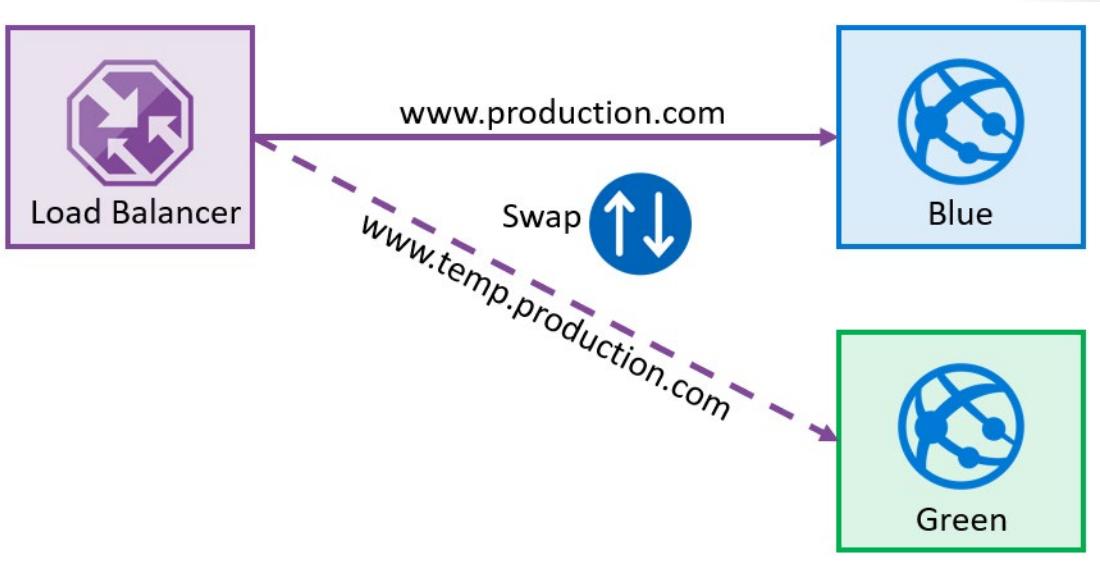
- Is your software built as one big monolith, or is it divided into multiple components?
- Can you deliver parts of your application separately?
- Can you guarantee the quality of your software when deploying multiple times a week?
- How do you test your software?
- Do you run one or multiple versions of your software?
- Can you run multiple versions of your software side-by-side?

- What do you need to improve to implement Continuous Delivery?

Implement Blue Green Deployment

Blue Green Deployment

Blue-Green deployment is a technique that reduces risk and downtime by running two identical environments. These environments are called Blue and Green. At any time, only one of the environments is live, with the live environment serving all production traffic.



For this example, Blue is currently live, and Green is idle.

As you prepare a new version of your software, the deployment and the final stage of testing takes place in the environment that is not live: in this example, Green. Once you have deployed and thoroughly tested the software in Green, you switch the router or load balancer, so all incoming requests now go to Green instead of Blue. Green is now live, and Blue is idle.

This technique can eliminate downtime due to app deployment. Besides, Blue-Green deployment reduces risk: if something unexpected happens with your new version on Green, you can immediately roll back to the last version by switching back to Blue.

When this involves database schema changes, this process is of course not straightforward. You can probably not swap your application. In that case, your application and architecture should be built in such a way, that it can handle both the old and the new database schema.

Deployment Slots

When using a cloud platform like Azure, doing Blue-Green deployments is relatively easy. You do not need to write your own code or set up infrastructure. When using web apps, you can use an out-of-the-box feature called Deployment Slots.

Deployment Slots are a feature of Azure App Service. They are live apps with their own hostnames. You can create different slots for your application (e.g., Dev, Test or Stage). The Production slot is the slot where your live app resides. With deployment slots, you can validate app changes in staging before swapping it with your production slot.

You can use a deployment slot to set up a new version of your application, and when ready, swap the production environment with the new, staging environment. This is done by an internal swapping of the IP addresses of both slots.

To learn more about Deployment Slots, see also:

- **Set up Staging Environments in Azure App Service**¹
- **Considerations on using Deployment Slots in your DevOps Pipeline**²

Demonstration Set Up a Blue-Green Deployment

In this demonstration, you will investigate Blue-Green Deployment.

Note: Before starting this walkthrough, make sure you have performed the steps in the prerequisites section and the previous walkthroughs.

Steps

Let's now take a look at how a release pipeline can be used to implement blue-green deployments.

We'll start by creating a new project with a release pipeline that can perform deployments, by using the **Parts Unlimited** template again.

An initial app deployment

1. In a browser, navigate to <http://https://azureddevopsdemogenerator.azurewebsites.net> and click **Sign in**.
You will be prompted to sign in if necessary.
2. In the **Create New Project** window, select your existing Organization, set the **Project Name** to **PU Hosted** and click **Choose template**.

The screenshot shows the 'Create New Project' interface. At the top, there's a 'Choose a template' dropdown and a navigation bar with tabs: General, DevOps Labs, Microsoft Learn, and Microsoft Cloud Adoption Framework. Below the tabs, there are six project templates displayed in a grid:

- Tailwind Traders**: An ASP.NET & React application using Agile, Data and AI, and React. It uses Azure App Service, AKS, Cosmos DB, Logic App, and Function App.
- SmartHotel360**: A template containing work items, code, and pipeline definitions for the public web site of SmartHotel360, an E2E reference sample app with several consumer and line-of-business apps and an Azure backend. It uses scrum, aspnetcore, and azureappservice.
- MyHealthClinic**: A template provisions a scrum based team project with code, work items for a sample ASP.NET Core web application - My Health Clinic. It includes pipeline definitions to build and deploy the web app to Azure App Service. It uses scrum, aspnetcore, and azureappservice.
- PartsUnlimited**: A scrum-based team project using scrum, aspnetcore, and azureappservice. It contains sample work items, complete source code, and pipeline definitions to deploy Parts Unlimited to Azure.
- PartsUnlimited-YAML**: A scrum-based team project using scrum, aspnetcore, and azureappservice. It contains sample work items, complete source code, and pipeline definitions to deploy Parts Unlimited to Azure.
- MyShuttle**: A Java application using scrum, java, application, and azure web app. It uses mysql.

At the bottom right of the grid is a blue 'Select Template' button.

3. Click on the **PartsUnlimited** project (not the PartsUnlimited-YAML project), and click **Select Template**, then click **Create Project**. When the deployment completes, click **Navigate to project**.
4. In the main menu for **PU Hosted**, click **Pipelines**, then click **Builds**, then **Queue** and finally **Run** to start a build.

¹ <https://docs.microsoft.com/en-us/azure/app-service/deploy-staging-slots>

² <https://blogs.msdn.microsoft.com/devops/2017/04/10/considerations-on-using-deployment-slots-in-your-devops-pipeline/>

The build should succeed.

Note: warnings might appear but can be ignored for this walkthrough

✓ #20190904.1: Updated FullEnvironmentSetupMerged.param.json

Manually run today at 12:26 by Greg Low ⚡ PartsUnlimited ⚡ master ⚡ 58d9b87

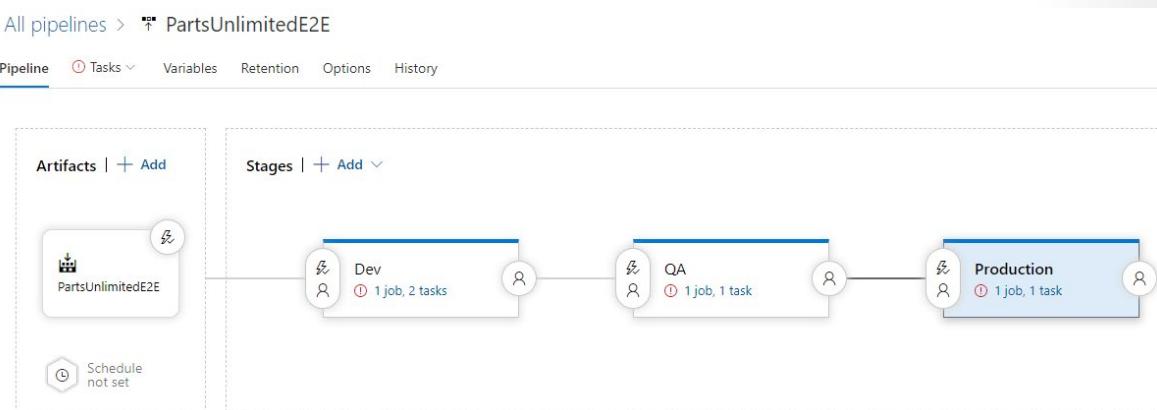
Logs Summary Tests WhiteSource Bolt Build Report

Phase 1

Pool: Azure Pipelines · Agent: Hosted Agent

- ✓ Prepare job · succeeded
- ✓ Initialize job · succeeded
- ✓ Checkout · succeeded
- ✓ NuGet restore · succeeded 3 warnings
- ✓ Build solution · succeeded 1 warning
- ✓ Test Assemblies · succeeded 1 warning
- ✓ Copy Files · succeeded
- ✓ Publish Artifact · succeeded
- ✓ Post-job: Checkout · succeeded
- ✓ Finalize Job · succeeded
- ✓ Report build status · succeeded

5. In the main menu, click **Releases**. Because a continuous integration trigger was in place, a release was attempted. However, we have not yet configured the release so it will have failed. Click **Edit** to enter edit mode for the release.



6. From the drop down list beside **Tasks**, select the **Dev** stage, then click to select the **Azure Deployment** task.
7. In the **Azure resource group deployment** pane, select your Azure subscription, then click **Authorize** when prompted. When authorization completes, select a **Location** for the web app.

Note: you might be prompted to log in to Azure at this point

The screenshot shows the configuration for an Azure resource group deployment task. The task version is set to 2.*. The display name is "Azure Deployment". Under "Azure Details", the "Azure subscription" dropdown is set to "Microsoft" (with a red box around it), and the "Location" dropdown is set to "East US" (also with a red box around it). The "Action" is set to "Create or update resource group", and the "Resource group" is set to "\${ResourceGroupName}". Under "Template", the "Template location" is "Linked artifact" and the "Template" path is "\${System.DefaultWorkingDirectory}/PartsUnlimitedE2E/drop/PartsUnlimited-aspnet45/env/PartsUnlimitedEnv/Templates/FullEnvironmentSetupMerged.json".

8. In the task list, click **Azure App Service Deploy** to open its settings. Again select your Azure subscription. Set the **Deployment slot** to **Staging**.

Azure App Service deploy ⓘ

Task version 3.*

Display name *

Azure App Service Deploy

Azure subscription * ⓘ | Manage ⓘ

Microsoft

Scoped to subscription 'Microsoft Azure Sponsorship'

App type * ⓘ

Web App

App Service name * ⓘ

\$(WebsiteName)

Deploy to slot ⓘ

Resource group * ⓘ

\$(ResourceGroupName)

Slot * ⓘ

Staging

Note: the template creates a production site and two deployment slots: Dev and Staging. We will use Staging for our Green site.

9. In the task list, click **Dev** and in the **Agent job** pane, select **Azure Pipelines** for the **Agent pool** and **vs2017-win2016** for the **Agent Specification**.

Agent job ⓘ

Display name *

Dev

Agent selection ⌂

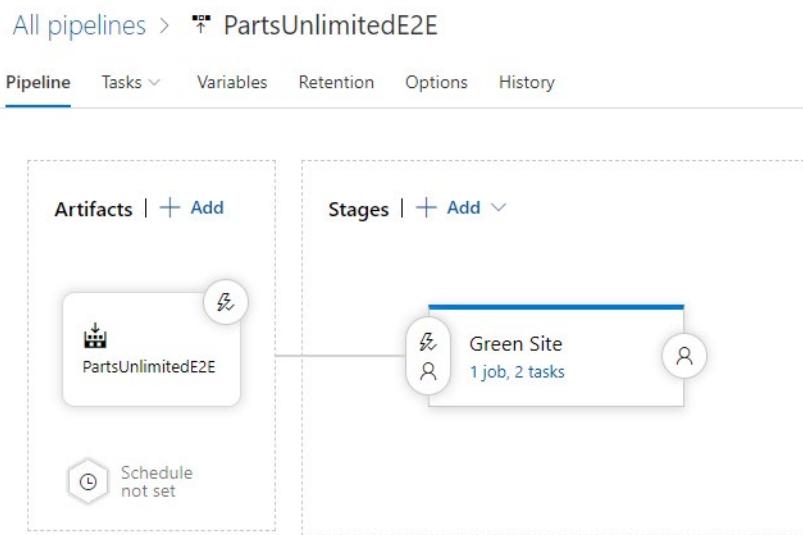
Agent pool ⓘ | Pool information | Manage ⓘ

Azure Pipelines

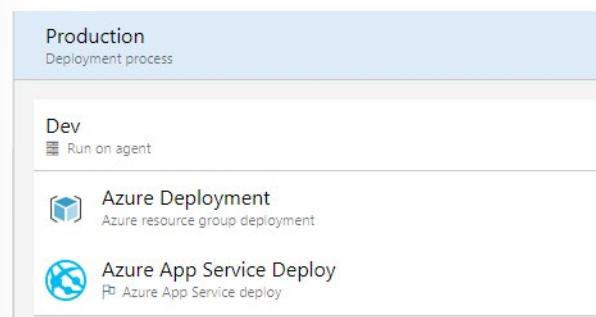
Agent Specification *

vs2017-win2016

10. From the top menu, click **Pipelines**. Click the **Dev** stage, and in the properties window, rename it to **Green Site**. Click the **QA** stage, and click **Delete** and **Confirm**. Click the **Production** stage, and click **Delete** and **Confirm**. Click **Save** then **OK**.



11. Hover over the **Green Site** stage, and click the **Clone** icon when it appears. Change the **Stage name** to **Production**. From the **Tasks** drop down list, select **Production**.

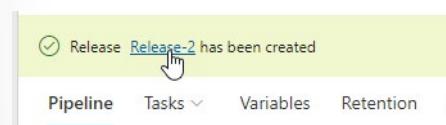


12. Click the **Azure App Service Deploy** task, and uncheck the **Deploy to slot** option. Click **Save** and **OK**.

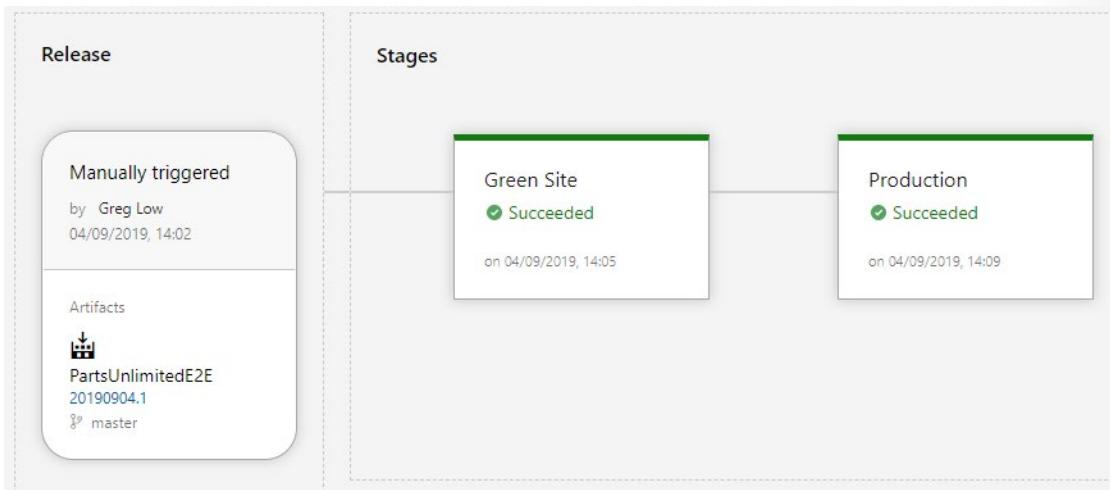
The configuration dialog for the 'Azure App Service Deploy' task is shown. The 'App Service name' field is set to '\$(WebsiteName)'. The 'Deploy to slot' checkbox is unchecked. The 'Virtual application' field is empty.

The production site isn't deployed to a deployment slot. It is deployed to the main site.

13. Click **Create release** then **Create** to create the new release. When it has been created, click the release link to view its status.



After a while, the deployment should succeed.

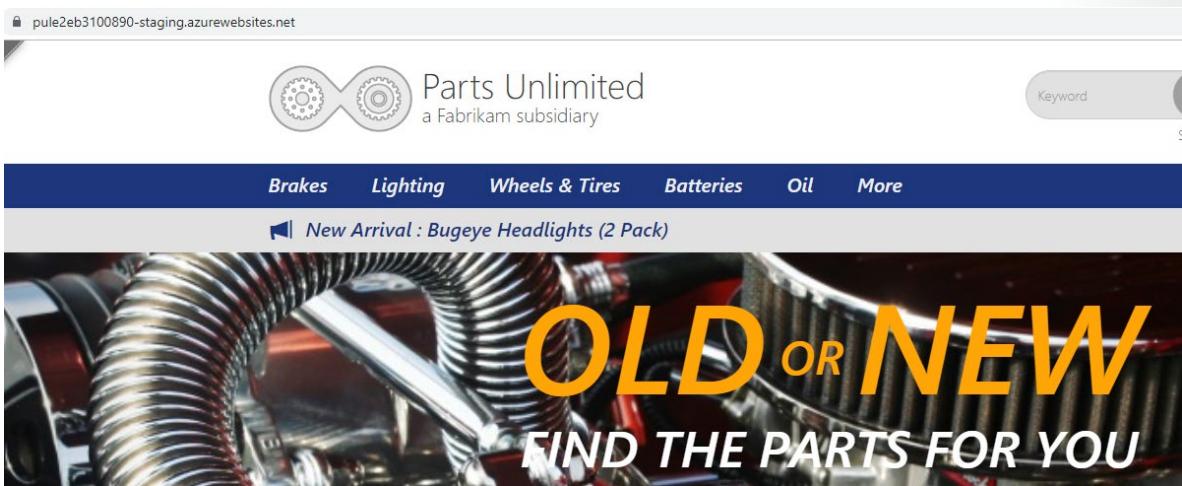


Test the Green Site and the Production Site

14. In the **Azure Portal**, open the blade for the **ASPDOTNET** resource group that was created by the project deployment. Notice the names of the web apps that have been deployed. Click to open the **Staging*** web app's blade. Copy the URL from the top left hand side.

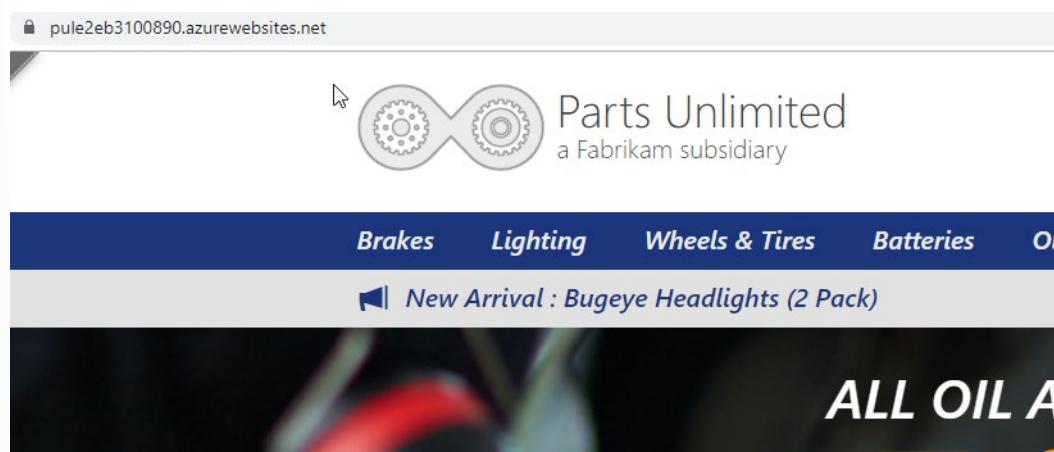
Resource group (change) : ASPDOTNET	URL : https://pule2eb3100890-staging.azurewebsites.net
Status : Running	App Service Plan : pule2e (S1: 1)
Location : East US	https://pule2eb3100890-staging.azurewebsites.net

15. Open a new browser tab and navigate to the copied URL. It will take the application a short while to compile but then the Green website (on the Staging slot) should appear.



*Note: you can tell that the staging slot is being used because of the **-staging** suffix in the website URL*

16. Open another new browser tab, and navigate to the same URL but without the **-staging** slot. The production site should also be working.



Note: Leave both browser windows open for later in the walkthrough

Configure Blue-Green swap and Approval

Now that both sites are working, let's configure the release pipeline for Blue-Green deployment.

17. In **Azure DevOps**, in the main menu for the **PU Hosted** project, click **Pipelines**, then click **Releases**, then click **Edit** to return to edit mode.
18. Click the **Production** stage, click **Delete**, then **Confirm** to remove it. Click **+Add** to add an extra stage, and click **Empty job** for the template. Set **Swap Blue-Green** for the **Stage name**.

Stages | + Add ▾



19. Click **Variables** and modify the **Scope** of **WebsiteName** to **Release**.

Variables Retention Options History

Name	Value	Scope
HostingPlan	pule2e	Release
ResourceGroupName	ASPDOTNET	Release
ServerName	pule2eb3100890	Release
WebsiteName	pule2eb3100890	Release

20. From the **Tasks** drop down list, click to select the **Swap Blue-Green** stage. Click the **+** to the right hand side of **Agent Job** to add a new task. In the **Search** box, type **cli**.

Add tasks | Refresh

Docker CLI installer
Install Docker CLI on agent machine.

Azure CLI
Run Azure CLI commands against an Azure subscription in a Shell script when running on Linux agent or Batch script when running on Windows agent.

Python pip authenticate
Authentication task for the pip client used for installing Python distributions

21. Hover over the **Azure CLI** template and when the **Add** button appears, click it, then click to select the **Azure CLI** task, to open its settings pane.

Azure CLI ⓘ

Task version 1.*

Display name *

Azure CLI

Azure subscription * ⓘ | Manage ⓘ

Script Location * ⓘ

Script path

Script Path * ⓘ

This setting is required.

Arguments ⓘ

22. Configure the pane as follows, with your subscription, a **Script Location** of **Inline script**, and the **Inline Script** as follows:

```
az webapp deployment slot swap -g $(ResourceGroupName) -n $(WebsiteName) --slot Staging --target-slot production
```

The screenshot shows the 'Task version' dropdown set to '1.*'. The 'Display name' field contains 'Azure CLI'. Under 'Azure subscription', 'Microsoft' is selected, with a note indicating it's scoped to the 'Microsoft Azure Sponsorship' subscription. The 'Script Location' is set to 'Inline script', and the inline script content is:

```
az webapp deployment slot swap -g $(ResourceGroupName) -n $(WebsiteName) --slot Staging --target-slot production
```

23. From the menu above the task list, click **Pineline**. Click the **Pre-deployment conditions** icon for the

Swap Blue-Green task, then in the **Triggers** pane, enable **Pre-deployment approvals**.

24. Configure yourself as an approver, click **Save**, then **OK**.

Stages | + Add ▾



Test the blue-green swap

25. In the **PU Hosted** main menu, click **Repos** then click **Files** to open the project files. Navigate to the following file.

The screenshot shows the GitHub repository interface for 'PartsUnlimited'. The 'Index.cshtml' file is open in the 'src/PartsUnlimitedWebsite/Views/Home' directory. The code content is:

```
1 @model PartsUnlimited.ViewModels.HomeViewModel
2
3 @{
4     ViewBag.Title = "Home Page";
5 }
6
7 @section scripts {
    ...
```

We will make a cosmetic change so that we can see that the website has been updated. We'll change the word **tires** in the main page rotation to **tyres** to target an international audience.

26. Click **Edit** to allow editing, then find the word **tires** and replace it with the word **tyres**. Click **Commit** and **Commit** to save the changes and trigger a build and release.

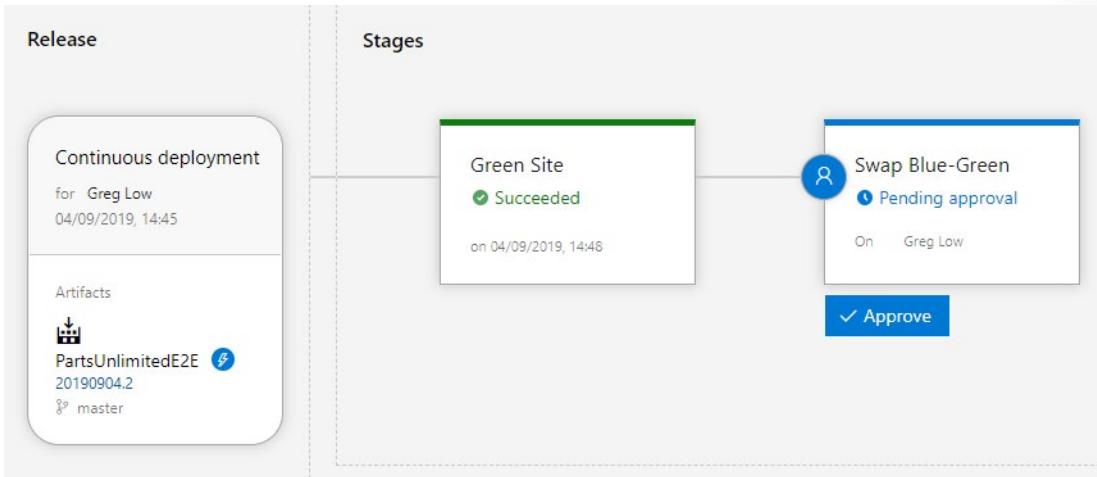
```
<div class="item" style="background-image: url('/Images/hero_imag
<a href="@Url.Action("Browse", "Store", new { categoryId = 3
    <div class="container">
        <p>New tyres</p>
        <ul>
            <li>Improved fuel efficiency</li>
            <li>Superior wet weather breaking</li>
            <li>Added durability</li>
        </ul>
    </div>
```

27. From the main menu, click **Pipelines**, then **Builds**. Wait for the continuous integration build to complete successfully.

PartsUnlimitedE2E

History	Analytics	Commit	Build #	Branch
 Updated Index.cshtml CI build for Greg Low			 20190904.2	 master
 Updated FullEnvironmentSetupMerged.param.json Manual build for Greg Low			 20190904.1	 master

28. From the main menu, click **Releases**. Click to open the latest release (at the top of the list).



You are now being asked to approve the deployment swap across to production. We'll check the green deployment first.

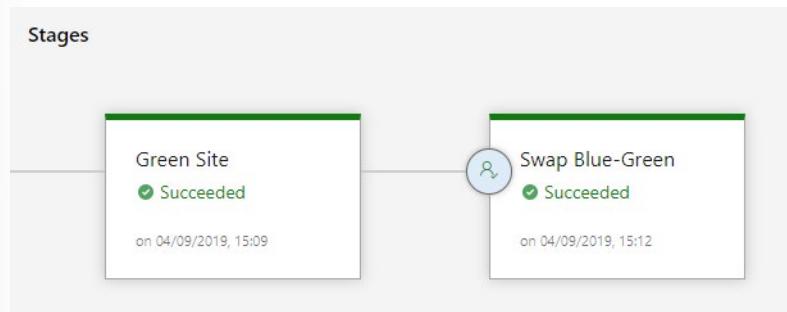
29. Refresh the Green site (i.e. Staging slot) browser tab and see if your change has appeared. It now shows the altered word.



30. Refresh the Production site browser tab and notice that it still isn't updated.



31. As you are happy with the change, in release details click **Approve**, then **Approve** and wait for the stage to complete.



32. Refresh the Production site browser tab and check that it now has the updated code.



Final Notes

If you check the production site, you'll see it now has the previous version of the code.

This is the key difference with using Swap, rather than just having a typical deployment process from one staged site to the next. You have a very quick fall back option by being able to swap the sites back again if needed.

Feature Toggles

Introduction to Feature Toggles

Feature Flags allow you to change how our system works without making changes to the code. Only a small configuration change is required. In many cases, this will also only be for a small number of users. Feature Flags offer a solution to the need to push new code into trunk and have it deployed, but not have it functional yet. They are commonly implemented as the value of variables that are used to control conditional logic.

Imagine that your team are all working in the main trunk branch of a banking application. You've decided it's worth trying to have all the work done in the main branch to avoid messy operations of merge later, but you need to make sure that substantial changes to how the interest calculations work can happen, and people depend on that code everyday. Worse, the changes will take you weeks to complete. You can't leave the main code broken for that period of time. A Feature Flag could help you get around this. You can change the code so that other users who don't have the Feature Flag set will keep using the original interest calculation code, and the members of your team who are working on the new interest calculations and who have the Feature Flag set see the code that's been created. This is an example of a business Feature Flag that's used to determine business logic.

The other type of Feature Flag is a Release Flag. Now, imagine that after you complete the work on the interest calculation code, you're perhaps nervous about publishing a new code out to all users at once. You have a group of users who are better at dealing with new code and issues if they arise, and these people are often called Canaries. The name is based on the old use of Canaries in coal mines. You change the configuration, so that the Canary users also have the Feature Flag set and they will start to test the new code as well. If problems occur, you can quickly disable the flag for them again.

Another release flag might be used for AB testing. Perhaps you want to find out if a new feature makes it faster for users to complete a task. You could have half the users working with the original version of the code and the other half of the users working with the new version of the code. You can then directly compare the outcome and decide if the feature is worth keeping. Note that Feature Flags are sometimes called Feature Toggles instead.

By exposing new features by just "flipping a switch" at runtime, we can deploy new software without exposing any new or changed functionality to the end user.

The question is what strategy you want to use in releasing a feature to an end user.

- Reveal the feature to a segment of users, so you can see how the new feature is received and used.
- Reveal the feature to a randomly selected percentage of users.
- Reveal the feature to all users at the same time.

The business owner plays a vital role in the process, and you need to work closely together with him to choose the right strategy.

Just as in all the other deployment patterns and mentioned in the introduction, the most important part is that you always look at the way the system behaves.

The whole idea of separating feature deployment from Feature exposure is compelling and something we want to incorporate in our Continuous Delivery practice. It helps us with more stable releases and better ways to roll back when we run into issues when we have a new feature that produces problems. We switch it off again and then create a hotfix. By separating deployments from revealing a feature, you create the opportunity to release any time a day, since the new software will not affect the system that already works.

What are feature toggles

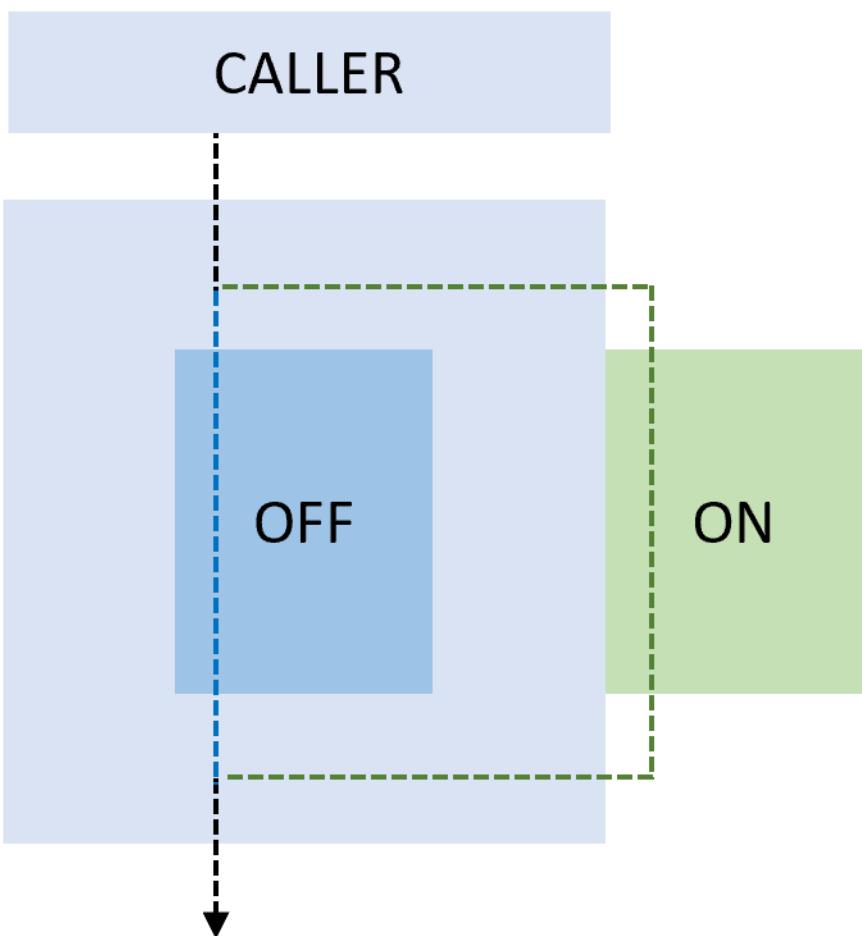
Feature toggles are also known as feature flippers, feature flags, feature switches, conditional features, etc.

Besides the power they give you on the business side, they provide an advantage on the development side as well. Feature toggles are a great alternative to branching as well. Branching is what we do in our version control system. To keep features isolated, we maintain a separate branch. The moment we want the software to be in production, we merge it with the release branch and deploy.

With feature toggles, you build new features behind a toggle. When a release occurs, your feature is "off" and should not be exposed to or impacting the production software.

How to implement a feature toggle

In the purest form, a feature toggle is an IF statement.



When the switch is off, it executes the code in the IF, otherwise the ELSE. Of course, you can make it much more intelligent, controlling the feature toggles from a dashboard, or building capabilities for roles, users, etc.

If you want to implement feature toggles, then there are many different frameworks available, both commercially as Open Source.

For more information, see also **Explore how to progressively expose your features in production for some or all users³**.

Feature Toggle Maintenance

A feature toggle is just code. And to be more specific, conditional code. It adds complexity to the code and increases the technical debt. Be aware of that when you write them, and clean up when you do not need them anymore.

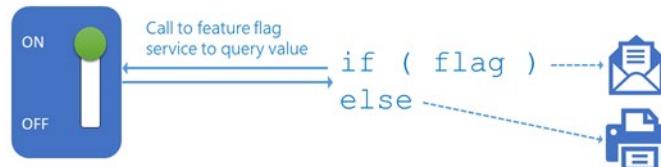
The idea of a toggle is that it is short lived and only stays in the software for the time it is necessary to release it to the customers.

You can classify the different types of toggles based on two dimensions as described by Martin Fowler. He states that you can look at the dimension of how long a toggle should be in your codebase and on the other side how dynamic the toggle needs to be.

You can see that only a few toggles stay in your software, but those are more toggles that you already used to authorize people in your system or some kill switches that you need as an administrator to enable you to turn features off when load or other issues make the system misbehave. So always evaluate the toggles you have and remove them when you can.

The most important thing is to remember that you need to remove the toggles from the software as soon as possible. If you do not do that, they will become a form of technical debt if you keep them around for too long.

Planning Feature Flag Lifecycles



While feature flags can be really useful, they can also introduce many issues of their own.

As soon as you introduce a feature flag, you have actually added to your overall technical debt. Just like other technical debt, they are easy to add but the longer they are part of your code, the bigger the technical debt becomes, because you've added scaffolding logic that's needed for the branching within the code.

The cyclomatic complexity of your code keeps increasing as you add more feature flags, as the number of possible paths through the code increases.

Using feature flags can make your code less solid and can also add these issues:

- The code is harder to test effectively as the number of logical combinations increases
- The code is harder to maintain because it's more complex
- The code might even be less secure
- It can be harder to duplicate problems when they are found

³ <https://docs.microsoft.com/en-us/azure/devops/articles/phase-features-with-feature-flags?view=vsts>

Because of this, feature flags need to be removed as soon as they aren't needed for the reason they were added. A plan for managing the lifecycle of feature flags is critical. As soon as you add a flag, you need to plan for when it will be removed.

Feature flags shouldn't be repurposed. There have been high profile failures that have occurred because teams decided to reuse an old flag they thought was no longer part of the code, for a new purpose.

Tooling for release flag management

The amount of effort required to manage feature flags should not be underestimated. It's important to consider using tooling that tracks:

- Which flags exist
- Which flags are enabled in which environments, situations, or target customer categories
- The plan for when the flags will be used in production
- The plan for when the flags will be removed

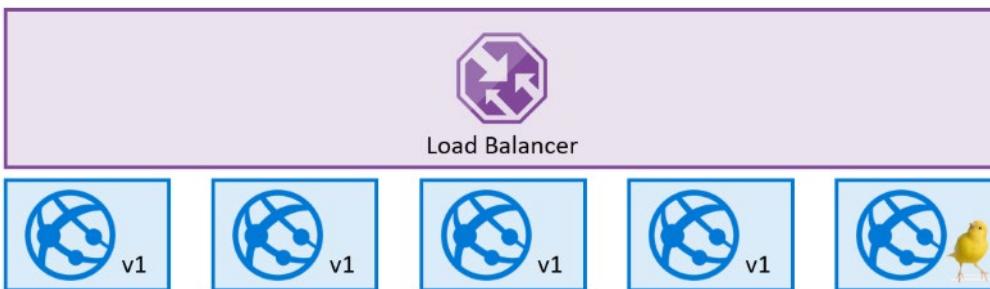
Using a feature flag management system lets you get the benefits of feature flags while minimizing the risk of increasing your technical debt too high.

Canary Releases

Canary Releases

The term canary release comes from the days that miners took a canary with them into the coal mines. The purpose of the canary was to identify the existence of toxic gasses. The canary would die much sooner than the miner, giving them enough time to escape the potentially lethal environment.

A canary release is a way to identify potential problems as soon as possible without exposing all your end users to the issue at once. The idea is that you expose a new feature only to a minimal subset of users. By closely monitoring what happens the moment you enable the feature, you can get relevant information from this set of users and decide to either continue or rollback (disable the feature). If the canary release shows potential performance or scalability problems, you can build a fix for that and apply that in the canary environment. After the canary release has proven to be stable, you can move the canary release to the actual production environment.



Traffic Manager

In the previous module, we saw how Deployment slots in Azure Web Apps, enable you to swap between 2 different versions of your application quickly. If you want to have more control over the traffic that flows to your different versions, deployment slots alone is not enough. To control traffic in Azure, you can use a component called Traffic Manager.

Azure Traffic Manager is a DNS-based traffic load balancer that enables you to distribute traffic optimally to services across global Azure regions while providing high availability and responsiveness.

Traffic Manager uses DNS to direct client requests to the most appropriate service endpoint based on a traffic-routing method and the health of the endpoints.

An endpoint is an Internet-facing service hosted inside or outside of Azure. Traffic Manager provides a range of traffic-routing methods and endpoint monitoring options to suit different application needs and automatic failover models. Traffic Manager is resilient to failure, including the breakdown of an entire Azure region.

Traffic manager provides four options to distribute traffic:

- **Route traffic based on availability** - This option is routing based on availability. This is predominantly a mechanism to provide failover if one app service instance appears to be down. Traffic manager will take care of routing the traffic to other instances that are available, so the end user will not experience any downtime. This is an excellent feature if you want to release new features without any downtime. You can create a new app service, then deploy the new software on the new app services and then bring the new app service online and let traffic manager move the traffic to the new instance. The moment all traffic is on the new instance you can then remove the old instance. This way

you move customers to new software, without causing downtime. This is more or less comparable to the deployment slots, but in this case, we can do this between different app services.

- **Route traffic round robin** - This option is the round-robin routing of traffic. This is primarily used for load distribution.
- **Route traffic based on weight** - This option is about routing traffic based on weights. The idea here is that you can set a weight, for example, 1:4, on the different app services it should route to. This causes 1/4 of the traffic to go to one instance and the other 3/4 of the traffic to go to the other instance. It selects the clients randomly. Therefore this is a useful mechanism to be used for A/B testing.
- **Route traffic based on latency** - The final option is to route based on latency. This is primarily used to ensure you route traffic to the right geolocation ensuring the client has the lowest latency possible. But in case of a problem, this also means it will fail over to another data center since the one with shorter latency is not available at that moment.

When we look at the options the traffic manager offers, the most used options for Continuous Delivery are the options to route traffic based on availability and the option to route based on weights.

For more information, see also:

- [What is Traffic Manager?](#)⁴
- [How Traffic Manager works](#)⁵

Controlling your canary release

Using a combination of feature toggles, deployment slots and traffic manager you can achieve full control over the traffic flow, and enable your canary release.

You deploy the new feature to the new deployment slot or a new instance of an application, and you enable the feature after verifying the deployment was successful.

Next, you set the traffic to be distributed to a small percentage of the users, and you carefully watch the behavior of the application, e.g. by using application insights to monitor performance and stability of the application.

Demonstration Ring-based Deployment

In this demonstration, you will investigate Ring-based Deployment.

Note: Before starting this walkthrough, make sure you have performed the steps in the prerequisites section and the previous walkthroughs.

Steps

Let's now take a look at how a release pipeline can be used to stage features by using ring-based deployments.

When I have a new features, I might want to just release it to a small number of users at first, just in case something goes wrong. In authenticated systems, I could do this by having those users as members of a security group, and letting members of that group use the new features.

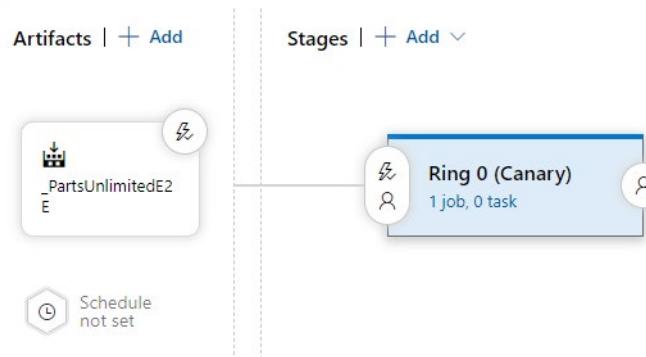
⁴ <https://docs.microsoft.com/en-us/azure/traffic-manager/traffic-manager-overview>

⁵ <https://docs.microsoft.com/en-us/azure/traffic-manager/traffic-manager-how-it-works>

However, on a public web site, I might not have logged in users. Instead, I might want to just direct a small percentage of the traffic to use the new features. Let's see how that's configured. We'll create a new release pipeline that isn't triggered by code changes, but manually when we want to slowly release a new feature.

We start by assuming that a new feature has already been deployed to the Green site (i.e. the staging slot).

1. In the main menu for the **PU Hosted** project, click **Pipelines**, then click **Release**, click **+New**, then click **New release pipeline**.
2. When prompted to select a template, click **Empty job** from the top of the pane.
3. Click on the **Stage 1** stage and rename it to **Ring 0 (Canary)**.



4. Hover over the **New release pipeline** name at the top of the page, and when a pencil appears, click it, and change the pipeline name to **Ring-based Deployment**.

All pipelines > **Ring-based Deployment**

Pipeline Tasks Variables Retention Options History

5. From the **Tasks** drop down list, select the **Ring 0 (Canary)** stage. Click the **+** to add a new task, and from the list of tasks, hover over **Azure CLI** and when the **Add** button appears, click it, then click to select the **Azure CLI** task in the task list for the stage.

Azure CLI ⓘ

Task version 1.*

Display name *

Azure CLI

Azure subscription * ⓘ | Manage ↗

① This setting is required.

Script Location * ⓘ

Script path

Script Path * ⓘ

This setting is required.

[View YAML](#) [Remove](#)

- In the **Azure CLI** settings pane, select your **Azure subscription**, set **Script Location** to **Inline script**, and set the **Inline Script** to the following, then click **Save** and **OK**.

```
az webapp traffic-routing set --resource-group $(ResourceGroupName) --name $(WebsiteName) --distribution staging=10
```

This distribution setting will cause 10% of the web traffic to be sent to the new feature Site (i.e. currently the staging slot).

- From the menu above the task list, click **Variables**. Create two new variables as shown. (Make sure to use your correct website name).

Name	Value	Scope
ResourceGroupName	ASPDOTNET	Release
WebsiteName	pule2eb3100890	Release

- From the menu above the variables, click **Pipeline** to return to editing the pipeline. Hover over the **Ring 0 (Canary)** stage, and click the **Clone** icon when it appears. Select the new stage, and rename it to **Ring 1 (Early Adopters)**.

Stages | + Add ▾



- From the **Tasks** drop down list, select the **Ring 1 (Early Adopters)** stage, and select the **Azure CLI** task. Modify the script by changing the value of **10** to **30** to cause 30% of the traffic to go to the new feature site.

Inline Script *

```
az webapp traffic-routing set --resource-group $(ResourceGroupName) --name $(WebsiteName) --distribution staging=30
```

This allows us to move the new feature into wider distribution if it was working ok in the smaller set of users.

10. From the menu above the tasks, click **Pipeline** to return to editing the release pipeline. Hover over the **Ring 1 (Early Adopters)** stage and when the **Clone** icon appears, click it. Click to select the new stage and rename it to **Public**. Click **Save** and **OK**.

Stages | + Add ▾



11. Click the **Pre-deployment conditions** icon for the **Ring 1 (Early Adopters)** stage, and add yourself as a pre-deployment approver. Do the same for the **Public** stage. Click **Save** and **OK**.

Stages | + Add ▾



The first step in letting the new code be released to the public, is to swap the new feature site (i.e. the staging site) with the production, so that production is now running the new code.

12. From the **Tasks** drop down list, select the **Public** stage. Select the **Azure CLI** task, change the **Display name** to **Swap sites** and change the **Inline Script** to the following:

```
az webapp deployment slot swap -g $(ResourceGroupName) -n $(WebsiteName) --slot staging --target-slot production
```

Inline Script *

```
az webapp deployment slot swap -g $(ResourceGroupName) -n $(WebsiteName) --slot staging --target-slot production
```

Next we need to remove any traffic from the staging site.

13. Right-click the **Swap sites** task, and click **Clone tasks(s)**. Select the **Swap sites copy** task, change its **Display name** to **Stop staging traffic**, and set the **Inline Script** to the following:

```
az webapp traffic-routing set --resource-group $(ResourceGroupName) --name $(WebsiteName) --distribution staging=0
```

14. Click **Save** then **OK** to save your work.

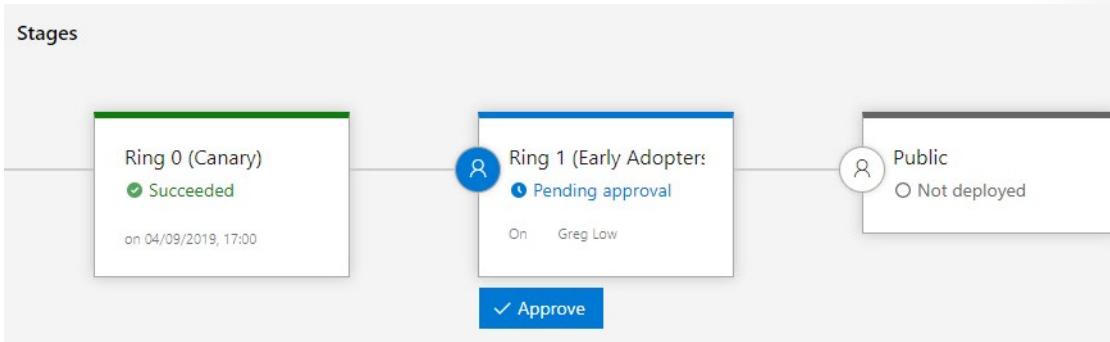
15. Click **Create release** and **Create** to start a release. Click the release link to see the progress.

All pipelines > Ring-based Deployment

Release Release-1 has been created

Pipeline Tasks Variables Retention Options History

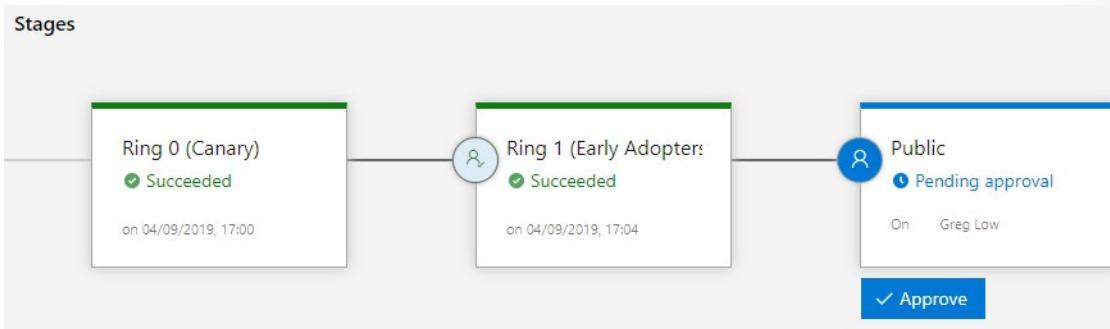
Wait until Ring 0 (Canary) has succeeded.



At this point, 10% of the traffic will be going to the new feature site.

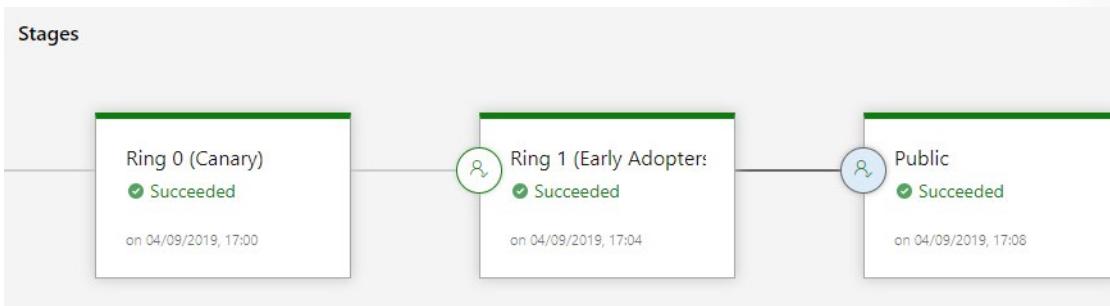
16. Click **Approve** on the **Ring 1 (Early Adopters)** stage, and then **Approve**.

When this stage completes, 30% of the traffic will now be going to the early adopters in ring 1.



17. Click **Approve** on the **Public** stage, and then **Approve**.

When this stage completes, all of the traffic will now be going to the swapped production site, running the new code.



The new feature has been fully deployed.

Dark Launching

Dark launching

Dark Launching is in many ways similar to Canary Releases. However, the difference here is that you are looking to assess the response of users to new features in your frontend, rather than testing the performance of the backend.

The idea is that rather than launch a new feature for all users, you instead release it to a small set of users. Usually, these users are not aware they are being used as test users for the new feature and often you do not even highlight the new feature to them, hence the term "Dark" launching.

Another example of Dark launching is launching a new feature and use it on the backend to get metrics. Let me illustrate this with a real world "launch" example.

As Elon Musk describes in his biography, he applies all kinds of Agile development principles in his company SpaceX. SpaceX builds and launches rockets to launch satellites. SpaceX also uses Dark Launching. When they have a new version of a sensor, they install it alongside the old one. All data is measured and gathered both by the old and the new sensor. Afterward, they compare the outcomes of both sensors. Only when the new one has the same or improved results the old sensor is replaced.

The same concept can be applied in software. You run all data and calculation through your new feature, but it is not "exposed" yet.

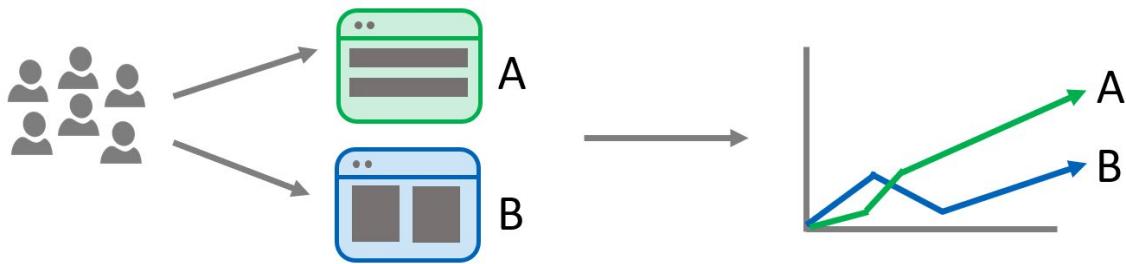
How to implement Dark Launching

In essence dark launching does not differ from a Canary Release or the implementation and switching of a feature toggle. The feature is released and only exposed at a particular time. Therefore, the techniques as described in the previous chapters, do also apply for Dark launching.

AB Testing

A/B Testing

A/B testing (also known as split testing or bucket testing) is a method of comparing two versions of a webpage or app against each other to determine which one performs better. A/B testing is mostly an experiment where two or more variants of a page are shown to users at random, and statistical analysis is used to determine which variation performs better for a given conversion goal.



A/B Testing is not part of Continuous Delivery or a pre-requisite for Continuous Delivery. It is more the other way around. Continuous Delivery allows to quickly delivery MVP's to a production environment and your end-users.

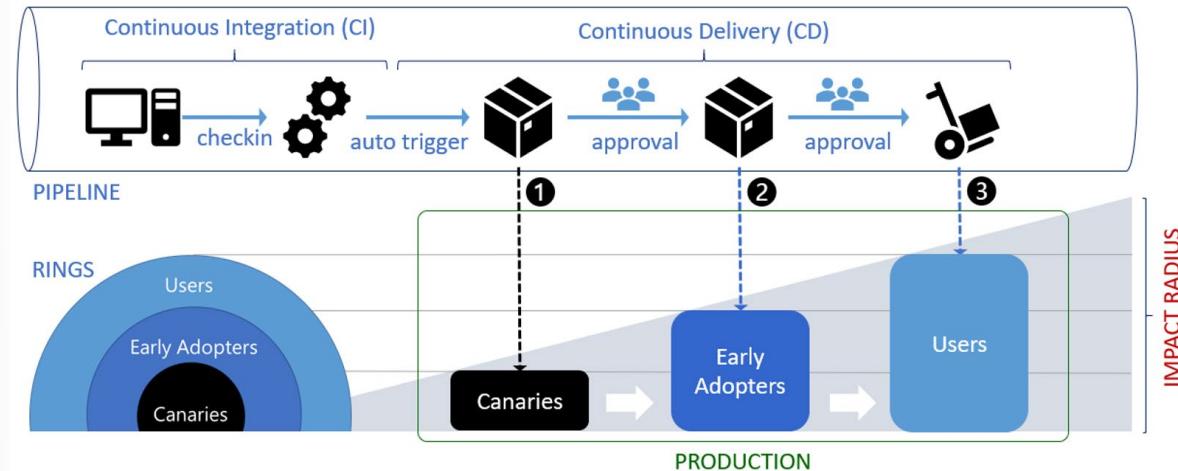
A/B testing is out of scope for this course. But because it is a powerful concept that is enabled by implementing Continuous Delivery, it is mentioned here for you to dive into further.

Progressive Exposure Deployment

Progressive Exposure Deployment

Progressive Exposure Deployment, or also called Ring based deployment, was first discussed in Jez Humble's Continuous Delivery book. They support the production-first DevOps mindset and limit the impact on end users, while gradually deploying and validating changes in production. Impact (also called blast radius), is evaluated through observation, testing, analysis of telemetry, and user feedback.

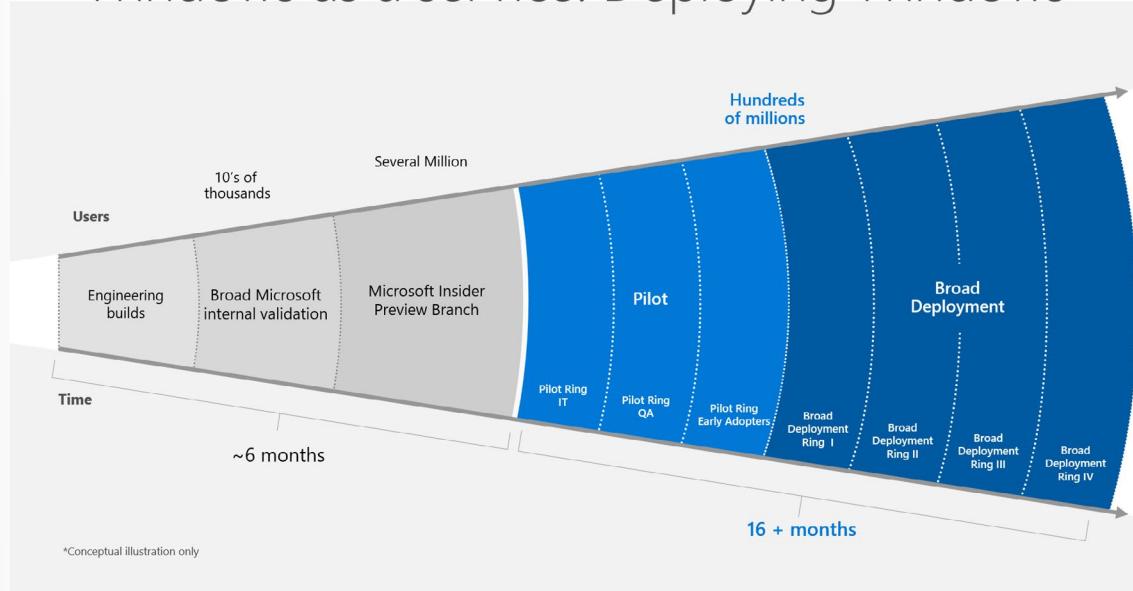
Rings are in essence an extension of the canary stage. The canary release releases to a stage to measure impact. Adding another ring is essentially the same thing.



With a ring based deployment, you deploy your changes to risk-tolerant customers first, and the progressively roll out to a larger set of customers.

The Microsoft Windows team, for example, uses these rings.

Windows as a service: Deploying Windows



When you have identified multiple groups of users, and you see value in investing in a ring-based deployment, you need to define your setup.

Some organizations that use canary releasing have multiple deployment slots set up as rings. They first release the feature to ring 0 that is targeting a very well known set of users, most of the time only their internal organization. After things have been proven stable in ring 0, they then propagate the release to the next ring that also has a limited set of users, but outside their organization.

And finally, the feature is released to everyone. The way this is often done is just by flipping the switch on the feature toggles in the software.

Just as in the other deployment patterns, monitoring and health checks are essential. By using post-deployment release gates that check a ring for health, you can define an automatic propagation to the next ring after everything is stable. When a ring is not healthy, you can halt the deployment to the next rings to reduce the impact.

For more information, see also **Explore how to progressively expose your Azure DevOps extension releases in production to validate, before impacting all users⁶**.

⁶ <https://docs.microsoft.com/en-us/azure/devops/articles/phase-rollout-with-rings?view=vsts>

Lab

Feature Flag Management with LaunchDarkly and Azure DevOps



LaunchDarkly is a continuous delivery platform that provides feature flags as a service and allows developers to iterate quickly and safely. LaunchDarkly gives you the power to separate feature rollout from code deployment and manage feature flags at scale.

In this lab, **Feature Flag Management with LaunchDarkly and AzureDevOps⁷**, you will investigate the use of feature flags and learn how to:

- Implement a very simple feature flag for an ASP.NET MVC application
- Integrate LaunchDarkly with Azure DevOps
- Roll out LaunchDarkly feature flags in Azure DevOps release pipelines

⁷ <https://www.azuredevopslabs.com/labs/vstsextend/launchdarkly/>

Module Review and Takeaways

Module Review Questions

Suggested answer

What is the easiest way to create a staging environment for an Azure webapp?

Suggested answer

What Azure-based tool can you use to divert a percentage of your web traffic to a newer version of an Azure website?

Suggested answer

What characteristics make users suitable for working with Canary deployments?

Suggested answer

What is a potential disadvantage of using Canary deployments?

Suggested answer

Apart from the traffic routing method, what else does Azure Traffic Manager consider when making routing decisions?

Answers

What is the easiest way to create a staging environment for an Azure webapp?

Create a webslot (deployment slot)

What Azure-based tool can you use to divert a percentage of your web traffic to a newer version of an Azure website?

Azure Traffic Manager

What characteristics make users suitable for working with Canary deployments?

High tolerance for issues; Like working with bleeding-edge code.

What is a potential disadvantage of using Canary deployments?

Needing to look after multiple versions of code at the same time. Or, the users might not be the right ones to test changes in the particular deployment.

Apart from the traffic routing method, what else does Azure Traffic Manager consider when making routing decisions?

Health of the end point. (It includes built-in endpoint monitoring and automatic endpoint failover)

Module 13 Implement Process for Routing System Feedback to Development Teams

Module Overview

Module Overview

When you go shopping for a car, do you refuse to take it on a test drive? Likely not. No matter how much a salesperson hypes up a car, you have to feel for yourself how smoothly it drives and how easy it brakes. You also need to drive the car on a real road and in real conditions.

Software is the same way. Just deploying code into production and doing a health check is no longer good enough. We're now looking beyond what we used to consider "done", and instead, continue to monitor how it runs. Getting feedback about what happens after the software is deployed to stay competitive and make our system better is essential.

Feedback loops are the essence of any process improvement and DevOps is no exception. The goal of almost any process improvement initiative is to shorten and amplify feedback loops so necessary corrections can be continually made.

A feedback loop in general systems uses its output as one of its inputs. The right feedback loop must bear these characteristics – Faster, Relevant, Actionable and Accessible. Engineering teams need to set rules for acting on different feedback and own the complete code quality checked in by engineering teams. Feedback is fundamental not only to DevOps practice but throughout the SDLC process. A customized feedback loop and process is necessary for every organization to act as a control center to alter the course early when things go wrong. As you could guess by now, every feedback loop should at least allow teams to capture feedback, both technical and system feedback, raise the visibility of this feedback and allow the teams to take this feedback actionable.

Learning Objectives

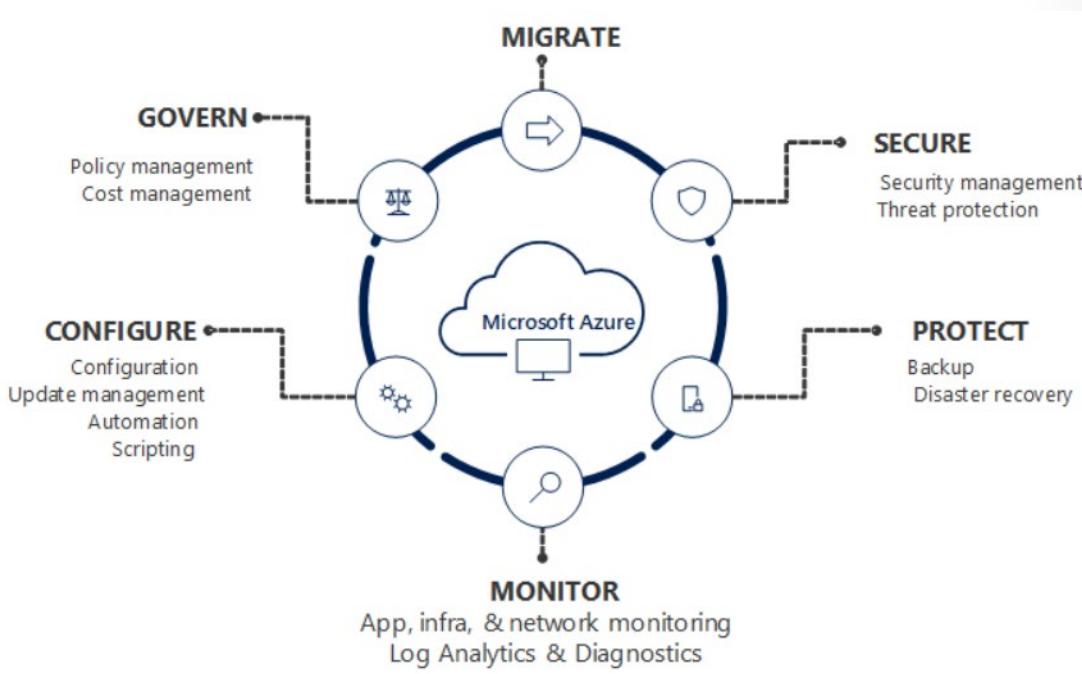
After completing this module, students will be able to:

- Implement tools to track system usage, feature usage, and flow
- Configure crash report integration for client applications

- Implement routing for client application crash report data
- Develop monitoring and status dashboards
- Integrate and configure ticketing systems with development team's work management

Implement Tools to Track System Usage, Feature Usage, and Flow

Introduction



Continuous monitoring refers to the process and technology required to incorporate monitoring across each phase of your DevOps and IT operations lifecycles. It helps to continuously ensure the health, performance, and reliability of your application and infrastructure as it moves from development to production. Continuous monitoring builds on the concepts of Continuous Integration and Continuous Deployment (CI/CD) which help you develop and deliver software faster and more reliably to provide continuous value to your users.

Azure Monitor¹ is the unified monitoring solution in Azure that provides full-stack observability across applications and infrastructure in the cloud and on-premises. It works seamlessly with **Visual Studio** and **Visual Studio Code**² during development and test and integrates with **Azure DevOps**³ for release management and work item management during deployment and operations. It even integrates across the ITSM and SIEM tools of your choice to help track issues and incidents within your existing IT processes.

This article describes specific steps for using Azure Monitor to enable continuous monitoring throughout your workflows. It includes links to other documentation that provides details on implementing different features.

¹ <https://docs.microsoft.com/en-us/azure/azure-monitor/overview>

² <https://visualstudio.microsoft.com/>

³ <https://docs.microsoft.com/en-us/azure/devops/user-guide/index>

Enable monitoring for all your applications⁴

In order to gain observability across your entire environment, you need to enable monitoring on all your web applications and services. This will allow you to easily visualize end-to-end transactions and connections across all the components.

- **Azure DevOps Projects⁵** give you a simplified experience with your existing code and Git repository, or choose from one of the sample applications to create a Continuous Integration (CI) and Continuous Delivery (CD) pipeline to Azure.
- **Continuous monitoring in your DevOps release pipeline⁶** allows you to gate or rollback your deployment based on monitoring data.
- **Status Monitor⁷** allows you to instrument a live .NET app on Windows with Azure Application Insights, without having to modify or redeploy your code.
- If you have access to the code for your application, then enable full monitoring with **Application Insights⁸** by installing the Azure Monitor Application Insights SDK for **.NET⁹, Java¹⁰, Node.js¹¹, or any other programming languages¹²**. This allows you to specify custom events, metrics, or page views that are relevant to your application and your business.

Enable monitoring for your entire infrastructure¹³

Applications are only as reliable as their underlying infrastructure. Having monitoring enabled across your entire infrastructure will help you achieve full observability and make it easier to discover a potential root cause when something fails. Azure Monitor helps you track the health and performance of your entire hybrid infrastructure including resources such as VMs, containers, storage, and network.

- You automatically get **platform metrics, activity logs and diagnostics logs¹⁴** from most of your Azure resources with no configuration.
- Enable deeper monitoring for VMs with **Azure Monitor for VMs¹⁵**.
- Enable deeper monitoring for AKS clusters with **Azure Monitor for containers¹⁶**.
- Add **monitoring solutions¹⁷** for different applications and services in your environment.

Infrastructure as code¹⁸ is the management of infrastructure in a descriptive model, using the same versioning as DevOps teams use for source code. It adds reliability and scalability to your environment and allows you to leverage similar processes that used to manage your applications.

- Use **Resource Manager templates¹⁹** to enable monitoring and configure alerts over a large set of resources.

⁴ <https://docs.microsoft.com/en-us/azure/azure-monitor/continuous-monitoring#enable-monitoring-for-all-your-applications>

⁵ <https://docs.microsoft.com/en-us/azure/devops-project/overview>

⁶ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-vsts-continuous-monitoring>

⁷ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-monitor-performance-live-website-now>

⁸ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-overview>

⁹ <https://docs.microsoft.com/en-us/azure/application-insights/quick-monitor-portal>

¹⁰ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-java-quick-start>

¹¹ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-nodejs-quick-start>

¹² <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-platforms>

¹³ <https://docs.microsoft.com/en-us/azure/azure-monitor/continuous-monitoring#enable-monitoring-for-your-entire-infrastructure>

¹⁴ <https://docs.microsoft.com/en-us/azure/azure-monitor/platform/data-sources>

¹⁵ <https://docs.microsoft.com/en-us/azure/azure-monitor/insights/vminsights-overview>

¹⁶ <https://docs.microsoft.com/en-us/azure/azure-monitor/insights/container-insights-overview>

¹⁷ <https://docs.microsoft.com/en-us/azure/azure-monitor/insights/solutions-inventory>

¹⁸ <https://docs.microsoft.com/en-us/azure/devops/learn/what-is-infrastructure-as-code>

¹⁹ <https://docs.microsoft.com/en-us/azure/azure-monitor/platform/template-workspace-configuration>

- Use **Azure Policy**²⁰ to enforce different rules over your resources. This ensures that those resources stay compliant with your corporate standards and service level agreements.

Combine resources in Azure Resource Groups²¹

A typical application on Azure today includes multiple resources such as VMs and App Services or microservices hosted on Cloud Services, AKS clusters, or Service Fabric. These applications frequently utilize dependencies like Event Hubs, Storage, SQL, and Service Bus.

- Combine resources in Azure Resource Groups to get full visibility across all your resources that make up your different applications. **Azure Monitor for Resource Groups**²² provides a simple way to keep track of the health and performance of your entire full-stack application and enables drilling down into respective components for any investigations or debugging.

Ensure quality through Continuous Deployment²³

Continuous Integration / Continuous Deployment allows you to automatically integrate and deploy code changes to your application based on the results of automated testing. It streamlines the deployment process and ensures the quality of any changes before they move into production.

- Use **Azure Pipelines**²⁴ to implement Continuous Deployment and automate your entire process from code commit to production based on your CI/CD tests.
- Use Quality Gates to integrate monitoring into your pre-deployment or post-deployment. This ensures that you are meeting the key health/performance metrics (KPIs) as your applications move from dev to production and any differences in the infrastructure environment or scale is not negatively impacting your KPIs.
- **Maintain separate monitoring instances**²⁵ between your different deployment environments such as Dev, Test, Canary, and Prod. This ensures that collected data is relevant across the associated applications and infrastructure. If you need to correlate data across environments, you can use **multi-resource charts in Metrics Explorer**²⁶ or create **cross-resource queries in Log Analytics**²⁷.

Create actionable alerts with actions²⁸

A critical aspect of monitoring is proactively notifying administrators of any current and predicted issues.

- Create **alerts in Azure Monitor**²⁹ based on logs and metrics to identify predictable failure states. You should have a goal of making all alerts actionable meaning that they represent actual critical conditions and seek to reduce false positives. Use **dynamic thresholds**³⁰ to automatically calculate baselines on metric data rather than defining your own static thresholds.
- Define actions for alerts to use the most effective means of notifying your administrators. Available **actions for notification**³¹ are SMS, e-mails, push notifications, or voice calls.

²⁰ <https://docs.microsoft.com/en-us/azure/governance/policy/overview>

²¹ <https://docs.microsoft.com/en-us/azure/azure-monitor/continuous-monitoring#combine-resources-in-azure-resource-groups>

²² <https://docs.microsoft.com/en-us/azure/azure-monitor/insights/resource-group-insights>

²³ <https://docs.microsoft.com/en-us/azure/azure-monitor/continuous-monitoring#ensure-quality-through-continuous-deployment>

²⁴ <https://docs.microsoft.com/en-us/azure/devops/pipelines>

²⁵ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-separate-resources>

²⁶ <https://docs.microsoft.com/en-us/azure/azure-monitor/platform/metrics-charts>

²⁷ <https://docs.microsoft.com/en-us/azure/azure-monitor/log-query/cross-workspace-query>

²⁸ <https://docs.microsoft.com/en-us/azure/azure-monitor/continuous-monitoring#create-actionable-alerts-with-actions>

²⁹ <https://docs.microsoft.com/en-us/azure/azure-monitor/platform/alerts-overview>

³⁰ <https://docs.microsoft.com/en-us/azure/azure-monitor/platform/alerts-dynamic-thresholds>

³¹ <https://docs.microsoft.com/en-us/azure/azure-monitor/platform/action-groups#create-an-action-group-by-using-the-azure-portal>

- Use more advanced actions to **connect to your ITSM tool**³² or other alert management systems through **webhooks**³³.
- Remediate situations identified in alerts as well with **Azure Automation runbooks**³⁴ or **Logic Apps**³⁵ that can be launched from an alert using webhooks.
- Use **autoscaling**³⁶ to dynamically increase and decrease your compute resources based on collected metrics.

Prepare dashboards and workbooks³⁷

Ensuring that your development and operations have access to the same telemetry and tools allows them to view patterns across your entire environment and minimize your Mean Time To Detect (MTTD) and Mean Time To Restore (MTTR).

- Prepare **custom dashboards**³⁸ based on common metrics and logs for the different roles in your organization. Dashboards can combine data from all Azure resources.
- Prepare **Workbooks**³⁹ to ensure knowledge sharing between development and operations. These could be prepared as dynamic reports with metric charts and log queries, or even as troubleshooting guides prepared by developers helping customer support or operations to handle basic problems.

Continuously optimize⁴⁰

Monitoring is one of the fundamental aspects of the popular Build-Measure-Learn philosophy, which recommends continuously tracking your KPIs and user behavior metrics and then striving to optimize them through planning iterations. Azure Monitor helps you collect metrics and logs relevant to your business and to add new data points in the next deployment as required.

- Use tools in Application Insights to **track end-user behavior and engagement**⁴¹.
- Use **Impact Analysis**⁴² to help you prioritize which areas to focus on to drive to important KPIs.

Azure Log Analytics

When you run at cloud scale, you need intelligent logging and monitoring tools that scale to your needs and provide insight on your data in real time. Azure Monitor is Microsoft's native cloud monitoring solution, Azure Monitor collects monitoring telemetry from a variety of on-premises and Azure sources. Azure Monitor provides Management tools, such as those in Azure Security Center and Azure Automation, also enables ingestion of custom log data to Azure. The service aggregates and stores this telemetry in a log data store that's optimised for cost and performance. With Azure Monitor you can analyse data, set up alerts, get end-to-end views of your applications, and use machine learning-driven insights to quickly identify and resolve problems.

³² <https://docs.microsoft.com/en-us/azure/azure-monitor/platform/itsmc-overview>

³³ <https://docs.microsoft.com/en-us/azure/azure-monitor/platform/activity-log-alerts-webhook>

³⁴ <https://docs.microsoft.com/en-us/azure/automation/automation-webhooks>

³⁵ <https://docs.microsoft.com/en-us/azure/connectors/custom-connectors/create-webhook-trigger>

³⁶ <https://docs.microsoft.com/en-us/azure/azure-monitor/learn/tutorial-autoscale-performance-schedule>

³⁷ <https://docs.microsoft.com/en-us/azure/azure-monitor/continuous-monitoring#prepare-dashboards-and-workbooks>

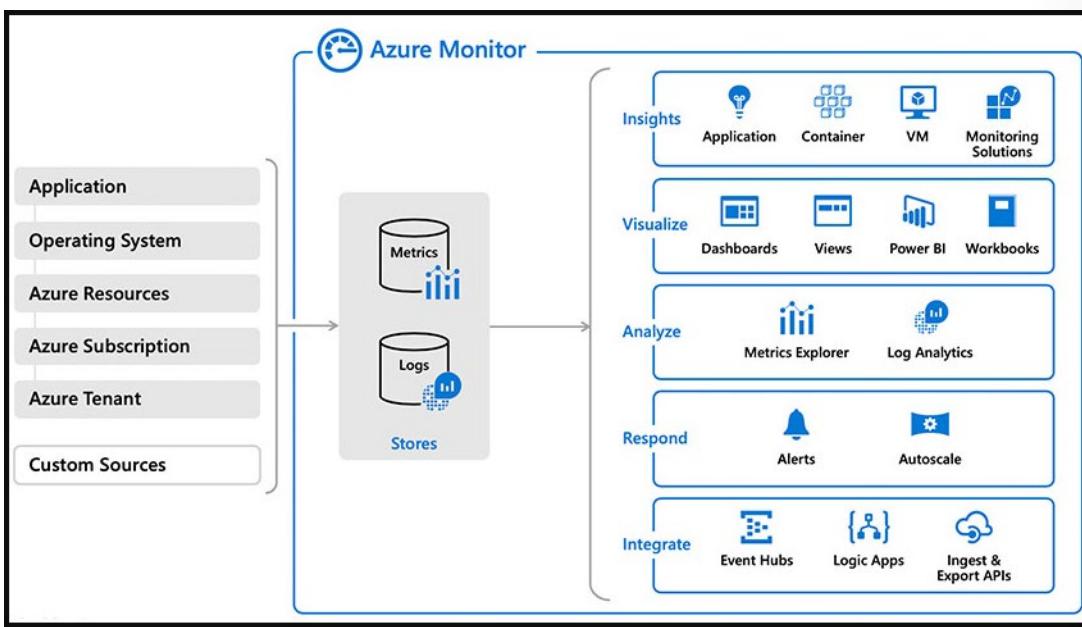
³⁸ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-tutorial-dashboards>

³⁹ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-usage-workbooks>

⁴⁰ <https://docs.microsoft.com/en-us/azure/azure-monitor/continuous-monitoring#continuously-optimize>

⁴¹ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-tutorial-users>

⁴² <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-usage-impact>



In this tutorial we'll focus on the Log Analytics part of Azure Monitor. We'll learn how to:

- Set up Log Analytics workspace
- Connect virtual machines into a log analytics workspace
- Configure Log Analytics workspace to collect custom performance counters
- Analyse the telemetry using Kusto Query Language

Getting Started

1. To follow along you'll need a resource group with one or more virtual machines that you have RDP access to.
2. Log into **Azure Shell**⁴³. Executing the command below will create a new resource group and create a new log analytics workspace. Take a note of the workspaceid of the log analytics workspace as we'll be using it again.

```
$ResourceGroup = "azwe-rg-devtest-logs-001"
$WorkspaceName = "azwe-devtest-logs-01"
$Location = "westeurope"

# List of solutions to enable
$Solutions = "CapacityPerformance", "LogManagement", "ChangeTracking",
"ProcessInvestigator"

# Create the resource group if needed
try {
    Get-AzResourceGroup -Name $ResourceGroup -ErrorAction Stop
} catch {
    New-AzResourceGroup -Name $ResourceGroup -Location $Location
}
```

⁴³ <http://shell.azure.com/powershell>

```
# Create the workspace
New-AzOperationalInsightsWorkspace -Location $Location -Name $WorkspaceName
-Sku Standard -ResourceGroupName $ResourceGroup

# List all solutions and their installation status
Get-AzOperationalInsightsIntelligencePacks -ResourceGroupName $Resource-
Group -WorkspaceName $WorkspaceName

# Add solutions
foreach ($solution in $Solutions) {
    Set-AzOperationalInsightsIntelligencePack -ResourceGroupName $Resource-
Group -WorkspaceName $WorkspaceName -IntelligencePackName $solution -Ena-
bled $true
}

# List enabled solutions
(Get-AzOperationalInsightsIntelligencePacks -ResourceGroupName $Resource-
Group -WorkspaceName $WorkspaceName).Where({$_.enabled -eq $true})

# Enable IIS Log Collection using agent
Enable-AzOperationalInsightsIISLogCollection -ResourceGroupName $Resource-
Group -WorkspaceName $WorkspaceName

# Windows Event
New-AzOperationalInsightsWindowsEventDataSource -ResourceGroupName $Re-
sourceGroup -WorkspaceName $WorkspaceName -EventLogName "Application"
-CollectErrors -CollectWarnings -Name "Example Application Event Log"
```

3. Retrieve the Log Analytics workspace secure key

```
Get-AzOperationalInsightsWorkspaceSharedKey ` 
    -ResourceGroupName azwe-rg-devtest-logs-001 ` 
    -Name azwe-devtest-logs-01
```

4. Map existing virtual machines with the Log Analytics workspace. The query below uses the workspaceid and workspace secret key of the log analytics workspace to install the Microsoft Enterprise Cloud Monitoring extension onto an existing VM.

```
$PublicSettings = @{"workspaceId" = "<myWorkspaceId>"}
$ProtectedSettings = @{"workspaceKey" = "<myWorkspaceKey>"}

Set-AzVMExtension -ExtensionName "Microsoft.EnterpriseCloud.Monitoring" ` 
    -ResourceGroupName "azwe-rg-devtest-logs-001" ` 
    -VMName "azsu-d-sql01-01" ` 
    -Publisher "Microsoft.EnterpriseCloud.Monitoring" ` 
    -ExtensionType "MicrosoftMonitoringAgent" ` 
    -TypeHandlerVersion 1.0 ` 
    -Settings $PublicSettings `
```

```
-ProtectedSettings $ProtectedSettings  
-Location westeurope
```

5. Run the script to configure the below listed performance counters to be collected from the virtual machine

```
#Login-AzureRmAccount

#Instance
#####
$instanceNameAll = "*"
$instanceNameTotal = '_Total'
#Objects
#####
$ObjectCache = "Cache"
$ObjectLogicalDisk = "LogicalDisk"
$ObjectMemory = "Memory"
$ObjectNetworkAdapter = "Network Adapter"
$ObjectNetworkInterface = "Network Interface"
$ObjectPagingFile = "Paging File"
$ObjectProcess = "Process"
$ObjectProcessorInformation = "Processor Information"
$ObjectProcessor = "Processor"

$ObjectSQLAgentAlerts = "SQLAgent:Alerts"
$ObjectSQLAgentJobs = "SQLAgent:Jobs"
$ObjectSQLAgentStatistics = "SQLAgent:Statistics"

$ObjectSQLServerAccessMethods = "SQLServer:Access Methods"
$ObjectSQLServerExecStatistics = "SQLServer:Exec Statistics"
$ObjectSQLServerLocks = "SQLServer:Locks"
$ObjectSQLServerSQLErrors = "SQLServer:SQL Errors"

$ObjectSystem = "System"

#Counters
#####
$CounterCache = "Copy Read Hits %"

$CounterLogicalDisk =
    "% Free Space" ` 
    , "Avg. Disk sec/Read" ` 
    , "Avg. Disk sec/Transfer" ` 
    , "Avg. Disk sec/Write" ` 
    , "Current Disk Queue Length" ` 
    , "Disk Read Bytes/sec" ` 
    , "Disk Reads/sec" ` 
    , "Disk Transfers/sec" ` 
    , "Disk Writes/sec"
```

```
$CounterMemory =
    "% Committed Bytes In Use" ` 
    , "Available MBytes" ` 
    , "Page Faults/sec" ` 
    , "Pages Input/sec" ` 
    , "Pages Output/sec" ` 
    , "Pool Nonpaged Bytes"

$CounterNetworkAdapter =
    "Bytes Received/sec" ` 
    , "Bytes Sent/sec"

$CounterNetworkInterface = "Bytes Total/sec"

$CounterPagingFile =
    "% Usage" ` 
    , "% Usage Peak"

$CounterProcess = "% Processor Time"

$CounterProcessorInformation =
    "% Interrupt Time" ` 
    , "Interrupts/sec"

$CounterProcessor = "% Processor Time"
$CounterProcessorTotal = "% Processor Time"

$CounterSQLAgentAlerts = "Activated alerts"
$CounterSQLAgentJobs = "Failed jobs"
$CounterSQLAgentStatistics = "SQL Server restarted"
$CounterSQLServerAccessMethods = "Table Lock Escalations/sec"
$CounterSQLServerExecStatistics = "Distributed Query"
$CounterSQLServerLocks = "Number of Deadlocks/sec"
$CounterSQLServerSQLErrors = "Errors/sec"

$CounterSystem = "Processor Queue Length"

#####
$global:number = 1 #Name parameter needs to be unique that why we will use
number ++ in fuction
#####

function AddPerfCounters ($PerfObject, $PerfCounters, $Instance)
{
    ForEach ($Counter in $PerfCounters)
    {
        New-AzureRmOperationalInsightsWindowsPerformanceCounterDataSource ` 
            -ResourceGroupName 'azwe-rg-devtest-logs-001' ` 
            -WorkspaceName 'azwe-devtest-logs-001' ` 
            -ObjectName $PerfObject ` 
            -InstanceName $Instance `
```

```
-CounterName $Counter  
-IntervalSeconds 10  
-Name "Windows Performance Counter $global:number"  
$global:number ++  
}  
}  
  
AddPerfCounters -PerfObject $ObjectLogicalDisk -PerfCounter $CounterLogicalDisk -Instance $InstanceNameAll  
AddPerfCounters -PerfObject $ObjectNetworkAdapter -PerfCounter $CounterNetworkAdapter -Instance $InstanceNameAll  
AddPerfCounters -PerfObject $ObjectNetworkInterface -PerfCounter $CounterNetworkInterface -Instance $InstanceNameAll  
AddPerfCounters -PerfObject $ObjectPagingFile -PerfCounter $CounterPagingFile -Instance $InstanceNameAll  
AddPerfCounters -PerfObject $ObjectProcess -PerfCounter $CounterProcess -Instance $InstanceNameAll  
AddPerfCounters -PerfObject $ObjectProcessorInformation -PerfCounter $CounterProcessorInformation -Instance $InstanceNameAll  
AddPerfCounters -PerfObject $ObjectProcessor -PerfCounter $CounterProcessor -Instance $InstanceNameAll  
AddPerfCounters -PerfObject $ObjectProcessor -PerfCounter $CounterProcessorTotal -Instance $InstanceNameTotal  
AddPerfCounters -PerfObject $ObjectSQLAgentAlerts -PerfCounter $CounterSQLAgentAlerts -Instance $InstanceNameAll  
AddPerfCounters -PerfObject $ObjectSQLAgentJobs -PerfCounter $CounterSQLAgentJobs -Instance $InstanceNameAll  
AddPerfCounters -PerfObject $ObjectSQLAgentStatistics -PerfCounter $CounterSQLAgentStatistics -Instance $InstanceNameAll  
AddPerfCounters -PerfObject $ObjectSQLServerAccessMethods -PerfCounter $CounterSQLServerAccessMethods -Instance $InstanceNameAll  
AddPerfCounters -PerfObject $ObjectSQLServerExecStatistics -PerfCounter $CounterSQLServerExecStatistics -Instance $InstanceNameAll  
AddPerfCounters -PerfObject $ObjectSQLServerLocks -PerfCounter $CounterSQLServerLocks -Instance $InstanceNameAll  
AddPerfCounters -PerfObject $ObjectSQLServerSQLErrors -PerfCounter $CounterSQLServerSQLErrors -Instance $InstanceNameAll  
AddPerfCounters -PerfObject $ObjectSystem -PerfCounter $CounterSystem -Instance $InstanceNameAll  
AddPerfCounters -PerfObject $ObjectMemory -PerfCounter $CounterMemory -Instance $InstanceNameAll  
AddPerfCounters -PerfObject $ObjectCache -PerfCounter $CounterCache -Instance $InstanceNameAll
```

6. In order to generate some interesting performance statistics. Download **HeavyLoad utility**⁴⁴ (a free load testing utility) and run this on the virtual machine to simulate high CPU, Memory and IOPS consumption.

⁴⁴ <https://www.jam-software.com/heavylload/>

Summary

So far, we've created a log analytics workspace in a resource group. The log analytics workspace has been configured to collect performance counters, event logs and IIS Logs. A virtual machine has been mapped to the log analytics workspace using the Microsoft Enterprise cloud monitoring extension. HeavyLoad has been used to simulate high CPU, memory and IOPS on the virtual machine.

How to

1. Log into **Azure Portal**⁴⁵ and navigate to the log analytics workspace. From the left blade in the log analytics workspace click Logs. This will open up the Logs window, ready for you to start exploring all the datapoints captured into the workspace.
2. To query the logs we'll need to use the Kusto Query Language. Run the query below to list the last heartbeat of each machine connected to the log analytics workspace

```
// Last heartbeat of each computer  
// Show the last heartbeat sent by each computer  
Heartbeat  
| summarize arg_max(TimeGenerated, *) by Computer
```

3. Show a list of all distinct counters being captured

```
// What data is being collected?  
// List the collected performance counters and object types (Process,  
Memory, Processor...)  
Perf  
| summarize by ObjectName, CounterName
```

4. Show a count of the data points collected in the last 24 hours. In the result below, you can see we have 66M data points that we are able to query against in near real time to analyse and correlate insights

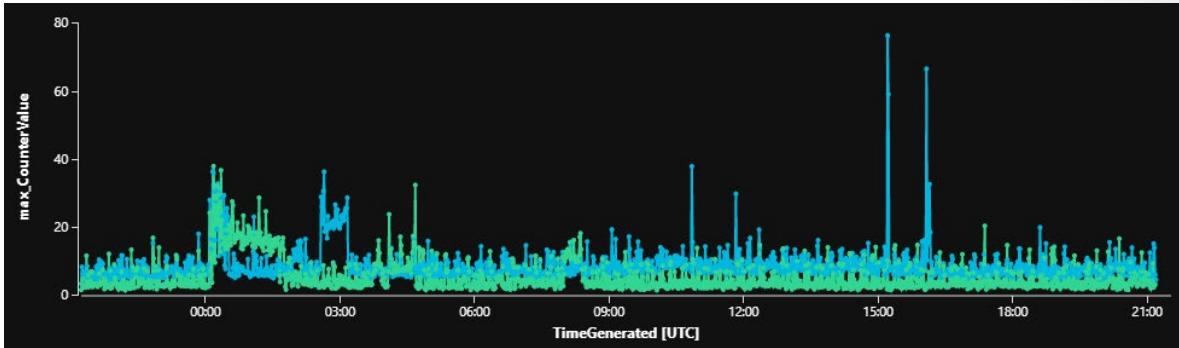
The screenshot shows a table with one row of data. The columns are labeled 'Count' and 'Value'. The 'Value' column contains the value '88,685,260' in yellow, indicating it is the current total count of data points.

Count	Value
>	88,685,260

5. Run the query below to generate the max CPU Utilization trend over the last 24 hours, aggregated at a granularity of 1 min. Render the data as timechart.

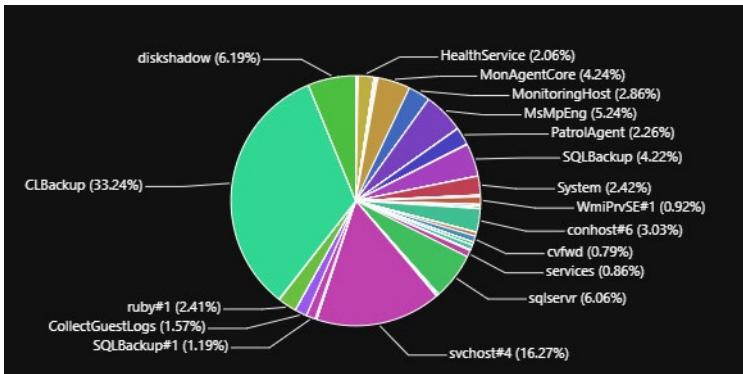
```
Perf  
| where ObjectName == "Processor" and InstanceName == "_Total"  
| summarize max(CounterValue) by Computer, bin(TimeGenerated, 1m)  
| render timechart
```

⁴⁵ <https://portal.azure.com>



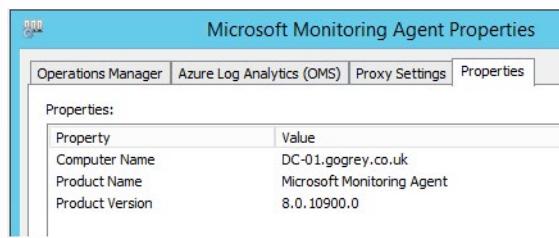
- Run the query below to see all the processes running on that machine that are contributing to the CPU Utilization. Render the data in a pie chart.

```
Perf  
| where ObjectName contains "process"  
    and InstanceName !in ("_Total", "Idle")  
    and CounterName == "% Processor Time"  
| summarize avg(CounterValue) by InstanceName, CounterName, bin(TimeGenerated, 1m)  
| render piechart
```



How it works

- Log Analytics works by running the Microsoft Monitoring Agent service on the machine. The service locally captures and buffers the events and pushes them securely out to the Log Analytics workspace in Azure.
- Log into the virtual machine and navigate to the C:\Program Files\Microsoft Monitoring Agent\MMA and open control panel. This will show you the details of the log analytics workspace connected. You also have the option of adding multiple log analytics workspaces to publish the log data into multiple workspaces.



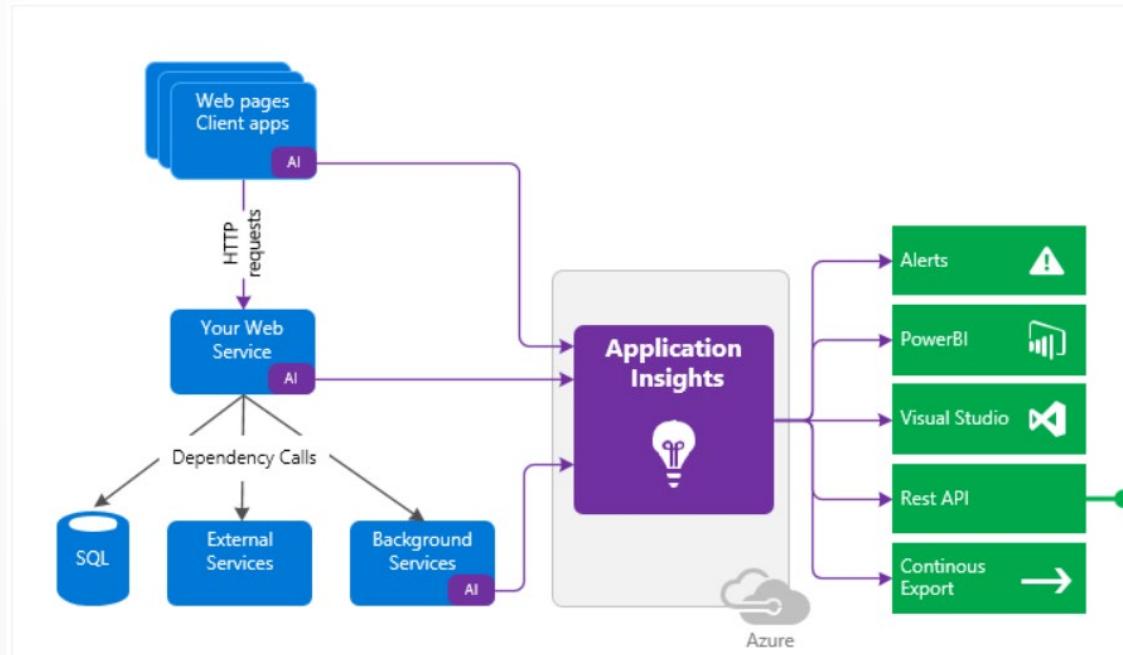
There's more

This tutorial has introduced you to the basic concepts of Log Analytics and how to get started with the basics. We've only scratched the surface of what's possible with Log Analytics. I would encourage you to try out the advanced tutorials available for Log Analytics on [Microsoft Docs](#)⁴⁶

How Does Application Insights Work

You install a small instrumentation package in your application, and set up an Application Insights resource in the Microsoft Azure portal. The instrumentation monitors your app and sends telemetry data to the portal. (The application can run anywhere - it doesn't have to be hosted in Azure.)

You can instrument not only the web service application, but also any background components, and the JavaScript in the web pages themselves.



In addition, you can pull in telemetry from the host environments such as performance counters, Azure diagnostics, or Docker logs. You can also set up web tests that periodically send synthetic requests to your web service.

All these telemetry streams are integrated in the Azure portal, where you can apply powerful analytic and search tools to the raw data.

⁴⁶ <https://docs.microsoft.com/en-us/azure/azure-monitor/>

What's the overhead?⁴⁷

The impact on your app's performance is very small. Tracking calls are non-blocking, and are batched and sent in a separate thread.

What does Application Insights monitor?⁴⁸

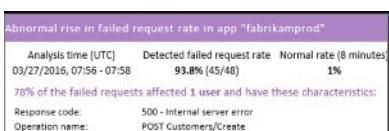
Application Insights is aimed at the development team, to help you understand how your app is performing and how it's being used. It monitors:

- Request rates, response times, and failure rates - Find out which pages are most popular, at what times of day, and where your users are. See which pages perform best. If your response times and failure rates go high when there are more requests, then perhaps you have a resourcing problem.
- Dependency rates, response times, and failure rates - Find out whether external services are slowing you down.
- Exceptions - Analyse the aggregated statistics, or pick specific instances and drill into the stack trace and related requests. Both server and browser exceptions are reported.
- Page views and load performance - reported by your users' browsers.
- AJAX calls from web pages - rates, response times, and failure rates.
- User and session counts.
- Performance counters from your Windows or Linux server machines, such as CPU, memory, and network usage.
- Host diagnostics from Docker or Azure.
- Diagnostic trace logs from your app - so that you can correlate trace events with requests.
- Custom events and metrics that you write yourself in the client or server code, to track business events such as items sold or games won.

Where do I see my telemetry?

There are plenty of ways to explore your data. Check out this article for more information - **Smart detection and manual alerts⁴⁹**

Automatic alerts adapt to your app's normal patterns of telemetry and trigger when there's something outside the usual pattern. You can also **set alerts⁵⁰** on particular levels of custom or standard metrics.



⁴⁷ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-overview?toc=/azure/azure-monitor/toc.json#whats-the-overhead>

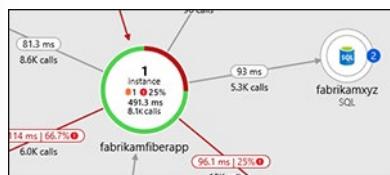
⁴⁸ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-overview?toc=/azure/azure-monitor/toc.json#what-does-application-insights-monitor>

⁴⁹ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-proactive-diagnostics>

⁵⁰ <https://docs.microsoft.com/en-us/azure/azure-monitor/app/alerts>

Application map⁵¹

The components of your app, with key metrics and alerts.



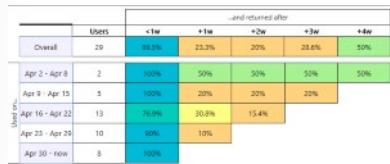
Profiler⁵²

Inspect the execution profiles of sampled requests.



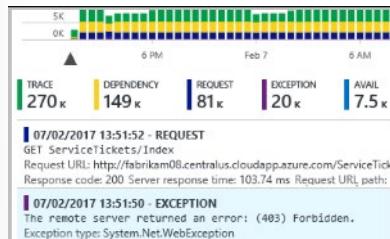
Usage analysis⁵³

Analyze user segmentation and retention.



Diagnostic search for instance data⁵⁴

Search and filter events such as requests, exceptions, dependency calls, log traces, and page views.



Metrics Explorer for aggregated data

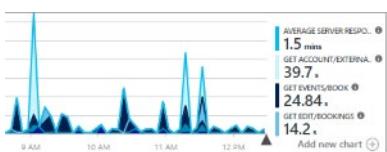
Explore, filter, and segment aggregated data such as rates of requests, failures, and exceptions; response times, page load times.

⁵¹ <https://docs.microsoft.com/en-us/azure/azure-monitor/app/app-map>

⁵² <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-profiler>

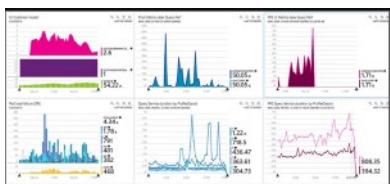
⁵³ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-usage-overview>

⁵⁴ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-diagnostic-search>



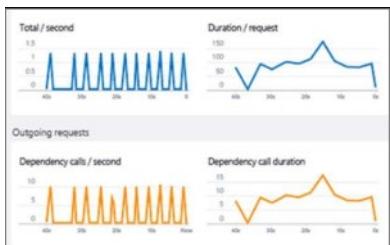
Dashboards

Mash up data from multiple resources and share with others. Great for multi-component applications, and for continuous display in the team room.



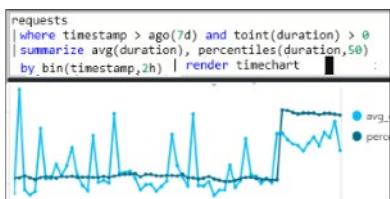
Live Metrics Stream

When you deploy a new build, watch these near-real-time performance indicators to make sure everything works as expected.



Analytics

Answer tough questions about your app's performance and usage by using this powerful query language.



Visual Studio

See performance data in the code. Go to code from stack traces.

The screenshot shows a debugger interface with the following details:

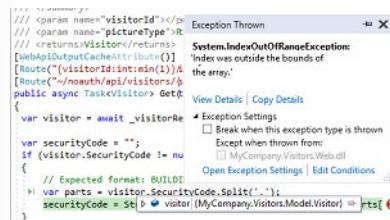
- Exception Details:** A tooltip for a red circled exception message: "System.OperationCanceledException : no such op".
- Stack Trace:** Shows the call stack:

```
at Application2.Controllers.HomeController.About()
at lambda_method
at System.Web.Mvc.ActionMethodDispatcher.Execute()
```
- Code:** The About() method definition:

```
public ActionResult About()
{
    ViewBag.Message = "Your application desc
if (DateTime.Now.IsDaylightSavingTime())
```

Snapshot debugger

Debug snapshots sampled from live operations, with parameter values.



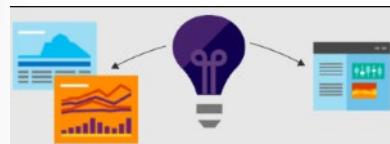
Power BI

Integrate usage metrics with other business intelligence.



REST API

Write code to run queries over your metrics and raw data.



Continuous export

Bulk export of raw data to storage as soon as it arrives.

```
HTTP/1.1 200
Content-Type: application/json; charset=utf-8

{
  "Tables": [
    {
      "TableName": "Table_0",
      "Columns": [
        {
          "ColumnName": "Count",
          "DataType": "Int64",
        }
      ]
    }
  ]
}
```

How do I use Application Insights

Monitor

Install Application Insights in your app, set up **availability web tests**⁵⁵, and:

- Set up a **dashboard**⁵⁶ for your team room to keep an eye on load, responsiveness, and the performance of your dependencies, page loads, and AJAX calls.
- Discover which are the slowest and most failing requests.
- Watch **Live Stream**⁵⁷ when you deploy a new release, to know immediately about any degradation.

Detect, Diagnose

When you receive an alert or discover a problem:

- Assess how many users are affected.
- Correlate failures with exceptions, dependency calls and traces.
- Examine profiler, snapshots, stack dumps, and trace logs.

Build, Measure, Learn

Measure the effectiveness⁵⁸ of each new feature that you deploy.

- Plan to measure how customers use new UX or business features.
- Write custom telemetry into your code.
- Base the next development cycle on hard evidence from your telemetry.

Get started

Application Insights is one of the many services hosted within Microsoft Azure, and telemetry is sent there for analysis and presentation. So before you do anything else, you'll need a subscription to **Microsoft Azure**⁵⁹. It's free to sign up, and if you choose the basic **pricing plan**⁶⁰ of Application Insights, there's no charge until your application has grown to have substantial usage. If your organization already has a subscription, they could add your Microsoft account to it.

⁵⁵ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-monitor-web-app-availability>

⁵⁶ <https://docs.microsoft.com/en-us/azure/azure-monitor/app/app-insights-dashboards>

⁵⁷ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-live-stream>

⁵⁸ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-usage-overview>

⁵⁹ <https://azure.com/>

⁶⁰ <https://azure.microsoft.com/pricing/details/application-insights/>

There are several ways to get started. Begin with whichever works best for you. You can add the others later.

At run time

Instrument your web app on the server. Avoids any update to the code. You need admin access to your server.

- **IIS on-premises or on a VM**⁶¹
- **Azure web app or VM**⁶²
- **J2EE**⁶³

At development time

Add Application Insights to your code. Allows you to write custom telemetry and to instrument back-end and desktop apps.

- **Visual Studio**⁶⁴ 2013 update 2 or later.
- **Java**⁶⁵
- Node.js
- **Other platforms**⁶⁶
- **Instrument your web pages**⁶⁷ for page view, AJAX and other client-side telemetry.
- **Analyze mobile app usage**⁶⁸ by integrating with Visual Studio App Center.
- **Availability tests**⁶⁹ - ping your website regularly from our servers.

Application Insights

Application performance management (APM) is a discipline that includes all the tools and activities involved in observing how software and hardware are performing. These tools present that performance information in a form product owners and software development teams can use to make decisions. Application Insights is Microsoft Azure native APM Tool that is cross platform and specialises in providing a rich & intelligent performance management toolset for Azure hosted web apps.

In this tutorial we'll learn how to get started with App Insights. We'll cover,

- Adding App Insights to your dotnet core app
- Accessing App Insights from within the Azure Portal

Getting Started

1. To add Application Insights to your ASP.NET website, you need to:

⁶¹ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-monitor-performance-live-website-now>

⁶² <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-monitor-performance-live-website-now>

⁶³ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-java-live>

⁶⁴ <https://docs.microsoft.com/en-us/azure/azure-monitor/app/asp-net>

⁶⁵ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-java-get-started>

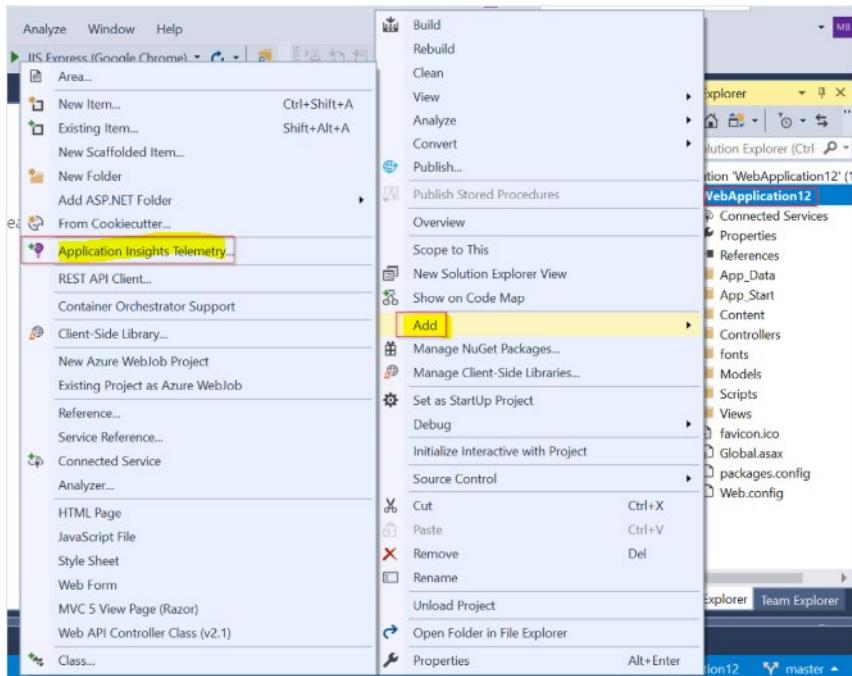
⁶⁶ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-platforms>

⁶⁷ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-javascript>

⁶⁸ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-mobile-center-quickstart>

⁶⁹ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-monitor-web-app-availability>

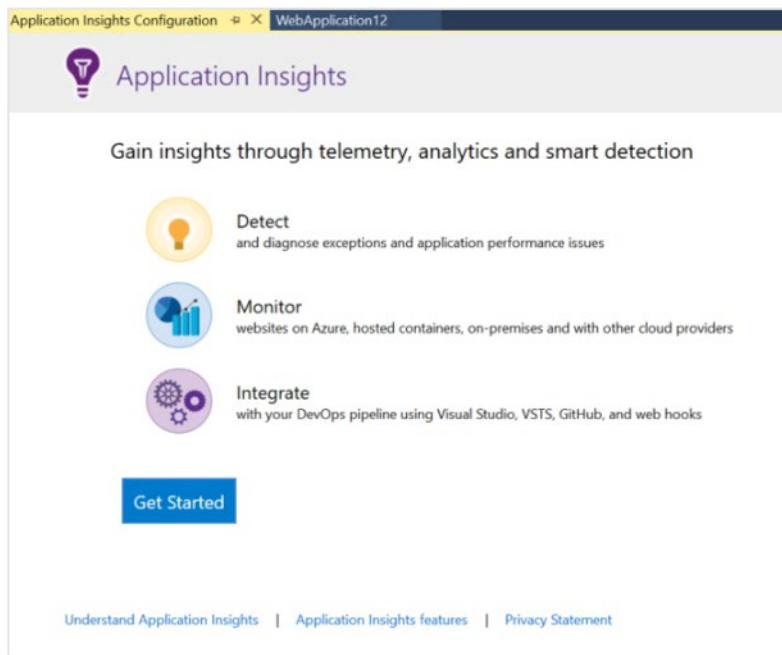
- Install Visual Studio 2019 for Windows with the following workloads:
 - ASP.NET and web development (Do not uncheck the optional components)
2. In Visual Studio create a new dotnet core project. Right click the project and from the context menu



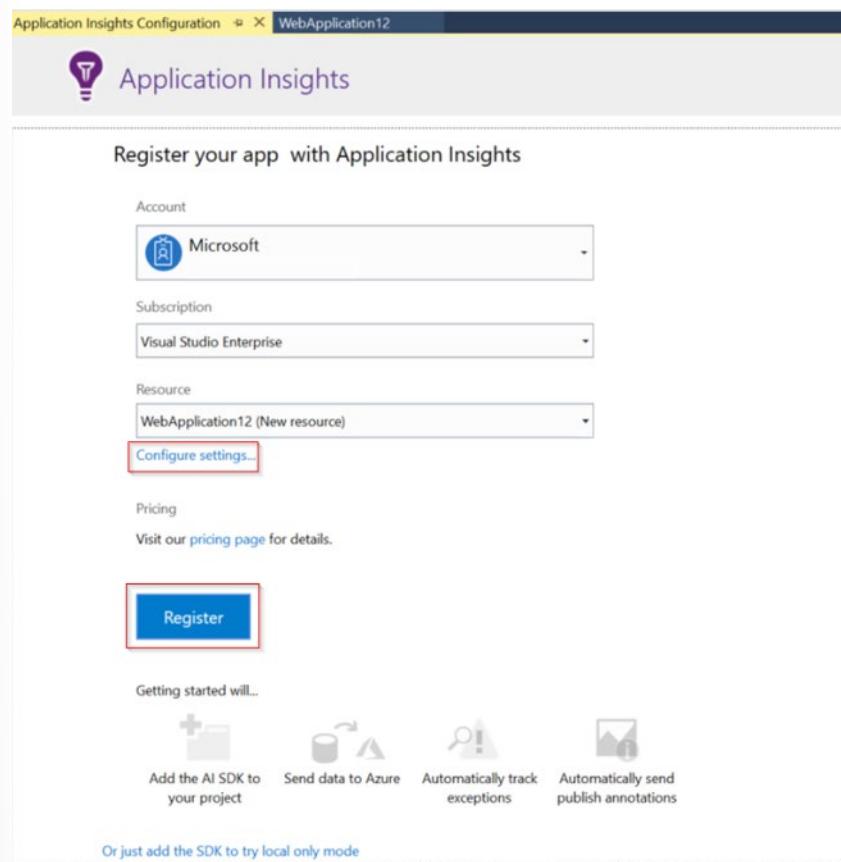
select AddApplication Insights Telemetry

(Depending on your Application Insights SDK version you may be prompted to upgrade to the latest SDK release. If prompted, select Update SDK.)

3. From the Application Insights configuration screen, click Get started to start setting up App Insights



4. Choose to set up a new resource group and select the location where you want the telemetry data to be persisted.

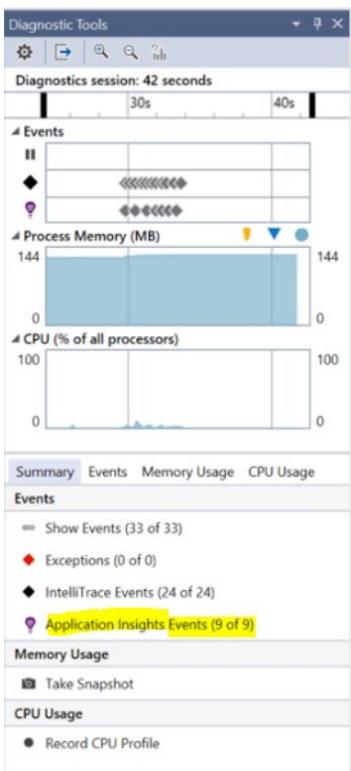


Summary

So far we've added App Insights in a dotnet core application. The Application Insights getting started experience gives you the ability to create a new resource group in the desired location where the App Insights instance gets created. The instrumentation key for the app insights instance is injected into the application configuration automatically.

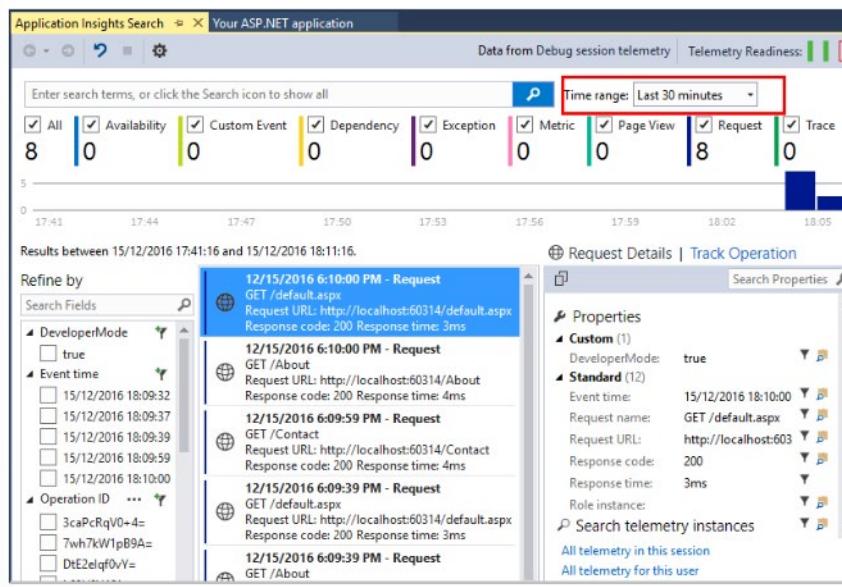
How to do it

1. Run your app with F5. Open different pages to generate some telemetry. In Visual Studio, you will see a count of the events that have been logged.



2. You can see your telemetry either in Visual Studio or in the Application Insights web portal. Search telemetry in Visual Studio to help you debug your app. Monitor performance and usage in the web portal when your system is live. In Visual Studio, to view Application Insights data. Select Solution Explorer > Connected Services > right-click Application Insights, and then click Search Live Telemetry.

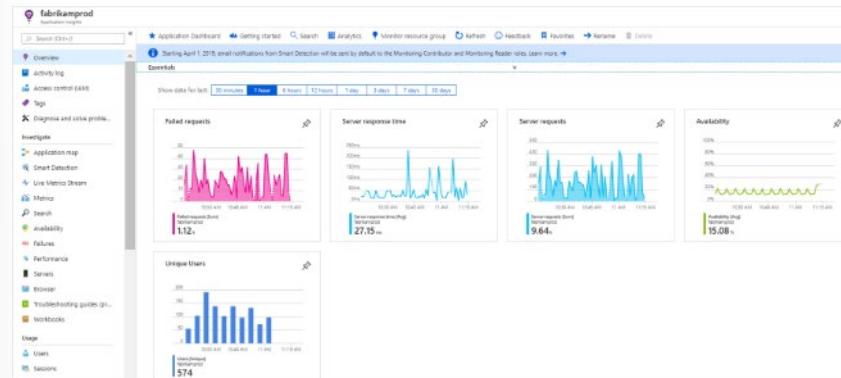
In the Visual Studio Application Insights Search window, you will see the data from your application for telemetry generated in the server side of your app. Experiment with the filters, and click any event to see more detail.



3. You can also see telemetry in the Application Insights web portal (unless you chose to install only the SDK). The portal has more charts, analytic tools, and cross-component views than Visual Studio. The portal also provides alerts.

Open your Application Insights resource. Either sign into the Azure portal and find it there, or select Solution Explorer > Connected Services > right-click Application Insights > Open Application Insights Portal and let it take you there.

The portal opens on a view of the telemetry from your app.



How it works

Application Insights configures an unique key (called Applnights Key) in your application. This key is used by the Application Insights SDK to identify the Azure App Insights workspace the telemetry data needs to be uploaded into. The SDK and the key are merely used to pump the telemetry data points out of your application. The heavy lifting of data correlation, analysis and insights is done within Azure.

There's more

In this tutorial we learned how to get started by adding Application Insights into your dotnet core application. App Insights offers a wide range of features, You can learn more about these at **Start Monitoring Your ASP.NET Core Web Application⁷⁰**.

⁷⁰ <https://docs.microsoft.com/en-us/azure/azure-monitor/learn/dotnetcore-quick-start>

Implement Routing for Mobile Application Crash Report Data

App center diagnostics

App Center Diagnostics is a cloud service that helps developers monitor the health of an application, delivering the data needed to understand what happens when an app fails during testing or in the wild. The App Center Diagnostics SDK collects information about crashes and errors in your apps and uploads them to the App Center portal for analysis by the development team - eliminating the guesswork about what really happened in the app when it failed.

Crashes

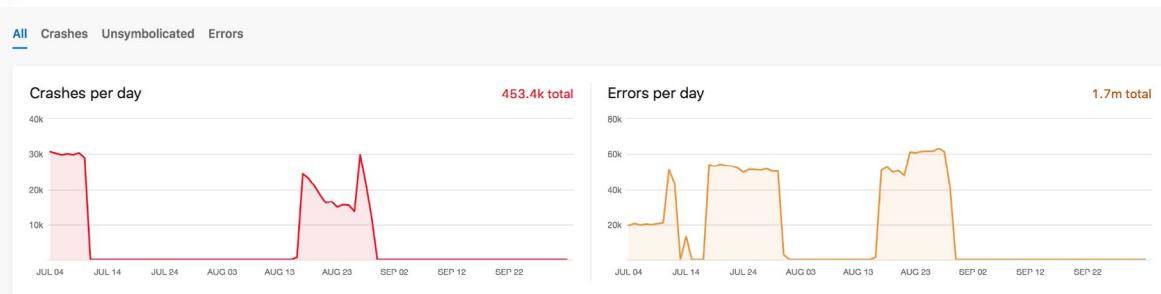
Crashes are what happens when a runtime exception occurs from an unexpected event that terminates the app. These are errors not handled by a try/catch block. When a crash occurs, App Center Crashes records the state of the app and device and automatically generates a crash log. These logs contain valuable information to help you fix the crash.

Crash and Errors Analytics

In App Center Diagnostics, you can view analytics data generated automatically by App Center to understand when a crash or error occurs in your app.

By default, App Center displays an app's crashes and errors per day in a side by side view.

Using the top-left tabs, drill down into Crashes and Errors. When you do this, the left chart indicates the number of crashes/errors per day, and the right chart shows the number of affected users. Filter the charts by app version, time frame and status for a more focused view.



Grouping

App Center Diagnostics groups crashes and errors by similarities, such as reason for the issue and where the issue occurred in the app. For each crash and error group, App Center displays the line of code that failed, the class or method name, file name, line number, crash or error type and message for you to better understand these groups at a glance. Select a group to view more information, and access a list of detailed issues reports and logs. This allows you to dive even deeper and use our feature set to better understand your app's behavior during a crash or an error.

Crash groups						
<input type="checkbox"/> Group		Count	Users	Version	Status	Last Crash ↓
<input type="checkbox"/> #277: CRLCrashROPPage.m line 42 SIGBUS		↳ 15	R 4	5.0.0 (60)	Open	4 days ago
<input type="checkbox"/> #268: CRLCrashNULL.m line 37 SIGSEGV	☰	↳ 7	R 2	5.0.0 (60)	Open	4 days ago
<input type="checkbox"/> #270: CRLCrashStackGuard.m line 38 SIGBUS		↳ 3	R 1	5.0.0 (60)	Open	5 days ago
<input type="checkbox"/> #304: CRLCrashNSLog.m line 41 SIGSEGV		↳ 1	R 1	5.0.0 (1529...)	Open	1 week ago
<input type="checkbox"/> #303: CRLCrashROPPage.m line 42 SIGBUS		↳ 1	R 1	5.0.0 (1529...)	Open	1 week ago
<input type="checkbox"/> #286: CRLCrashPrivInst.m line 52 SIGILL		↳ 9	R 3	5.0.0 (60)	Open	2 weeks ago
<input type="checkbox"/> #288: CRLCrashReleasedObject.m line 51 SIGSEGV		↳ 7	R 2	5.0.0 (60)	Open	2 weeks ago

Attachments

In the App Center Diagnostics UI, you can attach, view and download one binary and one text attachment to your crash reports.

You can learn how to add attachments to your crash reports by reading the SDK Crashes documentation for your [Android](#)⁷¹, [iOS](#)⁷², [macOS](#)⁷³, [React Native](#)⁷⁴, [Xamarin](#)⁷⁵, and [Apache Cordova](#)⁷⁶ apps.

To view and download the attachments, select a crash group, a specific device report and then click on the attachments tab.

⁷¹ <https://docs.microsoft.com/en-us/appcenter/sdk/crashes/android#add-attachments-to-a-crash-report>

⁷² <https://docs.microsoft.com/en-us/appcenter/sdk/crashes/ios#add-attachments-to-a-crash-report>

⁷³ <https://docs.microsoft.com/en-us/appcenter/sdk/crashes/macos#add-attachments-to-a-crash-report>

⁷⁴ <https://docs.microsoft.com/en-us/appcenter/sdk/crashes/react-native#add-attachments-to-a-crash-report>

⁷⁵ <https://docs.microsoft.com/en-us/appcenter/sdk/crashes/xamarin#add-attachments-to-a-crash-report>

⁷⁶ <https://docs.microsoft.com/en-us/appcenter/sdk/crashes/cordova#add-attachments-to-a-crash-report>

The screenshot shows a crash report interface. At the top, it displays a crash group (#277) for an iPhone 8 Plus running iOS 11.4 (15F79). The version is 5.0.0 (60), and the occurrence is Jun 13, 4:59 PM. There are download and close buttons. Below this, a list of individual crash events is shown, each with a timestamp and device information. The events are:

- iPhone 8 Plus (A1864/A1898/A1899) 1m ago
- iPhone 6s 1m ago
- iPhone 7 (A1778) 4d ago
- iPhone 7 (A1778) 4d ago
- iPhone 7 (A1778) 5d ago
- iPhone 7 (A1778) 2w ago

Below the event list, there are tabs for OVERVIEW, THREADS, EVENTS, and ATTACHMENTS. The ATTACHMENTS tab is selected, showing two attachments:

- ATTACHED BINARY: A file named 144x144.png (10.52KB)
- ATTACHED TEXT: A text file containing instructions for using CrashProbe, including steps for cloning the repository, opening the project in Xcode, integrating the SDK, building the app, and comparing symbolicated crash reports with available data.

Events Before a Crash

Track events leading up to a crash to capture useful information about the state of your app.

To define a custom event, check out our **SDK Documentation⁷⁷** for **Android⁷⁸**, **iOS⁷⁹**, **React Native⁸⁰**, **Xamarin⁸¹**, **UWP⁸²** and **macOS⁸³**.

To view events before a crash, select a crash group, a specific device report, and then click on the events tab.

⁷⁷ <https://docs.microsoft.com/en-us/appcenter/sdk/index>

⁷⁸ <https://docs.microsoft.com/en-us/appcenter/sdk/analytics/android>

⁷⁹ <https://docs.microsoft.com/en-us/appcenter/sdk/analytics/ios>

⁸⁰ <https://docs.microsoft.com/en-us/appcenter/sdk/analytics/react-native>

⁸¹ <https://docs.microsoft.com/en-us/appcenter/sdk/analytics/xamarin>

⁸² <https://docs.microsoft.com/en-us/appcenter/sdk/getting-started/uwp>

⁸³ <https://docs.microsoft.com/en-us/appcenter/sdk/analytics/macos>

Configure Alerts

Stay on top of your crashes by configuring your App Center app definition settings to send an email when a new crash group is created. To configure these alerts:

1. Log into App Center and select your app.
2. In the left menu, navigate to **Settings**.
3. Click on **Email Notifications**.
4. Select the box next to Crashes.

Create a Bug Tracker

You can integrate third party bug tracker tools with App Center to stay informed and better manage your crashes. Read the [bug tracker documentation](#)⁸⁴ to learn how to get started.

App Center has bug tracker integration for the crashes service. Users can be quickly informed about critical App Center events within the tools that you use regularly in your day to day flow for a seamless experience. App Center supports bug trackers like Jira, Azure DevOps (formerly Visual Studio Team Services (VSTS)), and GitHub. Users need to have manager or developer permissions to be able to create and configure the bug tracker.

For Azure DevOps (formerly VSTS):

1. Login with your Azure DevOps credentials and click Accept when prompted on app authorization.
2. Select which Azure DevOps projects to integrate the bug tracker with and click Next.

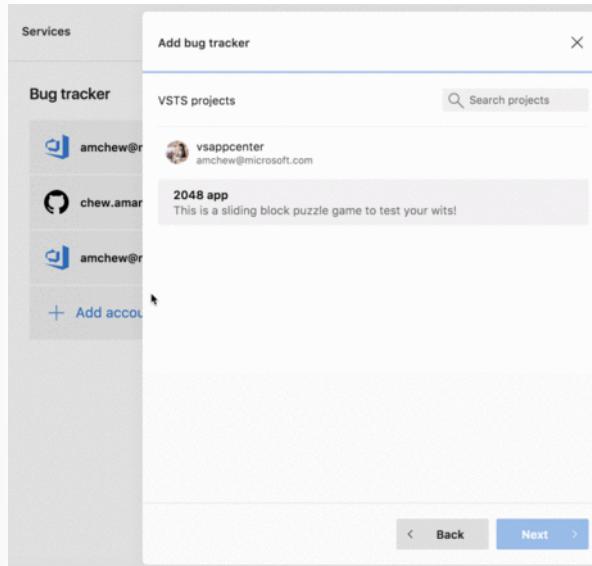
⁸⁴ <https://docs.microsoft.com/en-us/appcenter/dashboard/bugtracker/index>

3. Under Add bug tracker, fill in the fields for Number of crashes, Area and Default Payload, and click Add:

- Number of crashes is a threshold you can set for the minimum number of crashes to happen in a crash group before a ticket is created in Azure DevOps.
- Default payload is an optional field to fill in for use in work items. For example,

{"System.IterationPath": "Area\Iteration 1", "System.AssignedTo": "Fabrikam"}.

Please refer to the **work item types API⁸⁵** for additional information.



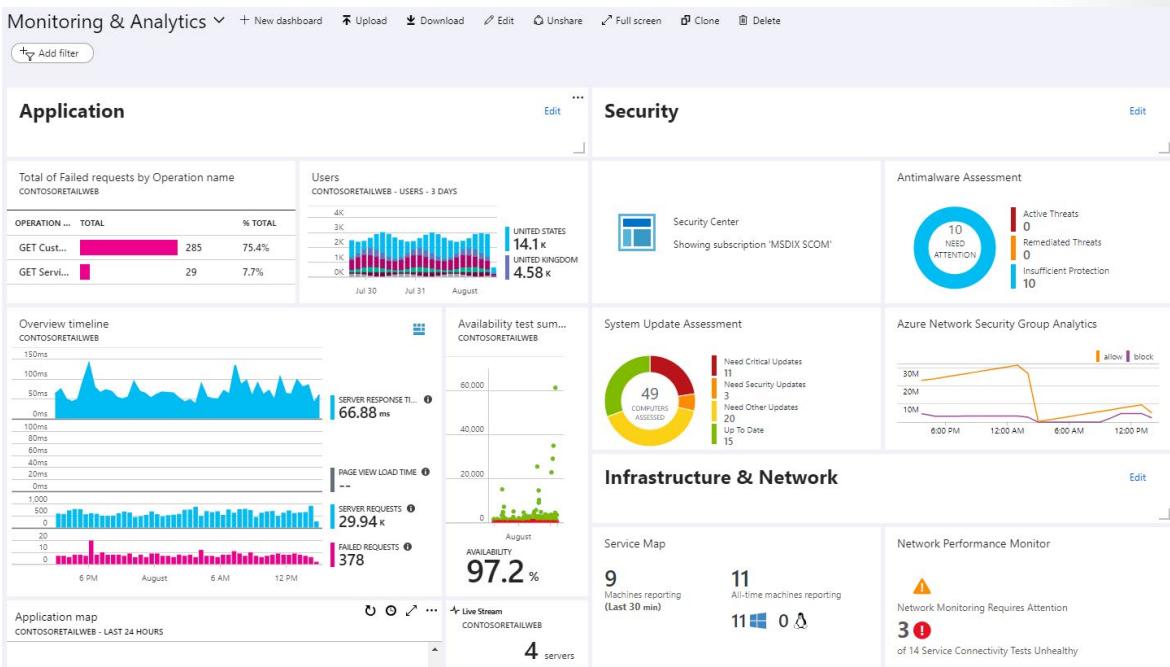
⁸⁵ <https://docs.microsoft.com/en-us/rest/api/vsts/wit/work%20item%20types>

Develop Monitoring and Status Dashboards

Azure Dashboards

Visualizations such as charts and graphs can help you analyze your monitoring data to drill-down on issues and identify patterns. Depending on the tool you use, you may also have the option to share visualizations with other users inside and outside of your organization.

Azure dashboards⁸⁶ are the primary dashboarding technology for Azure. They're particularly useful in providing single pane of glass over your Azure infrastructure and services allowing you to quickly identify important issues.



Advantages

- Deep integration into Azure. Visualizations can be pinned to dashboards from multiple Azure pages including metrics analytics, log analytics, and Application Insights.
- Supports both metrics and logs.
- Combine data from multiple sources including output from **Metrics explorer⁸⁷**, **Log Analytics queries⁸⁸**, and **maps⁸⁹** and **availability⁹⁰** in Application Insights.
- Option for personal or shared dashboards. Integrated with Azure **role based authentication (RBAC)⁹¹**.
- Automatic refresh. Metrics refresh depends on time range with minimum of five minutes. Logs refresh at one minute.

⁸⁶ <https://docs.microsoft.com/en-us/azure/azure-portal/azure-portal-dashboards>

⁸⁷ <https://docs.microsoft.com/en-us/azure/azure-monitor/platform/metrics-charts>

⁸⁸ <https://docs.microsoft.com/en-us/azure/azure-monitor/log-query/log-query-overview>

⁸⁹ <https://docs.microsoft.com/en-us/azure/azure-monitor/app/app-map>

⁹⁰ <https://docs.microsoft.com/en-us/azure/azure-monitor/visualizations?toc=/azure/azure-monitor/toc.json>

⁹¹ <https://docs.microsoft.com/en-us/azure/role-based-access-control/overview>

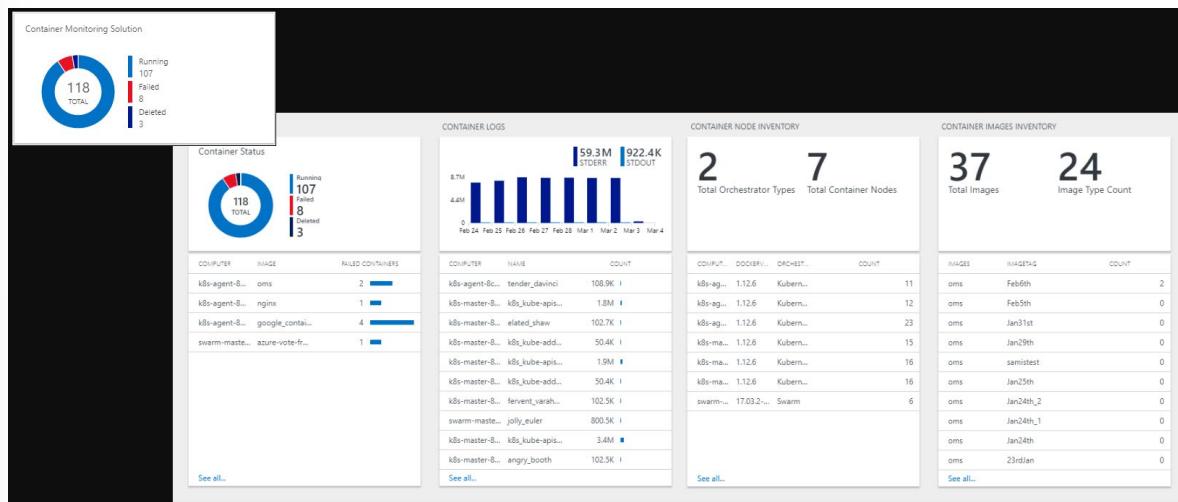
- Parametrized metrics dashboards with timestamp and custom parameters.
- Flexible layout options.
- Full screen mode.

Limitations

- Limited control over log visualizations with no support for data tables. Total number of data series is limited to 10 with further data series grouped under an *other* bucket.
- No custom parameters support for log charts.
- Log charts are limited to last 30 days.
- Log charts can only be pinned to shared dashboards.
- No interactivity with dashboard data.
- Limited contextual drill-down.

View Designer in Azure Monitor

View Designer in Azure Monitor⁹² allow you to create custom visualizations with log data. They are used by **monitoring solutions**⁹³ to present the data they collect.



Advantages

- Rich visualizations for log data.
- Export and import views to transfer them to other resource groups and subscriptions.
- Integrates into Log Analytics management model with workspaces and monitoring solutions.
- **Filters**⁹⁴ for custom parameters.
- Interactive, supports multi-level drill-in (view that drills into another view)

⁹² <https://docs.microsoft.com/en-us/azure/log-analytics/log-analytics-view-designer>

⁹³ <https://docs.microsoft.com/en-us/azure/azure-monitor/insights/solutions>

⁹⁴ <https://docs.microsoft.com/en-us/azure/azure-monitor/platform/view-designer-filters>

Limitations

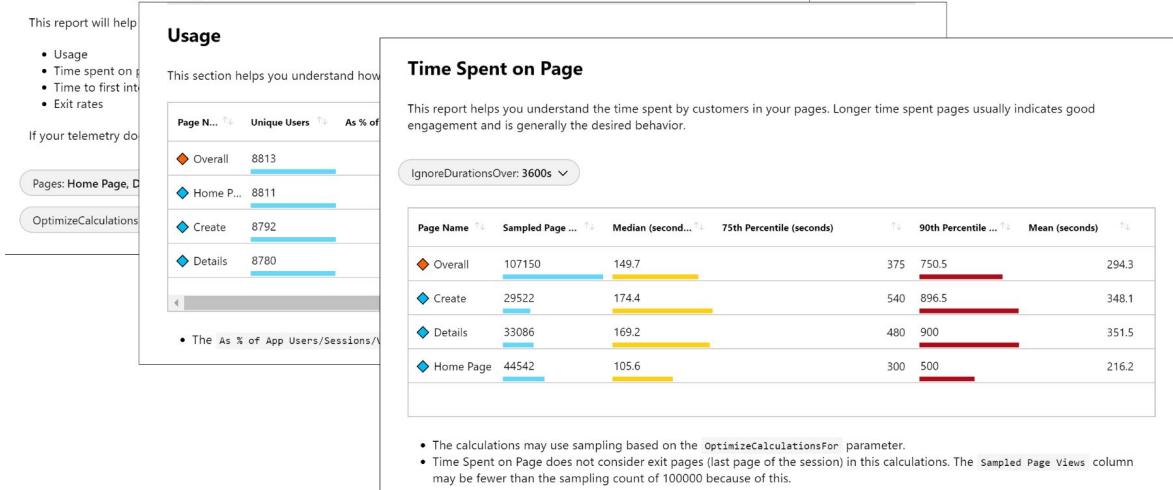
- Supports logs but not metrics.
- No personal views. Available to all users with access to the workspace.
- No automatic refresh.
- Limited layout options.
- No support for querying across multiple workspaces or Application Insights applications.
- Queries are limited in response size to 8MB and query execution time of 110 seconds.

Azure Monitor Workbooks

Workbooks⁹⁵ are interactive documents that provide deep insights into your data, investigation, and collaboration inside the team. Specific examples where workbooks are useful are troubleshooting guides and incident postmortem.

Analysis of Page Views

Page views correspond to user activity in your app. Understanding how your users interact with your pages will give you good insights into what is working in your app and what aspects need improvements.



Advantages

- Supports both metrics and logs.
- Supports parameters enabling interactive reports where selecting an element in a table will dynamically update associated charts and visualizations.
- Document-like flow.
- Option for personal or shared workbooks.
- Easy, collaborative-friendly authoring experience.
- Templates support public GitHub-based template gallery.

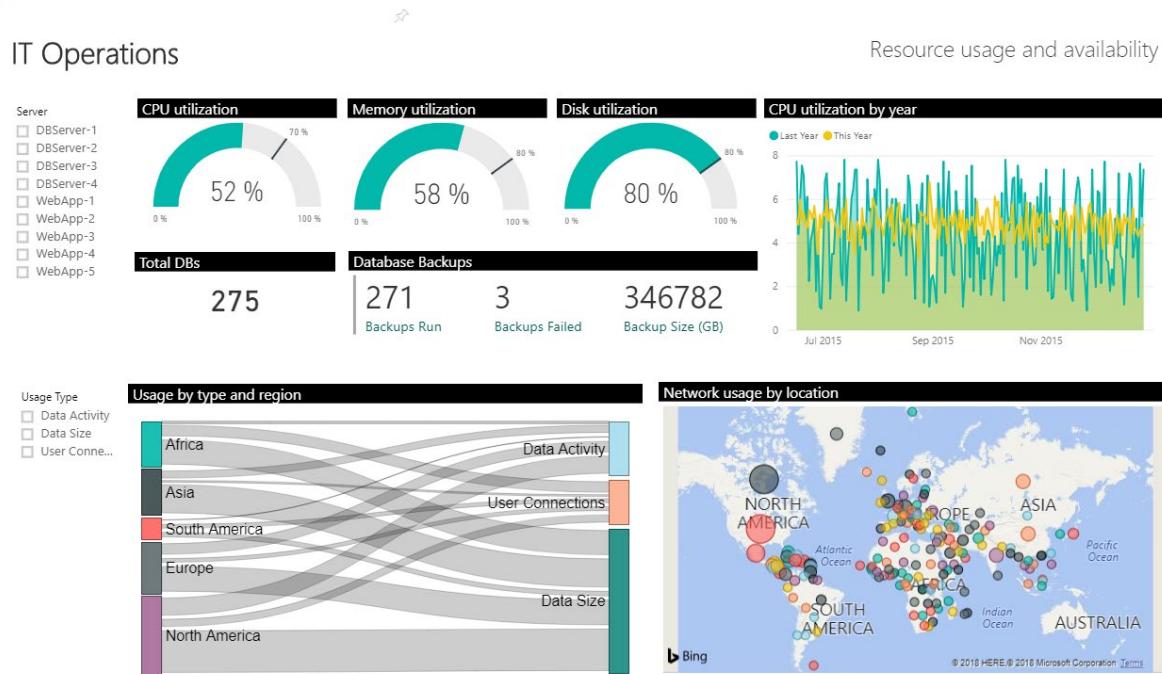
⁹⁵ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-usage-workbooks>

Limitations

- No automatic refresh.
- No dense layout like dashboards, which make workbooks less useful as a single pane of glass. Intended more for providing deeper insights.

Power BI

Power BI⁹⁶ is particularly useful for creating business-centric dashboards and reports, as well as reports analyzing long-term KPI trends. You can import the results of a log query⁹⁷ into a Power BI dataset so you can take advantage of its features such as combining data from different sources and sharing reports on the web and mobile devices.



Advantages

- Rich visualizations
- Extensive interactivity including zoom-in and cross-filtering
- Easy to share throughout your organization
- Integration with other data from multiple data sources
- Better performance with results cached in a cube

Limitations

- Supports logs but not metrics
- No Azure integration. Can't manage dashboards and models through Azure Resource Manager

⁹⁶ <https://powerbi.microsoft.com/documentation/powerbi-service-get-started/>

⁹⁷ <https://docs.microsoft.com/en-us/azure/log-analytics/log-analytics-powerbi>

- Query results need to be imported into Power BI model to configure. Limitation on result size and refresh
- Limited data refresh of eight times per day

Grafana

Grafana⁹⁸ is an open platform that excels in operational dashboards. It's particularly useful for detecting and isolating and triaging operational incidents. You can add **Grafana Azure Monitor data source plugin**⁹⁹ to your Azure subscription to have it visualize your Azure metrics data.



Advantages

- Rich visualizations.
- Rich ecosystem of datasources.
- Data interactivity including zoom in.
- Supports parameters.

Limitations

- Supports metrics but not logs.
- No Azure integration. Can't manage dashboards and models through Azure Resource Manager.
- Cost to support additional Grafana infrastructure or additional cost for Grafana Cloud.

Build Your Own Custom Application

You can access data in log and metric data in Azure Monitor through their API using any REST client, which allows you to build your own custom websites and applications.

⁹⁸ <https://grafana.com/>

⁹⁹ <https://docs.microsoft.com/en-us/azure/azure-monitor/platform/grafana-plugin>

Advantages

- Complete flexibility in UI, visualization, interactivity, and features.
- Combine metrics and log data with other data sources.

Disadvantages

- Significant engineering effort required.

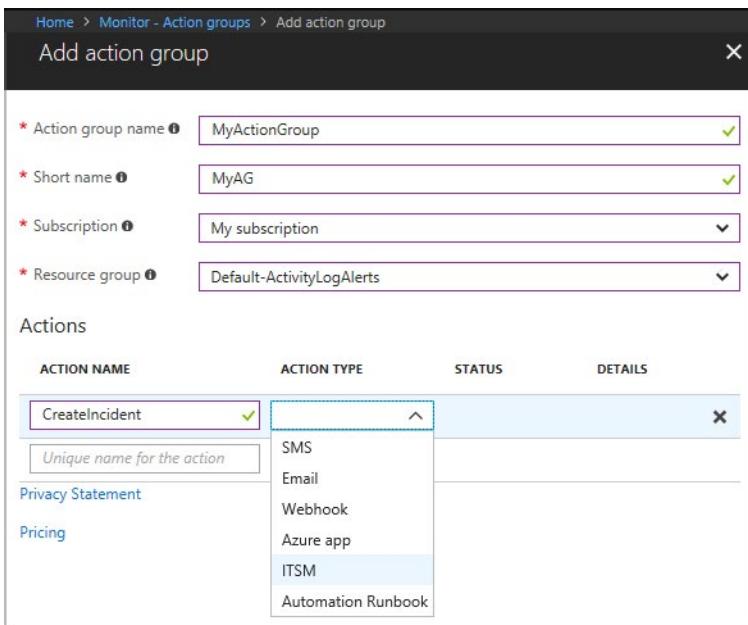
Integrate and Configure Ticketing Systems

IT Service Management Connector

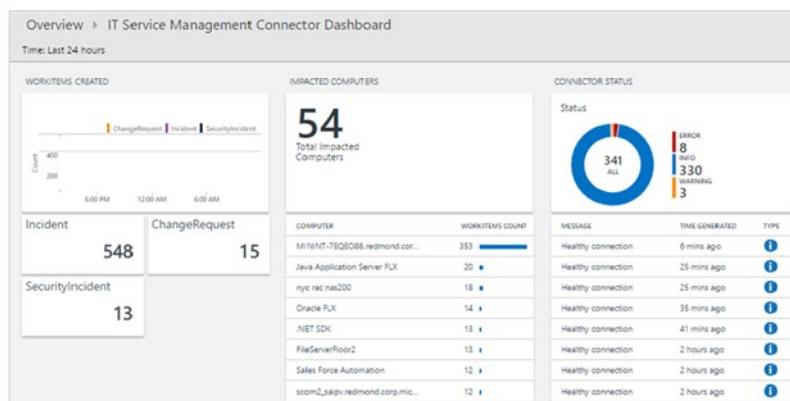
Customers use Azure monitoring tools to identify, analyze and troubleshoot issues. However, the work items related to an issue is typically stored in an ITSM tool. Instead of having to go back and forth between your ITSM tool and Azure monitoring tools, customers can now get all the information they need in one place. ITSMC will improve the troubleshooting experience and reduce the time it takes to resolve issues. IT Service Management Connector (ITSMC) for Azure provides bi-directional integration between Azure monitoring tools and your ITSM tools – ServiceNow, Provance, Cherwell, and System Center Service Manager. Specifically, you can use ITSMC to:

- Create or update work-items (Event, Alert, Incident) in the ITSM tools based on Azure alerts (Activity Log Alerts, Near Real-Time metric alerts and Log Analytics alerts)
- Pull the Incident and Change Request data from ITSM tools into Azure Log Analytics.

You can setup ITSMC by following the steps in our documentation. Once set up, you can send Azure alerts to ITSM tool using the ITSM action in Action groups.



You can also view your incident and change request data in Log Analytics to perform trend analysis or correlate it against operational data.



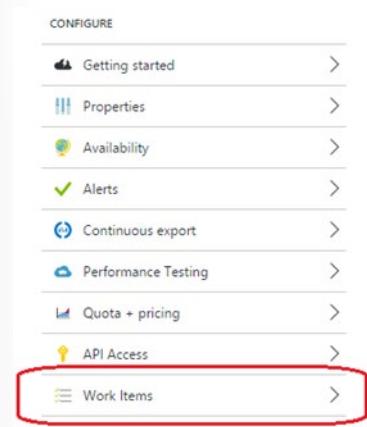
For more information, refer to [Connect Azure to ITSM tools using IT Service Management Connector](#)¹⁰⁰

Integration with Azure Boards

Work item integration functionality allows you to easily create work items in VSTS that have relevant Application Insights data embedded in them. It is very straightforward to configure this association and begin creating work items (this process should only take a minute or two).

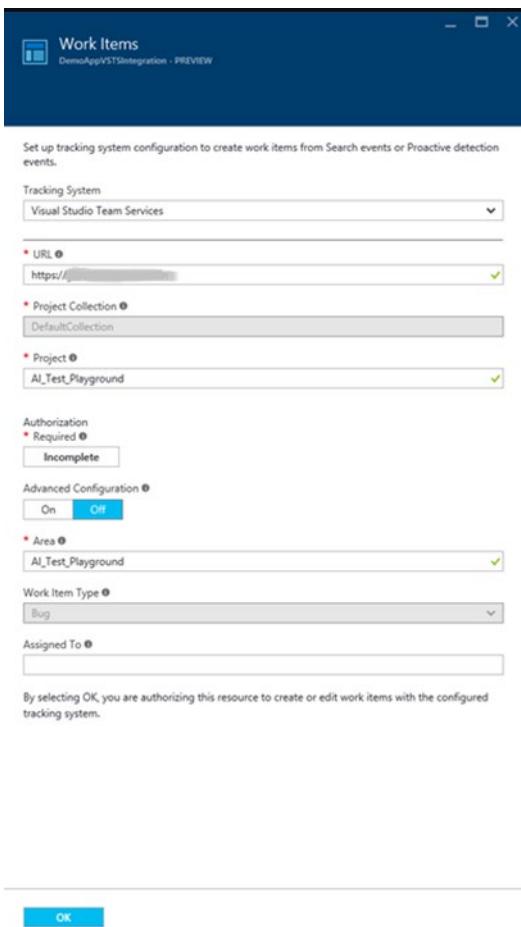
Configuring work item integration

To configure work item integration for an Application Insights resource, simply navigate to your settings blade for that resource. You'll note that there is now a new item in the "Configure" section of the settings blade that says "Work Items."



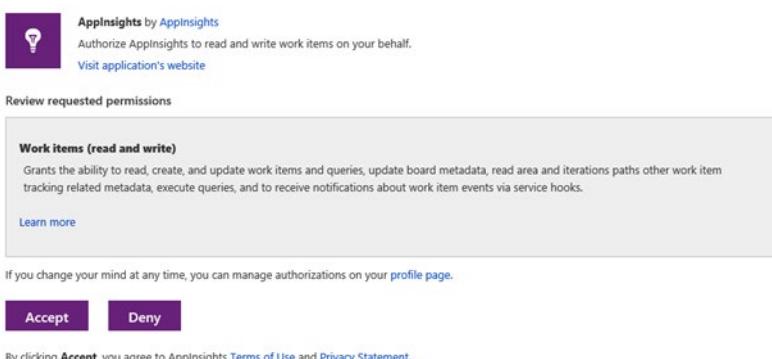
Click on this, and the configuration blade for work items will open. All you need to do is fill out the information about the VSTS system to which you want to connect, along with the project where you want to write your work items:

¹⁰⁰ <https://docs.microsoft.com/en-gb/azure/azure-monitor/platform/itsmc-overview>



Once that information is in place, you can click on the Authorization button, where you will be redirected to authorize access in your selected VSTS system so that work items can be written there:

Authorize application

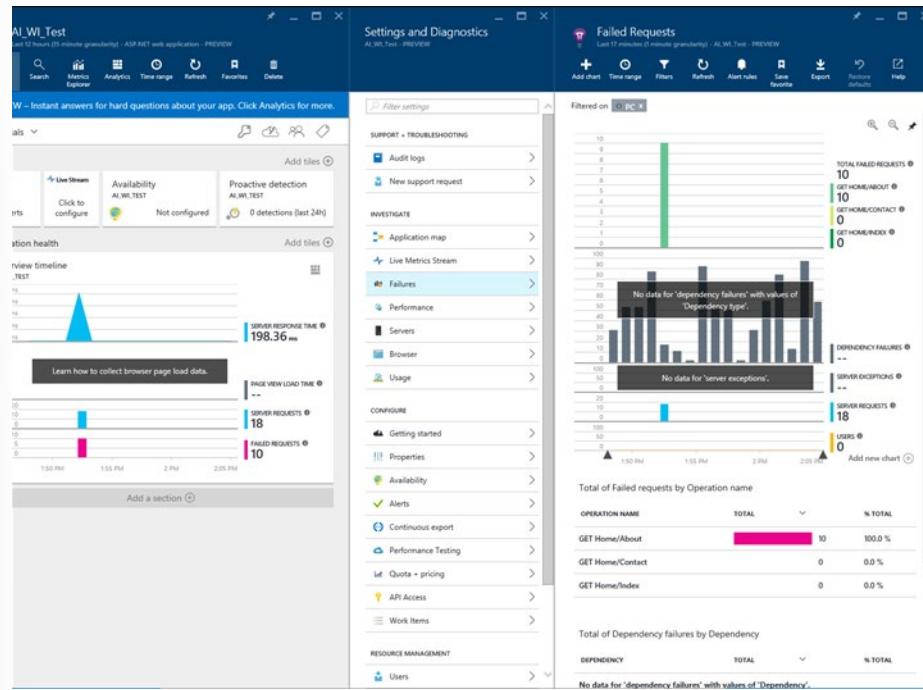


Once you've completed the authorization process, you can set defaults for "area path" and "assigned to." Only area path is required (if you haven't set up specific area paths in your project, that's ok. Just use the name of the project, as it is the top-level area path. Click OK, and assuming you've entered everything correctly, you'll see a message stating "Validation Successful" and the blade will close. You're now ready to start creating work items!

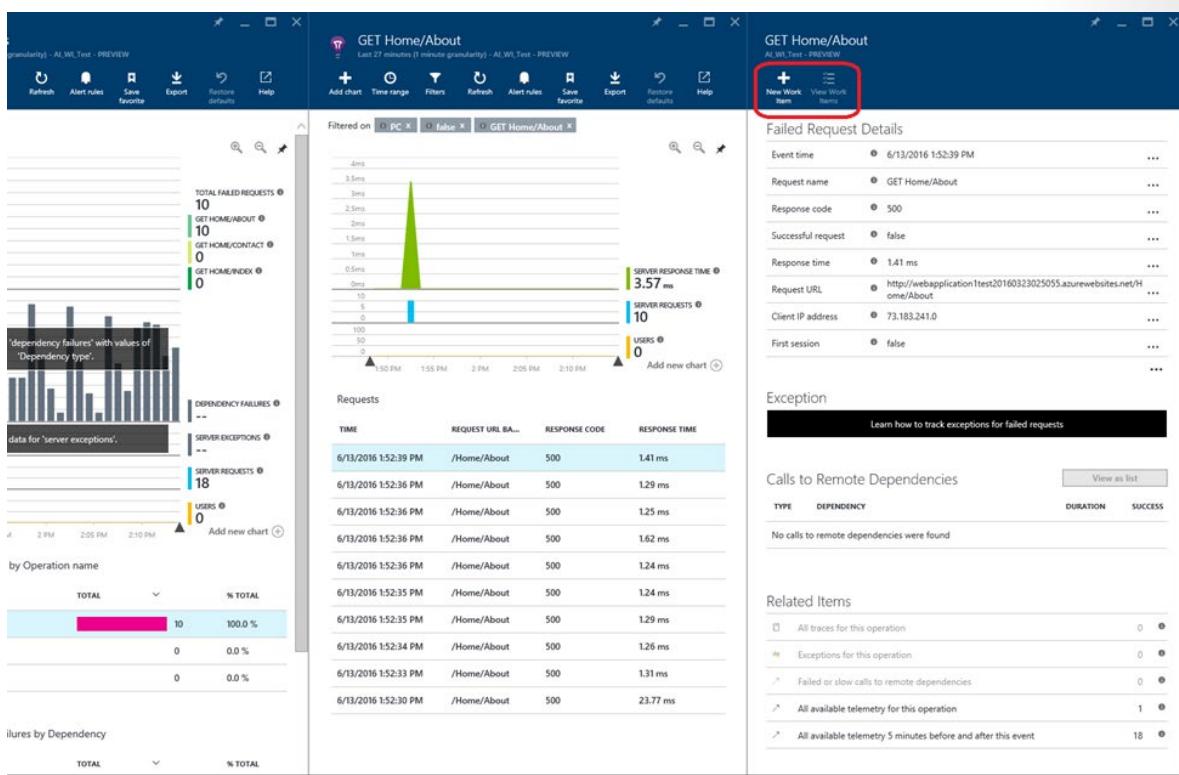
Creating work items

Creating work items from Application Insights is very easy. There are currently two locations from where you can create work items: Proactive detection and individual instances of activity (i.e., exceptions, failures, requests, etc.). I will show you a simple example of the latter, but the functionality is identical in either case.

In this example, I'm looking at a test web app that I've published to Azure. I've started to drill into the activity for this app by looking at the Failures blade (but we could also get to this same information through the Search button or the Metrics Explorer):



We can see that I have a number of exceptions that fired when the user clicked on the Home/About tab on this web app. If I drill into this group of exceptions, I can see the list, and then choose an individual exception:



Looking at the detail blade for this exception, we see that there are now two buttons available at the top of the blade that read "New Work Item" and "View Work Items." To create a work item, I simply click on the first of these buttons, and it opens the new work item blade:

The screenshot shows two overlapping windows. The left window is titled 'GET Home/About' and is labeled 'AI_WI_Test - PREVIEW'. It displays 'Failed Request Details' for a request made at 6/13/2016 1:52:39 PM. The details include: Request name: GET Home/About, Response code: 500, Successful request: false, Response time: 1.41 ms, Request URL: http://webapplication1test20160323025055.azurewebsites.net/Home/About, Client IP address: 73.183.241.0, and First session: false. Below this is an 'Exception' section with a link to 'Learn how to track exceptions for failed requests'. The right window is titled 'New Work Item' and is also labeled 'PREVIEW'. It shows a form for creating a work item in 'Visual Studio Team Services'. The fields are: Title (set to 'Issue with GET Home/About'), Area (set to 'AI_Test_Playground'), Work Item Type (set to 'Bug'), and Assigned To (empty). A 'Details' section contains captured information: 'Authenticated or anonymous traffic: Anonymous user traffic', 'City: Houston', 'Client IP address: 73.183.241.0', 'Continent: North America', 'Country or region: United States', and 'Device type: PC'. At the bottom of the right window is a blue 'OK' button.

As you can see, just about everything you need in your average scenario has been filled out for you. The default values for "area path" and "assigned to" that you designated in the initial configuration are set, and all of the detail information we have available for this exception has been added to the details field. You can override the title, area path and assigned to fields in this blade if you wish, or you can add to the captured details. When you're ready to create your work item, just click on the "OK" button, and your work item will be written to VSTS.

Viewing work items

Once you've created one or more work items in Application Insights, you can easily view them in VSTS. If you are in the Azure portal, the detail blade for the event associated to the work item(s) will have the "View Work Items" button enabled. To see the list, simply click the button:

The screenshot shows two main sections. On the left, under 'GET Home/About' (AI_WI_Test - PREVIEW), there's a 'Failed Request Details' table with columns for Event time, Request name, Response code, Successful request, Response time, Request URL, Client IP address, and First session. On the right, under 'View Work Items' (AI_WI_Test - PREVIEW), there's a list of work items with columns for ID and Title, showing entries for 'Issue with GET Home/About' and 'Issue with GET Home/About 2'.

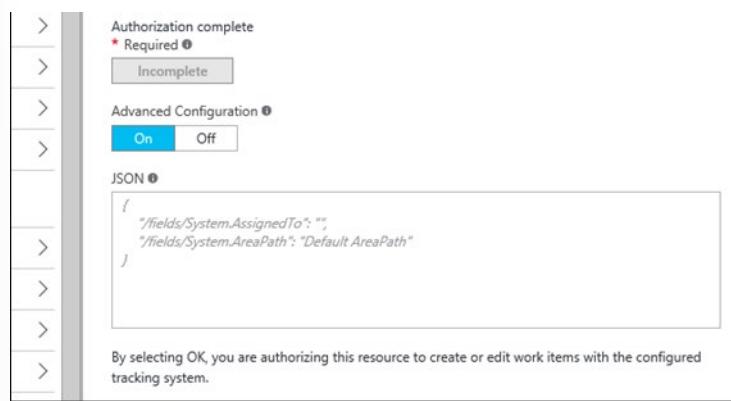
If you click the link for the work item that you want to view, it will open in VSTS:

The screenshot shows the detailed view of a work item titled '21 Issue with GET Home/About'. It includes tabs for HOME, CODE, WORK, BUILD, TEST, and RELEASE, with WORK selected. The work item is categorized under 'Area: AI_Test_Playground' and 'Iteration: AI_Test_Playground'. It is marked as 'New' and 'Unassigned'. The 'Repro Steps' section contains a detailed description of the issue, mentioning traffic from Houston, United States, PC device type, and specific event details. The 'Status' section shows it was 'New defect reported' and is now 'Found in Build'. The 'Details' section lists priority (2), severity (3 - Medium), and effort. The 'System Info' section is partially visible at the bottom.

Advanced Configuration

Some of you may have noticed that there is a switch on the configuration blade that is labeled "Advanced Configuration." This is additional functionality that we've provided to help you configure your ability to write to VSTS in scenarios where you've changed or extended some of the out-of-the-box settings. A good example of this is designating additional required fields. Currently, there is no way to handle this additional required mapping in the standard config, but you can handle it in advanced mode.

If you click on the switch, the controls at the bottom of the blade will change to look like this:



You can see that you are now given a JSON-based editing box where you can specify all of the particular settings/mappings that you might need to handle modifications to your VSTS project. Some sample text is filled in for you. In the near future, we plan to enhance this control with intellisense as well as publish some basic guidance to better understand the advanced configuration mode.

Next steps

We think that this is a good start to integrating work item functionality with Application Insights. But, please keep in mind that this is essentially the 1.0 version of this feature set. We have a lot of work planned, and you will see a significant evolution in this space over the upcoming months. Just for starters, let me outline a few of the things that we already have planned or are investigating:

- *Support for all work item types* – You probably noticed that the current feature set locks the work item type to just “bug.” Logging bugs was our primary ask for this space, so that’s where we started, but we certainly don’t think that’s where things should end. One of the more near-term changes that you will see is to handle all work item types for all supported processes in VSTS.
- *Links back to Application Insights* – It’s great to be able to create a work item with App Insights data in it, but what happens when you’re in your ALM system and looking at that item and want to quickly navigate back to the source of the work item in App Insights? We plan to add links to the work items in the very near future to make this as fast and easy as possible.
- *More flexible configuration* – Currently, our standard configuration only handles scenarios where the user has not significantly modified/extended their project in VSTS. Today, if you’ve made these kind of changes, you’ll need to switch to advanced configuration mode. Going forward, we want to handle common things that people might change (e.g., making additional fields required, adding new fields) in the standard configuration wherever possible. This requires some updates from our friends on the VSTS team, but they are already working on some of these for us. Once they’re available, we will begin to make the standard configuration more flexible. In the meantime (and in the future), you can always use the advanced configuration to overcome any limitations.
- *Multiple profiles* – Setting up a single configuration means that in shops where there are several ways in which users commonly create work items, the people creating work items from Application Insights would have to frequently override values. We plan to give users the capability to set up 1:n profiles, with standard values specified for each so that when you want to create a work item with that particular profile, you can simply choose it from a drop-down list.
- *More sources of creation for work items* – We will continue to investigate (and take feedback on) other places in Application Insights where it makes sense to create work items.

- *Automatic creation of work items* – There are certainly scenarios we can imagine (and I’m sure you can too) where we might want a work item to be created for us based upon criteria. This is definitely on the radar, but we are spending some design time with this to limit possibilities of super-noisy or runaway work item creation. We believe that this is a powerful and convenient feature, but we want to reduce the potential for spamming the ALM system as much as possible.
- *Support for other ALM systems* – Hey, we think that VSTS is an awesome product, but we recognize that many of our users may use some other product for their ALM, and we want to meet people where they are. So, we are working on additional first-tier integrations of popular ALM products. We also plan to provide a pure custom configuration choice (similar to advanced config for VSTS) so that end users will be able to hook up Application Insights to virtually any ALM system.

Lab

Monitoring Application Performance with Application Insights

Application Insights, a feature of Azure Monitor, is an extensible Application Performance Management (APM) service for developers and DevOps professionals. Use it to monitor your live applications. It will automatically detect performance anomalies, and includes powerful analytics tools to help you diagnose issues and to understand what users actually do with your app. It's designed to help you continuously improve performance and usability. It works for apps on a wide variety of platforms including .NET, Node.js and Java EE, hosted on-premises, hybrid, or any public cloud. It integrates with your DevOps process, and has connection points to a variety of development tools. It can monitor and analyze telemetry from mobile apps by integrating with Visual Studio App Center.

In this lab, **Monitoring Application Performance with Application Insights¹⁰¹**, you will learn how to:

- Implement application insights
- Track application usage
- Create application alerts

¹⁰¹ <https://azuredevopslabs.com/labs/azuredevops/appinsights/>

Module Review and Takeaways

Module Review Questions

Multiple choice

Does Azure Monitor allow you to create alerts from log query?

- yes
- no

Suggested answer

What features are provided by Azure Monitor?

Suggested answer

What query language can you use to query Azure Log Analytics?

Suggested answer

What platform integration does Azure Monitor provide to visualize your logs in real time?

Answers

Multiple choice

Does Azure Monitor allow you to create alerts from log query?

- yes
- no

What features are provided by Azure Monitor?

1. Detect and diagnose issues across applications and dependencies with Application Insights.
2. Correlate infrastructure issues with Azure Monitor for VMs and Azure Monitor for Containers.
3. Create visualizations with Azure dashboards and workbooks.
4. Support operations at scale with smart alerts and automated actions.

What query language can you use to query Azure Log Analytics?

Kusto

What platform integration does Azure Monitor provide to visualize your logs in real time?

Power BI and/or Grafana

Module 14 Infrastructure and Configuration Azure Tools

Module Overview

Module Overview

"Infrastructure as code" (IaC) doesn't quite trip off the tongue, and its meaning isn't always clear. But IaC has been with us since the beginning of DevOps—and some experts say DevOps wouldn't be possible without it.

As the name suggests, infrastructure as code is the concept of managing your operations environment in the same way you do applications or other code for general release. Rather than manually making configuration changes or using one-off scripts to make infrastructure adjustments, the operations infrastructure is managed instead using the same rules and strictures that govern code development—particularly when new server instances are spun up.

That means that the core best practices of DevOps—like version control, virtualized tests, and continuous monitoring—are applied to the underlying code that governs the creation and management of your infrastructure. In other words, your infrastructure is treated the same way that any other code would be.

The elasticity of the cloud paradigm and disposability of cloud machines can only truly be leveraged by applying the principles of Infrastructure as Code to all your infrastructure.

Learning Objectives

After completing this module, students will be able to:

- Apply infrastructure and configuration as code principles.
- Deploy and manage infrastructure using Microsoft automation technologies such as ARM templates, PowerShell, and Azure CLI

Infrastructure as Code and Configuration Management

Environment Deployment

If you've ever received a middle-of-the-night emergency support call because of a crashed server, you know the pain of searching through multiple spreadsheets or even your memory—to access the manual steps of setting up a new machine. Then there's also the difficulty of keeping the development and production environments consistent. An easier way to remove the possibility of human error when initializing machines is to use Infrastructure as Code.

Manual deployment versus Infrastructure as Code

A common analogy for understanding the differences between manual deployment and Infrastructure as Code is the distinction between owning pets and owning cattle. When you have pets, you name each one, and you regard them as individuals; if something bad happens to one of your pets, you are inclined to care a lot. If you have a herd of cattle, you might still name them, but you consider them as a herd.

In infrastructure terms, with a manual deployment approach there might be severe implications if a single machine crashes and you need to replace it (pets). If you adopt an Infrastructure as Code approach, if a single machine goes down you can more easily provision another machine without adversely impacting your entire infrastructure (cattle).

Implementing Infrastructure as Code

With Infrastructure as Code, you capture your environment (or environments) in a text file (script or definition). Your file might include any networks, servers, and other compute resources. You can check the script or definition file into version control, and then use it as the source for updating existing environments or creating new ones. For example, you can add a new server by editing the text file and running the release pipeline rather than remoting into the environment and manually provisioning a new server.

The following table lists the major differences between manual deployment and Infrastructure as Code.

Manual deployment	Infrastructure as Code
Snowflake servers	Consistent server between environments
Deployment steps vary by environment	Environments created or scaled easily
More verification steps, and more elaborate manual processes	Fully automated creation of environment Updates
Increased documentation to account for differences	Transition to immutable infrastructure
Deployment on weekends to allow time to recover from errors	Use blue/green deployments
Slower release cadence to minimize pain and long weekends	Treat servers as cattle, not pets

Benefits of Infrastructure as Code

The following list are benefits of Infrastructure as Code:

- Facilitates auditing by making it easier to trace what was deployed, when, and how. (In other words, improves traceability.)
- Provides consistent environments from release to release.
- Greater consistency across development, test, and production environments.
- Automates scale-up and scale-out processes.
- Allows configurations to be version controlled.
- Provides code review and unit-testing capabilities to help manage infrastructure changes.
- Uses *immutable* service processes, meaning if a change is needed to an environment, a new service is deployed and the old one taken down; it is not updated.
- Allows *blue/green* deployments. This is a release methodology to minimize downtime, where two identical environments exist—one is live and the other is not. Updates are applied to the server that is not live, and when testing is verified and complete, it's swapped with the other live server. It becomes the new live environment, and the previous live environment is no longer the live. This methodology is also referred to as *A/B deployments*.
- Treats infrastructure as a flexible resource that can be provisioned, de-provisioned, and re-provisioned as and when needed.

Environment Configuration

Configuration management refers to automated management of configuration, typically in the form of version-controlled scripts, for an application and all the environments needed to support it. Configuration management means lighter-weight, executable configurations that allow us to have configuration and environments as code.

For example, adding a new port to a Firewall, could be done by editing a text file and running the release pipeline, not by remoting into the environment and manually adding the port.

Note: The term *Configuration as Code* can also be used to mean configuration management, however is not used as widely, and in some cases, Infrastructure as Code is used to describe both provisioning and configuring machines. The term *Infrastructure as Code* is also sometimes used to include *Configuration as Code*, but not vice versa.

Manual configuration versus Configuration as Code

Manually managing the configuration of a single application and environment can be challenging. The challenges are even greater for managing multiple applications and environments across multiple servers. Automated configuration, or treating Configuration as Code, can alleviate some of the challenges associated with manual configuration.

The following table lists the major differences between manual configuration and Configuration as Code.

Manual configuration	Configuration as code
Configuration bugs difficult to identify	Bugs easily reproducible
Error prone	Consistent configuration

Manual configuration	Configuration as code
More verification steps, and more elaborate manual processes	Increase deployment cadence to reduce amount of incremental change
Increased documentation	Treat environment and configuration as executable documentation
Deployment on weekends to allow time to recover from errors	
Slower release cadence to minimize requirement for long weekends	

Benefits of configuration management

The following list are benefits of configuration management:

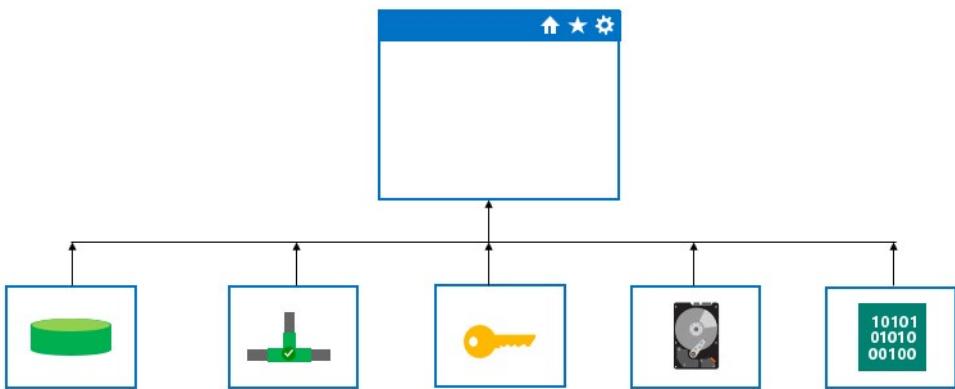
- Bugs more easily reproduced, facilitates auditing and improves traceability
- Provides consistency across environments such as dev, test and release
- Increased deployment cadence
- Less documentation needed and needing to be maintained as all configuration is available in scripts
- Enables automated scale-up and scale-out
- Allows version-controlled configuration
- Helps detect and correct configuration drift
- Provides code-review and unit-testing capabilities to help manage infrastructure changes
- Treats infrastructure as a flexible resource
- Facilitates automation

Modularization

When architecting automation solutions, how you structure or reference the various components that define your environment (such as text files, definitions, and scripts) is important. Managing, maintaining, and troubleshooting your automation files can become more difficult over time if automation components are developed and used in an improvised manner. This is because applications and environments change over time, and using this method increases complexity.

For small-to-medium solutions with small teams, having single files that define your application resources and configuration can be fine, and might be easier to understand and maintain. A benefit to this scenario is you can review all the resources and values in a single file. However, for advanced or more complex scenarios, breaking your automation resources into their constituent parts is a major benefit. This is sometimes referred to as *modularization*.

Modularizing your automation assets into their component parts collectively define the environment and configuration for your application. You can break your automation assets into their own logical parts, such as databases, networks, storage, security, and operating systems.



Benefits of modularization

Using modularization makes it easier to:

- Reuse components across different scenarios.
- Manage and maintain your code.
- Troubleshoot your automation resources.
- Help new team members understand how the infrastructure and components relate and are used.
- Sub-divide up the work and responsibilities across teams and area owners.
- Extend and add to your existing infrastructure definitions.

Imperative Vs Declarative

There are a few different approaches that you can adopt to implement Infrastructure as Code and Configuration as Code. Two of the main methods of approach are:

- **Declarative** (functional). The declarative approach states *what* the final state should be. When run, the script or definition will initialize or configure the machine to have the finished state that was declared, without defining *how* that final state should be achieved.



- **Imperative** (procedural). In the imperative approach, the script states the *how* for the final state of the machine by executing the steps to get to the finished state. It defines what the final state needs to be, but also includes how to achieve that final state. It also can include coding concepts such as *for*, **if-then*, *loops*, and matrices.



Best Practices

The **declarative** approach abstracts away the methodology of how a state is achieved. As such, it can be easier to read and understand what is being done. It also makes it easier to write and define. Declarative approaches also separate out the final desired state, and the coding required to achieve that state. Thus, it does not force you to use a particular approach, which allows for optimization where possible.

A **declarative** approach would generally be the preferred option where ease of use is the main goal. Azure Resource Manager template files are an example of a declarative automation approach.

An **imperative** approach may have some advantages where there are complex scenarios where changes in the environment take place relatively frequently, which need to be accounted for in your code.

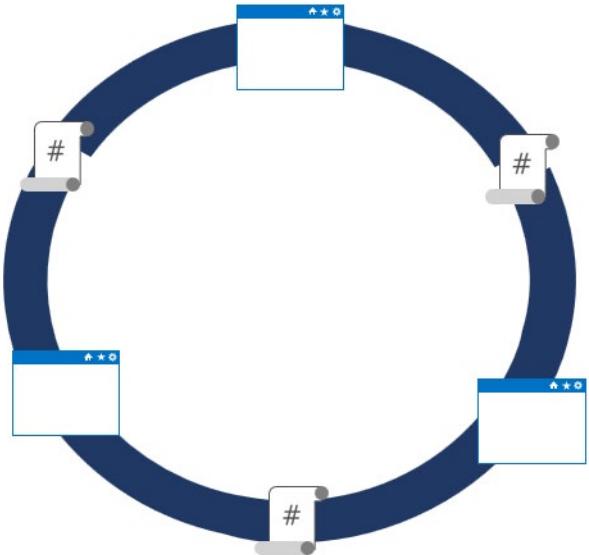
There is no absolute on which is the best approach to take, and individual tools may be able to be used in either *declarative* or *imperative* forms. The best approach for you to take will depend on your particular needs.

Idempotence

Idempotence is a mathematical term that can be used in the context of Infrastructure as Code and Configuration as Code. It is the ability to apply one or more operations against a resource, resulting in the same outcome.

For example, if you run a script on a system it should have the same outcome regardless of the number of times you execute the script. It should not error out, or perform duplicate actions regardless of the environment's starting state.

In essence, if you apply a deployment to a set of resources 1,000 times, you should end up with the same result after each application of the script or template.



You can achieve idempotency by:

- Automatically configuring and reconfiguring an existing set of resources.
Or
- Discarding the existing resources and recreating a fresh environment.

When defining Infrastructure as Code and Configuration as Code, as a best practice build the scripts and templates in such a way as to embrace idempotency. This is particularly important when working with cloud services, because resources and applications can be scaled in and out regularly, and new instances of services can be started up to provide service elasticity.

Note: You can read more about idempotence at [Idempotency for Windows Azure Message Queues](#)¹.

Inspecting and Validating Infrastructure for Compliance

Early attempts at automating the creation of virtual machines and entire environments focused on writing procedural code in languages like PowerShell. The problem with this type of coding was that it was hard to write idempotent code ie: code that produces the same outcome no matter how many times you run it. **PowerShell Desired State Configuration (DSC)**² was a big step forward because it allowed you to define the desired outcome instead of the process for achieving that outcome. It was a declarative mechanism rather than a procedural one.

Today, a variety of tools allow you to defining infrastructure as code, and these have already created significant benefits to DevOps teams. Reliable environments (including operating systems and services) can be created on demand, and in a completely repeatable way. The same code can be run again to ensure that the target state is present, and to remove any drift from that state.

Rather than just configuration of the environments though, there is a need to validate the environments, particularly in relation to organizational or regulatory policies that might relate to them.

¹ <https://www.wintellect.com/idempotency-for-windows-azure-message-queues/>

² <https://docs.microsoft.com/en-us/powershell/scripting/dsc/overview/overview?view=powershell-6>

Validation tooling

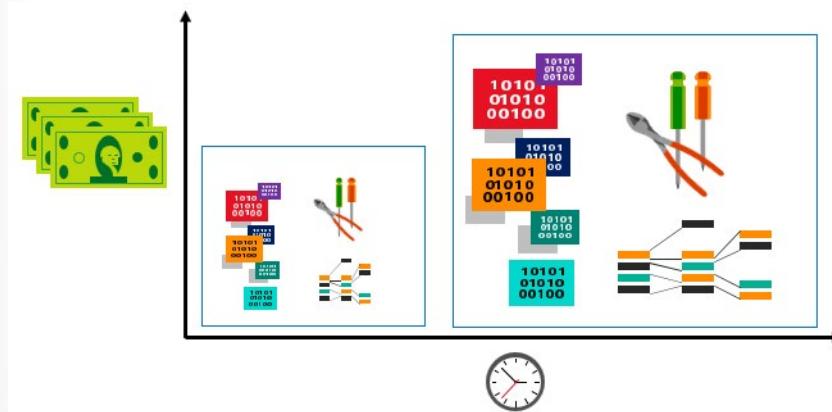
Azure Automation State Configuration³ builds on PowerShell DSC to make management easier. It allows you to author and manage DSC Configurations, import DSC Resources, and generate DSC Node Configurations (MOF documents), all in the cloud. These items are placed on an Automation server so that target nodes in the cloud or on-premises can retrieve them, and automatically conform to the desired state, and report back on their compliance.

As an example of alternate tooling, **InSpec**⁴ is an open-source testing framework for infrastructure with a language for specifying compliance, security and other policy requirements. It works by comparing the actual state of your system with the desired state that you express in InSpec code. It then detects violations and displays findings in the form of a report. The remediation is then done by the administrator.

Technical debt

Technical debt is a set of problems in a development effort that make progress on customer value inefficient. We think of our scripts, templates, and definition files as code; technical debt undermines productivity by making the code fragile, hard to understand, time-consuming to change, and difficult to validate. This in turn creates unplanned work that blocks progress. Technical debt saps an organization's strength because of high customer-support costs. Eventually, some combination of these issues produces larger problems.

Technical debt is also insidious, meaning it starts small and grows over time as a result of rushed changes, and lack of context and discipline. It can seemingly materialize out of nowhere (even for a project regarded as clean), because of changes in project circumstances. For example, a product produced for one particular market might be considered for international release. This instantly creates debt related to its ability to localize. Technical debt introduced to the application will have repercussions later on, and will need to be addressed at some stage.



Examples of technical debt

Technical debt includes anything the team must do to deploy production-quality code and keep it running in production. Examples of technical debt include:

- Bugs
- Performance issues

³ <https://docs.microsoft.com/en-us/azure/automation/automation-dsc-overview>

⁴ <https://www.inspec.io>

- Operational issues
- Accessibility
- Manual updates
- Configurations not implemented using Infrastructure as Code Configuration as Code methodologies (such as version control)
- Changes made quickly, or directly to an application without using DevOps methodologies
- Switching to technologies or versions not accounted for in your development process
- Updates to platform or services that you were not aware of or have not accounted for

Consider the following:

Question Can Azure Resource Manager templates contain technical debt?

Answer Yes, Azure Resource Manager templates deploy and configure resources in Azure. As a cloud-based platform, Azure is continually changing and evolving. When deploying resources using templates, you frequently need to retrieve information about the resource providers and types. As a resource provider enables new features, it releases a new version of the REST API. As such, components within your resource manager templates which rely on application programming interface (API) versions to provision and configure resources might need periodic updating. A solution to address this could be to validate API versions in your templates to ensure they are current and not deprecated, as part of your general work development work.

How to manage technical debt

Technical debt is not necessarily eliminated—rather it is managed. Some considerations when managing technical debt include:

- Understanding and identifying the debt and where it's located in the code.
- Evaluating the costs of remediation and non-remediation. Fixing technical debt has a cost; but not fixing it also has a cost, which could be larger and more complex to evaluate.
- Putting policies in place that focus on preventing debt from becoming worse, and managing it down. This could include:
 - Allowing for periodic updates to Microsoft Azure Resource Manager templates for new API versions as part of your general development work.
 - Ensuring that no manual configurations are allowed.
- Exposing developers in a non-overwhelming way to the debt required to meet the policies, so they can regard it as a natural part of their day-to-day development process and not as a time-consuming and tiresome chore.
- Tracking debt over time to ensure that it's trending in the right direction and meeting defined policies.
- Remediating debt as efficiently and automatically as possible.
- Prioritizing debt alongside new application features and desired functionality.

Note: A first step when working with technical debt is to try to quantify it. You can do this using a tool such as **SonarQube**⁵.

⁵ <https://www.sonarqube.org/>

Configuration Drift

Configuration drift is the process of a set of resources changing over time from their original deployment state. This can be because of changes made manually by people, or automatically by processes or programs.

Eventually, an environment can become a snowflake. A *snowflake* is a unique configuration that cannot be reproduced automatically, and is typically a result of configuration drift. With snowflakes, the infrastructure administration and maintenance invariably involves manual processes, which can be hard to track and prone to human error. The more an environment drifts from its original state, the more likely it is for an application to encounter issues. The greater the degree of configuration drift, the longer it takes to troubleshoot and rectify issues.



Security considerations

Configuration drift can also introduce security vulnerabilities into your environment. For example:

- Ports might be opened that should be kept closed.
- Updates and security patches might not be applied across environments consistently.
- Software might be installed that doesn't meet compliance requirements.

Solutions for managing configuration drift

While eliminating configuration drift entirely can be difficult, there are many ways you can manage it in your environments using configuration management tools and products such as:

- Windows PowerShell Desired State Configuration. This is a management platform in PowerShell that enables you to manage and enforce resource configurations. For more information about Windows PowerShell Desired State Configuration, go to **Windows PowerShell Desired State Configuration Overview**⁶.
- Azure Policy. Use Azure Policy to enforce policies and compliance standards for Azure resources. For more information about Azure Policy, go to: **Azure Policy**⁷.

There are also other third-party solutions that you can integrate with Azure.

Database as Code

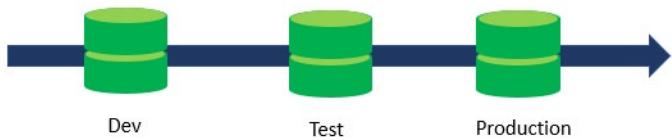
Regarding Infrastructure as Code and Configuration as Code, another key area of note is the database. By adopting a similar approach to deploying Infrastructure as Code, we can also treat the database as code, reducing database read/write operation times and improving the efficacy of database administration.

A database can be version-controlled in the same way that applications can, thereby enabling bugs to be reproduced more easily. By treating the database as code, its configuration stays more consistent. You

⁶ <https://docs.microsoft.com/en-us/powershell/scripting/dsc/overview/overview?view=powershell-7>

⁷ <https://azure.microsoft.com/en-us/services/azure-policy/>

can address drifts that occur in the production environment by initializing the database in a development environment for debugging.



Benefits to treating database as code

The main benefits to treating database as code are:

- Making changes in small increments allows you to change a few scripts, as opposed to performing a roll-up of schema changes that might result in time-consuming errors.
- Configuration scripts can be treated as executable documentation because they are small enough for people to thoroughly review and understand them.
- Schema changes can go both forwards and backwards because DevOps provides tools for comparing production with development (i.e. 'Diff'). These tools make it easier to identify differences between production and development, and determine which modifications need to be made.
- Improves *auditability*, which is the ability to track the history of changes to identify and address problems quickly.

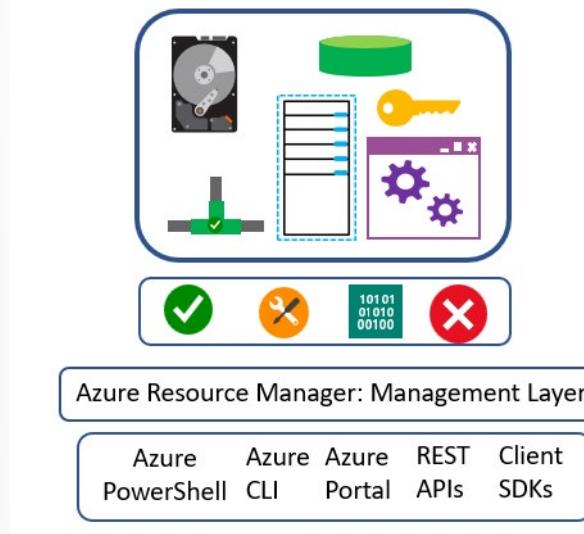
For more information about database DevOps, review Redgate's **Compliant Database DevOps**⁸.

⁸ <http://www.red-gate.com/solutions/database-devops/>

Create Azure Resources usign ARM Templates

What is Azure Resource Manager

Azure Resource Manager (also referred to as **ARM**) is a management layer in which a resource group and all the resources within it are created, configured, managed, and deleted. It provides a consistent management layer that allows you to automate deployment and configuration of resources, using different automation and scripting tools such as Microsoft Azure PowerShell, Azure Command-Line Interface (Azure CLI), Microsoft Azure portal, REST API, and client SDKs.



With Azure Resource Manager, you can deploy Application resources. You also can update, manage, and delete all the resources for your solution in a single, coordinated operation.

Azure Resource Manager has several components, one of which is *resource provider*. *Resource providers* offer a set of resources and operations for working with an Azure service, which are made available through Azure Resource Manager. Some common resource providers are:

- **Microsoft.Compute**, which supplies the virtual machine resource.
- **Microsoft.Storage**, which supplies the storage account resource.
- **Microsoft.Web**, which supplies resources related to web apps.

Note: For more detail about resource providers and types, visit [Azure resource providers and types⁹](https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-manager-supported-services)

Azure Resource Manager Templates

An **Azure Resource Manager template** precisely defines all the Resource Manager resources in a deployment. You can deploy a Resource Manager template into a resource group as a single operation.

Resource Manager templates are declarative in nature and are written in JSON format. By using a template, you can repeatedly deploy your solution in development, test, and production type environments throughout its life cycle, and have confidence that your resources are deployed in a consistent state.

Creating a template from scratch would be possible, but difficult. It's more typical to take an existing template and customize it to meet your specific needs.

⁹ <https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-manager-supported-services>

Also, when you create a deployment within the Azure portal, the portal generates a deployment template to mirror the configuration you created in the portal. You can download and use this deployment template as a starting template if you want.

Template Components

Azure Resource Manager templates are written in JSON, which allows you to express data stored as an object (such as a virtual machine) in text. A *JSON document* is essentially a collection of key-value pairs. Each key is a string that's value can be:

- A string
- A number
- A Boolean expression
- A list of values
- An object (which is a collection of other key-value pairs)

A Resource Manager template can contain sections that are expressed using JSON notation, but are not related to the JSON language itself:

```
{  
    "$schema": "http://schema.management.azure.com/schemas/2015-01-01/  
deploymentTemplate.json#",  
    "contentVersion": "",  
    "parameters": { },  
    "variables": { },  
    "functions": [ ],  
    "resources": [ ],  
    "outputs": { }  
}
```

Let's review each of these sections in a little more detail.

Parameters

This section is where you specify which values are configurable when the template runs. For example, you might allow template users to specify a username, password, or domain name.

Here's an example that illustrates two parameters: one for a virtual machine's (VM's) username, and one for its password:

```
"parameters": {  
    "adminUsername": {  
        "type": "string",  
        "metadata": {  
            "description": "Username for the Virtual Machine."  
        }  
    },  
    "adminPassword": {  
        "type": "securestring",  
        "metadata": {  
            "description": "Password for the Virtual Machine."  
        }  
    }  
}
```

```
    }
}
```

Variables

This section is where you define values that are used throughout the template. Variables can help make your templates easier to maintain. For example, you might define a storage account name one time as a variable, and then use that variable throughout the template. If the storage account name changes, you need only update the variable once.

Here's an example that illustrates a few variables that describe networking features for a VM:

```
"variables": {
  "nicName": "myVMNic",
  "addressPrefix": "10.0.0.0/16",
  "subnetName": "Subnet",
  "subnetPrefix": "10.0.0.0/24",
  "publicIPAddressName": "myPublicIP",
  "virtualNetworkName": "MyVNET"
}
```

Functions

This section is where you define procedures that you don't want to repeat throughout the template. Similar to variables, functions can help make your templates easier to maintain.

Here's an example that creates a function for creating a unique name to use when creating resources that have globally unique naming requirements:

```
"functions": [
  {
    "namespace": "contoso",
    "members": {
      "uniqueName": {
        "parameters": [
          {
            "name": "namePrefix",
            "type": "string"
          }
        ],
        "output": {
          "type": "string",
          "value": "[concat(toLower(parameters('namePrefix')), uniqueString(resourceGroup().id))]"
        }
      }
    }
  ],
  ]
},
```

Resources

This section is where you define the Azure resources that make up your deployment.

Here's an example that creates a public IP address resource:

```
{  
    "type": "Microsoft.Network/publicIPAddresses",  
    "name": "[variables('publicIPAddressName')]",  
    "location": "[parameters('location')]",  
    "apiVersion": "2018-08-01",  
    "properties": {  
        "publicIPAllocationMethod": "Dynamic",  
        "dnsSettings": {  
            "domainNameLabel": "[parameters('dnsLabelPrefix')]"  
        }  
    }  
}
```

Here, the type of resource is `Microsoft.Network/publicIPAddresses`. The **name** is read from the variables section, and the **location**, or *Azure region*, is read from the **parameters** section.

Because resource types can change over time, `apiVersion` refers to the version of the resource type you want to use. As resource types evolve, you can modify your templates to work with the latest features.

Outputs

This section is where you define any information you'd like to receive when the template runs. For example, you might want to receive your VM's IP address or fully qualified domain name (FQDN), information you will not know until the deployment runs.

Here's an example that illustrates an output named **hostname**. The FQDN value is read from the VM's public IP address settings:

```
"outputs": {  
    "hostname": {  
        "type": "string",  
        "value": "[reference(variables('publicIPAddressName')).dnsSettings.  
fqdn]"  
    }  
}
```

QuickStart Templates

What are Azure Quickstart templates?

Azure Quickstart templates are Resource Manager templates provided by the Azure community. Quickstart templates are accessible from [Azure Quickstart Templates¹⁰](#) or [Github Azure Quickstart Templates¹¹](#).

Many templates provide everything you need to deploy your solution, while others might serve as a starting point for your template. Either way, you can study these templates to learn how to best author and structure your own templates.

Discover what's in the Quickstart Template gallery

Let's say you want to find a Resource Manager template with a basic VM configuration, preferably one that includes a VM, basic network settings, and storage.

1. You could start by browsing to the [Azure Quickstart Templates gallery¹²](#) to find what's available. In the gallery you will find a number of popular and recently updated templates. These templates work with both Azure resources and popular software packages.

The screenshot shows a grid of four template cards under the heading "Most popular".

- Create a Standard Storage Account**
This template creates a Standard Storage Account.
by [Kay Singh](#), Last updated: 12/4/2018
- Create a Virtual Network with two Subnets**
This template allows you to create a Virtual Network with two subnets.
by [Telmo Sampaio](#), Last updated: 10/12/2018
- Create an Azure VM with a new AD Forest**
This template creates a new Azure VM, it configures the VM to be an AD DC for a new Forest.
by [Simon Davies](#), Last updated: 7/4/2018
- Join a VM to an existing domain**
This template demonstrates domain join to a private AD domain up in cloud.
by [Kay Singh](#), Last updated: 5/25/2018

2. Let's say you come across the [Deploy a simple Windows VM¹³](#) template.

¹⁰ <https://azure.microsoft.com/en-us/resources/templates>

¹¹ <https://github.com/Azure/azure-quickstart-templates>

¹² <https://azure.microsoft.com/resources/templates?azure-portal=true>

¹³ <https://azure.microsoft.com/resources/templates/101-vm-simple-windows?azure-portal=true>

The screenshot shows a template page for 'Deploy a simple Windows VM'. At the top, there's a navigation bar with 'Templates' and a breadcrumb trail 'Deploy a simple Windows VM'. Below that is the title 'Deploy a simple Windows VM' and a profile picture of Brian Moore. Two buttons are present: 'Deploy to Azure' and 'Browse on GitHub'. A descriptive text box below the buttons states: 'This template allows you to deploy a simple Windows VM using a few different options for the Windows version, using the latest patched version. This will deploy a A2 size VM in the resource group location and return the FQDN of the VM.'

Note: The **Deploy to Azure** button enables you to deploy the template directly through the Azure portal if you want.

Let's review this template more closely and discover what it accomplishes:

3. Select **Browse on GitHub** to navigate to the template's source code on GitHub.

The screenshot shows a GitHub page for 'Very simple deployment of a Windows VM'. It features a title 'Very simple deployment of a Windows VM' and two buttons: 'Deploy to Azure' and 'Visualize'. A text box below the buttons contains the same descriptive text as the previous screenshot: 'This template allows you to deploy a simple Windows VM using a few different options for the Windows version, using the latest patched version. This will deploy a A2 size VM in the resource group location and return the FQDN of the VM.'

The **Deploy to Azure** button enables you to deploy the template directly through the Azure portal, just like you could on the gallery page.

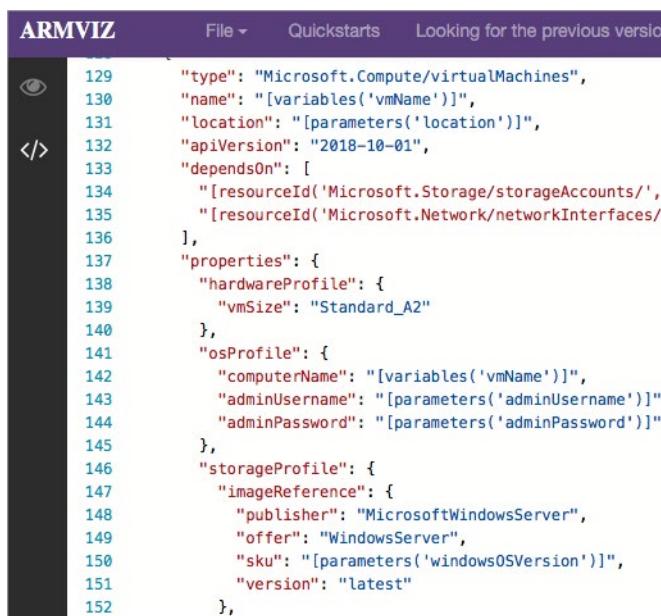
4. Select **Visualize** to navigate to the **Azure Resource Manager Visualizer**.

The screenshot shows the 'ARMVIZ' interface. The top navigation bar includes 'ARMVIZ', 'File ▾', 'Quickstarts', and a search bar 'Looking for the previous version?'. The main area displays three resources: 'SimpleWinVM' (Virtual Machine), 'myVMNic' (Network Interface), and 'MyVNET' (Virtual Network). Each resource has a small icon and a label indicating its type.

Here you can review the resources that make up the deployment, including a VM, a storage account, and network resources.

Use your mouse to arrange the resources. You can also use your mouse's scroll wheel to zoom in and out.

5. Select the VM resource labeled **SimpleWinVM**.



The screenshot shows a code editor window titled "ARMVIZ" with the following content:

```
129     "type": "Microsoft.Compute/virtualMachines",
130     "name": "[variables('vmName')]",
131     "location": "[parameters('location')]",
132     "apiVersion": "2018-10-01",
133     "dependsOn": [
134         "[resourceId('Microsoft.Storage/storageAccounts/',
135         "[resourceId('Microsoft.Network/networkInterfaces/'"
136     ],
137     "properties": {
138         "hardwareProfile": {
139             "vmSize": "Standard_A2"
140         },
141         "osProfile": {
142             "computerName": "[variables('vmName')]",
143             "adminUsername": "[parameters('adminUsername')]",
144             "adminPassword": "[parameters('adminPassword')]"
145         },
146         "storageProfile": {
147             "imageReference": {
148                 "publisher": "MicrosoftWindowsServer",
149                 "offer": "WindowsServer",
150                 "sku": "[parameters('windowsOSVersion')]",
151                 "version": "latest"
152             },
153         }
154     }
155 }
```

From here you can review the source code that defines the VM resource.

6. After briefly reviewing it, you find that:

- The resource's type is **Microsoft.Compute/virtualMachines**.
- It's location (or *Azure region*) comes from the template parameter named **location**.
- The VM's size is **Standard_A2**.
- The computer name is read from a template variable, and the username and password for the VM are read from template parameters.

You also have the option to review the **README** file on GitHub and further inspect the source code to decide whether the template suits your needs.

Manage Dependancies

For any given resource, other resources might need to exist before you can deploy the resource. For example, a Microsoft SQL Server must exist before attempting to deploy a SQL Database. You can define this relationship by marking one resource as dependent on the other. You define a dependency with the **dependsOn** element, or by using the **reference** function.

Resource Manager evaluates the dependencies between resources, and deploys them in their dependent order. When resources aren't dependent on each other, Resource Manager deploys them in parallel. You only need to define dependencies for resources that are deployed in the same template.

The **dependsOn** element

Within your template, the **dependsOn** element enables you to define one resource as a dependent on one or more other resources. Its value can be a comma-separated list of resource names.

```
129     "type": "Microsoft.Compute/virtualMachines",
130     "name": "[variables('vmName')]",
131     "location": "[parameters('location')]",
132     "apiVersion": "2018-10-01",
133     "dependsOn": [
134       "[resourceId('Microsoft.Storage/storageAccounts/', variables('storageAccountName'))]",
135       "[resourceId('Microsoft.Network/networkInterfaces/', variables('nicName'))]"
136     ],

```

Circular dependencies

A **circular dependency** is when there is a problem with dependency sequencing, resulting in the deployment going around in a loop and unable to proceed. As a result, Resource Manager cannot deploy the resources. Resource Manager identifies circular dependencies during template validation. If you receive an error stating that a circular dependency exists, evaluate your template to find whether any dependencies are not needed and can be removed.

If removing dependencies doesn't resolve the issue, you can move some deployment operations into child resources that are deployed after the resources with the circular dependency.

Modularize Templates

When using Azure Resource Manager templates, a best practice is to modularize them by breaking them out into the individual components. The primary methodology to use to do this is by using linked templates. These allow you to break out the solution into targeted components, and then reuse those various elements across different deployments.

Linked template

To link one template to another, add a deployment's resource to your main template.

```
"resources": [
  {
    "apiVersion": "2017-05-10",
    "name": "linkedTemplate",
    "type": "Microsoft.Resources/deployments",
    "properties": {
      "mode": "Incremental",
      <link-to-external-template>
    }
  }
]
```

Nested template

You can also nest a template within the main template, use the template property, and specify the template syntax. This does aid somewhat in the context of modularization, but dividing up the various components can result in a large main file, as all the elements are within that single file.

```
"resources": [
    {
        "apiVersion": "2017-05-10",
        "name": "nestedTemplate",
        "type": "Microsoft.Resources/deployments",
        "properties": {
            "mode": "Incremental",
            "template": {
                "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
                "contentVersion": "1.0.0.0",
                "resources": [
                    {
                        "type": "Microsoft.Storage/storageAccounts",
                        "name": "[variables('storageName')]",
                        "apiVersion": "2015-06-15",
                        "location": "West US",
                        "properties": {
                            "accountType": "Standard_LRS"
                        }
                    }
                ]
            }
        }
    }
]
```

Note: For nested templates, you cannot use parameters or variables that are defined within the nested template itself. You can only use parameters and variables from the main template.

The properties you provide for the deployment resource will vary based on whether you're linking to an external template or nesting an inline template within the main template.

Deployments modes

When deploying your resources using templates, you have three options:

- **validate.** This option compiles the templates, validates the deployment, ensures the template is functional (for example, no circular dependencies), and the syntax is correct.
- **incremental mode (default).** This option only deploys whatever is defined in the template. It does not remove or modify any resources that are not defined in the template. For example, if you have deployed a VM via template and then renamed the VM in the template, the first VM deployed will still remain after the template is run again. This is the default mode.
- **complete mode:** Resource Manager deletes resources that exist in the resource group, but aren't specified in the template. For example, only resources defined in the template will be present in the resource group after the template deploys. As best practice, use this mode for production environments where possible to try to achieve idempotency in your deployment templates.

When deploying with PowerShell, to set the deployment mode use the *Mode* parameter, as per the nested template example earlier in this topic.

Note: As a best practice, use one resource group per deployment.

Note: For both linked and nested templates, you can only use incremental deployment mode.

External template and external parameters

To link to an external template and parameter file, use **templateLink** and **parametersLink**. When linking to a template, ensure that the Resource Manager service can access it. For example, you can't specify a local file or a file that is only available on your local network. As such, you can only provide a Uniform Resource Identifier (URI) value that includes either http or https. One option is to place your linked template in a storage account, and use the URI for that item.

You can also provide the parameter inline. However, you can't use both inline parameters and a link to a parameter file. The following example uses the *templateLink* parameter:

```
"resources": [
    {
        "name": "linkedTemplate",
        "type": "Microsoft.Resources/deployments",
        "apiVersion": "2018-05-01",
        "properties": {
            "mode": "Incremental",
            "templateLink": {
                "uri": "https://linkedtemplateek1store.blob.core.windows.net/linkedtemplates/linkedStorageAccount.json?sv=2018-03-28&sr=b&sig=d09p7Xnbh-Gq56BO%2BSW3o9tX7E2WUdIk%2BpF1MTK2eFfs%3D&se=2018-12-31T14%3A32%3A29Z&sp=r"
            },
            "parameters": {
                "storageAccountName": {"value": "[variables('storageAccountName')]" },
                "location": {"value": "[parameters('location')]" }
            }
        }
    },
]
```

Securing an external template

Although the linked template must be available externally, it doesn't need to be made available to the public. Instead, you can add your template to a private storage account that is accessible to only the storage account owner, then create a shared access signature (SAS) token to enable access during deployment. You add that SAS token to the URI for the linked template. Even though the token is passed in as a secure string, the linked template's URI, including the SAS token, is logged in the deployment operations. To limit exposure, you can also set an expiration date for the token.

Managing Secrets in Templates

When you need to pass a secure value (such as a password) as a parameter during deployment, you can retrieve the value from an Azure Key Vault by referencing the Key Vault and secret in your parameter file. The value is never exposed because you only reference its Key Vault ID. The Key Vault can exist in a different subscription than the resource group you are deploying it to.

Deploy a Key Vault and secret

To create a Key Vault and secret, use either Azure CLI or PowerShell. To access the secrets inside this Key Vault from a Resource Manager deployment, the Key Vault property **enabledForTemplateDeployment** must be **true**.

Using Azure CLI

The following code snippet is an example how you can deploy a Key Vault and secret using Azure CLI:

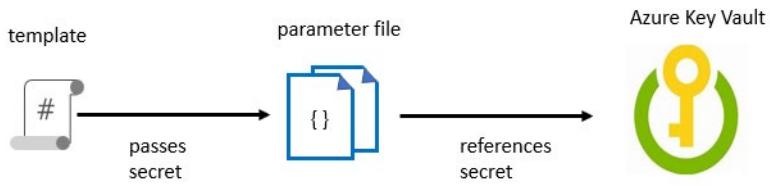
```
keyVaultName='{your-unique-vault-name}'  
resourceGroupName='{your-resource-group-name}'  
location='centralus'  
userPrincipalName='{your-email-address-associated-with-your-subscription}'  
  
# Create a resource group  
az group create --name $resourceGroupName --location $location  
  
# Create a Key Vault  
az keyvault create \  
  --name $keyVaultName \  
  --resource-group $resourceGroupName \  
  --location $location \  
  --enabled-for-template-deployment true  
az keyvault set-policy --upn $userPrincipalName --name $keyVaultName --se-  
cret-permissions set delete get list  
  
# Create a secret with the name, vmAdminPassword  
password=$(openssl rand -base64 32)  
echo $password  
az keyvault secret set --vault-name $keyVaultName --name 'vmAdminPassword'  
--value $password
```

Enable access to the secret

Other than setting the Key Vault property **enabledForTemplateDeployment** to **true**, the user deploying the template must have the `Microsoft.KeyVault/vaults/deploy/action` permission for scope that contains the Key Vault, including the resource group and Key Vault. The Owner and Contributor roles both grant this access. If you create the Key Vault, you are the owner, so you inherently have permission. However, if the Key Vault is under a different subscription, the owner of the Key Vault must grant the access.

Reference a secret with static ID

The Key Vault is referenced in the parameter file, and not the template. The following image shows how the parameter file references the secret and passes that value to the template.



The following template (also available at [GitHub - sqlserver.json](#)¹⁴) deploys a SQL database that includes an administrator password. The password parameter is set to a secure string. However, the template does not specify where that value comes from:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "adminLogin": {
            "type": "string"
        },
        "adminPassword": {
            "type": "securestring"
        },
        "sqlServerName": {
            "type": "string"
        }
    },
    "resources": [
        {
            "name": "[parameters('sqlServerName')]",
            "type": "Microsoft.Sql/servers",
            "apiVersion": "2015-05-01-preview",
            "location": "[resourceGroup().location]",
            "tags": {},
            "properties": {
                "administratorLogin": "[parameters('adminLogin')]",
                "administratorLoginPassword": "[parameters('adminPassword')]",
                "version": "12.0"
            }
        }
    ],
    "outputs": {}
}
```

Now you can create a parameter file for the preceding template. In the parameter file, specify a parameter that matches the name of the parameter in the template. For the parameter value, reference the secret from the Key Vault. You reference the secret by passing the resource identifier of the Key Vault and the

¹⁴ <https://github.com/Azure/azure-docs-json-samples/blob/master/azure-resource-manager/keyvaultparameter/sqlserver.json>

name of the secret. In the following parameter file (also available at [GitHub - keyvaultparameter¹⁵](#)), the Key Vault secret must already exist, and you provide a static value for its resource ID.

Copy this file locally, and set the subscription ID, vault name, and SQL server name:

```
{  
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/  
deploymentParameters.json#",  
    "contentVersion": "1.0.0.0",  
    "parameters": {  
        "adminLogin": {  
            "value": "exampleadmin"  
        },  
        "adminPassword": {  
            "reference": {  
                "keyVault": {  
                    "id": "/subscriptions/<subscription-id>/resourceGroups/  
examplegroup/providers/Microsoft.KeyVault/vaults/<vault-name>"  
                },  
                "secretName": "examplesecret"  
            }  
        },  
        "sqlServerName": {  
            "value": "<your-server-name>"  
        }  
    }  
}
```

All you would need to do now is deploy the template and pass in the parameter file to the template.

For more information, go to [Use Azure Key Vault to pass secure parameter value during deployment¹⁶](#) for more details. There also are details available on this webpage for how to reference a secret with a dynamic ID as well.

Template Extensions

The Custom Script Extension (CSE) is an easier way to download and run scripts on your Azure VMs. It's just one of the many ways you can configure a VM, either during its deployment or after it's up and running.

You can store your scripts in Azure storage or in a public location such as GitHub. You can run scripts manually or as part of a more automated deployment. In the following example we use, we define a resource that you can add to your Resource Manager template. The resource uses the **Custom Script Extension** to download a PowerShell script from GitHub, and then execute that script on your VM. The script enables the **IIS-WebServerRole** feature, and configures a basic home page.

Implementing CSEs

One way to discover how to use the CSE in your template is to search for examples. You might find an Azure Quickstart template that does something similar and adapt it to your needs. Another approach is

¹⁵ <https://github.com/Azure/azure-docs-json-samples/blob/master/azure-resource-manager/keyvaultparameter/sqlserver.parameters.json>

¹⁶ <https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-manager-keyvault-parameter>

to search for the resource you need from **Define resources in Azure Resource Manager templates¹⁷**. In this case, searching the documentation would reveal **Microsoft.Compute virtualMachines/extensions template reference¹⁸**.

You can start by copying the schema to a temporary document:

```
{  
    "name": "string",  
    "type": "Microsoft.Compute/virtualMachines/extensions",  
    "apiVersion": "2018-10-01",  
    "location": "string",  
    "tags": {},  
    "properties": {  
        "publisher": "string",  
        "type": "string",  
        "typeHandlerVersion": "string",  
        "autoUpgradeMinorVersion": boolean,  
        "settings": {},  
        "protectedSettings": {},  
        "instanceView": {  
            "name": "string",  
            "type": "string",  
            "typeHandlerVersion": "string",  
            "substatuses": [  
                {  
                    "code": "string",  
                    "level": "string",  
                    "displayStatus": "string",  
                    "message": "string",  
                    "time": "string"  
                }  
            ],  
            "statuses": [  
                {  
                    "code": "string",  
                    "level": "string",  
                    "displayStatus": "string",  
                    "message": "string",  
                    "time": "string"  
                }  
            ]  
        }  
    }  
}
```

¹⁷ <https://docs.microsoft.com/azure/templates?azure-portal=true>

¹⁸ <https://docs.microsoft.com/azure/templates/Microsoft.Compute/2018-10-01/virtualMachines/extensions?azure-portal=true>

Specify required properties

The schema lists all the properties that you can provide. Some properties are required, while others are optional. You might start by identifying all of the required properties. Locate these following the schema definition on the reference page:

- **name**
- **type**
- **apiVersion**
- **location**
- **properties**

After you remove all the parameters that aren't required, your resource definition might resemble the following example:

```
{  
  "name": "[concat(variables('vmName'), '/', 'ConfigureIIS')]",  
  "type": "Microsoft.Compute/virtualMachines/extensions",  
  "apiVersion": "2018-06-01",  
  "location": "[parameters('location')]",  
  "properties": {}  
}
```

The values for the **type** and **apiVersion** properties come directly from the Azure documentation. **Properties** is required, but for now can remain empty.

You know that your existing VM template has a parameter named *location*. The previous example uses the built-in **parameters** function to read that value.

The **name** property follows a special convention. The previous example uses the built-in **concat** function to concatenate (or *combine*) multiple strings. The complete string contains the VM name followed by a slash (/) character, followed by a name you choose (here, it's ConfigureIIS). The VM name helps the template identify which VM resource to run the script on.

Specify additional properties

Next, you might add in any additional properties that you require. You need the location (or *URI*) of the script file, and you might also include the resource's publisher name, its type, and version.

Referring back to the documentation, you need the following values, listed here using "dot" notation:

- **properties.publisher**
- **properties.type**
- **properties.typeHandlerVersion**
- **properties.autoUpgradeMinorVersion**
- **properties.settings.fileUris**
- **properties.protectedSettings.commandToExecute**

Your CSE resource now resembles this example:

```
{  
  "name": "[concat(variables('vmName'), '/', 'ConfigureIIS')]",
```

```
"type": "Microsoft.Compute/virtualMachines/extensions",
"apiVersion": "2018-06-01",
"location": "[parameters('location')]",
"properties": {
    "publisher": "Microsoft.Compute",
    "type": "CustomScriptExtension",
    "typeHandlerVersion": "1.9",
    "autoUpgradeMinorVersion": true,
    "settings": {
        "fileUris": [
            "https://raw.githubusercontent.com/Microsoft/PartsUnlimited/master/
Labfiles/AZ-400T05_Implementing_Application_Infrastructure/M01/configure-iis.
ps1"
        ]
    },
    "protectedSettings": {
        "commandToExecute": "powershell -ExecutionPolicy Unrestricted -File
configure-iis.ps1"
    }
}
}
```

Specify dependent resources

You can't run the CSE until the VM is available. All template resources provide a **dependsOn** property. This property helps Resource Manager determine the correct order to apply resources.

Here's what your template resource might be like after you add the **dependsOn** property:

```
{
    "name": "[concat(variables('vmName'), '/', 'ConfigureIIS')]",
    "type": "Microsoft.Compute/virtualMachines/extensions",
    "apiVersion": "2018-06-01",
    "location": "[parameters('location')]",
    "properties": {
        "publisher": "Microsoft.Compute",
        "type": "CustomScriptExtension",
        "typeHandlerVersion": "1.9",
        "autoUpgradeMinorVersion": true,
        "settings": {
            "fileUris": [
                "https://raw.githubusercontent.com/Microsoft/PartsUnlimited/master/
Labfiles/AZ-400T05_Implementing_Application_Infrastructure/M01/configure-iis.
ps1"
            ]
        },
        "protectedSettings": {
            "commandToExecute": "powershell -ExecutionPolicy Unrestricted -File
configure-iis.ps1"
        }
    }
},
```

```
    "dependsOn": [
        "[resourceId('Microsoft.Compute/virtualMachines/', variables('vm-
Name'))]"
    ]
}
```

The bracket ([]) syntax means that you can provide an array (or *list*) of resources that must exist before applying this resource.

There are multiple ways to define a resource dependency. You can provide just its name, such as SimpleWinVM, its full name (including its namespace, type, and name), such as Microsoft.Compute/virtualMachines/SimpleWinVM, or its resource ID.

Our example uses the built-in **resourceId** function to get the VM's resource ID using its full name. This helps clarify which resource you're referring to and can help avoid ambiguity when more than one resource has a similar name.

The existing template provides a `vmName` variable that defines the VM's name. Our example uses the built-in **variables** function to read it.

Finalize and Deploy

In the previous example, the PowerShell script we call will enable Internet Information Services (IIS) in your Windows VM, and then create home page content for you. All that remains is to either include this resource definition in the deployment template or link them, validate the templates, and then deploy them to Azure. (We will deploy this template later in this module.)

Subscription-Level Resources

Typically, you deploy resources to a resource group in your Azure subscription. However, you can use subscription-level deployments to create resource groups and resources that apply across your subscription.

To create a resource group in an Azure Resource Manager template, define a *Microsoft.Resources/resourceGroups* resource with a name and location for the resource group. You can create a resource group and deploy resources to that resource group in the same template.

There are some services that you might want to deploy across a subscription and make available to all resource groups such as:

- Azure Policy. Go to [Overview of the Azure Policy service¹⁹](#) for more information.
- RBAC. Go to [What is role-based access control \(RBAC\) for Azure resources?²⁰](#) for more information.
- Azure Security Center. Go to [What is Azure Security Center?²¹](#) for more information.

Name and location

When deploying to your subscription, you must provide a location for the deployment. You can also optionally provide a name for the deployment. If you don't specify a deployment name, the template

¹⁹ <https://docs.microsoft.com/en-us/azure/governance/policy/overview>

²⁰ <https://docs.microsoft.com/en-us/azure/role-based-access-control/overview>

²¹ <https://docs.microsoft.com/en-us/azure/security-center/security-center-intro>

name is used as the deployment name. For example, deploying a template named `azuredeploy.json` creates a default deployment name of `azuredeploy`.

The location of subscription level deployments is immutable. You can't create a deployment in one location when there's an existing deployment with the same name, even if it's in a different location. If you get the error code **InvalidDeploymentLocation**, either use a different name, or use the same location as the previous deployment for that name.

Using template functions

There are some important considerations when using template functions for subscription-level deployments:

- The **resourceGroup()** function is not supported.
- The **resourceId()** function is supported. You use it to retrieve the resource ID for resources that are used at subscription-level deployments. For example, you can get the resource ID for a policy definition with `resourceId('Microsoft.Authorization/policyDefinitions/', parameters('roleDefinition'))`
- The **reference()** and **list()** functions are supported.

You can review more specific details about subscription-level deployments at [Create resource groups and resources at the subscription level](#).²²

Why Use ARM Templates

Using Resource Manager templates will make your deployments faster and more repeatable. For example, you no longer have to create a VM in the portal, wait for it to finish, and then create the next VM. Resource Manager takes care of the entire deployment for you.

Here are some other template benefits to consider:

- Templates improve consistency. Resource Manager templates provide a common language for you and others to describe your deployments. Regardless of the tool or SDK that you use to deploy the template, the structure, format, and expressions inside the template remain the same.
- Templates help express complex deployments. Templates enable you to deploy multiple resources in the correct order. For example, you wouldn't want to deploy a VM prior to creating an operating system (OS) disk or network interface. Resource Manager maps out each resource and its dependent resources, and creates dependent resources first. Dependency mapping helps ensure that the deployment is carried out in the correct order.
- Templates reduce manual, error-prone tasks. Manually creating and connecting resources can be time consuming, and it's easy to make mistakes. Resource Manager ensures that the deployment happens the same way every time.
- Templates are code. Templates express your requirements through code. Think of a template as a type of Infrastructure as Code that can be shared, tested, and versioned similar to any other piece of software. Also, because templates are code, you can create a record that you can follow. The template code documents the deployment. Also, most users maintain their templates under some kind of revision control, such as GIT. When you change the template, its revision history also documents how the template (and your deployment) has evolved over time.
- Templates promote reuse. Your template can contain parameters that are filled in when the template runs. A parameter can define a username or password, a domain name, and other necessary items.

²² <https://docs.microsoft.com/en-us/azure/azure-resource-manager/deploy-to-subscription?view=sql-server-2017>

Template parameters also enable you to create multiple versions of your infrastructure, such as staging and production, while still utilizing the exact same template.

- Templates are linkable. You can link Resource Manager templates together to make the templates themselves modular. You can write small templates that each define a piece of a solution, and then combine them to create a complete system.

Create Azure Resources using Azure CLI

What is Azure CLI

Azure CLI is a command-line program that you use to connect to Azure and execute administrative commands on Azure resources. It runs on Linux, macOS, and Windows operating systems, and allows administrators and developers to execute their commands through a terminal or a command-line prompt (or script), instead of a web browser. For example, to restart a VM, you would use a command such as:

```
az vm restart -g MyResourceGroup -n MyVm
```

Azure CLI provides cross-platform command-line tools for managing Azure resources. You can install this locally on computers running the Linux, macOS, or Windows operating systems. You can also use Azure CLI from a browser through Azure Cloud Shell.

In both cases, you can use Azure CLI interactively or through scripts:

- Interactive. For Windows operating systems, launch a shell such as cmd.exe, or for Linux or macOS, use Bash. Then issue the command at the shell prompt.
- Scripted. Assemble the Azure CLI commands into a shell script using the script syntax of your chosen shell, and then execute the script.

Demonstration-Installing Azure CLI

Let's install Azure CLI on your local machine, and then run a command to verify your installation. The method you use for installing Azure CLI depends on your computer's operating system. Choose from the following steps for your operating system.

Windows

You install Azure CLI on the Windows operating system using the MSI installer:

To install Azure CLI on the Windows operating system:

1. Go to <https://aka.ms/installazurecliwindows>, and in the browser security dialog box, select **Run**.
2. In the installer, accept the license terms, and then select **Install**.
3. In the **User Account Control** dialog, select **Yes**.

Linux

For computers running the Linux operating system, install Azure CLI on Ubuntu Linux using Advanced Packaging Tool (**apt**) and the Bash command line.

The recommended package manager differs by OS and distribution:

- Ubuntu: **apt-get**
- Red Hat: **yum**
- OpenSUSE: **zypper**

Note: The following commands are for Ubuntu version 18.04. Other versions and Linux distributions have different instructions. Check the official [Install the Azure CLI²³](#) installation documentation if you are using a different Linux version.

To install Azure CLI on Ubuntu:

1. Modify your sources list so that the Microsoft repository is registered, and the package manager can locate the Azure CLI package:

```
AZ_REPO=$(lsb_release -cs)
echo "deb [arch=amd64] https://packages.microsoft.com/repos/azure-cli/
$AZ_REPO main" | \
sudo tee /etc/apt/sources.list.d/azure-cli.list
```

2. Import the encryption key for the Microsoft Ubuntu repository. This will allow the package manager to verify that Azure CLI package you install comes from Microsoft:

```
curl https://packages.microsoft.com/keys/microsoft.asc | sudo apt-key add -
```

3. Install Azure CLI:

```
sudo apt-get install apt-transport-https
sudo apt-get update && sudo apt-get install azure-cli
```

macOS

You install Azure CLI on macOS using Homebrew package manager.

Note: If the **brew** command is unavailable, you might need to install the Homebrew package manager. For details, go to the [Homebrew²⁴](#) website.

To install Azure CLI on macOS:

1. Update your brew repository to make sure you have the latest Azure CLI package:

```
brew update
```

2. Install Azure CLI:

```
brew install azure-cli
```

Verify Azure CLI installation

You run Azure CLI by opening a Bash shell for Linux or macOS, from the command prompt, or Windows PowerShell.

To verify the Azure CLI installation was successful:

- Start Azure CLI and verify your installation by running the version check:

```
az --version
```

²³ <https://docs.microsoft.com/cli/azure/install-azure-cli>

²⁴ <https://brew.sh/>

Note: Running Azure CLI from Windows PowerShell has some advantages over running Azure CLI from the Windows command prompt. PowerShell provides more tab completion features than the command prompt.

Working with Azure CLI

Azure CLI lets you control nearly every aspect of every Azure resource. You can work with Azure resources such as resource groups, storage, VMs, Azure Active Directory (Azure AD), containers, and machine learning.

Commands in the CLI are structured in **groups** and **subgroups**. Each group represents a service provided by Azure, and the subgroups divide commands for these services into logical groupings. For example, the **storage** group contains subgroups, including **account**, **blob**, **storage**, and **queue**.

So, how do you find the particular commands you need? One way is to use the **az find** command. For example, if you want to find commands that might help you manage a storage blob, you can use the following **find** command:

```
az find -q blob
```

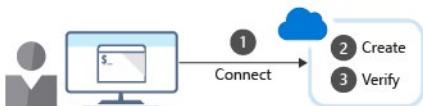
If you already know the name of the command you want, the **-help** argument for that command will get you more detailed information on the command, and for a command group, a list of the available subcommands. For example, here's how you would get a list of the subgroups and commands for managing blob storage:

```
az storage blob --help
```

Creating resources

When creating a new Azure resource, typically there are three high-level steps:

1. Connect to your Azure subscription.
2. Create the resource.
3. Verify that creation was successful.



1. Connect

Because you're working with a local Azure CLI installation, you'll need to authenticate before you can execute Azure commands. You do this by using the Azure CLI **login** command:

```
az login
```

Azure CLI will typically launch your default browser to open the Azure sign-in page. If this doesn't work, follow the command-line instructions and enter an authorization code in the [Enter Code] (<https://aka.ms/devicelogin>) **Enter Code**²⁵ dialog box.

After a successful sign in, you'll be connected to your Azure subscription.

2. Create

You'll often need to create a new resource group before you create a new Azure service, so we'll use resource groups as an example to show how to create Azure resources from the Azure CLI.

The Azure CLI **group create** command creates a resource group. You must specify a name and location. The *name* parameter must be unique within your subscription. The *location* parameter determines where the metadata for your resource group will be stored. You use strings like "West US", "North Europe", or "West India" to specify the location. Alternatively, you can use single word equivalents, such as "westus", "northeurope", or "westindia".

The core syntax to create a resource group is:

```
az group create --name <name> --location <location>
```

3. Verify installation

For many Azure resources, Azure CLI provides a **list** subcommand to get resource details. For example, the Azure CLI **group list** command lists your Azure resource groups. This is useful to verify whether resource group creation was successful:

```
az group list
```

To get more concise information, you can format the output as a simple table:

```
az group list --output table
```

If you have several items in the group list, you can filter the return values by adding a **query** option using, for example, the following command:

```
az group list --query "[?name == '<rg name>']"
```

Note: You format the query using **JMESPath**, which is a standard query language for JSON requests. You can learn more about this filter language at <http://jmespath.org>²⁶

Using Azure CLI in scripts

If you want to use the Azure CLI commands in scripts, you'll need to be aware of any issues around the shell or environment that you use for running the script. For example, in a bash shell, you can use the following syntax when setting variables:

```
variable="value"  
variable=integer
```

²⁵ <https://aka.ms/devicelogin>

²⁶ <http://jmespath.org>

If you use a PowerShell environment for running Azure CLI scripts, you'll need to use the following syntax for variables:

```
$variable="value"  
$variable=integer
```

Demonstration-Run Templates using Azure CLI

We have added the custom script extension to a sample deployment JSON template. You can review the sample Azure Deploy template with the custom script extension resource included here.

Prerequisites

To perform these steps you need an Azure subscription. If you don't have one already, you can create one by following the steps outlined on the [Create your Azure free account today²⁷](#) webpage.

Steps

In the following steps we will deploy the template and verify the result using Azure CLI:

1. Create a resource group to deploy your resources to, by running the following command:

```
az group create --name <resource group name> --location <your nearest  
datacenter>
```

2. From Cloud Shell, run the **curl** command to download the template you used previously from GitHub:

```
curl https://raw.githubusercontent.com/Microsoft/PartsUnlimited/master/  
Labfiles/AZ-400T05_Implementing_Application_Infrastructure/M01/azuredeploy.  
json > C:\temp\azuredeploy.json
```

3. Validate the template by running the following command, substituting the values with your own:

```
az group deployment validate \  
--resource-group <rgn>[sandbox resource group name]</rgn> \  
--template-file C:\temp\azuredeploy.json \  
--parameters adminUsername=$USERNAME \  
--parameters adminPassword=$PASSWORD \  
--parameters dnsLabelPrefix=$DNS_LABEL_PREFIX
```

4. Deploy the resource by running the following command, substituting the same values as earlier:

```
az group deployment create \  
--name MyDeployment \  
--resource-group <rgn>[sandbox resource group name]</rgn> \  
--template-file azuredeploy.json \  
--parameters adminUsername=$USERNAME \  
--parameters adminPassword=$PASSWORD \  
--parameters dnsLabelPrefix=$DNS_LABEL_PREFIX
```

5. Obtain the IP address by running the following command:

²⁷ <https://azure.microsoft.com/en-us/free/>

```
IPADDRESS=$(az vm show \
    --name SimpleWinVM \
    --resource-group <rgn>[sandbox resource group name]</rgn> \
    --show-details \
    --query [publicIps] \
    --output tsv)
```

6. Run **curl** to access your web server and verify that the deployment and running of the custom script extension was successful:

```
curl $IPADDRESS
```

You will have the following output:

```
<html><body><h2>Welcome to Azure! My name is SimpleWinVM.</h2></body></html>
```

- ✓ Note: Don't forget to delete any resources you deployed to avoid incurring additional costs from them.

Extensions

Azure CLI offers the ability to load extensions. *Extensions* are Python wheels that aren't shipped as part of the CLI but run as CLI commands. With extensions, you gain access to experimental and pre-release commands along with the ability to create your own Azure CLI interfaces.

Find Azure CLI extensions

To view the extensions that Microsoft provides and maintains, use the **az extension list-available** command:

```
az extension list-available --output table
```

Name	Version	Summary	Preview	Installed

aem	0.1.1	Manage Azure Enhanced Monitoring Extensions for SAP	False	False
aks-preview	0.1.0	Provides a preview for upcoming AKS features	True	False
alias	0.5.2	Support for command aliases	True	False
azure-batch-cli-extensions	2.5.1	Additional commands for working with Azure Batch service	False	False
azure-cli-iot-ext	0.6.1	Provides the data plane command layer for Azure IoT Hub, IoT Edge and IoT Device Provisioning Service	False	False
azure-devops	0.1.0	Tools for managing Azure DevOps.	True	False
azure-firewall	0.1.1	Manage Azure Firewall resources.	True	False
dev-spaces-preview	0.1.6	Dev Spaces provides a rapid, iterative Kubernetes development experience for teams.	True	False
dms-preview	0.6.0	Support for new Database Migration Service scenarios.	True	False
dns	0.0.2	An Azure CLI Extension for DNS zones	False	False
eventgrid	0.4.0	Support for Azure EventGrid 2018-09-15-preview features	True	False
express-route	0.1.3	Manage ExpressRoutes with preview features.	True	False
express-route-cross-connection	0.1.0	Manage customer ExpressRoute circuits using an ExpressRoute cross-connection.	False	False
find	0.3.0	Intelligent querying for CLI information.	True	False
front-door	0.1.1	Manage networking Front Doors.	True	False
image-copy-extension	0.0.9	Support for copying managed VM images between regions	False	False
interactive	0.4.1	Microsoft Azure Command-Line Interactive Shell	True	False
keyvault-preview	0.1.3	Preview Azure Key Vault commands.	True	False
log-analytics	0.1.3	Support for Azure Log Analytics query capabilities.	True	False
managementgroups	0.1.0	An Azure CLI Extension for Management Groups	False	False
managementpartner	0.1.2	Support for Management Partner preview	False	False
mesh	0.10.2	Support for Microsoft Azure Service Fabric Mesh - Public Preview	True	False
rdms-vnet	10.0.0	Support for Virtual Network rules in Azure MySQL and Azure PostgreSQL resources	False	False
resource-graph	0.1.8	Support for querying Azure resources with Resource Graph.	True	False
sap-hana	0.1.6	Additional commands for working with SAP HanaOnAzure instances.	False	False
signalr	0.1.0	Support for signalr management preview.	True	False
sqlvm-preview	0.1.0	Tools for managing SQL virtual machines, groups and availability group listeners.	True	False
storage-preview	0.2.0	Provides a preview for upcoming storage features.	True	False
subscription	0.1.1	Support for subscription management preview.	False	False
virtual-network-tap	0.1.0	Manage virtual network taps (VTA).	True	False
virtual-wan	0.1.0	Manage virtual WAN, hubs, VPN gateways and VPN sites.	True	False
webapp	0.2.16	Additional commands for Azure AppService.	True	False

Install Azure CLI extensions

After you find an extension you want, use the following command to install it:

```
az extension add --name <extension-name>
```

Uninstall Azure CLI extensions

You can uninstall extensions using the following command:

```
az extension remove --name <extension-name>
```

Note: For more detail about Azure CLI extensions and their use, view the video [Azure CLI Extensions²⁸](#).

Azure CLI VM extensions

You can also run VM extensions using Azure CLI. For example, if a VM requires a software installation, anti-virus protection, or Docker configuration, you can use a VM extension to complete these tasks. You can bundle extensions with a new VM deployment or run them against any existing system. To configure VM extensions using Azure CLI, use the following command:

```
az vm extension <sub-command> <parameter>
```

²⁸ <https://azure.microsoft.com/en-us/resources/videos/azure-friday-azure-cli-extensions/?azure-portal=true>

Create Azure Resources by using Azure PowerShell

PowerShell components

PowerShell is made up of two core components, which are as follows:

PowerShell cmdlets

A PowerShell command is called a *cmdlet* (pronounced "command-let"). A *cmdlet* is a command that manipulates a single feature. The term cmdlet is intended to imply that it is a small command. By convention, cmdlet authors are encouraged to keep cmdlets simple and single purpose.

The base PowerShell product ships with cmdlets that work with features such as sessions and background jobs. You add modules to your PowerShell installation to get cmdlets that manipulate other features. For example, there are third-party modules to work with FTP, administer your operating system, and access the file system.

Cmdlets follow a verb-noun naming convention; for example, **Get-Process**, **Format-Table**, and **Start-Service**. There is also a convention for verb choice. For example:

- **get** retrieves data.
- **set** inserts or updates data.
- **format** formats data.
- **out** directs output to a destination.

Cmdlet authors are encouraged to include a help file for each cmdlet. The **Get-Help** cmdlet displays the help file for any cmdlet. For example, you could get help on the **Get-ChildItem** cmdlet with the following statement:

```
Get-Help Get-ChildItem -detailed
```

PowerShell modules

Cmdlets are shipped in _modules. A *PowerShell module* is a DLL file that includes the code to process each available cmdlet. You load cmdlets into PowerShell by loading the module containing them. You can get a list of loaded modules using the **Get-Module** command:

```
Get-Module
```

This will output something like the following code in table format:

ModuleType	Version	Name	ExportedCommands
Manifest	3.1.0.0	Microsoft.PowerShell.Management	{Add-Computer,
		Add-Content, Checkpoint-Computer, Clear-Con...	
Manifest	3.1.0.0	Microsoft.PowerShell.Utility	{Add-Member,
		Add-Type, Clear-Variable, Compare-Object...}	
Binary	1.0.0.1	PackageManagement	{Find-Package,
		Find-PackageProvider, Get-Package, Get-Pack...	
Script	1.0.0.1	PowerShellGet	{Find-Command,

```
Find-DscResource, Find-Module, Find-RoleCap...
Script      2.0.0      PSReadline
LineKeyHandler, Get-PSReadLineOption, Remove-PS...
```

What is Azure PowerShell

Azure PowerShell is a module that you add to Windows PowerShell or PowerShell Core to enable you to connect to your Azure subscription and manage resources. Azure PowerShell requires PowerShell to function. PowerShell provides services such as the shell window and command parsing. Azure PowerShell adds the Azure-specific commands.

For example, Azure PowerShell provides the **New-AzVm** command that creates a virtual machine inside your Azure subscription. To use it, you would launch the PowerShell application and then issue a command such as the following command:

```
New-AzVm
  -ResourceGroupName "CrmTestingResourceGroup"
  -Name "CrmUnitTests"
  -Image "UbuntuLTS"
  ...
```

Azure PowerShell is also available two ways: inside a browser via the Azure Cloud Shell, or with a local installation on Linux, macOS, or the Windows operating system. In both cases, you have two modes from which to choose: you can use it in interactive mode in which you manually issue one command at a time, or in scripting mode where you execute a script that consists of multiple commands.

Installing Azure PowerShell on Different Platforms

Let's look at the general installation process for PowerShell across several platforms.

What must be installed?

We'll go through the actual PowerShell installation instructions shortly, but let's look at the two components which make up Azure PowerShell:

- The base PowerShell product. This comes in two variants: PowerShell on Windows, and PowerShell Core on macOS and Linux.
- The Azure PowerShell module. This extra module must be installed to add the Azure-specific commands to PowerShell.

Note: PowerShell is included with the Windows operating system. However, it might need to be updated, which you should generally do. For Linux and macOS, you will need to install PowerShell Core.

After you have installed the base product, you then add the Azure PowerShell module to your installation. We will cover just the base PowerShell product installation in the remainder of this topic.

How to install PowerShell Core

On both Linux and macOS, you use a package manager to install PowerShell Core. The recommended package manager differs by OS and distribution.

NOTE: PowerShell Core is available in the Microsoft repository, so you'll first need to add that repository to your package manager.

Linux

On Linux, the package manager will change based on the Linux distribution you choose:

Distribution(s)	Package manager
Ubuntu, Debian	apt-get
Red Hat, CentOS	yum
OpenSUSE	zypper
Fedora	dnf

The following steps will install PowerShell Core on Ubuntu Linux using the Advanced Packaging Tool (**apt**) and the Bash command line.

1. Import the encryption key for the Microsoft Ubuntu repository. This enables the package manager to verify that the PowerShell Core package you install comes from Microsoft:

```
curl https://packages.microsoft.com/keys/microsoft.asc | sudo apt-key add -
```

2. Register the Microsoft Ubuntu repository so the package manager can locate the PowerShell Core package:

```
sudo curl -o /etc/apt/sources.list.d/microsoft.list https://packages.microsoft.com/config/ubuntu/18.04/prod.list
```

3. Update the list of packages:

```
sudo apt-get update
```

4. Install PowerShell Core:

```
sudo apt-get install -y powershell
```

5. Start PowerShell to verify that it installed successfully:

```
pwsh
```

Mac

On macOS, you will use Homebrew to install PowerShell Core.

On macOS, the first step is to install PowerShell Core. You do this using the Homebrew package manager.

Note: If the **brew** command is unavailable, you might need to install the Homebrew package manager. For details see the [Homebrew website²⁹](#).

1. Install Homebrew-Cask to obtain more packages, including the PowerShell Core package:

```
brew tap caskroom/cask
```

²⁹ <https://brew.sh/>

2. Install PowerShell Core:

```
brew cask install powershell
```

3. Start PowerShell Core to verify that it installed successfully:

```
pwsh
```

Windows

PowerShell is included with the Windows operating system. However, there could be an update available for your machine, so be sure to check. (The Azure support we are going to use requires PowerShell version 5.0 or higher.) You can check the version you have installed through the following steps:

1. Open the **Start** menu and type **Windows PowerShell**. There could be multiple shortcut links available, including:
 - Windows PowerShell. This is the 64-bit version and generally what you should choose.
 - Windows PowerShell (x86). This is a 32-bit version installed on a 64-bit Windows operating system.
 - Windows PowerShell ISE. You will use the Integrated Scripting Environment (ISE) for writing scripts in PowerShell.
 - Windows PowerShell ISE (x86). This is the 32-bit version of the ISE.
2. Select the Windows PowerShell icon.
3. In PowerShell, type the following command to determine the version of PowerShell installed.

```
$PSVersionTable.PSVersion
```

If the version number is lower than 5.0, follow these instructions for **upgrading existing Windows PowerShell**³⁰.

Demonstration-Install Azure PowerShell

What is the Az module?

Az is the formal name for the Azure PowerShell module containing cmdlets to work with Azure features. It contains hundreds of cmdlets that let you control nearly every aspect of every Azure resource. You can work with the following features, and more:

- Resource groups
- Storage
- VMs
- Azure AD
- Containers
- Machine learning

³⁰ <https://docs.microsoft.com/en-us/powershell/scripting/install/installing-windows-powershell?view=powershell-7#upgrading-existing-windows-powershell>

This module is an open source component **available on GitHub**³¹.

NOTE: You might have seen or used Azure PowerShell commands that used an **-AzureRM** format. In December 2018 Microsoft released for general availability the AzureRM module replacement with the Az module. This new module has several features, notably a shortened cmdlet noun prefix of **-Az**, which replaces **AzureRM**. The **Az** module ships with backwards compatibility for the AzureRM module, so the **-AzureRM** cmdlet format will work. However, going forward you should transition to the Az module and use the **-Az** commands.

Install the Az module

The Az module is available from a global repository called the *PowerShell Gallery*. You can install the module onto your local machine through the **Install-Module** command. You need an elevated PowerShell shell prompt to install modules from the PowerShell Gallery.

To install the latest Azure PowerShell module, complete the following steps:

1. Open the **Start** menu, and type **Windows PowerShell**.
2. Right-click the **Windows PowerShell** icon, and select **Run as administrator**.
3. In the **User Account Control** dialog, select **Yes**.
4. Type the following command, and then press Enter:

```
Install-Module -Name Az -AllowClobber
```

This command installs the module for all users by default. (It's controlled by the scope parameter.) The command relies on the NuGet package manager to retrieve components.

Depending on the NuGet version you have installed you might get a prompt to download and install the latest version, as below:

```
NuGet provider is required to continue
PowerShellGet requires NuGet provider version '2.8.5.201' or newer to
interact with NuGet-based repositories. The NuGet
provider must be available in 'C:\Program Files (x86)\PackageManagement\
ProviderAssemblies' or
'C:\Users\<username>\AppData\Local\PackageManagement\ProviderAssemblies'.
You can also install the NuGet provider by running
'Install-PackageProvider -Name NuGet -MinimumVersion 2.8.5.201 -Force'. Do
you want PowerShellGet to install and import
the NuGet provider now?
```

By default, the PowerShell Gallery isn't configured as a trusted repository for PowerShellGet. The first time you use the PowerShell Gallery, you will see the following prompt:

```
You are installing the modules from an untrusted repository. If you trust
this repository, change its
InstallationPolicy value by running the Set-PSRepository cmdlet. Are you
sure you want to install the modules from
'PSGallery'?
```

³¹ <https://github.com/Azure/azure-powershell>

Demonstration-Run Templates with PowerShell

When creating new Azure resources using Resource Manager templates, similar to working with Azure CLI there are typically three steps involved:

1. Connect to your Azure subscription.
2. Create the resource.
3. Verify that creation was successful.

1. Connect

Because you're working with a local install of the PowerShell, you'll need to authenticate before you can execute Azure commands. To do this, open the PowerShell ISE, or a PowerShell console as administrator, and run the following command:

```
Connect-AzAccount
```

Follow the command-line instructions and enter an authorization code at <https://aka.ms/devicelogin>.

After successfully signing in, your account and subscription details should display in the PowerShell console window. You must now select either a subscription or context, in which you will deploy your resources. If only one subscription is present it will set the context to that subscription by default. Otherwise you can specify the subscription to deploy resources into by running the following commands in sequence:

```
get-azcontext  
Set-AzContext -subscription < your subscription ID >
```

2. Create

You'll often need to create a new resource group before you create a new Azure service or resource. We'll use resource groups as an example to show how to create Azure resources from Azure CLI.

The Azure CLI **group create** command creates a resource group. You must specify a name and location. The name must be unique within your subscription, and the location determines where the metadata for your resource group will be stored. You use strings such as West US, North Europe, or West India to specify the location. Alternatively, you can use single word equivalents, such as westus, northeurope, or westindia.

First, create the resource group into which we will deploy our resources using the following commands.

```
New-AzResourceGroup -Name < resource group name > -Location < your nearest datacenter >
```

```
New-AzResourceGroupDeployment -Name rg9deployment1 -ResourceGroupName rg9 -TemplateUri https://raw.githubusercontent.com/Microsoft/PartsUnlimited/master/Labfiles/AZ-400T05_Implementing_Application_Infrastructure/M01/azuredeploy.json
```

You will be prompted to enter values for:

- Adminusername. For example, azureuser.

- Password. Any compliance password will work, for example Passw0rd0134.
- DnsLabelprefix. This is any unique DNS name, such as your initials and random numbers.

To make scripts free of manual input, you can create a .ps1 file, and then enter all the commands and inputs. You could use parameter values in the script to define the *username*, *password* and *dnslabelprefix* values, and then run the PowerShell file without input. See the file **build.ps1**³² as an example of how you can do this.

Note: In the previous example, we called a publicly available template on GitHub. You could also call a local template or a secure storage location, and you could define the template filename and location as a variable for use in the script. You can also specify the mode of deployment, including incremental or complete. For more information, see **New-AzResourceGroupDeployment**³³.

3. Verify

Once you have successfully deployed the template, you need to verify the deployment. To do this, run the following commands:

```
Get-AzVM
```

Note the VM name, then run the following command to obtain additional VM details:

```
get-azvm -Name < your VM name i.e. SimpleWinVM > -resourcegroupname < your resource group name >
```

Note the extension value listed.

You can also list the VMs in your subscription with the **Get-AzVM -Status** command. This can also specify a VM with the **-Name** property. In the following example, we assign it to a PowerShell variable:

```
$vm = Get-AzVM -Name < your VM name i.e. SimpleWinVM > -ResourceGroupName < your resource group name >
```

The interesting thing is this is an object you can interact with. For example, you can take that object, make changes, and then push changes back to Azure with the **Update-AzVM** command:

```
$ResourceGroupName = "ExerciseResources"  
$vm = Get-AzVM -Name MyVM -ResourceGroupName $ResourceGroupName  
$vm.HardwareProfile.vmSize = "Standard_A3"
```

```
Update-AzVM -ResourceGroupName $ResourceGroupName -VM $vm
```

Note: Depending on your datacenter location, you could receive an error related to the VM size not being available in your region. You can modify the vmSize value to one that is available in your region.

PowerShell's interactive mode is appropriate for one-off tasks. In our example, we'll likely use the same resource group for the lifetime of the project, which means that creating it interactively is reasonable. Interactive mode is often quicker and easier for this task than writing a script and then executing it only once.

³² <https://github.com/Microsoft/PartsUnlimited/blob/master/build.ps1?azure-portal=true>

³³ <https://docs.microsoft.com/en-us/powershell/module/azurerm.resources/new-azurermresourcegroupdeployment?view=azurerm-ps-6.13.0?azure-portal=true>

VM Extensions with PowerShell

Azure VM extensions run on existing VMs, which is useful when you need to make configuration changes or recover connectivity on an already deployed VM. You can bundle VM extensions with Resource Manager template deployments. By using extensions with Resource Manager templates, You can deploy and configure Azure VMs without post-deployment intervention.

PowerShell extension commands

There are several PowerShell commands for running individual extensions. To see a list, use the **Get-Command** and filter on **Extension**, as in the following example:

```
Get-Command Set-*Extension* -Module Az.Compute
```

You should see options such as:

CommandType	Name	Version
Source		
Cmdlet	Set-AzVMAccessExtension	1.0.0
Az.Compute	Set-AzVMADDomainExtension	1.0.0
Az.Compute	Set-AzVMAEMExtension	1.0.0
Az.Compute	Set-AzVMBackupExtension	1.0.0
Az.Compute	Set-AzVMBginfoExtension	1.0.0
Az.Compute	Set-AzVMChefExtension	1.0.0
Az.Compute	Set-AzVMCustomScriptExtension	1.0.0
Az.Compute	Set-AzVMDiagnosticsExtension	1.0.0
Az.Compute	Set-AzVMDiskEncryptionExtension	1.0.0
Az.Compute	Set-AzVMDscExtension	1.0.0
Az.Compute	Set-AzVMExtension	1.0.0
Az.Compute	Set-AzVMSqlServerExtension	1.0.0
Az.Compute	Set-AzVmssDiskEncryptionExtension	1.0.0
Az.Compute		

Note: Some PowerShell commands might appear to offer similar functionality, such as **Set-AzureRMVM-CustomScriptExtension**³⁴ and **Set-AzureRMVMExtension**³⁵. However, there can be subtle differences, including different cmdlets having different parameters available.

Also, be aware that the help pages for AzureRM module cmdlets are still applicable to Az module cmdlets. Hence they can be used for reference, as was done in the previous note.

Example of a Custom Script Extension

The following example uses the Custom Script Extension to download a script from a GitHub repository onto the target VM, and then run the script:

```
Set-AzureRmVMCustomScriptExtension -ResourceGroupName < your resource group name > `  
    -VMName "SimpleWinVM" -Name "create-file" `  
    -FileUri "https://raw.githubusercontent.com/Microsoft/PartsUnlimited/master/Labfiles/AZ-400T05_Implementing_Application_Infrastructure/M01/create-file.ps1" `  
    -Run "create-file.ps1" -Location < your nearest datacenter location >
```

To view the details of the Custom Script Extension, run the following command:

```
get-azvmextension -vmname < your VM name > -resourcegroup < your resource group name > -name < your extension name >
```

³⁴ <https://docs.microsoft.com/en-us/powershell/module/azurerm.compute/set-azurermvmcustomscriptextension?view=azurermps-6.13.0?azure-portal=true>

³⁵ <https://docs.microsoft.com/en-us/powershell/module/azurerm.compute/set-azurermvmextension?view=azurermps-6.13.0?azure-portal=true>

Desired State Configuration (DSC)

Desired State Configuration (DSC)

Desired State Configuration (DSC) is a configuration management approach that you can use for configuration, deployment, and management of systems to ensure that an environment is maintained in a state that you specify (*defined state*), and doesn't deviate from that state. Using DSC helps eliminate configuration drift and ensures state is maintained for compliance, security, and performance purposes.

Windows PowerShell DSC is a management platform in PowerShell that provides desired State. PowerShell DSC lets you manage, deploy, and enforce configurations for physical or virtual machines, including Windows and Linux machines.

For more information, visit [Windows PowerShell Desired State Configuration Overview³⁶](#).

DSC components

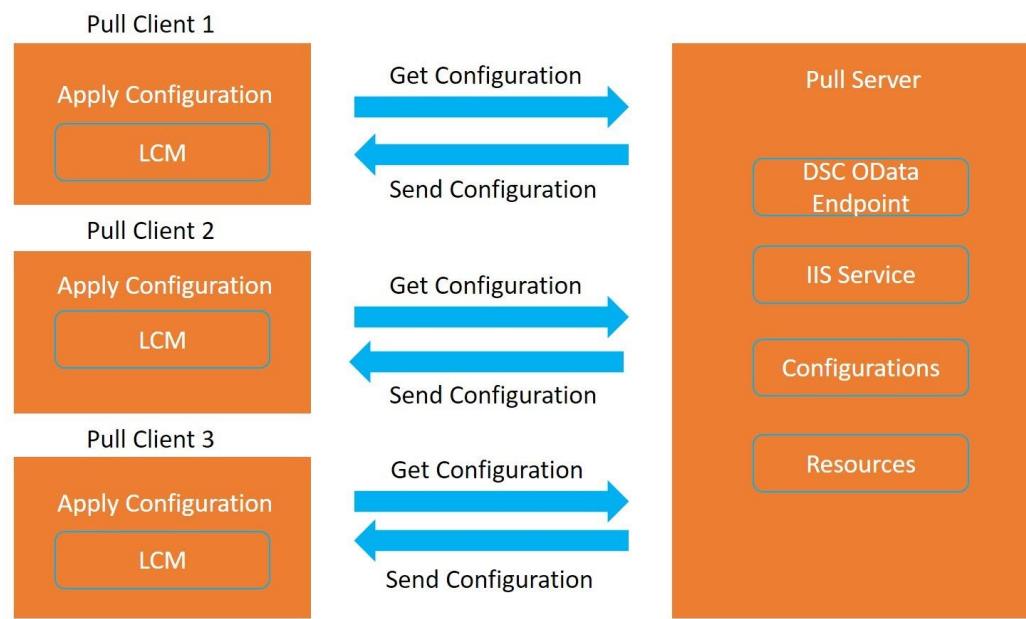
DSC consists of three primary components:

- Configurations. These are declarative PowerShell scripts that define and configure instances of resources. Upon running the configuration, DSC (and the resources being called by the configuration) will simply apply the configuration, ensuring that the system exists in the state laid out by the configuration. DSC configurations are also *idempotent*: the Local Configuration Manager (LCM) will continue to ensure that machines are configured in whatever state the configuration declares.
- Resources. They contain the code that puts and keeps the target of a configuration in the specified state. Resources reside in PowerShell modules and can be written to a model as something as generic as a file or a Windows process, or as specific as a Microsoft Internet Information Services (IIS) server or a VM running in Azure.
- Local Configuration Manager (LCM). The LCM runs on the nodes or machines you wish to configure. This is the engine by which DSC facilitates the interaction between resources and configurations. The LCM regularly polls the system using the control flow implemented by resources to ensure that the state defined by a configuration is maintained. If the system is out of state, the LCM makes calls to the code in resources to apply the configuration according what has been defined

There are two methods of implementing DSC:

1. Push mode - Where a user actively applies a configuration to a target node, and the pushes out the configuration.
2. Pull mode - Where *pull clients* are configured to get their desired state configurations from a remote pull service automatically. This remote pull service is provided by a *pull server* which acts as a central control and manager for the configurations, ensures that nodes conform to the desired state and report back on their compliance status. The pull server can be set up as an SMB-based pull server or a HTTPS-based server. HTTPS based pull-server use the Open Data Protocol (OData) with the OData Web service to communicate using REST APIs. This is the model we are most interested in, as it can be centrally managed and controlled. The diagram below provides an outline of the workflow of DSC pull mode.

³⁶ <https://docs.microsoft.com/en-us/powershell/scripting/dsc/overview/overview?view=powershell-6>



If you are not familiar with DSC, take some time to view [A Practical Overview of Desired State Configuration³⁷](#). This is a great video from the TechEd 2014 event, and it covers the basics of DSC.

Azure Automation State configuration (DSC)

Azure Automation State configuration DSC is an Azure cloud-based implementation of PowerShell DSC, available as part of Azure Automation. Azure Automation State configuration allows you to write, manage, and compile PowerShell DSC configurations, import DSC Resources, and assign configurations to target nodes, all in the cloud.

Why use Azure Automation DSC

The following outlines some of the reasons why we would consider using Azure Automation DSC:

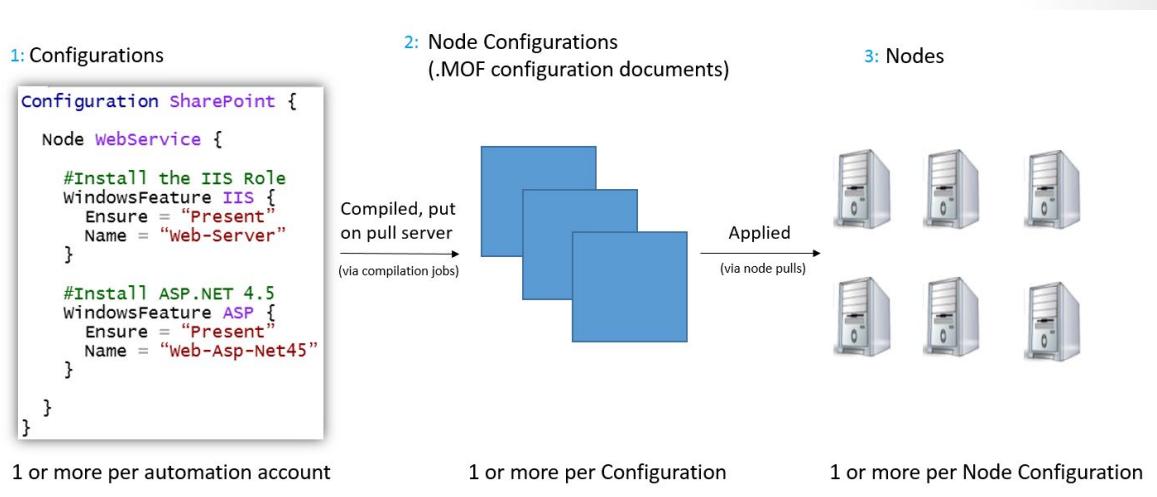
- Built-in pull server. Provides a DSC pull server similar to the Windows Feature DSC-service so that target nodes automatically receive configurations, conform to the desired state, and report back on their compliance. The built-in pull server in Azure Automation eliminates the need to set up and maintain your own pull server.
- Management of all your DSC artifacts. From either the Azure portal or PowerShell, you can manage all your DSC configurations, resources, and target nodes.
- Import reporting data into Log Analytics. Nodes that are managed with Azure Automation state configuration send detailed reporting status data to the built-in pull server. You can configure Azure Automation state configuration to send this data to your Log Analytics workspace.

³⁷ <https://channel9.msdn.com/Events/TechEd/NorthAmerica/2014/DCIM-B417#fbid=>

How Azure Automation state configuration works

The general process for how Azure Automation State configuration works is as follows:

1. Create a PowerShell script with the configuration element
2. Upload the script to Azure Automation and compile the script into a MOF file. The file is transferred to the DSC pull server, which is provided as part of the Azure Automation service. (See **Managed Object Format (MOF) file**³⁸ for more information on MOF files.)
3. Define the nodes that will use the configuration, and then apply the configuration.



DSC Configuration File

DSC configurations are Windows PowerShell scripts that define a special type of function. You can view some syntax examples and scenarios on the **Configuration syntax**³⁹ page.

DSC configuration elements

We'll provide the example configurations and then discuss the elements within them. Let's start with the following example configuration:

```
configuration LabConfig
{
    Node WebServer
    {
        WindowsFeature IIS
        {
            Ensure = 'Present'
            Name = 'Web-Server'
            IncludeAllSubFeature = $true
        }
    }
}
```

³⁸ [https://msdn.microsoft.com/en-us/library/aa823192\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/aa823192(v=vs.85).aspx)

³⁹ <https://docs.microsoft.com/en-us/powershell/scripting/dsc/configurations?view=powershell-6#configuration-syntax>

- **Configuration block.** The **Configuration** block is the outermost script block. In this case, the name of the configuration is **LabConfig**. Notice the curly brackets to define the block.
- **Node block.** There can be one or more **Node** blocks. These define the nodes (computers and VMs) that you are configuring. In this example, the node targets a computer called **WebServer**. You could also call it **localhost** and use it locally on any server.
- **Resource blocks.** There can be one or more resource blocks. This is where the configuration sets the properties for the resources. In this case, there is one resource block called **WindowsFeature**. Notice the parameters that are defined. (You can read more about resource blocks at **DSC resources**⁴⁰.

Here is another example:

```
Configuration MyDscConfiguration
{
    param
    (
        [string[]]$ComputerName='localhost'
    )

    Node $ComputerName
    {
        WindowsFeature MyFeatureInstance
        {
            Ensure = 'Present'
            Name = 'RSAT'
        }

        WindowsFeature My2ndFeatureInstance
        {
            Ensure = 'Present'
            Name = 'Bitlocker'
        }
    }
}

MyDscConfiguration
```

In this example, you specify the name of the node by passing it as the *ComputerName* parameter when you compile the configuration. The name defaults to "localhost".

Within a Configuration block, you can do almost anything that you normally could in a PowerShell function. You can also create the configuration in any editor, such as PowerShell ISE, and then save the file as a PowerShell script with a .ps1 file type extension.

Demonstration-Import and Compile

After creating your DSC configuration file, you must import the file and compile it to the DSC pull server. Compiling will create the MOF file. Read more about this at **Compiling a DSC Configuration with the Azure portal**⁴¹.

⁴⁰ <https://docs.microsoft.com/en-us/powershell/scripting/dsc/resources/resources?view=powershell-7>

⁴¹ <https://azure.microsoft.com/en-us/documentation/articles/automation-dsc-compile/#compiling-a-dsc-configuration-with-the-azure-portal>

Import and compile configurations

To import and compile a configuration, complete the following high-level steps:

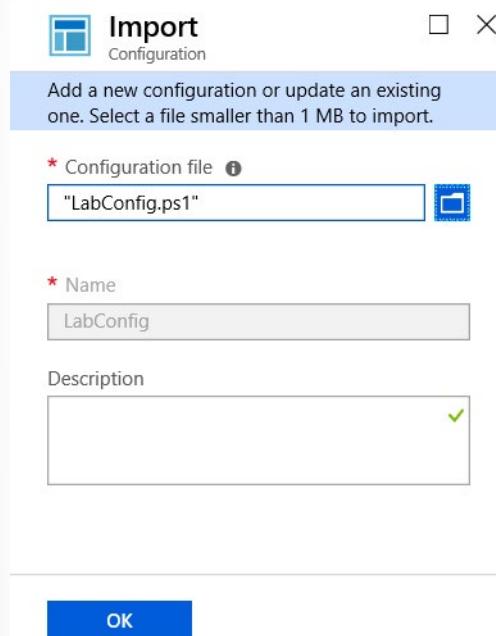
1. Create a configuration file by creating a file on your local machine. Then, copy and paste the following PowerShell code into the file, and name it **LabConfig.ps1**. This script configuration will ensure the IIS web-server role is installed on the servers:

```
configuration LabConfig
{
    Node WebServer
    {
        WindowsFeature IIS
        {
            Ensure = 'Present'
            Name = 'Web-Server'
            IncludeAllSubFeature = $true
        }
    }
}
```

2. In Azure Automation, account under **Configuration Management > State configuration (DSC)**, select the **Configurations** tab, and then select **+Add**.

The screenshot shows the 'az-auto-ac-1 - State configuration (DSC)' blade. On the left, there's a navigation menu with 'Configuration Management' (highlighted with a red box), 'Inventory', 'Change tracking', and 'State configuration (DSC)' (highlighted with a red box). At the top, there are buttons for '+ Add', 'Compose configuration', 'Refresh', and 'Reset filters'. Below these are tabs for 'Nodes' (highlighted with a red box), 'Configurations' (also highlighted with a red box), 'Compiled configurations', and 'Gallery'. A search bar says 'Search configurations...'. The main area shows a table with columns 'CONFIGURATION', 'COMPILED CONFIGURATION COUNT', and 'LAST MODIFIED'. The table displays 'No data'.

3. Point to the configuration file you want to import, and then select **OK**.



4. Once imported double click the file, select **Compile**, and then confirm by selecting **Yes**.

The screenshot shows the 'LabConfig' configuration file details page. At the top, there are three buttons: 'Compile' (highlighted with a red box), 'Export', and 'Delete'. Below this is a section titled 'Essentials' with the following details:

Resource group	az-auto-rg	Account	az-auto-ac-1
Location	westeurope	Subscription name	Pay-As-You-Go
Subscription ID	974e6e39-73eb-48b0-9226-dae31425c367	Status	Published
Last published	1/11/2019 1:00 PM	Configuration source	View configuration source

Below this is a section titled 'Deployments to Pull Server' which contains a table for 'Compilation jobs':

STATUS	CREATED	LAST UPDATED
No compilation jobs found.		

5. Once compiled, verify that the file has a status of completed.

The screenshot shows the LabConfig configuration interface. At the top, there are buttons for Compile, Export, and Delete. Below that, a section titled 'Essentials' displays various configuration details:

Resource group	az-auto-rg	Account	az-auto-ac-1
Location	westeurope	Subscription name	Pay-As-You-Go
Subscription ID	974e6e39-73eb-48b0-9226-dae31425c367	Status	Published
Last published	1/11/2019 1:00 PM	Configuration source	View configuration source

Below this is a section titled 'Deployments to Pull Server' containing a table for 'Compilation jobs':

STATUS	CREATED	LAST UPDATED
✓ Completed	1/11/2019 1:05 PM	1/11/2019 1:05 PM

✓ Note: If you prefer, you can also use the **PowerShell Start-AzAutomationDscCompilationJob** cmdlet. More information about this method is available at [Compiling a DSC Configuration with Windows PowerShell⁴²](#).

Demonstration-Onboarding machines for management

After your configuration is in place, you will select the Azure VMs or on-premises VMs that you want to onboard.

✓ Note: For more information on onboarding on-premises VMs, review the [Physical/virtual Windows machines on-premises, or in a cloud other than Azure/AWS⁴³](#) webpage.

You can onboard a VM and enable DSC in several different ways. Here we will cover onboarding through an Azure Automation account.

Onboard VMs to configure

When onboarding VMs using this method, you will need to deploy your VMs to Azure prior to starting:

1. In the left pane of the Automation account, select **State configuration (DSC)**.
2. Select the **Nodes** tab, and then select **+ Add** to open the Virtual Machines pane.
3. Find the VM you would like to enable. (You can use the search field and filter options to find a specific VM, if necessary.)

⁴² <https://azure.microsoft.com/en-us/documentation/articles/automation-dsc-compile/#compiling-a-dsc-configuration-with-windows-powershell>

⁴³ <https://docs.microsoft.com/en-us/azure/automation/automation-dsc-onboarding#physicalvirtual-windows-machines-on-premises-or-in-a-cloud-other-than-azureaws>

4. Select the VM, and then select **Connect**.

The screenshot shows two windows side-by-side. On the left is the 'Virtual Machines' blade with a search bar, filter dropdowns (Pay-As-You-Go, 2 selected, West Europe), and a table listing two VMs: 'SimpleWinVM' and 'SimpleWinVM'. On the right is a detailed view for 'SimpleWinVM' showing its status: POWER STATE (VM running), OS (Windows), and STATUS (Not connected). A red box highlights the 'Connect' button.

5. In the resultant Registration pane, configure the following settings, and then select **OK**.

The screenshot shows the 'Registration' configuration pane. It includes fields for 'Registration key' (Primary key selected), 'Node configuration name' (LabConfig.WebServer), 'Refresh Frequency' (30), 'Configuration Mode Frequency' (15), 'Configuration Mode' (ApplyAndMonitor), and checkboxes for 'Allow Module Override' and 'Reboot Node if Needed'. Below the form is a table of properties and descriptions.

Property	Description
Registration key	Primary or secondary, for registering the node with a pull service.
Node configuration name	The name of the node configuration that the VM should be configured to pull for Automation DSC
Refresh Frequency	The time interval, in minutes, at which the LCM checks a pull service to get updated configurations. This value is ignored if the LCM is not configured in pull mode. The default value is 30.
Configuration Mode Frequency	How often, in minutes, the current configuration is checked and applied. This property is ignored if the **ConfigurationMode** property is set to **ApplyOnly**. The default value is 15.

Configuration mode	Specifies how the LCM gets configurations. Possible values are `**ApplyOnly**`, `**ApplyAndMonitor**`, and `**ApplyAndAutoCorrect**`.
Allow Module Override	Controls whether new configurations downloaded from the Azure Automation DSC pull server are allowed to overwrite the old modules already on the target server.
Reboot Node if Needed	Set this to `**$true**` to automatically reboot the node after a configuration that requires reboot is applied. Otherwise, you will have to manually reboot the node for any configuration that requires it. The default value is `**$false**`.
Action after Reboot	Specifies what happens after a reboot during the application of a configuration. The possible values are `**ContinueConfiguration**` and `**StopConfiguration**`.

The service will then connect to the Azure VMs and apply the configuration.

6. Return to the State configuration (DSC) pane and verify that after applying the configuration, the status now displays as Compliant.

The screenshot shows the 'Nodes' tab in the Azure Automation DSC blade. At the top, there are buttons for 'Add', 'Refresh', 'Reset filters', and 'Enable Log Search'. Below the tabs, a summary chart shows 1 node overall, with 0 Failed, 0 Pending, 0 Not compliant, 0 In progress, 0 Unresponsive, and 1 Compliant. There are also links to 'Learn more' and 'Add non-Azure machine'. Below the chart are four filter dropdowns: 'Nodes' (1 selected), 'Status' (6 selected), 'Node configuration' (All), and 'VM DSC extension version > 2.70' (2 selected). The main table lists one node: 'SimpleWinVM' with 'Compliant' status, 'LabConfig.WebServer' configuration, last seen at '1/11/2019 2:17 PM', and version '2.77.0.0'. There is a '...' button next to the version column.

Node	Status	Node Configuration	Last Seen	Version
SimpleWinVM	Compliant	LabConfig.WebServer	1/11/2019 2:17 PM	2.77.0.0

Each time that Azure Automation DSC performs a consistency check on a managed node, the node sends a status report back to the pull server. You can review these reports on that node's blade. Access this by double-clicking or pressing the spacebar and then Enter on the node.

SimpleWinVM

Assign node configuration Unregister

Essentials

Resource group	IP address
az-auto-rg	10.0.0.4
Id	Account
b6bc9bd8-15a8-11e9-a80e-000d3a4664d5	az-auto-ac-1
Last seen time	Virtual machine
1/11/2019 2:27 PM	SimpleWinVM
Configuration	Node configuration
LabConfig	LabConfig.WebServer
Registration time	Status
1/11/2019 2:09 PM	Compliant

Reports

TYPE	STATUS	REPORT TIME
Consistency	✓ Compliant	1/11/2019 2:27 PM
Consistency	✓ Compliant	1/11/2019 2:27 PM
Initial	✓ Compliant	1/11/2019 2:07 PM
Consistency	✓ Compliant	1/11/2019 2:12 PM

✓ Note: You can also unregister the node and assign a different configuration to nodes.

For more details about onboarding VMs, see also:

- [Enable Azure Automation State Configuration⁴⁴](#)
- [Configuring the Local Configuration Manager⁴⁵](#)

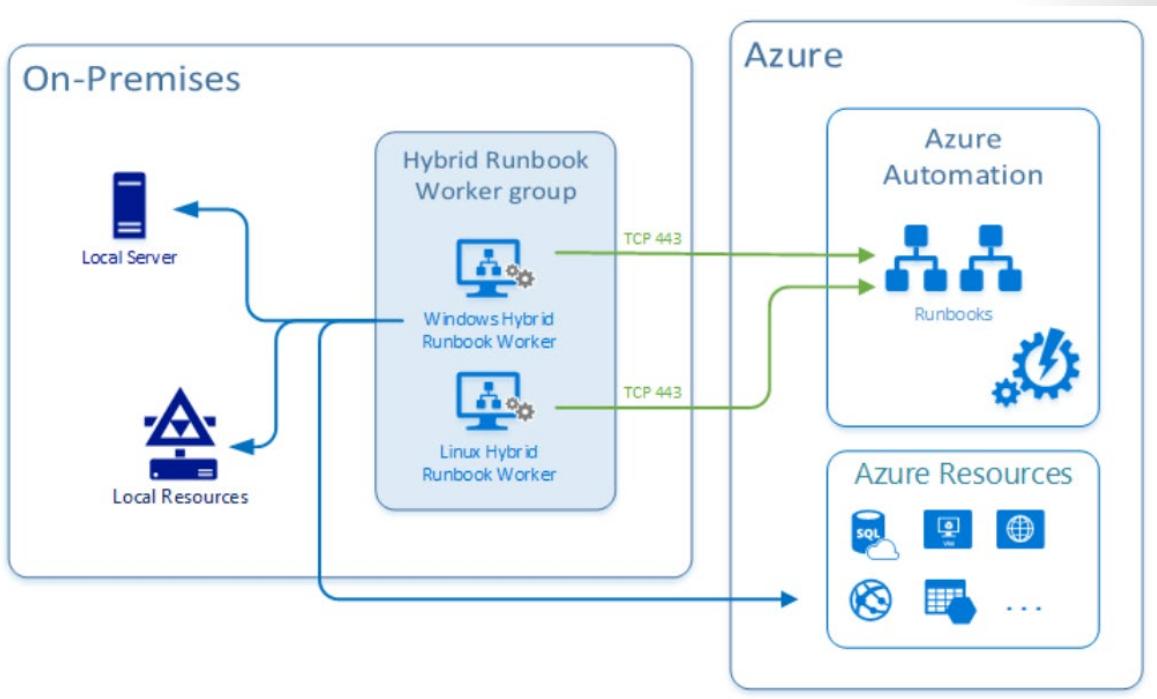
Hybrid Management

The Hybrid Runbook Worker feature of Azure Automation allows you to run runbooks that manage local resources in your private datacenter, on machines located in your datacenter. Azure Automation stores and manages the runbooks, and then delivers them to one or more on-premises machines.

The Hybrid Runbook Worker functionality is presented in the following graphic:

⁴⁴ <https://docs.microsoft.com/en-us/azure/automation/automation-dsc-onboarding>

⁴⁵ <https://docs.microsoft.com/en-us/powershell/scripting/dsc/managing-nodes/metaconfig?view=powershell-7>



Hybrid Runbook Worker workflow and characteristics

The following list are characteristics of the Hybrid Runbook Worker workflow:

- You can designate one or more computers in your datacenter to act as a Hybrid Runbook Worker, and then run runbooks from Azure Automation.
- Each Hybrid Runbook Worker is a member of a Hybrid Runbook Worker group, which you specify when you install the agent.
- A group can include a single agent, but you can install multiple agents in a group for high availability.
- There are no inbound firewall requirements to support Hybrid Runbook Workers, only Transmission Control Protocol (TCP) 443 is required for outbound internet access.
- The agent on the local computer initiates all communication with Azure Automation in the cloud.
- When a runbook is started, Azure Automation creates an instruction that is retrieved by the agent. The agent then pulls down the runbook and any parameters prior to running it.

To manage configuring your on-premises servers that support the Hybrid Runbook Worker role with DSC, you must add them as DSC nodes. For further information about onboarding them for management with DSC, see **Onboarding machines for management by Azure Automation State Configuration⁴⁶**.

For more information on installing and removing Hybrid Runbook Workers and groups, see:

- Automate resources in your datacenter or cloud by using Hybrid Runbook Worker⁴⁷**
- Hybrid Management in Azure Automation⁴⁸**

46 <https://docs.microsoft.com/en-us/azure/automation/automation-dsc-onboarding>

47 <https://docs.microsoft.com/en-us/azure/automation/automation-hybrid-runbook-worker#installing-hybrid-runbook-worker>

48 <https://azure.microsoft.com/en-us/blog/hybrid-management-in-azure-automation/>

DSC and Linux Automation on Azure

The following Linux operating system versions are currently supported by both PowerShell DSC and Azure Automation DSC:

- CentOS 5, 6, and 7 (x86/x64)
- Debian GNU/Linux 6, 7 and 8 (x86/x64)
- Oracle Linux 5, 6 and 7 (x86/x64)
- Red Hat Enterprise Linux Server 5, 6 and 7 (x86/x64)
- SUSE Linux Enterprise Server 10, 11 and 12 (x86/x64)
- Ubuntu Server 12.04 LTS, 14.04 LTS and 16.04 LTS (x86/x64)

More specific and up to date details are available at **Get started with Desired State Configuration (DSC) for Linux⁴⁹**.

⁴⁹ <https://docs.microsoft.com/en-us/powershell/scripting/dsc/getting-started/lnxGettingStarted?view=powershell-6>

Azure Automation with DevOps

What is Azure Automation

Manually executing environment provisioning and configuration management is both laborious and error-prone. Microsoft Azure DevOps advocates automation to reduce the probability of errors introduced through manual execution. Automation also delivers the added advantage of completing the work more quickly without relying on subject experts.

Microsoft Azure is built to support automation from the ground up. *Azure Automation* is an Azure service that provides a way for users to automate the manual, long-running, error-prone, and frequently repeated tasks that are commonly performed in a cloud and enterprise environment. Azure Automation saves time and increases the reliability of regular administrative tasks. You can even schedule the tasks to be performed automatically at regular intervals. You can automate processes using runbooks, or automate configuration management by using Desired State Configuration (DSC). For more information about Azure Automation, review **An introduction to Azure Automation**⁵⁰.



Azure Automation is not the only way to automate within Azure. You can also use open-source tools to perform some of these operations. However, the integration hooks available to Azure Automation remove much of the integration complexity that you would have to manage if you performed these operations manually.

Some Azure Automation capabilities are:

- Process automation. Azure Automation provides you with the ability to automate frequent, time-consuming, and error-prone cloud management tasks.
- Azure Automation State Configuration. This is an Azure service that allows you to write, manage, and compile Windows PowerShell DSC configurations, import DSC Resources, and assign configurations to target nodes, all in the cloud. For more information, visit **Azure Automation State Configuration Overview**⁵¹.
- Update management. Manage operating system updates for Windows and Linux computers in Azure, in on-premises environments, or in other cloud providers. Get update compliance visibility across Azure, on-premises, and for other cloud services. You can create scheduled deployments to orchestrate update installations within a defined maintenance window. For more information, visit **Update Management solution in Azure**⁵².
- Start and stop virtual machines (VMs). Azure Automation provides an integrated Start/Stop VM-related resource that enables you to start and stop VMs on user-defined schedules. It also provides insights through **Azure Log Analytics**, and can send emails by using action groups. For more information, go to **Start/Stop VMs during off-hours solution in Azure Automation**⁵³.

⁵⁰ <https://azure.microsoft.com/en-us/documentation/articles/automation-intro/>

⁵¹ <https://docs.microsoft.com/en-us/azure/automation/automation-dsc-overview>

⁵² <https://docs.microsoft.com/en-us/azure/automation/automation-update-management>

⁵³ <https://docs.microsoft.com/en-us/azure/automation/automation-solution-vm-management>

- Integration with GitHub, Azure DevOps, Git, or Team Foundation Version Control (TFVC) repositories. For more information, go to **Source control integration in Azure Automation**⁵⁴
- Automate Amazon Web Services (AWS) Resources. Automate common tasks with resources in AWS using Automation runbooks in Azure. For more information, go to **Authenticate Runbooks with Amazon Web Services**⁵⁵.
- Manage Shared resources. Azure Automation consists of a set of shared resources (such as *connections*, *credentials*, *modules*, *schedules*, and *variables*) that make it easier to automate and configure your environments at scale.
- Run backups. Azure Automation allows you to run regular backups of non-database systems, such as backing up Azure Blob Storage at certain intervals.

Azure Automation works across hybrid cloud environments in addition to Windows and Linux operating systems.

Automation Accounts

To start using the Microsoft Azure Automation service, you must first create an **Automation account**⁵⁶ from within the Azure portal. Steps to create an Azure Automation account are available on the **Create an Azure Automation account**⁵⁷ page.

Automation accounts are similar to Azure Storage accounts in that they serve as a container to store automation artifacts. These artifacts could be a container for all your runbooks, runbook executions (*jobs*), and the assets on which your runbooks depend.

An Automation account gives you access to managing all Azure resources via an API. To safeguard this, the Automation account creation requires subscription-owner access.

⁵⁴ <https://docs.microsoft.com/en-us/azure/automation/source-control-integration>

⁵⁵ <https://docs.microsoft.com/en-us/azure/automation/automation-config-aws-account>

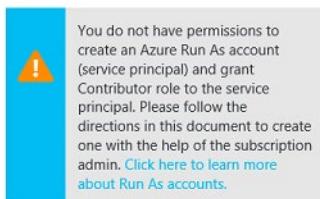
⁵⁶ <https://azure.microsoft.com/en-us/documentation/articles/automation-security-overview/>

⁵⁷ <https://docs.microsoft.com/en-us/azure/automation/automation-quickstart-create-account>

The screenshot shows the 'Add Automation Account' wizard in the Azure portal. The 'Name' field is filled with 'azautoac01'. The 'Subscription' dropdown shows 'Pay-As-You-Go'. The 'Resource group' dropdown shows '(New) az-auto-rg'. The 'Location' dropdown shows 'Japan East'. A red box highlights the 'Create Azure Run As account' section, which has a 'Yes' button. Below it is an info callout: 'The Run As account feature will create a Run As account and a Classic Run As account. Click here to learn more about Run As accounts.' A blue info icon is next to a link: 'Learn more about Automation pricing.' At the bottom is a 'Create' button.

You must be a subscription owner to create the Run As accounts that the service creates.

If you do not have the proper subscription privileges, you will see the following warning:



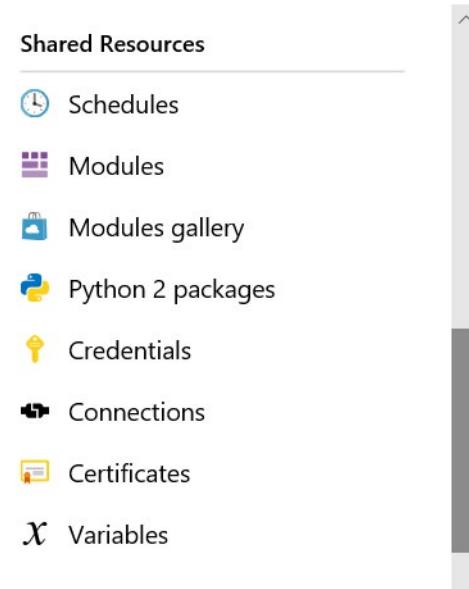
To use Azure Automation, you will need at least one Azure Automation account. However, as a best practice you should create multiple automation accounts to segregate and limit the scope of access, and minimize any risk to your organization. For example, you might use one account for development, another for production, and another for your on-premises environment. You can have up to 30 Automation accounts.

Automation Shared resources

Azure Automation contains shared resources that are globally available to be associated with or used in a runbook. There are currently eight shared resources categories:

- **Schedules:** Allows you to define a one-off or recurring schedule.

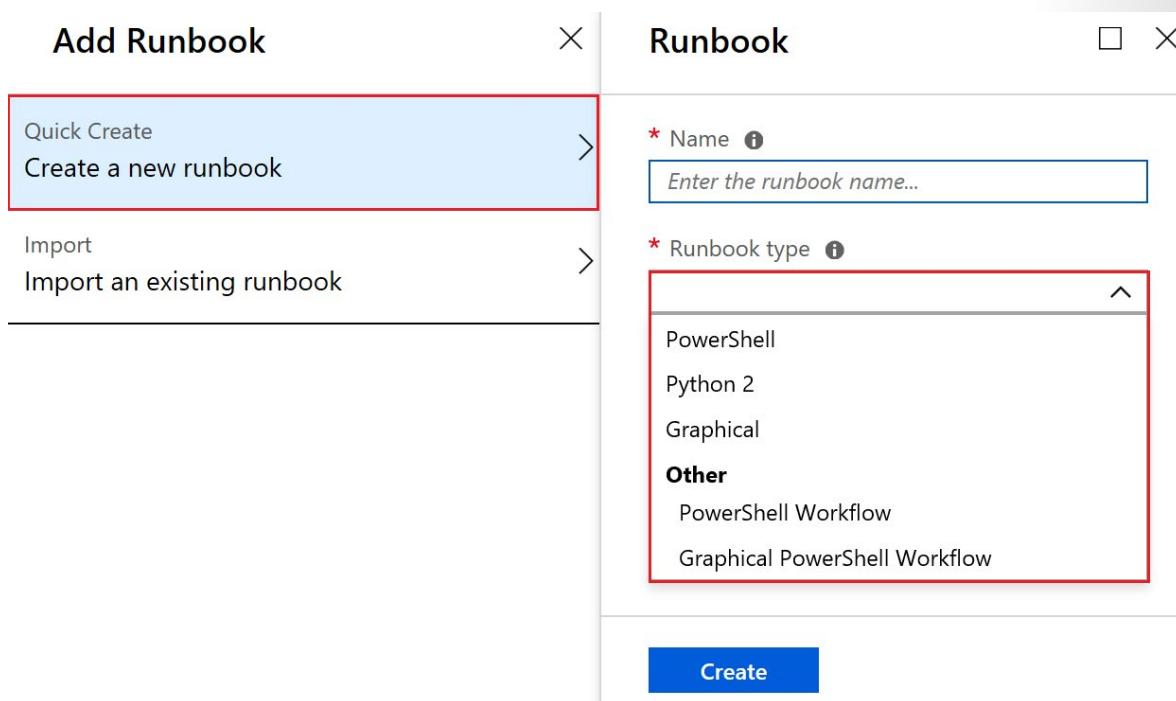
- **Modules:** Contains Azure PowerShell modules.
- **Modules gallery:** Allows you to identify and import PowerShell modules into your Azure automation account.
- **Python 2 packages:** Allows you to import a Python 2 package by uploading .whl or tar.gz packages.
- **Credentials:** Allows you to create username and password credentials.
- **Connections:** Allows you to specify Azure, Azure classic certificate, or Azure Service principal connections.
- **Certificates:** Allows you to upload certificates in .cer or pfx format.
- **Variables:** Allows you to define encrypted or unencrypted variables of types such as *String, Boolean, DateTime, Integer*, or of no specific type.



As a best practice, always try to create global assets so they can be used across your runbooks. This will save time and reduce the number of manual edits within individual runbooks.

What is a Runbook

Runbooks serve as repositories for your custom scripts and workflows. They also typically reference Automation shared resources such as credentials, variables, connections, and certificates. Runbooks can also contain other runbooks, thereby allowing you to build more complex workflows. You can invoke and run runbooks either on demand, or according to a schedule by leveraging Automation Schedule assets.



Creating runbooks

When creating runbooks, you have two options. You can either:

- Create your own runbook and import it. For more information about creating or importing a runbook in Azure Automation, go to [Manage runbooks in Azure Automation⁵⁸](#).
- Modify runbooks from the runbook gallery. This provides a rich ecosystem of runbooks that are available for your requirements. Visit [Runbook and module galleries for Azure Automation⁵⁹](#) for more information.

There is also a vibrant open-source community that creates runbooks you can apply directly to your use cases.

You can choose from different runbook types based on your requirements and Windows PowerShell experience. If you prefer to work directly with Windows PowerShell code, you can use either a PowerShell runbook, or a PowerShell Workflow runbook. Using either of these you can edit offline or with the textual editor in the Azure portal. If you prefer to edit a runbook without being exposed to the underlying code, you can create a graphical runbook by using the graphical editor in the Azure portal.

Graphical runbooks

Graphical runbooks and Graphical PowerShell Workflow runbooks are created and edited with the graphical editor in the Azure portal. You can export them to a file and then import them into another automation account, but you cannot create or edit them with another tool.

⁵⁸ <https://docs.microsoft.com/en-us/azure/automation/automation-creating-importing-runbook>

⁵⁹ <https://docs.microsoft.com/en-us/azure/automation/automation-runbook-gallery>

PowerShell runbooks

PowerShell runbooks are based on Windows PowerShell. You edit the runbook code directly, using the text editor in the Azure portal. You can also use any offline text editor and then import the runbook into Azure Automation. PowerShell runbooks do not use parallel processing.

PowerShell Workflow runbooks

PowerShell Workflow runbooks are text runbooks based on Windows PowerShell Workflow. You directly edit the runbook code using the text editor in the Azure portal. You can also use any offline text editor and import the runbook into Azure Automation.

PowerShell Workflow runbooks use parallel processing to allow for simultaneous completion of multiple tasks. Workflow runbooks take longer to start than PowerShell runbooks because they must be compiled prior to running.

Python runbooks

Python runbooks compile under Python 2. You can directly edit the code of the runbook using the text editor in the Azure portal, or you can use any offline text editor and import the runbook into Azure Automation. You can also utilize Python libraries. However, only Python 2 is supported at this time. To utilize third-party libraries, you must first import the package into the Automation Account.

- ✓ Note: You can't convert runbooks from graphical to textual type, or vice versa.

For more information on the different types of runbooks, visit [Azure Automation runbook types⁶⁰](#).

Automating Processes with Runbooks

A **runbook** is a set of tasks that perform some automated process in Azure Automation. It might be a simple process such as starting a VM and creating a log entry. Or you might have a complex runbook that combines other smaller runbooks to perform a complex process across multiple resources, clouds, or on-premises environments.

Traditionally, runbooks were a list of instructions that needed to be followed to complete an activity. Any activity that became a recurring task would usually end up with a runbook to reduce dependencies on specific individuals.

An example use case for a runbook would be to replace an existing manual process for truncating a SQL Azure database when it's approaching maximum size. That current manual process includes multiple steps, such as connecting to the server, connecting to the database, getting the current size of database, checking if threshold has been exceeded and then truncating it, and finally notifying the user.

Instead of performing each of these steps manually, you could create a runbook that would perform these tasks as a single process. You would start the runbook, provide the required information (such as the SQL server name, database name, and recipient email), and then wait while the process completes by itself.

What can runbooks automate?

Runbooks in Azure Automation can do anything that PowerShell can do because they are based on Windows PowerShell or Windows PowerShell Workflow and Python2. If an application or service has an

⁶⁰ <https://azure.microsoft.com/en-us/documentation/articles/automation-runbook-types>

API, then a runbook can work with it. Examples of some tasks that Azure Automation runbooks are capable of are:

- Automating resources in your on-premises datacenter or cloud using **Hybrid Runbook Worker**⁶¹.
- **Start/Stop Azure virtual machines during off-hours solution in Azure Automation**⁶² and autostop VMs based on low CPU usage, to optimize VM costs.
- **Automate provisioning of a virtual machine in Amazon Web Services (AWS)**⁶³
- **Forward job status and job streams from Automation to Log Analytics**⁶⁴. Send your Automation logs to Log Analytics for analysis and have alerts such as email configured to notify you of any failures or other relevant outcomes.

✓ Note: Before you can run a runbook, you must first publish it so it's available to run.

Runbook Gallery

Azure Automation runbooks are provided to help eliminate the time it takes to build custom solutions. These runbooks have already been built by Microsoft and the Microsoft community, and you can use them with or without modification. You can import runbooks from the runbook gallery at the Microsoft Script Center, **Script resources for IT professionals**⁶⁵ webpage.

✓ Note: A new Azure PowerShell module was released in December 2018, called the **Az** PowerShell module. This replaces the existing **AzureRM** PowerShell module, and is now the intended PowerShell module for interacting with Azure. This new **Az** module is now supported in Azure Automation. For more general details on the new Az PowerShell module, go to **Introducing the new Azure PowerShell Az module**⁶⁶.

Choosing items from the runbook gallery

In the Azure portal, you can import directly from the runbook gallery using the following high-level steps:

1. Open your Automation account, and then select **Process Automation > Runbooks**.
2. In the runbooks pane, select **Browse gallery**.
3. From the runbook gallery, locate the runbook gallery item that you want, select it, and then select **Import**.

When browsing through the runbooks in the script center library, you can review the code or a visualization of the code. You also can review information such as the source project along with a detailed description, ratings, and questions and answers. For more information, refer to **Script resources for IT professionals**⁶⁷.

61 <https://docs.microsoft.com/en-us/azure/automation/automation-hybrid-runbook-worker>

62 <https://docs.microsoft.com/en-us/azure/automation/automation-solution-vm-management>

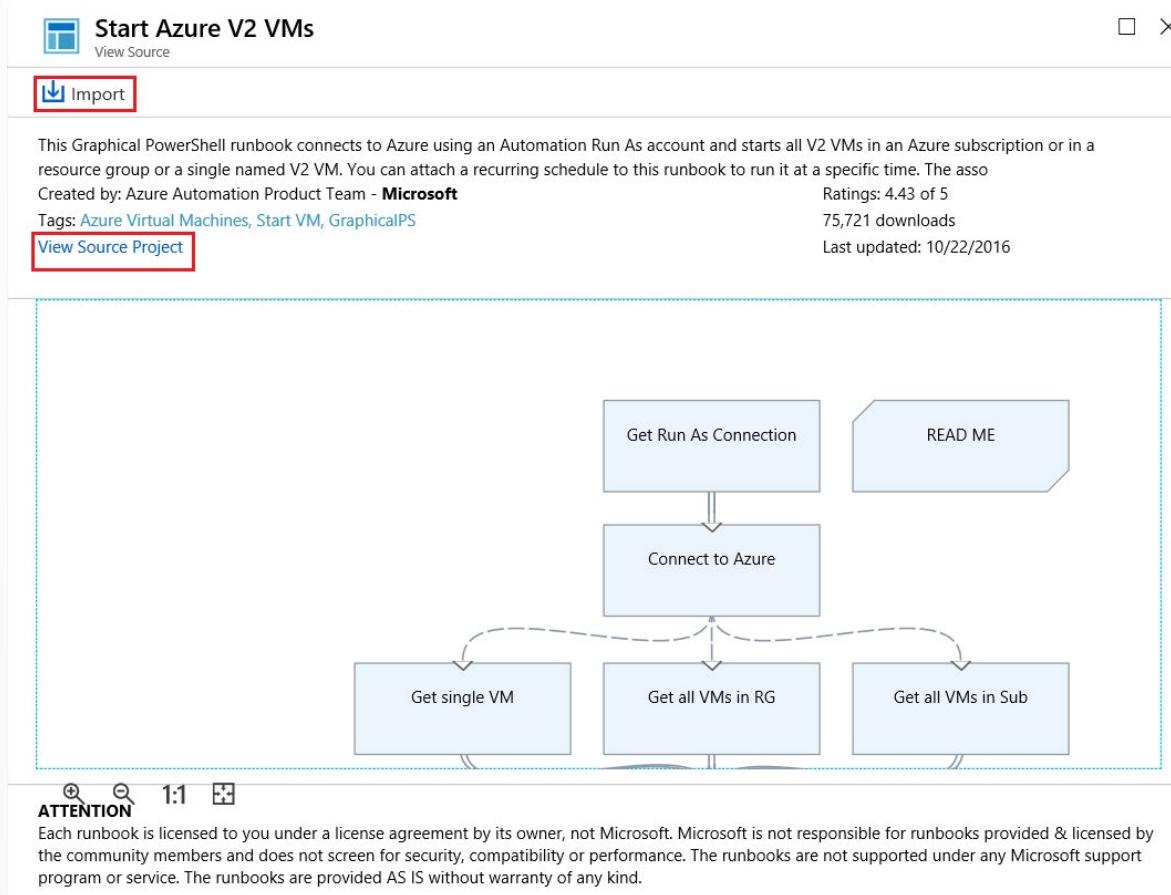
63 <https://docs.microsoft.com/en-us/azure/automation/automation-scenario-aws-deployment>

64 <https://docs.microsoft.com/en-us/azure/automation/automation-manage-send-joblogs-log-analytics#set-up-integration-with-log-analytics>

65 [https://gallery.technet.microsoft.com/scriptcenter/site/search?f\[0\].Type=RootCategory&f\[0\].Value=WindowsAzure&f\[1\].Type=SubCategory&f\[1\].Value=WindowsAzure_automation&f\[1\].Text=Automation](https://gallery.technet.microsoft.com/scriptcenter/site/search?f[0].Type=RootCategory&f[0].Value=WindowsAzure&f[1].Type=SubCategory&f[1].Value=WindowsAzure_automation&f[1].Text=Automation)

66 <https://docs.microsoft.com/en-us/powershell/azure/new-azurerm-module-az?view=azps-2.7.0>

67 <https://gallery.technet.microsoft.com/scriptcenter>



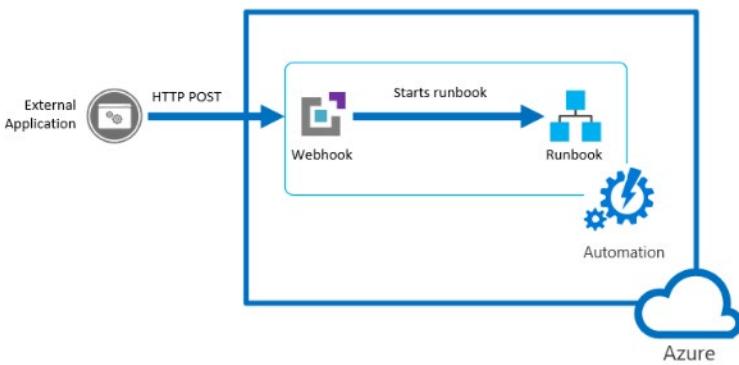
- ✓ Note: Python runbooks are also available from the script center gallery. To find them, filter by language and select **Python**.
- ✓ Note: You cannot use PowerShell to import directly from the runbook gallery.

Webhooks

You can automate the process of starting a runbook either by scheduling it, or by using a webhook.

A **webhook** allows you to start a particular runbook in Azure Automation through a single HTTPS request. This allows external services such as Azure DevOps, GitHub, or custom applications to start runbooks without implementing more complex solutions using the Azure Automation API
(More information about webhooks is available at [Starting an Azure Automation runbook with a webhook⁶⁸](#).)

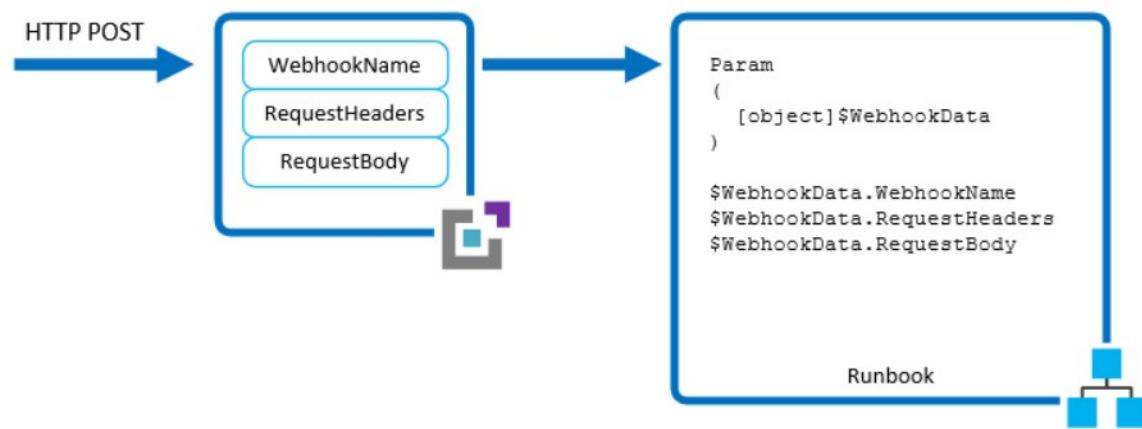
⁶⁸ <https://docs.microsoft.com/en-us/azure/automation/automation-webhooks>



Create a webhook

You create a webhook linked to a runbook using the following steps:

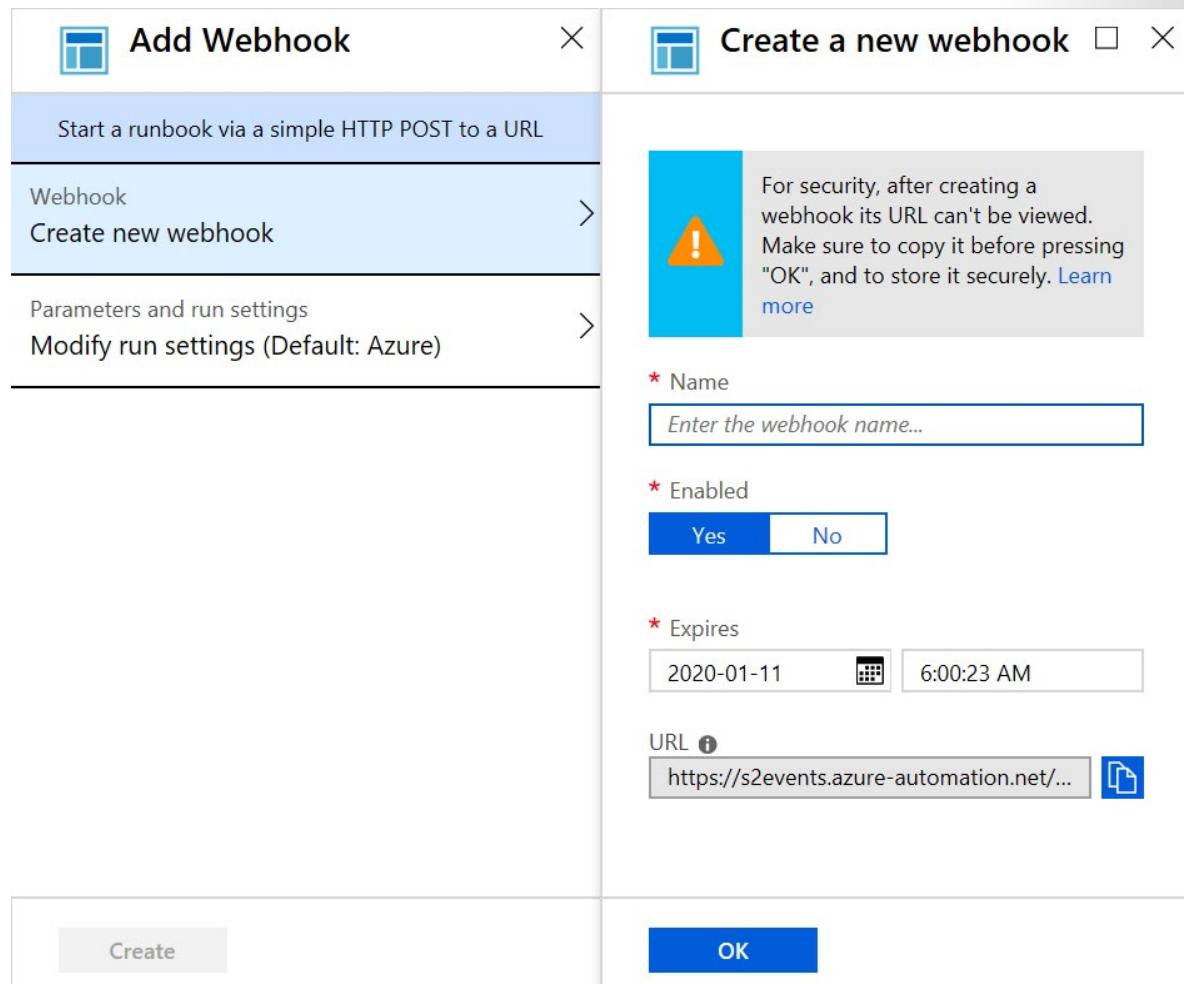
1. In the Azure portal, open the runbook that you want to create the webhook for.
2. In the runbook pane, under Resources, select **Webhooks**, and then select **+ Add webhook**.
3. Select **Create new webhook**.
4. In the **Create new webhook** dialog, there are several values you need to configure. After you configure them, select **Create**:
 - **Name**. Specify any name you want for a webhook, because the name is not exposed to the client; it's only used for you to identify the runbook in Azure Automation.
 - **Enabled**. A webhook is enabled by default when it is created. If you set it to Disabled, then no client is able to use it.
 - **Expires**. Each webhook has an expiration date at which time it can no longer be used. You can continue to modify the date after creating the webhook providing the webhook is not expired
 - **URL**. The URL of the webhook is the unique address that a client calls with an HTTP POST, to start the runbook linked to the webhook. It is automatically generated when you create the webhook, and you cannot specify a custom URL. The URL contains a security token that allows the runbook to be invoked by a third-party system with no further authentication. For this reason, treat it like a password. For security reasons, you can only view the URL in the Azure portal at the time the webhook is created. Make note of the URL in a secure location for future use.



✓ Note: Make sure you make a copy of the webhook URL when creating it, and then store it in a safe place. After you create the webhook, you cannot retrieve the URL again.

4. Select the **Parameters run settings (Default : Azure)** option. This option has the following characteristics, which allows you to complete the following actions:

- If the runbook has mandatory parameters, you will need to provide these mandatory parameters during creation. You are not able to create the webhook unless values are provided.
- If there are no mandatory parameters in the runbook, there is no configuration required here.
- The webhook must include values for any mandatory parameters of the runbook, but could also include values for optional parameters.
- When a client starts a runbook using a webhook, it cannot override the parameter values defined in the webhook.
- To receive data from the client, the runbook can accept a single parameter called `$WebhookData` of type `[object]` that contains data that the client includes in the POST request.
- There is no required webhook configuration to support the `$WebhookData` parameter.



- When finished, select **Create**.

Using a Webhook

To use a webhook after it has been created, your client application must issue an HTTP POST with the URL for the webhook.

- The syntax of the webhook is in the following format:

```
http://< Webhook Server >/token?=< Token Value >
```

- The client receives one of the following return codes from the POST request.

Code	Test	Description
202	Accepted	the request was accepted and the runbook is successfully queued
400	Bad request	The request was not accepted because the runbook has expired, been disabled, or the token in the URL is invalid

Code	Test	Description
404	Not found	The request was not accepted because the webhook, runbook, or account was not found
500	Internal Server Error	

- If successful, the webhook response contains the job ID in JSON format as follows:

```
{"JobIds": ["< JobId >"]}
```

The response will contain a single job ID, but the JSON format allows for potential future enhancements.

- You cannot determine when the runbook job completes or determine its completion status from the webhook. You can only determine this information using the job ID with another method such as PowerShell or the Azure Automation API.

More details are available on the [Starting an Azure Automation runbook with a webhook](#)⁶⁹ page.

Source Control Integration

Azure Automation supports source control integration that enables you to keep your runbooks in your Automation account up to date with your scripts in your GitHub or Azure DevOps source control repository.

Source control allows you to more easily collaborate with your team, track changes, and roll back to earlier versions of your runbooks. For example, source control allows you to sync different branches in source control to your development, test, or production Automation accounts. This makes it easier to promote code you've tested in your development environment to your production Automation account.

Azure Automation supports three types of source control:

- GitHub
- Azure DevOps (Git)
- Azure DevOps (TFVC)

Source control allows you to push code from Azure Automation to source control, or pull your runbooks from source control to Azure Automation. Source control sync jobs run under the user's Automation Account and are billed at the same rate as other Automation jobs.

Integrate source control with Azure Automation

You integrate source control with Azure Automation using the following steps:

1. In the Azure Portal, access your Automation account.
2. Under Account Settings, select **Source control**, and then select **+ Add**.
3. In the **Source Control Summary** blade, select **GitHub** as source control type, and then select **Authenticate**.
Note: You will require a GitHub account to complete the next step.
4. When the browser page opens prompting you to authenticate to <https://www.github.com>, select **Authorize azureautomation** and enter your GitHub account password.

⁶⁹ <https://docs.microsoft.com/en-us/azure/automation/automation-webhooks#details-of-a-webhook>

If successful you should receive an email notification from GitHub stating that *A third-party OAuth Application (Automation Source Control) with repo scope was recently authorized to access your account.*

- After authentication completes, fill in the details based on the following table, and then select **Save**.

Property	Description
Name	Friendly name
Source control type	GitHub, Azure DevOps Git or Azure Devops TFVC
Repository	The name of the repository or project
Branch	The branch from which to pull the source files. Branch targeting is not available for the TFVC source control type
Folder Path	The folder that contains the runbooks to sync.
Auto sync	Turns on or off automatic sync when a commit is made in the source control repository
Publish Runbook	If set to On , after runbooks are synced from source control they will be automatically published
Description	A text field to provide additional details

- If you set **Autosync** to **Yes**, a full sync will start. If you set **Auto sync** to **No**, open the **Source Control Summary** blade again by selecting your repository in Azure Automation, and then selecting **Start Sync**.

Source Control Summary

X

* Source control name
 ✓

* Source control type
 ▼

Authorization successful.

* Repository
 ▼

* Branch
 ▼

* Folder path
 ✓

Auto Sync i
 On Off

Publish Runbook i
 On Off

Description

7. Verify that your source control is listed in the **Azure Automation Source control** page for you to use.

The screenshot shows the 'Source control' blade in the Azure portal for an automation account named 'az-auto-ac1'. The left sidebar includes options like Event grid, Start/Stop VM, Account Settings, Properties, Keys, Pricing, and Source control (which is selected). The main area displays a table of sync jobs:

STATUS	SOURCE CONTROL	SYNC TYPE	CREATION TIME	END TIME
✓ Completed	github	Full	1/17/2019, 3:15 AM	1/17/2019, 3:16 AM
✓ Completed	github	Full	1/17/2019, 3:15 AM	1/17/2019, 3:16 AM

Update Management

You can use the Update Management solution in Azure Automation to manage operating system updates:

- For computers running the Windows and Linux operating systems.
- For computers deployed in Azure.
- In on-premises environments.
- In other cloud providers.

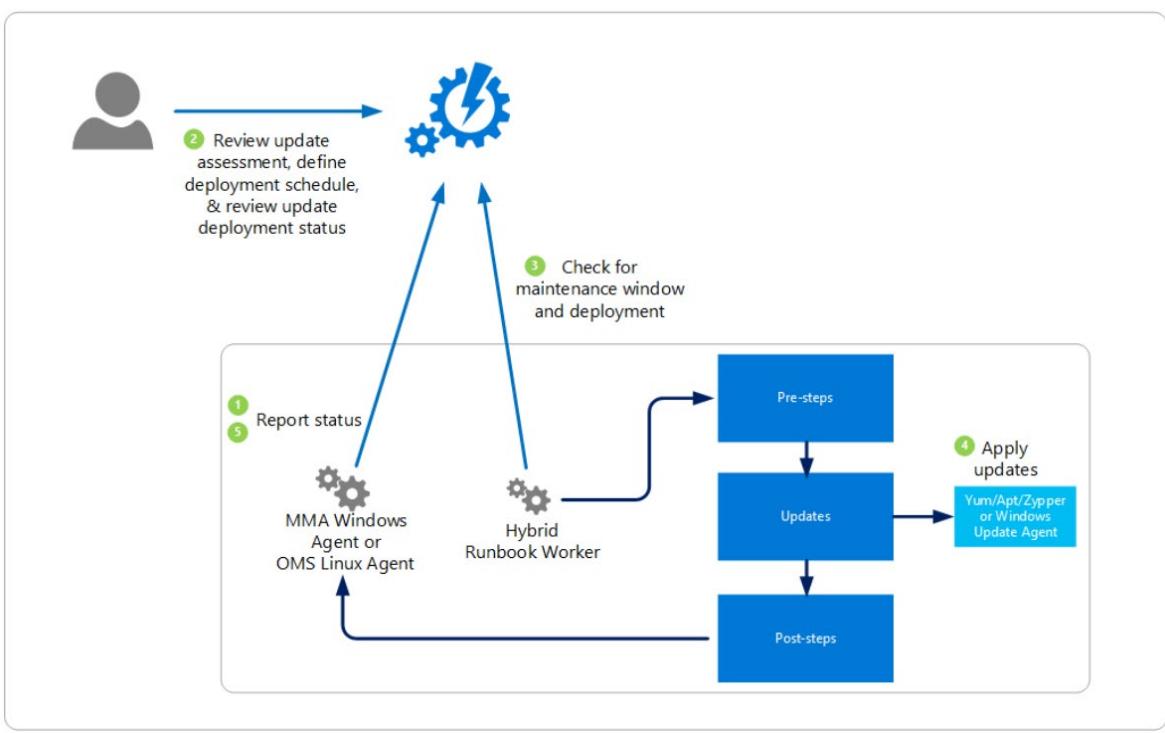
You can also use Update Management to quickly assess the status of available updates on all agent computers, and manage the installation process for required server updates.

Update Management solution overview

Computers that are managed by Update Management use the following configurations to perform assessments and update deployments:

- Microsoft Monitoring Agent (MMA) for Windows or Operations Management Suite (OMS) Linux Agent
- PowerShell DSC for Linux
- Automation Hybrid Runbook Worker
- Microsoft Update or Windows Server Update Services (WSUS) for computers running Windows operating systems i.e. the Windows Update Agent is a windows based agent that works with WSUS, or yum/apt/zypper which are package management services available on Linux.

The following diagram is a conceptual view of the behavior and data flow of how the Update Management solution assesses and applies security updates to all connected computers running the Windows Server and Linux operating systems in a workspace.



You can also use Update Management to natively onboard machines in multiple subscriptions in the same tenant.

PowerShell Workflows

IT pros often automate management tasks for their multi-device environments by running sequences of long-running tasks or workflows. These tasks can affect multiple managed computers or devices at the same time. PowerShell Workflow lets IT pros and developers leverage the benefits of Windows Workflow Foundation with the automation capabilities and ease of using Windows PowerShell. Refer to [A Developer's Introduction to Windows Workflow Foundation \(WF\) in .NET 4⁷⁰](#) for more information.

Windows PowerShell Workflow functionality was introduced in Windows Server 2012 and Windows 8, and is part of Windows PowerShell 3.0 and later. Windows PowerShell Workflow helps automate distribution, orchestration, and completion of multi-device tasks, freeing users and administrators to focus on higher-level tasks.

Activities

An **activity** is a specific task that you want a workflow to perform. Just as a script is composed of one or more commands, a workflow is composed of one or more activities that are carried out in sequence. You can also use a script as a single command in another script, and use a workflow as an activity within another workflow.

⁷⁰ [https://docs.microsoft.com/en-us/previous-versions/dotnet/articles/ee342461\(v=msdn.10\)](https://docs.microsoft.com/en-us/previous-versions/dotnet/articles/ee342461(v=msdn.10))

Workflow characteristics

A workflow can:

- Be long-running.
- Be repeated over and over.
- Run tasks in parallel.
- Be interrupted—can be stopped and restarted, suspended and resumed.
- Continue after an unexpected interruption, such as a network outage or computer/server restart.

Workflow benefits

A workflow offers many benefits, including:

- Windows PowerShell scripting syntax. Is built on PowerShell.
- Multidevice management. Simultaneously apply workflow tasks to hundreds of managed nodes.
- Single task runs multiple scripts and commands. Combine related scripts and commands into a single task. Then run the single task on multiple computers. The activity status and progress within the workflow are visible at any time.
- Automated failure recovery.
 - Workflows survive both planned and unplanned interruptions, such as computer restarts.
 - You can suspend a workflow operation, then restart or resume the workflow from the point at which it was suspended.
 - You can author checkpoints as part of your workflow, so that you can resume the workflow from the last persisted task (or checkpoint) instead of restarting the workflow from the beginning.
- Connection and activity retries. You can retry connections to managed nodes if network-connection failures occur. Workflow authors can also specify activities that must run again if the activity cannot be completed on one or more managed nodes (for example, if a target computer was offline while the activity was running).
- Connect and disconnect from workflows. Users can connect and disconnect from the computer that is running the workflow, but the workflow will remain running. For example, if you are running the workflow and managing the workflow on two different computers, you can sign out of or restart the computer from which you are managing the workflow, and continue to monitor workflow operations from another computer without interrupting the workflow.
- Task scheduling. You can schedule a task to start when specific conditions are met, as with any other Windows PowerShell cmdlet or script.

Creating a workflow

To write the workflow, use a script editor such as the Windows PowerShell Integrated Scripting Environment (ISE). This enforces workflow syntax and highlights syntax errors. For more information, review the tutorial **My first PowerShell Workflow runbook⁷¹**.

⁷¹ <https://azure.microsoft.com/en-us/documentation/articles/automation-first-runbook-textual/>

A benefit of using PowerShell ISE is that it automatically compiles your code and allows you to save the artifact. Because the syntactic differences between scripts and workflows are significant, a tool that knows both workflows and scripts will save you significant coding and testing time.

Syntax

When you create your workflow, begin with the **workflow** keyword, which identifies a workflow command to PowerShell. A script workflow requires the **workflow** keyword. Next, name the workflow, and have it follow the **workflow** keyword. The body of the workflow will be enclosed in braces.

A workflow is a Windows command type, so select a name with a verb-noun format:

```
workflow Test-Workflow

{
    ...
}
```

To add parameters to a workflow, use the **Param** keyword. These are the same techniques that you use to add parameters to a function.

Finally, add your standard PowerShell commands.

```
workflow MyFirstRunbook-Workflow

{
    Param(
        [string]$VMName,
        [string]$ResourceGroupName
    )
    ...
    Start-AzureRmVM -Name $VMName -ResourceGroupName $ResourceGroupName
}
```

Demonstration-Create and run a workflow runbook

This walkthrough will create a new PowerShell workflow runbook, test, publish and then run the runbook.

Prerequisites

- Note: You require an Azure subscription to perform the following steps. If you don't have one you can create one by following the steps outlined on the [Create your Azure free account today⁷²](#) webpage.

⁷² https://azure.microsoft.com/en-us/free/?ref=microsoft.com&utm_source=microsoft.com&utm_medium=docs&utm_campaign=visualstudio

Steps

Create a new runbook

1. In the **Azure portal**, open your Automation account.
2. Under **Process Automation**, select **Runbooks** to open the list of runbooks.
3. Create a new runbook by selecting the **Create a new runbook**.
4. Give the runbook the name **MyFirstRunbook-Workflow**.
5. You're going to create a PowerShell Workflow runbook, so for **Runbook type**, select **Powershell Workflow**.
6. Select **Create** to create the runbook and open the text editor.

Add code to a runbook

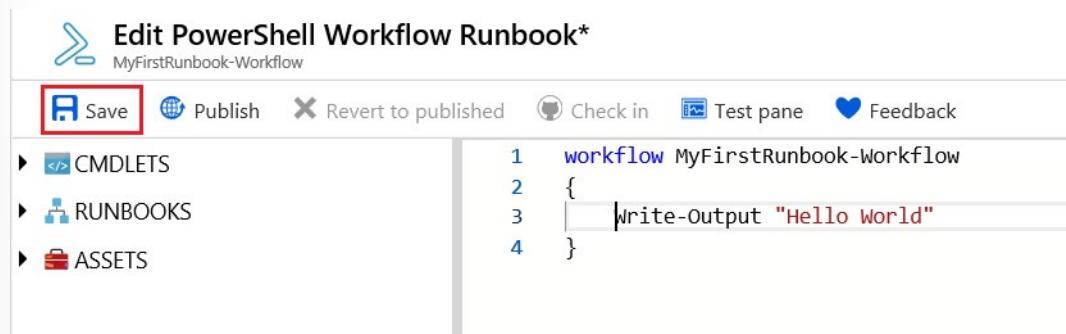
You have two options when adding code to a runbook. You can type code directly into the runbook, or you can select cmdlets, runbooks, and assets from the Library control and have them added to the runbook, along with any related parameters.

For this walkthrough, you'll use the type directly into the runbook method, as detailed in the following steps:

1. Type **Write-Output "Hello World."** between the braces, as per the below:

```
Workflow MyFirstRunbook-Workflow
{
    Write-Output "Hello World"
}
```

2. Save the runbook by selecting **Save**.



Test the runbook

Before you publish the runbook to production, you want to test it to ensure that it works properly. When you test a runbook, you run the draft version and view its output interactively, as demonstrated in the following steps:

1. Select the **Test** pane.

Edit PowerShell Workflow Runbook*
MyFirstRunbook-Workflow

Save Publish Revert to published Check in Test pane Feedback

CMDLETS RUNBOOKS ASSETS

```
1 workflow MyFirstRunbook-Workflow
2 {
3     Write-Output "Hello World"
4 }
```

2. Select **Start** to start the test. This should be the only enabled option.

Test
MyFirstRunbook-Workflow

Start Stop Suspend Resume View last test Refresh job streams

Parameters
No input parameters

Run Settings
Run on Azure

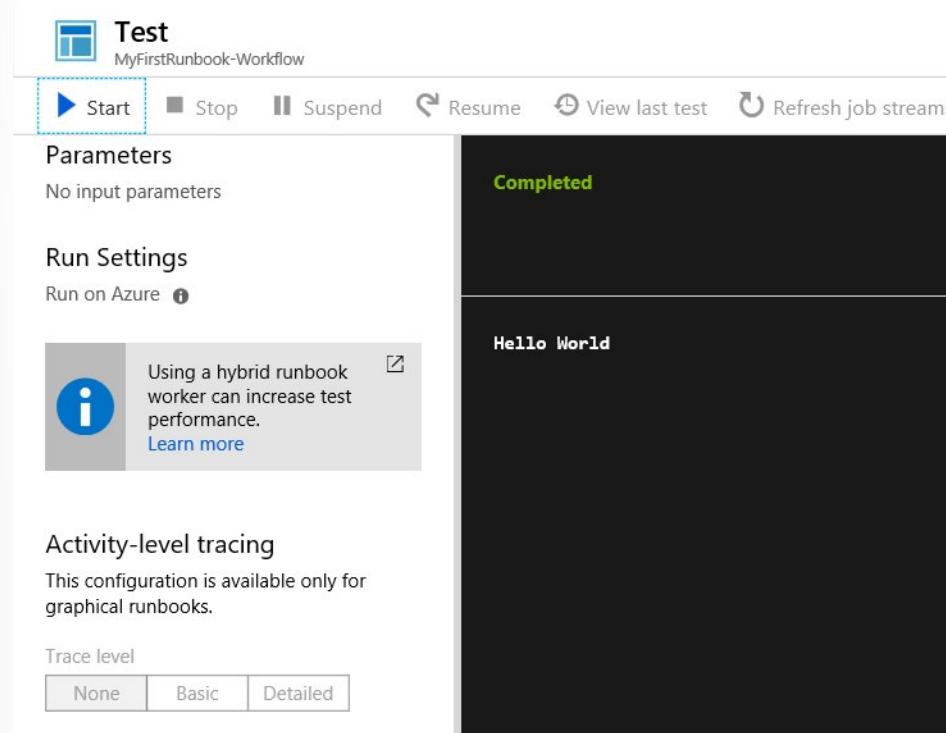
Activity-level tracing
This configuration is available only for graphical runbooks.

Trace level
None Basic Detailed

Click 'Start' to begin the test run.
Streams will display when the test completes.

A runbook job is created and its status displayed. The job status will start as Queued indicating that it is waiting for a runbook worker in the cloud to come available. It moves to Starting when a worker claims the job, and then Running when the runbook actually starts running. When the runbook job completes, its output displays. In your case, you should see Hello World.

3. When the runbook job finishes, close the **Test pane**.



Publish and run the runbook

The runbook that you created is still in draft mode. You need to publish it before you can run it in production. When you publish a runbook, you overwrite the existing published version with the draft version. In your case, you don't have a published version yet because you just created the runbook.

Use the following steps to publish your runbook:

1. In the runbook editor, select **Publish** to publish the runbook.
2. When prompted, select **Yes**.
3. Scroll left to view the runbook in the Runbooks pane, and ensure that it shows an **Authoring Status** of **Published**.
4. Scroll back to the right to view the pane for **MyFirstRunbook-Workflow**. Notice the options across the top:
 - Start
 - View
 - Edit
 - Link to schedule to start at some time in the future
 - Add a webhook
 - Delete
 - Export

The screenshot shows the Azure portal interface for a runbook named 'MyFirstRunbook-Workflow'. The left sidebar has a 'Runbook' icon and lists several sections: Overview (selected), Activity log, Tags, Diagnose and solve problems, Resources (Jobs, Schedules, Webhooks), Runbook settings, and Properties. The main content area is titled 'Overview' and contains the following details:

Resource group	az-auto-rg	Account	az-auto-ac-1
Subscription	Pay-As-You-Go	Subscription ID	974e6e39-73eb-48b0-9226-dae31425c367
Runbook type	PowerShell Workflow Runbook	Last modified	1/11/2019 10:14 AM
Tags	(change) Click here to add tags		

Below this, there is a section for 'Recent Jobs' with columns: STATUS, CREATED, and LAST UPDATED. A message indicates 'No jobs found.'

5. You just want to start the runbook, so select **Start**, and then when prompted, select **Yes**.
6. When the job pane opens for the runbook job that you created, leave it open so you can watch the job's progress.
7. Verify that at when the job completes, the job statuses that display in **Job Summary** match the statuses that you saw when you tested the runbook.

The screenshot shows the Azure portal interface for a workflow named 'MyFirstRunbook-Workflow'. At the top, there's a navigation bar with a blue square icon, the workflow name, the date and time (1/11/2019 10:16 AM), and standard controls like Resume, Stop, and Suspend.

The main area is divided into sections:

- Essentials:** Displays basic information:
 - Job Id: d91f3e85-7196-4fa8-8355-d88c0790ec8c
 - Created: 1/11/2019 10:16 AM
 - Job status: Completed
 - Last Update: 1/11/2019 10:17 AM
 - Run As: User
 - Runbook: MyFirstRunbook-Workflow
 - Ran on: Azure
 - Source snapshot: View source snapshot
- Overview:** A summary section with four metrics:
 - Input:** 0 (with a copy icon)
 - Output:** Output (with a copy icon)
 - All Logs:** (with a log icon)
 - Errors:** 0 (with a red X icon)
 - Warnings:** 0 (with a yellow exclamation mark icon)
- Exception:** None

Checkpoint and Parallel Processing

Workflows let you implement complex logic within your code. Two features available with workflows are checkpoints, and parallel processing.

Checkpoints

A **checkpoint** is a snapshot of the current state of the workflow. Checkpoints include the current value for variables, and any output generated up to that point. (For more information on what a checkpoint is, read the [checkpoint⁷³](#) webpage.)

If a workflow ends in an error or is suspended, the next time it runs it will start from its last checkpoint, instead of at the beginning of the workflow. You can set a checkpoint in a workflow with the **Checkpoint-Workflow** activity.

For example, in the following sample code if an exception occurs after Activity2, the workflow will end. When the workflow is run again, it starts with Activity2 because this followed just after the last checkpoint set.

⁷³ <https://docs.microsoft.com/en-us/azure/automation/automation-powershell-workflow#checkpoints>

```
<Activity1>
    Checkpoint-Workflow
        <Activity2>
            <Exception>
                <Activity3>
```

Parallel processing

A script block has multiple commands that run concurrently (or *in parallel*) instead of sequentially, as for a typical script. This is referred to as *parallel processing*. (More information about parallel processing is available on the [Parallel processing⁷⁴](#) webpage.)

In the following example, two *vm0* and *vm1* VMs will be started concurrently, and *vm2* will only start after *vm0* and *vm1* have started.

```
Parallel
{
    Start-AzureRmVM -Name $vm0 -ResourceGroupName $rg
    Start-AzureRmVM -Name $vm1 -ResourceGroupName $rg
}

Start-AzureRmVM -Name $vm2 -ResourceGroupName $rg
```

Another parallel processing example would be the following constructs that introduce some additional options:

- **ForEach -Parallel**. You can use the **ForEach -Parallel** construct to concurrently process commands for each item in a collection. The items in the collection are processed in parallel while the commands in the script block run sequentially.

In the following example, *Activity1* starts at the same time for all items in the collection. For each item, *Activity2* starts after *Activity1* completes. *Activity3* starts only after both *Activity1* and *Activity2* have completed for all items.

- *ThrottleLimit* - We use the *ThrottleLimit* parameter to limit parallelism. Too high of a *ThrottleLimit* can cause problems. The ideal value for the *ThrottleLimit* parameter depends on several environmental factors. Try start with a low *ThrottleLimit* value, and then increase the value until you find one that works for your specific circumstances:

```
ForEach -Parallel -ThrottleLimit 10 ($<item> in $<collection>)
{
    <Activity1>
    <Activity2>
}
<Activity3>
```

A real world example of this could be similar to the following code, where a message displays for each file after it is copied. Only after all files are completely copied does the final completion message display.

```
Workflow Copy-Files
{
```

⁷⁴ <https://docs.microsoft.com/en-us/azure/automation/automation-powershell-workflow#parallel-processing>

```
$files = @("C:\LocalPath\file1.txt","C:\LocalPath\file2.txt","C:\LocalPath\file3.txt")

ForEach -Parallel -ThrottleLimit 10 ($File in $Files)
{
    Copy-Item -Path $File -Destination \\NetworkPath
    Write-Output "$File copied."
}

Write-Output "All files copied."
}
```

Additional Automation Tools

Azure Software Development Kits

Azure provides several Software Development Kits (SDKs) to help you get operational and developing on Azure as quickly as possible. For a full and up-to-date list of available SDKs and developer tools, visit [Azure Developer Tools⁷⁵](#).

Many SDKs available for Azure include the following, which have different versions available for different platforms:

- [.NET SDK⁷⁶](#)
- [Java SDK⁷⁷](#)
- [Node SDK⁷⁸](#)
- [Python SDK⁷⁹](#)
- [PHP SDK⁸⁰](#)
- [Go SDK⁸¹](#)

✓ Note: You can also browse through or search [https://github.com/Azure⁸²](https://github.com/Azure) for languages relevant to you.

Azure REST APIs.md

Representational state transfer (*REST*) *APIs* are service endpoints that support sets of HTTP operations (methods). These service endpoints are responsible for providing, creating, retrieving, updating, or deleting access to the service's resources. To see a comprehensive set of REST APIs for Azure services, go to [REST API Browser⁸³](#).

Additional API specific services available on Azure are:

- API Management, which allows you to securely publish APIs to external, partner, and employee developers at scale. For more information, go to [API Management⁸⁴](#).
- Bing Maps API, which enables you to leverage Bing maps for services such as locations, routes, and fleet tracking. For more information, go to [BING Maps API⁸⁵](#).

Components of a REST API request/response

A REST API request/response pair can be separated into five different components:

1. The request URI. This consists of:

⁷⁵ <https://azure.microsoft.com/en-us/tools/>

⁷⁶ <https://azure.microsoft.com/en-us/develop/net/>

⁷⁷ <https://docs.microsoft.com/en-us/java/azure/?view=azure-java-stable>

⁷⁸ <https://docs.microsoft.com/en-us/javascript/azure/?view=azure-node-latest>

⁷⁹ <https://github.com/Azure/azure-sdk-for-python>

⁸⁰ <https://azure.microsoft.com/en-us/develop/php/>

⁸¹ <https://docs.microsoft.com/en-us/go/azure/>

⁸² <https://github.com/Azure>

⁸³ <https://docs.microsoft.com/en-us/rest/api/?view=Azure>

⁸⁴ <https://docs.microsoft.com/en-us/azure/api-management/>

⁸⁵ <https://www.microsoft.com/en-us/maps/choose-your-bing-maps-api>

```
{URI-scheme} :// {URI-host} / {resource-path} ? {query-string}.
```

- Although the request URI is included in the request message header, we call it out separately here because most languages or frameworks require you to pass it separately from the request message.
 - *URI scheme*. Indicates the protocol used to transmit the request. For example, http or https.
 - *URI host*. Specifies the domain name or IP address of the server where the REST service endpoint is hosted, such as graph.microsoft.com.
 - *Resource path*. Specifies the resource or resource collection, which might include multiple segments used by the service in determining the selection of those resources.
 - *Query string* (optional). Provides additional, simple parameters, such as the API version or resource selection criteria.
- 2. HTTP request message header fields. This is a required HTTP method (also known as an *operation* or *verb*) that tells the service what type of operation you are requesting. Azure REST APIs support **GET**, **HEAD**, **PUT**, **POST**, and **PATCH** methods.
- 3. Optional additional header fields. These are only if required by the specified URI and HTTP method. For example, an Authorization header that provides a bearer token containing client authorization information for the request.
- 4. HTTP response message header fields. This is an HTTP status code, ranging from 2xx success codes to 4xx or 5xx error codes.
- 5. Optional HTTP response message body fields. These MIME-encoded response objects are returned in the HTTP response body, such as a response from a **GET** method that is returning data. Typically, these objects are returned in a structured format such as JSON or XML, as indicated by the Content-type response header.

For additional details, review the video **How to call Azure REST APIs with Postman**⁸⁶.

Azure Cloud Shell

Azure Cloud Shell is a browser-based scripting environment for interacting with Azure. It provides the flexibility of choosing the shell experience that best suits the way you work, and is accessible from anywhere using the latest versions of the following browsers:

- Microsoft Edge
- Microsoft Internet Explorer
- Google Chrome
- Mozilla Firefox
- Safari

Linux users can opt for a Bash experience, while Windows users can also opt for Windows PowerShell.

You must have a storage account to use the Cloud Shell, and you will be prompted to create one when accessing Azure Cloud Shell.

✓ Note: You can access Azure Cloud Shell by going to <https://shell.azure.com/>⁸⁷.

⁸⁶ <https://docs.microsoft.com/en-us/rest/api/azure/>

⁸⁷ <https://shell.azure.com/>

Cloud Shell is also accessible from within the Azure portal by selecting the Azure Cloud Shell icon at the top of the browser.



If you have time you can also review the [PowerShell in Azure Cloud Shell GA⁸⁸](#) video for more details.

Package Management

Package management allows you to install all the software you need in an environment, into your VM during or after its deployment, or after its install.

Using package management, it's possible to manage all aspects of software such as installation, configuration, upgrade, and uninstall. There's a wide range of packaged software available for you to install using the package managers, such as Java, Microsoft Visual Studio, Google Chrome, GIT, and many more.

There are also a number of package management solutions available for you to use depending on your environment and needs:

- [apt⁸⁹](#): apt is the package manager for Debian Linux environments.
 - [Yum⁹⁰](#): Yum is the package manager for CentOS Linux environments.
 - [Chocolatey⁹¹](#): Chocolatey is the software management solution built on Windows PowerShell for Windows operating systems.
- ✓ Note: In the following section we will cover installing Chocolatey, as an example. However, while the other package management solutions use different syntax and commands, they have similar concepts.

Install Chocolatey

Chocolatey does not have an .msi package; it installs as a nupkg using a PowerShell install script. The installation script is available to review at <https://chocolatey.org/install.ps1⁹²>.

You can run the script and install it in a variety of ways, which you can read about at [More Install Options⁹³](#).

The following example installs the script via PowerShell:

1. Open a PowerShell window as administrator, and run the following command.

```
Set-ExecutionPolicy Bypass -Scope Process -Force; iwr https://chocolatey.org/install.ps1 -UseBasicParsing | iex
```

2. After the command completes, run the following command:

```
choco /?
```

3. To search for a Visual Studio package that you can use, run the following command:

⁸⁸ <https://azure.microsoft.com/en-us/resources/videos/azure-friday-powershell-in-azure-cloud-shell-ga/>

⁸⁹ <https://wiki.debian.org/Apt>

⁹⁰ <https://wiki.centos.org/PackageManagement/Yum>

⁹¹ <https://chocolatey.org/>

⁹² <https://chocolatey.org/install.ps1>

⁹³ <https://chocolatey.org/install#install-with-powershellexe>

```
choco search visualstudio2017
```

You can install packages manually via the command line using **choco install**. To install packages into your development, test, and production environments, identify a package you want to install, and then list that package in a PowerShell script. When deploying your VM, you then call the package as part of a custom script extension. An example of such a PowerShell script would be:

```
# Set PowerShell execution policy
Set-ExecutionPolicy RemoteSigned -Force

# Install Chocolatey
iwr https://chocolatey.org/install.ps1 -UseBasicParsing | iex

refreshenv

# Install Chocolatey packages
& choco install poshgit -y
& choco install googlechrome -y
& choco install firefox -y
& choco install notepadplusplus -y
& choco install putty -y
& choco installchefdk -y

refreshenv
```

Lab

Azure Deployments using Resource Manager Templates



Steps for the labs are available on **GitHub** at the following websites in their **Infrastructure as Code** sections:

- **Parts unlimited**⁹⁴
- **Parts unlimited MRP**⁹⁵

For the individual lab tasks for this module, select the following PartsUnlimited link and follow the outlined steps for each lab task.

PartsUnlimited (PU)

- **Azure Deployments using Resource Manager templates**⁹⁶

⁹⁴ <https://microsoft.github.io/PartsUnlimited>

⁹⁵ <https://microsoft.github.io/PartsUnlimitedMRP>

⁹⁶ <http://microsoft.github.io/PartsUnlimited/iac/200.2x-laC-AZ-400T05Applnra.html>

Module Review and Takeaways

Module Review Questions

Checkbox

What benefits from the list below can you achieve by modularizing your infrastructure and configuration resources?

(Choose three)

- Easy to reuse across different environments
- Easier to manage and maintain your code
- More difficult to sub-divide up work and ownership responsibilities
- Easier to troubleshoot
- Easier to extend and add to your existing infrastructure definitions

Multiple choice

Which method of approach for implementing Infrastructure as Code states what the final state of an environment should be without defining how it should be achieved?

- Scripted
- Imperative
- Object orientated
- Declarative

Review Question 3

Which term defines the ability to apply one or more operations against a resource, resulting in the same outcome every time?

- Declarative
- Idempotency
- Configuration drift
- Technical debt

Checkbox

Which of the following are possible causes of Technical debt?

(choose all that apply)

- Unplanned for localization of an application
- Accessibility
- Changes made quickly, or directly to an application without using DevOps methodologies
- Changing technologies or versions that are not accounted for as part of the dev process

Multiple choice

Which term is the process whereby a set of resources change their state over time from their original state in which they were deployed?

- Modularization
- Technical debt
- Configuration drift
- Imperative

Multiple choice

Which of the following options is a method for running configuration scripts on a VM either during or after deployment?

- Using the (CSE)
- Using Quickstart templates
- Using the dependsOn parameter
- Using Azure Key Vault

Multiple choice

When using Azure CLI, what's the first action you need to take when preparing to run a command or script ?

- Define the Resource Manager template.
- Specify VM extension details.
- Create a resource group.
- Log in to your Azure subscription.

Multiple choice

Which Resource Manager deployment mode only deploys whatever is defined in the template, and does not remove or modify any other resources not defined in the template?

- Validate
- Incremental
- Complete
- Partial

Multiple choice

Which package management tool is a software management solution built on Powershell for Windows operating systems?

- Yum
- Chocolatey
- apt
- Apache Maven

Checkbox

*Which of the following version control tools are available for use with Azure DevOps?
(choose all that apply)*

- Subversion
- Git
- BitBucket
- TFVC

Answers

Checkbox

What benefits from the list below can you achieve by modularizing your infrastructure and configuration resources?

(Choose three)

- Easy to reuse across different environments
- Easier to manage and maintain your code
- More difficult to sub-divide up work and ownership responsibilities
- Easier to troubleshoot
- Easier to extend and add to your existing infrastructure definitions

Explanation

The following answers are correct:

More difficult to sub-divide up work and ownership responsibilities is incorrect. It is easier to sub-divide up work and ownership responsibilities.

Multiple choice

Which method of approach for implementing Infrastructure as Code states what the final state of an environment should be without defining how it should be achieved?

- Scripted
- Imperative
- Object orientated
- Declarative

Explanation

Declarative is the correct answer. The declarative approach states what the final state should be. When run, the script or definition will initialize or configure the machine to have the finished state that was declared, without defining how that final state should be achieved.

All other answers are incorrect. Scripted is not a methodology, and in the imperative approach, the script states the how for the final state of the machine by executing through the steps to get to the finished state. It defines what the final state needs to be, but also includes how to achieve that final state.

Object orientated is a coding methodology, but does include methodologies for how states and outcomes are to be achieved.

Review Question 3

Which term defines the ability to apply one or more operations against a resource, resulting in the same outcome every time?

- Declarative
- Idempotency
- Configuration drift
- Technical debt

Explanation

Idempotency is the correct answer. It is a mathematical term that can be used in the context of Infrastructure as Code and Configuration as Code, as the ability to apply one or more operation against a resource, resulting in the same outcome.

All other answers are incorrect.

Checkbox

Which of the following are possible causes of Technical debt?

(choose all that apply)

- Unplanned for localization of an application
- Accessibility
- Changes made quickly, or directly to an application without using DevOps methodologies
- Changing technologies or versions that are not accounted for as part of the dev process

Explanation

All answers are correct.

Multiple choice

Which term is the process whereby a set of resources change their state over time from their original state in which they were deployed?

- Modularization
- Technical debt
- Configuration drift
- Imperative

Explanation

Configuration drift is the correct answer. It is the process whereby a set of resources change their state over time from the original state in which they were deployed.

All other answers are incorrect.

Multiple choice

Which of the following options is a method for running configuration scripts on a VM either during or after deployment?

- Using the (CSE)
- Using Quickstart templates
- Using the dependsOn parameter
- Using Azure Key Vault

Explanation

Using CSE is the correct answer, because it is a way to download and run scripts on your Azure VMs. All other answers are incorrect.

Quickstart templates are publicly available starter templates that allow you get up and running quickly with Resource Manager templates.

*The *dependsOn *parameter defines depend resources in a Resource Manager template.*

Azure Key Vault is a secrets-management service in Azure that allows you to store certificates, keys, passwords, and so forth.

Multiple choice

When using Azure CLI, what's the first action you need to take when preparing to run a command or script ?

- Define the Resource Manager template.
- Specify VM extension details.
- Create a resource group.
- Log in to your Azure subscription.

Explanation

Log in to your Azure subscription is the correct answer. You can do so using the command az login. All other answers are incorrect.

You do not need to define the Resource Manager template or specify the VM extension details, and you cannot create a resource group without first logging into your Azure subscription.

Multiple choice

Which Resource Manager deployment mode only deploys whatever is defined in the template, and does not remove or modify any other resources not defined in the template?

- Validate
- Incremental
- Complete
- Partial

Explanation

Incremental is the correct answer.

Validate mode only compiles the templates, and validates the deployment to ensure the template is functional. For example, it ensures there no circular dependencies and the syntax is correct.

Incremental mode only deploys whatever is defined in the template, and does not remove or modify any resources that are not defined in the template. For example, if you have deployed a VM via template, and then renamed the VM in the template, the first VM deployed will still remain after the template is run again. Incremental mode is the default mode.

In Complete mode, Resource Manager deletes resources that exist in the resource group but aren't specified in the template. For example, only resources defined in the template will be present in the resource group after the template is deployed. As a best practice, use the Complete mode for production environments where possible, to try to achieve idempotency in your deployment templates.

Multiple choice

Which package management tool is a software management solution built on Powershell for Windows operating systems?

- Yum
- Chocolatey
- apt
- Apache Maven

Explanation

Chocolatey is the correct answer.

apt is the package manager for Debian Linux environments.

Yum is the package manager for CentOS Linux environments.

Maven is a build automation for build artifacts used as part of a build and release pipeline, with Java-based projects

Checkbox

Which of the following version control tools are available for use with Azure DevOps?
(choose all that apply)

- Subversion
- Git
- BitBucket
- TFVC

Explanation

All answers are correct.

Subversion, Git, BitBucket and TFVC are all repository types that are available with Azure DevOps.

Module 15 Azure Deployment Models and Services

Module Overview

Module Overview

You're ready to start deploying and migrating applications into Microsoft's Azure cloud platform — but there are different deployment models to contend with. Which should you choose? Each has strengths and weaknesses depending on the service you are setting up. Some might require more attention than others, but offer additional control. Others integrate services like load balancing or Operating Systems as more of a Platform as a Service.

Learn the differences between IaaS, PaaS and FaaS, and when you might want to choose one over another.

Learning Objectives

After completing this module, students will be able to:

- Describe deployment models and services that are available with Azure

Deployment Modules and Options

Comparing IaaS, PaaS, and FaaS

Infrastructure as a Service (IaaS)

With IaaS, you provision the VMs that you need along with associated network and storage components. You then deploy whatever software and applications you want onto those VMs. This model is closest to a traditional on-premises environment except that Microsoft manages the infrastructure. You still manage the individual VMs.

Platform as a Service (PaaS)

PaaS provides a managed hosting environment where you can deploy your application without needing to manage VMs or networking resources. For example, instead of creating individual VMs, you specify an instance count and the service will provision, configure, and manage the necessary resources. Azure App Service is an example of a PaaS service.

There is a spectrum from IaaS to pure PaaS. For example, Azure VMs can auto-scale by using VM Scale Sets. This automatic scaling capability isn't strictly with PaaS, but it's the type of management feature that might be found in a PaaS service.

Function as a Service (FaaS)

FaaS goes even further in removing the need to worry about the hosting environment. Instead of creating compute instances and deploying code to those instances, you simply deploy your code and the service automatically runs it. You don't need to administer the compute resources, because the services make use of serverless architecture. They seamlessly scale up or down to whatever level necessary to manage the traffic. Azure Functions are a FaaS service.

When comparing the three environments, remember that:

- IaaS gives the most control, flexibility, and portability.
- FaaS provides simplicity, elastic scale, and potential cost savings because you pay only for the time your code is running.
- PaaS falls somewhere between the two.

In general, the more flexibility a service provides, the more responsible you are for configuring and managing the resources. FaaS services automatically manage nearly all aspects of running an application, while IaaS solutions require you to provision, configure, and manage the VMs and network components you create.

Azure Compute options

The main compute options currently available in Azure are:

- IaaS:
 - **Azure virtual machines**¹ allow you to deploy and manage VMs inside an Azure Virtual Network.

¹ <https://azure.microsoft.com/en-us/services/virtual-machines/>

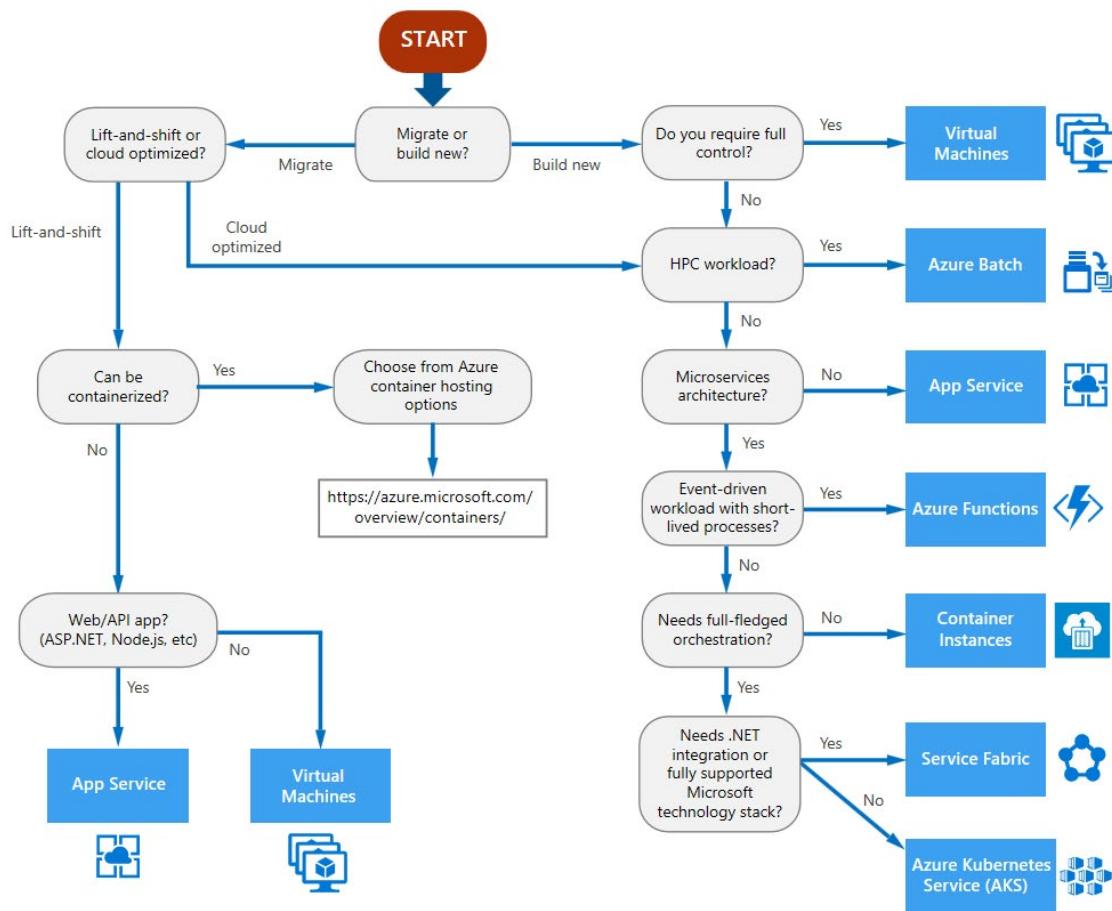
- PaaS:
 - **Azure App service²** is a managed PaaS offering for hosting web apps, mobile app back-ends, RESTful APIs, or automated business processes.
 - **Azure Container Instances³** offer the fastest and simplest way to run a container in Azure without having to provision any VMs or adopting a higher-level service.
 - **Azure Cloud services⁴** is a managed service for running cloud applications.
- FaaS:
 - **Azure Functions⁵** is a managed FaaS service.
 - **Azure Batch⁶** is also a managed FaaS service, and is for running large-scale parallel and HPC applications.
- Modern native cloud apps, providing massive scale and distribution:
 - **Azure Service Fabric⁷** is a distributed systems platform that can run in many environments, including Azure or on premises. Service Fabric is an orchestrator of microservices across a cluster of machines.
 - **AKS⁸** lets you create, configure, and manage a cluster of VMs that are preconfigured to run containerized applications.

Choosing a compute service

Azure offers a number of ways to host your application code. The term *compute* refers to the hosting model for the computing resources that your application runs on.

The following flowchart will help you to choose a compute service for your application. It guides you through a set of key decision criteria to reach a recommendation.

² <https://azure.microsoft.com/en-us/services/app-service/>
³ <https://azure.microsoft.com/en-us/services/container-instances/>
⁴ <https://azure.microsoft.com/en-us/services/cloud-services/>
⁵ <https://azure.microsoft.com/en-us/services/functions/>
⁶ <https://azure.microsoft.com/en-us/services/batch/>
⁷ <https://azure.microsoft.com/en-us/services/service-fabric/>
⁸ <https://azure.microsoft.com/en-us/services/kubernetes-service/>



Treat this flowchart as a starting point. Every application has unique requirements, so use the recommendation as a starting point. Then perform a more detailed evaluation, looking at aspects such as:

- Feature sets
- Service limits
- Cost
- SLA
- Regional availability
- Developer ecosystem and team skills
- Compute comparison tables

If your application consists of multiple workloads, evaluate each workload separately. A complete solution might incorporate two or more compute services.

Azure Infrastructure-as-a-Service (IaaS) Services

Azure virtual machine

A Microsoft Azure virtual machine (VM) provides the flexibility of virtualization without having to buy and maintain the physical hardware that runs it. However, you still need to maintain the VM by performing tasks such as deploying, configuring, and maintaining the software that runs on it. Azure VMs provides more control over computing environments than other choices offer.

Operating system support

Azure virtual machines supports both Windows operating system (OS) and Linux OS deployments. You can also choose to upload and use your own image.

Note: If you opt to use your own image, the publisher name, offer, and SKU aren't used.

Windows operating systems

Azure provides both Windows Server and Windows client images for use in development, test, and production. Azure provides several marketplace images to use with various versions and types of Windows Server operating systems. Marketplace images are identified by image publisher, offer, SKU, and version (typically version is specified as latest). However, only 64-bit operating systems are supported.

As an example, to find and list available Windows Server SKUs in westeurope, run the following commands one after the other:

```
az vm image list-publishers --location westeurope --query "[?starts_
with(name, 'Microsoft')]"
az vm image list-offers --location westeurope --publisher MicrosoftWindows-
Server
az vm image list-skus --location westeurope --publisher MicrosoftWindows-
Server --offer windowsserver
```

Linux

Azure provides endorsed Linux distributions. *Endorsed distributions* are distributions that are available on Azure Marketplace and are fully supported. The images in Azure Marketplace are provided and maintained by the Microsoft partner who produces them. Some of the endorsed distributions available in Azure Marketplace include:

- CentOS
- CoreOS
- Debian
- Oracle Linux
- Red Hat
- SUSE Linux Enterprise
- OpenSUSE

- Ubuntu
- RancherOS

There are several other Linux-based partner products that you can deploy to Azure VMs, including Docker, Bitnami by VMWare, and Jenkins. A full list of endorsed Linux distributions is available at [Endorsed Linux distributions on Azure](#)⁹.

As an example, to find and list available RedHat SKUs in westus, run the following commands one after the other:

```
az vm image list-publishers --location westus --query "[?contains(name, 'RedHat')]"  
az vm image list-offers --location westus --publisher RedHat  
az vm image list-skus --location westus --publisher RedHat --offer RHEL
```

If you want to use a Linux version not on the endorsed list and not available in Azure Marketplace, you can install it directly.

Azure VMs usage scenarios

You can use Azure VMs in various ways. Some examples are:

- Development and test. Azure VMs offer a quick and easier way to create a computer with specific configurations that are required to code and test an application.
- Applications in the cloud. Because demand for an application can fluctuate, it might make economic sense to run it on a VM in Azure. You pay for extra VMs when you need them, and shut them down when you don't.
- Extended datacenter. VMs in an Azure virtual network can more easily be connected to your organization's network.

The number of VMs that your application uses can scale up and out to whatever is required to meet your needs. Conversely, it can scale back down when you don't need it, thereby avoiding unnecessary charges.

Deployment, configuration management and extensions

Azure VMs support a number of deployment and configuration management toolsets, including:

- Azure Resource Manager (ARM) templates
- Windows PowerShell Desired State Configuration (DSC)
- Ansible
- Chef
- Puppet
- Terraform by HashiCorp

VM extensions give your VM additional capabilities through post-deployment configuration and automated tasks. Some common tasks you can complete using extensions include:

- Running custom scripts. The Custom Script Extension (CSE) helps you configure workloads on the VM by running your script when you provision the VM.

⁹ <https://docs.microsoft.com/en-us/azure/virtual-machines/linux/endorsed-distros>

- Deploying and managing configurations. The PowerShell DSC extension helps you set up DSC on a VM to manage configurations and environments.
- Collect diagnostics data. The Azure Diagnostics extension helps you configure the VM to collect diagnostics data that you can use to monitor your application's health.

Azure VM service limits and ARM

Each Azure service has its own service limits, quotas, and constraints. For example, some of the service limits for VMs when you use ARM and Azure resource groups are in the following table.

Resource	Default Limit
VMs per availability set	200
Certificates per subscription	Unlimited

Scaling Azure VMs

Scale is provided for in Azure virtual machines (VMs) using VM scale sets (or VMSS). Azure VM scale sets let you create and manage a group of identical, load-balanced VMs. The number of VM instances can automatically increase or decrease in response to demand or a defined schedule. Scale sets provide high availability to your applications, and allow you to centrally manage, configure, and update a large number of VMs. With VM scale sets, you can build large-scale services for areas such as compute, big data, and container workloads.

Why and when to use VM scale sets

Azure VM scale sets provide the management capabilities for applications that run across many VMs, automatic scaling of resources, and load-balancing of traffic. Scale sets provide the following key benefits:

- Easier to create and manage multiple VMs
- Provide high availability and application resiliency
- Allow your application to automatically scale as resource demand changes
- Work at large-scale
- Useful when deploying highly available infrastructure where a set of machines has similar configurations

There are two basic ways to configure VMs deployed in a scale set:

- Use extensions to configure the VM after it's deployed. With this approach, new VM instances could take longer to start than a VM with no extensions.
- Deploy a managed disk with a custom disk image. This option might be quicker to deploy, but it requires you to keep the image up to date.

Differences between virtual machines and scale sets

Scale sets are built from VMs. With scale sets, the management and automation layers are provided to run and scale your applications. However, you also could manually create and manage individual VMs, or integrate existing tools to build a similar level of automation. The following table outlines the benefits of scale sets compared to manually managing multiple VM instances.

Scenario	Manual group of VMs	VM scale set
Add additional VM instances	Manual process to create, configure, and ensure compliance	Automatically create from central configuration
Traffic balancing and distribution	Manual process to create and configure Azure Load Balancer or Application Gateway	Can automatically create and integrate with Azure Load Balancer or Application Gateway
High availability and redundancy	Manually create an availability set or distribute and track VMs across availability zones	Automatic distribution of VM instances across availability zones or availability sets
Scaling of VMs	Manual monitoring and Azure Automation	Autoscale based on host metrics, in-guest metrics, Application Insights, or schedule

VM scale set limits

Each Azure service has service limits, quotas, and constraints. Some of the service limits for VM scale sets are listed in the following table.

Resource	Default limit	Maximum limit
Maximum number of VMs in a scale set	1,000	1,000
Maximum number of scale sets in a region	600	600
Maximum number of scale sets in a region	2,000	2,000

✓ Note: There is no additional cost to use the VM scale sets service. You only pay for the underlying compute resources such as the VM instances, Azure Load Balancer, or managed disk storage that you consume. The management and automation features, such as autoscale and redundancy, incur no additional charges over the use of VMs.

Demonstration-Create virtual machine scale set

In the following steps we will create a virtual machine scale set (VM scale set), deploy a sample application, and configure traffic access with Azure Load Balancer.

Prerequisites

- You require an Azure subscription to perform the following steps. If you don't have one, you can create one by following the steps outlined on the [Create your Azure free account today¹⁰](#) webpage.

Note: If you use your own values for the parameters used in the following commands, such as resource group name and scale set name, remember to change them in the subsequent commands as well to ensure the commands run successfully.

¹⁰ https://azure.microsoft.com/en-us/free/?ref=microsoft.com&utm_source=microsoft.com&utm_medium=docs&utm_campaign=visualstudio

Steps

1. Create a scale set. Before you can create a scale set, you must create a resource group using the following command:

```
az group create --name myResourceGroup --location < your closest datacenter>
```

Now create a virtual machine scale set with the following command:

```
az vmss create \
--resource-group myResourceGroup \
--name myScaleSet \
--image UbuntuLTS \
--upgrade-policy-mode automatic \
--admin-username azureuser \
--generate-ssh-keys
```

This creates a scale set named **myScaleSet** that is set to automatically update as changes are applied. It also generates Secure Shell (SSH) keys if they do not exist in `~/.ssh/id_rsa`.

2. Deploy a sample application. To test your scale set, install a basic web application, and a basic NGINX web server. The Azure Custom Script Extension (CSE) downloads and runs a script that installs the sample web application on the VM instance (or instances). To install the basic web application, run the following command:

```
az vmss extension set \
--publisher Microsoft.Azure.Extensions \
--version 2.0 \
--name CustomScript \
--resource-group myResourceGroup \
--vmss-name myScaleSet \
--settings '{"fileUris": ["https://raw.githubusercontent.com/Microsoft/PartsUnlimitedMRP/master/Labfiles/AZ-400T05-ImplemntgAppInfra/Labfiles/automate_nginx.sh"], "commandToExecute": "./automate_nginx.sh"}'
```

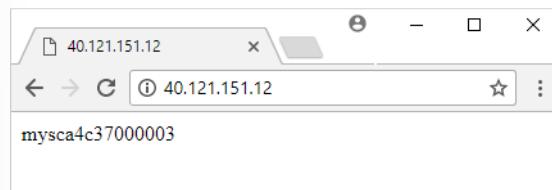
3. Allow traffic to access the application. When you created the scale set, the Azure Load Balancer deployed automatically. As a result, traffic distributes to the VM instances in the scale set. To allow traffic to reach the sample web application, create a load balancer rule with the following command:

```
az network lb rule create \
--resource-group myResourceGroup \
--name myLoadBalancerRuleWeb \
--lb-name myScaleSetLB \
--backend-pool-name myScaleSetLBEPool \
--backend-port 80 \
--frontend-ip-name loadBalancerFrontEnd \
--frontend-port 80 \
--protocol tcp
```

4. Obtain the public IP Address. To test your scale set and observe your scale set in action, access the sample web application in a web browser, and then obtain the public IP address of your load balancer using the following command:

```
az network public-ip show \
--resource-group myResourceGroup \
--name myScaleSetLBPublicIP \
--query '[ipAddress]' \
--output tsv
```

5. Test your scale set. Enter the public IP address of the load balancer in a web browser. The load balancer distributes traffic to one of your VM instances, as in the following screenshot:



6. Remove the resource group, scale set, and all related resources as follows. The `--no-wait` parameter returns control to the prompt without waiting for the operation to complete. The `--yes` parameter confirms that you wish to delete the resources without an additional prompt to do so.

```
az group delete --name myResourceGroup --yes --no-wait
```

Availability

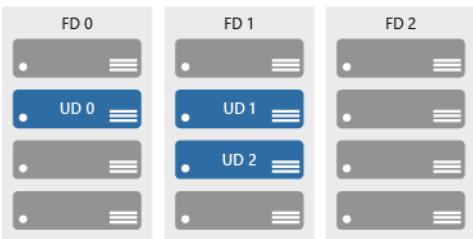
Availability in infrastructure as a service (IaaS) services in Azure is provided both through the core physical structural components of Azure (such as Azure regions, and Availability Zones), and also through logical components that together provide for overall availability during outages, maintenance, and other downtime scenarios.

Availability sets

Availability sets ensure that the VMs you deploy on Azure are distributed across multiple, isolated hardware clusters. Doing this ensures that if a hardware or software failure within Azure happens, only a subset of your VMs are impacted and your overall solution remains available and operational.

Availability sets are made up of update domains and fault domains:

- Update domains. When a maintenance event occurs (such as a performance update or critical security patch applied to the host), the update is sequenced through update domains. Sequencing updates using update domains ensures that the entire datacenter isn't unavailable during platform updates and patching. Update domains are a *logical* section of the datacenter, and they are implemented with software and logic.
- Fault domains. Fault domains provide for the physical separation of your workload across different hardware in the datacenter. This includes power, cooling, and network hardware that supports the physical servers located in the server racks. In the event the hardware that supports a server rack becomes unavailable, only that rack of servers is affected by the outage.



Update management

You can also leverage the Update management service within a Windows or Linux VM to manage updates and patches for the VMs. Directly from your VM, you can quickly assess the status of available updates, schedule required updates installation, and review deployment results to verify updates were applied successfully.

Enable update management

To enable **Update management** for your VM:

1. On the left side of the screen in the **Azure Portal**, select **Virtual machines**.
2. select a VM From the list.
3. On the VM screen, in the **Operations** section, click **Update management**. The **Enable Update Management** window opens.

If any of the following prerequisites were found to be missing during onboarding, they're automatically added:

- *Log Analytics workspace*. Provides a single location to review and analyze data generated by the VM features and services, such as update management.
- *Automation*. allows you to run runbooks against VMs, such as download and apply updates.
- A *Hybrid runbook worker* is enabled on the VM. Used to communicate with the VM and obtain information about the update status.

Additional IaaS considerations\

When using Infrastructure as a Service (IaaS) virtual machines that are relevant to DevOps scenarios, there are a number of other relevant areas to consider.

Monitoring and analytics

Boot diagnostics

The boot diagnostic agent captures screen output that can be used for troubleshooting purposes. This capability is enabled by default with Windows VMs, but it's not automatically enabled when you create a Linux VM using Azure CLI. The captured screenshots are stored in an Azure storage account, which is also created by default.

Host metrics

An Azure VM, for both the Windows and Linux operating systems, have a dedicated host in Azure that it interacts with. Metrics are automatically collected for the host, which you can view in the Azure portal.

Enable diagnostic extensions

To see more granular and VM-specific metrics, you need to install the Azure diagnostics extension on the VM. These allow you to retrieve additional monitoring and diagnostics data from the VM. You can view these performance metrics and create alerts based on the VM performance.

You can enable diagnostic extensions in the Portal using the following steps:

1. In the Azure portal, choose Resource Groups, select **myResourceGroupMonitor**, and then in the resource list, select **myVM**.
2. Select **Diagnosis settings**. In the **Pick a storage account** drop-down, choose or create a storage account.
3. Select the **Enable guest-level monitoring** button.

Alerts

You can create alerts based on specific performance metrics. You can use these alerts to notify you, for example, when average CPU usage exceeds a certain threshold or available free disk space drops below a certain amount. Alerts display in the Azure portal, or you can have them sent via email. You can also trigger Azure Automation runbooks or Azure Logic Apps in response to alerts being generated.

The following steps create an alert for average CPU usage:

1. In the Azure portal, select **Resource Groups**, select **myResourceGroupMonitor**, and then in the resource list **select myVM**.
2. On the **VM** blade, select **Alert rules**. Then from the top of the **Alerts** blade, select **Add metric alert**.
3. Provide a name for your alert, such as *myAlertRule*.
4. To trigger an alert when CPU percentage exceeds 1.0 for five minutes, leave all the other default settings selected.
5. Optionally, you can select the **Email owners**, **Contributors**, and **Readers** check boxes to send email notifications. The default action is to present a notification in the portal only.
6. Select **OK**.

Load balancing

Azure Load Balancer is a Layer-4 (Transmission Control Protocol (TCP), User Datagram Protocol (UDP)) load balancer that provides high availability by distributing incoming traffic among healthy VMs.

For load balancing, you define a front-end IP configuration that contains one or more public IP addresses. This configuration allows your load balancer and applications to be accessible over the internet.

Virtual machines connect to a load balancer using their virtual network interface card (NIC). To distribute traffic to the VMs, a back-end address pool contains the IP addresses of the virtual NICs connected to the load balancer.

To control the flow of traffic, you define load-balancer rules for specific ports and protocols that map to your VMs.

When creating a VM scale set, a load balancer is automatically created as part of that process.

Migrating workloads to Azure

Using Azure infrastructure as a service (IaaS) VMs, you can migrate workloads from AWS and on-premises to Azure. You can upload VHD files from Amazon Web Services (AWS) or on-premises virtualization solutions to Azure to create VMs that utilize the Azure feature, Managed Disks.

You can also move from classic Azure cloud services deployments to Azure Resource Manager type deployments.

VMs versus containers

Containers are becoming increasingly popular depending on an organization's needs. This is because they offer numerous advantages over other environments such as VMs. The following are descriptions of some of these advantages.

Less resource intensive

Compared to running a VM, running a container requires relatively few resources. This is because the operating system is shared. When you start a VM you boot an entirely new operating system on top of the running operating system, and they share only the hardware. With containers, you share the memory, the disk, and the CPU. This means that the overhead associated with starting a container is quite low, and that the container also provides optimal isolation.

Fast startup

Because running a container requires only a few extra resources over the operating system, startup time is faster, and roughly equivalent to the time required to start a new process. The only additional items the OS needs to set up is the isolation for the process, which is done at the kernel level, and occurs quickly.

Improved server density

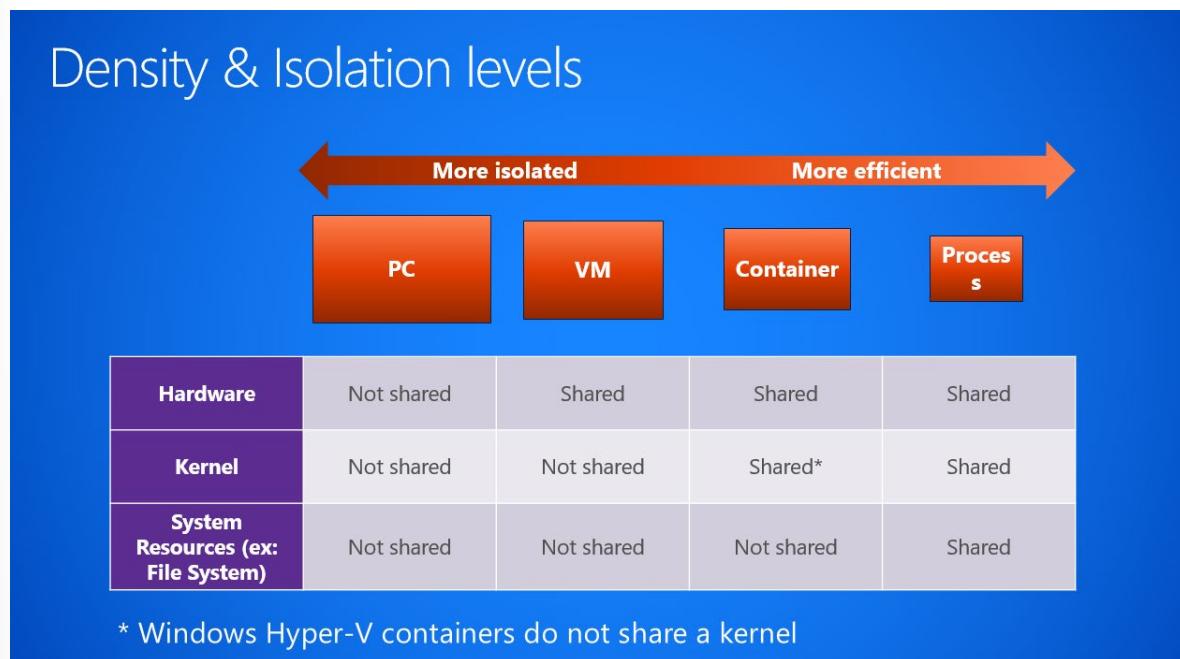
When you own hardware, you want to utilize that hardware as efficiently and cost-effectively as possible. With the introduction of VMs, sharing hardware among multiple VMs was introduced.

Containers take this sharing concept one step further by enabling even more efficient utilization of memory, disk, and CPU from the available hardware. This is because VMs only consume the memory and CPU that they need. This results in fewer idle servers, resulting in better utilization of the existing compute resources.

This is an especially important consideration for cloud providers. The higher the server density (the number of things that you can do with the hardware that you have), the more cost efficient the data-center becomes. It's not surprising that containers are becoming more popular and that new tools for managing and maintaining containerized solutions are rapidly emerging.

Density and isolation comparison

Containers allow for more density, scalability, and agility than VMs do, but they also enable more isolation than processes permit. The following diagram exhibits a comparison of density and isolation levels across the range of possible environments.



Windows vs. Linux containers

It is important to understand that containers are part of the operating system, and that each container shares the kernel of that operating system. This generally means that if you create an image on a machine running the Windows operating system, it is a Windows-specific image that needs a windows environment to run and vice versa.

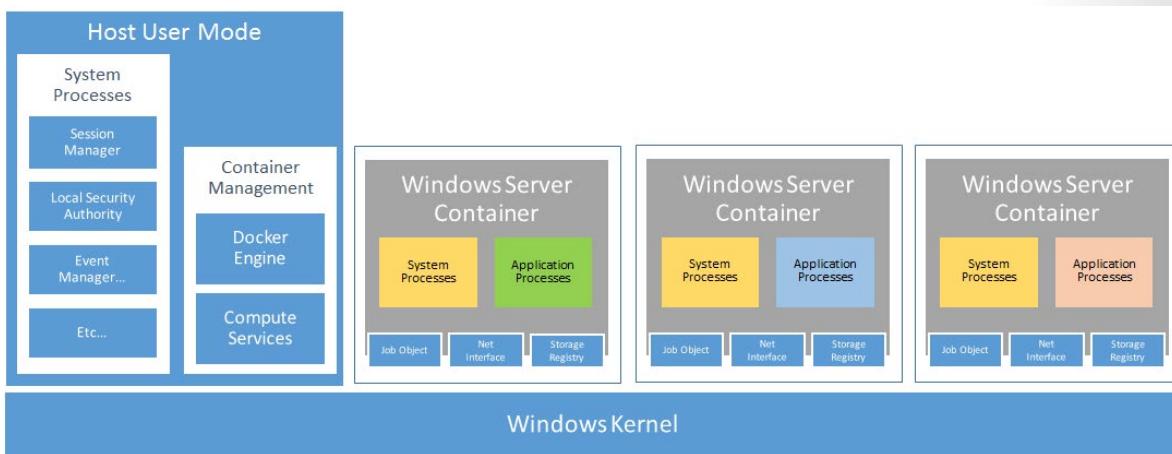
There are tools native with Windows, such as **Windows Subsystem on Linux (WSL)** available in both *Windows 10* and *Windows Server 2019*, and **Docker for Windows**, which do allow you to run Linux and Windows containers side by side.

Windows Server and Hyper-V containers

Windows Server and Windows 10 can create and run different types of containers and they each have different characteristics.

Windows Server containers

The following diagram of high-level architecture shows how containers works on the Windows operating system.



As the diagram shows, the Windows operating system always has its default host user mode processes running Windows. On Windows, you now have additional services such as Docker and compute services that manage containers.

When you start a new container, Docker talks to the compute services to create a new container based on an image. For each container, Docker will create a Windows container, each of which will require a set of system processes. These are always the same in every container. You then use your own application process to differentiate each container. These can be Microsoft Internet Information Services (IIS) or SQL Server processes that you run in the container.

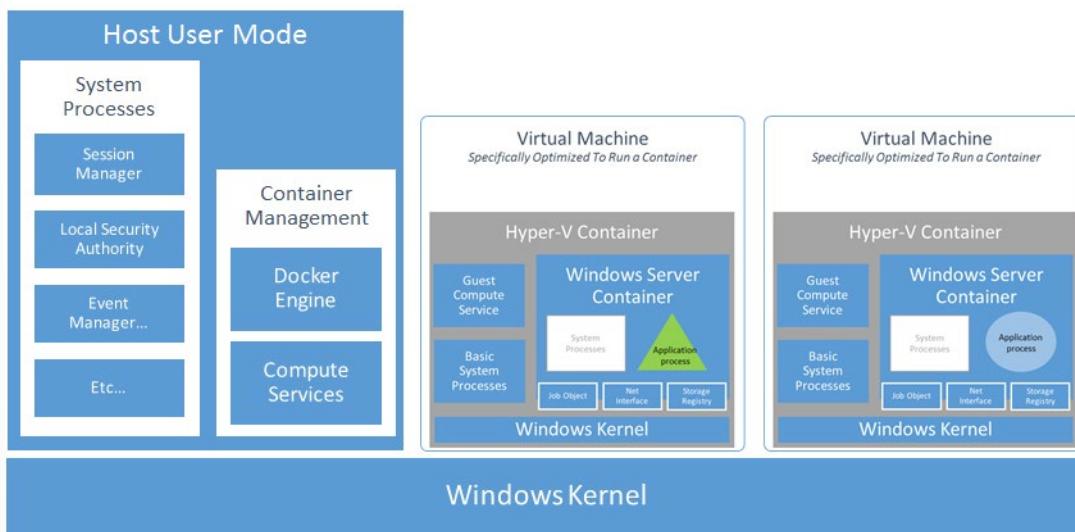
On Windows Server 2016 and Windows Server 2019, you can run these containers so that they share the Windows kernel. This method is quite efficient, and the processes that run in the container will have no performance effect on the container because they access the kernel objects without indirect action.

On Windows Server 2019, you can now run Windows and Linux containers alongside each other.

Hyper-V containers

When containers share the kernel and memory, it creates a slight chance that if a vulnerability occurs in the Windows operating system, an application might break out of its sandbox environment and inadvertently do something malicious. To avoid this, Windows provides a more secure alternative of running containers called *Hyper-V containers*.

The following diagram depicts the high-level architecture of Hyper-V containers on the Windows operating system. Hyper-V containers are supported on both Windows Server 2016 and newer versions, and on the Windows 10 Anniversary edition.



The main difference between Windows Server containers and Hyper-V containers is the isolation that the latter provides. Hyper-V containers are the only type of containers you can run on the Windows 10 operating system. Hyper-V containers have a small footprint and start fast compared to a full VM. You can run any image as a Hyper-V isolated container by using the **-isolation** option on the Docker command line, and specifying that the isolation be type **hyperv**. Refer to the following command for an example:

```
docker run -it --isolation hyperv microsoft/windowsservercore cmd
```

This command will run a new instance of a container based on the image `microsoft/windowsservercore`, and will run the command **cmd.exe** in interactive mode.

Nano Server

Nano Server is the headless deployment option for Windows Server 2016 and Windows Server 2019, available via the semi-annual channel releases. It is specifically optimized for private clouds and datacenters and for running cloud-based applications. It is intended to be run as a container in a container host, such as a Server Core installation of Windows Server.

It's a remotely administered server operating system optimized for private clouds and datacenters. It's similar to Windows Server in Server Core mode, but it's significantly smaller, has no local logon capability, and only supports 64-bit applications, tools, and agents.

Nano Server also takes up far less disk space, sets up significantly faster, and requires far fewer updates and restarts than Windows Server. When it does restart, it restarts much faster. Nano Server is ideal for a number of scenarios:

- As a compute host for Hyper-V VMs, either in clusters or not.
- As a storage host for Scale-Out File Server.
- As a Domain Name System (DNS) server
- As a web server running IIS
- As a host for applications that are developed using cloud application patterns and run in a container or VM guest operating system

Azure VMs and containers

It's also possible to run containers under and IaaS model in Azure using Azure VMs.

You can run the following containers:

- Windows Server or Windows Hyper-V containers
- Docker containers on Windows Server 2016 or Windows Server 2019 Nano servers
- Docker containers on Linux deployments.

You can use tools such as **cloud-init**¹¹ or the **custom script extension**¹² to install the Docker version of choice.

However, installing Docker does mean that when you need to deploy to multiple VMs in a load balanced infrastructure, you're dealing with infrastructure operations, VM OS patches, and infrastructure complexity for highly scalable applications. In production, for large scale or complex applications or for deployment models, this is not a best practice.

The main scenarios for using containers in an Azure VM are:

- Dev/test environment. A VM in the cloud is optimal for development and testing in the cloud. You can rapidly create or stop the environment depending on your needs.
- Small and medium scalability needs, In scenarios where you might need just a couple of VMs for your production environment, managing a small number of VMs might be affordable until you can move to more advanced platform as a service (PaaS) environments, such as Orchestrator.
- Production environments with existing deployment tools. You might be migrating from an on-premises environment that you have invested in tools to make complex deployments to VMs or bare-metal servers. To move to the cloud with minimal changes to production environment deployment procedures you could continue to use those tools to deploy to Azure VMs. However, you'll want to use Windows Containers as the unit of deployment to improve the deployment experience.

Automating IaaS Infrastructure

Azure VMs support a wide range of deployment and configuration management toolsets, including:

- Azure Resource Manager templates
- Scripting using bash, Azure CLI and PowerShell
- Windows PowerShell Desired State Configuration (DSC) or Azure Automation DSC
- Ansible
- Chef
- Puppet
- Terraform

VM extensions give your VM additional capabilities through post-deployment configuration and automated tasks. These following common tasks can be accomplished using extensions such as:

- Run custom scripts. The Custom Script Extension (CSE) helps you configure VM workloads by running your script when the VM is provisioned.

¹¹ <https://docs.microsoft.com/en-us/azure/virtual-machines/linux/tutorial-automate-vm-deployment>

¹² <https://docs.microsoft.com/en-us/azure/virtual-machines/extensions/custom-script-linux>

- Deploy and manage configurations. The PowerShell DSC extension helps you set up DSC on a VM to manage configurations and environments.
- Collect diagnostics data. The Azure Diagnostics extension helps you configure the VM to collect diagnostics data you can use to monitor the health of your applications.

You can read more about vm extensions at **Virtual machine extensions and features for Linux¹³**, and **Virtual machine extensions and features for Windows¹⁴**.

Azure DevTest Labs

Azure DevTest Labs enables you to quickly set up an environment for your team (for example: development or test environment) in the cloud. A lab owner creates a lab, provisions Windows VMs or Linux VMs, installs the necessary software and tools, and makes them available to lab users. Lab users connect to the VMs in the lab, and use them for their day-to-day short-term projects. Once users start utilizing resources in the lab, a lab admin can analyze cost and usage across multiple labs, and set overarching policies to optimize their organization's or team's costs.



✓ Note: *Azure DevTest Labs* is being expanded with new types of labs, namely *Azure Lab Services*. *Azure Lab Services* lets you create managed labs, such as classroom labs. The service itself handles all the infrastructure management for a managed lab, from spinning up VMs to handling errors, and scaling the infrastructure. The managed labs are currently in preview, at the time of writing. Once the preview ends, the new lab types and existing DevTest Labs come under the new common umbrella name of *Azure Lab Services* where all lab types continue to evolve.

Usage scenarios

Some common use cases for using *Azure DevTest Labs* are as follows:

- Use DevTest Labs for development environments. This enables you to host development machines for developers so they can:
 - Quickly provision their development machines on demand.
 - Provision Windows and Linux environments using reusable templates and artifacts.
 - More easily customize their development machines whenever needed.
- Use DevTest Labs for test environments. This enables you to host machines for testers so they can:
 - Test the latest version of their application by quickly provisioning Windows and Linux environments using reusable templates and artifacts.
 - Scale up their load testing by provisioning multiple test agents.

In addition, administrators can use DevTest Labs to control costs by ensuring that testers cannot get more VMs than they need for testing, and VMs are shut down when not in use.

¹³ <https://docs.microsoft.com/en-us/azure/virtual-machines/extensions/features-linux>

¹⁴ <https://docs.microsoft.com/en-us/azure/virtual-machines/extensions/features-windows>

- Integrate DevTest Labs with Azure DevOps CI/CD pipeline. You can use the Azure DevTest Labs Tasks extension that's installed in Azure DevOps to easily integrate your CI/CD build-and-release pipeline with Azure DevTest Labs. The extension installs three tasks:

- Create a VM
- Create a custom image from a VM
- Delete a VM

The process makes it easy to, for example, quickly deploy an image for a specific test task, and then delete it when the test completes.

For more information on DevTest Labs, go to **Azure Lab Services Documentation**¹⁵.

¹⁵ <https://docs.microsoft.com/en-us/azure/lab-services/>

Azure Platform-as-a-Service (PaaS) Services

Azure App Service

Azure App Service is a PaaS offering on Azure for hosting web applications, REST APIs, and mobile backends.

Web App, a component of Azure App Services, is a fully-managed compute platform that is optimized for hosting websites and web applications. You can build applications and run them natively in Windows or Linux environments, and many languages are supported for building your application.

Supported Languages include:

- Node.js
- Java
- PHP
- Python (Preview)
- .NET
- .NET Core
- Ruby

As a PaaS offering, Azure App services offers services such as security, load balancing, autoscaling, and automated management. You can also utilize its DevOps capabilities, such as continuous deployment from Azure DevOps, GitHub, Docker Hub, and other sources. Other capabilities you can utilize include package management, staging environments, custom domain, and Secure Sockets Layer (SSL) certificates.

Why use App Service?

App Service provides you with some key features, including:

- Multiple languages and frameworks. App Service supports many languages as listed above, and allows you to run PowerShell and other scripts or executables as background services.
- DevOps optimization. You can set up continuous integration and deployment using either Azure DevOps, GitHub, BitBucket, Docker Hub, or Azure Container Registry. You also can promote updates through test and staging environments, and manage your apps in App Service by using Azure PowerShell or the cross-platform command-line interface (CLI).
- Global scale with high availability. Using App Service, you can scale up or out manually or automatically, and host your apps anywhere in Microsoft's global datacenter infrastructure. In addition, the App Service service-level agreement (SLA) ensures high availability.
- Connections to SaaS platforms and on-premises data. Choose from more than 50 connectors for different enterprise systems.
- Security and compliance. App Service is ISO, Service Organization Controls (SOC), and Payment Card Industry (PCI) compliant. Users can authenticate with Azure Active Directory (Azure AD), or with social media accounts such as Google, Facebook, Twitter, and Microsoft. Create IP address restrictions and manage service identities.
- Application templates. Choose from an extensive list of application templates in the Azure Marketplace, such as WordPress, Joomla, and Drupal.

- Microsoft Visual Studio integration. Dedicated tools in Visual Studio streamline the work of creating, deploying, and debugging.
- API and mobile features. App Service provides turn-key CORS support for RESTful API scenarios, and simplifies mobile app scenarios by enabling authentication, offline data sync, push notifications, and more.
- Serverless code. Run a code snippet or script on-demand without having to explicitly provision or manage infrastructure. Pay only for the compute time your code actually uses.

More general details are available on the [App Service Documentation¹⁶](#) page.

App Service plans

In App Service, an app runs in an *App Service plan*. An *App Service plan* defines a set of compute resources for a web app to run. These compute resources are analogous to the server farm in conventional web hosting. You can configure one or more apps to run on the same computing resources, or in the same 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 (such as West US, East US)
- Number of VM instances
- Size of VM instances (small, medium, and large)
- Pricing tier (Free, Shared, Basic, Standard, Premium, PremiumV2, Isolated, Consumption)

App service plans - pricing tiers

The pricing tier for an App Service plan determines what App Service features you can use, and how much you pay for the plan. Pricing tiers are:

- Shared compute. Shared compute has two base tiers, Free, and Shared. They both run an app on the same Azure VM as other App Service apps, including apps of other customers. These tiers allocate CPU quotas to each app that runs on the shared resources, and the resources cannot scale out.
- Dedicated compute. The Dedicated compute Basic, Standard, Premium, and PremiumV2 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 for scale-out.
- Isolated. This tier runs dedicated Azure VMs on dedicated Azure virtual networks. This provides network isolation (on top of compute isolation) to your apps. It also provides the maximum scale-out capabilities.
- Consumption. This tier is only available to function apps. It scales the functions dynamically depending on workload.

More detail about the App Service plans are available on the [Azure App Service plan overview¹⁷](#) webpage.

¹⁶ <https://docs.microsoft.com/en-us/azure/app-service/>

¹⁷ <https://docs.microsoft.com/en-us/azure/app-service/overview-hosting-plans?toc=%2fazure%2fapp-service%2fcontainers%2ftoc.json>

Demonstration-Create a Java app in App Service on Linux

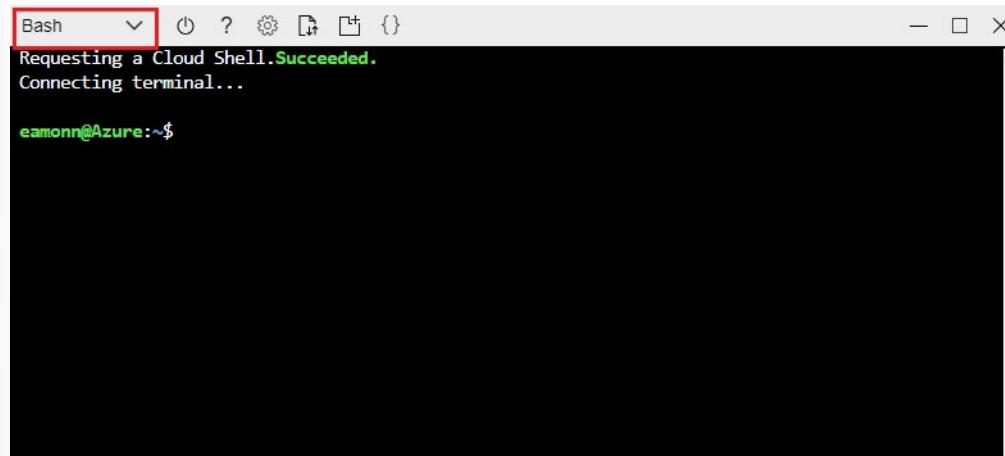
This walkthrough shows how to use the Azure CLI with the Maven Plugin for Azure Web Apps (Preview) to deploy a Java Web archive file.

Prerequisites

- You require an Azure subscription to perform the following steps. If you don't have one you can create one by following the steps outlined on the [Create your Azure free account today¹⁸](#) webpage.

Steps:

1. Open Azure Cloud Shell by going to <https://shell.azure.com¹⁹>, or by using the Azure Portal, and select **Bash** as the environment option.



2. Create a Java app by executing the Maven command in the Cloud Shell prompt to create a new app named `helloworld`: Accept the default values as you go:

```
mvn archetype:generate -DgroupId=example.demo -DartifactId=helloworld  
-DarchetypeArtifactId=maven-archetype-webapp
```

¹⁸ https://azure.microsoft.com/en-us/free/?ref=microsoft.com&utm_source=microsoft.com&utm_medium=docs&utm_campaign=visualstudio

¹⁹ <https://shell.azure.com>

```
Bash    v | ⌂ ? ⌂ ⌂ ⌂ {}  
Requesting a Cloud Shell.Succeeded.  
Connecting terminal...  
  
eamonn@Azure:~$ mvn archetype:generate -DgroupId=example.demo -DartifactId=helloworld -DarchetypeArtifactId=maven-archetype-webapp  
[INFO] Scanning for projects...  
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-clean-plugin/2.5/maven-clean-plugin-2.5.pom  
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-clean-plugin/2.5/maven-clean-plugin-2.5.pom (4 KB at 5.2 KB/sec)  
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-plugins/22/maven-plugins-22.pom  
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-plugins/22/maven-plugins-22.pom (13 KB at 326.5 KB/sec)  
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/maven-parent/21/maven-parent-21.pom  
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/maven-parent/21/maven-parent-21.pom
```

3. Select the braces icon in Cloud Shell to open the editor. Use this code editor to open the project file **pom.xml** in the **helloworld** directory.

4. Add the following plugin definition inside the `<build>` element of the `pom.xml` file:

```
<plugins>
<!-- **** Deploy to Tomcat in App Service Linux -->
<!-- Deploy to Tomcat in App Service Linux -->
<!-- **** -->

<plugin>
    <groupId>com.microsoft.azure</groupId>
    <artifactId>azure-webapp-maven-plugin</artifactId>
    <version>1.4.0</version>
    <configuration>

        <!-- App information -->
        <resourceGroup>${RESOURCEGROUP_NAME}</resourceGroup>
        <appName>${WEBAPP_NAME}</appName>
        <region>${REGION}</region>

        <!-- Java Runtime Stack for App on Linux-->
        <linuxRuntime>tomcat 8.5-jre8</linuxRuntime>

    </configuration>
</plugin>
</plugins>
```

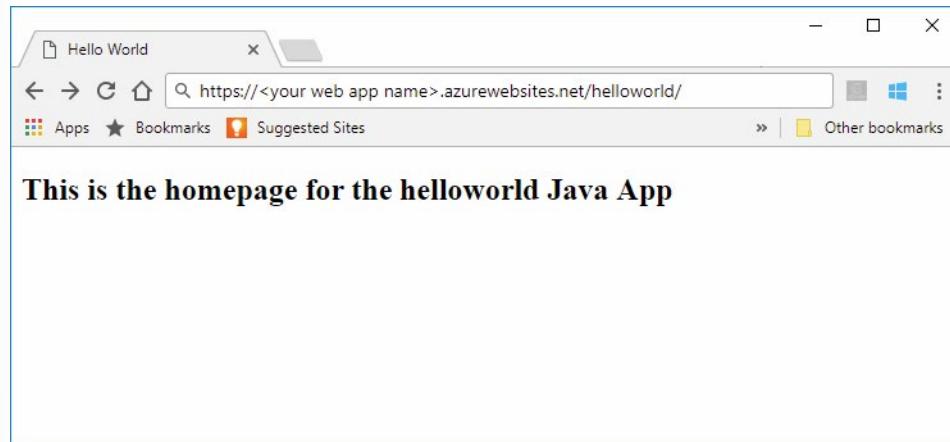
5. Update the following placeholders in the plugin configuration:

- RESOURCEGROUP_NAME (can be any name)
 - WEBAPP_NAME (must be a unique name)
 - REGION (your nearest datacenter location. For example, westus)

6. Deploy your Java app to Azure using the following command:

```
mvn package azure-webapp:deploy
```

7. After this step completes, verify deployment by opening the deployed application using the following URL in your web browser, replacing `webapp` with the name of the deployed application. For example, `http://<webapp>.azurewebsites.net/helloworld`.



Demonstration-Deploy a .NET Core based app

This walkthrough shows how to create an ASP.NET Core web app, and then deploy it to Azure App Services.

Prerequisites

You require the following items to complete these walkthrough steps:

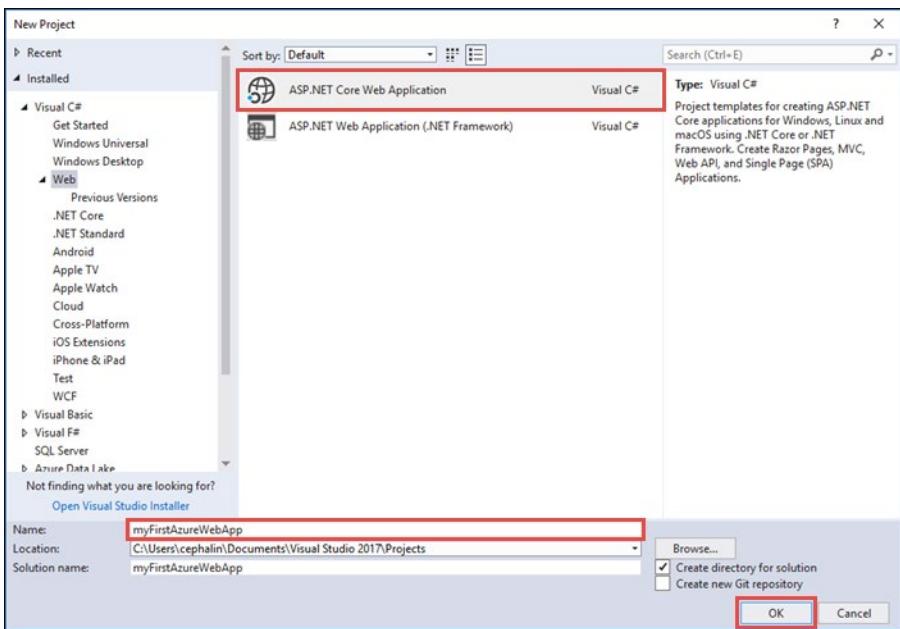
- Visual Studio 2017. If you don't have Visual Studio 2017, you can install the Visual Studio Community edition from the [Visual Studio downloads²⁰](#) webpage.
- An Azure subscription. If you don't have one you can create one by following the steps outlined on the [Create your Azure free account today²¹](#) webpage.

Steps

1. In Visual Studio, create a project by selecting **File**, **New**, and then **Project**.
2. In the **New Project** dialog, select **Visual C#, Web**, and then **ASP.NET Core Web Application**.
3. Name the application **myFirstAzureWebApp**, and then select **OK**.

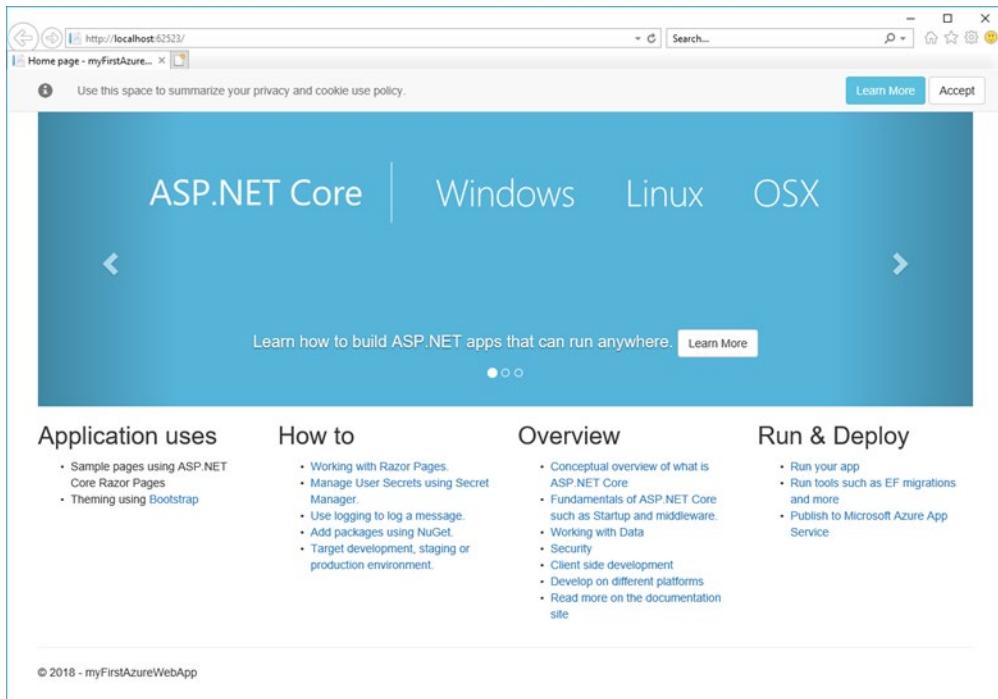
²⁰ <https://visualstudio.microsoft.com/downloads/>

²¹ https://azure.microsoft.com/en-us/free/?ref=microsoft.com&utm_source=microsoft.com&utm_medium=docs&utm_campaign=visualstudio

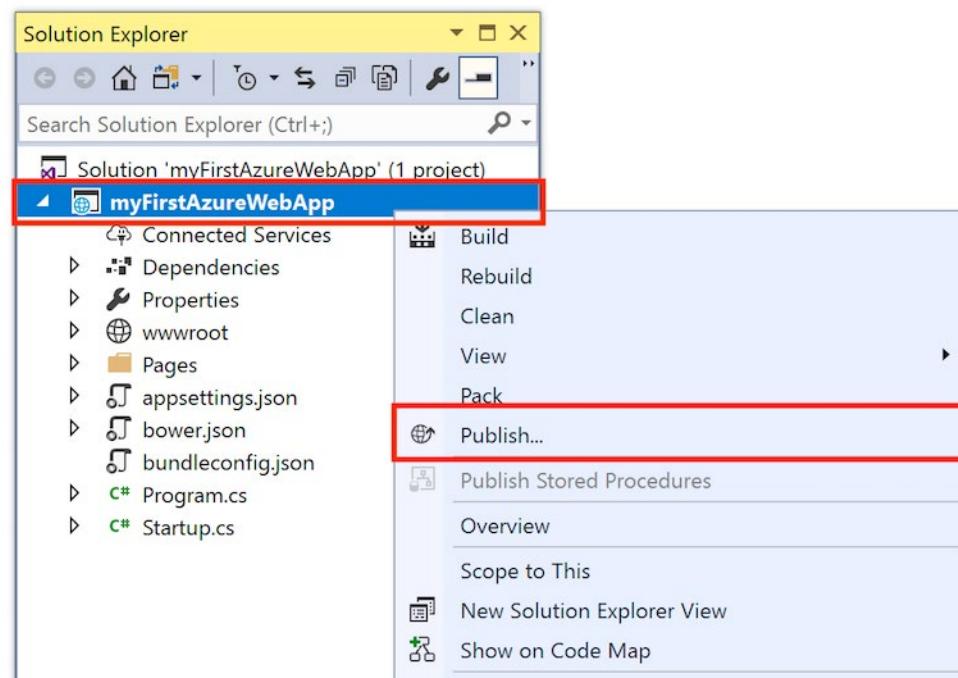


You can deploy any type of ASP.NET Core web app to Azure. For this walkthrough, select the **Web Application** template. Ensure authentication is set to **No Authentication** and no other option is selected, and then Select **OK**.

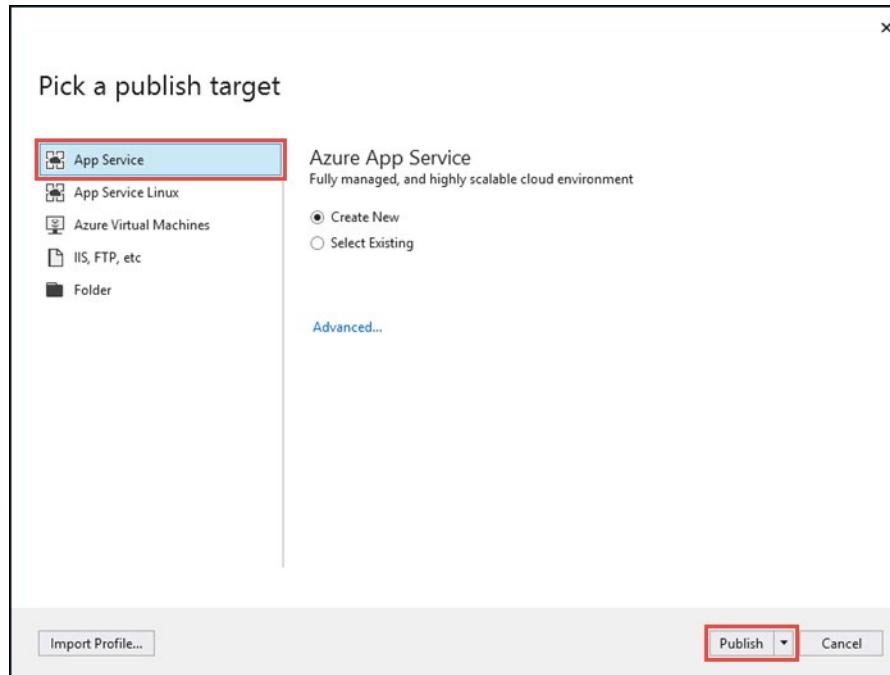
4. To run the web app locally. from the menu, select **Debug, Start**, without Debugging.



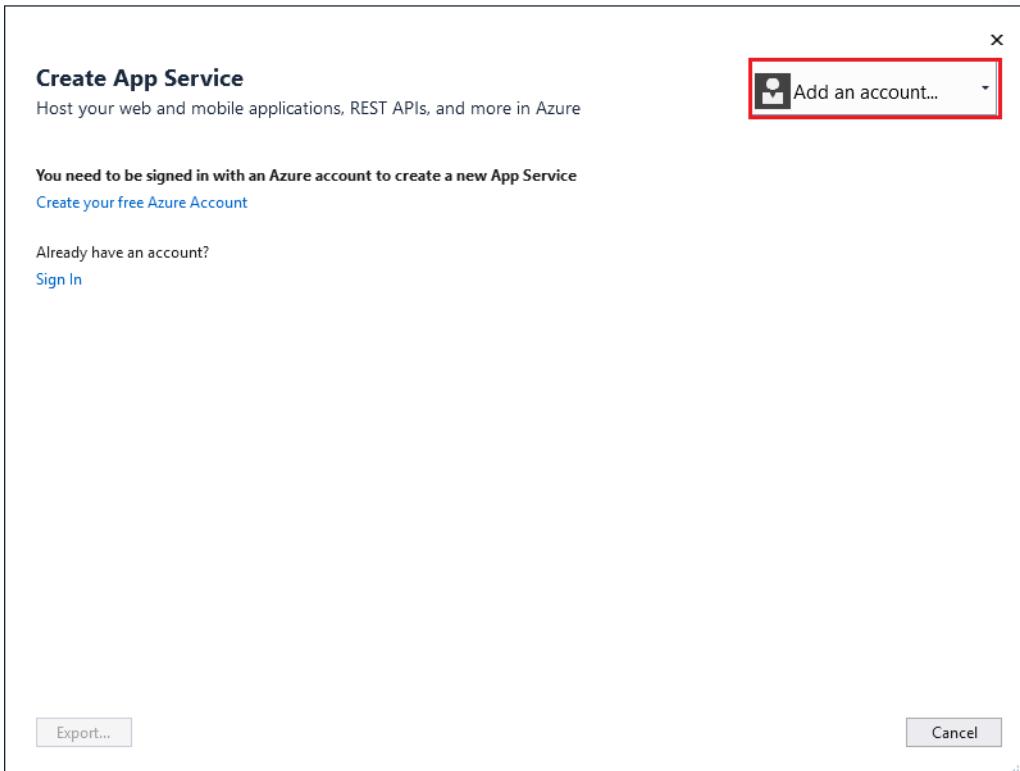
5. Launch the **Publish wizard** by going to **Solution Explorer**, right-clicking the **myFirstAzureWebApp** project, and then selecting **Publish**.



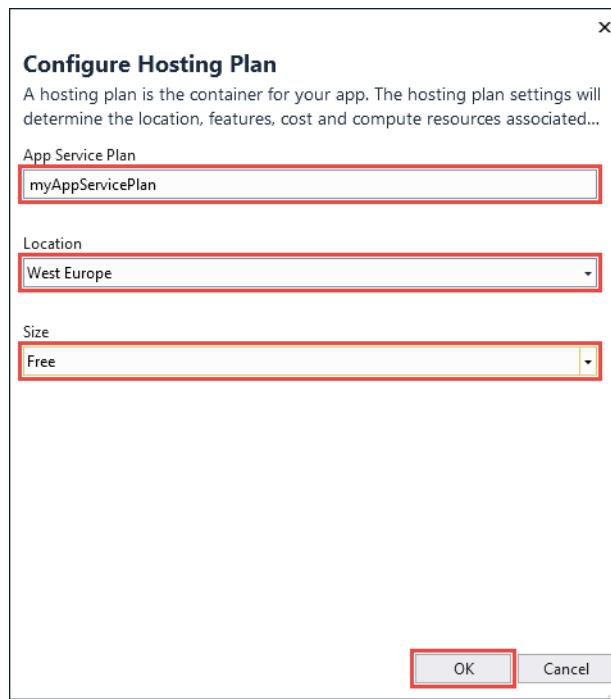
6. To open the **Create App Service** dialog, select **App Service**, and then select **Publish**.



7. Sign in to Azure by going to the **Create App Service** dialog, select **Add an account...**, and sign in to your Azure subscription. If you're already signed in, select the account you want from the drop-down. If you're already signed in, don't select the **Create** button.



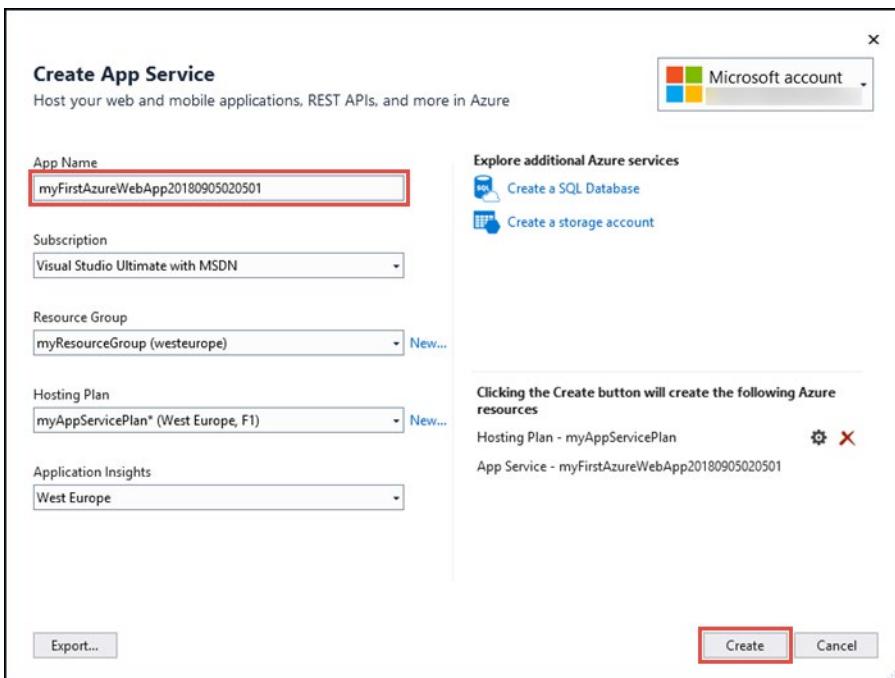
8. Next to **Resource Group**, select **New**. Name the resource group **myResourceGroup**, and then select **OK**.
9. Next to Hosting Plan, select **New**. Use the following values, and then select **OK**
 - App Service Plan: **myappserviceplan**
 - Location: **your nearest datacenter**
 - Size: **Free**



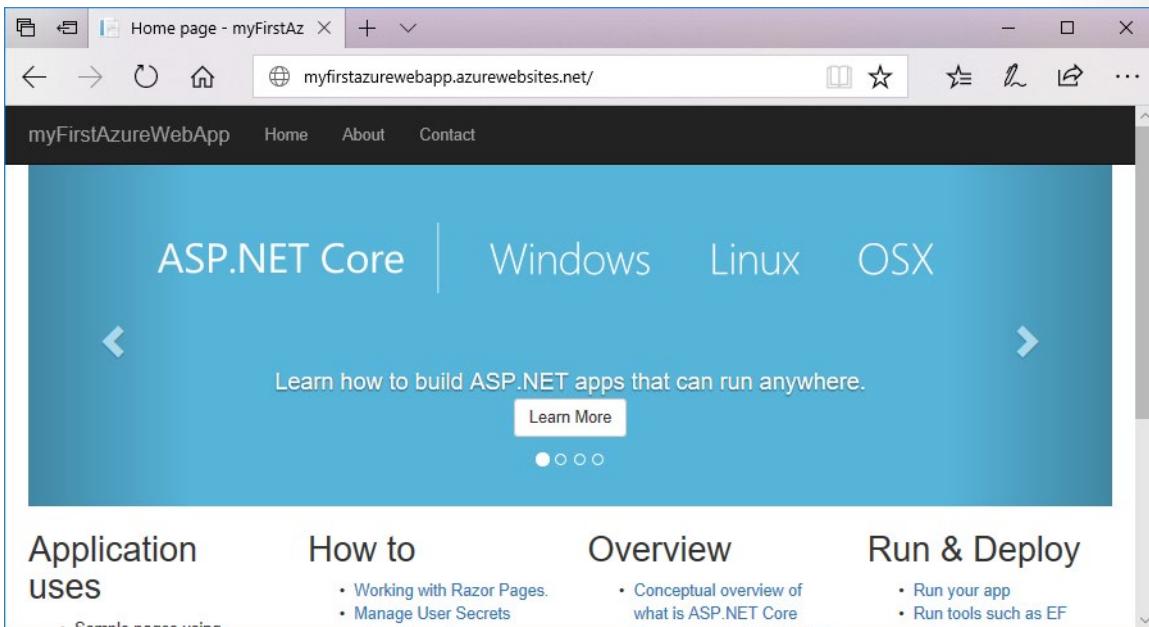
- ✓ Note: An App Service plan specifies the location, size, and features of the web server farm that hosts your app. You can save money when hosting multiple apps by configuring the web apps to share a single App Service plan"
- Region (for example: North Europe, East US, or Southeast Asia)
 - Instance size (small, medium, or large)
 - Scale count (1 to 20 instances)
 - SKU (Free, Shared, Basic, Standard, or Premium)

10. While still in the **Create App Service** dialog, enter a value for the app name, and then select **Create**.

- ✓ Note: The app name must be a unique value. Valid characters are a-z, 0-9, and -. Alternatively, you can accept the automatically-generated unique name. The resulting URL of the web app is `http://app_name.azurewebsites.net`, where *app_name* is your app's name.



11. After the wizard completes, it publishes the ASP.NET Core web app to Azure, and then launches the app in the default browser.



The app name specified in the create and publish step is used as the URL prefix in the format `http://<app_name>.azurewebsites.net`.

Congratulations, your ASP.NET Core web app is running live in Azure App Service.

- ✓ Note: If you do not plan on using the resources, you should delete them to avoid incurring charges.

Scale App Services

Scaling ensures that you have the right amount of resources running to manage an application's needs. Scaling can be done manually or automatically.

There are two workflows for scaling Azure App services:

- Scale up. Add additional resources to your app such as more CPU, memory, disk space. You can also add extra features such as dedicated VMs, custom domains and certificates, staging slots, and autoscaling.
 - To scale up, you will need to change the pricing tier of your App Service plan.
- Scale out. Increase the number of VM instances that run your app.
 - You can scale out to as many as 20 instances, depending on your pricing tier.
 - App Service environments in Isolated tier further increases your scale-out count to 100 instances.

Note: The scale settings take seconds to apply and affect all apps in your App Service plan. They don't require you to change your code or redeploy your application.

You can also scale based on a set of predefined rules or schedule, and configure webhooks and email notifications and alerts.

Autoscale

Autoscale settings help ensure that you have the right amount of resources running to manage the fluctuating load of your application. You can configure Autoscale settings to trigger based on metrics that indicate load or performance, or at a scheduled date and time.

Metric

You can scale based on a resource metric, such as:

- Scale based on CPU. You want to scale out or scale in based on a percentage CPU value.
- Scale based on custom metric. To scale based on a custom metric, you designate a specific metric that is relevant to your app architecture. For example, you might have a web front end and an API tier that communicates with the backend, and you want to scale the API tier based on custom events in the front end.

Schedule

You can scale based on a schedule as well. For example, you can:

- Scale differently on weekdays Vs weekends. You don't expect traffic on weekends, hence you want to scale down to 1 instance on weekends.
- Scale differently during holidays. During holidays or specific days that are important for your business, you might want to override the defaults scaling settings and have more capacity at your disposal.

Autoscale profiles

There are three types of Autoscale profiles that you can configure depending on what you want to achieve. Azure then evaluates which profile to execute at any given time. The profile types are:

- Regular profile. This is the most common profile. If you don't need to scale your resource based on the day of the week, or on a particular day, you can use a regular profile.
- Fixed date profile. This profile is for special cases. For example, let's say you have an important event coming up on December 26, 2019 (PST). You want the minimum and maximum capacities of your resource to be different on that day, but still scale on the same metrics.
- Recurrence profile. This type of profile enables you to ensure that this profile is always used on a particular day of the week. Recurrence profiles only have a start time. They run until the next recurrence profile or fixed date profile is set to start.

Example

An example of how this looks in an Azure Resource Manager template is an Autoscale setting with one profile, as below:

- There are two metric rules in this profile: one for scale out, and one for scale in.
 - The scale-out rule is triggered when the VM scale set's average percentage CPU metric is greater than 85 percent for the past 10 minutes.
 - The scale-in rule is triggered when the VM scale set's average is less than 60 percent for the past minute.

```
{  
    "id": "/subscriptions/s1/resourceGroups/rg1/providers/microsoft.insights/  
autoscalesettings/setting1",  
    "name": "setting1",  
    "type": "Microsoft.Insights/autoscaleSettings",  
    "location": "East US",  
    "properties": {  
        "enabled": true,  
        "targetResourceUri": "/subscriptions/s1/resourceGroups/rg1/providers/  
Microsoft.Compute/virtualMachineScaleSets/vmss1",  
        "profiles": [  
            {  
                "name": "mainProfile",  
                "capacity": {  
                    "minimum": "1",  
                    "maximum": "4",  
                    "default": "1"  
                },  
                "rules": [  
                    {  
                        "metricTrigger": {  
                            "metricName": "Percentage CPU",  
                            "metricResourceUri": "/subscriptions/s1/resourceGroups/rg1/  
providers/Microsoft.Compute/virtualMachineScaleSets/vmss1",  
                            "timeGrain": "PT1M",  
                            "operator": "GreaterThanOrEqual",  
                            "threshold": 85  
                        },  
                        "scaleAction": {  
                            "direction": "Out",  
                            "type": "ChangeCount",  
                            "count": 1  
                        }  
                    },  
                    {  
                        "metricTrigger": {  
                            "metricName": "Percentage CPU",  
                            "metricResourceUri": "/subscriptions/s1/resourceGroups/rg1/  
providers/Microsoft.Compute/virtualMachineScaleSets/vmss1",  
                            "timeGrain": "PT1M",  
                            "operator": "LessThan",  
                            "threshold": 60  
                        },  
                        "scaleAction": {  
                            "direction": "In",  
                            "type": "ChangeCount",  
                            "count": 1  
                        }  
                    }  
                ]  
            }  
        ]  
    }  
}
```

```
        "statistic": "Average",
        "timeWindow": "PT10M",
        "timeAggregation": "Average",
        "operator": "GreaterThan",
        "threshold": 85
    },
    "scaleAction": {
        "direction": "Increase",
        "type": "ChangeCount",
        "value": "1",
        "cooldown": "PT5M"
    }
},
{
    "metricTrigger": {
        "metricName": "Percentage CPU",
        "metricResourceUri": "/subscriptions/s1/resourceGroups/rg1/providers/Microsoft.Compute/virtualMachineScaleSets/vmss1",
        "timeGrain": "PT1M",
        "statistic": "Average",
        "timeWindow": "PT10M",
        "timeAggregation": "Average",
        "operator": "LessThan",
        "threshold": 60
    },
    "scaleAction": {
        "direction": "Decrease",
        "type": "ChangeCount",
        "value": "1",
        "cooldown": "PT5M"
    }
}
]
}
]
```

You can review some best auto-scale best practices on the [Best practices for Autoscale²²](#) page.

Web App for containers

Azure App Service also supports container deployment. The service container hosting service offering in Azure App Service is often referred to as *Web App for Containers*.

Web App for Containers allows customers to use their own containers and deploy them to App Service as a web app. Similar to the Web App solution, Web App for Containers eliminates time-consuming infra-

²² <https://docs.microsoft.com/en-us/azure/azure-monitor/platform/autoscale-best-practices>

structure management tasks during container deployment, updating, and scaling. This enables developers to focus on coding, and getting their apps in front of end users faster.

To boost developer productivity, Web App for Containers also provides:

- Integrated CI/CD capabilities with Docker Hub
- Azure Container Registry
- Visual Studio Team Services (VSTS)
- Built-in staging
- Rollback
- Testing-in-production
- Monitoring
- Performance testing capabilities

For Operations, Web App for Containers also provides rich configuration features that enable developers to more easily add custom domains, integrate with Azure AD authentication, add SSL certificates, and more. These are all crucial to web app development and management. Web App for Containers is an ideal environment to run web apps that do not require extensive infrastructure control.

Web App for Containers supports both Linux and Windows containers. It also provides the following features:

- Deploy containerized applications using Docker Hub, Azure Container Registry, or private registries.
- Incrementally deploy apps into production with deployment slots and slot swaps to allow for blue/green deployments (also known as A/B deployments).
- Scale out automatically with auto-scale.
- Enable application logs, and use the App Service Log Streaming feature to see logs from your application.
- Use PowerShell and Windows Remote Management (WinRM) to remotely connect directly into your containers.

Demonstration-Deploy custom Docker image to Web App for containers

In this walkthrough you will deploy a custom Docker image running a Go application to Web App for Containers.

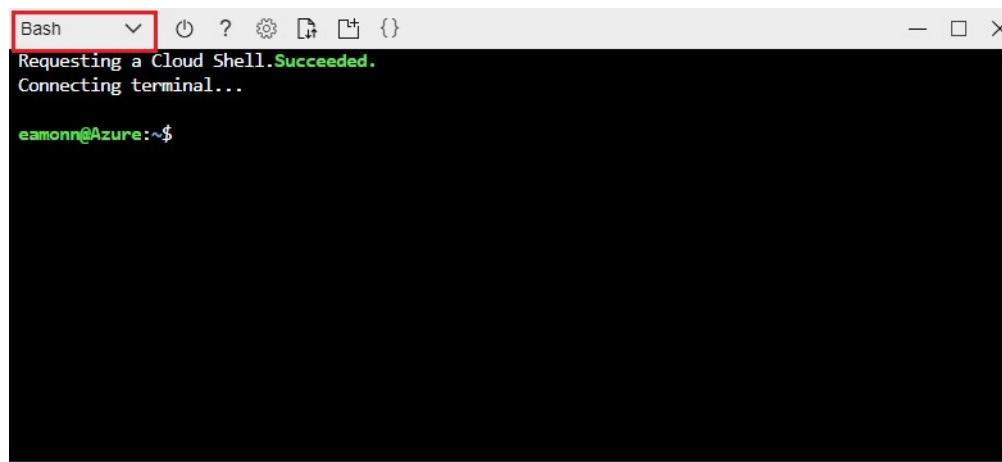
Prerequisites

- You require need an Azure subscription to perform these steps. If you don't have one you can create one by following the steps outlined on the [Create your Azure free account today²³](#) webpage.

Steps

1. Open Azure Cloud Shell by going to <https://shell.azure.com>, or by using the Azure portal and selecting **Bash** as the environment option.

²³ https://azure.microsoft.com/en-us/free/?ref=microsoft.com&utm_source=microsoft.com&utm_medium=docs&utm_campaign=visualstudio



2. Run the following command to configure a deployment user, replacing <username> and <password> (including brackets) with a new user name and password. The user name must be unique within Azure, and the password must be at least eight characters long with two of the following three elements: letters, numbers, symbols:

Note: This deployment user is required for FTP and local Git deployment to a web app. The user name and password are account level. They are different from your Azure subscription credentials.

```
az webapp deployment user set --user-name <*username*> --password <*password*>
```

You should get a JSON output, with the password shown as null. If you get a 'Conflict'. Details: 409 error, change the username. If you get a 'Bad Request'. Details: 400 error, use a stronger password.

3. Create a resource group in Azure by using the following command and substituting the resource group name value for one of your own choice, and the location to a datacenter near you:

```
az group create --name myResourceGroup --location "West Europe"
```

4. Create an Azure App Service plan by running the following command, which creates an App Service plan named *myAppServicePlan* in the *Basic* pricing tier:

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku B1 --is-linux
```

5. Run the following command to create a web app in your App service plan, replacing <app name> with a globally unique name, and the resource group and App Service plan names to values you created earlier. This command points to the public Docker Hub image:

```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name < app name > --deployment-container-image-name microsoft/azure-appservices-go-quickstart
```

When the web app has been created, the Azure CLI shows output similar to the following example:

```
{  
    "availabilityState": "Normal",  
    "clientAffinityEnabled": true,  
    "clientCertEnabled": false,  
    "cloningInfo": null,  
    "containerSize": 0,  
    "dailyMemoryTimeQuota": 0,  
    "defaultHostName": "<app name>.azurewebsites.net",  
    "deploymentLocalGitUrl": "https://<username>@<app name>.scm.azureweb-  
sites.net/<app name>.git",  
    "enabled": true,  
    < JSON data removed for brevity. >  
}
```

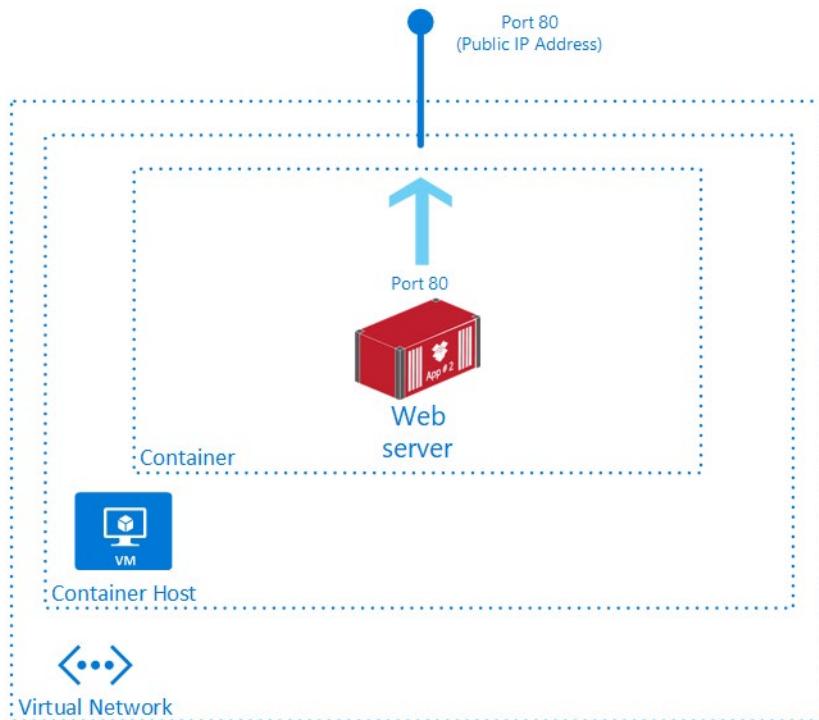
6. Browse to the app http://<app_name>.azurewebsites.net/hello.

Congratulations! You've deployed a custom Docker image running a Go application to Web App for Containers.

Azure Container Instances

Azure Container Instances is a PaaS service on Azure that offers the capability to run both Linux and Windows containers. By default, Azure Container Instances runs single containers, which means that individual containers are isolated from each other and cannot interact with one another.

Azure Container Instances offers the one of the fastest and simplest way to run a container in Azure, without having to provision any virtual machines and without having to adopt a higher-level service.



Eliminate VM management

With Azure Container Instances you don't need to own a VM to run your containers. This means that you don't need to worry about creating, managing, and scaling them. In the following picture, the network, virtual machine, and container host are entirely managed for you. However, this also means you have no control over them.

Hypervisor-level security

Historically, containers have offered application dependency isolation and resource governance. However, they have not been considered sufficiently hardened for hostile multi-tenant usage. In Azure Container Instances, your application is as isolated in a container as it would be in a VM.

The isolation between individual containers is achieved by using Hyper-V containers.

Public IP connectivity and DNS name

Azure Container Instances enables exposing your containers directly to the internet with an IP address and a fully qualified domain name (FQDN). When you create a container instance you can specify a custom Domain Name System (DNS) name label so your application is reachable at `customlabel.azureregion.azurecontainer.io`.

By assigning a public IP address to your container, you make it accessible from the outside world. If a container that exposes port 80, as in the previous image, the port is connected to a public IP address that accepts traffic on port 80 of the virtual host.

- ✓ Note: Port mappings are not available in Azure Container Instances, so the both the container and container host must use the same port number.

Custom sizes and resources

Containers are typically optimized to run a single application, but the exact needs of applications can differ greatly. Azure Container Instances provides optimum utilization by allowing exact specifications of CPU cores and memory.

You specify the amount of memory in gigabytes for each container, and the CPUs to assign. For compute-intensive jobs such as machine learning, Azure Container Instances can schedule Linux containers to use NVIDIA Tesla GPU resources (preview).

Because you pay based only on what you need and get billed by the second, you can fine-tune your spending based on actual need and not pay for resources you don't need or use.

Virtual network deployment (preview)

Currently in preview, this feature of Azure Container Instances enables you to deploy container instances to an Azure virtual network. By deploying container instances into a subnet within your virtual network, they can communicate securely with other resources in the virtual network, including those that are on-premises (through a virtual private network (VPN) gateway or ExpressRoute).

Persistent storage

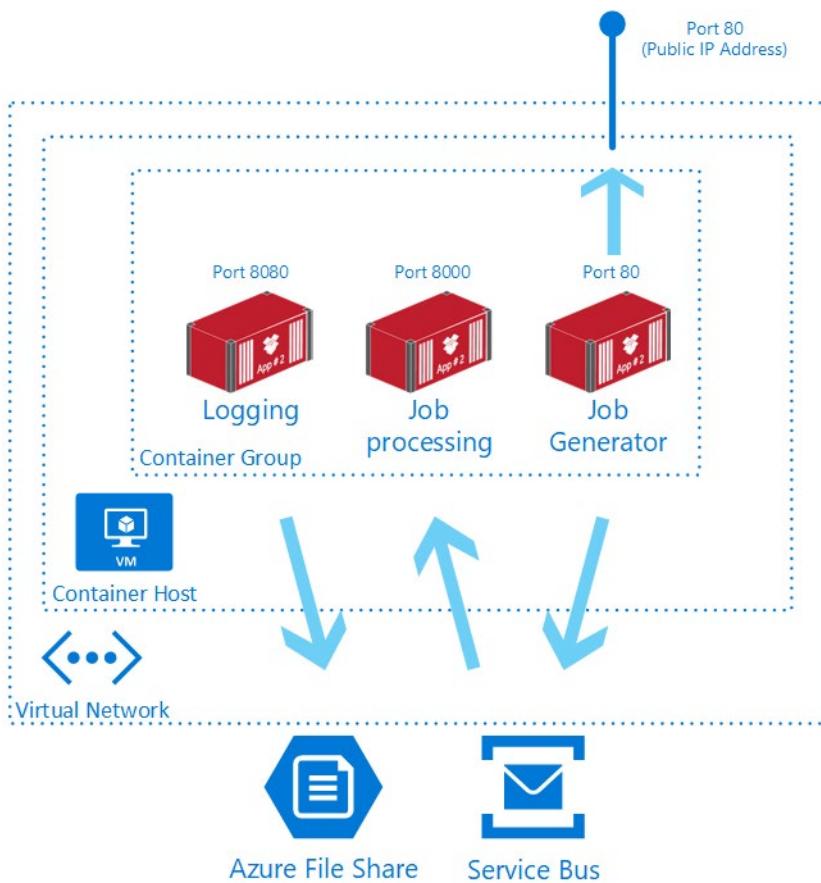
To retrieve and persist state with Azure Container Instances, use Azure Files shares.

It is possible to run both long-running processes and task-based containers. This is controlled by the container restart policy.

Container groups

By default, containers are isolated from each other. But what if you need interaction between containers? To support this kind of scenario, there is the concept of container groups. Containers inside a container group are deployed on the same machine, and they use the same network. They also share their lifecycle, meaning all containers in the group are started and stopped together.

Containers are always part of a container group. Even if you deploy a single container, it will be placed into a new group automatically. When using Windows containers, a group can have only one container. This is because network namespaces are not available on the Windows operating system.



Usage scenario

Azure Container Instances is a recommended compute option for any scenario that can operate in isolated containers, such as simple applications, task automation, and build jobs. For scenarios requiring full container orchestration (including service discovery across multiple containers, automatic scaling, and coordinated application upgrades) we recommend Azure Kubernetes Service (AKS).

Demonstration-Create a container on ACI

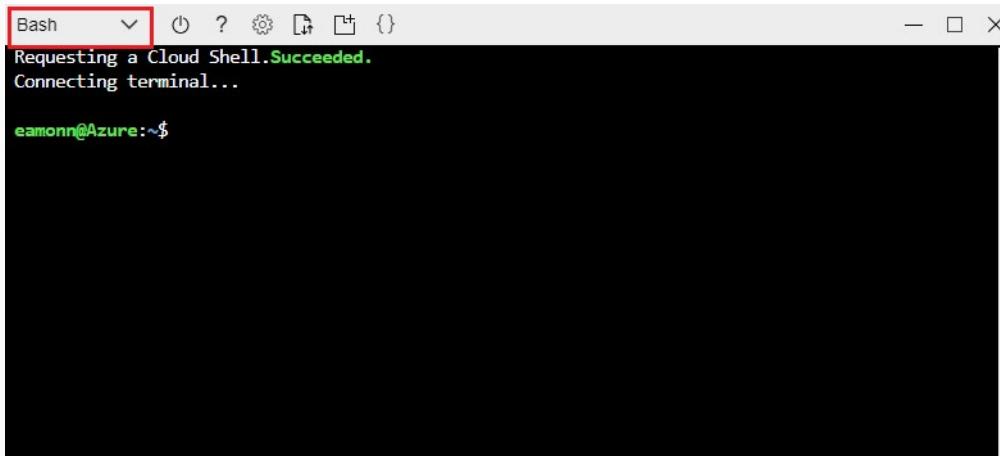
This walkthrough demonstrates how to use Azure CLI to create a container in Azure and make its application available with an FQDN. A few seconds after you execute a single deployment command, you can browse to the running application:

Prerequisites

- You require an Azure subscription to perform these steps. If you don't have one you can create one by following the steps outlined on the [Create your Azure free account today²⁴](#) webpage.

Steps

1. Open Azure Cloud Shell by going to <https://shell.azure.com>, or using the Azure portal and selecting **Bash** as the environment option



Note: You can use a local version on Azure CLI if you want, but it must be version 2.0.27 or later.

2. Create a resource group using the following command, substituting your values for the resource group name and location:

```
az group create --name myResourceGroup --location eastus
```

3. Create a container, substituting your values for the resource group name and container name. Ensure it is a unique DNS name. In the following command, we specify to open port 80 and apply a DNS name on the container:

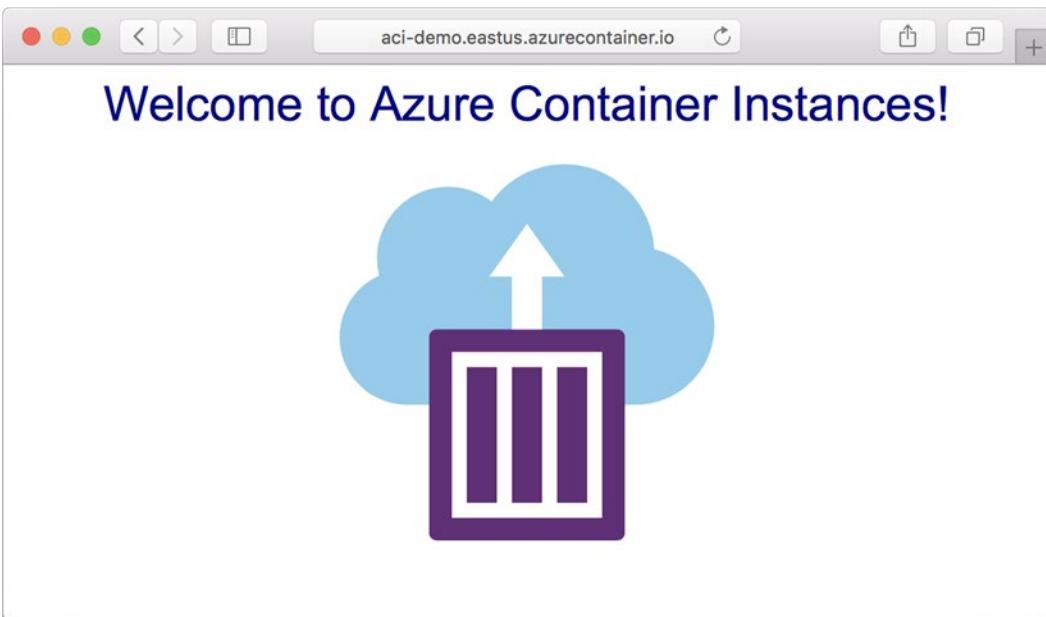
```
az container create --resource-group myResourceGroup --name mycontainer  
--image microsoft/aci-helloworld --dns-name-label aci-demo --ports 80
```

4. Verify the container status by running the following command, again substituting your values where appropriate:

²⁴ https://azure.microsoft.com/en-us/free/?ref=microsoft.com&utm_source=microsoft.com&utm_medium=docs&utm_campaign=visualstudio

```
az container show --resource-group myResourceGroup --name mycontainer  
--query "{FQDN:ipAddress.fqdn, ProvisioningState:provisioningState}" --out  
table
```

5. If the container's *ProvisioningState* is **Succeeded**, navigate to its FQDN in your browser. If you see a webpage similar to the follow example, congratulations! You've successfully deployed an application running in a Docker container to Azure.



- ✓ Note: You can check the container in the portal if you want. If you are finished using the resources in Azure, delete it to ensure you do not incur costs.

Serverless and HPC Computer Services

Serverless computing

Serverless computing is a cloud-hosted execution environment that runs your code, yet abstracts the underlying hosting environment. You create an instance of the service and you add your code. No infrastructure configuration or maintenance is required, or even allowed.

You configure your serverless apps to respond to events. An event could be a REST endpoint, a periodic timer, or even a message received from another Azure service. The serverless app runs only when it's triggered by an event.

Scaling and performance are managed automatically, and you're billed only for the exact resources you use. You don't even need to reserve resources.

Serverless definition

The core characteristics that define a serverless service are:

- Service is consumption based. The service provisions resources on demand, and you only pay for what you use. Billing is typically calculated by the number of function calls, code execution time, and memory used. (Supporting services such as networking and storage could be charged separately.)
- Low management overhead. Because serverless service is cloud-hosted, you won't need to be patching VMs or have a burdensome operational workflow. Serverless services provide for the full abstraction of servers, so developers can just focus on their code. There are no distractions around server management, capacity planning, or availability.
- Auto-scale. Compute execution can be in milliseconds, so it's almost instant. It provides for event-drive scalability. Application components react to events and trigger in near real-time with virtually unlimited scalability.

Benefits of the serverless computing model

The benefits of serverless computing are:

- Efficiency:
 - Serverless computing can result in a shorter times for the product to get to market as developers can focus more on their applications and customer value.
 - Fixed costs are converted to variable costs, and you are only paying for what is consumed.
 - Cost savings are realized by the variable costs model.
- Focus:
 - You can focus on solving business problems, and not on allocating time to defining and carrying out operational tasks such as VM management.
 - Developers can focus on their code. There are no distractions around server management, capacity planning, or availability.
- Flexibility:
 - Serverless computing provides a simplified starting experience.
 - Easier pivoting means more flexibility.

- Experimentation is easier as well.
- You can scale at your pace.
- Serverless computing is a natural fit for microservices.

Serverless Azure services

Some of the serverless services in Azure are listed in the following table.

Azure service	Functionality
Azure Event grid	Manage all events that can trigger code or logic
Azure Functions	Execute code based on events you specify
Azure Automation	Automate tasks across Azure and hybrid environments
Azure Logic Apps	Design workflows and orchestrate processes

The service we're interested in from a DevOps and compute point of view is Azure Functions.

Functions as a service

Function as a service (FaaS) is an industry programming model that uses Functions to help achieve serverless compute. These functions have the following characteristics:

- Single responsibility. Functions are single purposed, reusable pieces of code that process an input and return a result.
- Short-lived. Functions don't stick around when they've finished executing, which frees up resources for further executions.
- Stateless. Functions don't hold any persistent state and don't rely on the state of any other process.
- Event driven and scalable. Functions respond to predefined events and are instantly replicated as many times as needed.

Azure Functions

Azure Functions are Azure's implementation of the FaaS programming model, with additional capabilities.



Azure Functions are ideal when you're only concerned with the code running your service and not the underlying platform or infrastructure. Azure Functions are commonly used when you need to perform work in response to an event (often via a REST request, timer, or message from another Azure service), and when that work can be completed quickly, within seconds or less.

Azure Functions scale automatically, and charges accrue only when a function is triggered, so they're a good choice when demand is variable. For example, you might be receiving messages from an Internet of

Things (IoT) solution that monitors a fleet of delivery vehicles. You'll likely have more data arriving during business hours. Azure Functions can scale out to accommodate these busier times.

Furthermore, Azure Functions are stateless; they behave as if they're restarted every time they respond to an event. This is ideal for processing incoming data. And if state is required, they can be connected to an Azure storage service. See **Functions²⁵** for more details.

Azure Functions features

Some key features of Azure Functions are:

- Choice of language. Write functions using your choice of C#, F#, or JavaScript.
- Pay-per-use pricing model. Pay only for the time spent running your code.
- Bring your own dependencies. Functions support NuGet and NPM, so you can use your favorite libraries.
- Integrated security. Protect HTTP-triggered functions with OAuth providers such as Azure AD, Facebook, Google, Twitter, and your Microsoft Account.
- Simplified integration. Easily leverage Azure services and software-as-a-service (SaaS) offerings.
- Flexible development. Code your functions directly in the portal, or set up continuous integration and deploy your code through GitHub, Azure DevOps Services, and other supported development tools.
- Open-source. The Functions runtime is open-source and available on GitHub.

You can download the free eBook, *Azure Serverless Computing cookbook*, from the **Azure Serverless Computing Cookbook²⁶** webpage.

Demonstration—Create Azure Function using Azure CLI

This walkthrough shows how to create a function from the command line or terminal. You use the Azure CLI to create a function app, which is the serverless infrastructure that hosts your function. The function code project is generated from a template by using the Azure Functions Core Tools, which is also used to deploy the function app to Azure.

Prerequisites

- Use Azure Cloud Shell.
- You require an Azure subscription to perform these steps. If you don't have one, you can create one by following the steps outlined on the **Create your Azure free account today²⁷** webpage.

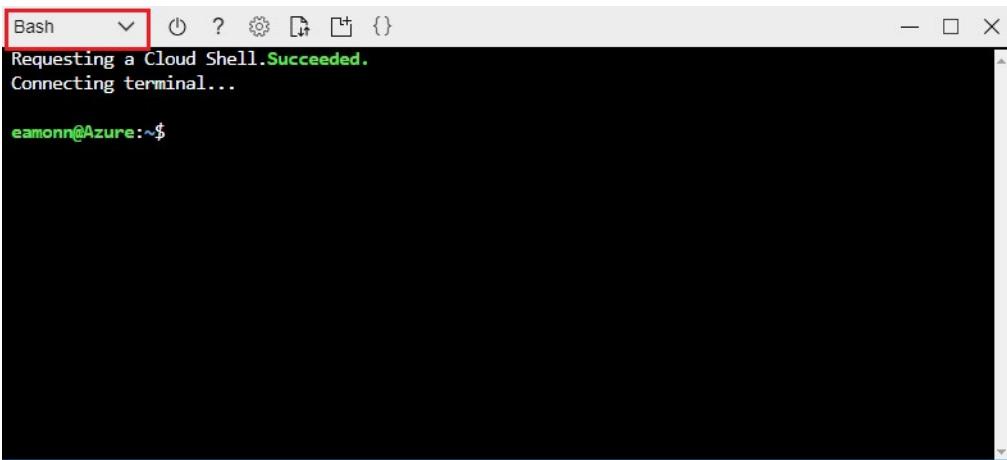
Steps

1. Open Azure Cloud Shell by going to <https://shell.azure.com>, or via the Azure Portal and selecting **Bash** as the environment option.

²⁵ <https://azure.microsoft.com/en-us/services/functions/>

²⁶ <https://azure.microsoft.com/en-us/resources/azure-serverless-computing-cookbook/>

²⁷ https://azure.microsoft.com/en-us/free/?ref=microsoft.com&utm_source=microsoft.com&utm_medium=docs&utm_campaign=visualstudio



2. Create the local function app project by running the following command from the command line to create a function app project in the MyFunctionProj folder of the current local directory. A GitHub repo is also created in MyFunctionProj:

```
func init MyFunctionProj
```

When prompted, select a worker runtime from the following language choices:

- dotnet. Creates a .NET class library project (.csproj).
- node. Creates a JavaScript project.

3. Use the following command to navigate to the new MyFunctionProj project folder:

```
cd MyFunctionProj
```

4. Create a function using the following command, which creates an HTTP-triggered function named MyHttpTrigger:

```
func new --name MyHttpTrigger --template "HttpTrigger"
```

5. Update the function. By default, the template creates a function that requires a function key when making requests. To make it easier to test the function in Azure, you need to update the function to allow anonymous access. The way that you make this change depends on your functions project language. For C#:

- Open the `MyHttpTrigger.cs` code file that is your new function. Use the following command to update the **AuthorizationLevel** attribute in the function definition to a value of anonymous, and save your changes:

```
[FunctionName("MyHttpTrigger")]
    public static IActionResult Run([HttpTrigger(AuthorizationLevel.
Anonymous,
    "get", "post", Route = null)]HttpRequest req, ILogger log)
```

6. Run the function locally. The following command starts the function app, which runs using the same Azure Functions runtime that is in Azure:

```
func host start --build
```

The –build option is required to compile C# projects.

7. Confirm the output. When the Functions host starts, it writes something similar to the following output, which has been truncated for readability:

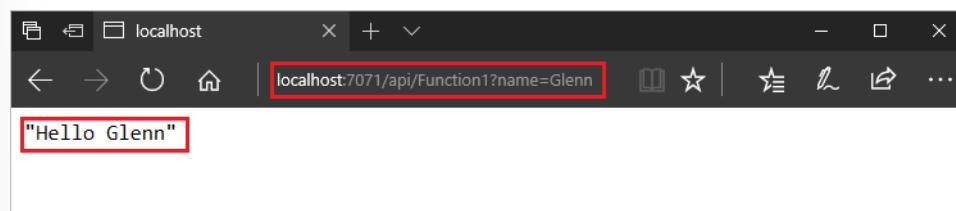
```
%%%%%%
%%%%%
@  %%%%   @
@@  %%%%
@ @  %%%%%%  @ @
@ @  %%%%  @ @
@ @  %%%%
@ @  %%%
@ @
%
...
Content root path: C:\functions\MyFunctionProj
Now listening on: http://0.0.0.0:7071
Application started. Press Ctrl+C to shut down.

...
Http Functions:

HttpTrigger: http://localhost:7071/api/MyHttpTrigger

[8/27/2018 10:38:27 PM] Host started (29486ms)
[8/27/2018 10:38:27 PM] Job host started
```

- Copy the URL of your **HttpTrigger** function from the runtime output, and paste it into your browser's address bar. Append the query string `?name=yourname` to this URL, and execute the request. The following image is the response in the browser to the GET request returned by the local function.



Now that you have run your function locally, you can create the function app and other required resources in Azure.

8. Create a resource group. An *Azure resource group* is a logical container into which Azure resources such as function apps, databases, and storage accounts are deployed and managed. Use the following command to create the resource group `az group create`.

```
az group create --name myResourceGroup --location westeurope
```

9. Create an Azure Storage account. Functions uses a general-purpose account in Azure Storage to maintain state and other information about your functions. Create a general-purpose storage account in the resource group you created by using the **az storage account create** command.

In the following command, substitute a globally unique storage account name where you see the `<storage_name>` placeholder. Storage account names must be between 3 and 24 characters in length, and can contain numbers and lowercase letters only:

```
az storage account create --name <storage_name> --location westeurope  
--resource-group myResourceGroup --sku Standard_LRS
```

After the storage account has been created, the Azure CLI shows information similar to the following example:

```
{  
    "creationTime": "2017-04-15T17:14:39.320307+00:00",  
    "id": "/subscriptions/bbbef702-e769-477b-9f16-bc4d3aa97387/resource-  
Groups/myresourcegroup/...",  
    "kind": "Storage",  
    "location": "westeurope",  
    "name": "myfunctionappstorage",  
    "primaryEndpoints": {  
        "blob": "https://myfunctionappstorage.blob.core.windows.net/",  
        "file": "https://myfunctionappstorage.file.core.windows.net/",  
        "queue": "https://myfunctionappstorage.queue.core.windows.net/",  
        "table": "https://myfunctionappstorage.table.core.windows.net/"  
    },  
    ....  
    // Remaining output has been truncated for readability.  
}
```

10. Create a function app. You must have a function app to host the execution of your functions. The function app provides an environment for serverless execution of your function code. It lets you group functions as a logic unit for easier management, deployment, and sharing of resources. Create a function app by using the **az functionapp create** command.

In the following command, substitute a unique function app name where you see the `<app_name>` placeholder, and the storage account name for `<storage_name>`. The `<app_name>` is used as the default DNS domain for the function app, and so the name needs to be unique across all apps in Azure. You should also set the `<language>` runtime for your function app, from dotnet (C#) or node (JavaScript).

```
az functionapp create --resource-group myResourceGroup --consump-  
tion-plan-location westeurope \  
--name <app_name> --storage-account <storage_name> --runtime <language>
```

Setting the *consumption-plan-location* parameter means that the function app is hosted in a Consumption hosting plan. In this serverless plan, resources are added dynamically as required by your functions, and you only pay when functions are running.

After the function app has been created, the Azure CLI shows information similar to the following example:

```
{  
    "availabilityState": "Normal",  
    "clientAffinityEnabled": true,
```

```
"clientCertEnabled": false,  
"containerSize": 1536,  
"dailyMemoryTimeQuota": 0,  
"defaultHostName": "quickstart.azurewebsites.net",  
"enabled": true,  
"enabledHostNames": [  
    "quickstart.azurewebsites.net",  
    "quickstart.scm.azurewebsites.net"  
,  
....  
// Remaining output has been truncated for readability.  
}
```

11. Deploy the function app project to Azure. After the function app is created in Azure, you can use the **func azure functionapp publish** command to deploy your project code to Azure:

```
func azure functionapp publish <FunctionAppName>
```

You'll see something like the following output, which has been truncated for readability.

```
Getting site publishing info...  
Preparing archive...  
Uploading content...  
Upload completed successfully...  
Deployment completed successfully...  
Syncing triggers...
```

You are now ready to test your functions in Azure.

12. Test the function. Use **cURL** to test the deployed function on a Mac or Linux computer, or using Bash on Windows. Execute the following **cURL** command, replacing the **< app_name >** placeholder with the name of your function app. Append the query string **&name=< yourname >** to the URL:

```
curl https://< app_name >.azurewebsites.net/api/MyHttpTrigger?name=< your-  
name >
```

- ✓ Note: Remember to delete the resources if you are no longer using them.

Batch services

Azure Batch is a fully-managed cloud service that provides job scheduling and compute resource management. It creates and manages a pool of compute nodes (VMs), installs the applications you want to run, and schedules jobs to run on the nodes. It enables applications, algorithms, and computationally intensive workloads to be broken into individual tasks for execution to be easily and efficiently run in parallel at scale.

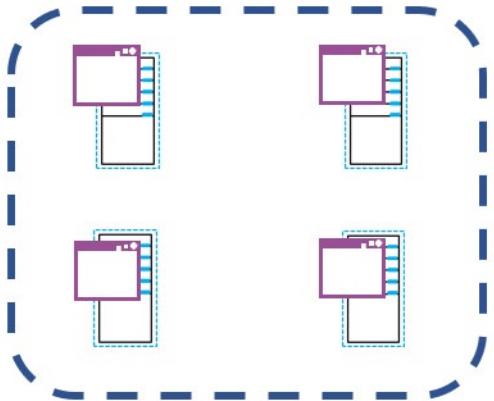
Using Azure Batch, there is no cluster or job scheduler software to install, manage, or scale. Instead, you use Batch APIs and tools, command-line scripts, or the Azure portal to configure, manage, and monitor your jobs.

Usage scenarios and application types

The following topics are just some of the usage scenarios for Azure Batch.

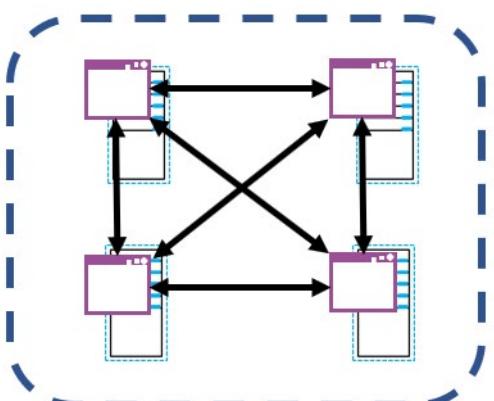
Independent\parallel

This is the most commonly used scenario. The applications or tasks, do not communicate with each other. Instead, they operate independently. The more VMs or nodes you can bring to a task, the quicker it will complete. Examples of usage would be Monte Carlo risk simulations, transcoding, and rendering a movie frame by frame.



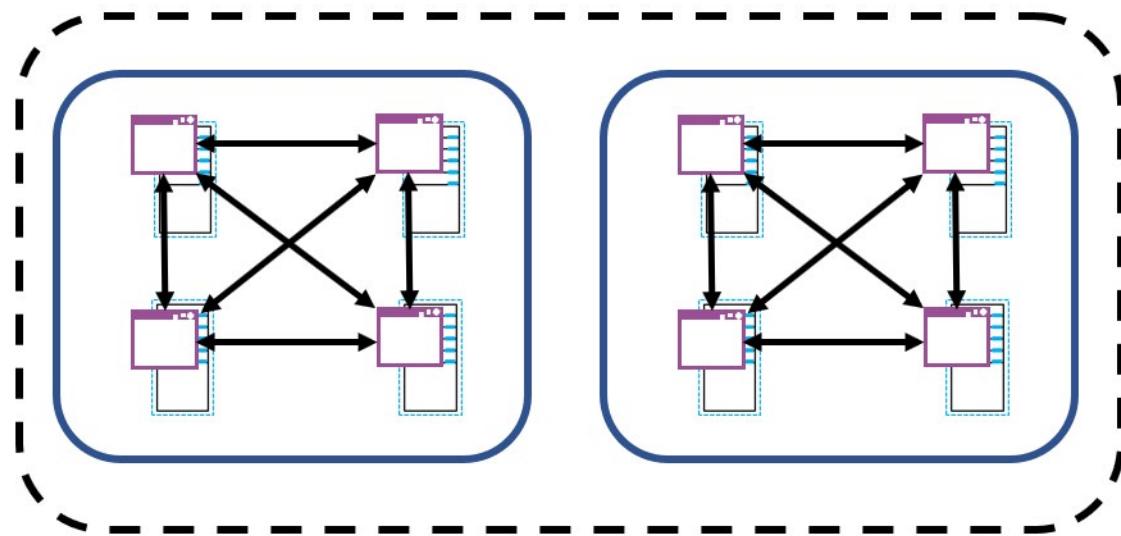
Tightly coupled

From traditional high performance computing (HPC) such as scientific, or computing and engineering tasks, applications or tasks communicate with each other. They would typically use the Message Passing Interface (MPI) API for this inter-node communication. However, they can also use low-latency, high-bandwidth Remote Direct Memory Access (RDMA) networking. Examples of usage would be car crash simulations, fluid dynamics, and Artificial Intelligence (AI) training frameworks.



Multiple tightly coupled in parallel

You can also expand on this tightly coupled MPI scenario. For example, instead of having four nodes carrying out a job, you can have 40 nodes and run the job 10 times in parallel to scale out the job task.



Batch Service components

Batch service primarily consists of two components:

- Resource management. Batch service manages resources by creating, managing, monitoring, and scaling the pool (or pools) of VMs, which are required to run the application. You can scale from a few VMs up to tens of thousands of VMs, enabling you to run the largest, most resource-intensive workloads. Furthermore, no on-premises infrastructure is required.
- Job Scheduler. Batch service provides a job scheduler. You submit your work via jobs, which are effectively a series of tasks, and specify individual tasks into the VM pool (or set of VM pools).

Running an application

To get an application to run, you must have the following items:

- An application. This could just be a standard desktop application; it doesn't need to be cloud aware.
 - Resource management. You need a pool of VMs, which Batch service creates, manages, monitors, and scales.
 - A method to get the application onto the VMs. You can:
 - Store the application in blob storage, and then copy it onto each VM.
 - Have a container image and deploy it.
 - Upload a zip or application package.
 - Create a custom VM image, then upload and use that.
 - Job scheduler. Create and define the tasks that will combine to make the job.
 - Output Storage: You need somewhere to place the output data, typically use Blob storage.
- ✓ Note: The unit of execution is what can be run on the command line in the VM. The application itself does not need to be repackaged.

Cost

Batch services provide:

- The ability to scale VMs as needed.
 - The ability to increase and decrease resources on demand.
 - Efficiency, as it makes best use of the resources.
 - Cost effectiveness, because you only pay for the infrastructure you use when you are using it.
- ✓ Note: There is no additional charge for using a Batch service. You only pay for the underlying resources consumed, such as the VMs, storage, and networking.

Azure Service Fabric

Azure Service Fabric Overview

Azure Service Fabric is a distributed systems platform that makes it easier to package, deploy, and manage scalable and reliable containerized microservice applications. It has two primary functions, which are providing for:

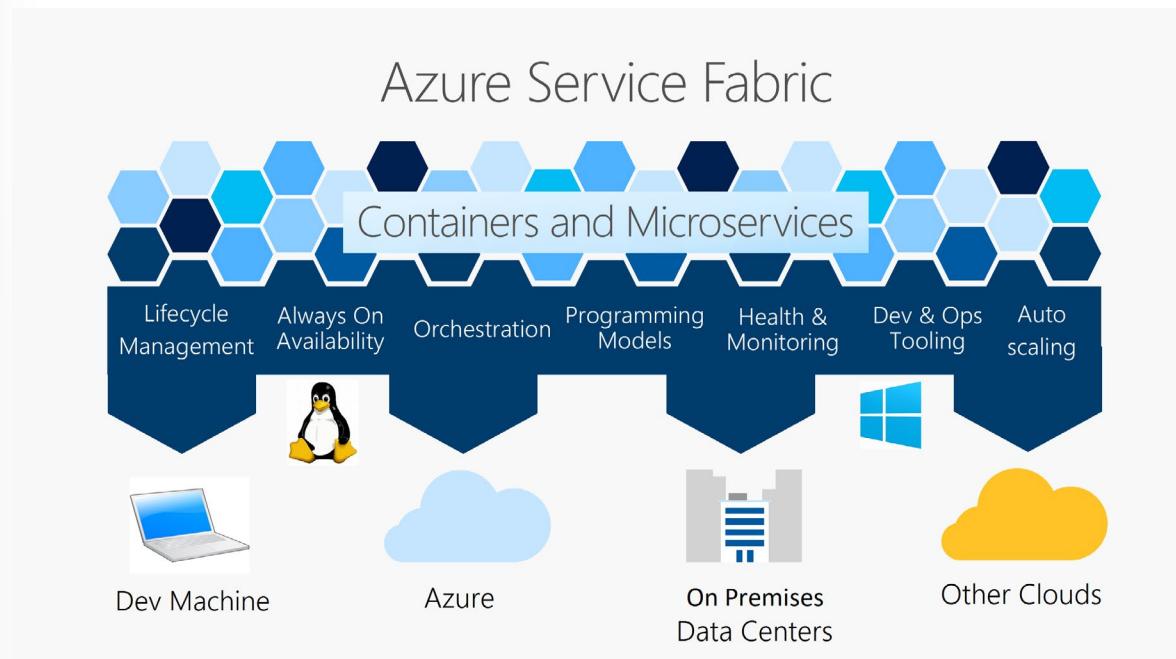
- Microservices applications
- Container orchestration

Developers and administrators can avoid complex infrastructure problems, and instead focus on implementing mission-critical workloads.

Service Fabric is designed for modern cloud, native application. It represents the next-generation platform for building and managing these enterprise-class, tier-1, cloud-scale applications running in containers.

Where and what can Service Fabric run?

Service Fabric runs everywhere. You can create clusters for Service Fabric in many environments, including Azure or on premises, on both Windows Server and Linux operating systems. You can even create clusters on other public clouds. In addition, the development environment in the SDK is identical to the production environment, with no emulators involved. In other words, what runs on your local development cluster deploys to the clusters in other environments



Applications and services

Service Fabric enables you to build and manage scalable and reliable applications. It is composed of microservices that run at high density on a shared pool of machines, which is referred to as a *cluster*. It provides a sophisticated, lightweight runtime to build that is distributed, scalable, stateless, and stateful

microservices running in containers. It also provides comprehensive application management capabilities to provision, deploy, monitor, upgrade/patch, and delete deployed applications including containerized services.

Key capabilities

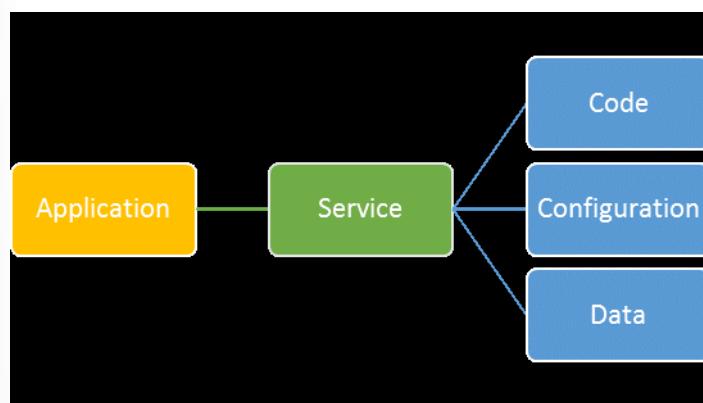
By using Service Fabric, you can:

- Deploy to Azure or to on-premises datacenters running Windows or Linux operating systems, with zero code changes. You write once, and then deploy anywhere to any Service Fabric cluster.
 - Develop scalable applications composed of microservices by using the Service Fabric programming models, containers, or any code.
 - Develop highly reliable stateless and stateful microservices. Simplify the design of your application by using stateful microservices.
 - Use the Reliable Actors programming model to create cloud objects with self-contained code and state.
 - Deploy and orchestrate containers that include Windows containers and Linux containers. Service Fabric is a data-aware, stateful container orchestrator.
 - Deploy applications in seconds at high density, with hundreds or thousands of applications or containers per machine.
 - Deploy different versions of the same application side by side, and upgrade each application independently.
 - Manage the lifecycle of your applications without any downtime, including breaking and nonbreaking upgrades.
 - Scale out or scale in the number of nodes in a cluster. As you scale nodes, your applications automatically scale.
 - Monitor and diagnose the health of your applications and set policies for performing automatic repairs.
 - Watch the resource balancer orchestrate the redistribution of applications across the cluster. Service Fabric recovers from failures and optimizes the distribution of load based on available resources.
- ✓ Note: Service Fabric is currently undergoing a transition to open development. The goal is to move the entire build, test, and development process to GitHub. You can view, investigate and contribute on the [28](https://github.com/Microsoft/service-fabric/) page. There are also many sample files and scenarios to help in deployment and configuration.

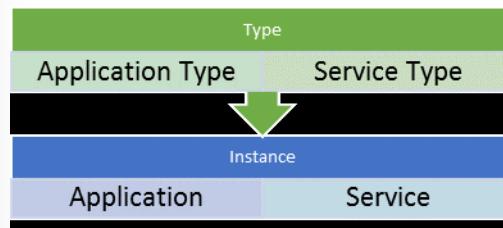
Application Model

Service Fabric applications consist of one or more services that work together to automate business processes. A *service* is an executable that runs independently of other services, and is composed of code, configuration, and data. Each element is separately versionable and deployable.

²⁸ <https://github.com/Microsoft/service-fabric/>



Application and service types

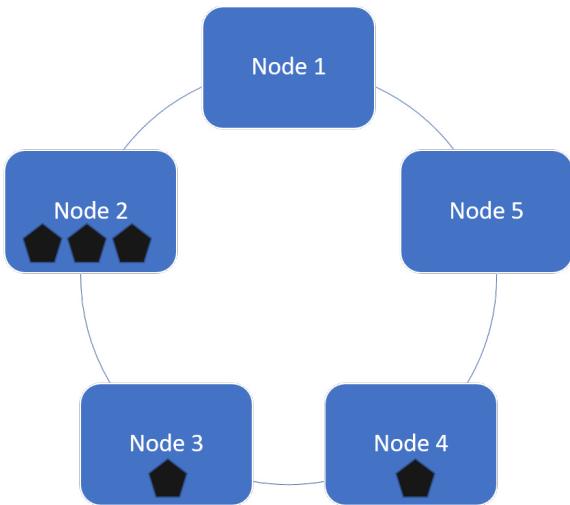


Creating an application instance requires an application type, which is the template that specifies which services are part of the application. This concept is similar to object-oriented programming. The application type is comparable to class definition, and the application is comparable to the instance. You can create multiple named application instances from one application type.

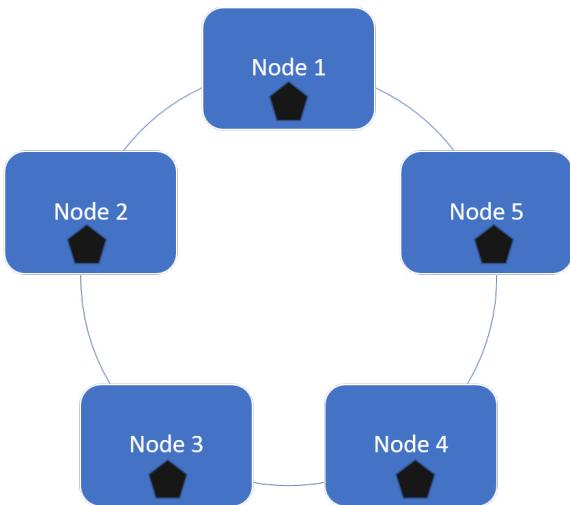
The same concept applies to services. The service type defines the code and configuration for the service and the endpoints that the service uses for interaction. You can create multiple service instances by using one service type. An application specifies how many instances of a service type should be created.

Both application type and service type are described through XML files. Every element of the application model is independently versionable and deployable.

Resource balancing



While applications are running in Azure Service Fabric, they are constantly monitored for health. Service Fabric ensures that services keep running well and that the available server resources are used optimally. This means that sometimes services are moved from busy nodes to less busy nodes to keep overall resource consumption well balanced. The image above displays an imbalanced cluster. Node 2 hosts three services, while Node 1, and Nodes 3-5 are empty. Service Fabric will detect this situation and resolve it.



After Service Fabric completes the balancing operation, your cluster will look like the image above. Every node now runs one service.

In reality, each node will likely run many services. Because every service is different, it's usually possible to combine services on one node to make optimal use of the server's resources. Service Fabric does all of this automatically.

Programming Models

When developing applications for use on Service Fabric, there are a number of options available.

Windows

Azure Service Fabric comes with an SDK, and development tool support. You can develop Windows clusters in C# using Microsoft Visual Studio 2015 or Visual Studio 2017.

Linux

Developing Java-based services for Linux clusters is probably easiest by using **Eclipse Neon²⁹**. However, it's also possible to program in C# using .NET Core and Visual Studio Code.

Programming models

You can choose from four different programming models to create a Service Fabric application:

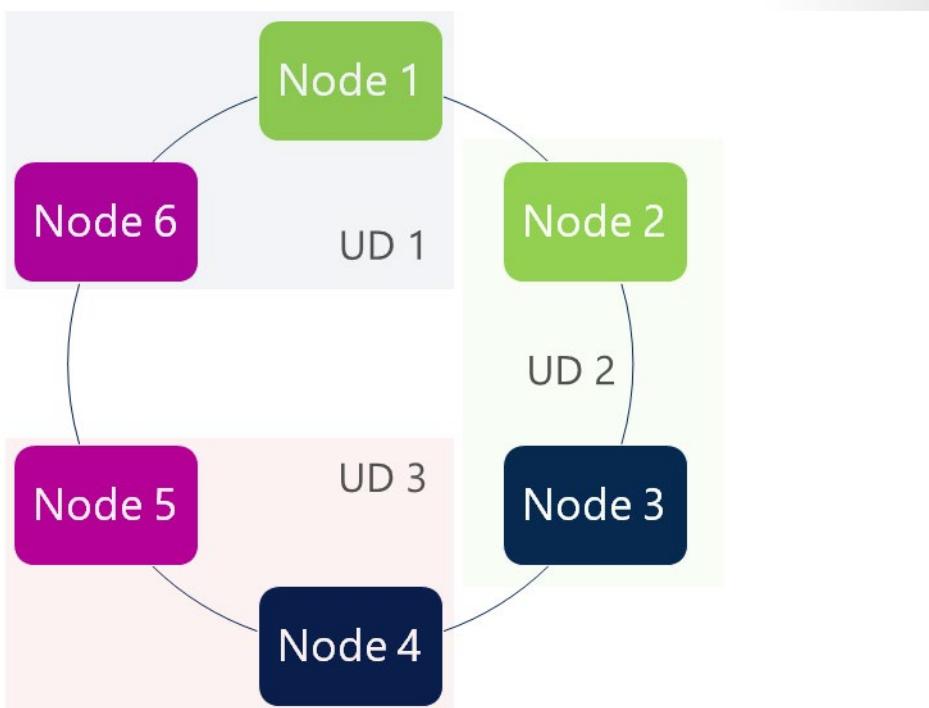
- **Reliable Services.** *Reliable Services* is a framework you can use to create services that use specific features, which Service Fabric provides. One important feature is a distributed data storage mechanism. Others are custom load and health reporting, and automatic endpoint registration. These enable discoverability and interaction between services.
 - There are two distinct types of Reliable Services that you can create:
 - *Stateless services* are intended to perform operations that don't require keeping an internal state. Examples are services that host ASP.NET Web APIs, or services that autonomously process items read from a Service Bus Queue.
 - *Stateful services* keep an internal state, which is automatically stored redundantly across multiple nodes for availability and error recovery. The data stores are called *Reliable Collections*.
- **Reliable Actors.** *Reliable Actors* is a framework built on top of Reliable Services, which implements the Virtual Actors' design pattern. An Actor encapsulates a small piece of state and behavior. One example is Digital Twins, in which an Actor represents the state and the abilities of a device in the real world. Many IoT applications use the Actor model to represent the state and abilities. The state of an Actor can be volatile, or it can be kept in the distributed store. This store can be memory-based or on a disk.
- **Guest executables.** You can also package and run existing applications as a Service Fabric (stateless) service. This makes Applications highly available. The platform ensures that the instances of an application are running. You can also upgrade Applications with no downtime. If problems are reported during an upgrade, Service Fabric can automatically roll back the deployment. Service Fabric also enables you to run multiple applications together in a cluster, which reduces the need for hardware resources.
 - ✓ Note: When using Guest executables, you cannot use some of the platform capabilities (such as the Reliable Collections).
- **Containers.** You can run Containers in a similar way as running guest executables. What's different is that Service Fabric can restrict resource consumption (CPU and memory, for example) per container. Limiting resource consumption per service enables you to achieve even higher densities on your cluster.

²⁹ <https://www.eclipse.org/neon/>

Scaling Azure Clusters

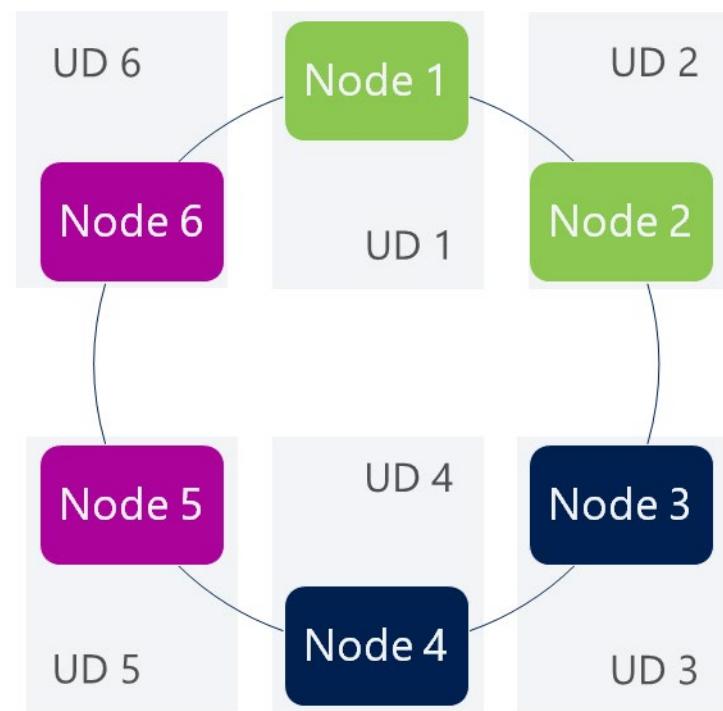
Before we talk about scaling, let's briefly explain some concepts:

- *Scaling in and out* is the process of removing and adding nodes to the cluster.
- *Scaling up and down* is the process of changing the size (SKU) of the VMs that make up the cluster.
- *Node types* are one or more individual VM scale sets that make up the cluster.
- *Fault domain* is a hierarchical structure of infrastructure levels in which faults can occur. For example, faults can happen on disk drives, machines, power supplies, or server racks, and in entire data centers. To create a system that is highly available, you must consider these fault domains. Having two nodes that share a fault domain means they share a single point of failure.
- *Upgrade domain* is a policy that describes groups of nodes that will be upgraded simultaneously. Nodes in different upgrade domains will not be upgraded at the same time. This means that upgrade domains are useful when performing rolling upgrades of your software. One by one, every upgrade domain will be processed.



For example, in this image you see six nodes. Imagine that node pairs one and two, three and four, and five and six each share a server rack, which means that they share a fault domain. Then, by policy, node pairs one and six, two and three, and four and five were put in upgrade domains. This means that changes to these node pairs are applied simultaneously. This includes changes to the cluster software and to running services.

By upgrading two nodes at the same time, they complete quickly. By adding more upgrade domains, your upgrades become more granular, and because of that, have a lower impact. The most commonly used setup is to have one upgrade domain for one fault domain.



This means that as exhibited in this image, upgrades are applied to one node at a time. When services are deployed to both multiple fault domains and correctly configured upgrade domains, they are able to manage node failures, even during upgrades.

Boundaries

You can grow or shrink the number of servers that define in the cluster by changing the size of a VM scale set. However, there are some restrictions that apply.

Lower boundary

Earlier in this module, you learned that one of the platform services of Service Fabric is a distributed data store. This means that data stored on one node is replicated to a quorum (majority) of secondary nodes. To work properly, you'll need to have multiple healthy nodes in your cluster. The precise number needed depends on the desired reliability of your services.

Upper boundary in Azure

When using existing platform images, VM scale sets are limited to 1,000 VMs. In Azure, Service Fabric can scale up to 100 nodes in each scale set.

Durability levels in Azure

Whether Service Fabric can automatically manage infrastructural changes depends on the cluster's configured durability level. The durability level indicates the level of privileges Service Fabric has to influence Azure infrastructural operations on the cluster's underlying VMs.

There are three durability levels, as exhibited in the following table.

Durability tier	Required minimum number of VMs	Supported VM SKUs	Updates you make to your virtual machine scale set	Updates and maintenance initiated by Azure
Gold	5	Full-node SKUs dedicated to a single customer (for example, L32s, GS5, G5, DS15_v2, D15_v2)	Can be delayed until approved by the Service Fabric cluster	Can be paused for 2 hours per update domain to allow additional time for replicas to recover from earlier failures
Silver	5	VMs of single core or above	Can be delayed until approved by the Service Fabric cluster	Cannot be delayed for any significant period of time
Bronze	1	All	Will not be delayed by the Service Fabric cluster	Cannot be delayed for any significant period of time

Adding or changing nodes in Azure

You can extend your Azure cluster capacity simply by increasing the VM scale set capacity. You can do this by using PowerShell, Azure Resource Manager templates, and by code. Changing the SKU influences all cluster nodes, so whether you can do this depends on the configured durability level. When you use Silver or Gold levels, you can do this, whereas when using the Bronze level, you'll have some downtime because all VMs upgrade simultaneously.

Unlike Azure Access Control (ACS) clusters, you can configure Service Fabric cluster scale sets to scale automatically based on performance counter metrics.

Adding nodes in standalone clusters

Scaling up a standalone cluster also must be done using PowerShell scripts. Running the script `Add-Node.ps1` on a prepared server will add it to an existing cluster.

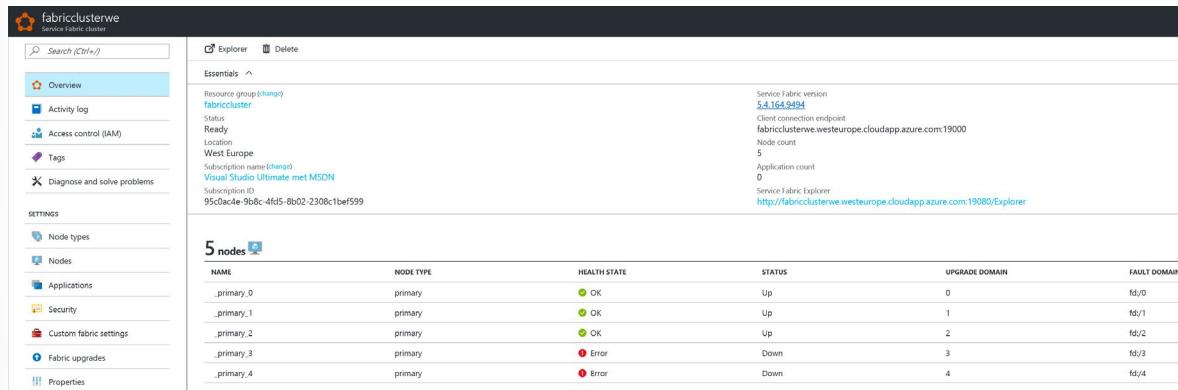
Removing nodes in Azure

Although adding nodes to an Azure cluster is a matter of increasing the scale set capacity, removing them can be more complicated. Removing nodes from a cluster has implications for the distributed data store that contains the Reliable Collections of your services. Decreasing the instance count of your VM scale set results in the removal of cluster nodes. The impact of this depends on the durability level.

If you're using the Bronze durability level, you must notify Service Fabric beforehand of your intention to remove a node. This instructs Service Fabric to move services and data away from the node. In other words, it drains the node.

Next, you need to remove that node from the cluster. You must run the PowerShell script `Disable-ServiceFabricNode` for each node that you want to remove, and wait for Service Fabric to complete the operation.

If you don't properly remove the node from the cluster, Service Fabric will assume that the nodes have simply failed and will return later after reporting them as having the status *Down*.



NAME	NODE TYPE	HEALTH STATE	STATUS	UPGRADE DOMAIN	FAULT DOMAIN
_primary_0	primary	OK	Up	0	fd/0
_primary_1	primary	OK	Up	1	fd/1
_primary_2	primary	OK	Up	2	fd/2
_primary_3	primary	Error	Down	3	fd/3
_primary_4	primary	Error	Down	4	fd/4

Of course, you can also remove nodes using code.

Removing nodes from standalone clusters

Removing nodes is somewhat different from the process in Azure clusters. Instead of executing `Disable-ServiceFabricNode`, you execute the script `RemoveNode.ps1` on the server that you want to remove.

Create Clusters anywhere

You can create Service Fabric clusters in many environments:

- Windows operating system
- Linux operating system
- On premises
- In the cloud
- On one machine
- On multiple servers

There are two distinct versions of the Service Fabric binaries, one that runs on the Windows operating system and one for Linux. Both are restricted to the 64-bit platform.

For more details see the [Create Service Fabric clusters on Windows Server or Linux³⁰](#) page.

Windows Server

You can deploy a cluster manually by using a set of PowerShell tools on a prepared group of servers, and then running Windows Server 2016 or Windows Server 2019. This approach is called a *Service Fabric standalone cluster*.

It's also possible to create a cluster in Azure. You can do this using the Azure Portal, or by using an Azure Resource Manager template. This will create a cluster and everything that's needed to run applications on it.

³⁰ <https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-deploy-anywhere>

Finally, there is an option to create a cluster specifically for development use that runs on just one machine simulating multiple servers. This is called a *local development cluster*, and it allows developers to debug their applications before deploying them to a production cluster.

- ✓ Note: It's important to know that for every type of deployment, the actual Service Fabric binaries are the same. This means that an application that works on a development cluster will also work on an on-premises or cloud-hosted cluster without requiring modifications to the code. This is similar to the portability that containerization offers.

Linux

At the time of this writing, Service Fabric for Linux has been released. However, it does not yet have complete feature parity between Windows and Linux. This means that some Windows features are not available on Linux. For example, you cannot create a Service Fabric anywhere cluster, and all programming models are in preview (including Java/C# Reliable Actors, Reliable Stateless Services, and Reliable Stateful Services).

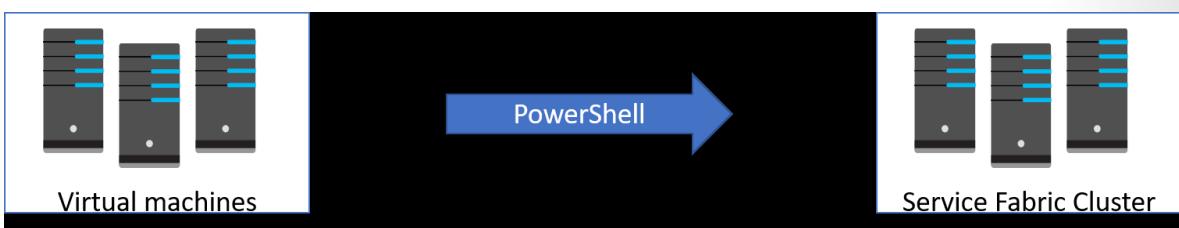
You can create a Linux cluster in Azure and create a local development cluster on the Linux Ubuntu 16.04 and Red Hat Enterprise Linux 7.4 (preview support) operating systems.

For more details, see the **Differences between Service Fabric on Linux and Windows³¹** page.

- ✓ Note: Standalone clusters currently aren't supported for Linux. Linux is supported on one-box for development and Linux virtual machine clusters.

Demonstration-Create a standalone Service Fabric cluster on Windows Server

Creating a standalone Azure Service Fabric cluster requires provisioning the hardware resources up front. After you have created and configured the required machines, you can use PowerShell to create a cluster. The required scripts and binaries can be downloaded in a single package, the *Service Fabric standalone-package*.



- ✓ Note: Detailed steps on how to set up a Service Fabric cluster are available on the **Create a standalone cluster running on Windows Server page.³²**

Prerequisites:

- You need to download setup files and PowerShell scripts for the Service Fabric standalone package, which you run to setup a service fabric cluster. You can download these from the **Download Link - Service Fabric Standalone Package - Windows Server page.³³**

³¹ <https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-linux-windows-differences>

³² <https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-cluster-creation-for-windows-server>

³³ <https://go.microsoft.com/fwlink/?LinkId=730690>

- ✓ Note: Remember that currently standalone clusters aren't supported for Linux. Linux is supported on one-box for development and Linux multi-machine clusters on Azure. As such there is no equivalent download package for Linux.

Process for creating a standalone server

The following steps broadly apply to all deployments. However, the steps below are completed in the context of a standalone deployment on Windows Server:

1. Prepare:

- You need to plan the required amount of server resources. Those servers need to meet the minimum requirements for Service Fabric.
- Fault domains and upgrade domains need to be defined.
- The software prerequisites need to be installed.
- Ensure you have downloaded the Service Fabric standalone package, which contains scripts and details that you need to set up a Service Fabric cluster. You can download the package from the **Download Link - Service Fabric Standalone Package - Windows Server page**.³⁴
- Familiarize yourself with the downloaded files.
- Several sample cluster configuration files are installed with the setup package that you downloaded. **ClusterConfig.Unsecure.DevCluster.json** is the simplest cluster configuration, which is an unsecure, three-node cluster running on a single computer.

2. Validate:

- A validation script `TestConfiguration.ps1` is provided as part of the package. It has a Best Practices Analyzer that can validate some of the criteria on your resources. You can validate the environment before creating the cluster by running the following command:

```
.\TestConfiguration.ps1 -ClusterConfigFilePath .\ClusterConfig.Unsecure.  
DevCluster.json
```

3. Create:

- A creation script `CreateServiceFabricCluster.ps1` is also provided. Running this will create the entire cluster for you on all designated machines. You can run the following command:

```
.\CreateServiceFabricCluster.ps1 -ClusterConfigFilePath .\ClusterConfig.json  
-AcceptEULA
```

4. Connect and Visualize:

- Connect to the cluster to verify that it is running and available, using the following command:

```
Connect-ServiceFabricCluster -ConnectionEndpoint <*IP Address of a Machine*>:<-  
Client connection end point port>
```

i.e.

³⁴ <https://go.microsoft.com/fwlink/?LinkId=730690>

```
Connect-ServiceFabricCluster -ConnectionEndpoint 192.13.123.2345:19000
```

- Service Fabric Explorer is a service that runs in the cluster, which you access using a browser. Open a browser and go to <http://localhost:19000/explorer>.

The screenshot shows the Service Fabric Explorer interface running in a Microsoft Edge browser. The title bar says "Service Fabric Explorer". The left sidebar has a tree view with "Cluster" expanded, showing "Applications" and "Nodes". "Nodes" is selected and expanded, showing three nodes: "vm0", "vm1", and "vm2". The main area is titled "Nodes" and contains a table with columns: Name, Address, Node Type, Upgrade Domain, Fault Domain, Health State, and Status. The data in the table is:

Name	Address	Node Type	Upgrade Domain	Fault Domain	Health State	Status
vm0	localhost	NodeType0	UD0	fd/dc1/r0	OK	Up
vm1	localhost	NodeType1	UD1	fd/dc2/r0	OK	Up
vm2	localhost	NodeType2	UD2	fd/dc3/r0	OK	Up

5. Upgrade:

- You can run the PowerShell script `Start-ServiceFabricClusterUpgrade` that is installed on the nodes as part of the cluster deployment, to upgrade the cluster software to a specific version.

6. Remove:

- If you need to remove a cluster, run the script `RemoveServiceFabricCluster.ps1`. This removes Service Fabric from each machine in the configuration. Use the following command:

```
# Removes Service Fabric from each machine in the configuration
.\RemoveServiceFabricCluster.ps1 -ClusterConfigFilePath .\ClusterConfig.json
-Force
```

- To removes Service Fabric from the current machine, use the `.\CleanFabric.ps1` script. You can also run the following command:

```
# Removes Service Fabric from the current machine
.\CleanFabric.ps1
```

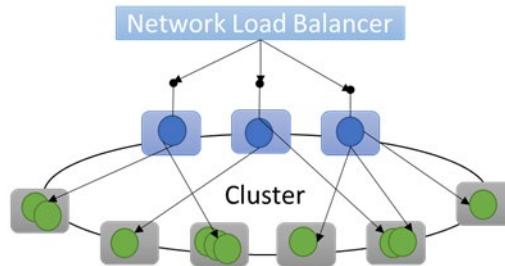
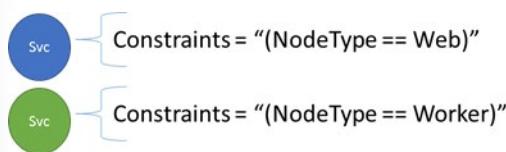
You'll likely add your own public IP address and load balancer to this cluster, if you need to run services that are reachable over the internet.

Placement Constraints

You can use Placement constraints to:

- Isolate workloads from each other.
- Lift and shift an existing N-tier application into Azure Service Fabric.

- Run services on specific server configurations.



Placement constraints are put in place in two steps:

- Add key-value pairs to cluster nodes. You can create a Web and Worker pool by creating two VM scale sets in the cluster, and marking one VM as:

```
'NodeType Web'
```

(These are the blue nodes in the image.)

and VMs in the other as:

```
'NodeType Worker'
```

(These are the green nodes in the image.)

- Add constraint statements to your service. Creating a service that must run on the Web pool would have a constraint statement such as:

```
'NodeType == Web'
```

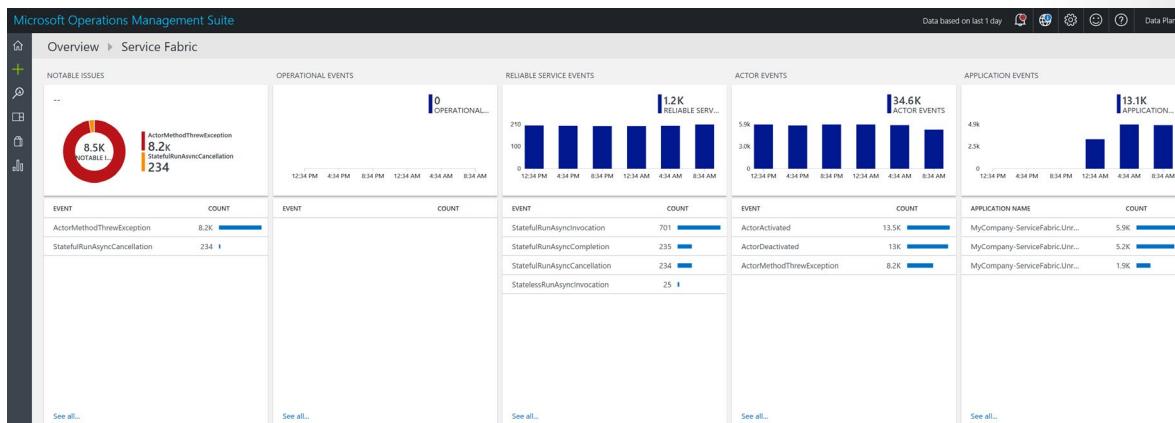
Service Fabric will take care of the rest. Services that were already running will be moved if necessary, and services will be placed on the proper nodes immediately for new deployments.

✓ Note: It's important to realize that placement constraints restrict Service Fabric in its ability to balance overall cluster resource consumption. Ensure that your placement constraints are not too restrictive. If Service Fabric is unable to comply with a placement constraint, your service won't be able to run. Always create pools of multiple nodes when defining constraints.

Configure Monitoring and Logging Overview

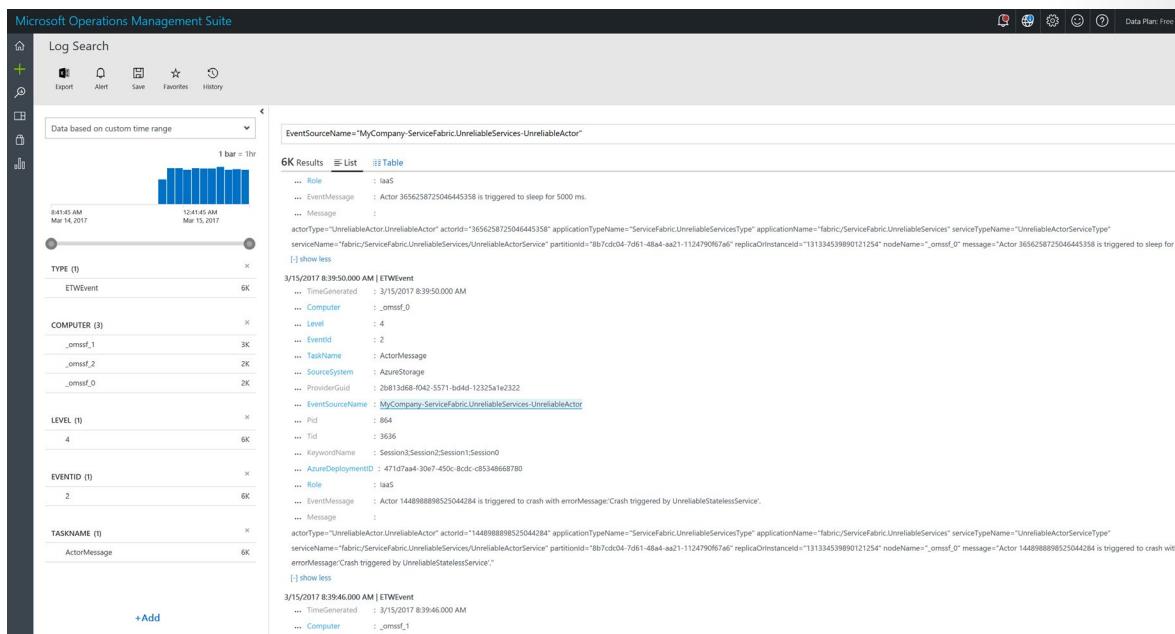
We'll look at some details about OMS and explore more options for logging and analyzing diagnostics.

OMS and Azure Service Fabric



In the Microsoft Operations Management Suite (OMS) There is a management solution—or plug-in—for OMS that is designed specifically for diagnostics on Service Fabric clusters. This solution is called *Service Fabric Analytics*.

Adding this to your OMS workspace provides you with a dashboard. In one glance, you'll get an overview of important issues and cluster and application events. These graphs are based on diagnostics data gathered from the servers forming the Service Fabric cluster. By using the VM extension Windows Azure Diagnostics, you install an agent on your VMs that is able to collect and upload this diagnostics data into a storage account. OMS can access that data to analyze and present the information on the dashboard.



If you want to drill down to the details of the graphs, you can do so by clicking on the items in the table. This will navigate you to the OMS Log Analytics management solution. By using Log Analytics, you can view detailed descriptions of all captured diagnostics data.

You can review details of diagnostics events such as Event Trace for Windows (ETW), that were generated by Services and Actors running in Service Fabric. *ETW* is a high-performance logging system that you can use for logging information such as errors or diagnostics traces from your application.

OMS has been used for a lot of our diagnostics so far, but there are alternative tools that you can use.

Service Fabric diagnostics alternatives

Logging by using an agent process is generally advisable, as it can keep working even if your service does not. This in-process approach works well for custom logs created by services that would otherwise require infrastructure changes to be properly logged. Specifically, all event sources used in code must be registered on every cluster node to work. The recommended way to do this is to register them in your Azure Resource Manager template.

If you don't want to register every event provider in your template, using Microsoft Diagnostics Event-Flow for these specific logs might be a good solution. However, it's fine to use a combination of both methods.

EventFlow

created by the Microsoft Visual Studio Team, *EventFlow* is an open-source library designed specifically for in-process log collection. This library enables your services to send logs directly to a central location, while not relying on an agent such as the Azure Diagnostics extension to do that. This makes sense if services come and go, or when services need to send their data to varying central locations.

EventFlow does not rely on the Event Trace for Windows infrastructure; it can send logs to many outputs, including Application Insights, OMS, Azure Event Hubs, and the console window.

Backup and recovery

Creating backups requires development effort, and a method to be called within a Reliable Service. This causes the data to be copied to a local backup folder.

After that, it's possible to copy that data to a central storage location. This location should be in a different fault domain, for example, in a different cloud. Because data is shared between partitions, every primary replica of your Stateful Reliable Services needs to create its own backups. Use the following code to do this:

```
public async Task BeginCreateBackup()
{
    var backupDescription = new BackupDescription(BackupOption.Full, PostBackupCallbackAsync);
    await BackupAsync(backupDescription);
}

private async Task<bool> PostBackupCallbackAsync(BackupInfo backupInfo,
CancellationToken cancellationToken)
{
    await _centralBackupStore.UploadBackupFolderAsync(backupInfo.Directory,
cancellationToken);
    return true;
}
```

In the code sample above, a backup is created first. After that operation completes, the method **Post-BackupCallAsync** will be invoked. In this method, the local backup folder is copied to the central location. The implementation of `_centralBackupStore` is omitted.

Restoring backups

Restoring backups also requires some development effort. The Reliable Service needs to have code that is executed by Service Fabric in a data loss situation such as a node failure or using code. After the data loss is triggered, your Reliable Service can access your central storage location and download the contents to the local backup folder. After that, data can be restored from the local backup folder. Every running primary replica of your Stateful Reliable Services must restore its own backups. The following command demonstrates how to do these steps:

```
public async Task BeginRestoreBackup()
{
    var partitionSelector = PartitionSelector.PartitionKeyOf(Context.Service-
Name,
    ((Int64RangePartitionInformation) Partition.PartitionInfo).LowKey);
    var operationId = Guid.NewGuid();

    await new FabricClient(FabricClientRole.Admin).TestManager.StartPartition-
DataLossAsync(operationId, partitionSelector, DataLossMode.FullDataLoss);
    //Causes OnDataLossAsync to be called.
}

protected override async Task<bool> OnDataLossAsync(RestoreContext re-
storeCtx, CancellationToken cancellationToken)
{
    string backupFolder = Context.CodePackageActivationContext.WorkDirectory;
    await _centralBackupStore.DownloadBackupFolderAsync(backupFolder, cancella-
tionToken);

    var restoreDescription = new RestoreDescription(backupFolder, RestorePoli-
cy.Force);
    await restoreCtx.RestoreAsync(restoreDescription, cancellationToken);
    return true;
}
```

In this code sample, backups are retrieved from a central location, and the local folder is used to call `RestoreAsync`. The call to `OnDataLossAsync` can be triggered by executing the following command:

```
FabricClient.TestManagementClient.StartPartitionDataLossAsync
```

Alternatively, you can use the following PowerShell command:

```
Start-ServiceFabricPartitionDataLoss
```

Again, the implementation of `_centralBackupStore` is omitted.

Fire drills

Consider your backup strategy carefully. The amount of data loss that is acceptable differs for every service. The size of the central store will grow quickly if you create many full backups.

Make sure to gain hands-on experience with creating and restoring backups by practicing it. This way, you'll know that your solution works, and you won't discover that your backup strategy is insufficient during a real disaster-recovery situation.

Demonstration-Docker Compose deployment to Service Fabric

Docker Compose is a way to deploy containers to a cluster by means of a declaration of desired state. It uses a YAML file to describe which containers need to be deployed.

YAML definition

A sample of a YAML definition would be similar to the following code:

```
version: '3'

services:

  web:
    image: microsoft/iis:nanoserver
    ports:
      - "80:80"

networks:
  default:
    external:
      name: nat
```

This sample file results in a single container based on the image named `microsoft/iis:nanoserver`. We did not specify a container registry to use, so Docker Hub is used by default. The container exposes IIS at port 80. We connect the container port to the host port 80, by specifying `80:80`. Finally, we selected the default `nat` network to connect containers.

Deploying a Docker Compose file

Use the following steps to deploy a Docker compose file:

1. Connect to the existing cluster using the PowerShell command **Connect-ServiceFabricCluster**:

```
`Connect-ServiceFabricCluster -ConnectionEndpoint devopsdemowin.westeurope.cloudapp.azure.com:19000 -FindType FindByThumbprint -FindValue B00B6FF-39F5A50702AF3493B2C13237E80DE6734 -StoreName My -StoreLocation CurrentUser -X509Credential -ServerCertThumbprint B00B6FF39F5A50702AF3493B-2C13237E80DE6734`
```

Note: The values for this command will be different for your own cluster.

2. Deploy the `docker-compose.yml` file by using the following command:

```
`New-ServiceFabricComposeDeployment -DeploymentName ComposeDemo -ComposePath:\docker-compose.yml`
```

3. Use the following command to check the deployment status:

```
`Get-ServiceFabricComposeDeploymentStatus -DeploymentName ComposeDemo`
```

4. When the deployment has finished, you can navigate to your new service by opening a browser and navigating to the Service Fabric cluster domain name at port 80.

For example: **http://devopsdemowin.westeurope.cloudapp.azure.com**.

You should see the IIS information page running as a container on your Service Fabric cluster.

5. To remove a deployment, run the following command:

```
Remove-ServiceFabricComposeDeployment -DeploymentName ComposeDemo
```

Lab

Deploying a Dockerized Java app to Azure Web App for Containers

In this lab, **Deploying a Dockerized Java app to Azure Web App for Containers³⁵**, you will learn:

- Configuring a CI pipeline to build and publish Docker image
- Deploying to an Azure Web App for containers
- Configuring MySQL connection strings in the Web App

³⁵ <https://azuredevopslabs.com/labs/vstsextend/dockerjava/>

Module Review and Takeaways

Module Review Questions

Multiple choice

Which of the following Azure products provides management capabilities for applications that run across multiple Virtual Machines, and allows for the automatic scaling of resources, and load balancing of traffic?

- Azure Service Fabric
- Virtual Machine Scale Sets
- Azure Kubernetes Service
- Virtual Network

Checkbox

*Availability sets are made up of which of the following?
(choose two)*

- Update Domains
- Azure AD Domain Services
- Fault Domains
- Event Domains

Dropdown

Complete the following sentence.

Azure App Service is an Azure Platform-as-Service offering that is used for _____.

- processing events with serverless code.
- detecting, triaging, and diagnosing issues in your web apps and services.
- building, testing, releasing, and monitoring your apps from within a single software application.
- hosting web applications, REST APIs, and mobile back ends.

Checkbox

Which of the following are features of Web App for Containers?

(choose all that apply)

- Deploys containerized applications using Docker Hub, Azure Container Registry, or private registries.
- Incrementally deploys apps into production with deployment slots and slot swaps.
- Scales out automatically with auto-scale.
- Uses the App Service Log Streaming feature to allow you to see logs from your application.
- Supports PowerShell and Win-RM for remotely connecting directly into your containers.

Multiple choice

Which of the following statements is best practice for Azure Functions?

- Azure Functions should be stateful.
- Azure Functions should be stateless.

Checkbox

*Which of the following features are supported by Azure Service Fabric?
(choose all that apply)*

- Reliable Services
- Reliable Actor patterns
- Guest Executables
- Container processes

Checkbox

*Which of the following describe primary uses for Placement Constraints?
(choose all that apply)*

- Isolate workloads from each other
- Control which nodes in a cluster that a service can run on
- 'Lift and shift' an existing N-tier application into Azure Service Fabric.
- Describe resources that nodes have, and that services consume, when they are run on a node.

Checkbox

*Which of the following are network models for deploying a clusters in Azure Kubernetes Service (AKS)?
(choose two)*

- Basic Networking
- Native Model
- Advanced Networking
- Resource Model

Multiple choice

True or false: containers are a natural fit for an event-driven architecture?

- True
- False

Multiple choice

Which of the following cloud service models provides the most control, flexibility, and portability?

- Infrastructure-as-a-Service (IaaS)
- Functions-as-a-Service (FaaS)
- Platform-as-a-Service (PaaS)

Answers

Multiple choice

Which of the following Azure products provides management capabilities for applications that run across multiple Virtual Machines, and allows for the automatic scaling of resources, and load balancing of traffic?

- Azure Service Fabric
- Virtual Machine Scale Sets
- Azure Kubernetes Service
- Virtual Network

Explanation

Virtual Machine Scale Sets is the correct answer.

All other answers are incorrect.

Azure Service Fabric is for developing microservices and orchestrating containers on Windows or Linux.

Azure Kubernetes Service (AKS) simplifies the deployment, management, and operations of Kubernetes.

Virtual Network is for setting up and connecting virtual private networks.

With Azure VMs, scale is provided for by Virtual Machine Scale Sets (VMSS). Azure VMSS let you create and manage groups of identical, load balanced VMs. The number of VM instances can increase or decrease automatically, in response to demand or a defined schedule. Azure VMSS provide high availability to your applications, and allow you to centrally manage, configure, and update large numbers of VMs. With Azure VMSS, you can build large-scale services for areas such as compute, big data, and container workloads.

Checkbox

Availability sets are made up of which of the following?

(choose two)

- Update Domains
- Azure AD Domain Services
- Fault Domains
- Event Domains

Explanation

Update Domains and Fault Domains are the correct answers.

Azure AD Domain Services and Event Domains are incorrect answers.

Azure AD Domain Service provides managed domain services to a Windows Server Active Directory in Azure. An event domain is a tool for managing and publishing information.

Update Domains are a logical section of the datacenter, implemented by software and logic. When a maintenance event occurs (such as a performance update or critical security patch applied to the host), the update is sequenced through Update Domains. Sequencing updates by using Update Domains ensures that the entire datacenter does not fail during platform updates and patching.

Fault Domains provide for the physical separation of your workload across different hardware in the datacenter. This includes power, cooling, and network hardware that supports the physical servers located in server racks. If the hardware that supports a server rack becomes unavailable, only that specific rack of servers would be affected by the outage.

Dropdown

Complete the following sentence.

Azure App Service is an Azure Platform-as-Service offering that is used for _____.

- processing events with serverless code.
- detecting, triaging, and diagnosing issues in your web apps and services.
- building, testing, releasing, and monitoring your apps from within a single software application.
- hosting web applications, REST APIs, and mobile back ends.

Explanation

Hosting web applications, REST APIs, and mobile back ends, is the correct answer.

The other answers are incorrect because:

Processing events with serverless code is performed by Azure Functions.

Detecting, triaging, and diagnosing issues in your web apps and services is performed by Application Insights.

Building, testing, releasing, and monitoring your apps from within a single software application is performed by Visual Studio App Center.

Azure App Service is a Platform as Service offering on Azure, for hosting web applications, REST APIs, and mobile back ends. With Azure App Service you can create powerful cloud apps quickly within a fully managed platform. You can use Azure App Service to build, deploy, and scale enterprise-grade web, mobile, and API apps to run on any platform. Azure App Service ensures your application meet rigorous performance, scalability, security and compliance requirements, and benefit from using a fully managed platform for performing infrastructure maintenance.

Checkbox

Which of the following are features of Web App for Containers?

(choose all that apply)

- Deploys containerized applications using Docker Hub, Azure Container Registry, or private registries.
- Incrementally deploys apps into production with deployment slots and slot swaps.
- Scales out automatically with auto-scale.
- Uses the App Service Log Streaming feature to allow you to see logs from your application.
- Supports PowerShell and Win-RM for remotely connecting directly into your containers.

Explanation

All of the answers are correct.

Web App for Containers from the Azure App Service allows customers to use their own containers, and deploy them to Azure App Service as a web app. Similar to the Azure Web App solution, Web App for Containers eliminates time-consuming infrastructure management tasks during container deployment, updating, and scaling to help developers focus on coding and getting their apps to their end users faster. Furthermore, Web App for Containers provides integrated CI/CD capabilities with DockerHub, Azure Container Registry, and VSTS, as well as built-in staging, rollback, testing-in-production, monitoring, and performance testing capabilities to boost developer productivity.

For Operations, Web App for Containers also provides rich configuration features so developers can easily add custom domains, integrate with AAD authentication, add SSL certificates and more — all of which are crucial to web app development and management. Web App for Containers provides an ideal environment to run web apps that do not require extensive infrastructure control.

Multiple choice

Which of the following statements is best practice for Azure Functions?

- Azure Functions should be stateful.
- Azure Functions should be stateless.

Explanation

Azure Functions should be stateless is the correct answer.

Azure Functions should be stateful is an incorrect answer.

Azure Functions are an implementation of the Functions-as-a-Service programming model on Azure, with additional capabilities. It is best practice to ensure that your functions are as stateless as possible. Stateless functions behave as if they have been restarted, every time they respond to an event. You should associate any required state information with your data instead. For example, an order being processed would likely have an associated state member. A function could process an order based on that state, update the data as required, while the function itself remains stateless. If you require stateful functions, you can use the Durable Functions Extension for Azure Functions or output persistent data to an Azure Storage service.

Checkbox

Which of the following features are supported by Azure Service Fabric?

(choose all that apply)

- Reliable Services
- Reliable Actor patterns
- Guest Executables
- Container processes

Explanation

All of the answers are correct.

Reliable Services is a framework for creating services that use specific features provided by Azure Service Fabric. The two distinct types of Reliable Services you can create are stateless services and stateful services. *Reliable Actors* is a framework built on top of Reliable Services which implements the Virtual Actors design pattern. An Actor encapsulates a small piece of a state or behavior. The state of an Actor can be volatile, or it can be kept persistent in a distributed store. This store can be memory-based or on a disk.

Guest Executables are existing applications that you package and run as Service Fabric services (stateless). This makes the applications highly available, as Service Fabric keeps the instances of your applications running. Applications can be upgraded with no downtime, and Service Fabric can automatically roll back deployments if needed.

Containers can be run in a way that is similar to running guest executables. Furthermore, with containers, Service Fabric can restrict resource consumption per container (by CPU processes or memory usage, for example). Limiting resource consumption per service allows you to achieve higher densities on your cluster.

Checkbox

Which of the following describe primary uses for Placement Constraints?
(choose all that apply)

- Isolate workloads from each other
- Control which nodes in a cluster that a service can run on
- 'Lift and shift' an existing N-tier application into Azure Service Fabric.
- Describe resources that nodes have, and that services consume, when they are run on a node.

Explanation

The correct answers are: Isolate workloads from each other, control which nodes in a cluster that a service can run on, and 'Lift and shift' an existing N-tier application into Azure Service Fabric.

Describe resources that nodes have, and that services consume, when they are run on a node, is an incorrect answer. Metrics are used to describe resources that nodes have, and services consume, when they are run on a node.

Placement Constraints can control which nodes in a cluster that a service can run on. You can define any set of properties by node type, and then set constraints for them. Placement Constraints are primarily used to: Isolate workloads from each other; 'Lift and shift' an existing N-tier application into Azure Service Fabric; Run services on specific server configurations.

Placement Constraints can restrict Service Fabric's ability to balance overall cluster resource consumption. Make sure that your Placement Constraints are not too restrictive. Otherwise, if Service Fabric cannot comply with a Placement Constraint, your service will not run.

Checkbox

Which of the following are network models for deploying a clusters in Azure Kubernetes Service (AKS)?
(choose two)

- Basic Networking
- Native Model
- Advanced Networking
- Resource Model

Explanation

Basic Networking and Advanced Networking are correct answers.

Native Model and Resource Model are incorrect answers because these are two deployment models supported by Azure Service Fabric.

In AKS, you can deploy a cluster to use either Basic Networking or Advanced Networking. With Basic Networking, the network resources are created and configured as the AKS cluster is deployed. Basic Networking is suitable for small development or test workloads, as you don't have to create the virtual network and subnets separately from the AKS cluster. Simple websites with low traffic, or to lift and shift workloads into containers, can also benefit from the simplicity of AKS clusters deployed with Basic Networking.

With Advanced Networking, the AKS cluster is connected to existing virtual network resources and configurations. Advanced Networking allows for the separation of control and management of resources. When you use Advanced Networking, the virtual network resource is in a separate resource group to the AKS cluster. For most production deployments, you should plan for and use Advanced Networking.

Multiple choice

True or false: containers are a natural fit for an event-driven architecture?

- True
- False

Explanation

False is the correct answer.

True is an incorrect answer.

Architecture styles don't require the use of particular technologies, but some technologies are well-suited for certain architectures. For example, containers are a natural fit for microservices, and an event-driven architecture is generally best suited to IoT and real-time systems.

An N-tier architecture model is a natural fit for migrating existing applications that already use a layered architecture

A Web-queue-worker architecture model is suitable for relatively simple domains with some resource-intensive tasks.

The CQRS architecture model makes the most sense when it's applied to a subsystem of a larger architecture.

A Big data architecture model divides a very large dataset into chunks, performing parallel processing across the entire set, for analysis and reporting.

Finally, the Big compute architecture model, also called high-performance computing (HPC), makes parallel computations across a large number (thousands) of cores.

Multiple choice

Which of the following cloud service models provides the most control, flexibility, and portability?

- Infrastructure-as-a-Service (IaaS)
- Functions-as-a-Service (FaaS)
- Platform-as-a-Service (PaaS)

Explanation

Infrastructure-as-a-Service (IaaS) is the correct answer.

Functions-as-a-Service (FaaS) and Platform-as-a-Service (PaaS) are incorrect answers.

Of the three cloud service models mentioned, IaaS provides the most control, flexibility, and portability.

FaaS provides simplicity, elastic scale, and potential cost savings, because you pay only for the time your code is running. PaaS falls somewhere between the two.

Module 16 Create and Manage Kubernetes Service Infrastructure

Module Overview

Module Overview

As most modern software developers can attest, containers have provided engineering teams with dramatically more flexibility for running cloud-native applications on physical and virtual infrastructure. Containers package up the services comprising an application and make them portable across different compute environments, for both dev/test and production use. With containers, it's easy to quickly ramp application instances to match spikes in demand. And because containers draw on resources of the host OS, they are much lighter weight than virtual machines. This means containers make highly efficient use of the underlying server infrastructure.

So far so good. But though the container runtime APIs are well suited to managing individual containers, they're woefully inadequate when it comes to managing applications that might comprise hundreds of containers spread across multiple hosts. Containers need to be managed and connected to the outside world for tasks such as scheduling, load balancing, and distribution, and this is where a container orchestration tool like Kubernetes comes into its own.

An open source system for deploying, scaling, and managing containerized applications, Kubernetes handles the work of scheduling containers onto a compute cluster and manages the workloads to ensure they run as the user intended. Instead of bolting on operations as an afterthought, Kubernetes brings software development and operations together by design. By using declarative, infrastructure-agnostic constructs to describe how applications are composed, how they interact, and how they are managed, Kubernetes enables an order-of-magnitude increase in operability of modern software systems.

Kubernetes was built by Google based on its own experience running containers in production, and it surely owes much of its success to Google's involvement. The Kubernetes platform is open source and growing dramatically through open source contributions at a very rapid pace. Kubernetes marks a breakthrough for devops because it allows teams to keep pace with the requirements of modern software development.

Learning Objectives

After completing this module, students will be able to:

- Deploy and configure a Managed Kubernetes cluster

Azure Kubernetes Service (AKS)

Kubernetes Overview

Kubernetes is a cluster orchestration technology that originated with Google. Sometimes referred to as *k8s*, it's an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts. This creates a container-centric infrastructure.



There are several other container cluster orchestration technologies available such as **Mesosphere DC/OS**¹ and **Docker Swarm**².

For more details about Kubernetes, go to **Production-Grade Container Orchestration**³ on the Kubernetes website.

Azure Kubernetes Service (AKS)

AKS is Microsoft's implementation of Kubernetes. AKS makes it easier to deploy a managed Kubernetes cluster in Azure. It also reduces the complexity and operational overhead of managing Kubernetes, by offloading much of that responsibility to Azure.



AKS manages much of the Kubernetes resources for the end user, making it quicker and easier to deploy and manage containerized applications without container orchestration expertise. It also eliminates the burden of ongoing operations and maintenance by provisioning, upgrading, and scaling resources on demand without taking applications offline.

Azure AKS manages the following aspects of a Kubernetes cluster for you:

- It manages critical tasks such as health monitoring and maintenance, such as Kubernetes version upgrades and patching.
- It performs simple cluster scaling.
- It enables master nodes to be fully managed by Microsoft.
- It leaves you responsible only for managing and maintaining the agent nodes.
- It ensures master nodes are free, and you only pay for running agent nodes.

AKS Architectural components

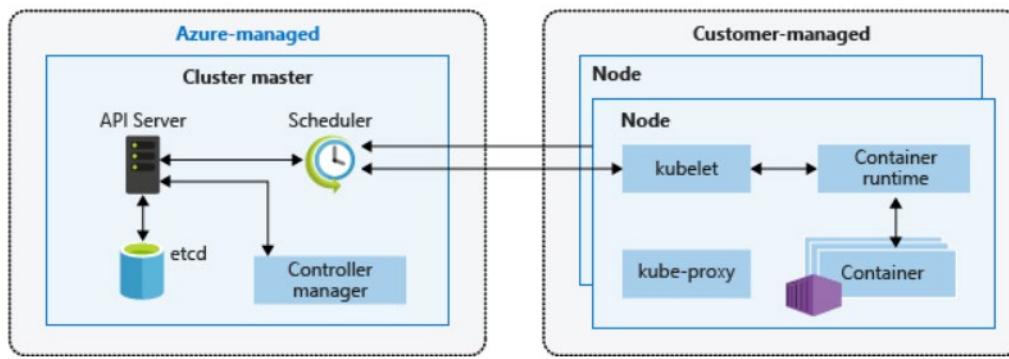
A Kubernetes cluster is divided into two components:

- Cluster master nodes, which provide the core Kubernetes services and orchestration of application workloads.
- Nodes that run your application workloads.

¹ <https://mesosphere.com/product/>

² <https://www.docker.com/products/orchestration>

³ <https://kubernetes.io/>



Cluster master

When you create an AKS cluster, a cluster master is automatically created and configured. This cluster master is provided as a managed Azure resource abstracted from the user. There is no cost for the cluster master, only the nodes that are part of the AKS cluster.

The cluster master includes the following core Kubernetes components:

- **kube-apiserver.** The API server is how the underlying Kubernetes APIs are exposed. This component provides the interaction for management tools such as kubectl or the Kubernetes dashboard.
- **etcd.** To maintain the state of your Kubernetes cluster and configuration, the highly available etcd is a key value store within Kubernetes.
- **kube-scheduler.** When you create or scale applications, the Scheduler determines what nodes can run the workload, and starts them.
- **kube-controller-manager.** The Controller Manager oversees a number of smaller controllers that perform actions such as replicating pods and managing node operations.

Nodes and node pools

To run your applications and supporting services, you need a Kubernetes node. An *AKS cluster*, which is an Azure virtual machine (VM) that runs the Kubernetes node components and container runtime, contains one or more nodes:

- The *kubelet* is the Kubernetes agent that processes the orchestration requests from the cluster master, and scheduling of running the requested containers.
- Virtual networking is handled by the *kube-proxy* on each node. The proxy routes network traffic and manages IP addressing for services and pods.
- The *container runtime* is the component that allows containerized applications to run and interact with additional resources such as the virtual network and storage. In AKS, Docker is used as the container runtime.

Nodes of the same configuration are grouped together into *node pools*. A Kubernetes cluster contains one or more node pools. The initial number of nodes and size are defined when you create an AKS cluster, which creates a default node pool. This default node pool in AKS contains the underlying VMs that run your agent nodes.

Pods

Kubernetes uses pods to run an instance of your application. A pod represents a single instance of your application. Pods typically have a 1:1 mapping with a container, although there are advanced scenarios where a pod might contain multiple containers. These multi-container pods are scheduled together on the same node, and allow containers to share related resources.

When you create a pod, you can define resource limits to request a certain amount of CPU or memory resources. The Kubernetes Scheduler attempts to schedule the pods to run on a node with available resources to meet the request. You can also specify maximum resource limits that prevent a given pod from consuming too much compute resource from the underlying node.

✓ Note: A best practice is to include resource limits for all pods to help the Kubernetes Scheduler understand what resources are needed and permitted.

A pod is a logical resource, but the container (or containers) is where the application workloads run. Pods are typically ephemeral, disposable resources. Therefore, individually scheduled pods miss some of the high availability and redundancy features Kubernetes provides. Instead, pods are usually deployed and managed by Kubernetes controllers, such as the Deployment controller.

Kubernetes networking

Kubernetes pods have limited lifespan, and are replaced whenever new versions are deployed. Settings such as the IP address change regularly, so interacting with pods by using an IP address is not advised. This is why Kubernetes services exist. To simplify the network configuration for application workloads, Kubernetes uses Services to logically group a set of pods together and provide network connectivity.

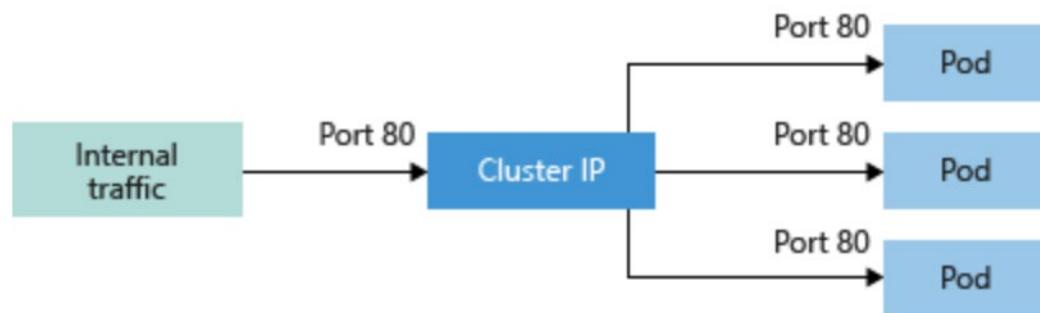
Kubernetes Service is an abstraction that defines a logical set of pods, combined with a policy that describes how to access them. Where pods have a shorter lifecycle, services are usually more stable and are not affected by container updates. This means that you can safely configure applications to interact with pods through the use of services. The service redirects incoming network traffic to its internal pods. Services can offer more specific functionality, based on the service type that you specify in the Kubernetes deployment file.

If you do not specify the service type, you will get the default type, which is `ClusterIP`. This means that your services and pods will receive virtual IP addresses that are only accessible from within the cluster. Although this might be a good practice for containerized back-end applications, it might not be what you want for applications that need to be accessible from the internet. You need to determine how to configure your Kubernetes cluster to make those applications and pods accessible from the internet.

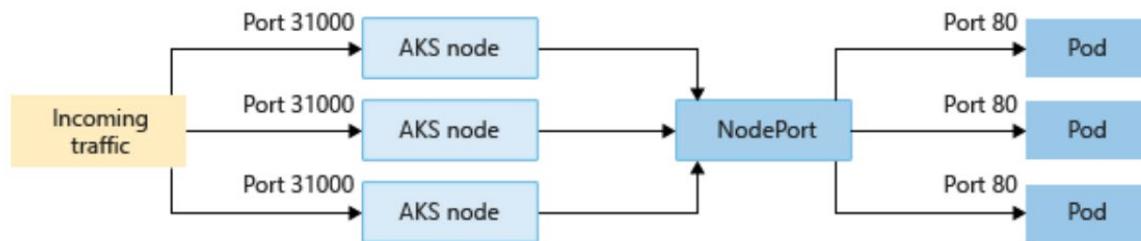
Services

The following Service types are available:

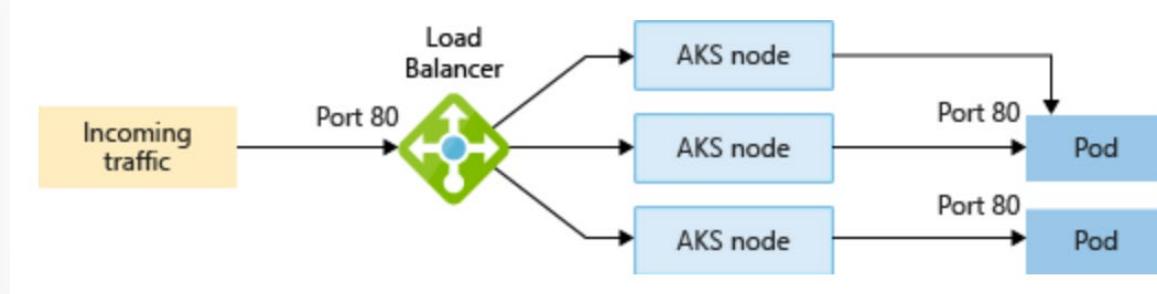
- Cluster IP. This service creates an internal IP address for use within the AKS cluster. However, it's good for internal-only applications that support other workloads within the cluster.



- NodePort. This service creates a port mapping on the underlying node, which enables the application to be accessed directly with the node IP address and port.



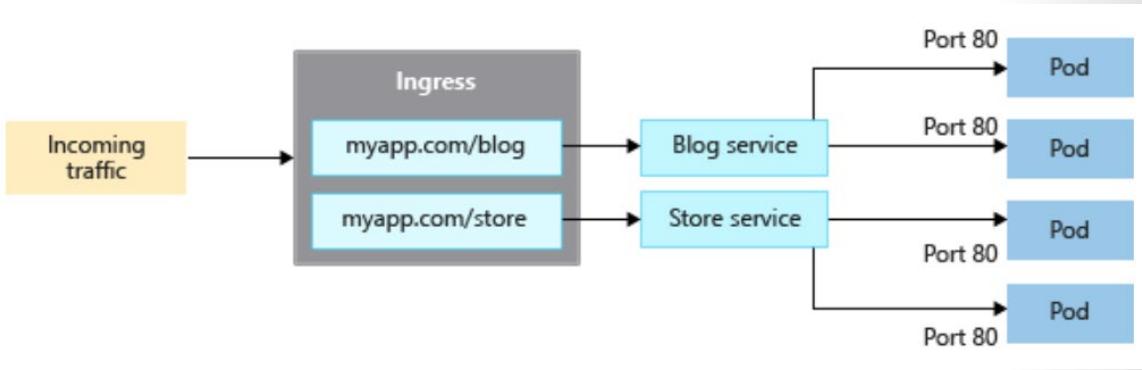
- Load Balancer. this service creates an Azure Load Balancer resource, configures an external IP address, and connects the requested pods to the load balancer backend pool. To allow customers traffic to reach the application, load balancing rules are created on the desired ports.



Ingress controllers

When you create a Load Balancer-type Service, an underlying Azure Load Balancer resource is created. The load balancer is configured to distribute traffic to the pods in your service on a given port. The Load Balancer only works at layer 4. The Service is unaware of the actual applications, and can't make any additional routing considerations.

Ingress controllers work at layer 7, and can use more intelligent rules to distribute application traffic. A common use of an Ingress controller is to route HTTP traffic to different applications based on the inbound URL.



There are different implementations of the `Ingress Controller` concept. One example is the `Nginx Ingress Controller`, which translates the `Ingress Resource` into a `nginx.conf` file. Other examples are the `ALB Ingress Controller` (AWS) and the `GCE Ingress Controllers` (Google Cloud), which make use of cloud native resources. Using the `Ingress` setup within `Kubernetes` makes it possible to easily switch the reverse proxy implementation so that your containerized workload leverages the most out of the cloud platform on which it is running.

Azure virtual networks

In AKS, you can deploy a cluster that uses one of the following two network models:

- Basic networking. The network resources are created and configured when the AKS cluster is deployed.
- Advanced networking. The AKS cluster is connected to existing virtual network resources and configurations.

Deployment

`Kubernetes` uses the term *pod* to package applications. A *pod* is a deployment unit, and it represents a running process on the cluster. It consists of one or more containers, and configuration, storage resources, and networking support. *Pods* are usually created by a controller, which monitors it and provides self-healing capabilities at the cluster level.

Pods are described by using YAML or JSON. *Pods* that work together to provide functionality are grouped into services to create *microservices*. For example, a front-end *pod* and a back-end *pod* could be grouped into one service.

You can deploy an application to `Kubernetes` by using the **kubectl** CLI, which can manage the cluster. By running **kubectl** on your build agent, it's possible to deploy `Kubernetes` pods from Azure DevOps. It's also possible to use the management API directly. There is also a specific `Kubernetes` task called Deploy To `Kubernetes` that is available in Azure DevOps. More information about this will be covered in the upcoming demonstration.

Continuous delivery

To achieve continuous delivery, the build-and-release pipelines are run for every check-in on the Source repository.

Demonstration-Deploying and connecting to an AKS cluster

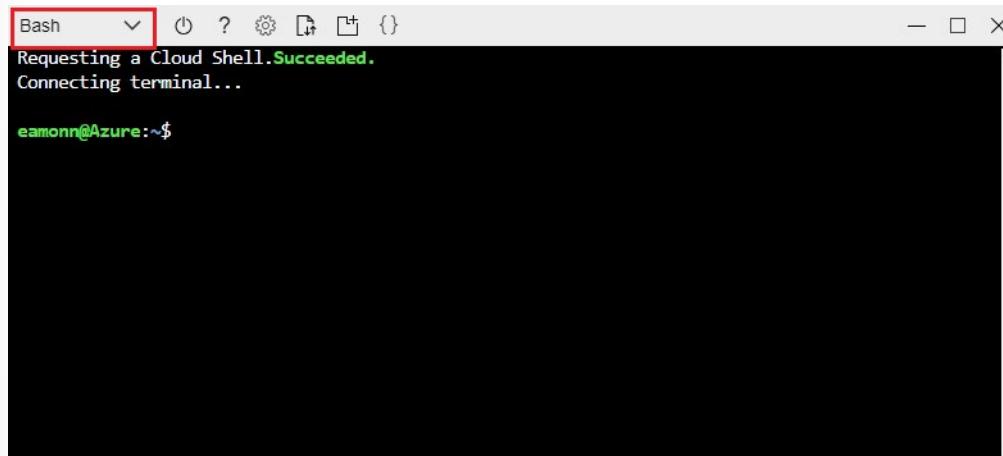
This walkthrough shows how to deploy an AKS cluster using the Azure CLI. A multi-container application that includes a web front end and a Redis Cache instance is run in the cluster. You then see how to monitor the health of the cluster and pods that run your application

Prerequisites

- Use the cloud shell.
- You require an Azure subscription to be able to perform these steps. If you don't have one, you can create it by following the steps outlined on the [Create your Azure free account today⁴](#) page.

Steps

1. Open Azure Cloud Shell by going to <https://shell.azure.com>, or using the Azure Portal and selecting **Bash** as the environment option.



2. Create an Azure resource group by running the following command:

```
az group create --name myResourceGroup --location < datacenter nearest you  
>
```

3. Create an AKS cluster by running the following command:

```
az aks create \  
    --resource-group myResourceGroup \  
    --name myAKScluster \  
    --node-count 1 \  
    --enable-addons monitoring \  
    --generate-ssh-keys
```

⁴ https://azure.microsoft.com/en-us/free/?ref=microsoft.com&utm_source=microsoft.com&utm_medium=docs&utm_campaign=visualstudio

After a few minutes, the command completes and returns JSON-formatted information about the cluster.

4. To manage a Kubernetes cluster, you use `kubectl`, the Kubernetes command-line client. If you use Azure Cloud Shell, `kubectl` is already installed. To install `kubectl` locally, use the following command:

```
az aks install-cli
```

5. To configure `kubectl` to connect to your Kubernetes cluster, use the `az aks get-credentials` command. This command downloads credentials and configures the Kubernetes CLI to use them:

```
az aks get-credentials --resource-group myResourceGroup --name myAKSCluster
```

6. Verify the connection to your cluster by running the following command. Make sure that the status of the node is Ready:

```
kubectl get nodes
```

7. Create a file named **azure-vote.yaml**, and then copy it into the following YAML definition. If you use the Azure Cloud Shell, you can create this file using **vi** or **nano** as if working on a virtual or physical system:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-back
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-back
  template:
    metadata:
      labels:
        app: azure-vote-back
    spec:
      containers:
        - name: azure-vote-back
          image: redis
          resources:
            requests:
              cpu: 100m
              memory: 128Mi
            limits:
              cpu: 250m
              memory: 256Mi
          ports:
            - containerPort: 6379
              name: redis
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-back
```

```
spec:
  ports:
  - port: 6379
    selector:
      app: azure-vote-back
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-front
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-front
  template:
    metadata:
      labels:
        app: azure-vote-front
    spec:
      containers:
      - name: azure-vote-front
        image: microsoft/azure-vote-front:v1
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 250m
            memory: 256Mi
        ports:
        - containerPort: 80
        env:
        - name: REDIS
          value: "azure-vote-back"
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-front
spec:
  type: LoadBalancer
  ports:
  - port: 80
    selector:
      app: azure-vote-front
```

8. Deploy the application by running the following command:

```
kubectl apply -f azure-vote.yaml
```

You should receive output showing the Deployments and Services were created successfully after it runs as per the below.

```
deployment "azure-vote-back" created
service "azure-vote-back" created
deployment "azure-vote-front" created
service "azure-vote-front" created
```

9. When the application runs, a Kubernetes service exposes the application front end to the internet. This process can take a few minutes to complete. To monitor progress run the command

```
kubectl get service azure-vote-front --watch
```

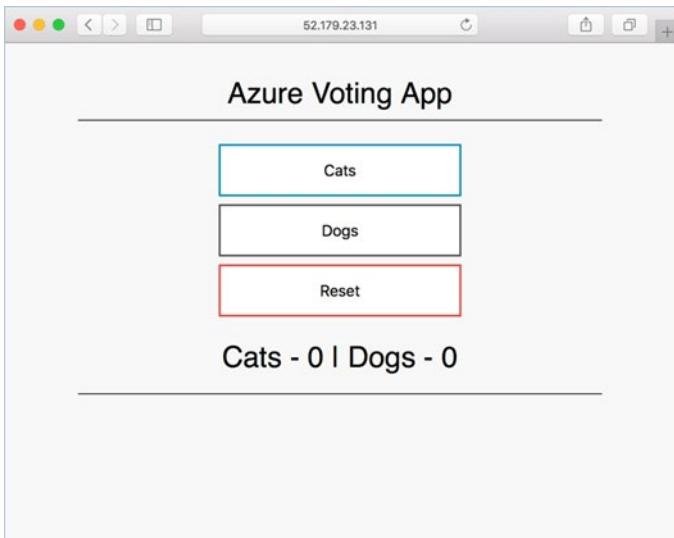
10. Initially the **EXTERNAL-IP** for the azure-vote-front service is shown as pending.

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT (S)
AGE				
azure-vote-front	LoadBalancer	10.0.37.27	< pending >	80:30572/TCP

11. When the **EXTERNAL-IP** address changes from pending to an actual public IP address, use **CTRL-C** to stop the `kubectl` watch process. The following example output shows a valid public IP address assigned to the service:

```
azure-vote-front    LoadBalancer    10.0.37.27    52.179.23.131    80:30572/TCP
2m
```

12. To see the Azure Vote app in action, open a web browser to the external IP address of your service.



Monitor health and logs. When the AKS cluster was created, Azure Monitor for containers was enabled to capture health metrics for both the cluster nodes and pods. These health metrics are available in the Azure portal. To see current status, uptime, and resource usage for the Azure Vote pods, complete the following steps in the Azure portal:

13. Open a web browser to the Azure portal <https://portal.azure.com>.

14. Select your resource group, such as myResourceGroup, then select your AKS cluster, such as myAKS-Cluster.

15. Under Monitoring on the left-hand side, choose Insights
16. Across the top, choose to + Add Filter
17. Select Namespace as the property, then choose < All but kube-system >
18. Choose to view the Containers. The azure-vote-back and azure-vote-front containers are displayed, as shown in the following example:

The screenshot shows the Azure Monitor Insights blade for the 'myAKSCluster - Insights' resource. On the left sidebar, under the 'Monitoring' section, 'Insights' is selected. In the main area, the 'Containers' tab is active. A table displays two items:

NAME	STATUS	EST... RST...	95TH POD	NODE	RESTA... 0	UPTIME	TREND 95TH % (1 BAR = 15M)
azure-vote-back	OK	0.3%	0.7 ms	azure-vote-b...	skos-nodepool...	5 mins	
azure-vote-front	OK	0.1%	0.2 ms	azure-vote-fr...	skos-nodepool...	4 mins	

To the right of the table, a detailed view for the 'azure-vote-back' container is shown:

- Container**: azure-vote-back
- View container logs (preview)**
- Container Name**: azure-vote-back
- Container ID**: 6e2893a24018e2bda9bf4353bf923934889ec8562e94f43b6b8b30d213044
- Container Status**: running
- Image**: redis
- Image Tag**: latest
- Container Creation Time Stamp**: 12/18/2018, 10:54:08 AM
- Start Time**: 12/18/2018, 10:54:08 AM
- Finish Time**: -
- CPU Limit**: 250 ms
- CPU Request**: 100 ms
- Memory Limit**: 256 MB
- Memory Request**: 128 MB

19. To see logs for the azure-vote-front pod, select the View container logs link on the right-hand side of the containers list. These logs include the stdout and stderr streams from the container

The screenshot shows the Azure Log Analytics workspace for the 'defaultworkspace-19da35d3-9a1a...' workspace. The 'Logs' tab is active. A query is displayed in the editor:

```
let startDateTime = datetime('2018-12-18T12:45:00.000Z');
let endDateTime = datetime('2018-12-18T18:59:37.982Z');
let ContainerName = '37d12cec-02f9-11e9-8d50-9e4fb98784de/azure-vote-back';
let ContainerID = ContainerName + '_ContainerID';
ContainerID | where TimeGenerated >= startDateTime and TimeGenerated <= endDateTime
| where ClusterName == "myAKSCluster"
| distinct ContainerID
| project LogEntry
| where TimeGenerated >= startDateTime and TimeGenerated <= endDateTime
ContainerID | order by TimeGenerated desc
| render table
```

The results pane shows a table of log entries for the 'azure-vote-front' container. The table has columns: LogEntrySource, LogEntry, TimeGenerated [UTC], Computer, and Image. There are 8 records listed.

LogEntrySource	LogEntry	TimeGenerated [UTC]	Computer	Image
> stdout	1:M 18 Dec 2018 18:54:08.774 * Server initialized	2018-12-18T18:54:08.774	aks-nodepool1-317:8369-0	redis:latest
> stdout	1:M 18 Dec 2018 18:54:08.774 * Ready to accept connections	2018-12-18T18:54:08.774	aks-nodepool1-317:8369-0	redis:latest
> stdout	1:M 18 Dec 2018 18:54:08.774 # WARNING you have Transparent Hu...	2018-12-18T18:54:08.774	aks-nodepool1-317:8369-0	redis:latest
> stdout	1:M 18 Dec 2018 18:54:08.774 # WARNING: The TCP backlog setting ...	2018-12-18T18:54:08.774	aks-nodepool1-317:8369-0	redis:latest
> stdout	1:M 18 Dec 2018 18:54:08.774 # Warning: no config file specified, us...	2018-12-18T18:54:08.774	aks-nodepool1-317:8369-0	redis:latest
> stdout	1:C 18 Dec 2018 18:54:08.773 # Redis version=5.0.3, bits=64, comm...	2018-12-18T18:54:08.773	aks-nodepool1-317:8369-0	redis:latest
> stdout	1:C 18 Dec 2018 18:54:08.773 # o0000000000 Redis is starting o...	2018-12-18T18:54:08.773	aks-nodepool1-317:8369-0	redis:latest

- ✓ Note: If you are not continuing to use the Azure resources, remember to delete them to avoid incurring costs.

Continuous Deployment

In Kubernetes you can update the service by using a rolling update. This will ensure that traffic to a container is first drained, then the container is replaced, and finally, traffic is sent back again to the container. In the meantime, your customers won't see any changes until the new containers are up and running on the cluster. The moment they are, new traffic is routed to the new containers and stopped to the old containers. Running a rolling update is easy to do with the following command:

```
kubectl apply -f nameofyamlfile
```

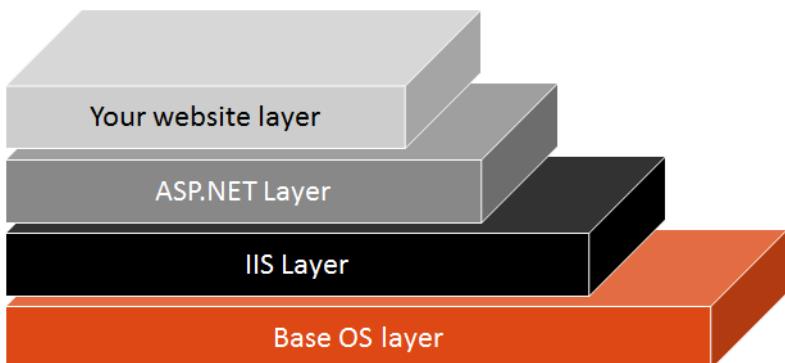
The YAML file contains a specification of the deployment. The **apply** command is convenient because it makes no difference whether the deployment was already on the cluster. This means that you can always use the exact same steps regardless of whether you are doing an initial deployment or an update to an existing deployment.

When you change the name of the image for a service in the YAML file Kubernetes will apply a rolling update, taking into account the minimum number of running containers you want and how many at a time it is allowed to stop. The cluster will take care of updating the images without downtime, assuming that your application container is built stateless.

Updating Images

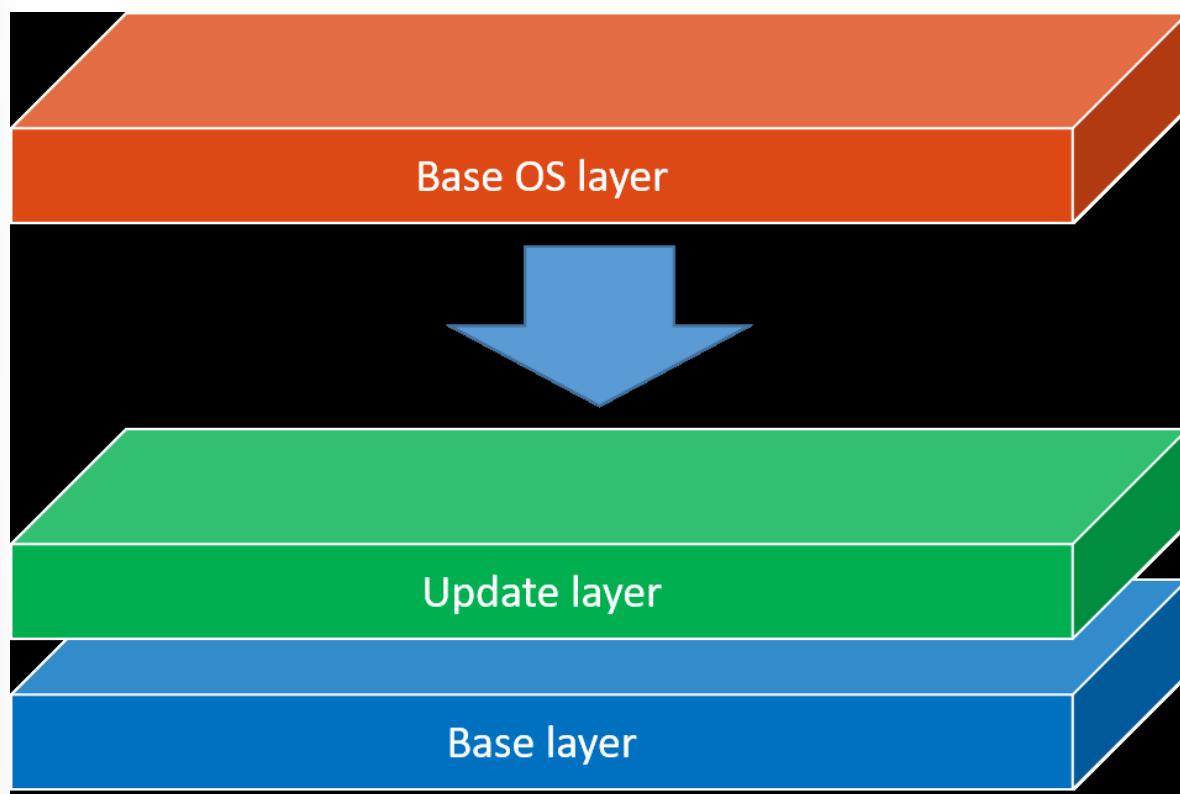
After you've successfully containerized your application, you'll need ensure that you update your image regularly. This entails creating a new image for every change you make in your own code, and ensuring that all layers receive regular patching.

A large part of a container image is the base OS layer, which contains the elements of the operating system that are not shared with the container host.



The base OS layer gets updated frequently. Other layers, such as the IIS layer and ASP.NET layer in the image are also updated. Your own images are built on top of these layers, and it's up to you to ensure that they incorporate those updates.

Fortunately, the base OS layer actually consists of two separate images: a larger base layer and a smaller update layer. The base layer changes less frequently than the update layer. Updating your image's base OS layer is usually a matter of getting the latest update layer.



If you're using a Docker file to create your image, patching layers should be done by explicitly changing the image version number using the following commands:

```
```yml
FROM microsoft/windowsservercore:10.0.14393.321
RUN cmd /c echo hello world
```
into

```yml
FROM microsoft/windowsservercore:10.0.14393.693
RUN cmd /c echo hello world
````
```

When you build this Docker file, it now uses version 10.0.14393.693 of the image `microsoft/windowsservercore`.

Latest tag

Don't be tempted to rely on the latest tag. To define repeatable custom images and deployments, you should always be explicit about the base image versions that you are using. Also, just because an image is tagged as the latest doesn't mean that it actually is the latest. The owner of the image needs to ensure this.

- ✓ Note: The last two segments of the version number of Windows Server Core and Nano images will match the build number of the operating system inside.

Lab

Deploying a multi-container application to Azure Kubernetes Services

Azure Kubernetes Service (AKS) is the quickest way to use Kubernetes on Azure. Azure Kubernetes Service (AKS) manages your hosted Kubernetes environment, making it quick and easy to deploy and manage containerized applications without container orchestration expertise. It also eliminates the burden of ongoing operations and maintenance by provisioning, upgrading, and scaling resources on demand, without taking your applications offline. Azure DevOps helps in creating Docker images for faster deployments and reliability using the continuous build option.

One of the biggest advantage to use AKS is that instead of creating resources in cloud you can create resources and infrastructure inside Azure Kubernetes Cluster through Deployments and Services manifest files.

In this lab, **Deploying a multi-container application to Azure Kubernetes Services⁵**, you will learn:

- Setting up an AKS Cluster
- CI/CD Pipeline for building artifacts and deploying to Kubernetes
- Access the Kubernetes web dashboard in Azure Kubernetes Service (AKS)

⁵ <https://azuredavolabs.com/labs/vstsextend/kubernetes/#access-the-kubernetes-web-dashboard-in-azure-kubernetes-service-aks>

Module Review and Takeaways

Module Review Questions

Multiple choice

Is this statement true or false?

Azure Policy natively integrates with AKS, allowing you to enforce rules across multiple AKS clusters. Track, validate and configure nodes, pods and container images for compliance.

- True
- False

Multiple choice

Kubernetes CLI is called?

- HELM
- ACI
- AKS
- KUBECTL

Checkbox

For workloads running in AKS Kubernetes Web Dashboard allows you to view _____. Select all that apply.

- Config Map & Secrets
- Logs
- Storage
- Azure Batch Metrics

Checkbox

Pods can be described using which of the following languages? Select all that apply.

- JSON
- XML
- PowerShell
- YAML

Answers

Multiple choice

Is this statement true or false?

Azure Policy natively integrates with AKS, allowing you to enforce rules across multiple AKS clusters.

Track, validate and configure nodes, pods and container images for compliance.

- True
- False

Multiple choice

Kubernetes CLI is called?

- HELM
- ACI
- AKS
- KUBECTL

Checkbox

For workloads running in AKS Kubernetes Web Dashboard allows you to view _____.

Select all that apply.

- Config Map & Secrets
- Logs
- Storage
- Azure Batch Metrics

Checkbox

Pods can be described using which of the following languages? Select all that apply.

- JSON
- XML
- PowerShell
- YAML

Module 17 Third Party Infrastructure as Code Tools available with Azure

Module Overview

Module Overview

Configuration management tools enable changes and deployments to be faster, repeatable, scalable, predictable, and able to maintain the desired state, which brings controlled assets into an expected state.

Some advantages of using configuration management tools include:

- Adherence to coding conventions that make it easier to navigate code
- Idempotency, which means that the end state remains the same, no matter how many times the code is executed
- Distribution design to improve managing large numbers of remote servers

Some configuration management tools use a pull model, in which an agent installed on the servers runs periodically to pull the latest definitions from a central repository and apply them to the server. Other tools use a push model, where a central server triggers updates to managed servers.

Configuration management tools enables the use of tested and proven software development practices for managing and provisioning data centers in real-time through plaintext definition files.

Learning Objectives

After completing this module, students will be able to:

- Deploy and configure infrastructure using 3rd party tools and services with Azure, such as Chef, Puppet, Ansible, and Terraform

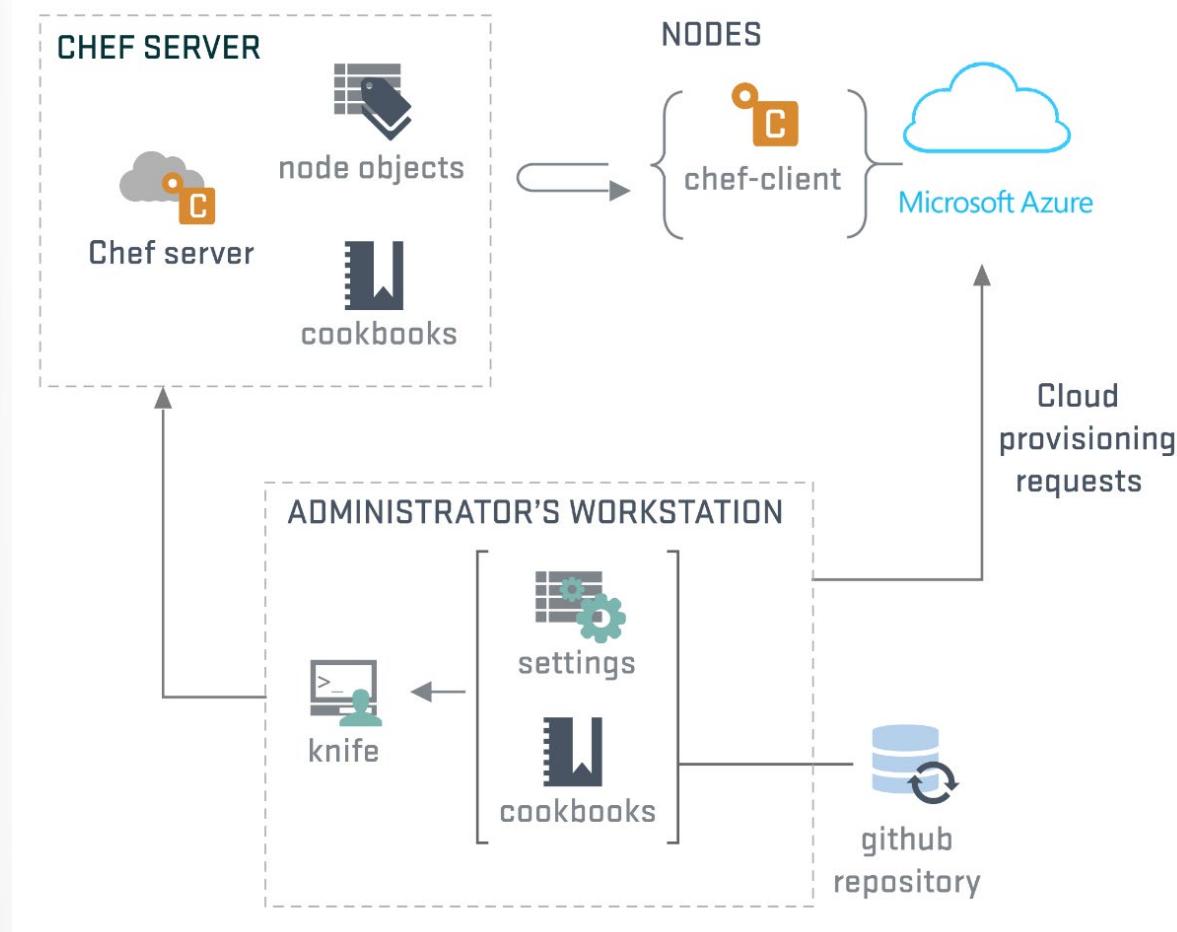
Chef

What is Chef

Chef is an infrastructure automation tool that you use for deploying, configuring, managing, and ensuring compliance of applications and infrastructure. It provides for a consistent deployment and management experience.

Chef helps you to manage your infrastructure in the cloud, on-premises, or in a hybrid environment by using instructions (or *recipes*) to configure nodes. A *node*, or chef-client is any physical or virtual machine (VM), cloud, or network device that is under management by Chef.

The following diagram is of the high-level Chef architecture:



Chef components

Chef has three main architectural components:

- Chef Server. This is the management point. There are two options for the Chef Server: a hosted solution and an on-premises solution.
- Chef Client (node). This is a Chef agent that resides on the servers you are managing.

- Chef Workstation. This is the Admin workstation where you create policies and execute management commands. You run the **knife** command from the Chef Workstation to manage your infrastructure.

Chef also uses concepts called *cookbooks* and *recipes*. Chef cookbooks and recipes are essentially the policies that you define and apply to your servers.

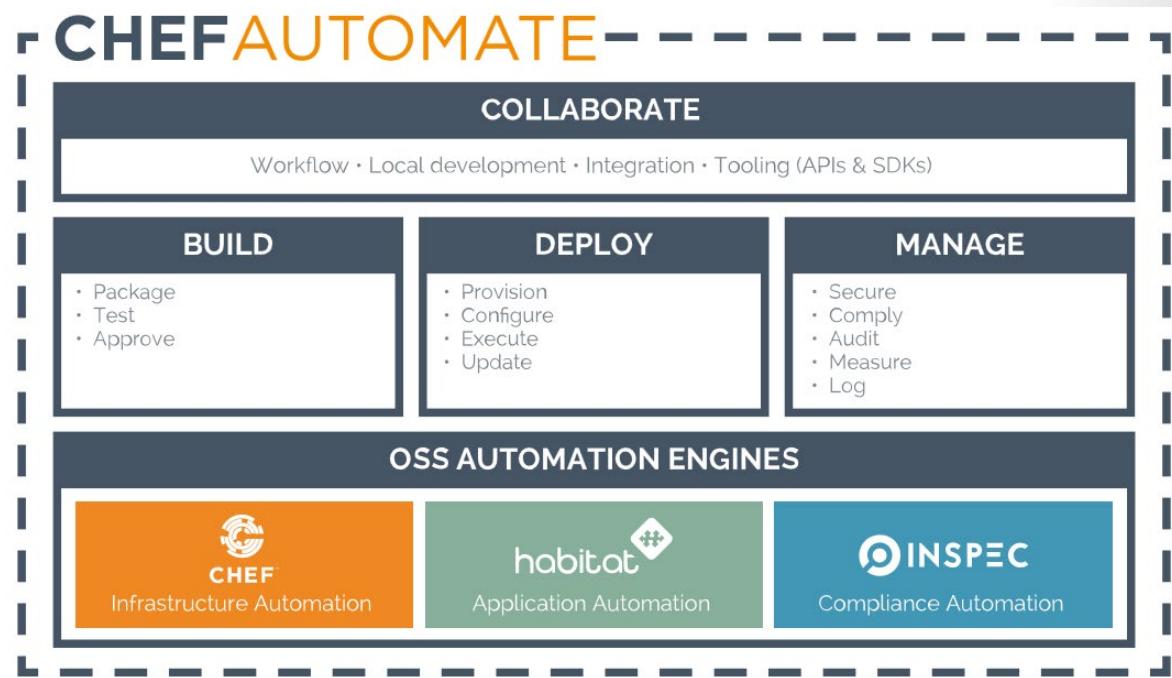
Chef Automate

You can deploy Chef on Microsoft Azure from the Azure Marketplace using the Chef Automate image. *Chef Automate* is a Chef product that allows you to package and test your applications, and provision and update your infrastructure. Using Chef, you can manage changes to your applications and infrastructure using compliance and security checks, and dashboards that give you visibility into your entire stack.

The Chef Automate image is available on the Azure Chef Server and has all the functionality of the legacy Chef Compliance server. You can build, deploy, and manage your applications and infrastructure on Azure. Chef Automate is available from the Azure Marketplace, and you can try it out with a free 30-day license. You can deploy it in Azure straight away.

Chef Automate structure and function

Chef Automate integrates with the open-source products Chef, Chef InSpec, Chef Habitat, and their associated tools, including chef-client and ChefDK. The following image is an overview of the structure of Chef Automate, and how it functions.



Let's break down the Chef Automate architecture components:

- **Habitat** is an open-source project that offers an entirely new approach to application management. It makes the application and its automation the unit of deployment by creating platform-independent build artifacts that can run on traditional servers and virtual machines (VMs). They also can be exported into your preferred container platform, enabling you to deploy your applications in any environment. When applications are wrapped in a lightweight habitat (the runtime environment), whether the

habitat is a container, a bare metal machine, or platform as a service (PaaS) is no longer the focus and does not constrain the application.

For more information about Habitat, go to **Use Habitat to deploy your application to Azure¹**.

- **InSpec** is a free and open-source framework for testing and auditing your applications and infrastructure. InSpec works by comparing the actual state of your system with the desired state that you express in easy-to-read and easy-to-write InSpec code. InSpec detects violations and displays findings in the form of a report, but you are in control of remediation.

You can use InSpec to validate the state of your VMs running in Azure. You can also use InSpec to scan and validate the state of resources and resource groups inside a subscription.

More information about InSpec is available at **Use InSpec for compliance automation of your Azure infrastructure²**.

Chef Cookbooks

Chef uses a **cookbook** to define a set of commands that you execute on your managed client. A *cookbook* is a set of tasks that you use to configure an application or feature. It defines a scenario, and everything required to support that scenario. Within a cookbook, there are a series of *recipes*, which define a set of actions to perform. Cookbooks and recipes are written in the Ruby language.

After you create a cookbook, you can then create a Role. A *Role* defines a baseline set of cookbooks and attributes that you can apply to multiple servers. To create a cookbook, you use the **chef generate cookbook** command.

Create a cookbook

Before creating a cookbook, you first configure your Chef workstation by setting up the Chef Development Kit on your local workstation. You'll use the Chef workstation to connect to, and manage your Chef server.

- ✓ Note: You can download and install the Chef Development Kit from **Chef downloads³**.

Choose the Chef Development Kit that is appropriate to your operating system and version. For example:

- macOS/macOS
- Debian
- Red Hat Enterprise Linux SUSE
- Linux Enterprise Server
- Ubuntu
- Windows

1. Installing the Chef Development Kit creates the Chef workstation automatically in your **C:\Chef** directory. After installation completes, run the following example command that calls the Cookbook web server for a policy that automatically deploys IIS:

```
chef generate cookbook webserver
```

¹ <https://docs.microsoft.com/en-us/azure/chef/chef-habitat-overview>

² <https://docs.microsoft.com/en-us/azure/chef/chef-inspec-overview>

³ <https://downloads.chef.io/chefdk>

This command generates a set of files under the directory **C:\Chef\cookbooks\webserver**. Next, you need to define the set of commands that you want the Chef client to execute on your managed VM. The commands are stored in the **default.rb** file.

2. For this example, we will define a set of commands that installs and starts Microsoft Internet Information Services (IIS), and copies a template file to the **wwwroot** folder. Modify the **C:\chef\cookbooks\webserver\recipes\default.rb** file by adding the following lines:

```
powershell_script 'Install IIS' do  
  
  action :run  
  
  code 'add-windowsfeature Web-Server'  
  
  end  
  
  service 'w3svc' do  
  
    action [ :enable, :start ]  
  
    end  
  
  template 'c:\inetpub\wwwroot\Default.htm' do  
  
    source 'Default.htm.erb'  
  
    rights :read, 'Everyone'  
  
    end
```

3. Save the file after you are done.
4. To generate the template, run the following command:

```
chef generate template webserver Default.htm
```

5. Now navigate to the **C:\chef\cookbooks\webserver\templates\default\Default.htm.erb** file. Edit the file by adding some simple **Hello World** HTML code, and then save the file.
6. Run the following command to upload the cookbook to the Chef server so that it appears under the **Policy** tab:

```
chef generate template webserver Default.htm
```

We have now created our cookbook and it's ready to use.

7. The next steps (which we will not be covering in detail at this time) would be to:
 - Create a role to define a baseline set of cookbooks and attributes that you can apply to multiple servers.
 - Create a node to deploy the configuration to the machine you want to configure.
 - Bootstrap the machine using Chef to add the role to the node that deployed the configuration to the machine.

Chef Knife command

Knife is a command that's available from the command line. It's made available as part of the Chef Development Kit installation. You can use the **Knife** command to complete a wide variety of tasks, such as:

- Generate a cookbook template. You do this by running the following command:

```
chef generate cookbook < cookbook name >
```

- Upload your cookbooks and recipes to the Chef Automate server using the following command:

```
knife cookbook upload < cookbook name> --include-dependencies
```

- Create a role to define a baseline set of cookbooks and attributes that you can apply to multiple servers. Use the following command to create this role:

```
knife role create < role name >
```

- Bootstrap the a node or client and assign a role using the following command:

```
knife bootstrap < FQDN-for-App-VM > --ssh-user <app-admin-username>
--ssh-password <app-vm-admin-password> --node-name < node name > --run-
list role[ < role you defined > ] --sudo --verbose
```

You can also bootstrap Chef VM extensions for the Windows and Linux operating systems, in addition to provisioning them in Azure using the **Knife** command. For more information, look up the 'cloud-api' bootstrap option in the Knife plugin documentation at <https://github.com/chef/knife-azure>⁴.

- ✓ Note: You can also install the Chef extensions to an Azure VM using Windows PowerShell. By installing the Chef Management Console, you can manage your Chef server configuration and node deployments via a browser window.

⁴ <https://github.com/chef/knife-azure>

Puppet

What is Puppet

Puppet is a deployment and configuration management toolset that provides you with enterprise tools that you need to automate an entire lifecycle on your Azure infrastructure. It also provides consistency and transparency into infrastructure changes.

Puppet provides a series of open-source configuration management tools and projects. It also provides Puppet Enterprise, which is a configuration management platform that allows you to maintain state in both your infrastructure and application deployments.



Puppet architectural components

Puppet operates using a client server model, and consists of the following core components:

- Puppet Master. The **Puppet Master** is responsible for compiling code to create agent catalogs. It's also where Secure Sockets Layer (SSL) certificates are verified and signed. Puppet Enterprise infrastructure components are installed on a single node, the master. The master always contains a compile master and a Puppet Server. As your installation grows, you can add additional compile masters to distribute the catalog compilation workload.
- Puppet Agent. **Puppet Agent** is the machine (or machines) managed by the Puppet Master. An agent that is installed on those managed machines allows them to be managed by the Puppet Agent.
- Console Services. **Console Services** are the web-based user interface for managing your systems.
- Facts. **Facts** are metadata related to state. Puppet will query a node and determine a series of facts, which it then uses to determine state.

Deploying Puppet in Azure

Puppet Enterprise lets you automate the entire lifecycle of your Azure infrastructure simply, scalably, and securely, from initial provisioning through application deployment.

Puppet Enterprise is available to install directly into Azure using the [Azure Marketplace](#)⁵. The Puppet Enterprise image allows you to manage up to 10 Azure VMs for free, and is available to use immediately.

After you select it, you need to fill in the VM's parameter values. A preconfigured system will then run and test Puppet, and will preset many of the settings. However, these can be changed as needed. The VM will then be created, and Puppet will run the install scripts.

Another option for creating a Puppet master in Azure is to install a Linux VM in Azure and deploy the Puppet Enterprise package manually.

⁵ <https://azure.microsoft.com/en-us/marketplace/>

Manifest files

Puppet uses a declarative file syntax to define state. It defines what the infrastructure state should be, but not how it should be achieved. You must tell it you want to install a package, but not how you want to install the package.

Configuration or state is defined in manifest files known as *Puppet Program files*. These files are responsible for determining the state of the application, and have the file extension **.pp**.

Puppet program files have the following elements:

- **class**. This is a bucket that you put resources into. For example, you might have an **Apache** class with everything required to run Apache (such as the package, config file, running server, and any users that need to be created). That class then becomes an entity that you can use to compose other workflows.
- **resources**. These are single elements of your configuration that you can specify parameters for.
- **module**. This is the collection of all the classes, resources, and other elements of the Puppet program file in a single entity.

Sample manifest (.pp) file

In the following sample .pp file, notice where classes are being defined, and within that, where resources and package details are defined.

✓ Note: The `->` notation is an “ordering arrow”: it tells Puppet that it must apply the “left” resource before invoking the “right” resource. This allows us to specify order, when necessary:

```
class mrapp {
    class { 'configuremongodb': }
    class { 'configurejava': }
}

class configuremongodb {
    include wget
    class { 'mongodb': }->

    wget::fetch { 'mongorecords':
        source => 'https://raw.githubusercontent.com/Microsoft/PartsUnlimitedM-RP/master/deploy/MongoRecords.js',
        destination => '/tmp/MongoRecords.js',
        timeout => 0,
    }->
    exec { 'insertrecords':
        command => 'mongo ordering /tmp/MongoRecords.js',
        path => '/usr/bin:/usr/sbin',
        unless => 'test -f /tmp/initcomplete'
    }->
    file { '/tmp/initcomplete':
        ensure => 'present',
    }
}

class configurejava {
```

```
include apt
$packages = ['openjdk-8-jdk', 'openjdk-8-jre']
apt::ppa { 'ppa:openjdk-r/ppa': }->
package { $packages:
  ensure => 'installed',
}

}
```

You can download customer Puppet modules that Puppet and the Puppet community have created from **puppetforge**⁶. *Puppetforge* is a community repository that contains thousands of modules for download and use, or modification as you need. This saves you the time necessary to recreate modules from scratch.

⁶ <https://forge.puppet.com/>

Ansible

What is Ansible

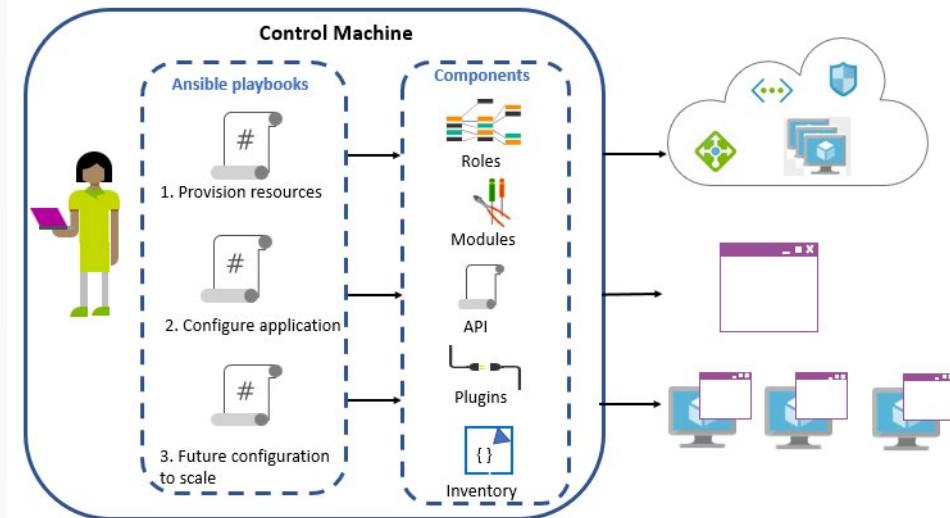
Ansible is an open-source platform by Red Hat that automates cloud provisioning, configuration management, and application deployments. Using Ansible, you can provision VMs, containers, and your entire cloud infrastructure. In addition to provisioning and configuring applications and their environments, Ansible enables you to automate deployment and configuration of resources in your environment such as virtual networks, storage, subnets, and resources groups.

Ansible is designed for multiple tier deployments. Unlike Puppet or Chef, Ansible is **agentless**, meaning you don't have to install software on the managed machines.

Ansible also models your IT infrastructure by describing how all of your systems interrelate, rather than managing just one system at a time.

Ansible workflow

The following workflow and component diagram outlines how playbooks can run in different circumstances, one after another. In the workflow, Ansible playbooks:



1. Provision resources. Playbooks can provision resources. In the following diagram, playbooks create load-balancer virtual networks, network security groups, and VM scale sets on Azure.
2. Configure the application. Playbooks can deploy applications to run particular services, such as installing Apache Tomcat on a Linux machine to allow you to run a web application.
3. Manage future configurations to scale. Playbooks can alter configurations by applying playbooks to existing resources and applications—in this instance to scale the VMs.

In all cases, Ansible makes use of core components such as roles, modules, APIs, plugins, inventory, and other components.

- ✓ Note: By default, Ansible manages machines using the *ssh* protocol.
- ✓ Note: You don't need to maintain and run commands from any particular central server. Instead, there is a control machine with Ansible installed, and from which playbooks are run.

Ansible Components

Ansible models your IT infrastructure by describing how all of your systems interrelate, rather than just managing one system at a time. The core components of Ansible are:

- Control Machine. This is the machine from which the configurations are run. It can be any machine with Ansible installed on it. However, it requires that Python 2 or Python 3 be installed on the control machine as well. You can have multiple control nodes, laptops, shared desktops, and servers all running Ansible.
- Managed Nodes. These are the devices and machines (or just machines) and environments that are being managed. Managed nodes are sometimes referred to as *hosts*. Ansible is not installed on nodes.
- Playbooks. Playbooks are ordered lists of tasks that have been saved so you can run them repeatedly in the same order. Playbooks are Ansible's language for configuration, deployment, and orchestration. They can describe a policy that you want your remote systems to enforce, or they can dictate a set of steps in a general IT process.

When you create a playbook, you do so using YAML, which defines a model of a configuration or process, and uses a declarative model. Elements such as **name**, **hosts**, and **tasks** reside within playbooks.

- Modules. Ansible works by connecting to your nodes, and then pushing small programs (or *units of code*)—called *modules*—out to the nodes. *Modules* are the units of code that define the configuration. They are modular, and can be reused across playbooks. They represent the desired state of the system (declarative), are executed over SSH by default, and are removed when finished.

A playbook is typically made up of many modules. For example, you could have one playbook containing three modules: a module for creating an Azure Resource group, a module for creating a virtual network, and a module for adding a subnet.

Your library of modules can reside on any machine, and do not require any servers, daemons, or databases. Typically, you'll work with your favorite terminal program, a text editor, and most likely a version control system to track changes to your content. A complete list of available modules is available on Ansible's [All modules⁷](#) page.

You can preview Ansible Azure modules on the Ansible [Azure preview modules⁸](#) webpage.

- Inventory. An **inventory** is a list of managed nodes. Ansible represents what machines it manages using a .INI file that puts all your managed machines in groups of your own choosing. When adding new machines, you don't need to use additional SSL-signing servers, thus avoiding Network Time Protocol (NTP) and Domain Name System (DNS) issues. You can create the inventory manually, or for Azure, Ansible supports dynamic inventories> This means that the host inventory is dynamically generated at runtime. Ansible supports host inventories for other managed hosts as well.
- Roles. **Roles** are predefined file structures that allow automatic loading of certain variables, files, tasks, and handlers, based on the file's structure. It allows for easier sharing of roles. You might, for example, create roles for a web server deployment.
- Facts. **Facts** are data points about the remote system that Ansible is managing. When a playbook is run against a machine, Ansible will gather facts about the state of the environment to determine the state before executing the playbook.
- Plug-ins. **Plug-ins** are code that supplements Ansible's core functionality.

⁷ https://docs.ansible.com/ansible/latest/modules/list_of_all_modules.html

⁸ https://galaxy.ansible.com/Azure/azure_preview_modules

Installing Ansible

To enable a machine to act as the control machine from which to run playbooks, you need to install both Python and Ansible.

Python

When you install Python, you must install either Python 2 (version 2.7), or Python 3 (versions 3.5 and later). You can use pip, the Python package manager, to install Python, or you can use other installation methods.

Ansible installation characteristics

An Ansible installation has the following characteristics:

- You only need to install Ansible on one machine, which could be a workstation or a laptop. You can manage an entire fleet of remote machines from that central point.
- No database is installed as part of the Ansible setup.
- No daemons are required to start or keep Ansible running.

Ansible on Linux

You can install Ansible on many different distributions of Linux, including, but not limited to:

- Red Hat Enterprise Linux
- CentOS
- Debian
- Ubuntu
- Fedora

✓ Note: Fedora is not supported as an endorsed Linux distribution on Azure. However, you can run it on Azure by uploading your own image. All other Linux distributions are supported on Azure as endorsed by Linux.

You can use the appropriate package manager software to install Ansible and Python, such as `yum`, `apt`, or `pip`. For example, To install Ansible on Ubuntu, run the following command:

```
## Install pre-requisite packages
sudo apt-get update && sudo apt-get install -y libssl-dev libffi-dev python-dev python-pip
## Install Ansible and Azure SDKs via pip
sudo pip install ansible[azure]
```

macOS

You can also install Ansible and Python on macOS, and use that environment as the control machine.

Windows operating system

You cannot install Ansible on the Windows operating system. However, you can run playbooks from Windows by utilizing other products and services. You can install Ansible and Python on operating systems such as:

- Windows Subsystem for Linux. This is an Ubuntu Linux environment available as part of Windows.
- Azure Cloud Shell. You can use Azure Cloud Shell via a web browser on a Windows machine.
- Microsoft Visual Studio Code. Using Visual Studio Code, choose one of the following options:
 - Run Ansible playbook in Docker.
 - Run Ansible playbook on local Ansible.
 - Run Ansible playbook in Azure Cloud Shell.
 - Run Ansible playbook remotely via SSH.

Upgrading Ansible

When Ansible manages remote machines, it doesn't leave software installed or running on them. Therefore, there's no real question about how to upgrade Ansible when moving to a new version.

Managed nodes

When managing nodes you need a way to communicate on the managed nodes or environments, which is normally using SSH by default. This uses the SSH file transfer protocol. If that's not available, you can switch to Simple Control Protocol (SCP), which you can do in **ansible.cfg**. For Windows machines, use Windows PowerShell.

You can find out more about installing Ansible on the [Install Ansible on Azure virtual machines⁹](#) page.

Ansible on Azure

There are several ways you can use Ansible in Azure.

Azure marketplace

You can use one of the following images available as part of the Azure Marketplace:

- Red Hat Ansible on Azure is available as an image on Azure Marketplace, and it provides a fully configured version. This enables easier adoption for those looking to use Ansible as their provisioning and configuration management tool. This solution template will install Ansible on a Linux VM along with tools configured to work with Azure. This includes:
 - Ansible (the latest version by default. You can also specify a version number.)
 - Azure CLI 2.0
 - MSI VM extension
 - apt-transport-https

⁹ <https://docs.microsoft.com/en-us/azure/virtual-machines/linux/ansible-install-configure?toc=%2Fen-us%2Fazure%2Fansible%2Ftoc.json&json&bc=%2Fen-us%2Fazure%2Fbread%2Ftoc.json>

- Ansible Tower (by Red Hat). Ansible Tower by Red Hat helps organizations scale IT automation and manage complex deployments across physical, virtual, and cloud infrastructures. Built on the proven open-source Ansible automation engine, Ansible Tower includes capabilities that provide additional levels of visibility, control, security, and efficiency necessary for today's enterprises. With Ansible Tower you can:
 - Provision Azure environments with ease using pre-built Ansible playbooks.
 - Use role-based access control (RBAC) for secure, efficient management.
 - Maintain centralized logging for complete auditability and compliance.
 - Utilize the large community of content available on Ansible Galaxy.

This offering requires the use of an available Ansible Tower subscription eligible for use in Azure. If you don't currently have a subscription, you can obtain one directly from Red Hat.

Azure VMs

Another option for running Ansible on Azure is to deploy a Linux VM on Azure virtual machines, which is infrastructure as a service (IaaS). You can then install Ansible and the relevant components, and use that as the control machine.

- ✓ Note: The Windows operating system is not supported as a control machine. However, you can run Ansible from a Windows machine by utilizing other services and products such as Windows Subsystem for Linux, Azure Cloud Shell, and Visual Studio Code.

For more details about running Ansible in Azure, visit:

- [Ansible on Azure documentation¹⁰](#) website
- [Microsoft Azure Guide¹¹](#)

Playbook structure

Playbooks are the language of Ansible's configurations, deployments, and orchestrations. You use them to manage configurations of and deployments to remote machines. Playbooks are structured with YAML (a data serialization language), and support variables. Playbooks are declarative and include detailed information regarding the number of machines to configure at a time.

YAML structure

YAML is based around the structure of key-value pairs. In the following example, the key is name, and the value is namevalue:

```
name: namevalue
```

In the YAML syntax, a child key value pair is placed on new, and indented, line below its parent key. Each sibling key value pair occurs on a new line at the same level of indentation as its sibling key value pair.

```
parent:  
  children:  
    first-sibling: value01
```

¹⁰ https://docs.microsoft.com/en-us/azure/ansible/?ocid=AID754288&wt.mc_id=CFID0352

¹¹ https://docs.ansible.com/ansible/latest/scenario_guides/guide_azure.html

```
second-sibling: value02
```

The specific number of spaces used for indentation is not defined. You can indent each level by as many spaces as you want. However, the number of spaces used for indentations at each level must be uniform throughout the file.

When there is indentation in a YAML file, the indented key value pair is the value of its parent key.

Playbook components

The following list is of some of the playbook components:

- `name`. The name of the playbook. This can be any name you wish.
- `hosts`. Lists where the configuration is applied, or machines being targeted. Hosts can be a list of one or more groups or host patterns, separated by colons. It can also contain groups such as web servers or databases, providing that you have defined these groups in your inventory.
- `connection`. Specifies the connection type.
- `remote_user`. Specifies the user that will be connected to for completing the tasks.
- `var`. Allows you to define the variables that can be used throughout your playbook.
- `gather_facts`. Determines whether to gather node data or not. The value can be `yes` or `no`.
- `tasks`. Indicates the start of the modules where the actual configuration is defined.

Running a playbook

You run a playbook using the following command:

```
ansible-playbook < playbook name >
```

You can also check the syntax of a playbook using the following command.

```
ansible-playbook --syntax-check
```

The syntax check command runs a playbook through the parser to verify that it has included items, such as files and roles, and that the playbook has no syntax errors. You can also use the `--verbose` command.

- To see a list of hosts that would be affected by running a playbook, run the command:

```
ansible-playbook playbook.yml --list-hosts
```

Sample Playbook

The following code is a sample playbook that will create a Linux virtual machine in Azure:

```
- name: Create Azure VM
  hosts: localhost
  connection: local
  vars:
    resource_group: ansible_rg5
    location: westus
```

```
tasks:
- name: Create resource group
  azure_rm_resourcegroup:
    name: "{{ resource_group }}"
    location: "{{ location }}"
- name: Create virtual network
  azure_rm_virtualnetwork:
    resource_group: myResourceGroup
    name: myVnet
    address_prefixes: "10.0.0.0/16"
- name: Add subnet
  azure_rm_subnet:
    resource_group: myResourceGroup
    name: mySubnet
    address_prefix: "10.0.1.0/24"
    virtual_network: myVnet
- name: Create public IP address
  azure_rm_publicipaddress:
    resource_group: myResourceGroup
    allocation_method: Static
    name: myPublicIP
    register: output_ip_address
- name: Dump public IP for VM which will be created
  debug:
    msg: "The public IP is {{ output_ip_address.state.ip_address }}."
- name: Create Network Security Group that allows SSH
  azure_rm_securitygroup:
    resource_group: myResourceGroup
    name: myNetworkSecurityGroup
    rules:
      - name: SSH
        protocol: Tcp
        destination_port_range: 22
        access: Allow
        priority: 1001
        direction: Inbound
- name: Create virtual network interface card
  azure_rm_networkinterface:
    resource_group: myResourceGroup
    name: myNIC
    virtual_network: myVnet
    subnet: mySubnet
    public_ip_name: myPublicIP
    security_group: myNetworkSecurityGroup
- name: Create VM
  azure_rm_virtualmachine:
    resource_group: myResourceGroup
    name: myVM
    vm_size: Standard_DS1_v2
    admin_username: azureuser
    ssh_password_enabled: false
```

```

ssh_public_keys:
  - path: /home/azureuser/.ssh/authorized_keys
    key_data: <your-key-data>
network_interfaces: myNIC
image:
  offer: CentOS
  publisher: OpenLogic
  sku: '7.5'
  version: latest

```

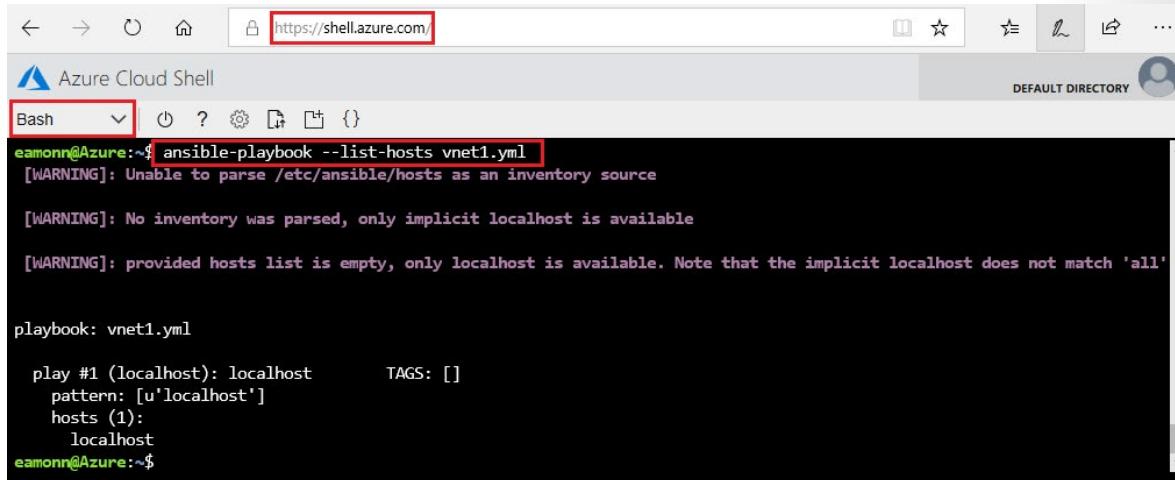
- ✓ Note: Ansible Playbook samples for Azure are available on GitHub on the [Ansible Playbook Samples for Azure¹²](#) page.

Demonstration-Run Ansible in Azure Cloud Shell

You can run Ansible playbooks on a Windows machine by using the Azure Cloud Shell with Bash. This is the quickest and easiest way to begin using playbook's provisioning and management features in Azure.

Run commands

Azure Cloud Shell has Ansible preinstalled. After you are signed into Azure Cloud Shell, specify the bash console. You do not need to install or configure anything further to run Ansible commands from the Bash console in Azure Cloud Shell.



The screenshot shows the Azure Cloud Shell interface. The browser address bar displays "https://shell.azure.com/". The Azure Cloud Shell header includes the "Azure Cloud Shell" logo, a dropdown menu set to "Bash", and a "DEFAULT DIRECTORY" button. Below the header is a toolbar with icons for back, forward, refresh, and other shell functions. The main terminal window shows the command "ansible-playbook --list-hosts vnet1.yml" being run. The output indicates several warnings: "WARNING: Unable to parse /etc/ansible/hosts as an inventory source", "[WARNING]: No inventory was parsed, only implicit localhost is available", and "[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'". The command also specifies "playbook: vnet1.yml", "play #1 (localhost): localhost", "TAGS: []", and "hosts (1): localhost". The command concludes with "eamonn@Azure:~\$".

Editor

You can also use the Azure Cloud Shell editor to review, open, and edit your playbook .yml files. You can open the editor by selecting the curly brackets icon on the Azure Cloud Shell toolbar.

¹² <https://github.com/Azure-Samples/ansible-playbooks>

The screenshot shows the Azure Cloud Shell interface. The URL bar at the top has 'https://shell.azure.com/' highlighted with a red box. The main area is a terminal window titled 'Bash'. On the left, there's a file browser showing various Ansible files like '.ansible', 'vnet1.yml', and 'rg.yml'. The terminal content shows the creation of a resource group and a virtual network using Ansible. The code is as follows:

```
---  
- hosts: localhost  
  vars:  
    resource_group: ansible_rg5  
    location: westus  
  tasks:  
    - name: Create a resource group on Azure  
      azure_rm_resourcegroup:  
        name: "{{ resource_group }}"  
        location: "{{ location }}"  
    - name: Create virtual network  
      azure_rm_virtualnetwork:  
        resource_group: "{{ resource_group }}"  
        name: myNet  
        address_prefixes: "10.0.0.0/16"  
    - name: Add subnet  
      azure_rm_subnet:  
        resource_group: "{{ resource_group }}"  
        name: mySubnet  
        address_prefix: "10.0.1.0/24"  
        virtual_network: myVnet
```

Below the terminal, status messages say 'Requesting a Cloud Shell...Succeeded.' and 'Connecting terminal...'. The prompt shows '#>'. The bottom of the terminal displays help text for the Azure CLI.

Create a resource group

The following steps outline how to create a resource group in Azure using Ansible in Azure Cloud Shell with bash:

1. Go to the **Azure Cloud Shell**¹³. You can also launch Azure Cloud Shell from within the Azure portal by selecting the Azure PowerShell icon on the taskbar.
2. Authenticate to Azure by entering your credentials, if prompted.
3. On the taskbar, ensure **Bash** is selected as the shell.
4. Create a new file using the following command:

```
vi rg.yml
```
5. Enter insert mode by selecting the **I** key.
6. Copy and paste the following code into the file, and remove the **#**, comment character. (It's included here for displaying code in the learning platform.) The code should be aligned as in the previous screenshot.

```
#---  
- hosts: localhost  
  connection: local  
  tasks:  
    - name: Create resource group  
      azure_rm_resourcegroup:  
        name: ansible-rg
```

¹³ <https://shell.azure.com>

```
location: eastus
```

7. Exit insert mode by selecting the **Esc** key.
8. Save the file and exit the vi editor by entering the following command:

```
:wq
```

9. Run the playbook with the following command:

```
ansible-playbook rg.yml
```

10. Verify that you receive output similar to the following code:

```
PLAY [localhost] ****
*****
TASK [Gathering Facts] ****
*****
ok: [localhost]

TASK [Create resource group] ****
*****
changed: [localhost]

TASK [debug] ****
*****
ok: [localhost] => {
    "rg": {
        "changed": true,
        "contains_resources": false,
        "failed": false,
        "state": {
            "id": "/subscriptions/XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX/resourceGroups/ansible-rg",
            "location": "eastus",
            "name": "ansible-rg",
            "provisioning_state": "Succeeded",
            "tags": null
        }
    }
}

PLAY RECAP ****
*****
localhost : ok=3      changed=1      unreachable=0      failed=0
```

11. Open Azure portal and verify that the resource group is now available in the portal.

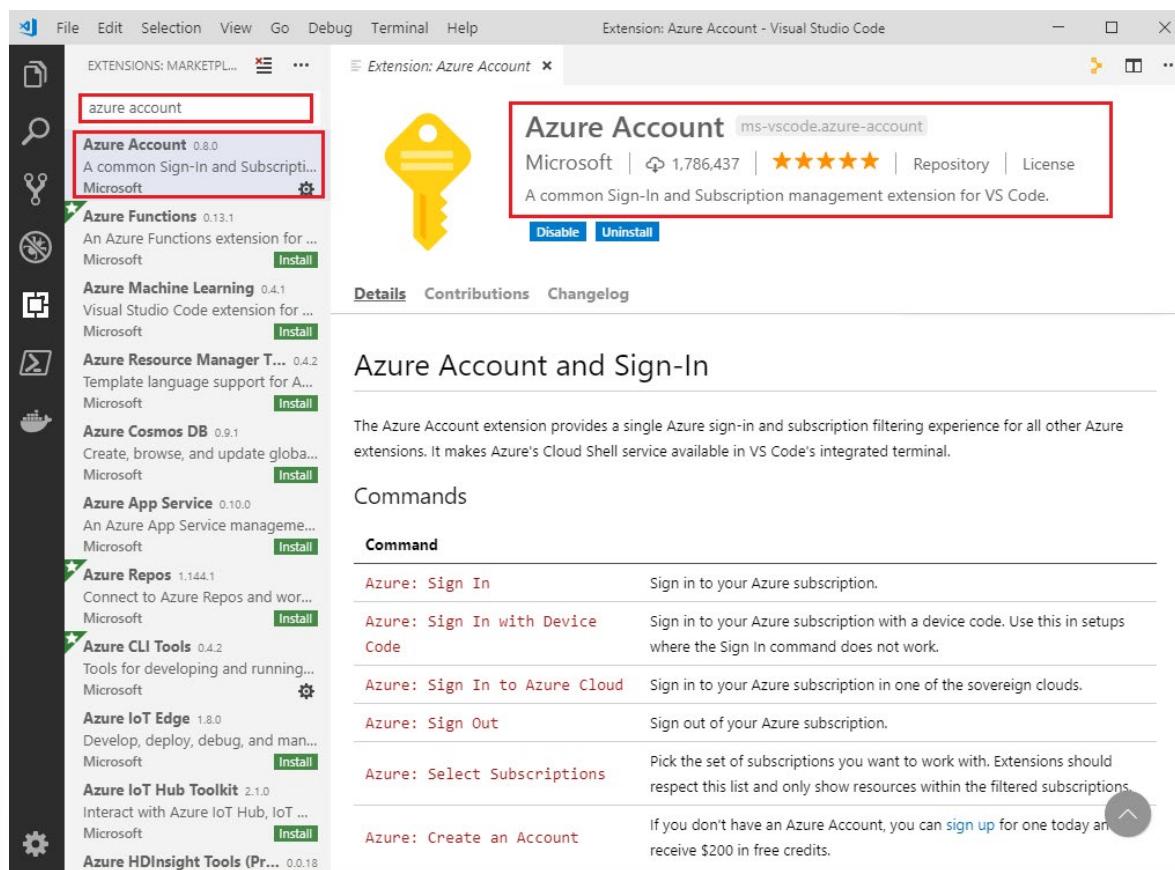
Demonstration-Run Ansible in Visual Studio Code

You can also run **Ansible** playbooks on a Windows machine using Visual Studio Code. This leverages other services that can also be integrated using Visual Studio Code.

Create network resources in Azure using Visual Studio Code

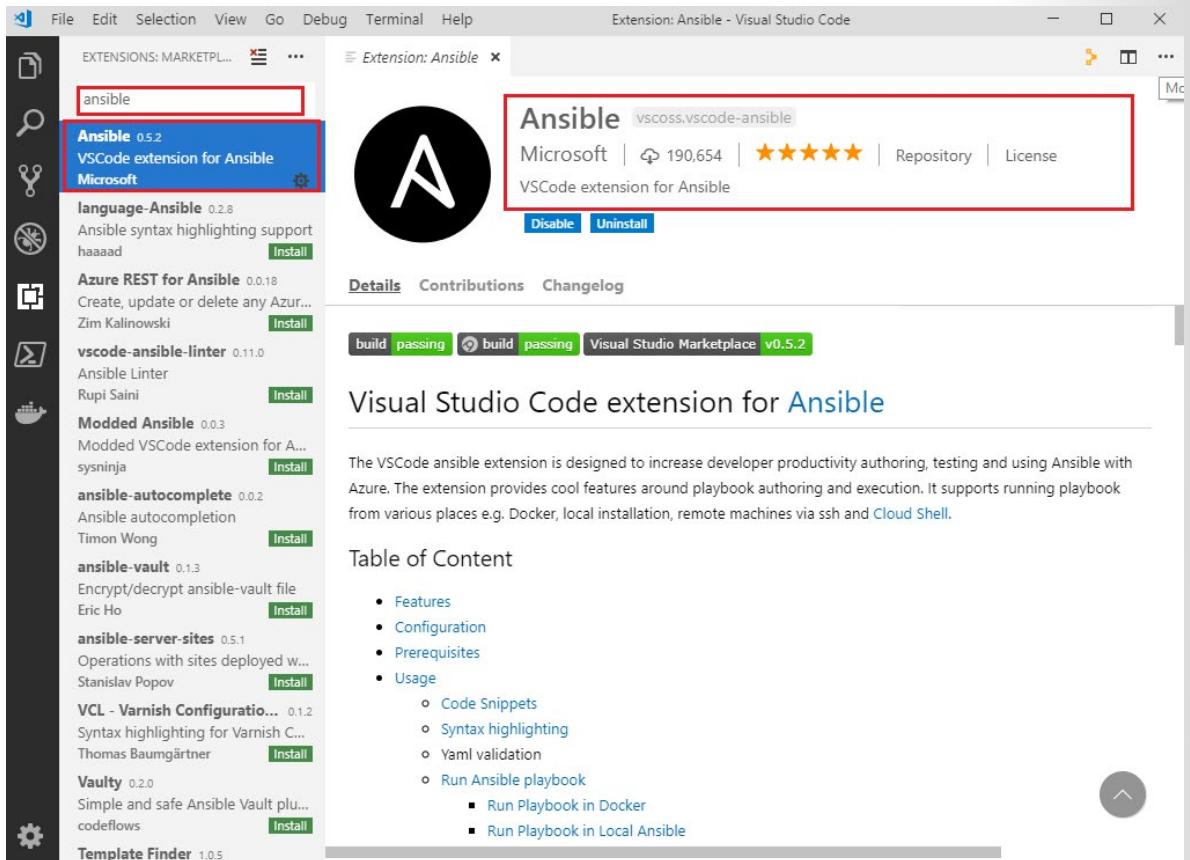
Complete the following steps to create network resources in Azure using Visual Studio Code:

1. If not already installed, install Visual Studio Code by downloading it from the <https://code.visualstudio.com/>¹⁴ page. You can install it on the Windows, Linux, or macOS operating systems.
2. Go to **File > Preferences > Extensions**.
3. Search for and install the extension **Azure Account**.



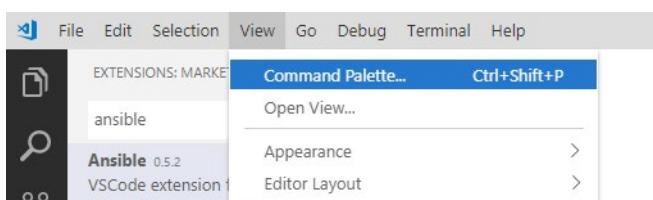
4. Search for and install the extension **Ansible**.

¹⁴ <https://code.visualstudio.com/>



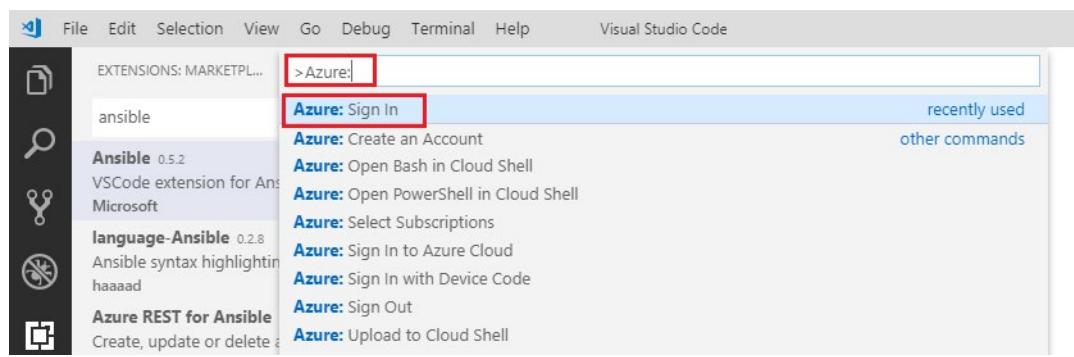
You can also view details of this extension on the Visual Studio Marketplace [Ansible](#)¹⁵ page.

5. In Visual Studio Code, go to **View > Command Palette....** Alternatively, you can select the **settings** (cog) icon in the bottom, left corner of the **Visual Studio Code** window, and then select **Command Palette**.

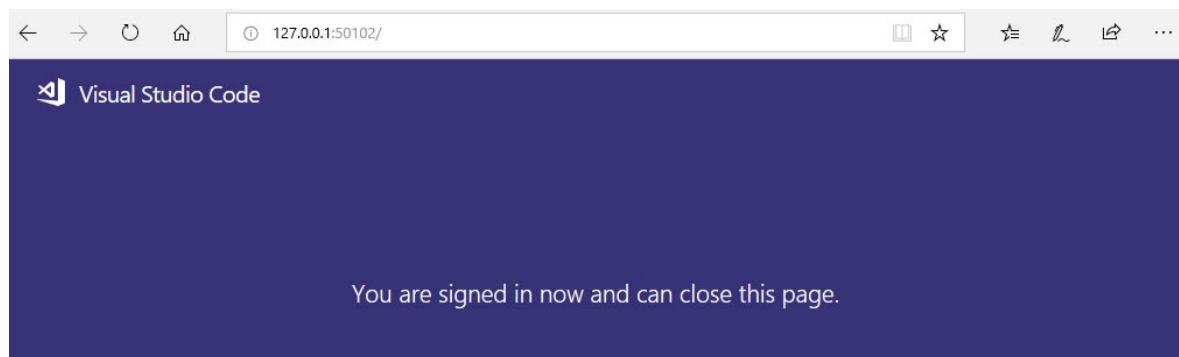


6. In the Command Palette, Type **Azure:**, select **Azure:Sign in**.

¹⁵ https://marketplace.visualstudio.com/items?itemName=vscozz.vscode-ansible&ocid=AID754288&wt.mc_id=CFID0352



7. When a browser launches and prompts you to sign in, select your Azure account. Verify that a message displays stating that you are now signed in and can close the page.



8. Verify that your Azure account now displays at the bottom of the Visual Studio Code window.
9. Create a new file and paste in the following playbook text:

```
- name: Create Azure VM
  hosts: localhost
  connection: local
  tasks:
    - name: Create resource group
      azure_rm_resourcegroup:
        name: myResourceGroup
        location: eastus
    - name: Create virtual network
      azure_rm_virtualnetwork:
        resource_group: myResourceGroup
        name: myVnet
        address_prefixes: "10.0.0.0/16"
    - name: Add subnet
      azure_rm_subnet:
        resource_group: myResourceGroup
        name: mySubnet
        address_prefix: "10.0.1.0/24"
        virtual_network: myVnet
    - name: Create public IP address
      azure_rm_publicipaddress:
        resource_group: myResourceGroup
```

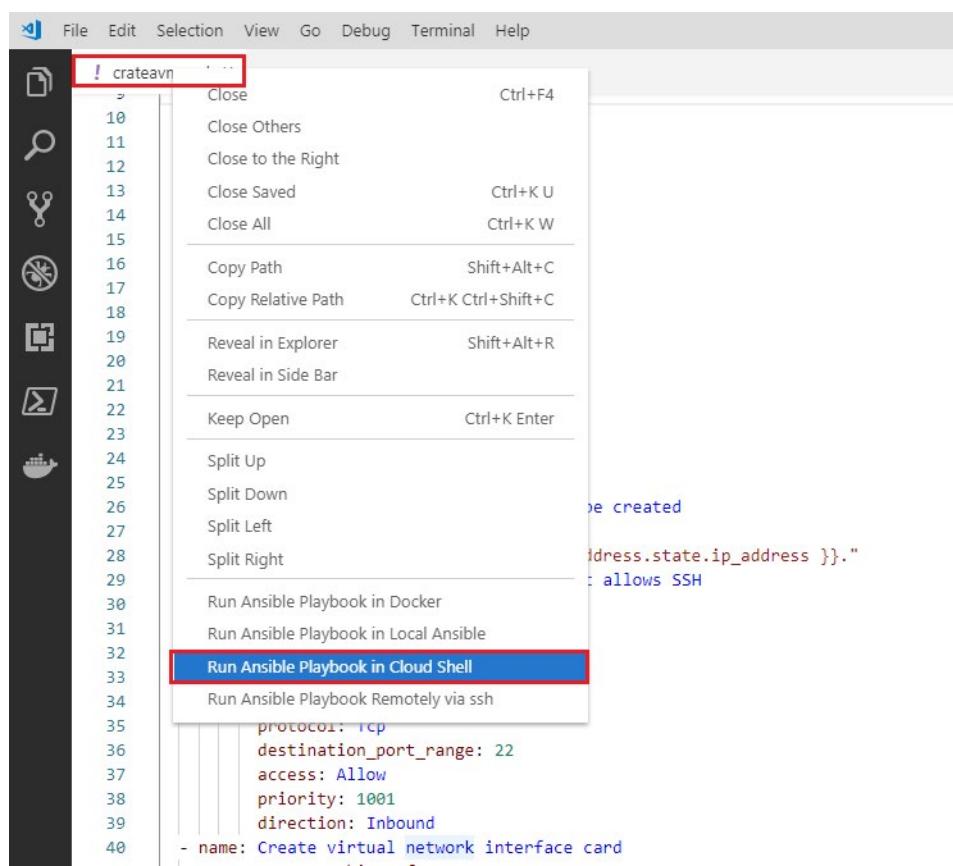
```
        allocation_method: Static
        name: myPublicIP
    register: output_ip_address
- name: Dump public IP for VM which will be created
  debug:
    msg: "The public IP is {{ output_ip_address.state.ip_address }}."
- name: Create Network Security Group that allows SSH
  azure_rm_securitygroup:
    resource_group: myResourceGroup
    name: myNetworkSecurityGroup
    rules:
      - name: SSH
        protocol: Tcp
        destination_port_range: 22
        access: Allow
        priority: 1001
        direction: Inbound
- name: Create virtual network interface card
  azure_rm_networkinterface:
    resource_group: myResourceGroup
    name: myNIC
    virtual_network: myVnet
    subnet: mySubnet
    public_ip_name: myPublicIP
    security_group: myNetworkSecurityGroup
- name: Create VM
  azure_rm_virtualmachine:
    resource_group: myResourceGroup
    name: myVM
    vm_size: Standard_DS1_v2
    admin_username: azureuser
    ssh_password_enabled: true
    admin_password: Password0134
    network_interfaces: myNIC
    image:
      offer: CentOS
      publisher: OpenLogic
      sku: '7.5'
      version: latest
```

10. Save the file locally, and name it **createavm.yml**.

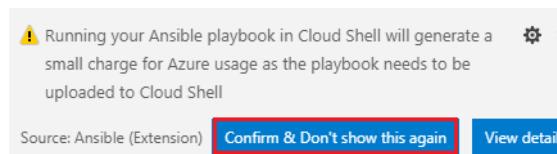
11. Right-click on the file name in the tab at the top of Visual Studio Code, and review the available options available to run the Ansible playbook:

- Run Ansible Playbook in Docker
- Run Ansible Playbook in Local Ansible
- Run Ansible Playbook Cloud Shell
- Run Ansible Playbook Remotely via ssh

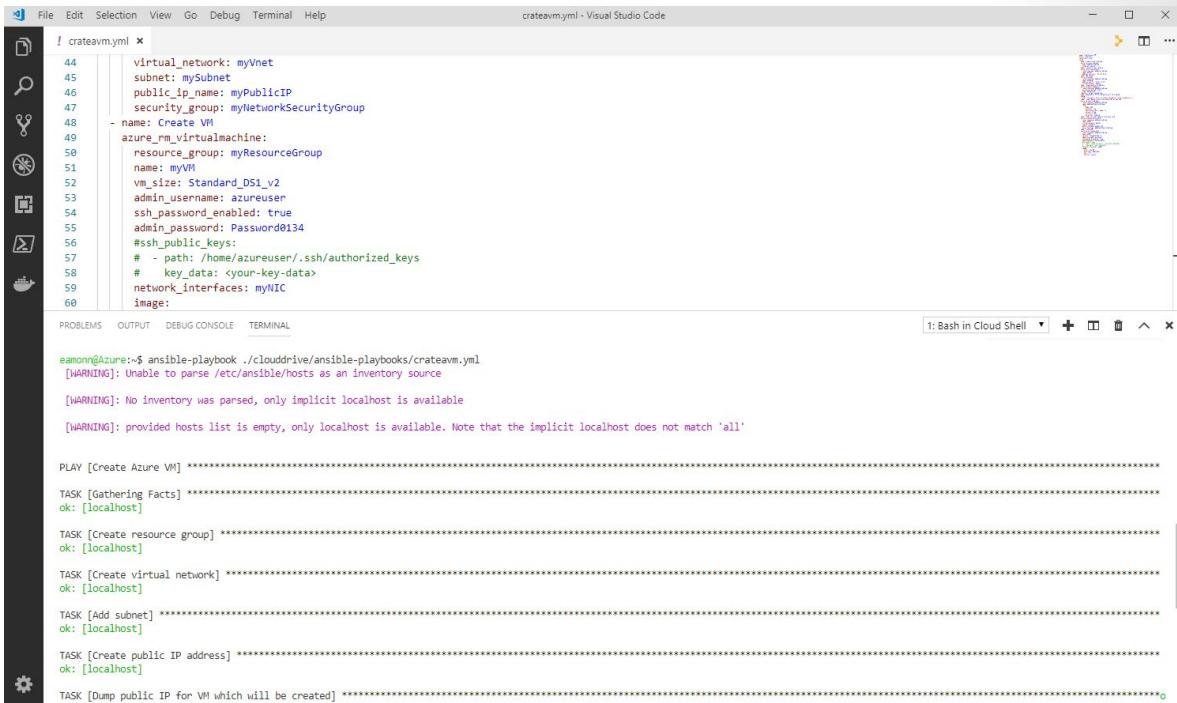
12. Select the third option, **Run Ansible Playbook Cloud Shell**.



13. A notice might appear in the bottom, left side, informing you that the action could incur a small charge as it will use some storage when the playbook is uploaded to cloud shell. Select **Confirm & Don't show this message again**.



14. Verify that the Azure Cloud Shell pane now displays in the bottom of Visual Studio Code and is running the playbook.



```

File Edit Selection View Go Debug Terminal Help
cratevm.yml - Visual Studio Code
I cratevm.yml x
44     virtual_network: myVnet
45     subnet: mySubnet
46     public_ip_name: myPublicIP
47     security_group: myNetworkSecurityGroup
48   - name: Create VM
49     azure_rm_virtualmachine:
50       resource_group: myResourceGroup
51       name: myVM
52       vm_size: Standard_DS1_v2
53       admin_username: azureuser
54       ssh_password_enabled: true
55       admin_password: Password0134
56       #ssh_public_keys:
57       #   - path: /home/azureuser/.ssh/authorized_keys
58       #     key_data: <your key data>
59       network_interfaces: myNIC
60       image:
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: Bash in Cloud Shell + ^ x
eamonn@Azure:~$ ansible-playbook ./clouddrive/ansible-playbooks/createvm.yml
[WARNING]: Unable to parse /etc/ansible/hosts as an inventory source
[WARNING]: No inventory was parsed, only implicit localhost is available
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'

PLAY [Create Azure VM] ****
TASK [Gathering Facts] ****
ok: [localhost]

TASK [Create resource group] ****
ok: [localhost]

TASK [Create virtual network] ****
ok: [localhost]

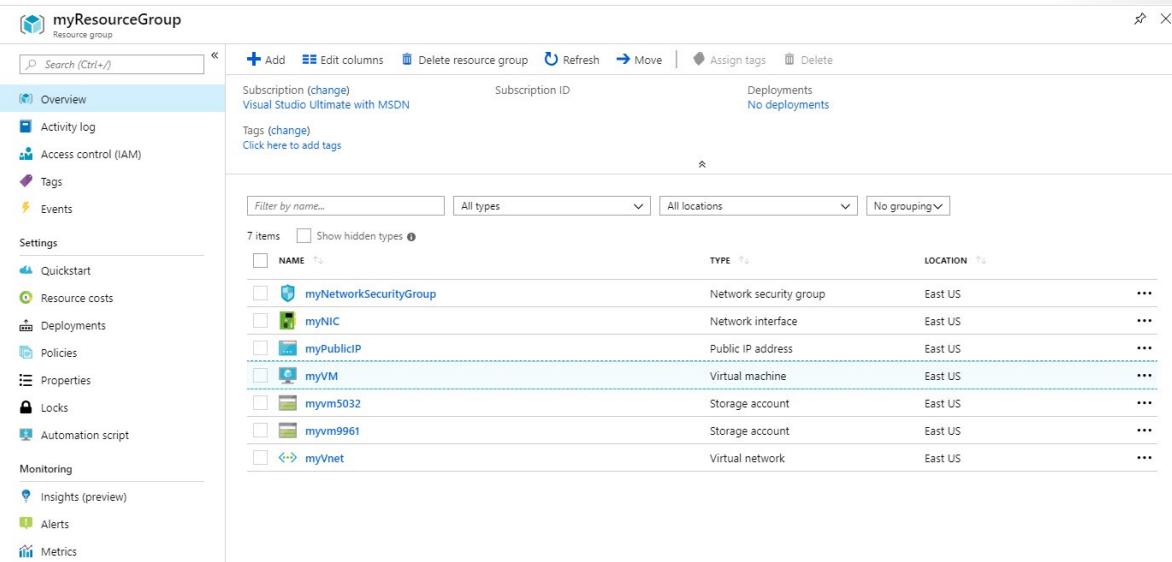
TASK [Add subnet] ****
ok: [localhost]

TASK [Create public IP address] ****
ok: [localhost]

TASK [Dump public IP for VM which will be created] ****

```

15. When the playbook finishes running, open Azure and verify the resource group, resources, and VM have all been created. If you have time, sign in with the user name and password specified in the playbook to verify as well.



| NAME | TYPE | LOCATION |
|------------------------|------------------------|----------|
| myNetworkSecurityGroup | Network security group | East US |
| myNIC | Network interface | East US |
| myPublicIP | Public IP address | East US |
| myVM | Virtual machine | East US |
| myvm5032 | Storage account | East US |
| myvm9961 | Storage account | East US |
| myVnet | Virtual network | East US |

- ✓ Note: If you want to use a public or private key pair to connect to the Linux VM, instead of a user name and password you could use the following code in the previous Create VM module steps:

```

admin_username: adminUser
ssh_password_enabled: false
ssh_public_keys:
  - path: /home/adminUser/.ssh/authorized_keys
    key_data: < insert your ssh public key here... >

```

Terraform

What is Terraform

HashiCorp Terraform is an open-source tool that allows you to provision, manage, and version cloud infrastructure. It codifies infrastructure in configuration files that describes the topology of cloud resources such as VMs, storage accounts, and networking interfaces.



Terraform's command-line interface (CLI) provides a simple mechanism to deploy and version the configuration files to Azure or any other supported cloud service. The CLI also allows you to validate and preview infrastructure changes before you deploy them.

Terraform also supports multi-cloud scenarios. This means it enables developers to use the same tools and configuration files to manage infrastructure on multiple cloud providers.

You can run Terraform interactively from the CLI with individual commands, or non-interactively as part of a continuous integration pipeline.

There is also an enterprise version of Terraform available, **Terraform Enterprise**.

You can view more details about Terraform on the **HashiCorp Terraform¹⁶** website.

Terraform components

Some of Terraform's core components include:

- Configuration files. Text-based configuration files allow you to define infrastructure and application configuration. These files end in the .tf or .tf.json extension. The files can be in either of the following two formats:
 - Terraform. The Terraform format is easier for users to review, thereby making it more user friendly. It supports comments, and is the generally recommended format for most Terraform files. Terraform files ends in .tf
 - JSON. The JSON format is mainly for use by machines for creating, modifying, and updating configurations. However, it can also be used by Terraform operators if you prefer. JSON files end in .tf.json.

The order of items (such as variables and resources) as defined within the configuration file does not matter, because Terraform configurations are declarative.

- Terraform CLI. This is a command-line interface from which you run configurations. You can run command such as **Terraform apply** and **Terraform plan**, along with many others. A CLI configuration file that configures per-user setting for the CLI is also available. However, this is separate from the CLI infrastructure configuration. In Windows operating system environments, the configuration file is named **terraform.rc**, and is stored in the relevant user's %APPDATA% directory. On Linux systems, the file is named **.terraformrc** (note the leading period), and is stored in the home directory of the relevant user.

¹⁶ <https://www.terraform.io/>

- **Modules.** **Modules** are self-contained packages of Terraform configurations that are managed as a group. You use modules to create reusable components in Terraform and for basic code organization. A list of available modules for Azure is available on the [Terraform Registry Modules¹⁷](#) webpage.
- **Provider.** The provider is responsible for understanding API interactions and exposing resources.
- **Overrides.** Overrides are a way to create configuration files that are loaded last and merged into (rather than appended to) your configuration. You can create overrides to modify Terraform behavior without having to edit the Terraform configuration. They can also be used as temporary modifications that you can make to Terraform configurations without having to modify the configuration itself.
- **Resources.** **Resources** are sections of a configuration file that define components of your infrastructure, such as VMs, network resources, containers, dependencies, or DNS records. The resource block creates a resource of the given *TYPE* (first parameter) and *NAME* (second parameter). However, the combination of the type and name must be unique. The resource's configuration is then defined and contained within braces.
- **Execution plan.** You can issue a command in the Terraform CLI to generate an execution plan. The *execution plan* shows what Terraform will do when a configuration is applied. This enables you to verify changes and flag potential issues. The command for the execution plan is **Terraform plan**.
- **Resource graph.** Using a resource graph, you can build a dependency graph of all resources. You can then create and modify resources in parallel. This helps provision and configure resources more efficiently.

Terraform on Azure

You download Terraform for use in Azure via: Azure Marketplace, Terraform Marketplace, or Azure VMs.

Azure Marketplace

Azure Marketplace offers a fully-configured Linux image containing Terraform with the following characteristics:

- The deployment template will install Terraform on a Linux (Ubuntu 16.04 LTS) VM along with tools configured to work with Azure. Items downloaded include:
 - Terraform (latest)
 - Azure CLI 2.0
 - Managed Service Identity (MSI) VM extension
 - Unzip
 - Jq
 - apt-transport-https
- This image also configures a remote back-end to enable remote state management using Terraform.

Terraform Marketplace

The Terraform Marketplace image makes it easy to get started using Terraform on Azure, without having to install and configure Terraform manually. There are no software charges for this Terraform VM image.

¹⁷ <https://registry.terraform.io/browse?provider=azurerm>

You pay only the Azure hardware usage fees that are assessed based on the size of the VM that's provisioned.

Azure VMs

You can also deploy a Linux or Windows VM in Azure VM's IaaS service, install Terraform and the relevant components, and then use that image.

Installing Terraform

To get started, you must install Terraform on the machine from which you are running the Terraform commands.

Terraform can be installed on Windows, Linux or macOS environments. Go to the [Download Terraform¹⁸](#) page, and choose the appropriate download package for your environment.

Windows operating system

If you download Terraform for the Windows operating system:

1. Find the install package, which is bundled as a zip file.
2. Copy files from the zip to a local directory such as C:\terraform. That is the Terraform PATH, so make sure that the Terraform binary is available on the PATH.
3. To set the PATH environment variable, run the command **set PATH=%PATH%;C:\terraform**, or point to wherever you have placed the Terraform executable.
4. Open an administrator command window at C:\Terraform and run the command **Terraform** to verify the installation. You should be able to view the terraform help output.

¹⁸ <https://www.terraform.io/downloads.html>

```
C:\ Administrator: Command Prompt
C:\Terraform>terraform
Usage: terraform [-version] [-help] <command> [args]

The available commands for execution are listed below.
The most common, useful commands are shown first, followed by
less common or more advanced commands. If you're just getting
started with Terraform, stick with the common commands. For the
other commands, please read the help and docs before usage.

Common commands:
  apply      Builds or changes infrastructure
  console    Interactive console for Terraform interpolations
  destroy    Destroy Terraform-managed infrastructure
  env        Workspace management
  fmt        Rewrites config files to canonical format
  get        Download and install modules for the configuration
  graph     Create a visual graph of Terraform resources
  import    Import existing infrastructure into Terraform
  init      Initialize a Terraform working directory
  output    Read an output from a state file
  plan      Generate and show an execution plan
  providers Prints a tree of the providers used in the configuration
  push      Upload this Terraform module to Atlas to run
  refresh   Update local state file against real resources
  show      Inspect Terraform state or plan
  taint     Manually mark a resource for recreation
  untaint   Manually unmark a resource as tainted
  validate  Validates the Terraform files
  version   Prints the Terraform version
  workspace Workspace management

All other commands:
  debug     Debug output management (experimental)
  force-unlock Manually unlock the terraform state
  state     Advanced state management
```

Linux

1. Download Terraform using the following command:

```
 wget https://releases.hashicorp.com/terraform/0.xx.x/terraform_0.xx.x_linux_amd64.zip
```

2. Install Unzip using the command:

```
 sudo apt-get install unzip
```

3. Unzip and set the path using the command:

```
 unzip terraform_0.11.1_linux_amd64.zip
 sudo mv terraform /usr/local/bin/
```

4. Verify the installation by running the command **Terraform**. Verify that the Terraform help output displays.

```
azureuser@Terraform: ~
azur
azureuser@Terraform:~$ terraform
Usage: terraform [-version] [-help] <command> [args]

The available commands for execution are listed below.
The most common, useful commands are shown first, followed by
less common or more advanced commands. If you're just getting
started with Terraform, stick with the common commands. For the
other commands, please read the help and docs before usage.

Common commands:
  apply      Builds or changes infrastructure
  console    Interactive console for Terraform interpolations
  destroy    Destroy Terraform-managed infrastructure
  env        Workspace management
  fmt        Rewrites config files to canonical format
  get        Download and install modules for the configuration
  graph     Create a visual graph of Terraform resources
  import    Import existing infrastructure into Terraform
  init      Initialize a Terraform working directory
  output    Read an output from a state file
  plan      Generate and show an execution plan
  providers Prints a tree of the providers used in the configuration
  push      Upload this Terraform module to Atlas to run
  refresh   Update local state file against real resources
  show      Inspect Terraform state or plan
  taint     Manually mark a resource for recreation
  untaint   Manually unmark a resource as tainted
  validate  Validates the Terraform files
  version   Prints the Terraform version
  workspace Workspace management

All other commands:
  debug     Debug output management (experimental)
  force-unlock Manually unlock the terraform state
  state     Advanced state management
azur
azur
azuruser@Terraform:~$
```

Authenticating Terraform with Azure

Terraform supports a number of different methods for authenticating to Azure. You can use:

- The Azure CLI
- A Managed Service Identity (MSI)
- A service principal and a client certificate
- A service principal and a client secret

When running Terraform as part of a continuous integration pipeline, you can use either an Azure service principal or MSI to authenticate.

To configure Terraform to use your Azure Active Directory (Azure AD) service principal, set the following environment variables:

- ARM_SUBSCRIPTION_ID
- ARM_CLIENT_ID
- ARM_CLIENT_SECRET
- ARM_TENANT_ID
- ARM_ENVIRONMENT

These variables are then used by the Azure Terraform modules. You can also set the environment if you are working with an Azure cloud other than an Azure public cloud.

Use the following sample shell script to set these variables:

```
#!/bin/sh
echo "Setting environment variables for Terraform"
export ARM_SUBSCRIPTION_ID=your_subscription_id
export ARM_CLIENT_ID=your_appId
export ARM_CLIENT_SECRET=your_password
export ARM_TENANT_ID=your_tenant_id

# Not needed for public, required for usgovernment, german, china
export ARM_ENVIRONMENT=public
```

- ✓ Note: After you install Terraform, before you can apply config .tf files you must run the following command to initialize Terraform for the installed instance:

```
Terraform init
```

Terraform config file structure

Take a moment to skim through the following example of a terraform .tf file. Try to identify the different elements within the file. The file performs the following actions on Azure:

- Authenticates
- Creates a resource group
- Creates a virtual network
- Creates a subnet
- Creates a public IP address
- Creates a network security group and rule
- Creates a virtual network interface card
- Generates random text for use as a unique storage account name
- Creates a storage account for diagnostics
- Creates a virtual machine

Sample Terraform .tf file

```
# Configure the Microsoft Azure Provider
provider "azurerm" {
    subscription_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
    client_id       = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
    client_secret   = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
    tenant_id       = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
}

# Create a resource group if it does not exist
resource "azurerm_resource_group" "myterraformgroup" {
    name      = "myResourceGroup"
    location = "eastus"

    tags {
```

```
        environment = "Terraform Demo"
    }
}

# Create virtual network
resource "azurerm_virtual_network" "myterraformnetwork" {
    name          = "myVnet"
    address_space = ["10.0.0.0/16"]
    location      = "eastus"
    resource_group_name = "${azurerm_resource_group.myterraformgroup.name}"

    tags {
        environment = "Terraform Demo"
    }
}

# Create subnet
resource "azurerm_subnet" "myterraformsubnet" {
    name          = "mySubnet"
    resource_group_name = "${azurerm_resource_group.myterraformgroup.name}"
    virtual_network_name = "${azurerm_virtual_network.myterraformnetwork.name}"
    address_prefix     = "10.0.1.0/24"
}

# Create public IPs
resource "azurerm_public_ip" "myterraformpublicip" {
    name          = "myPublicIP"
    location      = "eastus"
    resource_group_name = "${azurerm_resource_group.myterraformgroup.name}"
    public_ip_address_allocation = "dynamic"

    tags {
        environment = "Terraform Demo"
    }
}

# Create Network Security Group and rule
resource "azurerm_network_security_group" "myterraformnsg" {
    name          = "myNetworkSecurityGroup"
    location      = "eastus"
    resource_group_name = "${azurerm_resource_group.myterraformgroup.name}"

    security_rule {
        name          = "SSH"
        priority      = 1001
        direction     = "Inbound"
        access        = "Allow"
        protocol      = "Tcp"
    }
}
```

```
        source_port_range      = "*"
        destination_port_range = "22"
        source_address_prefix   = "*"
        destination_address_prefix = "*"
    }

    tags {
        environment = "Terraform Demo"
    }
}

# Create network interface
resource "azurerm_network_interface" "myterraformnic" {
    name          = "myNIC"
    location      = "eastus"
    resource_group_name = "${azurerm_resource_group.myterraformgroup.name}"
    network_security_group_id = "${azurerm_network_security_group.myterraformsg.id}"

    ip_configuration {
        name                = "myNicConfiguration"
        subnet_id           = "${azurerm_subnet.myterraformsubnet.id}"
        private_ip_address_allocation = "dynamic"
        public_ip_address_id      = "${azurerm_public_ip.myterraformpublicip.id}"
    }

    tags {
        environment = "Terraform Demo"
    }
}

# Generate random text for a unique storage account name
resource "random_id" "randomId" {
    keepers = {
        # Generate a new ID only when a new resource group is defined
        resource_group = "${azurerm_resource_group.myterraformgroup.name}"
    }

    byte_length = 8
}

# Create storage account for boot diagnostics
resource "azurerm_storage_account" "mystorageaccount" {
    name          = "diag${random_id.randomId.hex}"
    resource_group_name = "${azurerm_resource_group.myterraformgroup.name}"
    location      = "eastus"
    account_tier   = "Standard"
```

```
    account_replication_type      = "LRS"

    tags {
        environment = "Terraform Demo"
    }
}

# Create virtual machine
resource "azurerm_virtual_machine" "myterraformvm" {
    name                  = "myVM"
    location              = "eastus"
    resource_group_name   = "${azurerm_resource_group.myterraformgroup.name}"
    network_interface_ids = ["${azurerm_network_interface.myterraformnic.id}"]
    vm_size               = "Standard_DS1_v2"

    storage_os_disk {
        name          = "myOsDisk"
        caching       = "ReadWrite"
        create_option = "FromImage"
        managed_disk_type = "Premium_LRS"
    }

    storage_image_reference {
        publisher = "Canonical"
        offer     = "UbuntuServer"
        sku       = "16.04.0-LTS"
        version   = "latest"
    }

    os_profile {
        computer_name  = "myvm"
        admin_username = "azureuser"
    }

    os_profile_linux_config {
        disable_password_authentication = true
        ssh_keys {
            path      = "/home/azureuser/.ssh/authorized_keys"
            key_data = "ssh-rsa AAAAB3Nz{snip}hwhqT9h"
        }
    }

    boot_diagnostics {
        enabled = "true"
        storage_uri = "${azurerm_storage_account.mystorageaccount.primary_blob_endpoint}"
    }

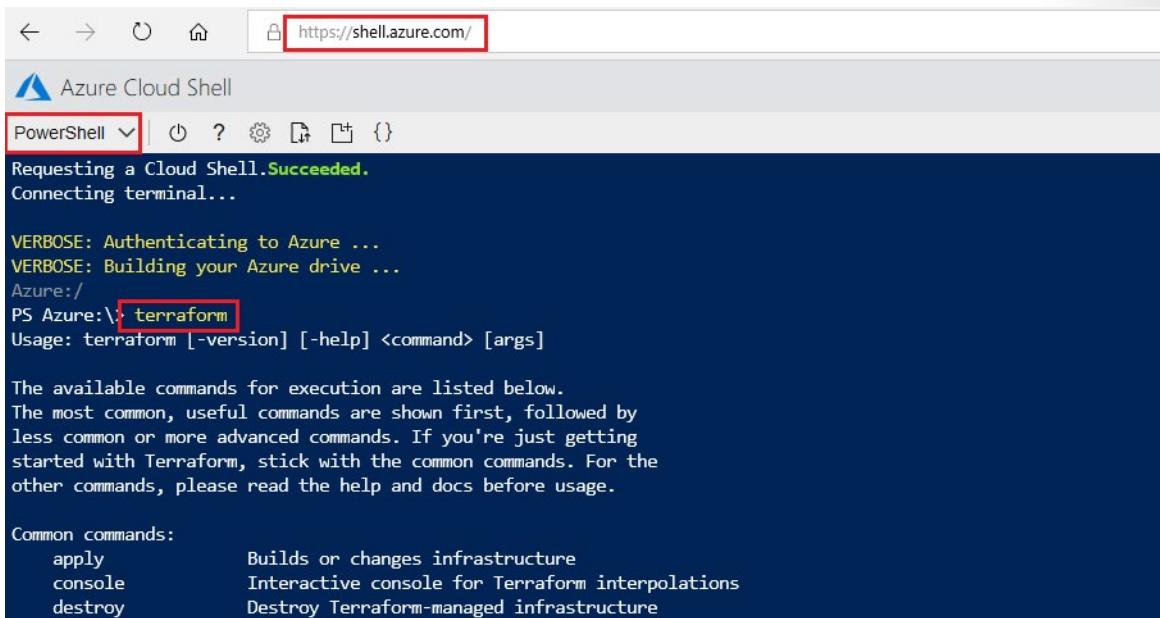
    tags {
```

```
        environment = "Terraform Demo"
    }
}
```

Demonstration-Run Terraform in Azure Cloud Shell

Terraform is pre-installed in Azure Cloud Shell, so you can use it immediately and no additional configuration is required. Because you can install Terraform on both the Windows and Linux operating systems, you can use either a PowerShell or Bash shell to run it. In this walkthrough you create a resource group in Azure using Terraform, in Azure Cloud Shell, with Bash.

The following screenshot displays Terraform running in Azure Cloud Shell with PowerShell.



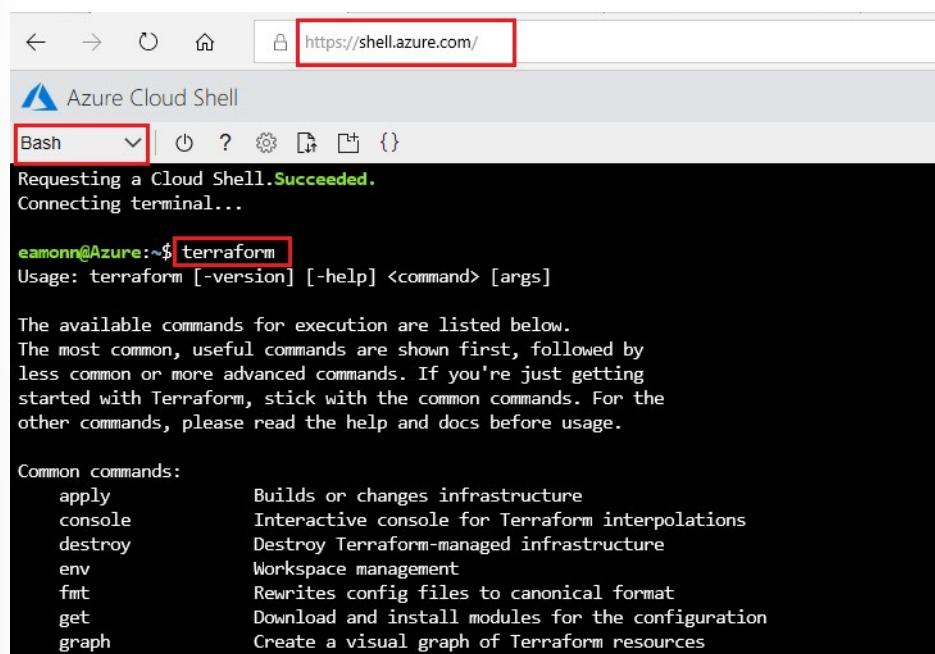
A screenshot of the Azure Cloud Shell interface. The browser address bar shows 'https://shell.azure.com/'. The shell interface itself has a header 'Azure Cloud Shell' and a dropdown menu set to 'PowerShell'. The main terminal window shows the following output:

```
Requesting a Cloud Shell. Succeeded.
Connecting terminal...
VERBOSE: Authenticating to Azure ...
VERBOSE: Building your Azure drive ...
Azure:/
PS Azure:\> terraform
Usage: terraform [-version] [-help] <command> [args]

The available commands for execution are listed below.
The most common, useful commands are shown first, followed by
less common or more advanced commands. If you're just getting
started with Terraform, stick with the common commands. For the
other commands, please read the help and docs before usage.

Common commands:
  apply           Builds or changes infrastructure
  console         Interactive console for Terraform interpolations
  destroy         Destroy Terraform-managed infrastructure
```

The following image is an example of running Terraform in Azure Cloud Shell with a Bash shell.



The screenshot shows the Azure Cloud Shell interface. The title bar says "Azure Cloud Shell". The tab bar has "Bash" selected, indicated by a red box. The URL bar shows "https://shell.azure.com/" with a red box around it. The main area displays the output of a Terraform command:

```
Requesting a Cloud Shell.Succeeded.
Connecting terminal...
eamonn@Azure:~$ terraform
Usage: terraform [-version] [-help] <command> [args]

The available commands for execution are listed below.
The most common, useful commands are shown first, followed by
less common or more advanced commands. If you're just getting
started with Terraform, stick with the common commands. For the
other commands, please read the help and docs before usage.

Common commands:
  apply      Builds or changes infrastructure
  console    Interactive console for Terraform interpolations
  destroy   Destroy Terraform-managed infrastructure
  env       Workspace management
  fmt        Rewrites config files to canonical format
  get        Download and install modules for the configuration
  graph     Create a visual graph of Terraform resources
```

Editor

You can also use the Azure Cloud Shell editor to review, open, and edit your .tf files. To open the editor, select the braces on the Azure Cloud Shell taskbar.

```

https://shell.azure.com/
Azure Cloud Shell
Bash | ⚡ ? 🛡 🏷️ 📁 🎯
FILESTerraform-createrg.tf
.bashrc
.profile
.selected_editor
.tmux.conf
.viminfo
azuredploy.json
cloud-init.txt
linkedStorageAccount.json
playbook1.yml
resgrpcreate1.yml
rg.yml
rg1.yml
subscriptionterraform.sh
unscriptionterraform.sh
terraform-createrg.tf
test.rf
test.tf
vnet1.yml
All other commands:
debug           Debug output management (experimental)
force-unlock   Manually unlock the terraform state
state          Advanced state management
eamonn@Azure:~$ vi terraform-createrg.tf
eamonn@Azure:~$ []

```

Prerequisites

- You do require an Azure subscription to perform these steps. If you don't have one you can create one by following the steps outlined on the [Create your Azure free account today¹⁹](#) webpage.

Steps

The following steps outline how to create a resource group in Azure using Terraform in Azure Cloud Shell, with bash.

- Open the Azure Cloud Shell at <https://shell.azure.com>. You can also launch Azure Cloud Shell from within the Azure portal by selecting the Azure Cloud Shell icon.
- If prompted, authenticate to Azure by entering your credentials.
- In the taskbar, ensure that **Bash** is selected as the shell type.
- Create a new .tf file and open the file for editing with the following command:
`vi terraform-createrg.tf`
- Enter **insert** mode by selecting the **I** key.
- Copy and paste the following code into the file:

¹⁹ https://azure.microsoft.com/en-us/free/?ref=microsoft.com&utm_source=microsoft.com&utm_medium=docs&utm_campaign=visualstudio

```
provider "azurerm" {
}
resource "azurerm_resource_group" "rg" {
    name = "testResourceGroup"
    location = "westus"
}
```

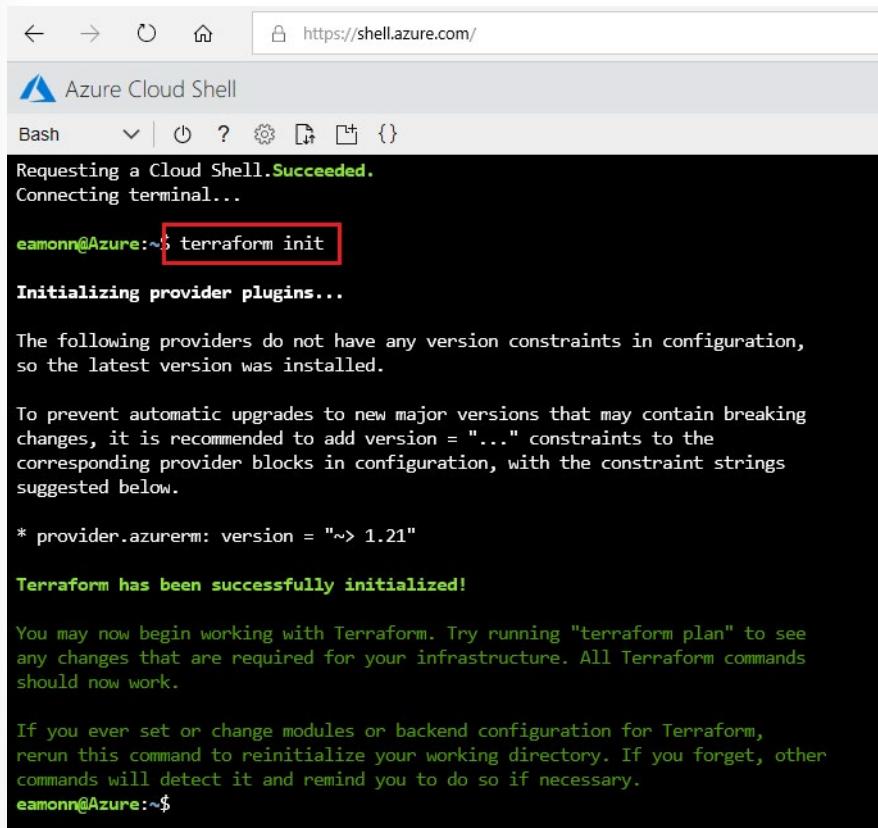
7. Exit **insert** mode by selecting the **Esc** key.
8. Save the file and exit the **vi** editor by entering the following command:

```
:wq
```

9. Use the following command to initialize Terraform:

```
terraform init
```

You should receive a message saying Terraform was successfully initiated.



The screenshot shows a terminal window in the Azure Cloud Shell. The URL bar at the top shows <https://shell.azure.com/>. The terminal itself has a header bar with icons for back, forward, refresh, and a gear. The main area shows the following text output:

```
← → ⏪ https://shell.azure.com/
Azure Cloud Shell
Bash | ⚡ ? ⚙ { }

Requesting a Cloud Shell.Succeeded.
Connecting terminal...

eamonn@Azure:~$ terraform init
Initializing provider plugins...
The following providers do not have any version constraints in configuration,
so the latest version was installed.

To prevent automatic upgrades to new major versions that may contain breaking
changes, it is recommended to add version = "..." constraints to the
corresponding provider blocks in configuration, with the constraint strings
suggested below.

* provider_azurerm: version = "~> 1.21"

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
eamonn@Azure:~$
```

10. Run the configuration .tf file with the following command:

```
terraform apply
```

You should receive a prompt to indicate that a plan has been generated. Details of the changes should be listed, followed by a prompt to apply or cancel the changes.

The screenshot shows the Azure Cloud Shell interface. The URL bar at the top displays <https://shell.azure.com/>. The main area shows the terminal session:

```

Azure Cloud Shell
Bash | ⚡ ? 🌐 🔍 { }

Requesting a Cloud Shell. Succeeded.
Connecting terminal...

eamonn@Azure:~$ terraform apply
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

+ azurerm_resource_group.rg
  id:      <computed>
  location: "westus"
  name:    "terraform-rg1"
  tags.%:  <computed>

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value:

```

11. Enter a value of **yes**, and then select Enter. The command should run successfully, with output similar to the following screenshot.

The screenshot shows the terminal output after entering 'yes' and pressing Enter. The output includes the creation of a resource group and a success message.

```

Enter a value: yes

azurerm_resource_group.rg: Creating...
  location: "" => "westus"
  name:    "" => "terraform-rg1"
  tags.%:  "" => "<computed>"
azurerm_resource_group.rg: Creation complete after 2s (ID: /subscriptions/subscriptionId/resourceGroups/terraform-rg1)

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
eamonn@Azure:~$ 

```

12. Open Azure portal and verify the new resource group now displays in the portal.

Demonstration-Run Terraform in Visual Studio Code

You can also run Terraform configuration files using Visual Studio Code. This leverages other Terraform services that you can integrate with Visual Studio Code. Two Visual Studio code extensions that are required, are **Azure Account**, and **Terraform**.

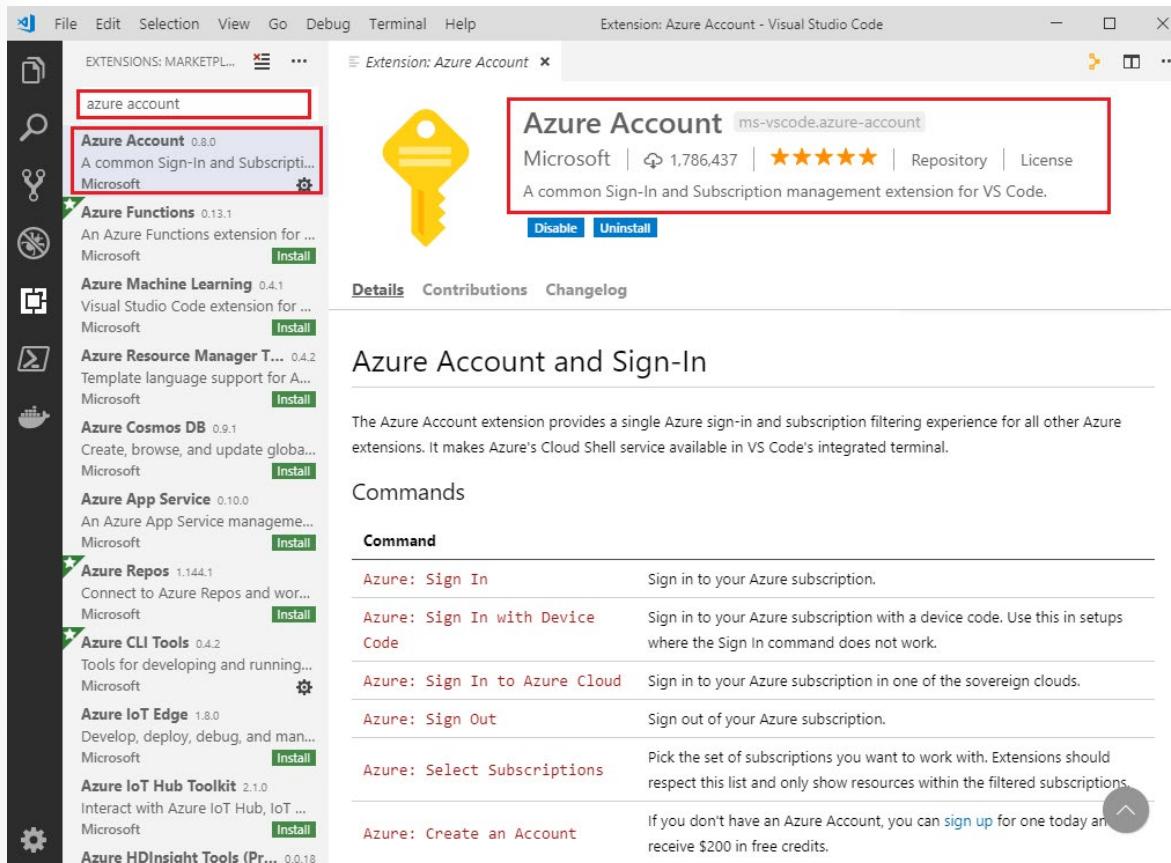
In this walkthrough you will create a VM in Visual Studio Code using Terraform

Prerequisites

- This walkthrough requires Visual Studio Code. If you do not have Visual Studio Code installed, you can download it from <https://code.visualstudio.com/>²⁰. Download and install a version of Visual Studio Code that is appropriate to your operating system environment, for example Windows, Linux, or macOS.
- You will require an active Azure subscription to perform the steps in this walkthrough. If you do not have one, create an Azure subscription by following the steps outlined on the [Create your Azure free account today](#)²¹ webpage.

Steps

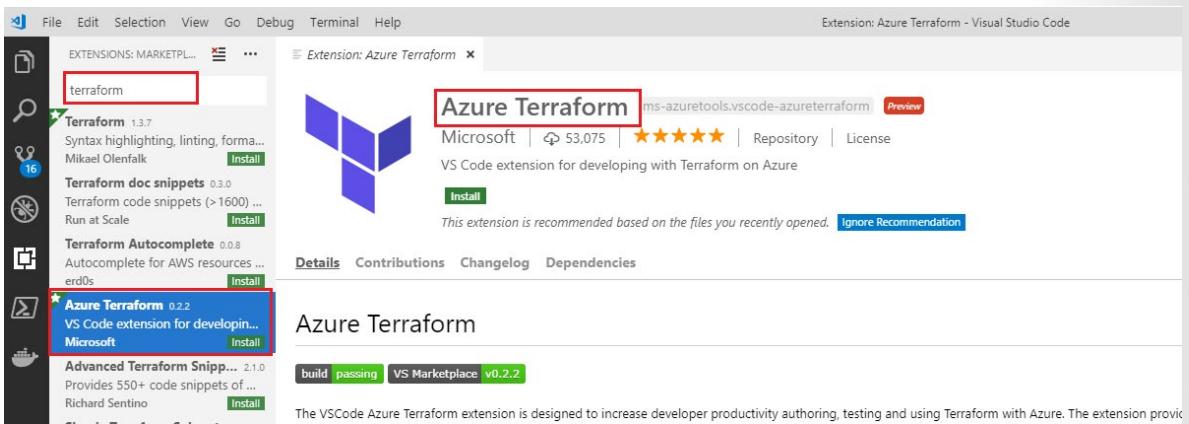
- Launch the Visual Studio Code editor.
- The two Visual Studio Code extensions *Azure Account* and *Azure Terraform* must be installed. To install the first extension, from inside Visual Studio Code, select **File > Preferences > Extensions**.
- Search for and install the extension **Azure Account**.



- Search for and install the extension **Terraform**. Ensure that you select the extension authored by Microsoft, as there are similar extensions available from other authors

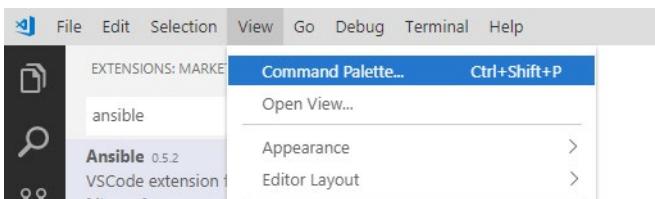
²⁰ <https://code.visualstudio.com/>

²¹ https://azure.microsoft.com/en-us/free/?ref=microsoft.com&utm_source=microsoft.com&utm_medium=docs&utm_campaign=visualstudio

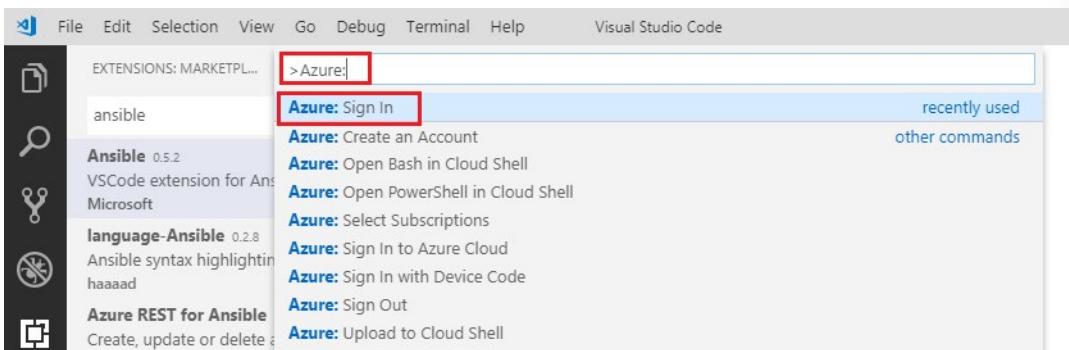


You can view more details of this extension at the Visual Studio Marketplace on the [Azure Terraform²²](#) page.

5. In Visual Studio Code, open the command palette by selecting **View > Command Palette**. You can also access the command palette by selecting the **settings** (cog) icon on the bottom, left side of the **Visual Studio Code** window, and then selecting **Command Palette**.

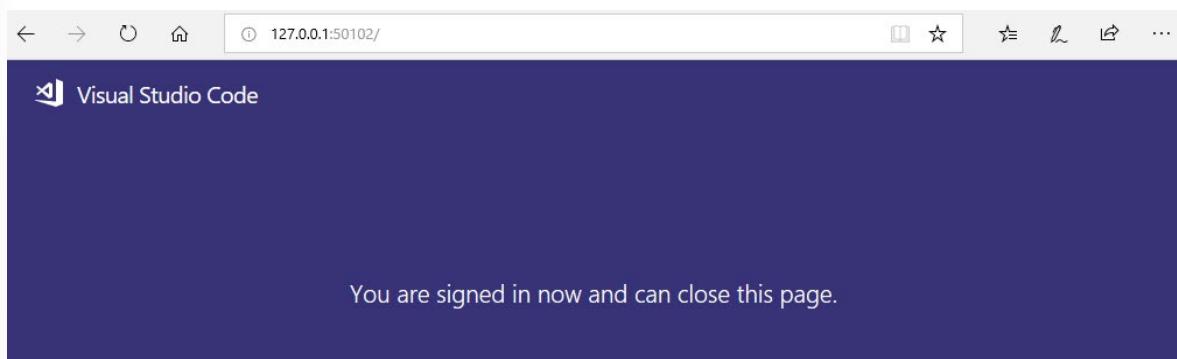


6. In the Command Palette search field, type **Azure:**, and from the results, select **Azure: Sign In**.



7. When a browser launches and prompts you to sign in to Azure, select your Azure account. The message *You are signed in now and can close this page.*, should display in the browser.

²² <https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-azurterraform>



8. Verify that your Azure account now displays at the bottom of the Visual Studio Code window.

9. Create a new file, then copy the following code and paste it into the file.

```
# Create a resource group if it doesn't exist
resource "azurerm_resource_group" "myterraformgroup" {
    name      = "terraform-rg2"
    location  = "eastus"

    tags {
        environment = "Terraform Demo"
    }
}

# Create virtual network
resource "azurerm_virtual_network" "myterraformnetwork" {
    name          = "myVnet"
    address_space = ["10.0.0.0/16"]
    location      = "eastus"
    resource_group_name = "${azurerm_resource_group.myterraformgroup.name}"

    tags {
        environment = "Terraform Demo"
    }
}

# Create subnet
resource "azurerm_subnet" "myterraformsubnet" {
    name          = "mySubnet"
    resource_group_name = "${azurerm_resource_group.myterraformgroup.name}"
    virtual_network_name = "${azurerm_virtual_network.myterraformnetwork.name}"
    address_prefix     = "10.0.1.0/24"
}

# Create public IPs
```

```
resource "azurerm_public_ip" "myterraformpublicip" {
    name          = "myPublicIP"
    location      = "eastus"
    resource_group_name = "${azurerm_resource_group.myterraformgroup.name}"
    public_ip_address_allocation = "dynamic"

    tags {
        environment = "Terraform Demo"
    }
}

# Create Network Security Group and rule
resource "azurerm_network_security_group" "myterraformmsg" {
    name          = "myNetworkSecurityGroup"
    location      = "eastus"
    resource_group_name = "${azurerm_resource_group.myterraformgroup.name}"

    security_rule {
        name          = "SSH"
        priority      = 1001
        direction     = "Inbound"
        access         = "Allow"
        protocol       = "Tcp"
        source_port_range = "*"
        destination_port_range = "22"
        source_address_prefix = "*"
        destination_address_prefix = "*"
    }

    tags {
        environment = "Terraform Demo"
    }
}

# Create network interface
resource "azurerm_network_interface" "myterraformnic" {
    name          = "myNIC"
    location      = "eastus"
    resource_group_name = "${azurerm_resource_group.myterraformgroup.name}"
    network_security_group_id = "${azurerm_network_security_group.myterraformmsg.id}"

    ip_configuration {
        name          = "myNicConfiguration"
        subnet_id     = "${azurerm_subnet.myterraformsubnet.id}"
        private_ip_address_allocation = "dynamic"
        public_ip_address_id        = "${azurerm_public_ip.myterraformpublicip.id}"
    }
}
```

```
    }

    tags {
        environment = "Terraform Demo"
    }
}

# Generate random text for a unique storage account name
resource "random_id" "randomId" {
    keepers = {
        # Generate a new ID only when a new resource group is defined
        resource_group = "${azurerm_resource_group.myterraformgroup.name}"
    }

    byte_length = 8
}

# Create storage account for boot diagnostics
resource "azurerm_storage_account" "mystorageaccount" {
    name                  = "diag${random_id.randomId.hex}"
    resource_group_name   = "${azurerm_resource_group.myterraformgroup.name}"
    location              = "eastus"
    account_tier          = "Standard"
    account_replication_type = "LRS"

    tags {
        environment = "Terraform Demo"
    }
}

# Create virtual machine
resource "azurerm_virtual_machine" "myterraformvm" {
    name                  = "myVM"
    location              = "eastus"
    resource_group_name   = "${azurerm_resource_group.myterraformgroup.name}"
    network_interface_ids = ["${azurerm_network_interface.myterraformnic.id}"]
    vm_size               = "Standard_DS1_v2"

    storage_os_disk {
        name      = "myOsDisk"
        caching   = "ReadWrite"
        create_option = "FromImage"
        managed_disk_type = "Premium_LRS"
    }

    storage_image_reference {
        publisher = "Canonical"
        offer     = "UbuntuServer"
    }
}
```

```

        sku          = "16.04.0-LTS"
        version     = "latest"
    }

    os_profile {
        computer_name  = "myvm"
        admin_username = "azureuser"
        admin_password = "Password0134!"
    }

    os_profile_linux_config {
        disable_password_authentication = false
    }
}

boot_diagnostics {
    enabled = "true"
    storage_uri = "${azurerm_storage_account.mystorageaccount.primary_blob_endpoint}"
}

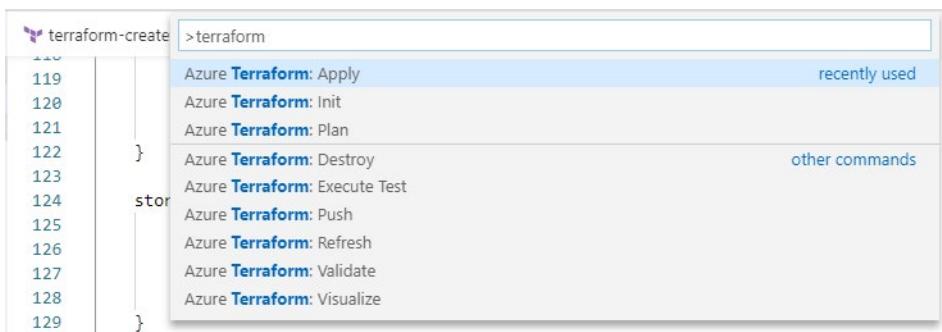
tags {
    environment = "Terraform Demo"
}
}

```

10. Save the file locally with the file name `terraform-createvm.tf`.

11. In Visual Studio Code, select **View > Command Palette**. Search for the command by entering **terraform** into the search field. Select the following command from the dropdown list of commands:

Azure Terraform: apply



12. If Azure Cloud Shell is not open in Visual Studio Code, a message might appear in the bottom, left corner asking you if you want to open Azure Cloud Shell. Choose **Accept**, and select **Yes**.

13. Wait for the Azure Cloud Shell pane to appear in the bottom of Visual Studio Code window, and start running the file `terraform-createvm.tf`. When you are prompted to apply the plan or cancel, type **Yes**, and then press **Enter**.

```

File Edit Selection View Go Debug Terminal Help
terraformer-createvm.tf - linked - Visual Studio Code

EXPLORER
OPEN EDITORS
terraform-createvm.tf
LINKED
! ansiblelocalhostping.yml
! azuredploy.json
! createavm.yml
linkedtemplate.json
! rg1.yml
terraformer-createvm.tf

storage_os_disk {
  name      = "myOsDisk"
  caching   = "ReadWrite"
  create_option = "FromImage"
  managed_disk_type = "Premium_LRS"
}

storage_image_reference {
  publisher = "Canonical"
  offer     = "UbuntuServer"
  sku       = "16.04.0-LTS"
  version   = "latest"
}

os_profile {
  computer_name = "myVm"
  admin_username = "azureuser"
  admin_password = "Password0134!"
}

os_profile_linux_config {
  disable_password_authentication = true
}

storage_image_reference.363552096.publisher: "Canonical"
storage_image_reference.363552096.sku: "16.04.0-LTS"
storage_image_reference.363552096.version: "latest"
storage_os_disk.#: "1"
storage_os_disk.0.caching: "ReadWrite"
storage_os_disk.0.create_option: "FromImage"
storage_os_disk.0.disk_size_gb: "<computed>"
storage_os_disk.0.managed_disk_id: "<computed>"
storage_os_disk.0.managed_disk_type: "Premium_LRS"
storage_os_disk.0.name: "myOsDisk"
storage_os_disk.0.write_accelerator_enabled: "false"
tags.%: "1"
tags.environment: "Terraform Demo"
vm_size: "Standard_DS1_v2"

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes
  
```

14. After the command completes successfully, review the list of resources created.

```

PROBLEMS 36 OUTPUT DEBUG CONSOLE TERMINAL
1: Bash in Cloud Shell + - x

storage_os_disk.0.caching: "" => "ReadWrite"
storage_os_disk.0.create_option: "" => "FromImage"
storage_os_disk.0.disk_size_gb: "" => "<computed>"
storage_os_disk.0.managed_disk_id: "" => "<computed>"
storage_os_disk.0.managed_disk_type: "" => "Premium LRS"
storage_os_disk.0.name: "" => "myOsDisk"
storage_os_disk.0.write_accelerator_enabled: "" => "false"
tags.%: "" => "1"
tags.environment: "" => "Terraform Demo"
vm_size: "" => "Standard_DS1_v2"

azurerm_virtual_machine.myterraformvm: Still creating... (10s elapsed)
azurerm_virtual_machine.myterraformvm: Still creating... (20s elapsed)
azurerm_virtual_machine.myterraformvm: Still creating... (30s elapsed)
azurerm_virtual_machine.myterraformvm: Still creating... (40s elapsed)
azurerm_virtual_machine.myterraformvm: Still creating... (50s elapsed)
azurerm_virtual_machine.myterraformvm: Still creating... (1m0s elapsed)
azurerm_virtual_machine.myterraformvm: Still creating... (1m10s elapsed)
azurerm_virtual_machine.myterraformvm: Still creating... (1m20s elapsed)
azurerm_virtual_machine.myterraformvm: Still creating... (1m30s elapsed)
azurerm_virtual_machine.myterraformvm: Creation complete after 1m33s (ID: /subscriptions/6e9a285a-37ea-40e6-b2fc-...Microsoft.Compute/virtualMachines/myVm)

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
earon@Azure:~/clouddrive/linked$ 
  
```

15. Open the Azure Portal and verify the resource group, resources, and the VM has been created. If you have time, sign in with the user name and password specified in the .tf config file to verify.

| NAME | TYPE | LOCATION | ... |
|------------------------|------------------------|----------|-----|
| diag763291beb6ee1511 | Storage account | East US | ... |
| myNetworkSecurityGroup | Network security group | East US | ... |
| myNIC | Network interface | East US | ... |
| myOsDisk | Disk | East US | ... |
| myPublicIP | Public IP address | East US | ... |
| myVM | Virtual machine | East US | ... |
| myVnet | Virtual network | East US | ... |

Note: If you wanted to use a public or private key pair to connect to the Linux VM instead of a user name and password, you could use the **os_profile_linux_config** module, set the **disable_password_authentication** key value to **true** and include the ssh key details, as in the following code.

```
os_profile_linux_config {
    disable_password_authentication = true
    ssh_keys {
        path      = "/home/azureuser/.ssh/authorized_keys"
        key_data = "ssh-rsa AAAAB3Nz{snip}hwhqT9h"
    }
}
```

You'd also need to remove the password value in the **os_profile** module that present in the example above.

Note: You could also embed the Azure authentication within the script. In that case, you would not need to install the Azure account extension, as in the following example:

```
provider "azurerm" {
    subscription_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
    client_id       = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
    client_secret   = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
    tenant_id       = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
}
```

Labs

Infrastructure as Code



Steps for the labs are available on **GitHub** at the below sites under the **Infrastructure as Code** sections

- <https://microsoft.github.io/PartsUnlimited>
- <https://microsoft.github.io/PartsUnlimitedMRP>

You should click on the links below, for the individual lab tasks for this module, and follow the steps outlined there for each lab task.

PartsUnlimitedMRP (PUMRP)

- [Deploy app with Chef on Azure²³](#)
- [Deploy app with Puppet on Azure²⁴](#)
- [Ansible with Azure²⁵](#)

Automating your infrastructure deployments

Overview

Terraform²⁶ is a tool for building, changing, and versioning infrastructure safely and efficiently. Terraform can manage existing and popular cloud service providers as well as custom in-house solutions.

Configuration files describe to Terraform the components needed to run a single application or your entire datacenter. Terraform generates an execution plan describing what it will do to reach the desired state, and then executes it to build the described infrastructure. As the configuration changes, Terraform is able to determine what changed and create incremental execution plans which can be applied.

What's covered in this lab

In the lab, **Automating infrastructure deployments in the Cloud with Terraform and Azure Pipelines**²⁷, you will see:

- How open source tools, such as Terraform can be leveraged to implement Infrastructure as Code (IaC)
- How to automate your infrastructure deployments in the Cloud with Terraform and Azure Pipelines

²³ <http://microsoft.github.io/PartsUnlimitedMRP/iac/200.2x-IaC-DeployappwithChefonAzure.html>

²⁴ <http://microsoft.github.io/PartsUnlimitedMRP/iac/200.2x-IaC-DeployappwithPuppetonAzure.html>

²⁵ <http://microsoft.github.io/PartsUnlimitedMRP/iac/200.2x-IaC-AnsiblewithAzure.html>

²⁶ <https://www.terraform.io/intro/index.html>

²⁷ <https://azuredovplabs.com/labs/vstsextend/terraform/>

Module Review and Takeaways

Module Review Questions

Checkbox

*Which of the following are main architectural components of Chef?
(choose all that apply)*

- Chef Server
- Chef Facts
- Chef Client
- Chef Workstation

Checkbox

Which of the following are open-source products that are integrated into the Chef Automate image available from Azure Marketplace?

- Habitat
- Facts
- Console Services
- InSpec

Checkbox

*Which of the following are core components of the Puppet automation platform?
(choose all that apply)*

- Master
- Agent
- Facts
- Habitat

Dropdown

Complete the following sentence.

The main elements of a Puppet Program (PP) Manifest file are Class, Resource and _____.

- Module
- Habitat
- InSpec
- Cookbooks

Checkbox

*Which of the following platforms use Agents to communicate with target machines?
(choose all that apply)*

- Puppet
- Chef
- Ansible

Multiple choice

True or false: The Control Machine in Ansible must have Python installed?

- True
- False

Checkbox

Which of the following statements about the cloud-init package are correct?

- The --custom-data parameter passes the name of the configuration file (.txt).
- Configuration files (.txt) are encoded in base64.
- The YML syntax is used within the configuration file (.txt).
- cloud-init works across Linux distributions.

Multiple choice

True or false: Terraform ONLY supports configuration files with the file extension .tf.

- True
- False

Multiple choice

Which of the following core Terraform components can modify Terraform behavior, without having to edit the Terraform configuration?

- Configuration files
- Overrides
- Execution plan
- Resource graph

Answers

Checkbox

Which of the following are main architectural components of Chef?
(choose all that apply)

- Chef Server
- Chef Facts
- Chef Client
- Chef Workstation

Explanation

The correct answers are Chef Server, Chef Client and Chef Workstation.

Chef Facts is an incorrect answer.

Chef Facts is not an architectural component of Chef. Chef Facts misrepresents the term 'Puppet Facts'. Puppet Facts are metadata used to determine the state of resources managed by the Puppet automation tool.

Chef has the following main architectural components. 'Chef Server' is the Chef management point. The two options for the Chef Server are 'hosted' and 'on-premises'. 'Chef Client (node)' is an agent that sits on the servers you are managing. 'Chef Workstation' is an Administrator workstation where you create Chef policies and execute management commands. You run the Chef 'knife' command from the Chef Workstation to manage your infrastructure.

Checkbox

Which of the following are open-source products that are integrated into the Chef Automate image available from Azure Marketplace?

- Habitat
- Facts
- Console Services
- InSpec

Explanation

The correct answers are Habitat and InSpec.

Facts and Console Services are incorrect answers.

Facts are metadata used to determine the state of resources managed by the Puppet automation tool. Console Services is a web-based user interface for managing your system with the Puppet automation tool. Habitat and InSpec are two open-source products that are integrated into the Chef Automate image available from Azure Marketplace. Habitat makes the application and its automation the unit of deployment, by allowing you to create platform-independent build artifacts called 'habitats' for your applications. InSpec allows you to define desired states for your applications and infrastructure. InSpec can conduct audits to detect violations against your desired state definitions, and generate reports from its audit results.

Checkbox

Which of the following are core components of the Puppet automation platform?
(choose all that apply)

- Master
- Agent
- Facts
- Habitat

Explanation

The correct answers are Master, Agent and Facts.

Habitat is an incorrect answer.

Habitat is used with Chef for creating platform-independent build artifacts called for your applications. Master, Agent and Facts are core components of the Puppet automation platform. Another core component is 'Console Services'. Puppet Master acts as a center for Puppet activities and processes. Puppet Agent runs on machines managed by Puppet, to facilitate management. Console Services is a toolset for managing and configuring resources managed by Puppet. Facts are metadata used to determine the state of resources managed by Puppet.

Dropdown

Complete the following sentence.

The main elements of a Puppet Program (PP) Manifest file are Class, Resource and _____.

- Module
- Habitat
- InSpec
- Cookbooks

Explanation

Module is the correct answer.

All other answers are incorrect answers.

Habitat, InSpec and Cookbooks are incorrect because they relate to the Chef automation platform.

The main elements of a Puppet Program (PP) Manifest file are Class, Resource and Module. Classes define related resources according to their classification, to be reused when composing other workflows. Resources are single elements of your configuration which you can specify parameters for. Modules are collections of all the classes, resources and other elements in a single entity.

Checkbox

Which of the following platforms use Agents to communicate with target machines?
(choose all that apply)

- Puppet
- Chef
- Ansible

Explanation

The correct answers are: Puppet and Chef.

Ansible is an incorrect answer.

Ansible is agentless because you do not need to install an Agent on each of the target machines it manages. Ansible uses the Secure Shell (SSH) protocol to communicate with target machines. You choose when to conduct compliance checks and perform corrective actions, instead of using Agents and a Master to perform

these operations automatically.

Puppet and Chef use Agents to communicate with target machines. With Puppet and Chef, you install an Agent on each target machine managed by the platform. Agents typically run as a background service and facilitate communication with a Master, which runs on a server. The Master uses information provided by Agents to conduct compliance checks and perform corrective actions automatically.

Multiple choice

True or false: The Control Machine in Ansible must have Python installed?

- True
- False

Explanation

A Control Machine in Ansible must have Python installed. Control Machine is one of the core components of Ansible. Control Machine is for running configurations. The other core components of Ansible are Managed Nodes, Playbooks, Modules, Inventory, Roles, Facts, and Plug-ins. Managed Nodes are resources managed by Ansible. Playbooks are ordered lists of Ansible tasks. Modules are small blocks of code within a Playbook that perform specific tasks. Inventory is list of managed nodes. Roles allow for the automatic and sequenced loading of variables, files, tasks and handlers. Facts are data points about the remote system which Ansible is managing. Plug-ins supplement Ansible's core functionality.

Checkbox

Which of the following statements about the cloud-init package are correct?

- The --custom-data parameter passes the name of the configuration file (.txt).
- Configuration files (.txt) are encoded in base64.
- The YML syntax is used within the configuration file (.txt).
- cloud-init works across Linux distributions.

Explanation

All of the answers are correct answers.

In Azure, you can add custom configurations to a Linux VM with cloud-init by appending the --custom-data parameter, and passing the name of a configuration file (.txt), to the az vm create command. The --custom-data parameter passes the name of the configuration file (.txt) as an argument to cloud-init. Then, cloud-init applies Base64 encoding to the contents of the configuration file (.txt), and sends it along with any provisioning configuration information that is contained within the configuration file (.txt). Any provisioning configuration information contained in the specified configuration file (.txt) is applied to the new VM, when the VM is created. The YML syntax is used within the configuration file (.txt) to define any provisioning configuration information that needs to be applied to the VM.

Multiple choice

True or false: Terraform ONLY supports configuration files with the file extension .tf.

- True
- False

Explanation

False is the correct answer.

True is an incorrect answer because Terraform supports configuration files with the file extensions .tf and .tf.json.

Terraform configuration files are text based configuration files that allow you to define infrastructure and application configurations. Terraform uses the file extension .tf for Terraform format configuration files, and

the file extension .tf.json for Terraform JSON format configuration files. Terraform supports configuration files in either .tf or .tf.json format. The Terraform .tf format is more human-readable, supports comments, and is the generally recommended format for most Terraform files. The JSON format .tf.json is meant for use by machines, but you can write your configuration files in JSON format if you prefer.

Multiple choice

Which of the following core Terraform components can modify Terraform behavior, without having to edit the Terraform configuration?

- Configuration files
- Overrides
- Execution plan
- Resource graph

Explanation

Overrides is the correct answer.

All other answers are incorrect answers.

Configuration files, in .tf or .tf.json format, allow you to define your infrastructure and application configurations with Terraform.

Execution plan defines what Terraform will do when a configuration is applied.

Resource graph builds a dependency graph of all Terraform managed resources.

Overrides modify Terraform behavior without having to edit the Terraform configuration. Overrides can also be used to apply temporary modifications to Terraform configurations without having to modify the configuration itself.

Module 18 Implement Compliance and Security in your Infrastructure

Module Overview

Module Overview

As many as four out of five companies leveraging a DevOps approach to software engineering do so without integrating the necessary information security controls, underscoring the urgency with which companies should be evaluating “Rugged” DevOps (also known as “shift left”) to build security into their development life cycle as early as possible.

Rugged DevOps represents an evolution of DevOps in that it takes a mode of development in which speed and agility are primary and integrates security, not just with automated tools and processes, but also through cultural change emphasizing ongoing, flexible collaboration between release engineers and security teams. The goal is to bridge the traditional gap between the two functions, reconciling rapid deployment of code with the imperative for security.

For many companies, a common pitfall on the path to implementing rugged DevOps is implementing the approach all at once rather than incrementally, underestimating the complexity of the undertaking and producing cultural disruption in the process. Putting these plans in place is not a one-and-done process; instead the approach should continuously evolve to support the various scenarios and needs that DevOps teams encounter. The building blocks for Rugged DevOps involves understanding and implementation of the following concepts,

- Code Analysis
- Change Management
- Compliance Monitoring
- Threat Investigation
- Vulnerability assessment & KPIs

Learning Objectives

After completing this module, students will be able to:

- Define an infrastructure and configuration strategy and appropriate toolset for a release pipeline and application infrastructure
- Implement compliance and security in your application infrastructure

Security and Compliance Principles with Dev-Ops

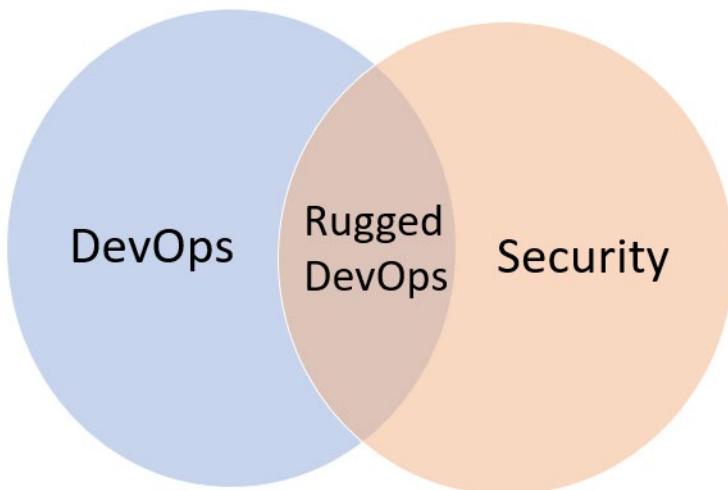
What is Rugged DevOps

While the adoption of cloud computing is on the rise to support business productivity, a lack of security infrastructure can result in inadvertently compromising data. The 2018 Microsoft Security Intelligence Report **Understand top trends in the threat landscape¹** finds that:

- Data is **not** encrypted both at rest and in transit by:
 - 7% of software as a service (SaaS) storage apps.
 - 86% percent of SaaS collaboration apps.
- HTTP headers session protection is supported by **only**:
 - 4% of SaaS storage apps.
 - 3% of SaaS collaboration apps.

Rugged DevOps (or DevSecOps)

DevOps is about working faster. *Security* is about emphasizing thoroughness. Security concerns are typically addressed at the end of the cycle. This can potentially create unplanned work right at the end of the pipeline. *Rugged DevOps* integrates DevOps with security into a set of practices that are designed to meet the goals of both DevOps and security more effectively.



The goal of a Rugged DevOps pipeline is to allow development teams to work fast without breaking their project by introducing unwanted security vulnerabilities.

Note: Rugged DevOps is also sometimes referred to as *DevSecOps*. You might encounter both terms, but each term refers to the same concept.

¹ <https://www.microsoft.com/en-us/security/operations/security-intelligence-report>

Security in the context of Rugged DevOps

Historically, security typically operated on a slower cycle and involved traditional security methodologies, such as:

- Access control
- Environment hardening
- Perimeter protection

Rugged DevOps includes these traditional security methodologies, and more. With Rugged DevOps, security is about securing the pipeline. Rugged DevOps involves determining where you can add security to the elements that plug into your build and release pipelines. Rugged DevOps can show you how and where you can add security to your automation practices, production environments, and other pipeline elements, while benefiting from the speed of DevOps.

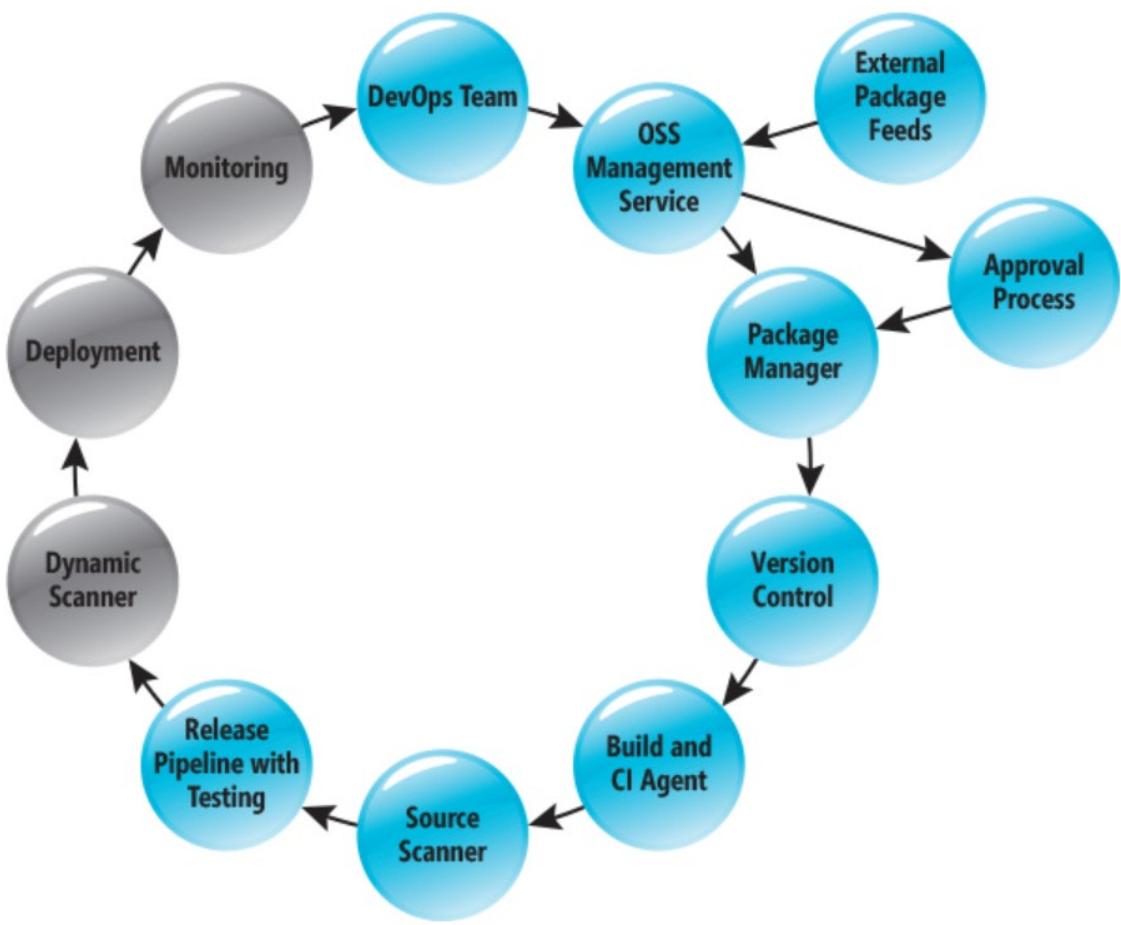
Rugged DevOps addresses broader questions, such as:

- Is my pipeline consuming third-party components, and if so, are they secure?
- Are there known vulnerabilities within any of the third-party software we use?
- How quickly can I detect vulnerabilities (also referred to as *time to detect*)?
- How quickly can I remediate identified vulnerabilities (also referred to as *time to remediate*)?

Security practices for detecting potential security anomalies need to be as robust and as fast as the other parts of your DevOps pipeline, including infrastructure automation and code development.

Rugged DevOps pipeline

As previously stated, the goal of a *Rugged DevOps* pipeline is to enable development teams to work fast without introducing unwanted vulnerabilities into their project.



Two important features of Rugged DevOps pipelines that are not found in standard DevOps pipelines are:

- Package management and the approval process associated with it. The previous workflow diagram details additional steps that account for how software packages are added to the pipeline, and the approval processes that packages must go through before they are used. These steps should be enacted early in the pipeline, so that issues can be identified sooner in the cycle.
- Source Scanner, also an additional step for scanning the source code. This step allows for security scanning and checking for security vulnerabilities that are not present in the application code. The scanning occurs *after* the app is built, but *before* release and pre-release testing. Source scanning can identify security vulnerabilities earlier in the cycle.

In the remainder of this lesson, we address these two important features of Rugged DevOps pipelines, the problems they present, and some of the solutions for them.

Software Composition Analysis (SCA)

Two important areas from the Rugged DevOps pipeline are Package management and Open Source Software OSS components.

Package management

Just as teams use version control as a single source of truth for source code, Rugged DevOps relies on a package manager as the unique source of binary components. By using binary package management, a

development team can create a local cache of approved components, and make this a trusted feed for the Continuous Integration (CI) pipeline.

In Azure DevOps, *Azure Artifacts* is an integral part of the component workflow for organizing and sharing access to your packages. Azure Artifacts allows you to:

- Keep your artifacts organized. Share code easily by storing Apache Maven, npm, and NuGet packages together. You can store packages using Universal Packages, eliminating the need to store binaries in Git.
- Protect your packages. Keep every public source package you use (including packages from npmjs and NuGet .org) safe in your feed where only you can delete it and where it's backed by the enterprise-grade Azure Service Level Agreement (SLA).
- Integrate seamless package handling into your Continuous Integration (CI)/ Continuous Development (CD) pipeline. Easily access all your artifacts in builds and releases. Azure Artifacts integrates natively with the Azure Pipelines CI/CD tool.

For more information about Azure Artifacts, visit the webpage [What is Azure Artifacts?](#)²

Versions and compatibility

The following table lists the package types supported by Azure Artifacts. The availability of each package in *Azure DevOps Services* also displays. The following table details the compatibility of each package with specific versions of Azure DevOps Server, previously known as *Team Foundation Server* (TFS).

| Feature | Azure DevOps Services | TFS |
|-----------|-----------------------|-----------------------------|
| NuGet | Yes | TFS 2017 |
| npm | Yes | TFS 2017 update 1 and later |
| Maven | Yes | TFS 2017 update 1 and later |
| Gradle | Yes | TFS 2018 |
| Universal | Yes | No |
| Python | Yes | No |

Maven, npm, and NuGet packages can be supported from public and private sources with teams of any size. Azure Artifact comes with Azure DevOps, but the extension is also available from the [Visual Studio Marketplace Azure DevOps page](#)³.

² <https://docs.microsoft.com/en-us/azure/devops/artifacts/overview?view=vsts>

³ <https://marketplace.visualstudio.com/items?itemName=ms.feed>

The screenshot shows the Azure DevOps interface for the 'Parts-Unlimited' project. The left sidebar lists several navigation items: Overview, Boards, Repos, Pipelines, Test Plans, and Artifacts. The 'Artifacts' item is highlighted with a red box. To the right, a 'Connect to feed' section is displayed, featuring a list of supported package managers: NuGet, npm, Maven, Gradle, Universal, and Python. The 'NuGet' item is at the top of the list.

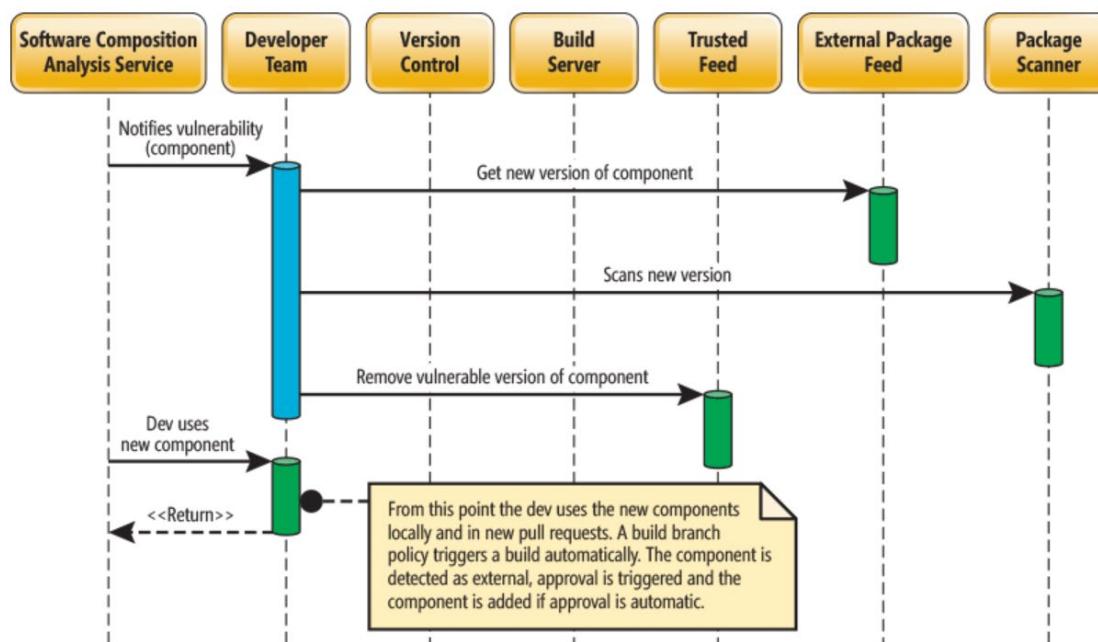
Note: After you publish a particular version of a package to a feed, that version number is permanently reserved. You cannot upload a newer revision package with that same version number, or delete that version and upload a new package with the same version number. The published version is immutable.

The Role of OSS components

Development work is more productive as a result of the wide availability of reusable Open-source software (OSS) components. This practical approach to reuse includes runtimes, which are available on Windows and Linux operating systems such as Microsoft .NET Core and Node.js.

However, OSS component reuse comes with the risk that reused dependencies can have security vulnerabilities. As a result, many users find security vulnerabilities in their applications due to the Node.js package versions they consume.

To address these security concerns, OSS offers a new concept called *Software Composition Analysis (SCA)*, which is depicted in the following image.



When consuming an OSS component, whether you are creating or consuming dependencies, you'll typically want to follow these high-level steps:

1. Start with the latest, correct version to avoid any old vulnerabilities or license misuses.
2. Validate that the OSS components are the correct binaries for your version. In the release pipeline, validate binaries to ensure that they are correct and to keep a traceable bill of materials.
3. Get notifications of component vulnerabilities immediately, and correct and redeploy the component automatically to resolve security vulnerabilities or license misuses from reused software.

Whitesource integration with Azure DevOps pipeline

Visual Studio Code marketplace⁴ is an important site for addressing Rugged DevOps issues. From here you can integrate specialist security products into your Azure DevOps pipeline. Having a full suite of extensions that allow seamless integration into Azure DevOps pipelines is invaluable.

WhiteSource

The **WhiteSource⁵** extension is available on the Azure DevOps Marketplace. Using WhiteSource, you can integrate extensions with your CI/CD pipeline to address Rugged DevOps security-related issues. For a team consuming external packages, the WhiteSource extension specifically addresses open source security, quality, and license compliance concerns. Because most breaches today target known vulnerabilities in common components, robust tools are essential to securing problematic open source components.

⁴ <https://marketplace.visualstudio.com/>

⁵ <https://marketplace.visualstudio.com/items?itemName=whitesource.whitesource>

Continuously detect all open-source components in your software

WhiteSource will automatically detect all open-source components—including their transitive dependencies—every time you run a build. This means you can generate a comprehensive inventory report within minutes based on the last build you ran. It also gives full visibility to your security, DevOps, and legal teams into your organization's software development process.

| Inventory Report - NewProj | | | | | Export |
|---|----------------|---|-----------------------|--------------------------|---------------------------------|
| Library Name | Type | Description | Licenses | Occurrences | |
| Accord.MachineLearning.GPL-2.13.1.dll | .NET | | GPL 3.0 | 1 - Test Project - 1.0.0 | |
| AjaxControlToolkit-4.1.60919.dll | .NET | | Requires Review | 1 - Test | WhiteSource Component Inventory |
| AK.Aspects-0.1.0.dll | .NET | | GPL 3.0 | 1 - Test Project - 1.0.0 | |
| android-external-curl-master_2014-03-28 | Source Library | Curl with jni interface library for android | MIT | 1 - curl | |
| aopalliance-1.0.jar | Java | AOP Alliance | Public Domain | 1 - Test Project - 1.0.0 | |
| Apache.NMS.ActiveMQ-1.5.6.dll | .NET | | Apache 2.0 | 1 - Test Project - 1.0.0 | |
| bash-bash-4.3-rc1 | Source Library | Mirror of git://git.savannah.gnu.org/bash.git | GPL 3.0 | 1 - bash-test | |
| BouncyCastle.Crypto-1.7.0.dll | .NET | | Bouncy Castle License | 1 - Test | |
| curl-7.40.0-master_2015-02-22 | Source Library | | MIT | 1 - curl | |
| curl-curl-7_39_0 | Source Library | Curl is a tool and libcurl is a library for transferring data with URL syntax, supporting FTP, FTPS, HTTP, HTTPS, Gopher, TFTP, SCP, SFTP, Telnet, DICT, LDAP, LDAPS, FILE, IMAP, SMTP, POP3, RTSP and RTMP. libcurl offers a myriad of powerful features | MIT | 1 - curl | |
| curl-master_2014-12-20 | Source Library | | MIT | 1 - curl | |
| exp-edit-trunk_2011-03-09 | Source Library | Lightweight graphics editor. Usefull for artists. | Requires Review | 1 - Test Project - 1.0.0 | |
| Hummer-master_2012-07-02 | Source Library | | Requires Review | 1 - Test Project - 1.0.0 | |
| is-my-json-valid-v2.7.6 | Source Library | | Artistic 2.0 | 1 - Test Project - 1.0.0 | |
| libBlenderWindows-master_2011-03-25 | Source Library | | Unspecified License | 1 - Test Project - 1.0.0 | |
| new_election-master_2014-01-07 | Source Library | | Requires Review | 1 - Test Project - 1.0.0 | |

Receive alerts on open-source security vulnerabilities

When a new security vulnerability is discovered, WhiteSource automatically generates an alert and provides targeted remediation guidance. This can include links to patches, fixes, relevant source files, even recommendations to change system configuration to prevent exploitation.

| Vulnerabilities | | | | | | | Export |
|-----------------|-------------------------|-------------------|------------------|-------|---|------------|---------------|
| Severity | Library | Occurrences | Vulnerability Id | Score | Description (hover for full text) | Published | Newer Version |
| High | validator.js | 1 project details | CVE-2014-8882 | 7.1 | The validator module before version 3.22.1 is vulnerable to Regular Expression Denial of Service (ReDoS) in the isURL method. | 13-11-2014 | - |
| Medium | AjaxControlToolkit.dll | 1 project details | CVE-2015-4670 | 6.4 | Directory traversal vulnerability in the AjaxFileUpload control in DevExpress AJAX Control Toolkit (aka AjaxControlToolkit) | 18-08-2015 | - |
| Medium | is-my-json-valid-v1.3.4 | 1 project details | CVE-2016-2537 | 5.0 | The is-my-json-valid package before 2.12.4 for Node.js has an incorrect exports['utc-millisecond'] regular expression, which allows | 22-02-2016 | - |
| Medium | BouncyCastle.Crypto.dll | 1 project details | CVE-2013-1624 | 4.0 | The TLS implementation in the Bouncy Castle Java library before 1.48 and C# library before 1.8 does not properly | 07-02-2013 | - |

(hover icon for details) ✓ - No Vulnerabilities ⚠ - At least one Low Severity Vulnerability ⚡ - At least one High or Medium Severity Vulnerability

Automatically enforce open-source security and license compliance policies

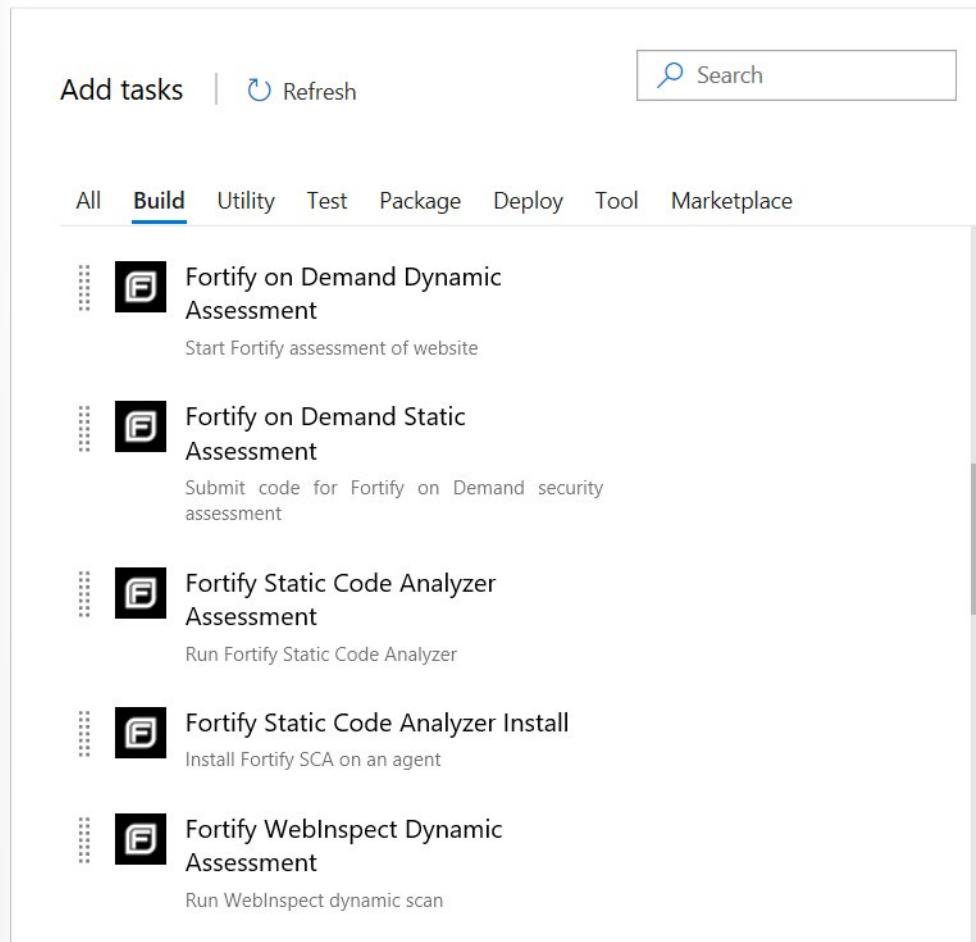
According to a company's policies, WhiteSource automatically approves, rejects, or triggers a manual approval process every time a new open-source component is added to a build. Developers can set up policies based on parameters such as security-vulnerability severity, license type, or library age. When a developer attempts to add a problematic open source component, the service will send an alert and fail the build.

For searching online repositories such as GitHub and Maven Central, WhiteSource also offers an innovative browser extension. Even before choosing a new component, a developer can review its security vulnerabilities, quality, and license issues, and whether it fits their company's policies.

Micro Focus Fortify integration with Azure Dev-Ops pipeline

Micro Focus Fortify⁶ is another example of an extension that you can leverage from the Azure DevOps Marketplace for integration with your CI/CD pipeline. Micro Focus Fortify allows you to address Rugged DevOps security-related concerns by adding build tasks for continuous integration to help you to identify vulnerabilities in your source code.

Micro Focus Fortify provides a comprehensive set of software security analyzers that search for violations of security-specific coding rules and guidelines. Development groups and security professionals use it to analyze application source code for security issues.



Fortify Static Code Analyzer

The Micro Focus Fortify **Static Code Analyzer** (Fortify SCA) identifies root causes of software security vulnerabilities. It then delivers accurate, risk-ranked results with line-of-code remediation guidance.

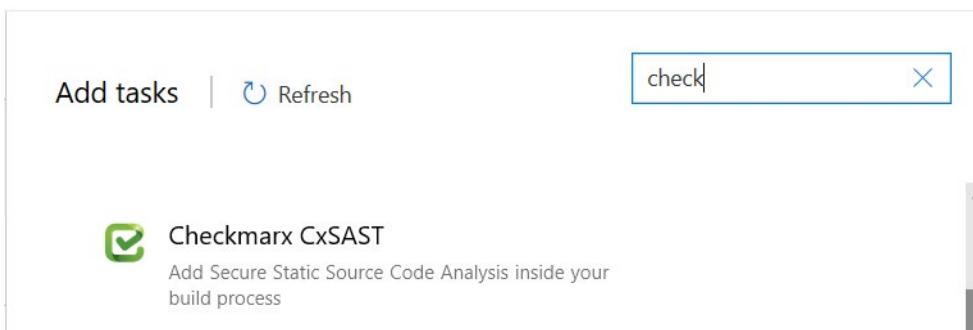
⁶ <https://marketplace.visualstudio.com/items?itemName=fortifyvsts.hpe-security-fortify-vsts>

Fortify on Demand

Fortify on Demand delivers application SaaS. It automatically submits static and dynamic scan requests to the application's SaaS platform. Static assessments are uploaded to Fortify on Demand. For dynamic assessments, you can pre-configure a specific application URL.

CheckMarx integration with Azure DevOps

Checkmarx CxSAST⁷ is another example of an extension from the Azure DevOps Marketplace that you can apply to your CI/CD pipeline to address Rugged DevOps security-related issues. Checkmarx CxSAST is designed to identify, track, and fix technical and logical security flaws. Checkmarx is a powerful, unified security solution for Static Application Security Testing (SAST) and Checkmarx Open Source Analysis (CxOSA). You can download Checkmarx from the Azure DevOps Marketplace.



Checkmarx functionality

Checkmarx functionality includes:

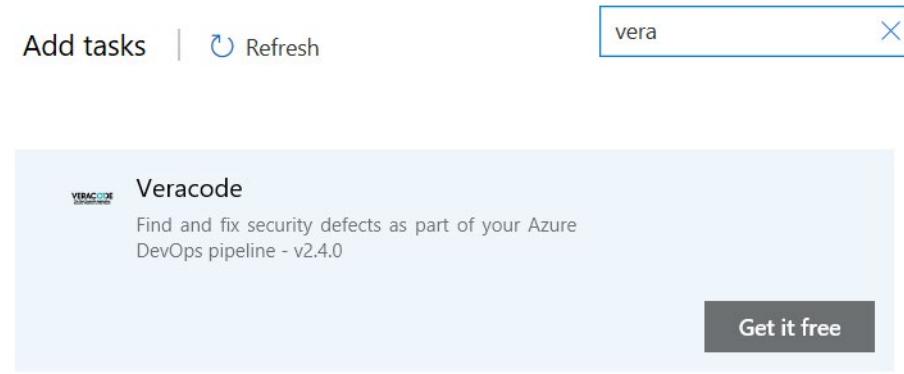
- Best fix location. Checkmarx highlights the best place to fix your code to minimize the time required to remediate the issue. A visual chart of the data flow graph indicates the ideal location in the code to address multiple vulnerabilities within the data flow using a single line of code.
- Quick and accurate scanning. Checkmarx helps reduce false positives, adapt the rule set to minimize false positives, and understand the root cause for results.
- Incremental scanning. Using Checkmarx, you can test just the code parts that have changed since last code check in. This helps reduce scanning time by more than 80 percent. It also enables you to incorporate the security gate within your continuous integration pipeline.
- Seamless integration. Checkmarx works with all integrated development environments (IDEs), build management servers, bug tracking tools, and source repositories.
- Code portfolio protection. Checkmarx helps protect your entire code portfolio, both open source and in-house source code. It analyzes open-source libraries, ensuring licenses are being adhered to, and removing any open-source components that expose the application to known vulnerabilities. In addition, Checkmarx Open Source helps provide complete code portfolio coverage under a single unified solution with no extra installations or administration required.
- Easy to initiate Open Source Analysis. With Checkmarx's Open Source analysis, you don't need additional installations or multiple management interfaces; you simply turn it on, and within minutes a detailed report is generated with clear results and detailed mitigation instructions. Because analysis

⁷ <https://marketplace.visualstudio.com/items?itemName=checkmarx.cxsast>

results are designed with the developer in mind, no time is wasted trying to understand the required action items to mitigate detected security or compliance risks.

Veracode integration with Azure DevOps

Veracode⁸ is another example of an Azure DevOps Marketplace extension that you can integrate with your CI/CD pipeline to address Rugged DevOps security-related issues. The *Veracode Application Security Platform* is a SaaS that enables developers to automatically scan an application for security vulnerabilities. The SAST, dynamic application security testing (DAST), and software composition analysis (SCA) capabilities provided by Veracode allow development teams to assess both first-party code and third-party components for security risks.



Veracode functionality

Veracode's functionality includes the following features:

- Integrate application security into the development tools you already use. From within Azure DevOps and Microsoft Team Foundation Server (TFS) you can automatically scan code using the Veracode Application Security Platform to find security vulnerabilities. With Veracode you can import any security findings that violate your security policy as work items. Veracode also gives you the option to stop a build if serious security issues are discovered.
- No stopping for false alarms. Because Veracode gives you accurate results and prioritizes them based on severity, you don't need to waste resources responding to hundreds of false positives. Microsoft has assessed over 2 trillion lines of code in 15 languages and over 70 frameworks. In addition, this process continues to improve with every assessment because of rapid update cycles and continuous improvement processes. If something does get through, you can mitigate it using the easy Veracode workflow.
- Align your application security practices with your development practices. Do you have a large or distributed development team? Do you have too many revision control branches? You can integrate your Azure DevOps workflows with the Veracode Developer Sandbox, which supports multiple development branches, feature teams, and other parallel development practices.
- Find vulnerabilities and fix them. Veracode gives you remediation guidance with each finding and the data path that a malicious user would use to reach the application's weak point. Veracode also highlights the most common sources of vulnerabilities to help you prioritize remediation. In addition, when vulnerability reports don't provide enough clarity, you can set up one-on-one developer

⁸ <https://marketplace.visualstudio.com/items?itemName=Veracode.veracode-vsts-build-extension>

consultations with Microsoft experts who have backgrounds in both security and software development. Security issues that are found by Veracode and which could prevent you from releasing your code show up automatically in your teams' list of work items, and are automatically updated and closed after you scan your fixed code.

- Proven onboarding process allows for scanning on day one. The cloud-based Veracode Application Security Platform is designed to get you going quickly, in minutes even. Veracode's services and support team can make sure that you are on track to build application security into your process.

How to Integrate SCA checks into pipelines

Security scanning used to be thought of as an activity that was completed once per release by a dedicated security team whose members had little involvement with other teams. This practice creates a dangerous pattern in which security specialists find large batches of issues at the exact time when developers are under the most pressure to release a software product. This pressure often results in software deployment with security vulnerabilities that will need to be addressed after a product has been released.

By integrating scanning into a team's workflow at multiple points along the development path, Rugged DevOps can help to make all quality-assurance activities, including security, continuous and automated.

Pull request code scan analysis integration

DevOps teams can submit proposed changes to an application's (master) codebase using pull requests (PRs). To avoid introducing new issues, before creating a PR, developers need to verify the effects of the code changes that they make. In a DevOps process a PR is typically made for each small change. Changes are continuously merged with the master codebase to keep the master codebase up to date. Ideally, a developer should check for security issues prior creating to a PR.

Azure Marketplace extensions that facilitate integrating scans during PRs include:

- **WhiteSource⁹**. Facilitates validating dependencies with its binary fingerprinting.
- **Checkmarx¹⁰**. Provides an incremental scan of changes.
- **Veracode¹¹**. Implements the concept of a developer sandbox.
- **Black Duck by Synopsis¹²**. An auditing tool for open-source code to help identify, fix, and manage compliance.

These extensions allow a developer to experiment with changes prior to submitting them as part of a PR.

Build and release definition code scan, analysis, and integration

Developers need to optimize CI for speed so that build teams get immediate feedback about build issues. Code scanning can be performed quickly enough for it to be integrated into the CI build definition thereby preventing a broken build. This enables developers to restore a build's status to ready/ green by fixing potential issues immediately.

At the same time, CD needs to be thorough. In Azure DevOps, CD is typically managed through release definitions (which progress the build output across environments), or via additional build definitions.

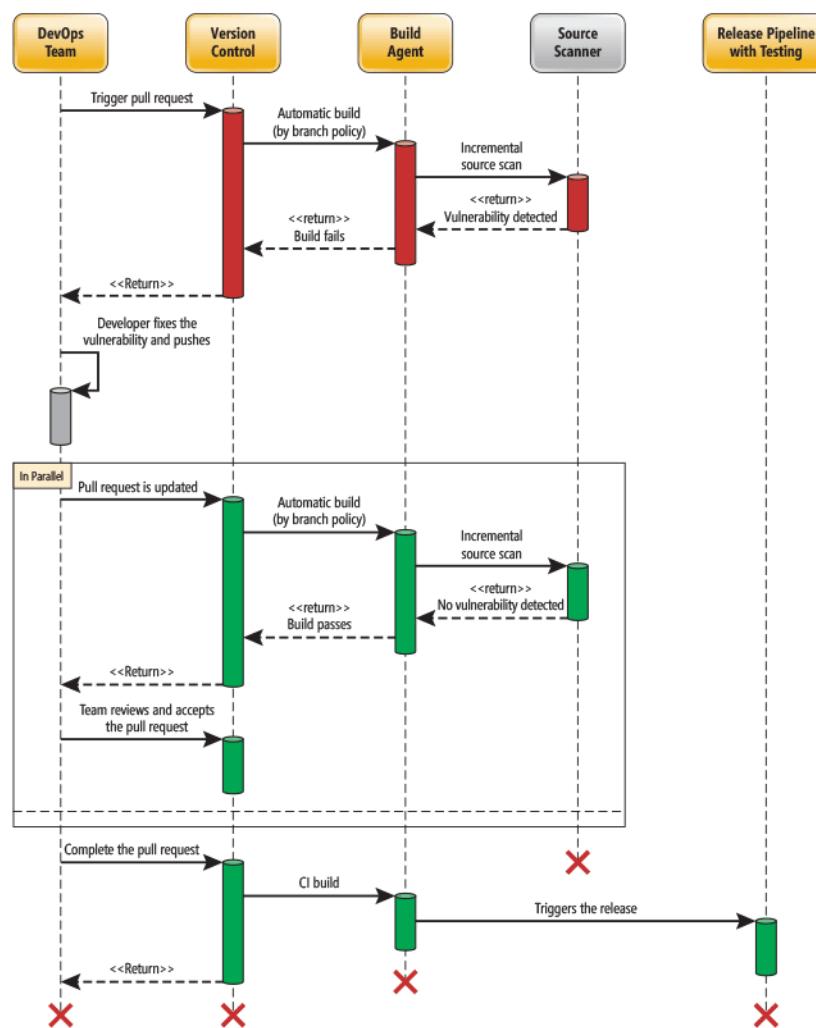
⁹ <https://www.whitesourcesoftware.com/>

¹⁰ <https://www.checkmarx.com/>

¹¹ <https://www.veracode.com/>

¹² <https://www.blackducksoftware.com/>

Build definitions can be scheduled (perhaps daily), or triggered with each commit. In either case, the build definition can perform a longer static analysis scan (as illustrated in the following image). You can scan the full code project and review any errors or warnings offline without blocking the CI flow.



Ops and pipeline security

In addition to protecting your code, it's essential to protect credentials and secrets. In particular, phishing is becoming ever more sophisticated. The following list is several operational practices that a team ought to apply to protect itself:

- Authentication and authorization. Use multifactor authentication (MFA), even across internal domains, and just-in-time administration tools such as Azure PowerShell **Just Enough Administration (JEA)**¹³, to protect against escalations of privilege. Using different passwords for different user accounts will limit the damage if a set of access credentials is stolen.
- The CI/CD Release Pipeline. If the release pipeline and cadence are damaged, use this pipeline to rebuild infrastructure. If you manage Infrastructure as Code (IaC) with Azure Resource Manager, or use the Azure platform as a service (PaaS) or a similar service, then your pipeline will automatically create new instances and then destroy them. This limits the places where attackers can hide malicious

¹³ <http://aka.ms/jea>

code inside your infrastructure. Azure DevOps will encrypt the secrets in your pipeline, as a best practice rotate the passwords just as you would with other credentials.

- Permissions management. You can manage permissions to secure the pipeline with role-based access control (RBAC), just as you would for your source code. This keeps you in control of who can edit the build and release definitions that you use for production.
- Dynamic scanning. This is the process of testing the running application with known attack patterns. You could implement penetration testing as part of your release. You also could keep up to date on security projects such as the Open Web Application Security Project ([OWASP¹⁴](#)) Foundation, then adopt these projects into your processes.
- Production monitoring. This is a key DevOps practice. The specialized services for detecting anomalies related to intrusion are known as *Security Information and Event Management*. **Azure Security Center**¹⁵ focuses on the security incidents that relate to the Azure cloud.
 - ✓ Note: In all cases, use Azure Resource Manager Templates or other code-based configurations. You should also implement IaC best practices, such as only making changes in templates, to make changes traceable and repeatable. Also, use provisioning and configuration technologies such as Desired State Configuration (DSC), Azure Automation, and other third-party tools and products that can integrate seamlessly with Azure.

Secure DevOps kit for Azure (AzSK)

The **Secure DevOps Kit for Azure**¹⁶ (AzSK) was created by the Core Services Engineering and Operations (CSEO) division at Microsoft. AzSK is a collection of scripts, tools, extensions, automations, and other resources that cater to the end-to-end security needs of DevOps teams who work with Azure subscriptions and resources. Using extensive automation, AzSK smoothly integrates security into native DevOps workflows. AzSK operates across the different stages of DevOps to maintain your control of security and governance.

AzSK can help to secure DevOps by:

- Helping secure the subscription. A secure cloud subscription provides a core foundation upon which to conduct development and deployment activities. An engineering team can deploy and configure security in the subscription by using alerts, ARM policies, RBAC, Security Center policies, JEA, and Resource Locks. Likewise, it can verify that all settings conform to a secure baseline.
- Enabling secure development. During the coding and early development stages, developers need to write secure code and then test the secure configuration of their cloud applications. Similar to build verification tests (BVTs), AzSK introduces the concept of security verification tests (SVTs), which can check the security of various resource types in Azure.
- Integrating security into CI/CD. Test automation is a core feature of DevOps. Secure DevOps provides the ability to run SVTs as part of the Azure DevOps CI/CD pipeline. You can use SVTs to ensure that the target subscription (used to deploy a cloud application) and the Azure resources that the application is built on are set up securely.
- Providing continuous assurance. In a constantly changing DevOps environment, it's important to consider security as more than a milestone. You should view your security needs as varying in accordance with the continually changing state of your systems. Secure DevOps can provide assurances that security will be maintained despite changes to the state of your systems, by using a combination of tools such as automation runbooks and schedules.

¹⁴ <https://www.owasp.org>

¹⁵ <https://azure.microsoft.com/en-us/services/security-center/>

¹⁶ <https://github.com/azsk/DevOpsKit-docs>

- Alerting & monitoring. Security status visibility is important for both individual application teams and central enterprise teams. Secure DevOps provides solutions that cater to the needs of both. Moreover, the solution spans across all stages of DevOps, in effect bridging the security gap between the Dev team and the Ops team through the single, integrated view it can generate.
- Governing cloud risks. Underlying all activities in the Secure DevOps kit is a telemetry framework that generates events such as capturing usage, adoption, and evaluation results. This enables you to make measured improvements to security by targeting areas of high risk and maximum usage.

You can leverage and utilize the tools, scripts, templates, and best practice documentation that are available as part of AzSK.

Azure security Center

Azure Security Center

Azure Security Center is a monitoring service that provides threat protection across all your services both in Azure, and on-premises. Security Center can:

- Provide security recommendations based on your configurations, resources, and networks.
- Monitor security settings across on-premises and cloud workloads, and automatically apply required security to new services as they come online.
- Continuously monitor all your services, and perform automatic security assessments to identify potential vulnerabilities before they can be exploited.
- Use Azure Machine Learning to detect and block malicious software from being installed on your virtual machines (VMs) and services. You can also define a list of allowed applications to ensure that only the apps you validate are allowed to execute.
- Analyze and identify potential inbound attacks, and help to investigate threats and any post-breach activity that might have occurred.
- Provide just-in-time (JIT) access control for ports, thereby reducing your attack surface by ensuring the network only allows traffic that you require.



Azure Security Center is part of the **Center for Internet Security (CIS) Benchmarks¹⁷** recommendations.

Azure Security Center versions

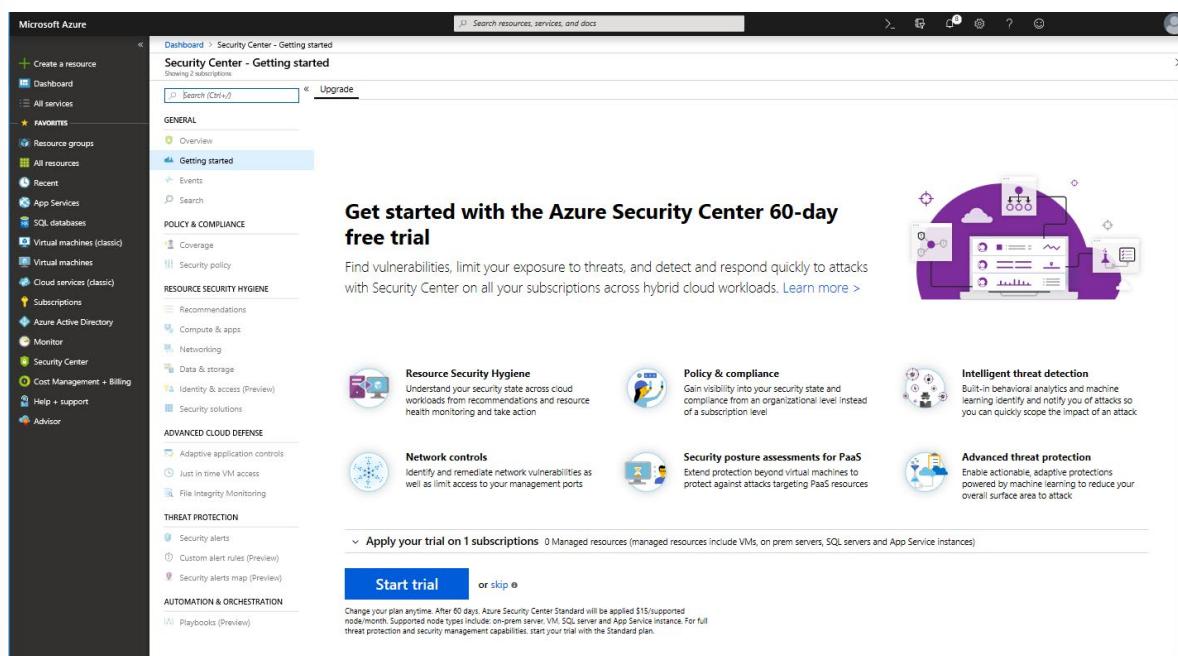
Azure Security Center supports both Windows and Linux operating systems. It can also provide security to features in both IaaS and platform as a service (PaaS) scenarios.

Azure Security Center is available in two versions:

- Free. Available as part of your Azure subscription, this tier is limited to assessments and Azure resources' recommendations only.
- Standard. This tier provides a full suite of security-related services including continuous monitoring, threat detection, JIT access control for ports, and more.

To access the full suite of Azure Security Center services you'll need to upgrade to a Standard version subscription. You can access the 60-day free trial from within the Azure Security Center dashboard in the Azure portal.

¹⁷ <https://www.cisecurity.org/cis-benchmarks/>



The screenshot shows the Microsoft Azure Security Center - Getting started page. It features a sidebar with various service links like Dashboard, All services, Resource groups, and Security Center. The main content area has a heading 'Get started with the Azure Security Center 60-day free trial'. Below this, there's a paragraph about vulnerabilities and threats, followed by six cards describing different security features: Resource Security Hygiene, Policy & compliance, Intelligent threat detection, Network controls, Security posture assessments for PaaS, and Advanced threat protection. At the bottom, there's a 'Start trial' button and a note about the trial plan.

After the 60-day trial period is over, Azure Security Center is \$15 per node per month. To upgrade a subscription from the Free trial to the Standard version, you must be assigned the role of Subscription Owner, Subscription Contributor, or Security Admin.

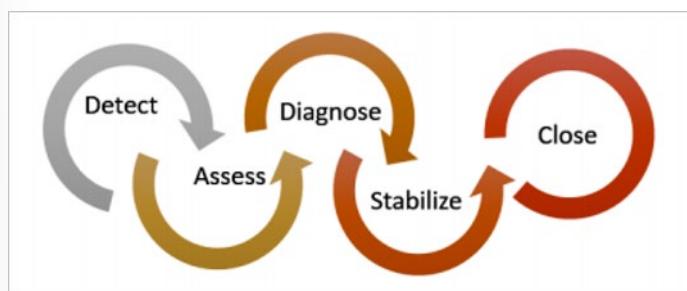
You can read more about Azure Security Center at [Azure Security Center¹⁸](#).

Azure Security Center usage scenarios

You can integrate Azure Security Center into your workflows and use it in many ways. Here are two example usage scenarios:

- Use Azure security center as part of your incident response plan.

Many organizations only respond to security incidents after an attack has occurred. To reduce costs and damage, it's important to have an incident response plan *before* an attack occurs.



The following examples are of how you can use Azure Security Center for the detect, assess, and diagnose stages of your incident response plan.

- Detect. Review the first indication of an event investigation. For example, use the Azure Security Center dashboard to review the initial verification of a high-priority security alert occurring.

¹⁸ <https://azure.microsoft.com/en-us/services/security-center/>

- **Assess.** Perform the initial assessment to obtain more information about a suspicious activity. For example, you can obtain more information from Azure Security Center about a security alert.
- **Diagnose.** Conduct a technical investigation and identify containment, mitigation, and workaround strategies. For example, you can follow the remediation steps described by Azure Security Center for a particular security alert.
- **Use Azure Security Center recommendations to enhance security.**

You can reduce the chances of a significant security event by configuring a security policy, and then implementing the recommendations provided by Azure Security Center. A *security policy* defines the set of controls that are recommended for resources within a specified subscription or resource group. In Azure Security Center, you can define policies according to your company's security requirements.

Azure Security Center analyzes the security state of your Azure resources. When it identifies potential security vulnerabilities, it creates recommendations based on the controls set in the security policy. The recommendations guide you through the process of configuring the corresponding security controls. For example, if you have workloads that don't require the Azure SQL Database Transparent Data Encryption (TDE) policy, turn off the policy at the subscription level and enable it only on the resource groups where SQL Database TDE is required.

You can read more about Azure security center at [Azure security center¹⁹](#). More implementation and scenario details are also available in the [Azure security center planning and operations guide²⁰](#).

Azure Policy

Azure Policy is an Azure service that you can use to create, assign, and, manage policies. Policies enforce different rules and effects over your Azure resources, which ensures that your resources stay compliant with your standards and SLAs.



Azure Policy uses policies and initiatives to provide policy enforcement capabilities. Azure Policy evaluates your resources by scanning for resources that do not comply with the policies you create. For example, you might have a policy that specifies a maximum size limit for VMs in your environment. After you implement your maximum VM size policy, whenever a VM is created or updated Azure Policy will evaluate the VM resource to ensure that the VM complies with the size limit that you set in your policy.

Azure Policy can help to maintain the state of your resources by evaluating your existing resources and configurations, and remediating non-compliant resources automatically. It has built-in policy and initiative definitions for you to use. The definitions are arranged in categories, such as Storage, Networking, Compute, Security Center, and Monitoring.

Azure Policy can also integrate with Azure DevOps by applying any continuous integration (CI) and continuous delivery (CD) pipeline policies that apply to the pre-deployment and post-deployment of your applications.

¹⁹ <https://azure.microsoft.com/en-us/services/security-center/>

²⁰ <https://docs.microsoft.com/en-us/azure/security-center/security-center-planning-and-operations-guide>

CI/CD pipeline integration

An example of an Azure policy that you can integrate with your DevOps CI/CD pipeline is the Check Gate task. Using Azure policies, Check gate provides security and compliance assessment on the resources with an Azure resource group or subscription that you can specify. Check gate is available as a release pipeline deployment task.

For more information go to:

- [Azure Policy Check Gate task²¹](https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-policy-check-gate?view=vsts)
- [Azure Policy²²](https://azure.microsoft.com/en-us/services/azure-policy/)

Policies

Applying a policy to your resources with Azure Policy involves the following high-level steps:

1. Policy definition. Create a policy definition.
2. Policy assignment. Assign the definition to a scope of resources.
3. Remediation. Review the policy evaluation results and address any non-compliances.

Policy definition

A **policy definition** specifies the resources to be evaluated and the actions to take on them. For example, you could prevent VMs from deploying if they are exposed to a public IP address. You could also prevent a specific hard disk from being used when deploying VMs to control costs. Policies are defined in the JavaScript Object Notation (JSON) format.

The following example defines a policy that limits where you can deploy resources:

```
{
  "properties": {
    "mode": "all",
    "parameters": {
      "allowedLocations": {
        "type": "array",
        "metadata": {
          "description": "The list of locations that can be specified when deploying resources",
          "strongType": "location",
          "displayName": "Allowed locations"
        }
      }
    },
    "displayName": "Allowed locations",
    "description": "This policy enables you to restrict the locations your organization can specify when deploying resources.",
    "policyRule": {
      "if": {
        "not": {
          "field": "location",

```

²¹ <https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-policy-check-gate?view=vsts>

²² <https://azure.microsoft.com/en-us/services/azure-policy/>

```
        "in": "[parameters('allowedLocations')]"
    }
},
"then": {
    "effect": "deny"
}
}
}
```

The following list are example policy definitions:

- Allowed Storage Account SKUs. This policy defines conditions (or *rules*) that limit storage accounts to a set of specified sizes, or Stock Keeping Units (SKUs). Its effect is to deny all storage accounts that do not adhere to the set of defined SKU sizes.
- Allowed Resource Type. This policy definition has a set of conditions to specify the resource types that can be deployed. Its effect is to deny all resources that are on the list.
- Allowed Locations. This policy restricts the locations that new resources can be deployed to. It's used to enforce geographic compliance requirements.
- Allowed Virtual Machine SKUs. This policy specifies a set of VM SKUs that can be deployed. The policy effect is that VMs cannot be deployed from unspecified SKUs.

Policy assignment

Policy definitions, whether custom or built in, need to be assigned. A *policy assignment* is a policy definition that has been assigned to a specific scope. Scopes can range from a management group to a resource group. Child resources will inherit any policy assignments that have been applied to their parents. This means that if a policy is applied to a resource group, it's also applied to all the resources within that resource group. However, you can define subscopes for excluding particular resources from policy assignments.

You can assign policies via:

- Azure portal
- Azure CLI
- PowerShell

Remediation

Resources found not to comply to a *deployIfNotExists* or *modify* policy condition can be put into a compliant state through Remediation. *Remediation* instructs Azure Policy to run the *deployIfNotExists* effect or the tag operations of the policy on existing resources. To minimize configuration drift, you can bring resources into compliance using automated bulk remediation instead of going through them one at a time.

You can read more about Azure Policy on the [Azure Policy²³](#) webpage.

²³ <https://azure.microsoft.com/en-us/services/azure-policy/>

Initiatives

Initiatives work alongside policies in Azure Policy. An *initiative definition* is a set of policy definitions to help track your compliance state for meeting large-scale compliance goals. Even if you have a single policy, we recommend using initiatives if you anticipate increasing your number of policies over time. The application of an initiative definition to a specific scope is called an *initiative assignment*.

Initiative definitions

Initiative definitions simplify the process of managing and assigning policy definitions by grouping sets of policies into a single item. For example, you can create an initiative named *Enable Monitoring in Azure Security Center* to monitor security recommendations from Azure Security Center. Under this example initiative, you would have the following policy definitions:

- Monitor unencrypted SQL Database in Security Center. This policy definition monitors unencrypted SQL databases and servers.
- Monitor OS vulnerabilities in Security Center. This policy definition monitors servers that do not satisfy a specified OS baseline configuration.
- Monitor missing Endpoint Protection in Security Center. This policy definition monitors servers without an endpoint protection agent installed.

Initiative assignments

Like a policy assignment, an *initiative assignment* is an initiative definition assigned to a specific scope. Initiative assignments reduce the need to make several initiative definitions for each scope. Scopes can range from a management group to a resource group. You can assign initiatives in the same way that you assign policies.

You can read more about policy definition and structure at [Azure Policy definition structure²⁴](#).

Azure Key Vault

Azure Key Vault is a centralized cloud service for storing your applications' secrets. Key Vault helps you control your applications' secrets by keeping them in a single central location and by providing secure access, permissions control, and access logging capabilities.



Usage scenarios

Azure Key Vault capabilities and usage scenarios include:

- Secrets management. You can use Key Vault to securely store and control access to tokens, passwords, certificates, API keys, and other secrets.

²⁴ <https://docs.microsoft.com/en-us/azure/governance/policy/concepts/definition-structure>

- Key management. As a key management solution, Key Vault makes it easy to create and control encryption keys used to encrypt your data.
- Certificate management. Key Vault lets you provision, manage, and deploy public and private Secure Sockets Layer (SSL)/ Transport Layer Security (TLS) certificates for your Azure subscription and connected resources more easily.
- Store secrets backed by Hardware Security Modules (HSMs). Secrets and keys can be protected either by software, or by Federal Information Processing Standard (FIPS) 140-2 Level 2 validated HSMs.

Key Vault benefits

The benefits of using Key Vault include:

- Centralized application secrets. Centralizing storage for application secrets allows you to control their distribution, and reduces the chances that secrets might accidentally be leaked.
- Securely stored secrets and keys. Azure uses industry-standard algorithms, key lengths, and HSMs to ensure that access requires proper authentication and authorization.
- Monitoring access and use. Using Key Vault, you can monitor and control access to company secrets.
- Simplified administration of application secrets. Key Vault makes it easier to enroll and renew certificates from public Certificate Authorities (CAs). You can also scale up and replicate content within regions, and use standard certificate management tools.
- Integration with other Azure services. You can integrate Key Vault with storage accounts, container registries, event hubs, and many more Azure services.

You can learn more about Key Vault services on the [Key Vault](#)²⁵ webpage.

Role-Based Access Control (RBAC)

RBAC provides fine-grained access management for Azure resources, which enables you to grant users only the rights they need to perform their jobs. RBAC is provided at no additional cost to all Azure subscribers.

Usage scenarios

The following examples provide use-case scenarios for RBAC:

- Allow a specific user to manage VMs in a subscription, and another user to manage virtual networks.
- Permit the database administrator (DBA) group management access to Microsoft SQL Server databases in a subscription.
- Grant a user management access to certain types of resources in a resource group, such as VMs, websites, and subnets.
- Give an application access to all resources in a resource group.

To review access permissions, in a deployed VM, open the **Access Control (IAM)** blade in the Azure portal. On this blade, you can review who has access, their role, and grant or remove access permissions.

²⁵ <https://azure.microsoft.com/en-us/services/key-vault/>

The following illustration is an example of the **Access Control (IAM)** blade for a resource group. Note how the **Roles** tab displays the built-in roles that are available.

| Role name | Role type | Count |
|---|-------------|-------|
| DevTest Labs User | BuiltInRole | 0 |
| DNS Zone Contributor | BuiltInRole | 0 |
| DocumentDB Account Contributor | BuiltInRole | 0 |
| EventGrid EventSubscription Contributor | BuiltInRole | 0 |
| EventGrid EventSubscription Reader | BuiltInRole | 0 |
| HDIInsight Domain Services Contributor | BuiltInRole | 0 |
| Intelligent Systems Account Contributor | BuiltInRole | 0 |
| Key Vault Contributor | BuiltInRole | 0 |
| Lab Creator | BuiltInRole | 0 |
| Log Analytics Contributor | BuiltInRole | 0 |
| Log Analytics Reader | BuiltInRole | 0 |
| Logic App Contributor | BuiltInRole | 0 |
| Logic App Operator | BuiltInRole | 0 |
| Managed Application Operator Role | BuiltInRole | 0 |
| Managed Applications Reader | BuiltInRole | 0 |
| Managed Identity Contributor | BuiltInRole | 0 |
| Managed Identity Operator | BuiltInRole | 0 |

For a full list of available built-in roles, go to the [Built-in roles for Azure resources²⁶](#) webpage.

RBAC uses an allow model for permissions. This means that when you are assigned a role, RBAC *allows* you to perform certain actions such as read, write, or delete. Therefore, if one role assignment grants you read permissions to a resource group, and a different role assignment grants you write permissions to the same resource group, you will have both read and write permissions for that resource group.

Best practice

When using RBAC, segregate duties within your team and only grant users the access they need to perform their jobs. Instead of giving everybody unrestricted access to your Azure subscription and resources, allow only certain actions at a particular scope. In other words, grant users the minimal privileges that they need to complete their work.

Note: For more information about RBAC, visit [What is role-based access control \(RBAC\) for Azure resources?²⁷](#).

Locks

Locks help you prevent accidental deletion or modification of your Azure resources. You can manage locks from within the Azure portal. In Azure portal, locks are called **Delete** and **Read-only** respectively. To review, add, or delete locks for a resource in Azure portal, go to the **Settings** section on the resource's settings blade.

You might need to lock a subscription, resource group, or resource to prevent users from accidentally deleting or modifying critical resources. You can set a lock level to **CanNotDelete** or **ReadOnly**:

- **CanNotDelete** means that authorized users can read and modify a resource, but they cannot delete the resource.
- **ReadOnly** means that authorized users can read a resource, but they cannot modify or delete it.

You can read more about Locks on the [Lock resources to prevent unexpected changes²⁸](#) webpage.

Subscription governance

The following considerations apply to creating and managing Azure subscriptions:

- Billing. You can generate billing reports for Azure subscriptions. If, for example, you have multiple internal departments and need to perform a chargeback, you can create a subscription for a department or project.
- Access control. A subscription acts as a deployment boundary for Azure resources. Every subscription is associated with an Azure Active Directory (Azure AD) tenant that provides administrators with the ability to set up RBAC. When designing a subscription model, consider the deployment boundary. Some customers keep separate subscriptions for development and production, and manage them using RBAC to isolate one subscription from the other (from a resource perspective).
- Subscription limits. Subscriptions are bound by hard limitations. For example, the maximum number of Azure ExpressRoute circuits per subscription is 10. If you reach a limit, there is no flexibility. Keep these limits in mind during your design phase. If you need to exceed the limits, you might require additional subscriptions.

²⁶ <https://docs.microsoft.com/en-us/azure/role-based-access-control/built-in-roles>

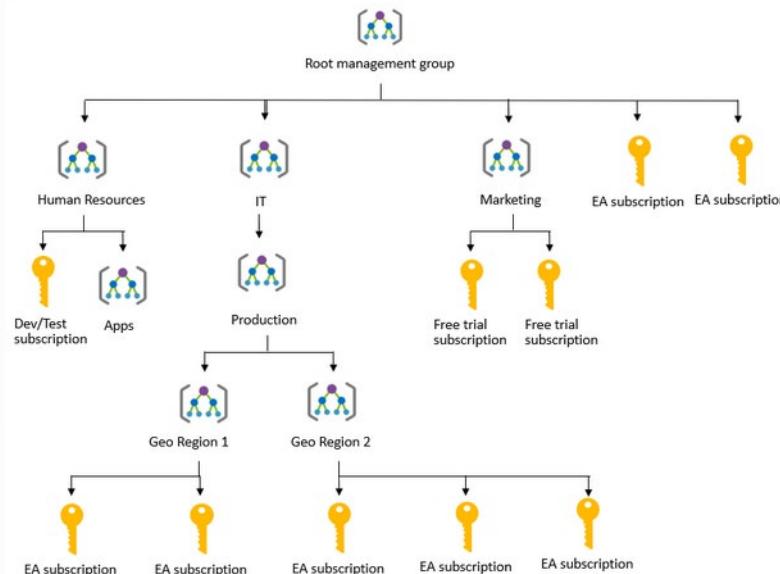
²⁷ <https://docs.microsoft.com/en-us/azure/role-based-access-control/overview>

²⁸ <https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-lock-resources>

Management groups

Management Groups can assist you with managing your Azure subscriptions. *Management groups* manage access, policies, and compliance across multiple Azure subscriptions. They allow you to order your Azure resources hierarchically into collections. Management groups facilitate the management of resources at a level above the level of subscriptions.

In the following image, access is divided across different regions and business functions, such as marketing and IT. This helps you track costs and resource usage at a granular level, add security layers, and segment workloads. You could even divide these areas further into separate subscriptions for Dev and QA, or for specific teams.



You can manage your Azure subscriptions more effectively by using Azure Policy and Azure RBACs. These tools provide distinct governance conditions that you can apply to each management group. Any conditions that you apply to a management group will automatically be inherited by the resources and subscriptions within that group.

Note: For more information about management groups and Azure, go to the [Azure management groups documentation, Organize your resources²⁹](#) webpage.

Azure Blueprints

Azure Blueprints enables cloud architects to define a repeatable set of Azure resources that implement and adhere to an organization's standards, patterns, and requirements. Azure Blueprints helps development teams build and deploy new environments rapidly with a set of built-in components that speed up development and delivery. Furthermore, this is done while staying within organizational compliance requirements.

²⁹ <https://docs.microsoft.com/en-us/azure/governance/management-groups/>



Azure Blueprints provides a declarative way to orchestrate deployment for various resource templates and artifacts, including:

- Role assignments
- Policy assignments
- Azure Resource Manager templates
- Resource groups

To implement Azure Blueprints, complete the following high-level steps:

1. Create a blueprint.
2. Assign the blueprint.
3. Track the blueprint assignments.

With Azure Blueprints, the relationship between the blueprint definition (what *should be* deployed) and the blueprint assignment (what *is* deployed) is preserved.

The blueprints in Azure Blueprints are different from Azure Resource Manager templates. When Azure Resource Manager templates deploy resources, they have no active relationship with the deployed resources. (They exist in a local environment or in source control). By contrast, with Azure Blueprints, each deployment is tied to an Azure Blueprints package. This means that the relationship with resources will be maintained, even after deployment. This way, maintaining relationships improves deployment tracking and auditing capabilities.

Usage scenario

Adhering to security and compliance requirements, whether government, industry, or organizational requirements, can be difficult and time consuming. To help you to trace your deployments and audit them for compliance, Azure Blueprints uses artifacts and tools that expedite your path to certification.

Azure Blueprints is also useful in Azure DevOps scenarios where blueprints are associated with specific build artifacts and release pipelines, and blueprints can be tracked rigorously.

You can learn more about Azure Blueprints at [Azure Blueprints³⁰](https://azure.microsoft.com/services/blueprints/).

Azure Advanced Threat Protection (ATP)

Azure Advanced Threat Protection (Azure ATP) is a cloud-based security solution that identifies and detects advanced threats, compromised identities, and malicious insider actions directed at your organization. Azure ATP is capable of detecting known malicious attacks and techniques, and can help you investigate security issues and network vulnerabilities.

³⁰ <https://azure.microsoft.com/services/blueprints/>

Azure ATP components

Azure ATP consists of the following components:

- Azure ATP portal. Azure ATP has its own portal. Through Azure ATP portal, you can monitor and respond to suspicious activity. The Azure ATP portal allows you to manage your Azure ATP instance and review data received from Azure ATP sensors.

You can also use the Azure ATP portal to monitor, manage, and investigate threats to your network environment. You can sign in to the Azure ATP portal at <https://portal.atp.azure.com>³¹. Note that you must sign in with a user account that is assigned to an Azure AD security group that has access to the Azure ATP portal.

- Azure ATP sensor. Azure ATP sensors are installed directly on your domain controllers. The sensors monitor domain controller traffic without requiring a dedicated server or port mirroring configurations.
- Azure ATP cloud service. The Azure ATP cloud service runs on the Azure infrastructure, and is currently deployed in the United States, Europe, and Asia. The Azure ATP cloud service is connected to the Microsoft Intelligent Security Graph.

³¹ <https://portal.atp.azure.com>

The screenshot shows the Azure Advanced Threat Protection Timeline page for the contoso-corp tenant. The timeline lists several events:

- 4:04 PM Today**: Honeytoken activity (Updated). The following activities were performed by Bob Minion:
 - Logged in to 2 computers via Contoso-DC.
 - Authenticated from 2 computers using Kerberos when accessing 5 resources against Contoso-DC.
 - Authenticated from ITARGOET-T4705 using NTLM against corporate resources via Contoso-DC.
 Started at 3:08 PM Jan 22, 2018
- 3:23 PM Jan 22, 2018**: Remote execution attempt detected. The following remote execution attempts were performed on Contoso-DC from ALICE-DESKTOP:
 - Attempted remote execution of one or more WMI methods by AdminUser.
- 3:06 PM Jan 22, 2018**: Suspicious service creation. AdminUser created 10 services in order to execute potentially malicious commands on Contoso-DC.
- 3:03 PM Jan 22, 2018**: Brute force attack using LDAP simple bind. 200 password guess attempts were made on 2 accounts from ALICE-DESKTOP. 2 account passwords were successfully guessed.
- 2:59 PM Jan 22, 2018**: Reconnaissance using account enumeration. Suspicious account enumeration activity using Kerberos protocol, originating from ALICE-DESKTOP, was detected. The attacker performed a total of 101 guess attempts for account names. 2 guess attempts matched existing account names in Active Directory.
- 12:38 PM Jan 21, 2018**: Malicious replication of directory services. Malicious replication requests were attempted by Alice Liddel, from ALICE-DESKTOP against Contoso-DC.
- 11:59 AM Jan 21, 2018**: Reconnaissance using DNS. Suspicious DNS activity was observed, originating from ALICE-DESKTOP (which is not a DNS server) against Contoso-DC.

Purchasing Azure ATP

Azure ATP is available as part of the Microsoft *Enterprise Mobility + Security E5* offering, and as a standalone license. You can acquire a license directly from the **Enterprise Mobility + Security pricing options³²** page, or through the Cloud Solution Provider (CSP) licensing model.

- ✓ Note: Azure ATP is not available for purchase via the Azure portal. For more information about Azure ATP, review the **Azure Advanced Threat Protection³³** webpage.

³² <https://www.microsoft.com/en-ie/cloud-platform/enterprise-mobility-security-pricing>

³³ <https://azure.microsoft.com/en-us/features/azure-advanced-threat-protection/>

Lab

Implement Security and Compliance in an Azure DevOps pipeline



Steps for the labs are available on **GitHub** at the below sites under the **Infrastructure as Code** sections

- <https://microsoft.github.io/PartsUnlimited>
- <https://microsoft.github.io/PartsUnlimitedMRP>

You should click on the link below, for the lab tasks for this module, and follow the steps outlined there for each lab task.

PartsUnlimited (PU)

- [Implement Security and Compliance in Azure DevOps pipelines](#)³⁴

³⁴ <http://microsoft.github.io/PartsUnlimited/iac/200.2x-iaC-SecurityandComplianceinpipeline.html>

Module Review and Takeaways

Module Review Questions

Checkbox

Rugged DevOps combines which two elements? Choose two.

- DevOps
- Cost management
- Microservice Architecture
- Security
- Hackathons

Multiple choice

Which term broadly defines what security means in Rugged DevOps?

- Access control
- Application server hardening
- perimeter protection
- Securing the pipeline

Dropdown

What component in Azure DevOps can you use to store, organize and share access to packages, and integrate those packages them with your continuous integration and continuous delivery pipeline?

- Test Plans
- Azure Artifacts
- Boards
- Pipelines

Checkbox

Which of the following package types are available to use with Azure Artifacts? Choose three.

- NuGet
- npm
- PowerShell
- Maven

Multiple choice

Which description from the list below best describes the term Software Composition Analysis?

- Assessment of production hosting infrastructure just before deployment
- Analyze build software to identify load capacity
- Analyzing open source software (OSS) to identify potential security vulnerabilities and provide validation that the software meets a defined criteria to use in your pipeline
- Analyzing open source software after it has been deployed to production to identify security vulnerabilities

Multiple choice

From where can extensions be sourced from, to be integrated into Azure DevOps CI/CD pipelines and help provide security composition analysis?

- Azure DevOps Marketplace
- www.microsoft.com
- Azure Security Center
- TFVC git repos

Checkbox

Which products, from the below list, are available as extensions in Azure DevOps Marketplace, and can provide either OSS or source code scanning as part of an Azure DevOps pipeline? Choose all that apply.

- Whitesource
- CheckMarx
- Micro Focus Fortify
- Veracode

Multiple choice

Which Azure service from the below list is a monitoring service that can provide threat protection and security recommendations across all of your services both in Azure and on-premises? <<

- Azure Policy
- Azure Security Center
- Azure Key vault
- Role-based access control

Multiple choice

Which Azure service should you use from the below list to monitor all unencrypted SQL databases in your organization?

- Azure Policy
- Azure Security Center
- Azure Key Vault
- Azure Machine Learning

Multiple choice

Which facility from the below list, allows you to prevent accidental deletion of resources in Azure?

- Key Vault
- Azure virtual machines
- Azure Blueprints
- Locks

Answers

Checkbox

Rugged DevOps combines which two elements? Choose two.

- DevOps
- Cost management
- Microservice Architecture
- Security
- Hackathons

Explanation

DevOps and Security are the correct answers. All other answers are incorrect. Rugged DevOps brings together the notions of DevOps and Security. DevOps is about working faster. Security is about emphasizing thoroughness, which is typically done at the end of the cycle, resulting in potentially generating unplanned work right at the end of the pipeline. Rugged DevOps is a set of practices designed to integrate DevOps and security, and to meet the goals of both more effectively.

Multiple choice

Which term broadly defines what security means in Rugged DevOps?

- Access control
- Application server hardening
- Perimeter protection
- Securing the pipeline

Explanation

Securing the pipeline is the correct answer.

All other answers, while covering some elements of security, and while being important in their own right, do not cover what is meant by security in Rugged DevOps.

With Rugged DevOps, security is more about securing the pipeline, determining where you can add security to the elements that plug into your build and release pipeline. For example, it's about how and where you can add security to your automation practices, production environments, and other pipeline elements while attempting to gain the speed of DevOps.

Rugged DevOps includes bigger questions such as:

Is my pipeline consuming third-party components, and if so, are they secure?

Are there known vulnerabilities within any of the third-party software we use?

How quickly can I detect vulnerabilities (time to detect)?

How quickly can I remediate identified vulnerabilities (time to remediate)?

Dropdown

What component in Azure DevOps can you use to store, organize and share access to packages, and integrate those packages them with your continuous integration and continuous delivery pipeline?

- Test Plans
- Azure Artifacts
- Boards
- Pipelines

Explanation

Azure Artifacts is the correct answer. Azure Artifacts is an integral part of the component workflow, which you can use to organize and share access to your packages. It allows you to:

Keep your artifacts organized. Share code easily by storing Apache Maven, npm, and NuGet packages together. You can store packages using Universal Packages, eliminating the need to store binaries in Git. Protect your packages. Keep every public source package you use, including packages from npmjs and nuget.org, safe in your feed where only you can delete it, and where it's backed by the enterprise-grade Azure SLA.

Integrate seamless package handling into your CI/CD pipeline. Easily access all your artifacts in builds and releases. Artifacts integrate natively with the Azure Pipelines CI/CD tool.

Checkbox

Which of the following package types are available to use with Azure Artifacts? Choose three.

- NuGet
- npm
- PowerShell
- Maven

Explanation

NuGet, npm and Maven are the correct answers. Powershell is not a package type and is incorrect. Azure Artifacts allows the sharing of code easily by storing Apache Maven, npm, and NuGet packages together. You can also store packages using Universal Packages, eliminating the need to store binaries in Git.

Multiple choice

Which description from the list below best describes the term Software Composition Analysis?

- Assessment of production hosting infrastructure just before deployment
- Analyze build software to identify load capacity
- Analyzing open source software (OSS) to identify potential security vulnerabilities and provide validation that the software meets a defined criteria to use in your pipeline
- Analyzing open source software after it has been deployed to production to identify security vulnerabilities

Explanation

Analyzing open source software (OSS) to identify potential security vulnerabilities and provide validation that the software meets a defined criteria to use in your pipeline is the correct answer.

When consuming an OSS component, whether you're creating or consuming dependencies, you'll typically want to follow these high-level steps:

Multiple choice

From where can extensions be sourced from, to be integrated into Azure DevOps CI/CD pipelines and help provide security composition analysis?

- Azure DevOps Marketplace
- www.microsoft.com
- Azure Security Center
- TFVC git repos

Explanation

Azure DevOps Marketplace is the correct answer. All other answers are incorrect.

Azure DevOps Marketplace is an important site for addressing Rugged DevOps issues. From here you can integrate specialist security products into your Azure DevOps pipeline. Having a full suite of extensions that allow seamless integration into Azure DevOps pipelines is invaluable

Checkbox

Which products, from the below list, are available as extensions in Azure DevOps Marketplace, and can provide either OSS or source code scanning as part of an Azure DevOps pipeline? Choose all that apply.

- Whitesource
- CheckMarx
- Micro Focus Fortify
- Veracode

Explanation

All answers are correct.

All of the listed products are available as extensions in Azure DevOps Marketplace, and can provide either OSS or static source code scanning as part of the Azure devOps pipeline

Multiple choice

Which Azure service from the below list is a monitoring service that can provide threat protection and security recommendations across all of your services both in Azure and on-premises? <<

- Azure Policy
- Azure Security Center
- Azure Key vault
- Role-based access control

Explanation

Azure Security Center is the correct answer. All other answers are incorrect.

Azure Security Center is a monitoring service that provides threat protection across all of your services both in Azure, and on-premises. Security Center can:

None of the other services provide a monitoring service that can provide threat protection and security recommendations across all of your services both in Azure and on-premises

Multiple choice

Which Azure service should you use from the below list to monitor all unencrypted SQL databases in your organization?

- Azure Policy
- Azure Security Center
- Azure Key Vault
- Azure Machine Learning

Explanation

Azure Policy is the correct answer. All other answers are incorrect.

Azure Policy is a service in Azure that you use to create, assign, and, manage policies. These policies enforce different rules and effects over your resources, which ensures they stay compliant with your corporate standards and service-level agreements (SLAs). A policy definition expresses what to evaluate and what action to take. For example, you could prevent VMs from deploying if they are exposed to a public IP address. You also could prevent a particular hard disk from being used when deploying VMs to control costs.

Initiative definitions simplify the process of managing and assigning policy definitions by grouping a set of policies as one single item. For example, you could create an initiative named Enable Monitoring in Azure Security Center, with a goal to monitor all the available security recommendations in your Azure Security Center. Under this initiative, you would have the following policy definitions:

Multiple choice

Which facility from the below list, allows you to prevent accidental deletion of resources in Azure?

- Key Vault
- Azure virtual machines
- Azure Blueprints
- Locks

Explanation

Locks is the correct answer. All other answers are incorrect. Locks help you prevent accidental deletion or modification of your Azure resources. You can manage these locks from within the Azure portal. To view, add, or delete locks, go to the SETTINGS section of any resource's settings blade. You may need to lock a subscription, resource group, or resource to prevent other users in your organization from accidentally deleting or modifying critical resources. You can set the lock level to CanNotDelete or ReadOnly.

Module 19 Recommend and Design System Feedback Mechanisms

Module Overview

Module Overview

The ultimate goal of DevOps is to increase the speed and quality of software services, something every IT organization needs. But how can the parties involved in DevOps accelerate delivery if they don't understand what they deliver or know the people to whom they deliver? Despite great advancements in delivery, quality service still begins with understanding the needs of users.

Unsatisfied customers are probably costing you a lot of money. The first step to overcoming this is to admit that you have room for improvement. The second step is to measure customer satisfaction to find out where you currently stand.

Engaging customers throughout your product lifecycle is a primary Agile principle. Empower each team to interact directly with customers on the feature sets they own.

- Continuous feedback: Build in customer feedback loops. These can take many forms:
 - Customer voice: Make it easy for customers to give feedback, add ideas, and vote on next generation features.
 - Product feedback: In-product feedback buttons are another way to solicit feedback about the product experience or specific features.
 - Customer demos: Regularly scheduled demos that solicit feedback from your customers can help shape next generation products and keep you on track to build applications your customers want to consume.
- Early adopter programs: Such programs should be developed with the idea that all teams may want to participate at some point. Early adopters gain access to early versions of working software which they then can provide feedback. Oftentimes, these programs work by turning select feature flags on for an early adopter list.

- Data-driven decisions: Find ways to instrument your product to obtain useful data and that can test various hypotheses. Help drive to an experiment-friendly culture that celebrates learning.

Learning Objectives

After completing this module, students will be able to:

- Design practices to measure end-user satisfaction
- Design processes to capture and analyze user feedback from external sources
- Design routing for client application crash report data
- Recommend monitoring tools and technologies
- Recommend system and feature usage tracking tools

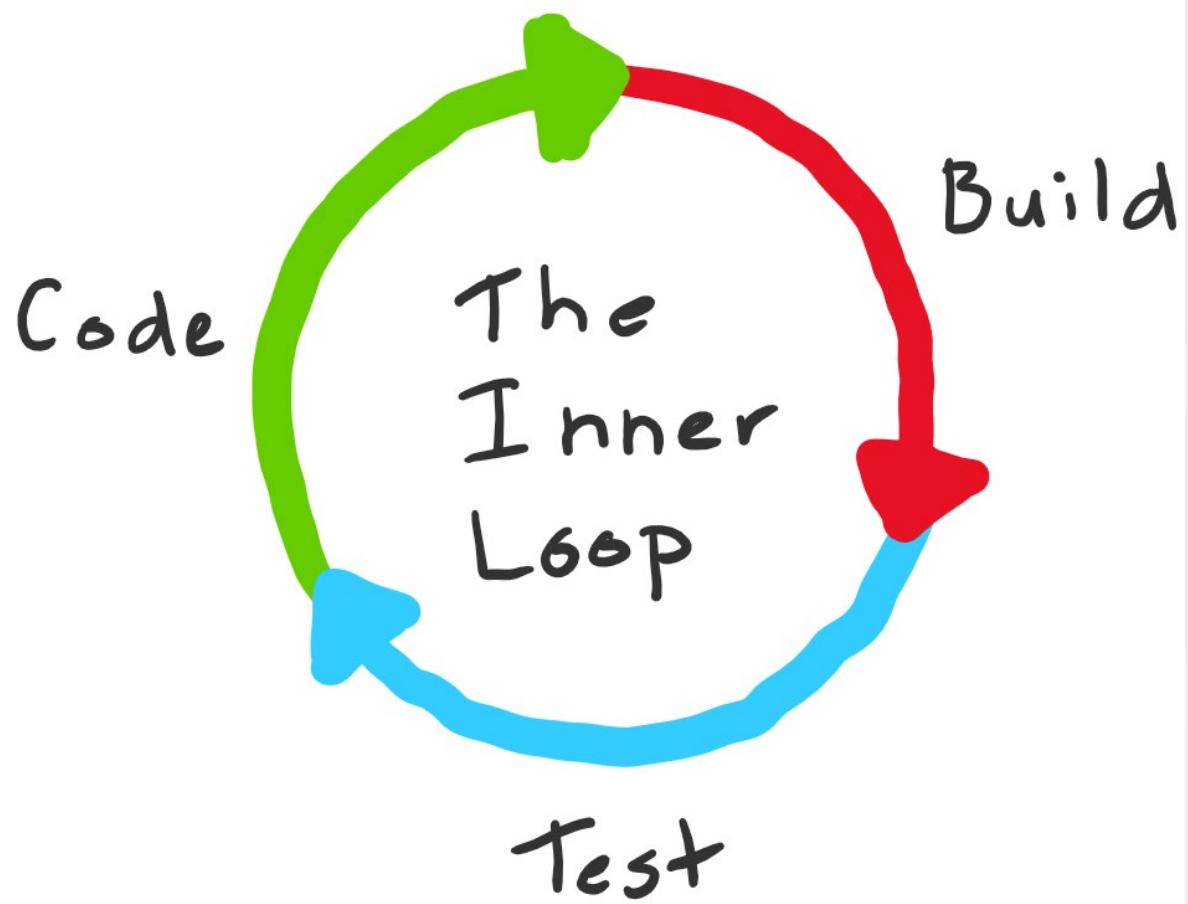
The Inner Loop

The inner loop

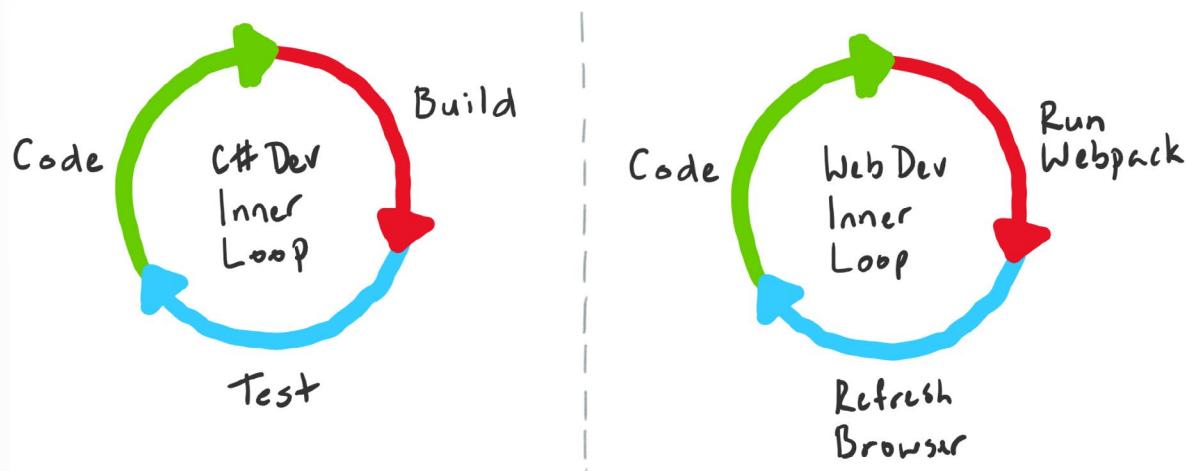
It's not clear who coined the term "inner loop" in the context of software engineering, but within Microsoft at least the term seems to have stuck. Many of the internal teams that I work with see it as something that they want to keep as short as possible - but what is the inner loop?

Definitions

The easiest way to define the inner loop is the iterative process that a developer performs when they write, build, and debug code. There are other things that a developer does, but this is the tight set of steps that are performed over and over before they share their work with their team or the rest of the world.



Exactly what goes into an individual developer's inner loop will depend a great deal on the technologies that they are working with, the tools being used and of course their own preferences. If I was working on a library my inner loop would include coding, build, test execution & debugging with regular commits to my local Git repository. On the other hand if I was doing some web front-end work I would probably be optimized around hacking on HTML & JavaScript, bundling and refreshing the browser (followed by regular commits).



In reality most codebases are comprised of multiple moving parts and so the definition of a developer's inner loop on any single codebase might alternate depending on what is being worked on.

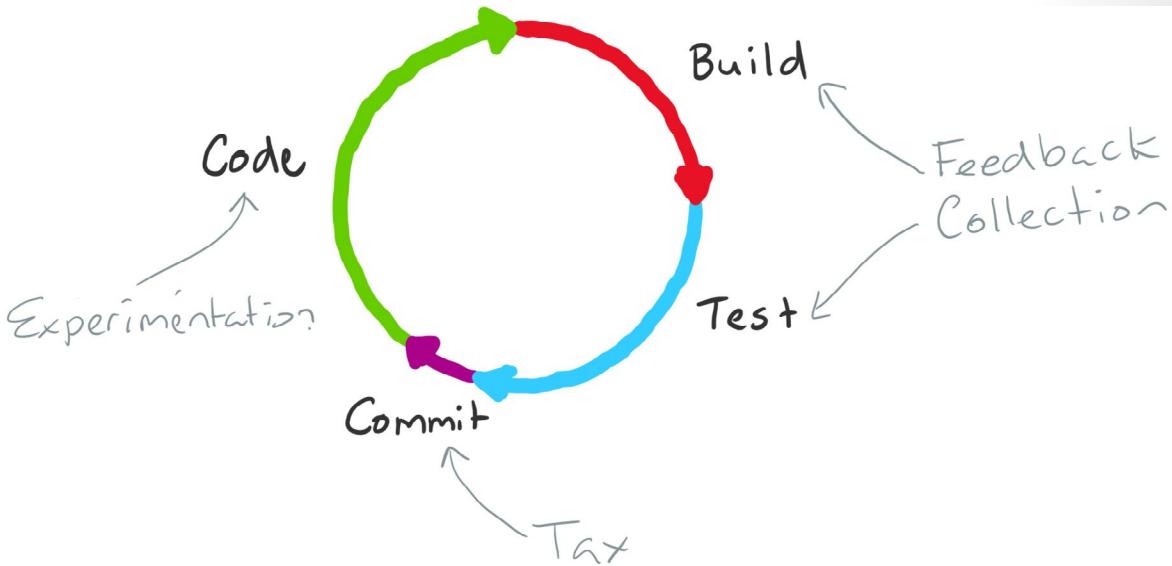
Understanding the Loop

The steps within the inner loop can be grouped into three broad buckets of activity - experimentation, feedback collection and tax. If we flick back to my library development scenario I mentioned earlier, there are four steps I mentioned and how I would bucket them.

- Coding (Experimentation)
- Building (Feedback Collection)
- Testing / Debugging (Feedback Collection)
- Committing (Tax)

Of all the steps in the inner loop, coding is the only one that adds customer value. Building and testing code are important, but ultimately we use them to give the developer feedback about the what they have written to see if it delivers sufficient value (does the code even compile, and does it satisfy the requirements that we've agreed etc).

Putting committing code in the tax bucket is perhaps a bit harsh, but the purpose of the bucket is to call out those activities that neither add value, or provide feedback. Tax is necessary work, if its unnecessary work then it is waste and should be eliminated.



Loop Optimization

Having categorized the steps within the loop it is now possible to make some general statements:

- You want to execute the loop as fast as possible and for the total loop execution time to be proportional to the changes being made.
- You want to minimize the time feedback collection takes, but maximize the quality of the feedback that you get.
- You want to minimize the tax you pay by eliminating it where it isn't necessary on any particular run through the loop (can you defer some operations until you commit for example).
- As new code and more complexity is added to any codebase the amount of outward pressure to increase the size of the inner loop also increases (more code means more tests which in turn means more execution time and a slow execution of the inner loop).

If you have ever worked on a large monolithic codebase it is possible to get into a situation where even small changes require a disproportionate amount of time to execute the feedback collection steps of the inner loop. This is a problem and you should fix it.

There are a number of things that a team can do to optimize the inner loop for larger codebases:

1. Only build and test what was changed.
2. Cache intermediate build results to speed up full builds.
3. Break up the code-base into small units and share binaries.

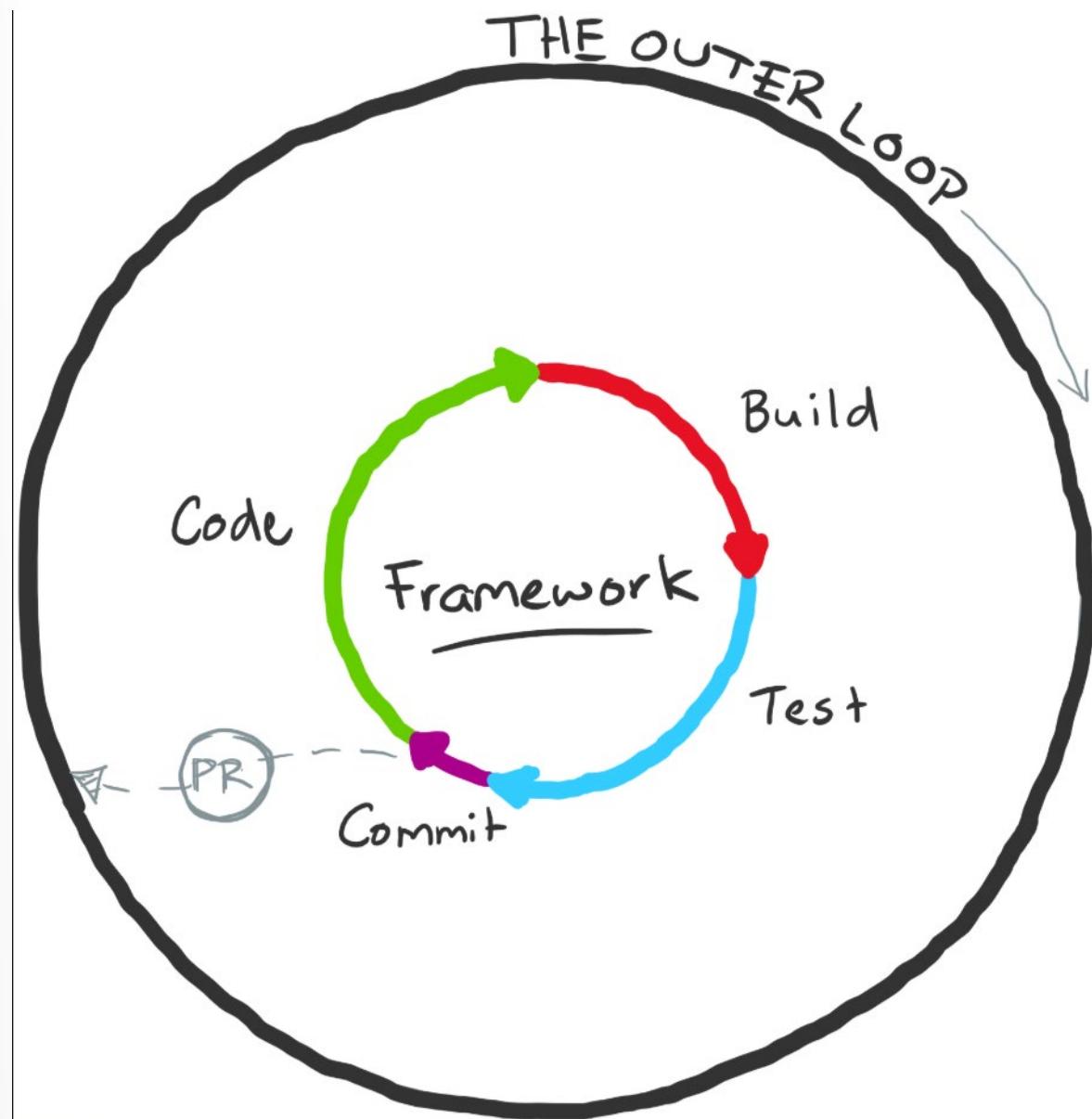
How you tackle each one of those is probably a blog post in their own right. At Microsoft, for some of our truly massive monolithic codebases we are investing quite heavily in #1 and #2 - but #3 requires a special mention because it can be a double edged sword and if done incorrectly and can have the opposite of the desired impact.

Tangled Loops

To understand the problem we need to look beyond the inner loop. Let's say that our monolithic code-base has an application specific framework which does a lot of heavy lifting. It would be tempting to extract that framework into a set of packages.

To do this you would pull that code into a separate repository (optional, but this is generally the way it is done), then setup a separate CI/CD pipeline that builds and publishes the package. This separate build and release pipeline would also be fronted by a separate pull-request process to allow for changes to be inspected before the code is published.

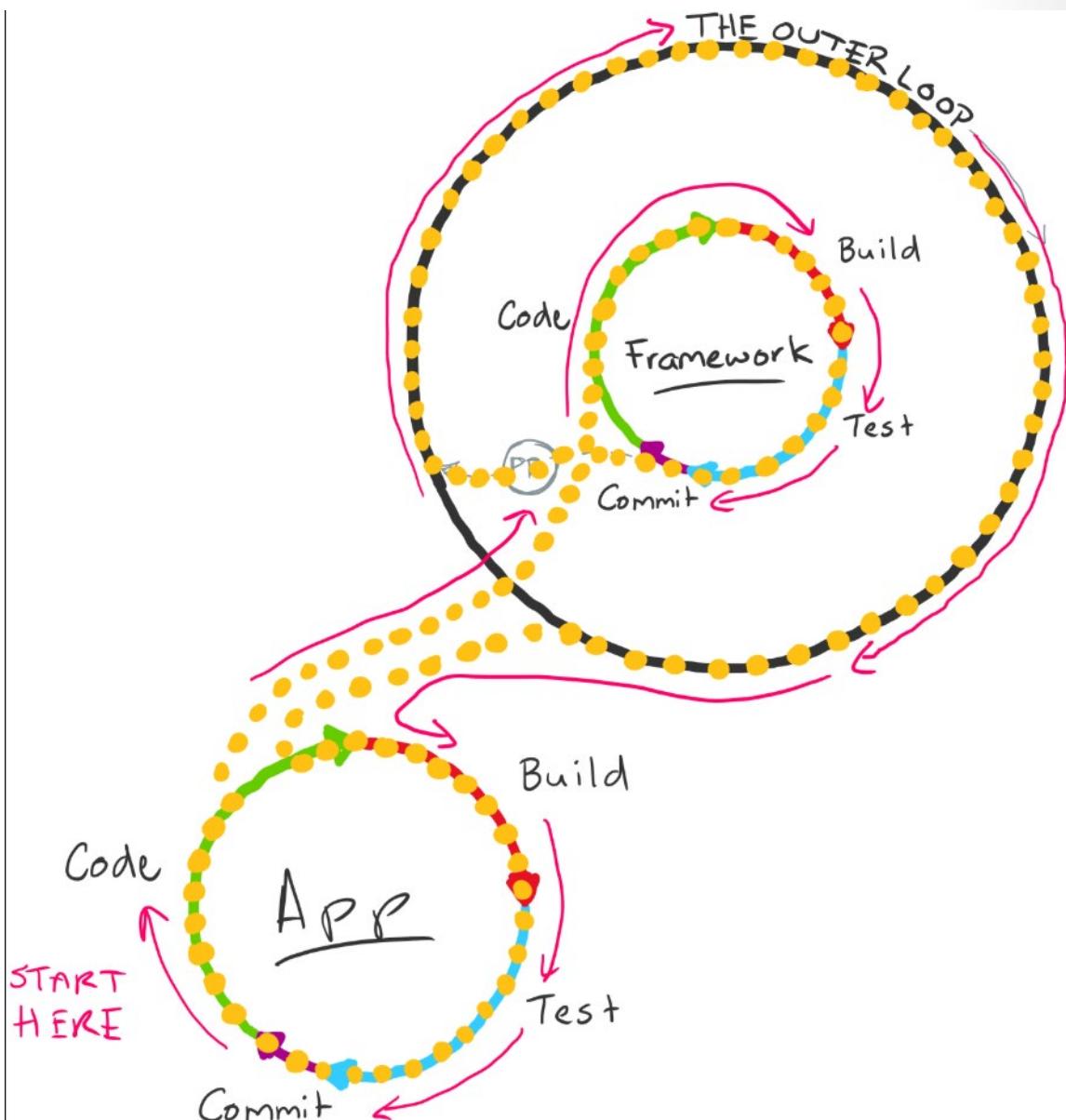
When someone needs to change this framework code they clone down the repository, make their changes (a separate inner loop) and submit a PR which is the transition of the workflow from the inner loop to the outer loop. The framework package would then be available to be pulled into dependent applications (in this case the monolith).



Initially things might work out well, however at some point in the future it is likely that you'll want to develop a new feature in the application that requires extensive new capabilities to be added to the framework. This is where teams that have broken their codebases up in sub-optimal ways will start to feel pain.

If you are having to co-evolve code in two separate repositories where a binary/library dependency is present then you are going to experience some friction. In loop terms - the inner loop of the original codebase now (temporarily at least) includes the outer loop of the framework code that was previously broken out.

Outer loops include a lot of tax such as code reviews, scanning passes, binary signing, release pipelines and approvals. You don't want to pay that every time you've added a method to an class in the framework and now want to use it in your application.



What generally ends up happening next is a series of local hacks by the developer to try and stitch the inner loops together so that they can move forward efficiently - but it gets pretty messy quick and you have to pay that outer loop tax at some point.

This isn't to say that breaking code up into separate packages is an inherently bad thing - it can work brilliantly, you just need to make those incisions carefully.

Closing Thoughts

There is no silver bullet solution that will ensure that your inner loop doesn't start slowing down, but it is important to understand when it starts happening, what the cause is and work to address it.

Decisions such as how you build, test and debug, to the actual architecture itself will all impact how productive developers are. Improving one aspect will often cause issues in another.

Continuous Experimentation Mindset

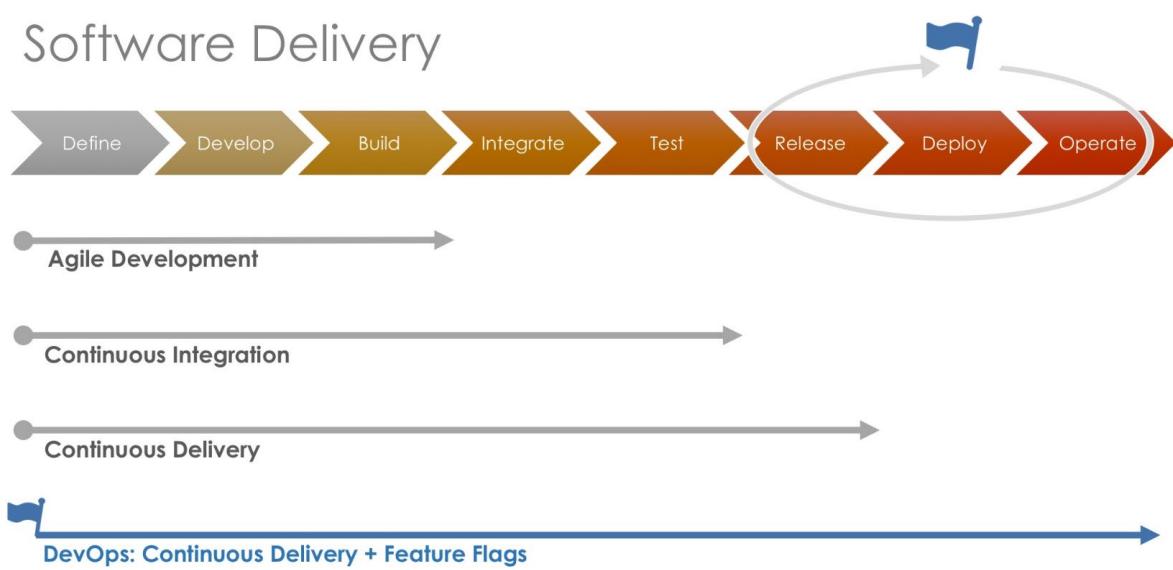
Continuous Experimentation mindset

We are in the era of continuous delivery, where we are expected to quickly deliver software that is stable and performant. We see development teams embracing a suite of continuous integration/delivery tools to automate their testing and QA, all while deploying at an accelerated cadence.

However, no matter how hard we try to mitigate the risk of software delivery, almost all end-user software releases are strictly coupled with some form of code deployment. This means that companies must rely on testing and QA to identify all issues before a release hits production. It also means that companies primarily rely on version control systems or scraped together config files to control feature releases and mitigate risk. Once a release is in production, it is basically out in the wild. Without proper controls, rolling back to previous versions becomes a code deployment exercise, requiring engineering expertise and increasing the potential for downtime.

One way to mitigate risk in feature releases is to introduce **feature flags**¹(feature toggles) into the continuous delivery process. These flags allow features (or any code segment) to be turned on or off for particular users. Feature Flags are a powerful technique, allowing teams to modify system behavior without changing code.

In this new reality, software development and operations capabilities set the boundaries for success. The first step to creating a company-wide culture of experimentation is for executives, the business, developers, and ops—and other key stakeholders—to understand how changes in engineering productivity align with business outcomes. Innovation is the key to success, and success depends on hypothesis-testing through experimentation. By adopting a culture of continuous experimentation, features can be tested by creating an instrumented minimal viable product rapidly and release to a subset of customers in production for testing, this enables the team to make fact based decisions and quickly evolve towards an optimal solution.



By using feature flags in the continuous delivery process, teams are able to efficiently integrate release, deployment, and operational management into the software development cycle.

¹ <http://blog.launchdarkly.com/feature-flag-driven-development/>

Continuous Delivery

Benefits

- Faster software development
- Smaller and more frequent releases
- Automated testing
- Reduced development risk
- Better development coordination
- Quickly adapt to shifting markets

Benefits With Feature Flags

- Decouple rollout from code deployment
- Releases with substantially mitigated risk
- Customer feedback loop
- Percentage rollouts and targeted releases
- Feature rollbacks without redeploying
- Configuration and long-term management

From their onset, feature flagging platforms have accelerated the progression of DevOps practices. Feature flags have always fit into DevOps practices due to the increased control over the delivery cycle that they offer, but the mitigation of risk and prevention of associated technical debt has brought flagging platforms to the spotlight as quintessentially “DevOps”. These changes alongside the newly offered ability to monitor all changes to a flag via an audit log have widened the reach of the methodology. Simultaneously, these systems deeply supplement the methodology as it currently exists and provide additional use cases and benefits that push the limits of DevOps beyond what was possible before.

Using Feature Flags to Safely Get Feedback on Features

We are in an era of continuous delivery, where we are expected to quickly deliver software that is stable and performant. We see software development teams embracing a suite of continuous integration/delivery tools to automate their testing and QA, all while deploying at an accelerated cadence. No matter how hard we try to mitigate the risk of software delivery, almost all end-user software releases are strictly coupled with some form of code deployment. This means that companies must rely on testing and QA to identify all issues before a release hits production. There are two key challenges when testing features in test environments:

- Testing in test environments can be challenging if your test scenarios depend on production quality data. It can take a lot of effort to create this kind of data in test environments and it's likely you'll still miss out on key test scenarios, since in some cases the effort involved in creating this data outweights the benefits.
- The other most common scenario is doing user testing, inspecting, and adapting the functionality of your product based on usage data. End users may be invested in the success of your product, but it can get increasingly difficult to get constant feedback on every functionality in a test environment.

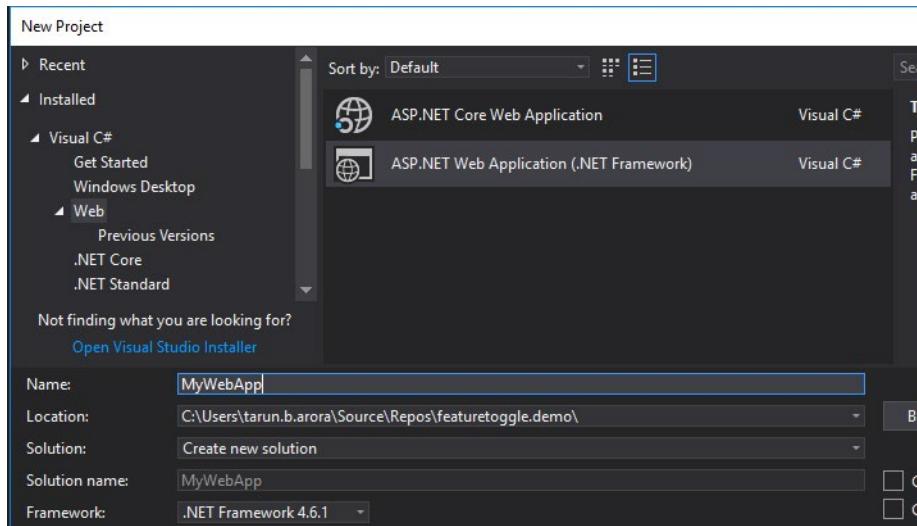
Once a release is in production, it is basically out in the wild. Without proper controls, rolling back to previous versions becomes a code deployment exercise, requiring engineering expertise and increasing the potential for downtime. One way to mitigate risk in feature releases is to introduce feature flags (feature toggles) into the continuous delivery process. These flags allow features (or any code segment) to be turned on or off for particular users. Feature flags are a powerful technique, allowing teams to modify system behaviour without changing code.

Innovation is the key to success, and success depends on hypothesis-testing through experimentation. By adopting a culture of continuous experimentation, features can be tested by creating an instrumented minimal viable product rapidly and released to a subset of customers in production for testing; this enables the team to make fact-based decisions and quickly evolve towards an optimal solution.

In this tutorial we'll learn how to get into a true continuous testing culture by leveraging feature flags.

Getting ready

1. Create a new web application using the ASP.NET Web Application template in Visual Studio, name it MyWebApp, and save it in a new folder called featuretoggle.demo:



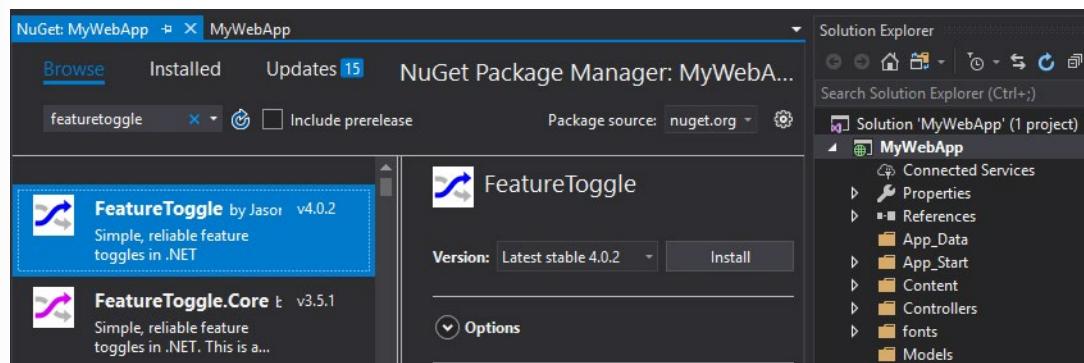
2. Simply build and run the website, then navigate to the Contact form:

The screenshot shows a browser window titled 'Contact - My ASP.NET Application'. The address bar shows 'localhost:64192/Home/Contact'. The page has a dark header with 'Application name' and navigation links for 'Home', 'About', and 'Contact'. The main content area displays the text 'Contact.' and 'Your contact page.' followed by address details: 'One Microsoft Way', 'Redmond, WA 98052-6399', and 'P: 425.555.0100'. It also includes support and marketing contact information: 'Support: Support@example.com' and 'Marketing: Marketing@example.com'. At the bottom, a copyright notice reads '© 2019 - My ASP.NET Application'.

In the next section, we'll see how to use feature flags to deploy changes to the Contact form without releasing the changes to everyone.

How to do it

1. In the MyWebApp project, add a reference to the FeatureToggle package



2. Next, create a folder called Toggle and add a class called NewContactForm.cs. Copy the following code into this class file

```
using FeatureToggle;
namespace MyWebApp.Toggle
{
    public class NewContactForm : SimpleFeatureToggle
    {
    }
}
```

3. Add a new app key in the web.config file, set the key name to. FeatureToggle.NewContactForm and the value to false. This key will be used to control the feature flag:

```
<appSettings>
<add key="webpages:Version" value="3.0.0.0" />
<add key="webpages:Enabled" value="false" />
<add key="ClientValidationEnabled" value="true" />
<add key="UnobtrusiveJavaScriptEnabled" value="true" />
<add key="FeatureToggle.NewContactForm" value="false" />
</appSettings>
```

4. Next, modify the Contact.cshtml page under Views\Home to include the following code:

```
@{ var toggle = new MyWebApp.Toggle.NewContactForm();
if (toggle.FeatureEnabled)
{
    
}
}
```

5. Build and run the project. Navigate to the Contact form page and you'll notice it's unchanged. Update the value of the FeatureToggle.NewContactForm key in the web.config file from false to true.
6. Now refresh the Contact form. You'll see the updated page with the image:



A screenshot of a web browser showing a local host page at localhost:64192/Home/Contact. The page has a header with 'Application name' and navigation links for 'Home', 'About', and 'Contact'. The main content area displays the word 'Contact.' followed by the text 'Your contact page.' Below this is a large image of Satya Nadella, CEO of Microsoft, standing in front of a Microsoft logo. At the bottom left of the image, there is a caption: 'One Microsoft Way Redmond, WA 98052-6399'.

How it works

The feature toggle package includes a series of providers that can be used to control the value of an object that can, in turn, be used to decide whether the feature is accessible. You may ask why we use feature toggle. Well, it is easy to construct a simple if....else condition using a config key to control when the page gets shown. While magic strings can be used, toggles should be real things (objects), not just a loosely typed string. This helps effectively manage the feature flags overtime. When using real toggles you can do the following:

- Find uses of the Toggle class to see where it's used
- Delete the Toggle class and see where a build fails

Feature flags allow you to decouple code deployments from feature releases. This simplifies testing code changes in production without impacting end users. By using feature flags it's possible to control who can see a feature; it's also possible to phase in traffic to a new feature rather than opening up all users at once. You can read more about feature flags and their benefits at this [Feature Toggles blog post²](#)

There's more

The feature toggle package also provides the following feature toggle types:

- AlwaysOffFeatureToggle
- AlwaysOnFeatureToggle
- EnabledOnOrAfterDateFeatureToggle

² <https://martinfowler.com/articles/feature-toggles.html>

- EnabledOnOrBeforeDateFeatureToggle
- EnabledBetweenDatesFeatureToggle
- SimpleFeatureToggle
- RandomFeatureToggle
- EnabledOnDaysOfWeekFeatureToggle
- SqlFeatureToggle
- EnabledOnOrAfterAssemblyVersionWhereToggleIsDefinedToggle

More details on these feature toggle types and their usage can be found at [Installing and Using Feature Toggles³](#)

³ <http://jason-roberts.github.io/FeatureToggle.Docs/pages/usage.html>

Design Practices to Measure End-User Satisfaction

Design practices to measure end user satisfaction

"Measurement is the first step that leads to control and eventually to improvement. If you can't measure something, you can't understand it. If you can't understand it, you can't control it. If you can't control it, you can't improve it." — H. James Harrington

'User experience' encompasses all aspects of the end-user's interaction with the company, its services, and its products. UI (user interface) is not UX (user experience). A car with all its looks, dashboard and steering wheel is the UI. Driving it is the UX. So the interface directly contributes to the experience (beautiful car interior makes a better experience sitting in one), but is not the experience itself.

The only real measure of software quality that's worth its salt is end-user satisfaction. Forget about what applications have been designed to do; forget about developer and service provider promises that don't include an end-user analytical strategy. Cut to the chase and look at what's actually happening; Measuring customer satisfaction doesn't have to be complicated or expensive. In fact, it's fairly simple to incorporate customer satisfaction measurement into your current customer success strategy. No matter how you cut it, measuring customer satisfaction comes down to gathering customer feedback via surveys. To accurately gauge customer sentiment, we'll simply need to ask them how their experience was.

Of course, there are multiple ways you can execute said survey, from the survey design to timing, sample, and even how you analyze the data. Let's briefly cover the five steps to easily measuring customer satisfaction.

1. Outline your goals, and make a plan

When embarking on any sort of campaign, it's helpful to take a step back and ask, "Why are we doing this?" In business, one must weigh the value of additional information (i.e. the customer satisfaction data) in relation to the cost of collecting it (i.e. the survey process). To be honest, if you won't change anything after collecting your customer satisfaction data, you're better off not collecting it. It's going to take time and effort, so you need to put it to use.

Depending on your business or organizational capabilities, there is a lot you can do with the information. One use is simply to wake you up to the reality of any business: A portion of your customers is going to have an unsatisfactory experience. Every business faces this problem.

When you wake up to that fact, you can choose from many routes to correction. You can:

- Improve key UX bottlenecks that contribute to poor customer experience.
- Expedite customer support interactions with the most frustrated customers.
- Operationalize proactive support like a knowledge base and customer education.
- Test different live chat scripts and support strategies.

The specific solution isn't necessarily the important part here. The important part is stepping back and saying, "If we see that a segment of our customers is unsatisfied, what will we do about it?"

2. Create a customer satisfaction survey.

What types of metrics measure customer satisfaction? You can choose among a few different options for

customer satisfaction surveys. There's no unanimous agreement on which one is best. A few popular methods are:

- Customer Satisfaction Score (CSAT)
- Customer Effort Score (CES)
- Net Promoter Score® (NPS)

These are all "one-question" methods that vastly simplify the process of collecting customer insights.

While you may not think the survey methodology matters much, how you ask the question does seem to measure slightly different things.

For instance, a 2010 study found twenty percent of "satisfied" customers said they intended to leave the company in question; 28% of the "dissatisfied" customers intended to stay. So "satisfied" doesn't necessarily equate to "loyal."

- Customer Satisfaction Score (CSAT) is the most commonly used satisfaction method. You just ask your customers to rate their satisfaction on a linear scale. Your survey scale can be 1 – 3, 1 – 5, or 1 – 10, and there is no universal agreement on which scale is best to use.
CSAT is a metric used to immediately evaluate a customer's specific experience. "CSAT is a transactional metric that is based on what's happening now to a user's satisfaction with a product or service."
- Customer Effort Score (CES) is very similar, but instead of asking how satisfied the customer was, you ask them to gauge the ease of their experience. You're still measuring satisfaction, but in this way, you're gauging effort (the assumption being that the easier it is to complete a task, the better the experience). As it turns out, making an experience a low-effort one is one of the greatest ways to reduce frustration and disloyalty.
- Net Promoter Score (NPS) asks the question, "How likely is it that you would recommend this company to a friend or colleague?" This attempts to measure customer satisfaction but also customer loyalty. In doing so, you can come up with an aggregate score, but you can also easily segment your responses into three categories: detractors, passives, and promoters. You calculate your Net Promoter Score by subtracting the percentage of Detractors from the percentage of Promoters.

You can, of course, use more than one methodology, as well (since they all measure something very slightly different). You can optionally combine multiple scores for a greater picture:

For example, a customer that has had 3 continuous negative CSAT scores over a period and is also a detractor on NPS would be an immediate at-risk customer, while a customer with positive CSAT and a promoter on NPS are potentially the best source of advocates & candidates to cross-sell/upsell since they already have seen the value in their interactions with the process & product.

In addition, I recommend always appending a qualitative open-ended question, regardless of the customer satisfaction survey you use. Without an open-ended question, you risk limiting your insight into "why" the dissatisfaction may be occurring. Qualitative user feedback can give you tons of ideas when it comes to implementing solutions.

3. Choose your survey's trigger and timing.

This step is all about to whom you send the survey and when you send it. If you go back to your goals outline, this shouldn't be too hard to determine, at least strategically. People tend to forget this step, though, but it's of crucial importance and affects the quality and utility of your data. Tactically, you can trigger a survey pretty much anywhere at any time and to anyone nowadays, but strategically, it matters specifically when and where.

"Although there is no "one size fits all" approach to customer satisfaction surveys, there are 3 factors that every company should consider before surveying: What event or action took place before you asked for

feedback (these can be time or action based events like completing your onboarding campaign), the time since your last survey to the customer, and is your team's ability to reply to feedback in a timely manner.

Good examples of event data that can be used to fire a survey are:

- Time since signup
- Key actions taken in your app
- Complete user on-boarding
- Surveying too often will result in low response rates, we recommend a customer satisfaction (NPS) survey seven days after signup, 30 days after the first survey and every 90 days during the customer lifecycle."

With all the options for triggering, though, let's start with some best practices:

- The closer the survey is to the experience, the better.
- People forget how they felt the longer you wait.
- Who you survey changes what insights you get. If you survey website visitors about their satisfaction, the respondents are anonymous and may be a customer – or may not be. This will bring you different data than sending an email to recent customers will. Keep that in mind.
- You should survey your customers more than once to see how things change longitudinally. Especially if you operate a SaaS company or a subscription service, regular NPS surveys can help you analyze trends both at the aggregate and individual level.
- Survey people after a key moment of their customer journey.
- If a respondent gives you a high score, think about adding a follow-up ask. For instance, Tinder asks you to rate their app in the app store if you give them a high score.

4. Analyze the survey data.

Once you've collected your data, make sure it doesn't just sit there dormant and unused. You've got all this customer insight, and it's just waiting to be uncovered!

5. Adjust and repeat.

Back to my first point: Now that you have these insights, what are you going to do about it? Ultimately, this is a personal decision that will reflect your own findings and capabilities. You may find that a whole segment is dissatisfied because of a particular experience. In that case, you may need to further investigate why that experience (or product) is causing dissatisfaction and run experiments to try to improve upon it. You may find that you have a small percentage of super fans.

Now that you can identify these people, perhaps you can work with your customer marketing and customer success teams to plan advocacy programs.

The possibilities are endless, but it all starts with accurately measuring customer satisfaction. But asking for scores is only a part of it – make sure you're creating conditions for customers to leave you high scores, too.

In product feedback

It's very likely that if you visit a web property for a little while, you'll eventually be prompted to provide some feedback or sign up for something. Rate the app. Complete a one-question survey... Send a smile/frown, shake the device to provide feedback and various other interesting ways to engage users for feedback.

It is about meeting your customers where they are, done well, **in-product** feedback is a fantastic way to gather impactful, relevant information from people as they use your product. Done poorly, it's a lethally effective way to drive people away or irritate them so much that they want to exact their revenge.

What kind of feedback can you expect?

Feedback from people who are already using your products, who have chosen to visit your website, who have otherwise opted-in to what you are selling...feedback from these people is worth its weight in pixels. Providing them a way to share feedback with you from within the context of your product makes this type of feedback especially valuable. It may be used to improve the usability of a particular feature or to recruit users who are already performing certain actions for deeper discussions. The main value is the immediacy of the feedback given the user's context within your product. The feedback will tend to be more concrete, and users may be more likely to provide candid, quick responses.

Benefits

- Context-Sensitive Feedback. Users will already be using your product, so they can provide feedback based on their actual usage or needs at the time. Priceless.
- Always on. By implementing feedback mechanisms within the product itself, you've made it very easy for users to provide input, at any point, without sending a formal survey or otherwise cluttering an inbox in hopes of getting a hit.
- High response rates. Since the feedback mechanism is built into your services, users are able to access it if and when they need it. That could mean reporting a problem, bug, enhancement or glitch or complementing the team on their choice of user experience.

Weaknesses

- Too. Much. Feedback. There are a lot of channels which users can tap into to provide feedback. Sometimes, there are just too many to stay on top of them all. After all, it would be a shame to collect feedback from multiple channels and not have means to review it all...
- Not enough detail. If you are posting micro-surveys within your site, the information you get back from respondents may not be sufficiently detailed to allow it to be actionable.
- Always on. By implementing feedback mechanisms within the product itself, you've made it very easy for users to provide input, at any point, without sending a formal survey or otherwise cluttering an inbox in hopes of getting a hit. But, sometimes that feedback may be irrelevant given new decisions.
- Limited future follow-up. Depending on the tools being used, you may not have enough contact information to follow-up directly with the person who submitted the feedback and delve deeper into their responses.

Tapping into In-Product feedback is helpful throughout the Product Development Lifecycle... In spite of its weaknesses if done right, In-app feedback is an excellent way to validate existing or proposed functionality, and to solicit ideas for improvements to the status quo.

Once the product is in production, use in-app tools to support users, allowing them to report issues, frustrations, improvements, and enhancements.

If you sell a software product, asking for feedback directly inside the app is a fantastic method for collecting product feedback.

It helps you narrow in on specific issues your customers are experiencing. However, it can also feel like paradox of choice since you can ask ANY question. Here are a few example questions that may be helpful to ask:

- "What is {insert product feature} helping you accomplish?"
- "What issues, if any, are you having with {insert product feature}?"
- "What features do you think we're missing today for {insert product feature}?"

There are hundreds of in-app questions you can ask. Here's a preview of the pros and cons for in-app surveys.

Pros	Cons
Lots of flexibility - you can ask whichever question you see fit, whether you're evaluating a new design, gauging how customers feel about a new feature launch, etc.	Difficult to comb through open-ended responses and extract insights.
Gives us access to the customer/user where they are in the app.	Low response rates.
Gives us context on what the user/customer is looking at in the app right before their response.	No ability to throttle NPS based on if a user has recently responded to feedback – need to be able to suppress certain users.
Allows us to respond in-app so I can keep all of my feedback in one place.	

Feedback on product roadmap

The big thing that gets missed from feedback surveys is priority. Wouldn't it be great if there was a way to capture feedback and allow your customers to vote on what is most useful to them?

Effective communication on your product roadmap helps keep your customers engaged.. However, by giving your users the ability to request for features you can make them part of your product development inner loop. This is powerful, but at the same time you run the risk of driving your product into a direction where it becomes a niche solution only for a subset of the envisaged target market. By increasing the visibility on the requested features you can empower your full customer base to vote on features they would like to see most. This helps add a third dimension to product backlog and aids prioritisation.

It's been shown that increased communication with customers promotes more engagement and feedback. By updating supporters of an idea as it moves through different product development stages - from discovery to planning to development and launch - your customers never feel like they're in the dark.

The Azure DevOps team uses public projects to show its feature plan for the next few months, you can check it out [here](#)⁴.

Pros	Cons
Customers feel that they're an active part of building your product roadmap. It provides a place to make customers feel their voice is heard	Likely biased towards your highest-intent customers. The people who aren't using your product are much more likely to withhold feedback or product suggestions.

⁴ https://dev.azure.com/mseng/Azure%20DevOps%20Roadmap/_workitems/recentlyupdated

Pros	Cons
Builds a sense of community and heightened loyalty when you can collaborate with the company on ideas.	Low volume unless customers are explicitly prompted to suggest an idea in the board.
Provides a channel through which you can make users feel appreciated for their contributions by letting them know that you're taking action on their suggestions.	

Feature Request Board

A massive part of build a product is identifying new features customers desire. The easiest way to figure it out? Ask them! Creating a "feature request board" is a common tool for gauging product feedback from existing customers. Let's see how the integration between User Voice and Azure DevOps can help you create your own feature board for opening the feedback channels with your customers.

For more information, refer to **Azure DevOps Cloud (formerly VSTS) Integration**⁵.

⁵ <https://feedback.uservoice.com/knowledgebase/articles/363410-vsts-azure-devops-integration>

Design processes to capture and analyze user feedback

Introduction

"People are talking about you and your competitors constantly. They could be asking for help, complaining about a bug, or raving about how much they love you. And you want to stay on top of all those conversations. Your customers' opinions matter, not just to PR & marketing, but to every team – from customer support to product development to sales. But sifting through irrelevant posts on multiple channels is overwhelming and can be a huge time drain. Sometimes it's not even possible when they don't tag or link to you. That's where scanning tools comes in. Scanning tools make it easy for you to find people talking about you, but not necessarily to you (when they don't @mention your account) - and reach out or take notes when necessary. There are so many business opportunities to uncover if you know where to look."

Although there isn't a specific survey question you can ask ... it's important to get a general pulse on what your customers are saying over time about your company. Here's the pros and cons:

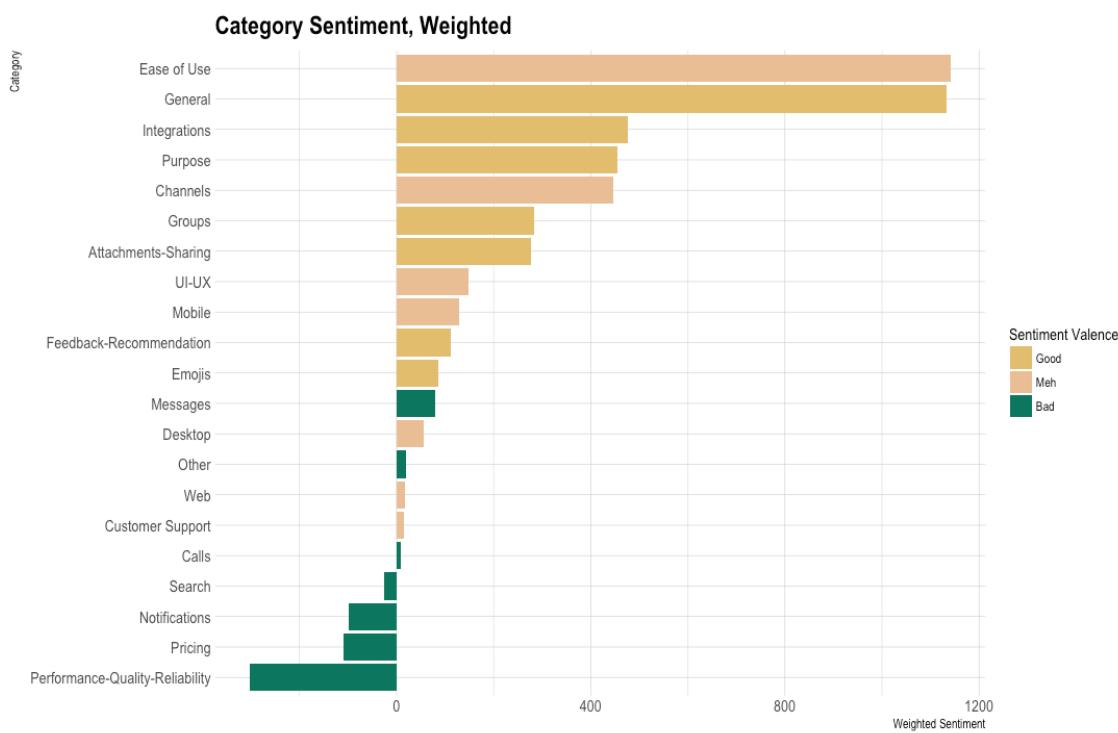
Pros	Cons
No burden on the customer to complete a survey. In their natural environment.	Difficult to measure and quantify which makes it nearly impossible to track performance over time.
Get a true measure of what customers think about you as this method is entirely organic.	Challenging to tie social media comments back to a CRM system at scale.

Customer feedback doesn't just come in through your site's contact form – it's everywhere. You only have to search the Twitter handle of any product with more than a few hundred users to see that customers love to offer their opinion – positive and negative. It's useful to be monitoring this and learning from it, but casually collecting feedback on an ad-hoc basis isn't enough.

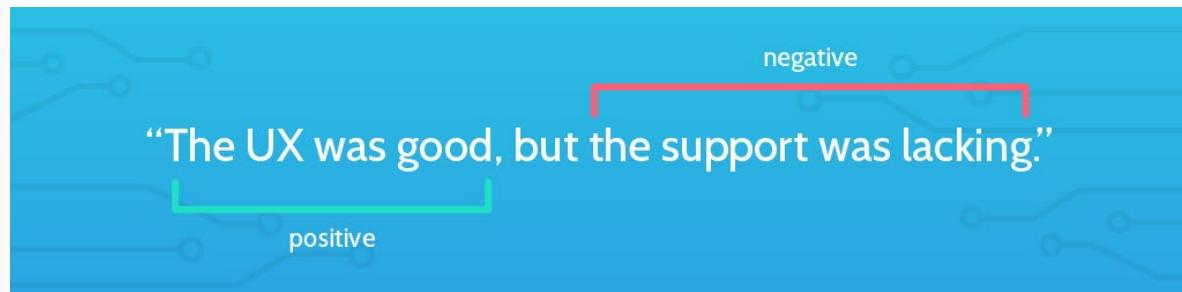
Startups thrive on feedback as their 'North star', and are constantly evolving based on what their customers request, break, and complain about. Enterprises also can't overlook the fact that customers are what make any company tick, and must struggle harder than startups to stay relevant and innovate.

So, if you're just collecting feedback 'as and when' it comes in, you're missing out on data that's just as important as page views or engagement. It's like deciding not to bother setting up Google Analytics on your homepage, or not properly configuring your CRM; in the end, you're deciding to not benefit from data that will have a transformative effect on your product strategy. With a dataset of feedback – whether that's from customer reviews, support tickets, or social media – you can dig into the words your customers are using to describe certain parts of your product and get insights into what they like, and what they don't like.

Here's the kind of end result you can get with this



The outcome of AI analysis on Slack reviews. The categories on the left refer to different parts of Slack's product, and the bars represent how positively or negatively customers feel about each.



As the saying goes, “For every customer who bothers to complain, 20 other customers remain silent.”

Unless the experience is really bad, customers usually don't bother to share feedback about an experience that didn't meet their expectations. Instead, they decide never to do business with the service provider again. That's a high price to pay for lost feedback.

An excellent source of feedback is on other websites, such as online communities, blogs, local listings, and so on. If your customers are not happy with the resolution to a negative experience, they are likely to vent their ire on these forums. The lost customer is not the only casualty. Studies have shown that each dissatisfied customer typically shares the unsatisfactory experience with 8 to 10 (sometimes even 20) others. With the growing use of social media, it's not uncommon for negative feedback to go viral and hurt the credibility of a brand.

Release Gates

Any responsible DevOps practice uses techniques to limit the damage done by bugs that get deployed into production. One of the common techniques is to break up a production environment into a set of separate instances of an app and then configure deployments to only update one instance at a time, with a waiting period between them. During that waiting period, you watch for any signs (telemetry, customer complaints, etc.) that there is a problem and if so, halt the deployment, fix the issue and then continue the deployment. This way, any bug you deploy only affects a small fraction of your user base. In fact, often, the first product environment in the sequence is often one only available to internal people in your organization so you can validate the changes before they hit “real” customers. None-the-less, sometimes issues make it through.

Release gates automate the waiting period between environments in release pipelines. They enable you to configure conditions that will cause the release wait. Out of the box, a few conditions are supported namely Azure monitoring alerts and Work item queries. Using the first, you can have your release hold if your monitoring alerts are indicating that the environments you’ve already deployed to are unhealthy. And the second allows you to automatically pause releases if anyone files a “blocking bug” against the release.

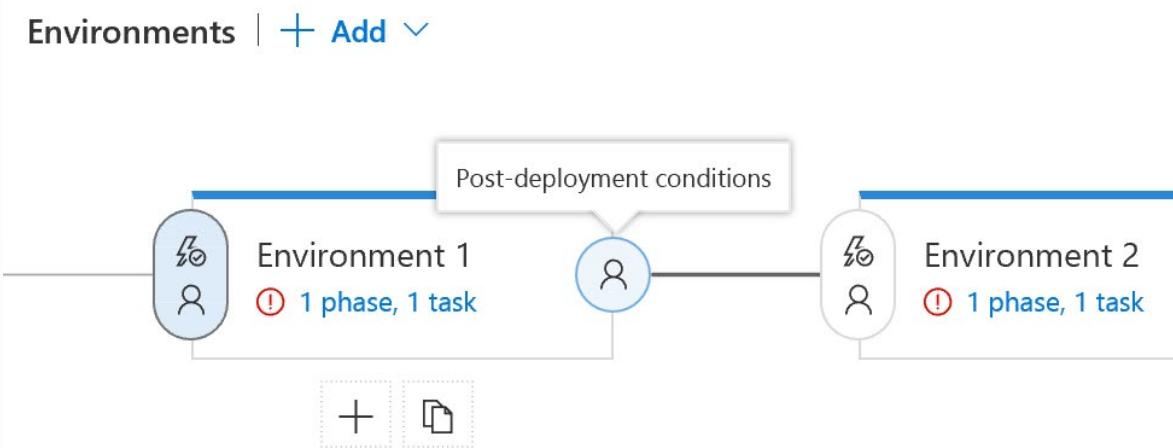
However, one of the things you’ll quickly learn is that no amount of monitoring will catch every single problem and, particularly, if you have a popular application, your users will know within seconds and turn very quickly to Twitter to start asking about the problem. Twitter can be a wonderful “alert” to let you know something is wrong with your app.

The Twitter sentiment release gate that can be downloaded from the [Visual Studio Marketplace](#)⁶ enables exactly this. It leverages Azure DevOps, Azure functions and Microsoft AI to analyze sentiment on your Twitter handle and gate your release progress based on it. The current implementation of the analysis is relatively simple and serves as a sample as much as anything else. It shows how easy it is to extend Azure DevOps release gates to measure any signal you choose and use that signal to manage your release process.

Once you install the Twitter sentiment extension from the marketplace, you’ll need to follow the instructions to configure an Azure function to measure and analyze your sentiment. Then you can go into your Azure DevOps release pipelines and you will find a new release gate enabled.

Start by clicking on Post-deployment conditions on one of your environments.

⁶ <https://marketplace.visualstudio.com/items?itemName=ms-devlabs.vss-services-twittersentimentanalysis>



Then enable the release gates.



Then choose the Twitter Sentiment item and configure it.

The screenshot shows the configuration of a release gate. On the left, there is a list of items: 'Pre-deployment approvals' (with a note to select approvers), 'Gates*' (with a note to define gates), and 'Delay before evaluation *' (set to 15). On the right, a modal dialog is open for the 'Get Twitter Sentiment' gate. The dialog lists five options: 'Azure Monitor' (observe configured Azure monitor rules), 'Get Twitter Sentiment' (gate releases based on average sentiment of tweets for a hashtag), 'Invoke Azure Function' (invoke Azure Function as part of the process), 'Invoke REST API' (invoke REST API as part of the process), and 'Query Work Items' (execute a work item query and check for the number of items returned). The 'Get Twitter Sentiment' option is highlighted. At the bottom of the dialog, there is a 'Deployment gates' button with a note and a '+ Add' button.

You can read up more about release gates at [Release deployment control using gates⁷](#).

⁷ <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/approvals/gates?view=vsts>

Deploy with RM Green light capabilities

A release process comprises of more than just deployment. Therefore a release pipeline needs more than just steps for application deployment. There are a whole raft of checks one would prefer to do before releasing a new version of the software through a release pipeline. To name a few... checks to ensure that there are no active critical issues open and being investigated, that there are no known service degradation or performance alerts. While the deployment process is automated, a lot of these checks are carried out manually which force automated release pipelines to have manual approval steps. No more! Azure Release Pipelines supports Gates.

Gates allow automatic collection of health signals from external services, and then promote the release when all the signals are successful at the same time or stop the deployment on timeout. Typically, gates are used in connection with incident management, problem management, change management, monitoring, and external approval systems. Gates provide you a way to enable progressive exposure and phased deployments

Getting Started

1. Open the desired Team Project in AzureDevOps, navigate to the Queries page under Azure Boards.

The screenshot shows the Azure Boards interface. The top navigation bar includes 'Geeks / PartsUnlimited / Boards / Work Items'. Below the navigation is a toolbar with 'Recently updated', 'New Work Item', 'Open in Queries', 'Column Options', and 'Recycle Bin'. A search bar says 'Filter by keyword'. On the left is a sidebar with icons for 'Work Items', 'Boards', 'Backlogs', 'Sprints', 'Queries' (which is selected and highlighted in yellow), and 'Plans'. The main area displays a list of work items:

- PR banner wording with compliance - Assigned To: Anuradha Arora
- Create the issue in staging after recreating the auth workflow f... - Assigned To: Anuradha Arora
- ture flag to test workflow with customers - Assigned To: Tarun Arora
- ferral workflow - Assigned To: Tarun Arora
- d for review - Assigned To: Tarun Arora

2. Click to create a new Query, add filter to return work item types Bug of status New with priority 1 and

The screenshot shows the 'Shared Queries' page. The top navigation bar includes 'Queries > Shared Queries > Untriaged Critical Bugs'. Below the navigation is a toolbar with 'Editor' (which is selected and highlighted in blue), 'Run query', 'New', 'Save query', 'Save as...', 'Rename', 'Revert changes', and 'Column options'. The main area shows the query details:

Type of query: Flat list of work items

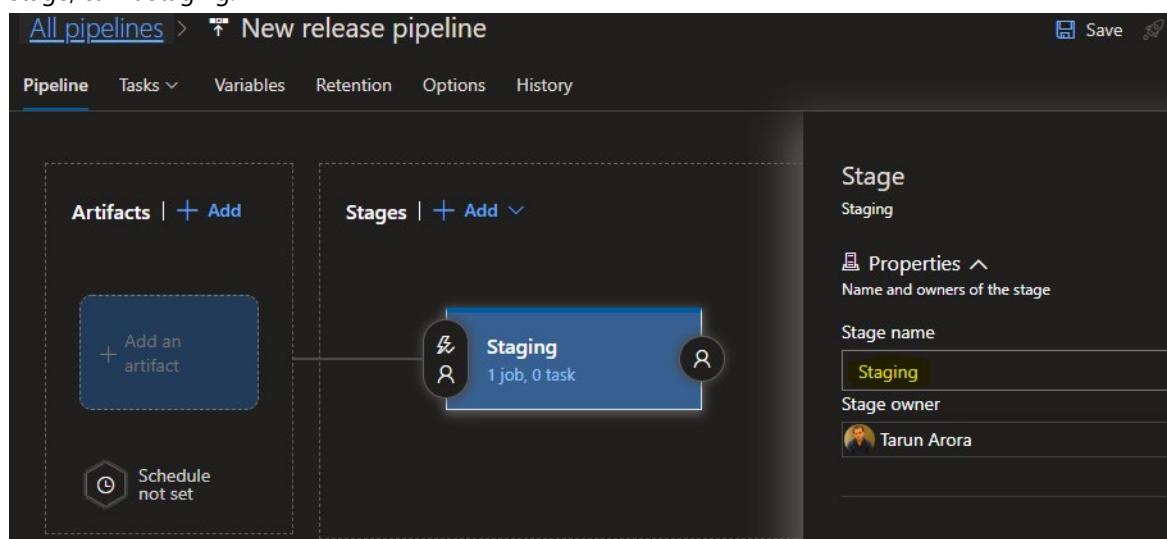
Filters for top level work items:

And/Or	Field	Operator	Value
+ X	Work Item Type	=	Bug
+ X	And State	=	New
+ X	And Priority	=	1
+ X	And Severity	=	1 - Critical

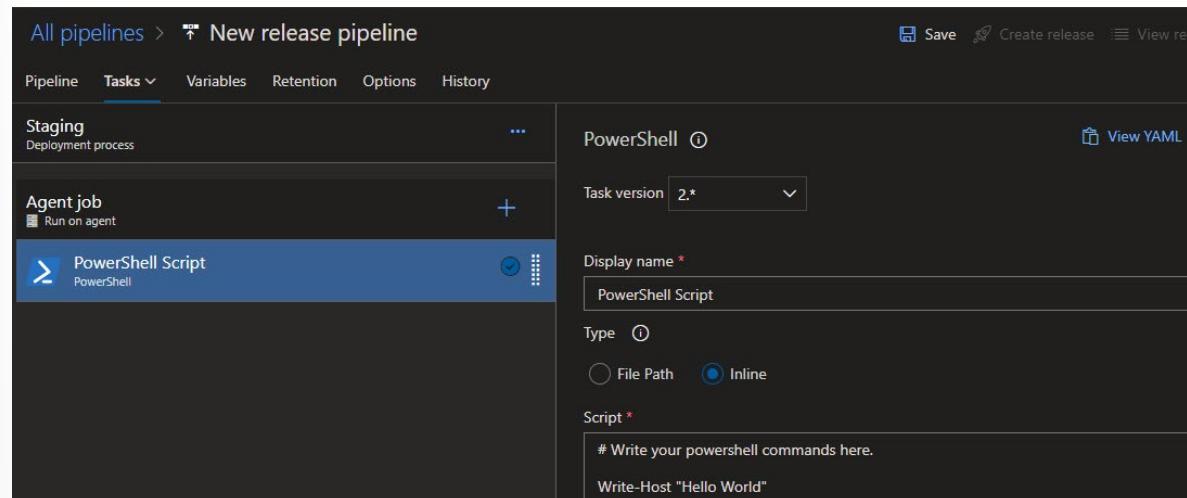
Add new clause

severity Critical. Save the query in shared queries as Untriaged Critical Bugs

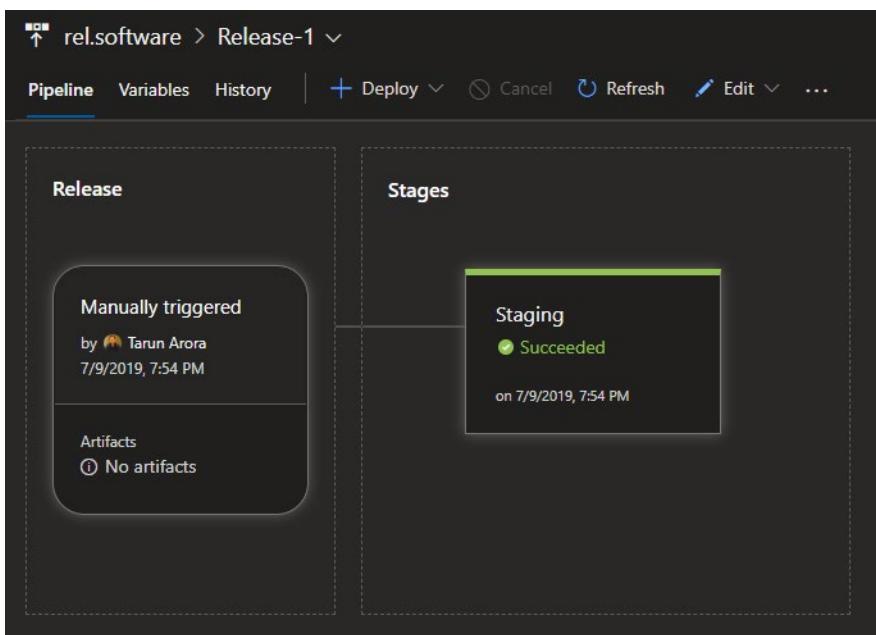
3. In the same team project, navigate to Azure Pipelines. Create an empty release pipeline with one stage, call it staging.



4. In Staging add a PowerShell task, set it to inline mode to print "Hello World"



5. Name the pipeline as rel.software and click save. Create a release and see the release go through successfully.

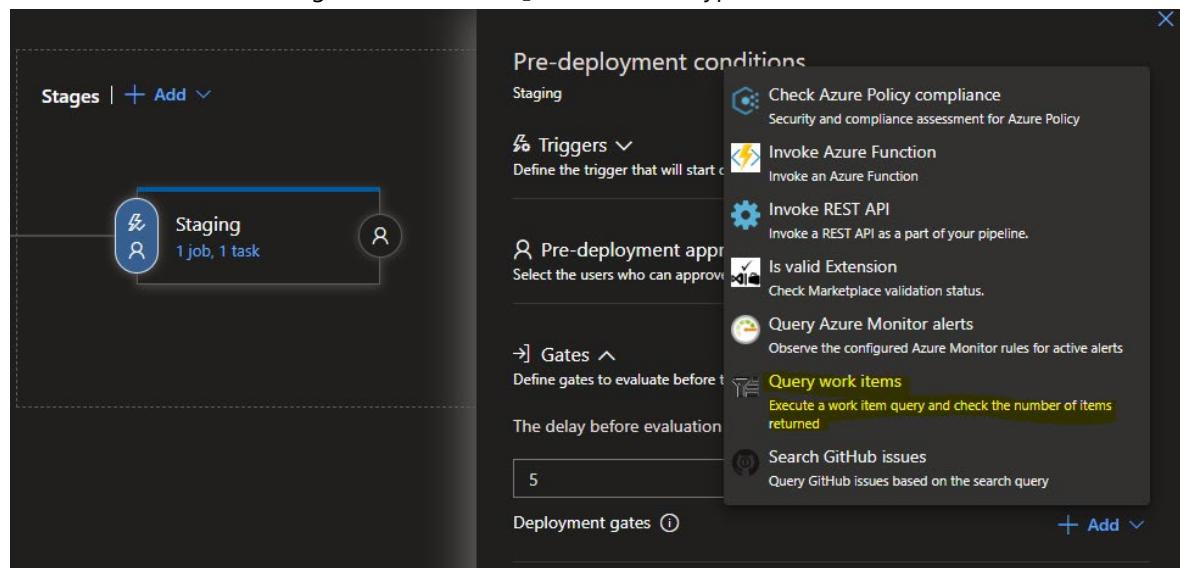


How to do it

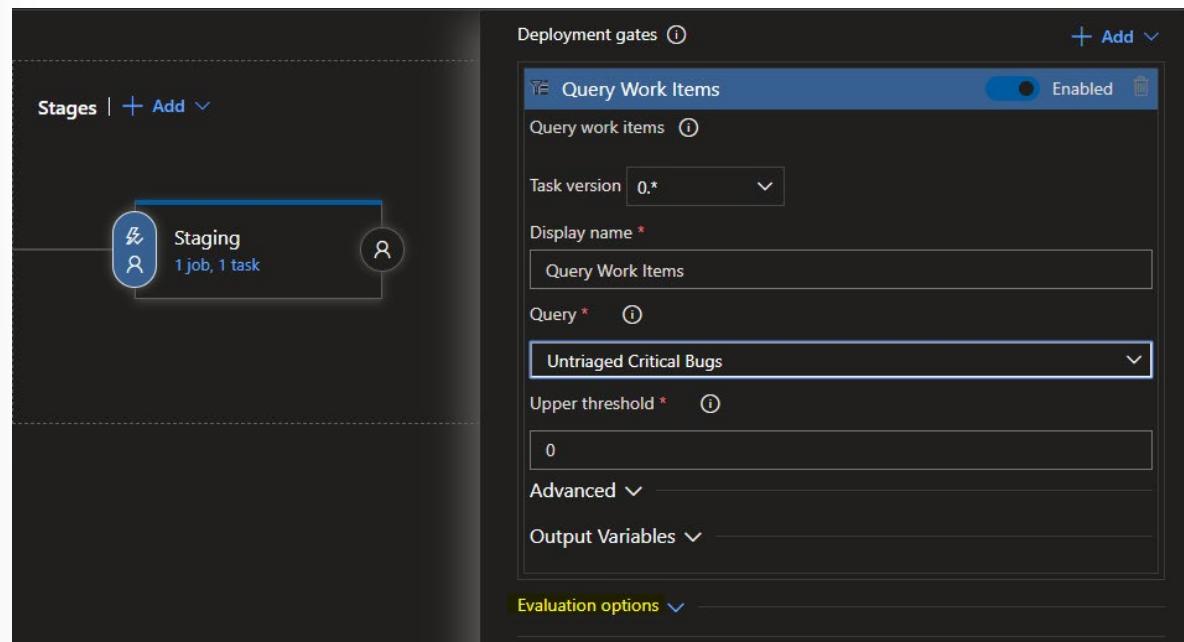
1. Edit the release pipeline, click on pre-deployment condition and from the right pane enable gates.

A screenshot of the 'All pipelines' interface. The left side shows the pipeline configuration with sections for 'Artifacts' (with an 'Add an artifact' button) and 'Stages' (with a 'Staging' stage containing 1 job and 1 task). On the right, the 'Pre-deployment conditions' section is expanded. It includes a 'Triggers' section (disabled), a 'Pre-deployment approvals' section (disabled), and a 'Gates' section which is enabled. Under 'Gates', there is a field for 'The delay before evaluation' set to '5 Minutes'. A 'Deployment gates' section with a '+ Add' button is also visible.

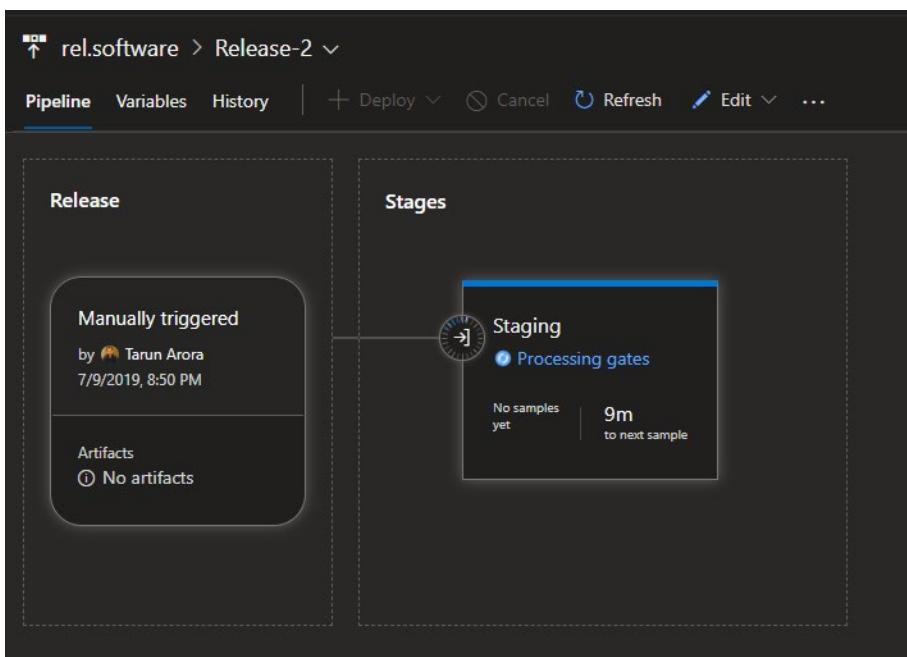
2. Click on Add to add a new gate, choose Query Work Item type Gate



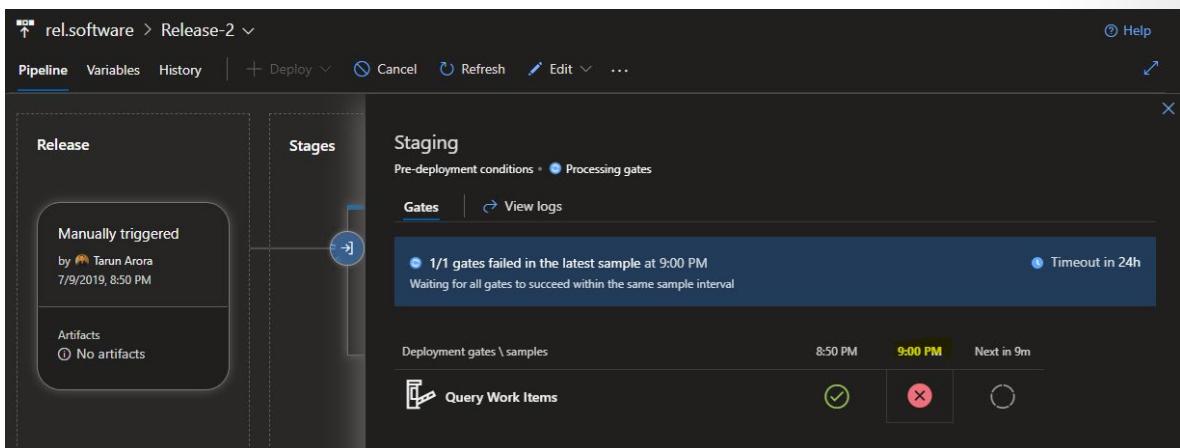
3. From the query work item drop down select the shared query 'untriaged critical bugs' created earlier and set the upper threshold to 0. Set the delay between evaluation period to 1 minute. Expand the evaluation options section and ensure that the time between evaluation of gates is set to 10 minutes and the Minimum duration for steady results results after a successful gates evaluation is configured to 15 minutes.



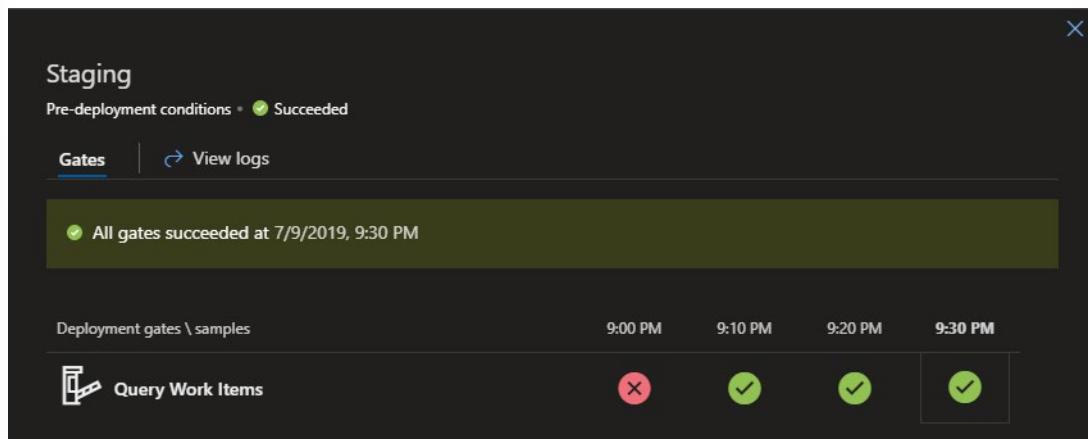
4. Save and queue a release. You'll see that the gate gets evaluated right away...



5. Now before the next evaluation of the gate, create a new work item of type Bug with Priority 1 and severity critical. Wait for the next evaluation of the gate in the release pipeline to complete.



6. Close the bug as fixed, you'll see that after periodic evaluations and a stable period of 15 minutes the release is completed successfully.



How it works

- When a new release is triggered, the release goes into a state of pre-approvals. At this time, the automated gate is evaluated at the specified interval of 10 minutes. The release will only move into approval state if the gate passes for the duration of steady results, specified as 15 minutes in this case. As you can see in the logs below, the gate failed in the 2nd validation check as one critical bug of 1 priority is identified in new state using the configured work item query.

A screenshot of the Azure DevOps Query Work Items log. The title is "Query Work Items" with a red 'X' icon. To the right, it says "Sample at 9:00 PM - 1/1 gates Failed". The log shows the following JSON output:

```
21 |     "referenceName": "System.WorkItemType",
22 |     "name": "Work Item Type",
23 |     "url": "https://dev.azure.com/Geeks/_apis/wit/fields/System.WorkItemType"
24 | },
25 | {
26 |     "referenceName": "System.Title",
27 |     "name": "Title",
28 |     "url": "https://dev.azure.com/Geeks/_apis/wit/fields/System.Title"
29 | },
30 | {
31 |     "referenceName": "System.AssignedTo",
32 |     "name": "Assigned To",
33 |     "url": "https://dev.azure.com/Geeks/_apis/wit/fields/System.AssignedTo"
34 | },
35 | {
36 |     "referenceName": "System.State",
37 |     "name": "State",
38 |     "url": "https://dev.azure.com/Geeks/_apis/wit/fields/System.State"
39 | },
40 | {
41 |     "referenceName": "System.Tags",
42 |     "name": "Tags",
43 |     "url": "https://dev.azure.com/Geeks/_apis/wit/fields/System.Tags"
44 | },
45 | ],
46 | "workItems": [
47 | {
48 |     "id": 1418,
49 |     "url": "https://dev.azure.com/Geeks/5ca464fe-a04d-4971-8c38-619d7b38bd1d/_apis/wit/workItems/1418"
50 | }
51 | ]
52 | }
53 | ||| Evaluation of expression 'xor(and(or(eq(root['queryType'], 'oneHop'), eq(root['queryType'], 'tree')), and(le(count(roo
54 |
```

A small screenshot of the Azure DevOps interface is visible on the right side of the log window.

- Detailed logs of the gate checks can be downloaded and inspected along with the release pipeline logs.

There's more

In this tutorial we looked at the Work Item Query gate. The following other gates are supported out of the box.

The following gates are available by default:

- Invoke Azure function: Trigger execution of an Azure function and ensure a successful completion. For more details, see Azure function task.
- Query Azure monitor alerts: Observe the configured Azure monitor alert rules for active alerts. For more details, see Azure monitor task.
- Invoke REST API: Make a call to a REST API and continue if it returns a successful response. For more details, see HTTP REST API task.
- Query Work items: Ensure the number of matching work items returned from a query is within a threshold. For more details, see Work item query task.
- Security and compliance assessment: Assess Azure Policy compliance on resources within the scope of a given subscription and resource group, and optionally at a specific resource level. For more details, see Security Compliance and Assessment task.

In addition to this you can develop your own gates using the Azure DevOps API's, check out the gates developed by the community for inspiration [here⁸](#)

⁸ <https://www.visualstudiogeeks.com/DevOps/IntegratingServiceNowWithVstsReleaseManagementUsingDeploymentGate>

Design process to automate application analytics

Introduction

In an Agile environment, you may typically find multiple development teams that work simultaneously, introducing new code or code changes on a daily basis, and sometimes several times a day. In such a rapid environment it is extremely common to find problems that have "slipped through the cracks" to find themselves in the production environment. When these issues arise they have probably already impacted end-users, requiring a speedy resolution.

This means that teams have to conduct an extremely rapid investigation to identify the root cause of the problem. Identifying where these symptoms are coming from and then isolating the root cause is a challenging task. Symptoms can be found across various layers of a large hybrid IT environment, such as different servers/VMs, storage devices, databases, to the front-end and server side code. Investigations that traditionally would take hours or days to complete must be completed within minutes.

Teams must examine the infrastructure and application logs as part of this investigation process. However, the massive amount of log records that are produced in these environments makes it virtually impossible to do this manually. It's much like trying to find a needle in a haystack. In most cases these investigations are conducted through the use of log management and analysis systems that collect and aggregate these logs (from infrastructure elements and applications), centralizing them in a single place and then providing search capabilities to explore the data. These solutions make it possible to conduct the investigation, but they still rely completely on the investigation skills and the knowledge of the user. The user must know exactly what to search for, and have a deep understanding of the environment in order to use them effectively.

It's important to understand that the log files of applications are far less predictable than the log files of infrastructure elements. The errors are essentially messages and error numbers that have been introduced to the code by developers in a non-consistent manner. Consequently, in most cases search queries yield thousands of results and do not include important ones, even when the user is skilled. That leaves the user with the same "needle in the haystack" situation.

Assisting DevOps with Augmented Search

A new breed of log management and analysis technologies has evolved to solve this challenge. These technologies expedite the identification and investigation processes through the use of Augmented Search. Designed specifically to deal with the chaotic and unpredictable nature of application logs, Augmented Search takes into consideration that users don't necessarily know what to search for, especially in the chaotic application layer environment.

The analysis algorithm automatically identifies errors, risk factors, and problem indicators, while analyzing their severity by combining semantic processing, statistical models, and machine learning to analyze and "understand" the events in the logs. These insights are displayed as intelligence layers on top of the search results, helping the user to quickly discover the most relevant and important information.

Although, DevOps engineers may be familiar with the infrastructure and system architecture the data is constantly changing with continuous fast pace deployment cycles and constant code changes. This means that DevOps teams can use their intuition and knowledge to start investigating each problem, but they have blind spots that consume time due to dynamic nature of the log data.

Combining the decisions that DevOps engineers makes during their investigation with the Augmented Search engine information layers on the important problems that occurred during the period of interest

can help guide them through these blind spots quickly. The combination of the user's intellect, acquaintance with the system's architecture, and Augmented Search machine learning capabilities on the dynamic data makes it faster and easier to focus on the most relevant data. Here's how that works in practice:

One of the servers went down and any attempt to reinitiate the server has failed. However, since the process is running, the server seems to be up. In this case, end-users are complaining that an application is not responding. This symptom could be related to many problems in a complex environment with a large number of servers.

Focusing on the server that is behind this problem can be difficult, as it seems to be up. But finding the root cause of the problem requires a lengthy investigation even when you know which server is behind this problem.

Augmented Search will display a layer which highlights critical events that occurred during the specified time period instead of going over thousands of search results. These highlights provide information regarding the sources of the events, assisting in the triage process. At this point, DevOps engineers can understand the impact of the problem (e.g. which servers are affected by it) and then continue the investigation to find the root cause of these problems.

Using Augmented Search, DevOps engineers can identify a problem and the root cause in a matter of seconds instead of examining thousands of log events or running multiple checks on the various servers. Adding this type of visibility to log analysis, and the ability to surface critical events out of tens of thousands - and often millions - of events, is essential in a fast paced environment, in which changes are constantly introduced.

Recommend system and feature usage tracking tools

A key factor to automating feedback is telemetry. By inserting telemetric data into your production application and environment, the DevOps team can automate feedback mechanisms while monitoring applications in real-time. DevOps teams use telemetry to see and solve problems as they occur, but this data can be useful to both technical and business users.

When properly instrumented, telemetry can also be used to see and understand in real time how customers are engaging with the application. This could be critical information for product managers, marketing teams, and customer support. Thus it's important that feedback mechanisms share continuous intelligence with all stakeholders.

What is Telemetry and Why Should I Care?

In the software development world, telemetry can offer insights on which features end users use most, detection of bugs and issues, and offering better visibility into performance without the need to solicit feedback directly from users. In DevOps and the world of modern cloud apps, we are tracking the health and performance of an application. That telemetry data comes from application logs, infrastructure logs, metrics and events. The measurements are things like memory consumption, CPU performance, and database response time, events can be used to measure everything else such as when a user logged in, when an item is added to a basket, when a sale is made, etc.

The concept of telemetry is often confused with just logging. But logging is a tool used in the development process to diagnose errors and code flows, and it's focused on the internal structure of a website, app, or another development project. But logging only gives you a single dimension view, combined with insights of infrastructure logs, metrics and events you have a 360 degree view to understand user intent and behaviour. Once a project is released, telemetry is what you're looking for to enable automatic

collection of data from real-world use. Telemetry is what makes it possible to collect all that raw data that becomes valuable, actionable analytics.

Benefits of Telemetry

The primary benefit of telemetry is the ability of an end user to monitor the state of an object or environment while physically far removed from it. Once you've shipped a product, you can't be physically present, peering over the shoulders of thousands (or millions) of users as they engage with your product to find out what works, what's easy, and what's cumbersome. Thanks to telemetry, those insights can be delivered directly into a dashboard for you to analyze and act on.

Because telemetry provides insights into how well your product is working for your end users – as they use it – it's an incredibly valuable tool for ongoing performance monitoring and management. Plus, you can use the data you've gathered from version 1.0 to drive improvements and prioritize updates for your release of version 2.0.

Telemetry enables you to answer questions such as:

- Are your customers using the features you expect? How are they engaging with your product?
- How frequently are users engaging with your app, and for what duration?
- What settings options do users select most? Do they prefer certain display types, input modalities, screen orientation, or other device configurations?
- What happens when crashes occur? Are crashes happening more frequently when certain features or functions are used? What's the context surrounding a crash?

Obviously, the answers to these and the many other questions that can be answered with telemetry are invaluable to the development process, enabling you to make continuous improvements and introduce new features that, to your end users, may seem as though you've been reading their minds – which you have been, thanks to telemetry.

Challenges of Telemetry

Telemetry is clearly a fantastic technology, but it's not without its challenges. The most prominent challenge – and a commonly occurring issue – is not with telemetry itself, but with your end users and their willingness to allow what some see as Big Brother-esque spying. In short, some users immediately turn it off when they notice it, meaning any data generated from their use of your product won't be gathered or reported.

That means the experience of those users won't be accounted for when it comes to planning your future roadmap, fixing bugs, or addressing other issues in your app. Although this isn't necessarily a problem by itself, the issue is that users who tend to disallow these types of technologies can tend to fall into the more tech-savvy portion of your user base. This can result in the dumbing-down of software. Other users, on the other hand, take no notice to telemetry happening behind the scenes or simply ignore it if they do.

It's a problem without a clear solution — and it doesn't negate the overall power of telemetry for driving development — but one to keep in mind as you analyze your data. Therefore when designing a strategy for how you consider the feedback from application telemetry it's important to account for users who don't participate in providing the telemetry.

Recommend-monitoring-tools-and-technologies

Continuous monitoring of applications in production environments is typically implemented with application performance management (APM) solutions that intelligently monitor, analyze and manage cloud, on-premise and hybrid applications and IT infrastructure. These APM solutions enable you to monitor your users' experience and improve the stability of your application infrastructure. It helps identify the root cause of issues quickly to proactively prevent outages and keep users satisfied.

With a DevOps approach, we are also seeing more customers broaden the scope of continuous monitoring into the staging, testing and even development environments. This is possible because development and test teams that are following a DevOps approach are striving to use production-like environments for testing as much as possible. By running APM solutions earlier in the life cycle, development teams get feedback in advance of how applications will eventually perform in production and can take corrective action much earlier. In addition, operations teams that now are advising the development teams get advance knowledge and experience to better prepare and tune the production environment, resulting in far more stable releases into production.

Applications are more business critical than ever. They must be always up, always fast and always improving. Embracing a DevOps approach will allow you to reduce your cycle times to hours instead of months, but you have to keep ensuring a great user experience! Continuous monitoring of your entire DevOps life cycle will ensure development and operations teams collaborate to optimize the user experience every step of the way, leaving more time for your next big innovation.

When shortlisting a monitoring tool, you should seek the following advanced features:

Synthetic Monitoring: Developers, testers and operations staff all need to ensure that their internet and intranet mobile applications and web applications are tested and operate successfully from different points of presence around the world.

Alert Management: Developers, testers and operations staff all need to send notifications via email, voice mail, text, mobile push notifications and Slack messages when specific situations or events occur in development, testing or production environments, to get the right people's attention and to manage their response.

Deployment Automation: Developers, testers and operations staff use different tools to schedule and deploy complex applications and configure them in development, testing and production environments. We will discuss the best practices for these teams to collaborate effectively and efficiently and avoid potential duplication and erroneous information.

Analytics: Developers need to be able to look for patterns in log messages to identify if there is a problem in the code. Operations need to do root cause analysis across multiple log files to identify the source of the problem in complex application and systems.

Lab

Integration between Azure DevOps and Teams

Microsoft Teams is your chat-centered workspace in Office 365. Software development teams get instant access to everything they need in a dedicated hub for teamwork that brings your teams, conversations, content and tools from across Office 365 and Azure DevOps together into one place to help improve the inner feedback loop.

Complete this lab, **Microsoft Teams with Azure DevOps Services (Collaborate, Communicate and Celebrate)**⁹, to learn about how Azure DevOps integrates with Microsoft Teams to provide a comprehensive chat and collaboration experience, across your Agile and development work.

⁹ <https://azuredevopslabs.com/labs/vsts/teams>

Module Review and Takeaways

Module Review Questions

Checkbox

What are some of the ways to measure end user satisfaction for your product?

- CSAT
- CES
- STAR
- NPM

Multiple choice

True or False: Azure DevOps has a feature request board.

- True
- False

Checkbox

Release Gates in Azure DevOps support following gates by default? Select all that apply.

- Invoke Azure Function
- Access Azure Policy Security & Compliance assessment
- Invoke SQL Queries
- Query Azure Monitor Alerts

Checkbox

Which statement(s) correctly describes feature flags? Select all that apply.

- Feature Flags can be used as a substitute for if-else statements to improve code navigation
- Feature Flags can be used to deploy changes with releasing them
- Feature Flags can be used to control exposure to new product functionality
- Feature Flags can be used to perform testing of new features in production

Answers

Checkbox

What are some of the ways to measure end user satisfaction for your product?

- CSAT
- CES
- STAR
- NPM

Multiple choice

True or False: Azure DevOps has a feature request board.

- True
- False

Checkbox

Release Gates in Azure DevOps support following gates by default? Select all that apply.

- Invoke Azure Function
- Access Azure Policy Security & Compliance assessment
- Invoke SQL Queries
- Query Azure Monitor Alerts

Checkbox

Which statement(s) correctly describes feature flags? Select all that apply.

- Feature Flags can be used as a substitute for if-else statements to improve code navigation
- Feature Flags can be used to deploy changes with releasing them
- Feature Flags can be used to control exposure to new product functionality
- Feature Flags can be used to perform testing of new features in production

Module 20 Optimize Feedback Mechanisms

Module Overview

Module Overview

A core part of DevOps knowledge is being able to monitor and optimize applications. Managing production systems takes much more than just monitoring applications, there is both a technical and cultural side to it. Site reliability engineering (SRE) was born at Google in 2003, prior to the DevOps movement. It proved to be so successful that it was adopted by other companies such as Amazon, Netflix, Microsoft, etc. A successful DevOps process requires a continuously optimized feedback loop.

Learning Objectives

After completing this module, students will be able to:

- Analyze alerts to establish a baseline
- Analyze telemetry to establish a baseline
- Perform live site reviews and capture feedback for system outages
- Perform ongoing tuning to reduce meaningless or non-actionable alerts

Site Reliability Engineering

What is Site Reliability Engineering

Software developers spend a lot of time chasing bugs and putting out production fires. Thanks to agile development, we are constantly shipping new code. By-products of constant change are constant issues with performance, software defects, and other issues that eat up our time. Web applications that receive even a modest amount of traffic require constant care and feeding. This includes overseeing deployments, monitoring overall performance, reviewing error logs, and troubleshooting software defects. These tasks have traditionally been handled by a mixture of lead developers, development management, system administrators and more often than not, nobody. The problem is that these critical tasks lacked a clear owner ... until now.

What is site reliability engineering?

Site reliability engineering (SRE) empowers software developers to own the ongoing daily operation of their applications in production. The goal is to bridge the gap between the development team that wants to ship things as fast as possible and the operations team that doesn't want anything to blow up in production. In many organizations, you could argue that site reliability engineering eliminates much of the IT operations workload related to application monitoring. It shifts the responsibility to be part of the development team itself.

"Fundamentally, it's what happens when you ask a software engineer to design an operations function."

– Niall Murphy, Google

Site reliability engineers typically spend up to 50% of their time dealing with the daily care and feeding of software applications. They spend the rest of their time writing code like any other software developer would.

A key skill of a software reliability engineer is that they have a deep understanding of the application, the code, and how it runs, is configured, and scales. That knowledge is what makes them so valuable at also monitoring and supporting it as a site reliability engineer.

Some of the typical responsibilities of a site reliability engineer:

- Proactively monitor and review application performance
- Handle on-call and emergency support
- Ensure software has good logging and diagnostics
- Create and maintain operational runbooks
- Help triage escalated support tickets
- Work on feature requests, defects and other development tasks
- Contribute to overall product roadmap

Perform live site reviews and capture feedback for system outages

The concept of site reliability engineering started in 2003 within Google. As Google continued to grow and scale to become the massive company they are today, they encountered many of their own growing pains. Their challenge was how to support large-scale systems while also introducing new features continuously.

To accomplish the goal, they created a new role that had the dual purpose of developing new features while also ensuring that production systems ran smoothly. Site reliability engineering has grown signifi-

cantly within Google and most projects have site reliability engineers as part of the team. Google now has over 1,500 site reliability engineers.

Site reliability engineering vs DevOps

So, I know what you are thinking ... how does site reliability engineering compare to DevOps?

DevOps is a more recent movement, designed to help organizations' IT department move in agile and performant ways. It builds a healthy working relationship between the Operations staff and Dev team, allowing each to see how their work influences and affects the other. By combining knowledge and effort, DevOps should produce a more robust, reliable, agile product.

Both SRE and DevOps are methodologies addressing organizations' needs for production operation management. But the differences between the two doctrines are quite significant: While DevOps raise problems and dispatch them to Dev to solve, the SRE approach is to find problems and solve some of them themselves. While DevOps teams would usually choose the more conservative approach, leaving the production environment untouched unless absolutely necessary, SREs are more confident in their ability to maintain a stable production environment and push for rapid changes and software updates. Not unlike the DevOps team, SREs also thrive on a stable production environment, but one of the SRE team's goals is to improve performance and operational efficiency.

For other companies like us who were born in the cloud and heavily use PaaS services, I believe they will also see site reliability engineering as the missing element to their development team success. For larger companies or companies who don't use the cloud, I could see them using both DevOps and site reliability engineering. DevOps practices can help ensure IT helps rack, stack, configure, and deploy the servers and applications. The site reliability engineers can then handle the daily operation of the applications. They also work as a fast feedback loop to the entire team about how the application is performing and running in production.

Site reliability engineering skills

The type of skills needed will vary wildly based on your type of application, how and where it is deployed, and how it is monitored. For organizations using Serverless such as Azure PaaS in-depth knowledge of Windows or Linux systems management isn't much of a priority. However, it may be really critical to teams if you are using servers for deployments.

The other key skills for a good site reliability engineer are more focused on application monitoring and diagnostics. You want to hire people who are good problem solvers and have a knack for finding problems. Experience with application performance management tools like App Insights, New Relic, and others would be really valuable. They should be well versed at application logging best practices and exception handling.

The future of site reliability engineering

Software developers are increasingly taking a larger role in deployments, production operations, and application monitoring. The tools available today make it extremely easy to deploy our applications and monitor them. Things like PaaS and application monitoring solutions make it easy for developers to own their projects from ideation all the way to production.

While IT operations will always exist in most medium to large enterprises. Their type of work will continue to change because of the cloud, PaaS, containers, and other technologies.

Analyze telemetry to establish a baseline

When would I get a notification

Application Insights¹ automatically analyzes the performance of your web application, and can warn you about potential problems. You might be reading this because you received one of our smart detection notifications.

This feature requires no special setup, other than configuring your app for Application Insights (on **ASP.NET**², **Java**³, or Node.js, and in **web page code**⁴). It is active when your app generates enough telemetry.

When would I get a smart detection notification?⁵

Application Insights has detected that the performance of your application has degraded in one of these ways:

- Response time degradation - Your app has started responding to requests more slowly than it used to. The change might have been rapid, for example because there was a regression in your latest deployment. Or it might have been gradual, maybe caused by a memory leak.
- Dependency duration degradation - Your app makes calls to a REST API, database, or other dependency. The dependency is responding more slowly than it used to.
- Slow performance pattern - Your app appears to have a performance issue that is affecting only some requests. For example, pages are loading more slowly on one type of browser than others; or requests are being served more slowly from one particular server. Currently, our algorithms look at page load times, request response times, and dependency response times.

Smart Detection requires at least 8 days of telemetry at a workable volume in order to establish a baseline of normal performance. So, after your application has been running for that period, any significant issue will result in a notification.

Does my app definitely have a problem

No, a notification doesn't mean that your app definitely has a problem. It's simply a suggestion about something you might want to look at more closely.

How do I fix it

The notifications include diagnostic information. Here's an example:

¹ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-overview>

² <https://docs.microsoft.com/en-us/azure/azure-monitor/app/asp-net>

³ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-java-get-started>

⁴ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-javascript>

⁵ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-proactive-performance-diagnostics#when-would-i-get-a-smart-detection-notification>

Server response time degradation
fabricampred

Send a smile Send a frown New Work Item View Work Items More

Was this detection helpful? Please send us a smile or a frown

Detection Properties ^

Rule name	Degradation in server response time
When	3/23 2:00 AM - 3/24 1:59 AM
Operation name	GET Home/Index
Detected response time	1.62 sec
Normal response time	0.524 sec

Detection Analysis

Affected users 76

Server response time

Date	Avg Response Time (sec)	90th Percentile (sec)
Mar 16	0.4	1.5
Mar 17	0.4	1.3
Mar 18	0.4	1.6
Mar 19	0.4	1.2
Mar 20	0.4	1.4
Mar 21	0.4	1.7
Mar 22	1.6	2.8

Server requests

Date	Server Requests (K)
Mar 16	1.8
Mar 17	1.8
Mar 18	1.8
Mar 19	1.8
Mar 20	1.8
Mar 21	1.8
Mar 22	2.2

Degradation in related dependency duration

Date	Dependency Duration (ms)
Mar 16	0
Mar 17	0
Mar 18	0
Mar 19	0
Mar 20	0
Mar 21	0
Mar 22	220

Related items and reports ^

- Diagnose example profiler traces
- Diagnose response times (8 days)
- View requests with this operation name (24 hours)
- View failed requests with this operation name

1. Triage. The notification shows you how many users or how many operations are affected. This can help you assign a priority to the problem.
2. Scope. Is the problem affecting all traffic, or just some pages? Is it restricted to particular browsers or locations? This information can be obtained from the notification.
3. Diagnose. Often, the diagnostic information in the notification will suggest the nature of the problem. For example, if response time slows down when request rate is high, that suggests your server or dependencies are overloaded. Otherwise, open the Performance blade in Application Insights. There, you will find **Profiler**⁶ data. If exceptions are thrown, you can also try the **snapshot debugger**⁷.

⁶ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-profiler>

⁷ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-snapshot-debugger>

Configure Email Notifications

- Smart Detection notifications are enabled by default and sent to those who have **owners, contributors and readers access to the Application Insights resource⁸**. To change this, either click Configure in the email notification, or open Smart Detection settings in Application Insights.

NAME	SEVERITY
Slow page load time	Information
Slow server response time	Information
Long dependency duration	Information
Degradation in server response time	Information
Azure cloud service issues	Information
Degradation in dependency duration	Information
Failure Anomalies - fabrikamprod	Alert

- You can use the unsubscribe link in the Smart Detection email to stop receiving the email notifications.

Emails about Smart Detections performance anomalies are limited to one email per day per Application Insights resource. The email will be sent only if there is at least one new issue that was detected on that day. You won't get repeats of any message.

How can I improve performance?

- Slow and failed responses are one of the biggest frustrations for web site users, as you know from your own experience. So, it's important to address the issues.

Triage⁹

- First, does it matter? If a page is always slow to load, but only 1% of your site's users ever have to look at it, maybe you have more important things to think about. On the other hand, if only 1% of users open it, but it throws exceptions every time, that might be worth investigating. Use the impact statement (affected users or % of traffic) as a general guide, but be aware that it isn't the whole story. Gather other evidence to confirm. Consider the parameters of the issue. If it's geography-dependent, set up **availability tests¹⁰** including that region: there might simply be network issues in that area.

Diagnose slow page loads¹¹

- Where is the problem? Is the server slow to respond, is the page very long, or does the browser have to do a lot of work to display it? Open the Browsers metric blade. The segmented display of browser

⁸ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-resources-roles-access-control>

⁹ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-proactive-performance-diagnostics#triage>

¹⁰ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-monitor-web-app-availability>

¹¹ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-proactive-performance-diagnostics#diagnose-slow-page-loads>

page load time shows where the time is going.

- If Send Request Time is high, either the server is responding slowly, or the request is a post with a lot of data. Look at the **performance metrics**¹² to investigate response times.
- Set up **dependency tracking**¹³ to see whether the slowness is due to external services or your database.
- If Receiving Response is predominant, your page and its dependent parts - JavaScript, CSS, images and so on (but not asynchronously loaded data) are long. Set up an **availability test**¹⁴, and be sure to set the option to load dependent parts. When you get some results, open the detail of a result and expand it to see the load times of different files.
- High Client Processing time suggests scripts are running slowly. If the reason isn't obvious, consider adding some timing code and send the times in trackMetric calls.

Improve slow pages¹⁵

- There's a web full of advice on improving your server responses and page load times, so we won't try to repeat it all here. Here are a few tips that you probably already know about, just to get you thinking:
 - Slow loading because of big files: Load the scripts and other parts asynchronously. Use script bundling. Break the main page into widgets that load their data separately. Don't send plain old HTML for long tables: use a script to request the data as JSON or other compact format, then fill the table in place. There are great frameworks to help with all this. (They also entail big scripts, of course.)
 - Slow server dependencies: Consider the geographical locations of your components. For example, if you're using Azure, make sure the web server and the database are in the same region. Do queries retrieve more information than they need? Would caching or batching help?
 - Capacity issues: Look at the server metrics of response times and request counts. If response times peak disproportionately with peaks in request counts, it's likely that your servers are stretched.

Server Response Time Degradation

- The response time degradation notification tells you:
 - The response time compared to normal response time for this operation.
 - How many users are affected.
 - Average response time and 90th percentile response time for this operation on the day of the detection and 7 days before.
 - Count of this operation requests on the day of the detection and 7 days before.
 - Correlation between degradation in this operation and degradations in related dependencies.

¹² <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-web-monitor-performance#metrics>

¹³ <https://docs.microsoft.com/en-us/azure/azure-monitor/app/asp-net-dependencies>

¹⁴ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-monitor-web-app-availability>

¹⁵ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-proactive-performance-diagnostics#improve-slow-pages>

- Links to help you diagnose the problem.
 - Profiler traces to help you view where operation time is spent (the link is available if Profiler trace examples were collected for this operation during the detection period).
 - Performance reports in Metric Explorer, where you can slice and dice time range/filters for this operation.
 - Search for this call to view specific call properties.
 - Failure reports - If count > 1 this mean that there were failures in this operation that might have contributed to performance degradation.

Dependency Duration Degradation

- Modern application more and more adopt micro services design approach, which in many cases leads to heavy reliability on external services. For example, if your application relies on some data platform or even if you build your own bot service you will probably rely on some cognitive services provider to enable your bots to interact in more human ways and some data store service for bot to pull the answers from. Example dependency degradation notification:

Dependency duration degradation
DepInsightsDataMonitoring_PROD

Send a smile Send a frown + New Work Item View Work Items More

Was this detection helpful? Please send us a smile or a frown

Detection Properties

Rule name	Degradation in dependency duration
When	4.9.03:00 - 12.9.03:00
Base name	SyntheticTsaMonitor_1
Dependency name	SQL: SyntheticTsaMonitor_1
Dependency type	SQL
Detected duration	1.88 sec
Normal duration	367 ms

Detection Analysis

Affected users 4

Dependency duration

Dependency calls

Related items and reports

- View Dependency duration reports
- View Dependency properties
- View Dependency failures reports
- View queries in analytics portal

Notice that it tells you:

- The duration compared to normal response time for this operation
- How many users are affected
- Average duration and 90th percentile duration for this dependency on the day of the detection and 7 days before
- Number of dependency calls on the day of the detection and 7 days before
- Links to help you diagnose the problem
 - Performance reports in Metric Explorer for this dependency
 - Search for this dependency calls to view calls properties
 - Failure reports - If count > 1 this means that there were failed dependency calls during the detection period that might have contributed to duration degradation.
 - Open Analytics with queries that calculate this dependency duration and count

Smart Detection of slow performing patterns

Application Insights finds performance issues that might only affect some portion of your users, or only affect users in some cases. For example, notification about pages load is slower on one type of browser than on other types of browsers, or if requests are served more slowly from a particular server. It can also discover problems associated with combinations of properties, such as slow page loads in one geographical area for clients using particular operating system.

Anomalies like these are very hard to detect just by inspecting the data, but are more common than you might think. Often they only surface when your customers complain. By that time, it's too late: the affected users are already switching to your competitors!

Currently, our algorithms look at page load times, request response times at the server, and dependency response times.

You don't have to set any thresholds or configure rules. Machine learning and data mining algorithms are used to detect abnormal patterns.

Don't forget, page load times are critical to success for web applications. Slow pages are frustrating for your users.

[Check out the page view performance of fabrikamprod in Application Insights](#)

[Learn how to monitor web page performance](#)

The screenshot shows a Smart detection report from Application Insights. It includes a header with 'Smart detection' and 'Last 24 hours - VillageHall'. Below it are 'Settings', 'Time range', 'Refresh', and 'Help' buttons. The main content area has 'When:' set to '11/11 4:28 AM to 9:12 AM' and 'What:' describing 'Slow request response performance was identified for requests with these parameters: Request name: GET Home/Index'. A table compares 'Slower requests' (1.78sec) with 'All other requests' (96.7ms). A red arrow points from the top text about page load times to the 'Slow request response performance' section in the report.

GROUPING	AVERAGE TIME
Slower requests	1.78sec
All other requests	96.7ms

- When shows the time the issue was detected.
- What describes:
 - The problem that was detected;
 - The characteristics of the set of events that we found displayed the problem behavior.
- The table compares the poorly-performing set with the average behavior of all other events.

Click the links to open Metric Explorer and Search on relevant reports, filtered on the time and properties of the slow performing set.

Modify the time range and filters to explore the telemetry.

For more information, view the video **Application Performance Management with Azure Application Insights**¹⁶.

¹⁶ <https://channel9.msdn.com/Events/Connect/2016/112/player>

Perform ongoing tuning to reduce meaningless or non-actionable alerts

Monitoring and alerting

Monitoring and alerting enables a system to tell us when it's broken, or perhaps to tell us what's about to break. When the system isn't able to automatically fix itself, we want a human to investigate the alert, determine if there's a real problem at hand, mitigate the problem, and determine the root cause of the problem. Unless you're performing security auditing on very narrowly scoped components of a system, you should never trigger an alert simply because "something seems a bit weird." When you are reviewing existing alerts or writing new alerting rules, consider these things to keep your alerts relevant and your on-call rotation happier:

- Alerts that trigger call-out should be urgent, important, actionable, and real.
- They should represent either ongoing or imminent problems with your service.
- Err on the side of removing noisy alerts – over-monitoring is a harder problem to solve than under-monitoring.
- You should almost always be able to classify the problem into one of: availability & basic functionality; latency; correctness (completeness, freshness and durability of data); and feature-specific problems.
- Symptoms are a better way to capture more problems more comprehensively and robustly with less effort.
- Include cause-based information in symptom-based pages or on dashboards, but avoid alerting directly on causes.
- The further up your serving stack you go, the more distinct problems you catch in a single rule. But don't go so far you can't sufficiently distinguish what's going on.
- If you want a quiet oncall rotation, it's imperative to have a system for dealing with things that need timely response, but are not imminently critical.

Monitor for your users

I call this "symptom-based monitoring," in contrast to "cause-based monitoring". Do your users care if your MySQL servers are down? No, they care if their queries are failing. Do your users care if a support (i.e. non-serving-path) binary is in a restart-loop? No, they care if their features are failing. Do they care if your data push is failing? No, they care about whether their results are fresh.

Users, in general, care about a small number of things:

- Basic availability and correctness. no "Oops!", no 500s, no non-responding requests or half-loaded pages or missing Javascript or CSS or images or videos. - Anything that breaks the core service in some way should be considered unavailability.
- Latency. Fast. Fast. Fast. Also, fast.
- Completeness/freshness/durability. Your users data should be safe, should come back when you ask, and search indices should be up-to-date.
- Even if it is temporarily unavailable, users should have complete faith that it's coming back.
- Features. Your users care that all the features of the service work—you should be monitoring for anything that is an important aspect of your service even if it's not core functionality/availability.

That's pretty much it. There's a subtle but important difference between database servers being unavailable and user data being unavailable. The former is a proximate cause, the latter is a symptom. You can't always cleanly distinguish these things, particularly when you don't have a way to mimic the client's perspective (e.g. a blackbox probe or monitoring their perspective directly). But when you can, you should.

Case based alerts

Cause-based alerts are bad (but sometimes necessary).

"But," you might say, "I know database servers that are unreachable results in user data unavailability." That's fine. Alert on the data unavailability. Alert on the symptom: the 500, the Oops!, the whitebox metric that indicates that not all servers were reached from the database's client. Why?

- You're going to have to catch the symptom anyway. Maybe it can happen because of network disconnection, or CPU contention, or myriad other problems you haven't thought of yet. So you have to catch the symptom.
- Once you catch the symptom and the cause, you have redundant alerts; these need separate tuning, and result in either duplication or complicated dependency trees
- The allegedly inevitable result is not always inevitable: maybe your database servers are unavailable because you're turning up a new instance or turning down an old one. Or maybe a feature was added to do fast-failover of requests, and so you don't care anymore about a single server's availability. Sure, you can catch all these cases with increasingly complicated rules, but why bother? The failure mode is more bogus pages, more confusion, and more tuning, with no gain, and less time spent on fixing the alerts that matter.

But sometimes they're necessary. There's (often) no symptoms to "almost" running out of quota or memory or disk I/O, etc., so you want rules to know you're walking towards a cliff. Use these sparingly; don't write cause-based rules that trigger on call alerts for symptoms you can catch otherwise.

Tickets alerts and email

Tickets, Reports and Email

One way or another, you have some alerts that need attention soon, but not right now. You can refer to these as "sub-critical alerts".

- Bug or ticket-tracking systems can be useful. Having alerts open a bug can work out great, as long as multiple firings of the same alert get correctly threaded into a single ticket/bug. This system fails if there's no accountability for triaging and closing bugs; if the alert-opened bugs might go unseen for weeks, this clearly fails as a way of dealing with sub-critical alerts before they become critical! It also fails if your team is simply overloaded or is not assigning enough people to deal with followup; you need to be honest about how much time this is consuming, or you'll fall further and further behind.
- A daily (or more frequent) report can work too. One way this can work is to write sub-critical rules that are long-lived (e.g. "the database is over 90% full" or "we've served over 1000 very slow requests in the last day"), and send out a report periodically that shows all currently-firing rules. Again, without a system of accountability this amounts to less-spammy email alerts, so make sure the oncall person (or someone else) is designated to triage these every day (or every shift hand-off, or whatever works).
- Every alert should be tracked through a workflow system. Don't only dump them into an email list or IRC channel. In general, this quickly turns into specialized "foo-alerts" mailing lists or channels so that they can be summarily ignored. Except as a brief (usually days, at most weeks) period to vet that a

new rule will not page too often, it's almost always a bad idea. It's also easy to ignore the volume of these alerts, and suddenly some old, mis-tuned rule is firing every minute for all of your thousand application servers, clogging up mailboxes. Oops.

The underlying point is to create a system that still has accountability for responsiveness, but doesn't have the high cost of waking someone up, interrupting their dinner, or preventing snuggling with a significant other.

Playbooks

Playbooks (or runbooks) are an important part of an alerting system; it's best to have an entry for each alert or family of alerts that catch a symptom, which can further explain what the alert means and how it might be addressed.

In general, if your playbook has a long detailed flow chart, you're potentially spending too much time documenting what could be wrong and too little time fixing it—unless the root causes are completely out of your control or fundamentally require human intervention (like calling a vendor). The best playbooks I've seen have a few notes about exactly what the alert means, and what's currently interesting about an alert ("We've had a spate of power outages from our widgets from VendorX; if you find this, please add it to Bug 12345 where we're tracking things for patterns".) Most such notes should be ephemeral, so a wiki or similar is a great tool.

Tracking & Accountability

Track your on-call alerts, and all your other alerts. If an on-call is firing and people just say "I looked, nothing was wrong", that's a pretty strong sign that you need to remove the rule, or demote it or collect data in some other way. Alerts that are less than 50% accurate are broken; even those that are false positives 10% of the time merit more consideration.

Having a system in place (e.g. a weekly review of all triggered on-call alerts, and quarterly statistics) can help keep a handle on the big picture of what's going on, and tease out patterns that are lost when the pager is handed from one human to the next.

When to seek the exception from the rule

Here are some great reasons to break the above guidelines:

- You have a known cause that actually sits below the noise in your symptoms. For example, if your service has 99.99% availability, but you have a common event that causes 0.001% of requests to fail, you can't alert on it as a symptom (because it's in the noise) but you can catch the causing event. It might be worth trying to trickle this information up the stack, but maybe it really is simplest just to alert on the cause.
- You can't monitor at the spout, because you lose data resolution. For example, maybe you tolerate some handlers/endpoints/backends/URLs being pretty slow (like a credit card validation compared to browsing items for sale) or low-availability (like a background refresh of an inbox). At your load balancers, this distinction may be lost. Walk down the stack and alert from the highest place where you have the distinction.
- Your symptoms don't appear until it's too late, like you've run out of quota. Of course, you need to page before it's too late, and sometimes that means finding a cause to page on (e.g. usage > 80% and will run out in < 4h at the growth rate of the last 1h). But if you can do that, you should also be able to find a similar cause that's less urgent (e.g. quota > 90% and will run out in < 4d at the growth rate

of the last 1d) that will catch most cases, and deal with that as a ticket or email alert or daily problem report, rather than the last-ditch escalation that a page represents.

- Your alert setup sound more complex than the problems they're trying to detect. Sometimes they will be. The goal should be to tend towards simplicity, robust, self-protecting systems (how did you not notice that you were running out of quota? Why can't that data go somewhere else?) In the long term, they should trend towards simplicity, but at any given time the local optimum may be relatively complex rules to keep things quiet and accurate.

Analyze alerts to establish a baseline

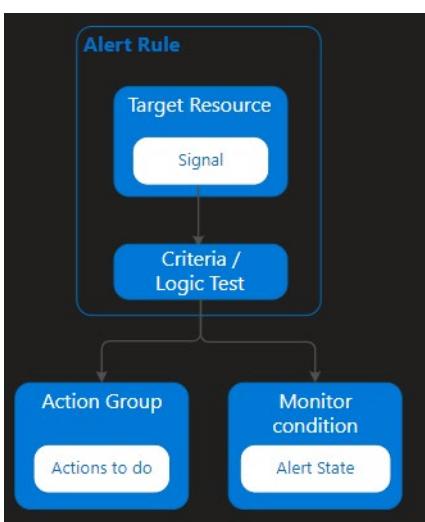
What are alerts in Microsoft Azure

Alerts proactively notify you when important conditions are found in your monitoring data. They allow you to identify and address issues before the users of your system notice them.

This article discusses the unified alert experience in Azure Monitor, which now includes Log Analytics and Application Insights. The **previous alert experience**¹⁷ and alert types are called classic alerts. You can view this older experience and older alert type by clicking on View classic alerts at the top of the alert page.

Overview¹⁸

The diagram below represents the flow of alerts.



Alert rules are separated from alerts and the action that are taken when an alert fires.

Alert rule - The alert rule captures the target and criteria for alerting. The alert rule can be in an enabled or a disabled state. Alerts only fire when enabled.

The key attributes of an alert rule are:

- Target Resource - Defines the scope and signals available for alerting. A target can be any Azure resource. Example targets: a virtual machine, a storage account, a virtual machine scale set, a Log Analytics workspace, or an Application Insights resource. For certain resources (like Virtual Machines), you can specify multiple resources as the target of the alert rule.
- Signal - Signals are emitted by the target resource and can be of several types. Metric, Activity log, Application Insights, and Log.
- Criteria - Criteria is combination of Signal and Logic applied on a Target resource. Examples:
 - Percentage CPU > 70%
 - Server Response Time > 4 ms

¹⁷ <https://docs.microsoft.com/en-us/azure/azure-monitor/platform/alerts-overview>

¹⁸ <https://docs.microsoft.com/en-us/azure/azure-monitor/platform/alerts-overview#overview>

- Result count of a log query > 100
- Alert Name – A specific name for the alert rule configured by the user
- Alert Description – A description for the alert rule configured by the user
- Severity – The severity of the alert once the criteria specified in the alert rule is met. Severity can range from 0 to 4.
- Action - A specific action taken when the alert is fired. For more information, see **Action Groups¹⁹**.

What you can alert on

You can alert on metrics and logs as described in **Sources of Monitoring Data for Azure Monitor²⁰**.

These include but are not limited to:

- Metric values
- Log search queries
- Activity Log events
- Health of the underlying Azure platform
- Tests for web site availability

Manage alerts

You can set the state of an alert to specify where it is in the resolution process. When the criteria specified in the alert rule is met, an alert is created or fired, it has a status of *New*. You can change the status when you acknowledge an alert and when you close it. All state changes are stored in the history of the alert.

The following alert states are supported.

State	Description
New	The issue has just been detected and has not yet been reviewed.
Acknowledged	An administrator has reviewed the alert and started working on it.
Closed	The issue has been resolved. After an alert has been closed, you can reopen it by changing it to another state.

Alert state is different and independent of the monitor condition. Alert state is set by the user. Monitor condition is set by the system. When an alert fires, the alert's monitor condition is set to *fired*. When the underlying condition that caused the alert to fire clears, the monitor condition is set to *resolved*. The alert state isn't changed until the user changes it. Learn **how to change the state of your alerts and smart groups²¹**.

¹⁹ <https://docs.microsoft.com/en-us/azure/azure-monitor/platform/action-groups>

²⁰ <https://docs.microsoft.com/en-us/azure/azure-monitor/platform/data-sources>

²¹ <https://aka.ms/managing-alert-smart-group-states>

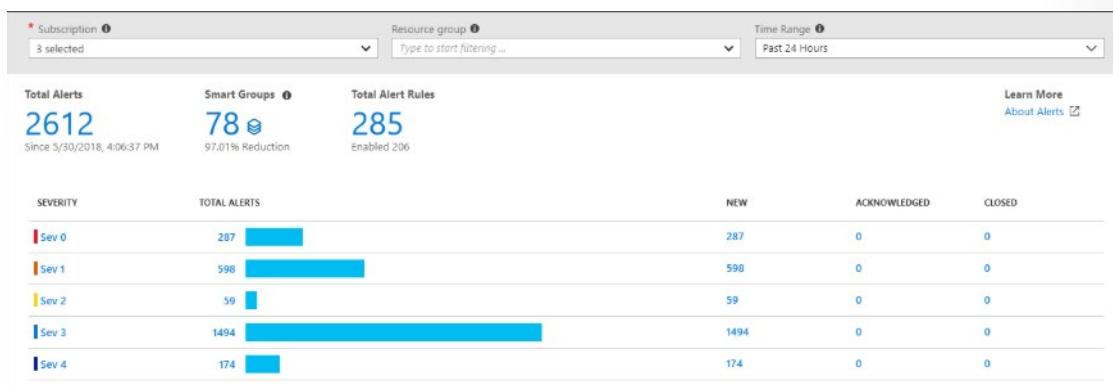
Smart groups

Smart groups are aggregations of alerts based on machine learning algorithms, which can help reduce alert noise and aid in trouble-shooting. [Learn more about Smart Groups²²](#) and [how to manage your smart groups²³](#).

Alerts experience

The default Alerts page provides a summary of alerts that are created within a particular time window. It displays the total alerts for each severity with columns that identify the total number of alerts in each state for each severity. Select any of the severities to open the [All Alerts²⁴](#) page filtered by that severity.

It does not show or track older [classic alerts²⁵](#). You can change the subscriptions or filter parameters to update the page.



You can filter this view by selecting values in the dropdown menus at the top of the page.

Column	Description
Subscription	Select up to five Azure subscriptions. Only alerts in the selected subscriptions are included in the view.
Resource group	Select a single resource group. Only alerts with targets in the selected resource group are included in the view.
Time range	Only alerts fired within the selected time window are included in the view. Supported values are the past hour, the past 24 hours, the past 7 days, and the past 30 days.

Select the following values at the top of the Alerts page to open another page.

Value	Description
Total alerts	The total number of alerts that match the selected criteria. Select this value to open the All Alerts view with no filter.

²² <https://aka.ms/smart-groups>

²³ <https://aka.ms/managing-smart-groups>

²⁴ <https://docs.microsoft.com/en-us/azure/azure-monitor/platform/alerts-overview#all-alerts-page>

²⁵ <https://docs.microsoft.com/en-us/azure/azure-monitor/platform/alerts-overview#classic-alerts>

Value	Description
Smart groups	The total number of smart groups that were created from the alerts that match the selected criteria. Select this value to open the smart groups list in the All Alerts view.
Total alert rules	The total number of alert rules in the selected subscription and resource group. Select this value to open the Rules view filtered on the selected subscription and resource group.

Manage alert rules

Click on Manage alert rules to show the Rules page. Rules is a single place for managing all alert rules across your Azure subscriptions. It lists all alert rules and can be sorted based on target resources, resource groups, rule name, or status. Alert rules can also be edited, enabled, or disabled from this page.

NAME	CONDITION	STATUS	TARGET RESOURCE
VM-CPU-RG1	Percentage CPU GreaterThanOrEqual 0	Enabled	k8s-master-10625516-0
<input checked="" type="checkbox"/> VM-Disk-RG2	Percentage CPU GreaterThan 22 and Disk Read Operations/Sec GreaterThan 333	Disabled	GraphanaTest
<input checked="" type="checkbox"/> NetworkOut-GreaterThan3333	Network Out GreaterThan 3333	Enabled	GraphanaTest
VM-CPU-17-RG1	Percentage CPU GreaterThan 17	Enabled	GraphanaTest

Create an alert rule

Alerts can be authored in a consistent manner regardless of the monitoring service or signal type. All fired alerts and related details are available in single page.

You create a new alert rule with the following three steps:

1. Pick the *target* for the alert.
2. Select the *signal* from the available signals for the target.
3. Specify the *logic* to be applied to data from the signal.

This simplified authoring process no longer requires you to know the monitoring source or signals that are supported before selecting an Azure resource. The list of available signals is automatically filtered based on the target resource that you select. Also based on that target, you are guided through defining the logic of the alert rule automatically.

You can learn more about how to create alert rules in **Create, view, and manage alerts using Azure Monitor²⁶**.

²⁶ <https://docs.microsoft.com/en-us/azure/azure-monitor/platform/alerts-metric>

Alerts are available across several Azure monitoring services. For information about how and when to use each of these services, see [Monitoring Azure applications and resources²⁷](#). The following table provides a listing of the types of alert rules that are available across Azure. It also lists what's currently supported in which alert experience.

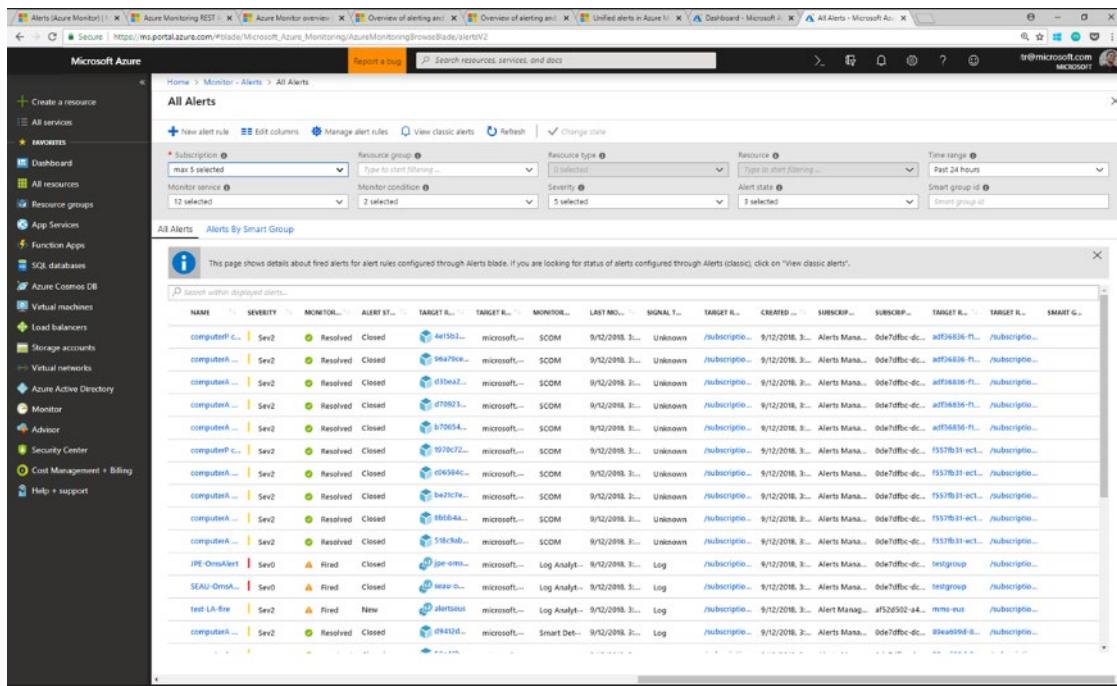
Previously, Azure Monitor, Application Insights, Log Analytics, and Service Health had separate alerting capabilities. Overtime, Azure improved and combined both the user interface and different methods of alerting. This consolidation is still in process. As a result, there are still some alerting capabilities not yet in the new alerts system.

Monitor source	Signal type	Description
Service health	Activity log	Not supported. See Create activity log alerts on service notifications (https://docs.microsoft.com/en-us/azure/azure-monitor/platform/alerts-activity-log-service-notifications).
Application Insights	Web availability tests	Not supported. See Web test alerts (https://docs.microsoft.com/en-us/azure/application-insights/app-insights-monitor-web-app-availability). Available to any website that's instrumented to send data to Application Insights. Receive a notification when availability or responsiveness of a website is below expectations.

All alerts page

Click on Total Alerts to see the all alerts page. Here you can view a list of alerts that were created within the selected time window. You can view either a list of the individual alerts or a list of the smart groups that contain the alerts. Select the banner at the top of the page to toggle between views.

²⁷ <https://docs.microsoft.com/en-us/azure/azure-monitor/overview>



You can filter the view by selecting the following values in the dropdown menus at the top of the page.

Column	Description
Subscription	Select up to five Azure subscriptions. Only alerts in the selected subscriptions are included in the view.
Resource group	Select a single resource group. Only alerts with targets in the selected resource group are included in the view.
Resource type	Select one or more resource types. Only alerts with targets of the selected type are included in the view. This column is only available after a resource group has been specified.
Resource	Select a resource. Only alerts with that resource as a target are included in the view. This column is only available after a resource type has been specified.
Severity	Select an alert severity, or select All to include alerts of all severities.
Monitor condition	Select a monitor condition, or select All to include alerts of conditions.
Alert state	Select an alert state, or select All to include alerts of states.
Monitor service	Select a service, or select All to include all services. Only alerts created by rules that use service as a target are included.

Column	Description
Time range	Only alerts fired within the selected time window are included in the view. Supported values are the past hour, the past 24 hours, the past 7 days, and the past 30 days.

Select Columns at the top of the page to select which columns to display.

Blameless Retrospectives and a Just Culture

A Blameless Retrospective

Anyone who's worked with technology at any scale is familiar with failure. Failure cares not about the architecture designs you drudge over, the code you write and review, or the alerts and metrics you meticulously pore through. So: failure happens. This is a foregone conclusion when working with complex systems. But what about those failures that have resulted due to the actions (or lack of action, in some cases) of individuals? What do you do with those careless humans who caused everyone to have a bad day?

Maybe they should be fired. Or maybe they need to be prevented from touching the dangerous bits again. Or maybe they need more training.

This is the traditional view of "human error", which focuses on the characteristics of the individuals involved. This is called the "Bad Apple Theory" – get rid of the bad apples, and you'll get rid of the human error. Seems simple, right? Organizations that have pioneered DevOps are shying away from this traditional view. These DevOps practising organizations instead want to view mistakes, errors, slips, lapses, etc. with a perspective of *learning*. Having blameless Post-Mortems on outages and accidents are part of that.

What does it mean to have a 'blameless' retrospective?

Does it mean everyone gets off the hook for making mistakes? No.

Well, maybe. It depends on what "gets off the hook" means. Let me explain.

Having a **Just Culture** means that you're making effort to balance safety **and** accountability. It means that by investigating mistakes in a way that focuses on the situational aspects of a failure's mechanism and the decision-making process of individuals proximate to the failure, an organization can come out safer than it would normally be if it had simply punished the actors involved as a remediation.

Having a "blameless" retrospective process means that engineers whose actions have contributed to an accident can give a detailed account of:

- what actions they took at what time,
- what effects they observed,
- expectations they had,
- assumptions they had made,
- and their understanding of timeline of events as they occurred.

...and that they can give this detailed account **without fear of punishment or retribution**.

Why shouldn't they be punished or reprimanded? Because an engineer who thinks they're going to be reprimanded are *disincentivized* to give the details necessary to get an understanding of the mechanism, pathology, and operation of the failure. This lack of understanding of how the accident occurred all but guarantees that it **will** repeat. If not with the original engineer, another one in the future.

If we go with "blame" as the predominant approach, then we're implicitly accepting that *deterrence* is how organizations become safer. This is founded in the belief that individuals, not situations, cause errors. It's also aligned with the idea there has to be some fear that **not** doing one's job correctly could lead to punishment. Because the fear of punishment will motivate people to act correctly in the future. Right?

This cycle of name/blame/shame can be looked at like this:

1. Engineer takes action and contributes to a failure or incident.

2. Engineer is punished, shamed, blamed, or retrained.
3. Reduced trust between engineers on the ground (the "sharp end") and management (the "blunt end") looking for someone to scapegoat
4. Engineers become silent on details about actions/situations/observations, resulting in "Cover-Your-Mistake" engineering (from fear of punishment)
5. Management becomes less aware and informed on how work is being performed day to day, and engineers become less educated on lurking or latent conditions for failure due to silence mentioned in #4, above
6. Errors more likely, latent conditions can't be identified due to #5, above
7. Repeat from step 1

We need to avoid this cycle. We want the engineer who has made an error give details about why (either explicitly or implicitly) he or she did what they did; why the action made sense to them at the time. This is paramount to understanding the pathology of the failure. The action made sense to the person at the time they took it, because if it hadn't made sense to them at the time, they **wouldn't have taken the action in the first place.**

The base fundamental here is something **Erik Hollnagel²⁸** has said:

We must strive to understand that accidents don't happen because people gamble and lose.

Accidents happen because the person believes that:

...what is about to happen is not possible,

...or what is about to happen has no connection to what they are doing,

...or that the possibility of getting the intended outcome is well worth whatever risk there is.

Allowing Engineers to Own Their Own Stories

A funny thing happens when engineers make mistakes and feel safe when giving details about it: they are not only willing to be held accountable, they are also enthusiastic in helping the rest of the company avoid the same error in the future. They are, after all, the most expert in their own error. They ought to be heavily involved in coming up with remediation items.

So technically, engineers are not at all "off the hook" with a blameless PostMortem process. They are very much on the hook for helping become safer and more resilient, in the end. And lo and behold: most engineers I know find this idea of making things better for others a worthwhile exercise.

So what do we do to enable a "Just Culture"?

- Encourage learning by having these blameless Post-Mortems on outages and accidents.
- The goal is to understand **how **an accident could have happened, in order to better equip ourselves from it happening in the future
- Gather details from multiple perspectives on failures, and don't punish people for making mistakes.
- Instead of punishing engineers, we instead give them the requisite authority to improve safety by allowing them to give detailed accounts of their contributions to failures.
- Enable and encourage people who *do* make mistakes to be the experts on educating the rest of the organization how not to make them in the future.
- Accept that there is always a discretionary space where humans can decide to make actions or not, and that the judgement of those decisions lie in hindsight.

²⁸ <http://www.erikhollnagel.com/>

- Accept that the **Hindsight Bias²⁹** will continue to cloud our assessment of past events, and work hard to eliminate it.
- Accept that the **Fundamental Attribution Error³⁰** is also difficult to escape, so we focus on the environment and circumstances people are working in when investigating accidents.
- Strive to make sure that the blunt end of the organization understands how work is actually getting done (as opposed to how they imagine it's getting done, via Gantt charts and procedures) on the sharp end.
- The sharp end is relied upon to inform the organization where the line is between appropriate and inappropriate behavior. This isn't something that the blunt end can come up with on its own.

Failure happens. In order to understand how failures happen, we first have to understand our ***reactions*** to failure.

One option is to assume the single cause is incompetence and scream at engineers to make them "pay attention!" or "be more careful!"

Another option is to take a hard look at how the accident actually happened, treat the engineers involved with respect, and *learn* from the event.

For more information, see also:

- **Brian Harry's Blog - A good incident postmortem³¹**

²⁹ <http://en.wikipedia.org/wiki/Hindsight>

³⁰ http://en.wikipedia.org/wiki/Fundamental_attribution_error

³¹ <https://blogs.msdn.microsoft.com/bharry/2018/03/02/a-good-incident-postmortem/>

Module Review and Takeaways

Module Review Questions

Multiple choice

True or False: Application Insights analyses the traffic from your website against historic trends and sends you smart detection notifications on degradation?

- True
- False

Answers

Multiple choice

True or False: Application Insights analyses the traffic from your website against historic trends and sends you smart detection notifications on degradation?

True

False

