

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	11
	Module overview	11
	Transformation planning	12
	Project selection	24
	Team structures	28
	Migrating to DevOps	34
	Lab	38
	Module review and takeaways	39
■	Module 2 Getting Started with Source Control	41
	Module overview	41
	What is source control?	42
	Benefits of source control	44
	Types of source control systems	46
	Introduction to Azure Repos	59
	Introduction to GitHub	60
	Migrating from Team Foundation Version Control (TFVC) to Git in Azure Repos	63
	Lab	65
	Module review and takeaways	66
■	Module 3 Managing technical debt	69
	Module overview	69
	Identifying technical debt	70
	Knowledge sharing within teams	77
	Modernizing development environments with GitHub Codespaces	79
	Lab	80
	Module review and takeaways	81
■	Module 4 Working with Git for Enterprise DevOps	83
	Module overview	83
	How to structure your Git Repo	84
	Git branching workflows	86
	Collaborating with pull requests in Azure Repos	104
	Why care about Git hooks?	111

Fostering inner source	116
Managing Git Repositories	123
Lab	125
Module review and takeaways	126
Module 5 Configuring Azure Pipelines	129
Module overview	129
The concept of pipelines in DevOps	130
Azure Pipelines	132
Evaluate use of Microsoft-hosted versus self-hosted agents	136
Agent pools	138
Pipelines and concurrency	141
Azure DevOps and open-source projects (public projects)	144
Azure Pipelines YAML versus Visual Designer	146
Lab	148
Module review and takeaways	149
Module 6 Implementing Continuous Integration with Azure Pipelines	151
Module overview	151
Continuous integration overview	152
Implementing a build strategy	157
Integration with Azure Pipelines	161
Integrating external source control with Azure Pipelines	177
Set up self-hosted agents	178
Labs	182
Module review and takeaways	184
Module 7 Managing Application Configuration and Secrets	187
Module overview	187
Introduction to security	188
Implement a secure development process	192
Rethinking application configuration data	198
Manage secrets, tokens, and certificates	201
Integrating with identity management systems	204
Implementing application configuration	207
Lab	212
Module review and takeaways	213
Module 8 Implementing Continuous Integration with GitHub Actions	215
Module overview	215
Github Actions	216
Continuous integration with GitHub Actions	223
Securing secrets for GitHub Actions	228
Lab	230
Module review and takeaways	231
Module 9 Designing and Implementing a Dependency Management Strategy	233
Module overview	233
Packaging dependencies	234
Package management	237
Migrating and consolidating artifacts	246
Package security	248
Implement a versioning strategy	252
Lab	258

Module Review and Takeaways	259
Module 10 Designing a Release Strategy	261
Module overview	261
Introduction to continuous delivery	263
Release strategy recommendations	268
Building a high-quality release pipeline	300
Choosing the right release management tool	304
Labs	311
Module review and takeaways	313
Module 11 Implementing Continuous Deployment using Azure Pipelines	315
Module overview	315
Create a release pipeline	317
Provision and configure environments	324
Manage and modularize tasks and templates	334
Configure automated integration and functional test automation	346
Automate inspection of health	349
Labs	360
Module review and takeaways	362
Module 12 Implementing an Appropriate Deployment Pattern	365
Module overview	365
Introduction to deployment patterns	366
Implement blue-green deployment	369
Feature toggles	382
Canary releases	386
Dark launching	388
A/B testing	389
Progressive exposure deployment	390
Lab	396
Module review and takeaways	397
Module 13 Managing Infrastructure and Configuration using Azure Tools	399
Module overview	399
Infrastructure as code and configuration management	400
Create Azure resources using ARM templates	405
Create Azure resources by using Azure CLI	415
Azure Automation with DevOps	419
Desired State Configuration (DSC)	440
Lab	453
Module review and takeaways	454
Module 14 Using Third Party Infrastructure as Code Tools Available with Azure	459
Module overview	459
Chef	460
Puppet	465
Ansible	468
Terraform	484
Labs	506
Module review and takeaways	508
Module 15 Managing Containers using Docker	515
Module overview	515

Implementing a container build strategy	516
Implementing Docker multi-stage builds	526
Lab	532
Module review and takeaways	534
Module 16 Creating and Managing Kubernetes Service Infrastructure	537
Module overview	537
Azure Kubernetes Service (AKS)	539
Kubernetes tooling	551
Integrating AKS with Pipelines	553
Lab	558
Module review and takeaways	559
Module 17 Implementing Feedback for Development Teams	561
Module overview	561
Implement tools to track system usage, feature usage, and flow	563
Implement routing for mobile application crash report data	592
Develop monitoring and status dashboards	597
Integrate and configure ticketing systems	603
Lab	611
Module Review and Takeaways	612
Module 18 Implementing System Feedback Mechanisms	615
Module overview	615
Site reliability engineering	617
Design practices to measure end-user satisfaction	619
Design processes to capture and analyze user feedback	625
Design processes to automate application analytics	636
Managing alerts	640
Blameless retrospectives and a just culture	645
Lab	648
Module review and takeaways	649
Module 19 Implementing Security in DevOps Projects	651
Module overview	651
Security in the Pipeline	653
Azure Security Center	667
Lab	677
Module review and takeaways	678
Module 20 Validating Code Bases for Compliance	685
Module overview	685
Open-source software	686
Managing security and compliance policies	690
Integrating license and vulnerability scans	693
Lab	698
Module review and takeaways	699

Module 0 Welcome

Start here

Microsoft DevOps curriculum

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

¹ <https://docs.microsoft.com/en-us/learn/certifications/exams/AZ-400>

AZ-400 Study Areas	Weights
Define and implement a continuous delivery and release management strategy	10-15%

About this course

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 designing and implementing DevOps processes or in passing the Microsoft Azure DevOps Solutions certification exam.

Pre-requisites

Successful learners will have prior knowledge and understanding of:

- Cloud computing concepts, including an understanding of PaaS, SaaS, and IaaS implementations.
- Both Azure administration and Azure development with proven expertise in at least one of these areas.
- Version control, Agile software development, and core software development principles. It would be helpful to have experience in an organization that delivers software.

If you are new to Azure and cloud computing, consider one of the following resources:

- Free online: **Azure Fundamentals**²
- Instructor-led course: **AZ-900: Azure Fundamentals**³

If you are new to Azure Administration, consider taking:

- Free online: **Prerequisites for Azure Administrators**⁴
- Instructor-led courses: **AZ-104: Microsoft Azure Administrator**⁵ and **AZ-010: Azure Administration for AWS SysOps**⁶

If you are new to Azure Developer, consider taking:

- Free online: **Create serverless applications**⁷
- Instructor-led courses: **AZ-204: Developing Solutions for Microsoft Azure**⁸ and **AZ-020: Microsoft Azure Solutions for AWS Developers**⁹

Expected learning

² <https://docs.microsoft.com/en-us/learn/paths/azure-fundamentals/>
³ <https://docs.microsoft.com/en-us/learn/certifications/courses/az-900t01>
⁴ <https://docs.microsoft.com/en-us/learn/paths/az-104-administrator-prerequisites/>
⁵ <https://docs.microsoft.com/en-us/learn/certifications/courses/az-104t00>
⁶ <https://docs.microsoft.com/en-us/learn/certifications/courses/az-010t00>
⁷ <https://docs.microsoft.com/en-us/learn/paths/create-serverless-applications/>
⁸ <https://docs.microsoft.com/en-us/learn/certifications/courses/az-204t00>
⁹ <https://docs.microsoft.com/en-us/learn/certifications/courses/az-020t00>

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
- Design a tool integration strategy
- Design a license management strategy (e.g. Azure DevOps and GitHub 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
- Describe the benefits of using Source Control
- Describe Azure Repos and GitHub
- Migrate from TFVC to Git
- Manage code quality including technical debt SonarCloud, and other tooling solutions
- Build organizational knowledge on code quality
- 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
- Explain the role of Azure Pipelines and its components
- Configure Agents for use in Azure Pipelines
- Explain why continuous integration matters
- Implement continuous integration using Azure Pipelines
- Define Site Reliability Engineering
- Design processes to measure end-user satisfaction and analyze user feedback
- Design processes to automate application analytics
- Manage alerts and reduce meaningless and non-actionable alerts
- Carry out blameless retrospectives and create a just culture
- Define an infrastructure and configuration strategy and appropriate toolset for a release pipeline and application infrastructure
- Implement compliance and security in your application infrastructure
- Describe the potential challenges with integrating open-source software
- Inspect open-source software packages for security and license compliance
- Manage organizational security and compliance policies
- Integrate license and vulnerability scans into build and deployment pipelines
- Configure build pipelines to access package security and license ratings

Course syllabus

This course includes content that will help you prepare for the Microsoft 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 optional 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 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: Lab
 - Version controlling with Git in Azure Repos
- Lesson 9: Module review and takeaways

Module 3 – Managing Technical Debt

- Lesson 1: Module overview
- Lesson 2: Identifying technical debt
- Lesson 3: Knowledge sharing within teams
- Lesson 4: Modernizing development environments with Codespaces
- Lesson 5: Lab
 - Sharing team knowledge using Azure Project Wikis
- Lesson 6: Module review and takeaways

Module 4 – Working with 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 Git Hooks?
- Lesson 6: Fostering inner source
- Lesson 7: Managing Git Repositories
- Lesson 8: Lab
 - Version controlling with Git in Azure Repos
- Lesson 9: Module review and takeaways

Module 5 - Configuring Azure Pipelines

- Lesson 1: Module overview
- Lesson 2: The concept of pipelines in DevOps
- Lesson 3: Azure Pipelines
- Lesson 4: Evaluate use of hosted versus self-hosted 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 versus Visual Designer
- Lesson 9: Lab
 - Configuring agent pools and understanding pipeline styles
- Lesson 10: Module review and takeaways

Module 6 - Implementing Continuous Integration using Azure Pipelines

- Lesson 1: Module overview
- Lesson 2: Continuous integration overview
- Lesson 3: Implementing a build strategy
- Lesson 4: Integration with Azure Pipelines
- Lesson 5: Integrating external source control with Azure Pipelines
- Lesson 6: Set up self-hosted agents
- Lesson 7: Labs
 - Enabling continuous integration with Azure Pipelines
 - Integrating external source control with Azure Pipelines
- Lesson 8: Module review and takeaways

Module 7 - Managing Application Configuration and Secrets

- Lesson 1: Module overview

- Lesson 2: Introduction to security
- Lesson 3: Implement a secure development process
- Lesson 4: Rethinking application configuration data
- Lesson 5: Manage secrets, tokens, and certificates
- Lesson 6: Integrating with identity management systems
- Lesson 7: Implementing Application Configuration
- Lesson 8: Lab
 - Integrating Azure Key Vault with Azure DevOps
- Lesson 9: Module review and takeaways

Module 8: Implementing Continuous Integration with GitHub Actions

- Lesson 1: Module overview
- Lesson 2: GitHub Actions
- Lesson 3: Continuous integration with GitHub Actions
- Lesson 4: Securing secrets for GitHub Actions
- Lesson 5: Lab
 - Implementing GitHub Actions by using DevOps Starter
- Lesson 6: Module review and takeaways

Module 9 - Designing and Implementing a Dependency Management Strategy

- Lesson 1: Module overview
- Lesson 2: Packaging dependencies
- Lesson 3: Package management
- Lesson 4: Migrating and consolidating artifacts
- Lesson 5: Package security
- Lesson 6: Implementing a versioning strategy
- Lesson 7: Lab
 - Package management with Azure Artifacts
- Lesson 8: Module review and takeaways

Module 10 - Designing 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 the right release management tool

- Lesson 6: Labs
 - Controlling deployments using release gates
 - Creating a release dashboard
- Lesson 7: Module review and takeaways

Module 11 - Implementing Continuous Deployment using Azure Pipelines

- 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: Configure automated integration and functional test automation
- Lesson 6: Automate inspection of health
- Lesson 7: Labs
 - Configuring pipelines as code with YAML
 - Setting up and running functional tests
- Lesson 8: Module review and takeaways

Module 12 - Implementing an Appropriate Deployment Pattern

- Lesson 1: Module pverview
- 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: A/B 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 - Managing Infrastructure and Configuration using 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: Azure automation with DevOps
- Lesson 6: Desired State Configuration (DSC)

- Lesson 7: Lab
 - Azure Deployments using resource manager templates
- Lesson 8: Module review and takeaways

Module 14 - 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
-
- Ansible with Azure
 - Automating infrastructure deployments in the cloud with Terraform and Azure Pipelines
- Lesson 7: Module review and takeaways

Module 15 - Managing Containers using Docker

- Lesson 1: Module overview
 - Lesson 2: Implementing a container build strategy
 - Lesson 3: Implementing Docker multi-stage builds
 - Lesson 4: Lab
-
- Deploying Docker containers to Azure App Service web apps
 - Modernizing existing ASP.NET apps with Azure
- Lesson 5: Module review and takeaways

Module 16 - Creating and Managing Kubernetes Service Infrastructure

- Lesson 1: Module overview
 - Lesson 2: Azure Kubernetes Service (AKS)
 - Lesson 3: Kubernetes tooling
 - Lesson 4: Integrating AKS with pipelines
 - Lesson 5: Lab
-
- Deploying a multi-container application to Azure Kubernetes Services
- Lesson 6: Module review and takeaways

Module 17 - Implementing Feedback for 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 with Azure Application Insights
- Lesson 7: Module review and takeaways

Module 18 - Implementing System Feedback Mechanisms

- Lesson 1: Module overview
- Lesson 2: Site reliability engineering
- Lesson 3: Design practices to measure end-user satisfaction
- Lesson 4: Design processes to capture and analyze user feedback
- Lesson 5: Design processes to automate application analytics
- Lesson 6: Managing alerts
- Lesson 7: Blameless retrospectives and a just culture
- Lesson 8: Lab
 - Integration between Azure DevOps and Microsoft Teams
- Lesson 9: Module review and takeaways

Module 19 - Implementing Security in DevOps Projects

- Lesson 1: Module overview
- Lesson 2: Security in the pipeline
- Lesson 3: Azure Security Center
- Lesson 4: Lab
 - Implement security and compliance in an Azure DevOps Pipeline
- Lesson 5: Module review and takeaways

Module 20 - Validating Code Bases for Compliance

- Lesson 1: Module overview
- Lesson 2: Open-source software
- Lesson 3: Managing security and compliance policies
- Lesson 4: Integrating license and vulnerability scans
- Lesson 5: Lab
 - Managing technical debt with SonarQube and Azure DevOps
- Lesson 6: Module review and takeaways

Validate Lab Environment

Lab overview

In this lab, you will set up the lab environment for the rest of the course.

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- **AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions¹⁰**

¹⁰ <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/>

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:

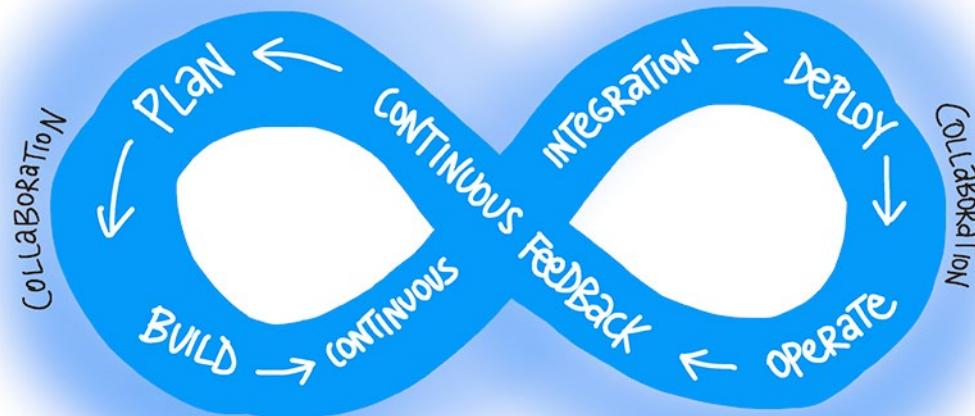
- Plan for the transformation with shared goals and timelines
- Select a project and identify project metrics and Key Performance Indicators (KPI's)
- Create a team and agile organizational structure
- Design a tool integration strategy
- Design a license management strategy (e.g. Azure DevOps and GitHub 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?

According to Donovan Brown, "DevOps is the union of people, process, and products to enable continuous delivery of value to our end users." **What is DevOps?**¹

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.

¹ <https://www.donovanbrown.com/post/what-is-devops>

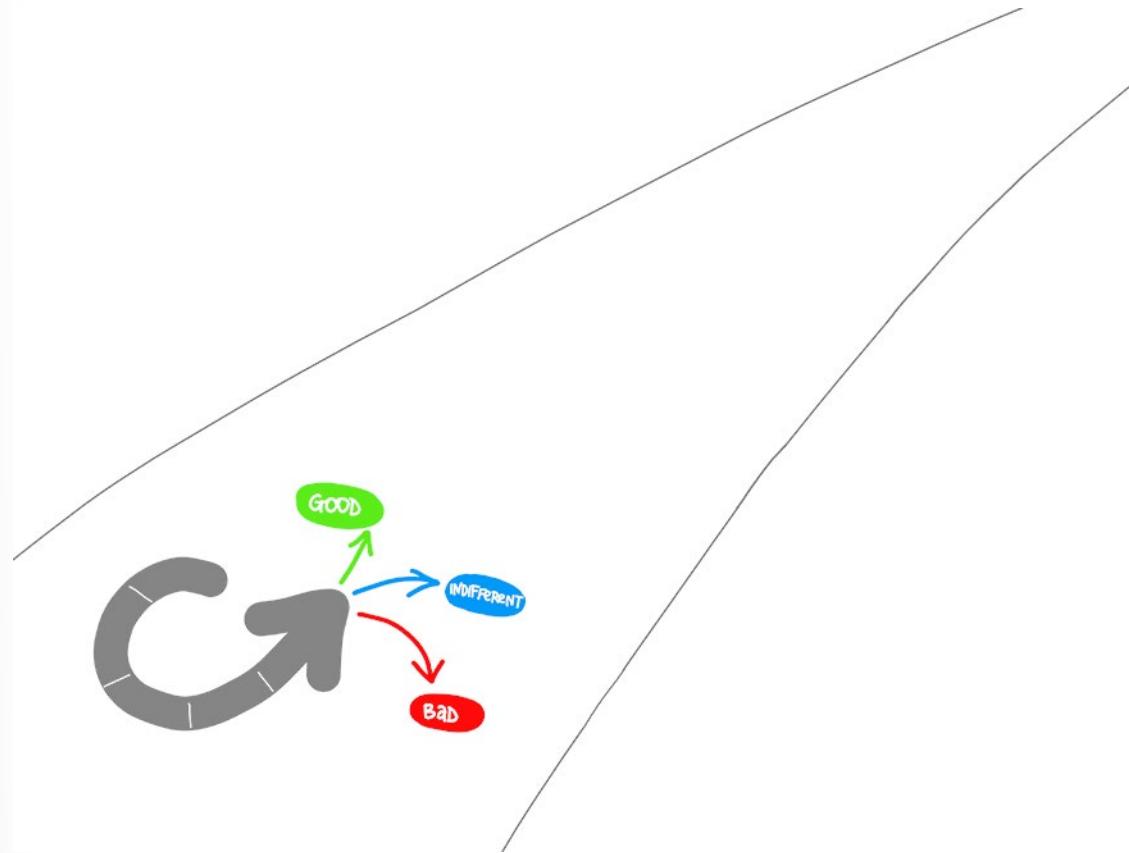


Become data-informed

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.

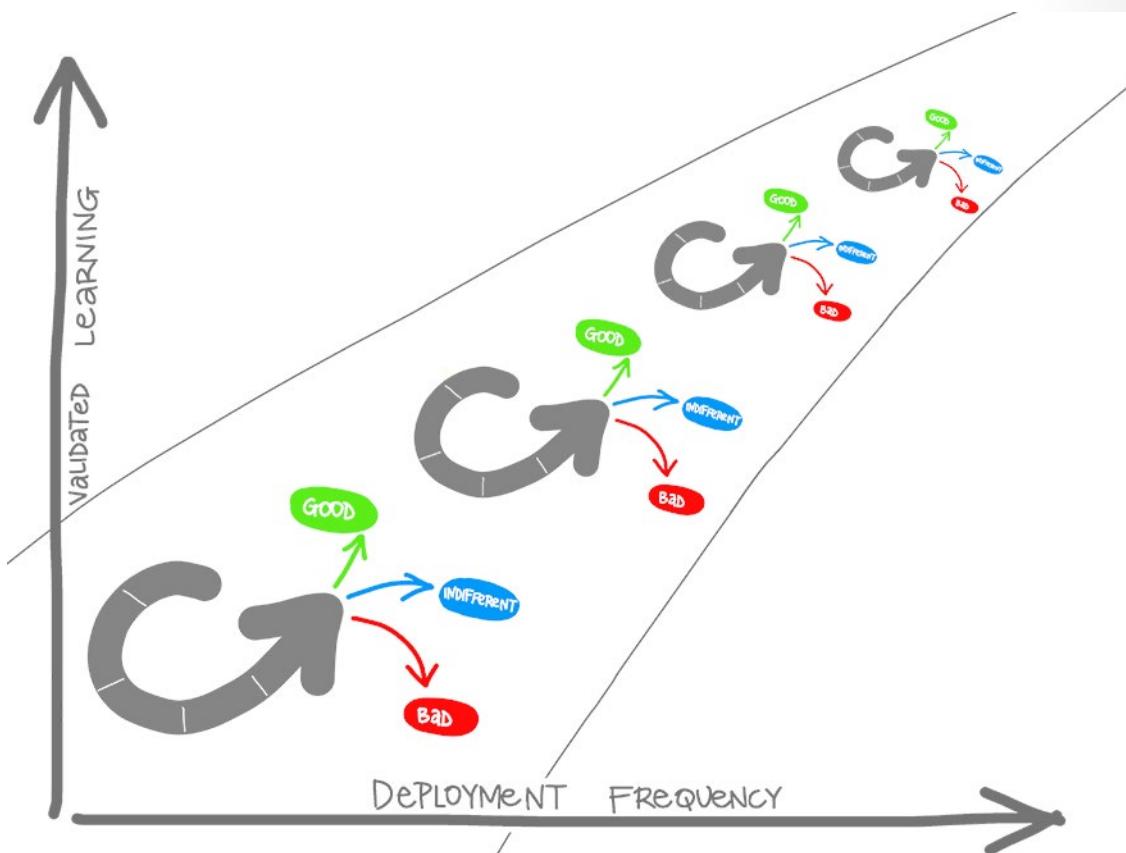
Strive for validated learning

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.



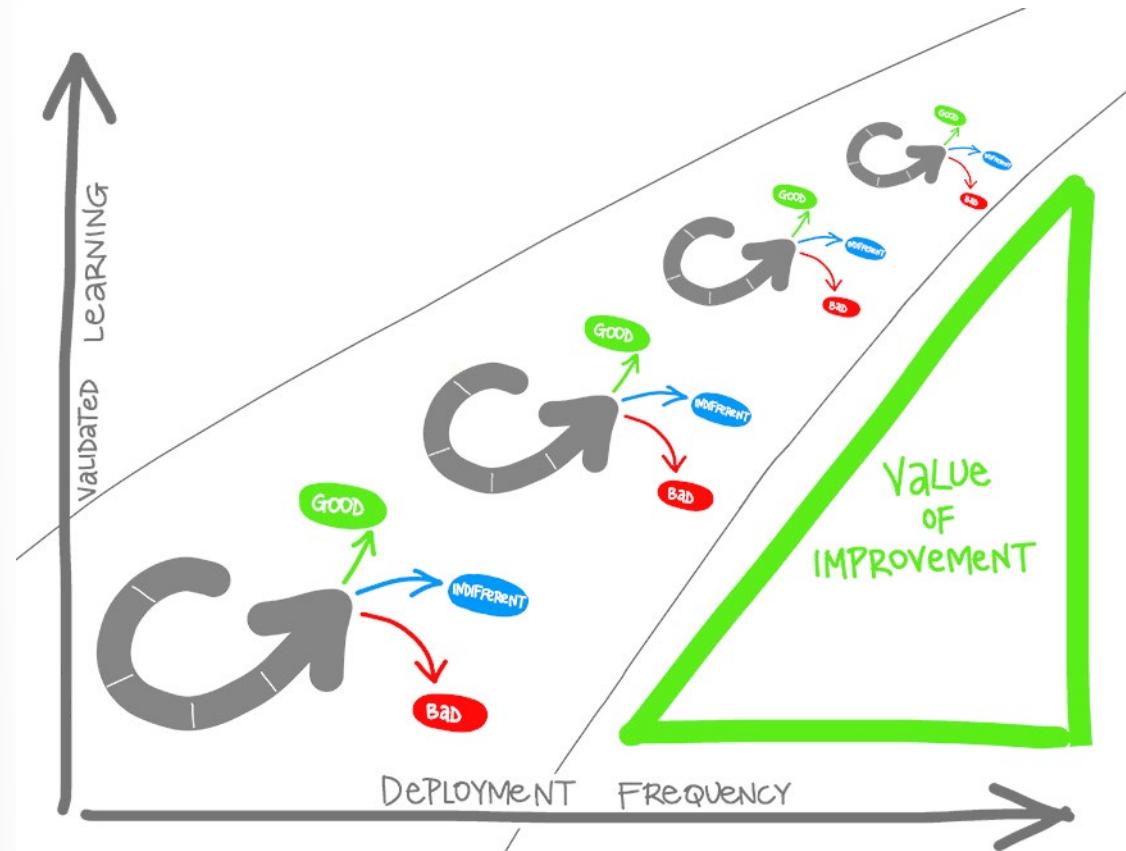
Shorten your cycle time

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.

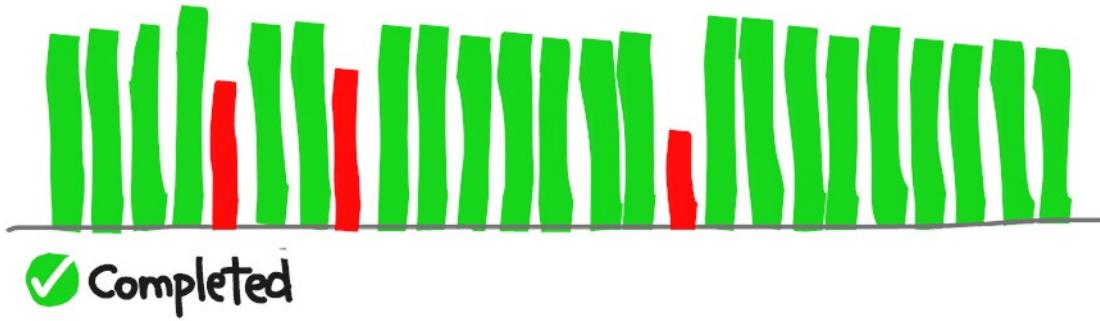


The DevOps journey

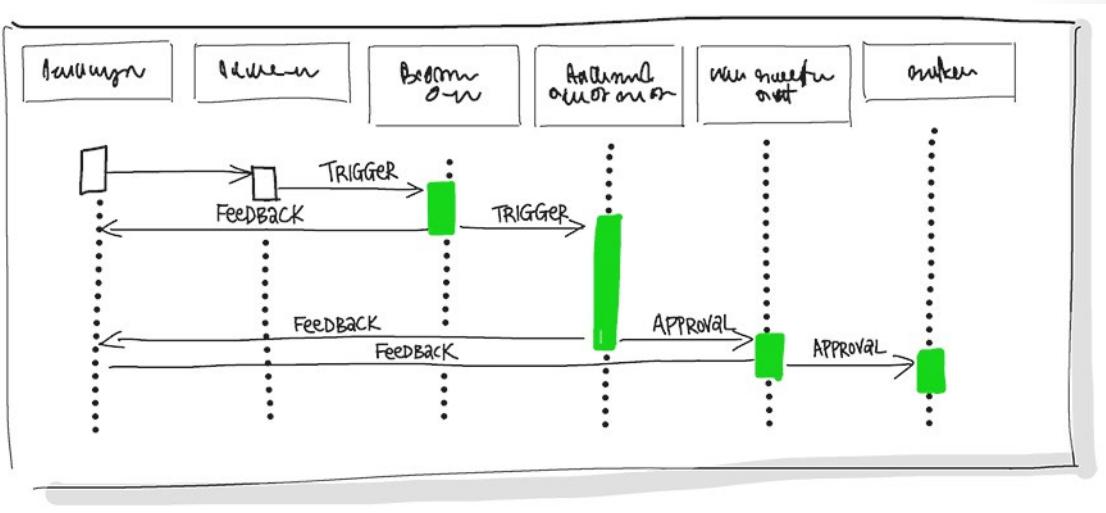
Remember, the goal is to shorten cycle time. Start with the release pipeline. How long does it take you to deploy a change of one line of code or configuration? Ultimately, that's the brake on your velocity.

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.

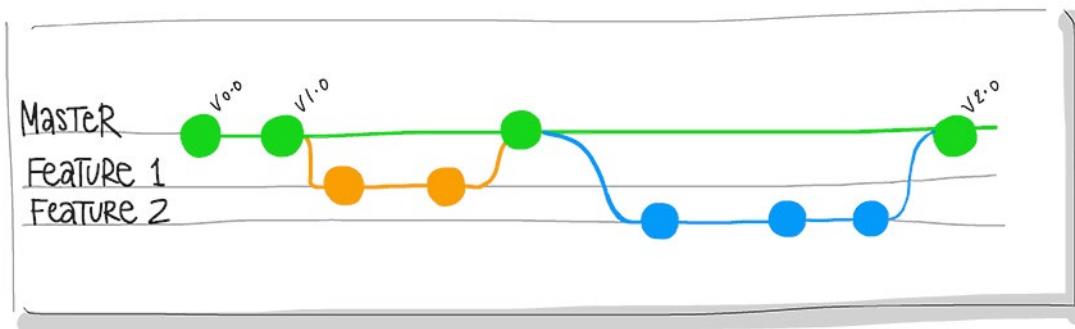
BUILD Succeeded



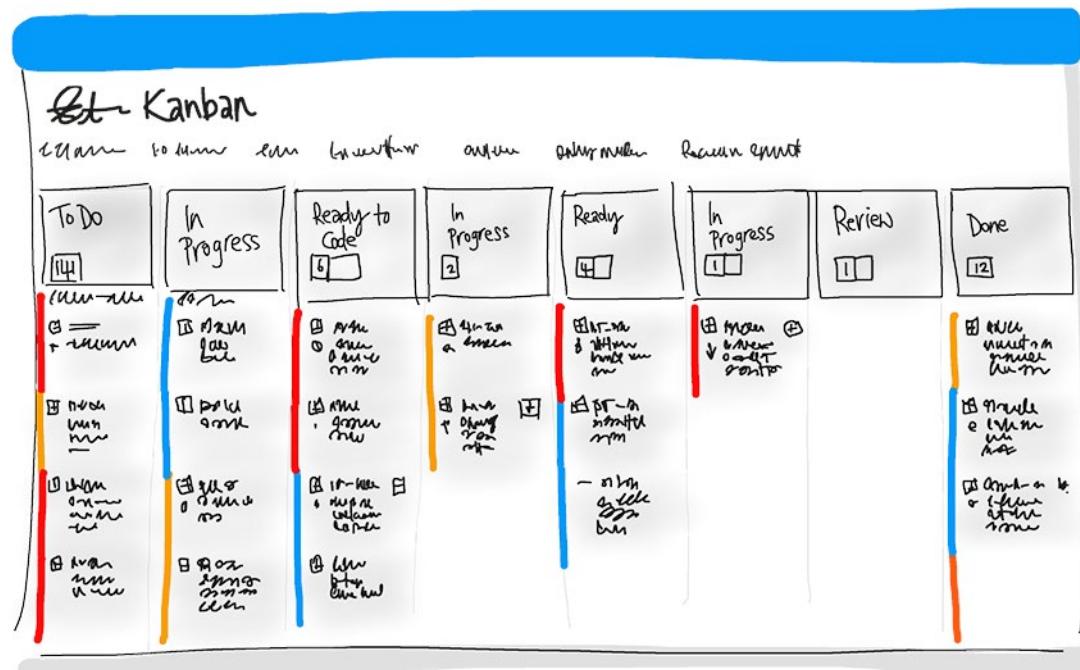
2. Continuous Delivery of software solutions to production and testing environments helps organizations quickly fix bugs and respond to ever-changing business requirements.



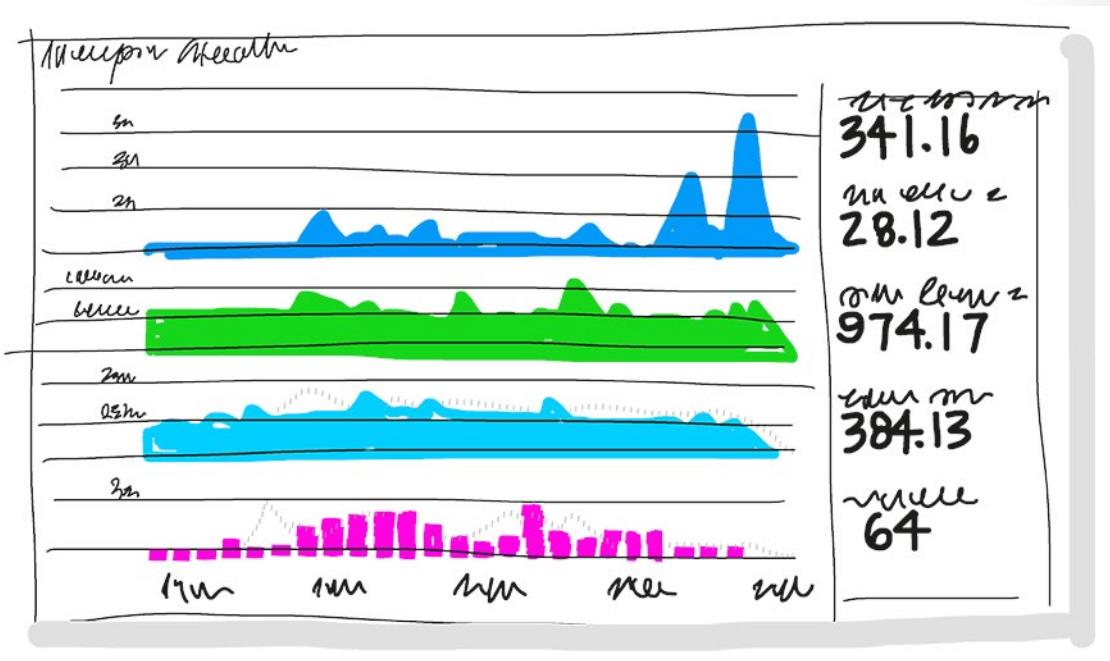
3. Version Control, usually with a Git-based Repository, enables teams located anywhere in the world to communicate effectively during daily development activities as well as to integrate with software development tools for monitoring activities such as deployments.



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.



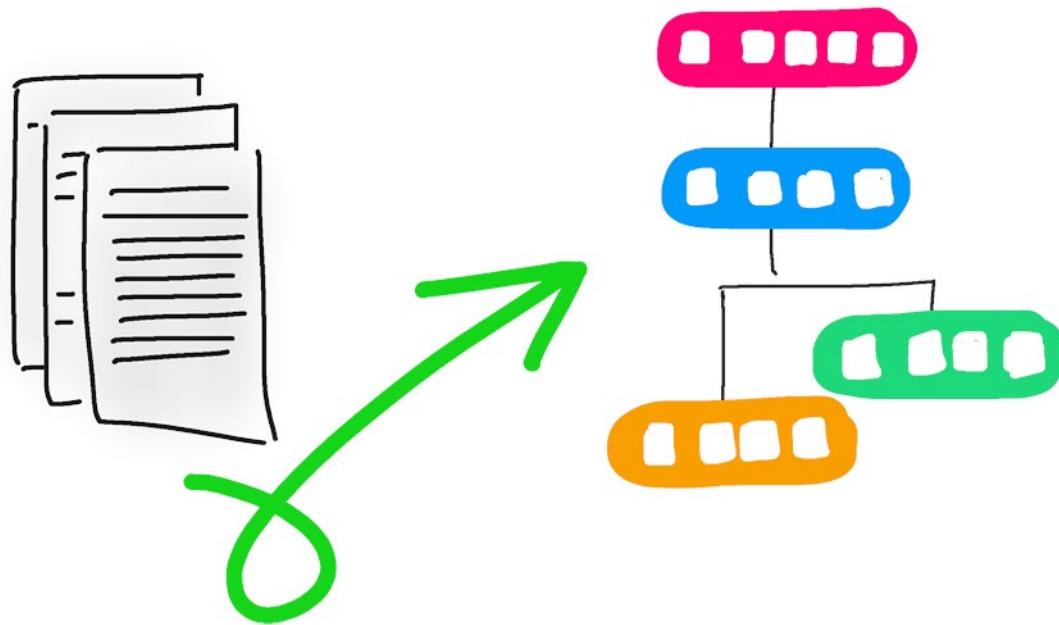
5. Monitoring and Logging of running applications including production environments for application health as well as customer usage, helps organizations form a hypothesis and quickly validate or disprove strategies. Rich data is captured and stored in various logging formats.



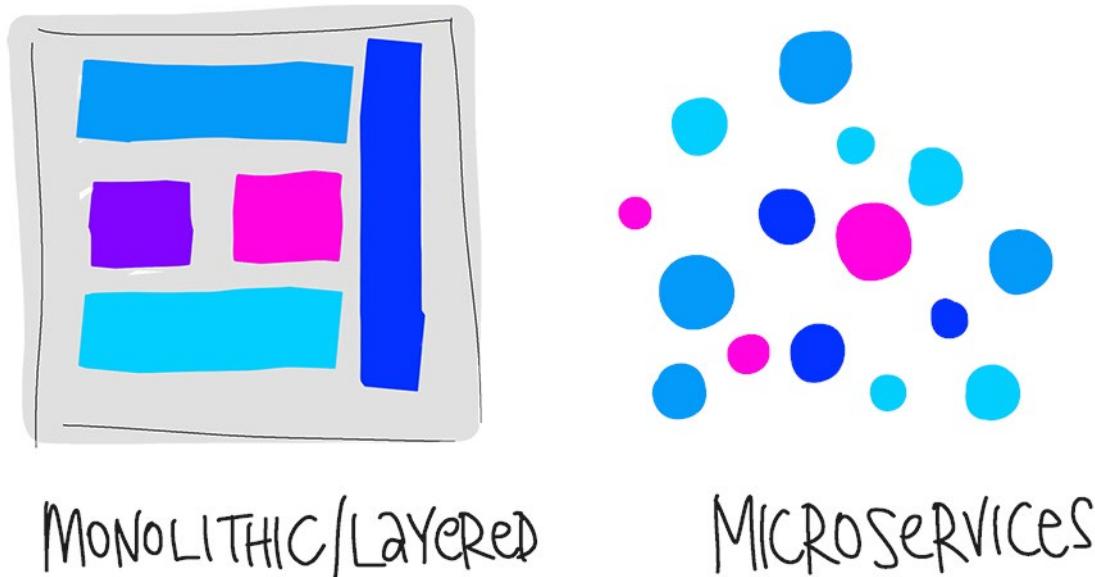
6. Public and Hybrid Clouds have made the impossible easy. The cloud has removed traditional bottlenecks and helped commoditize infrastructure. Whether you use Infrastructure as a Service (IaaS) to lift and shift your existing apps, or Platform as a Service (PaaS) to gain unprecedented productivity,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



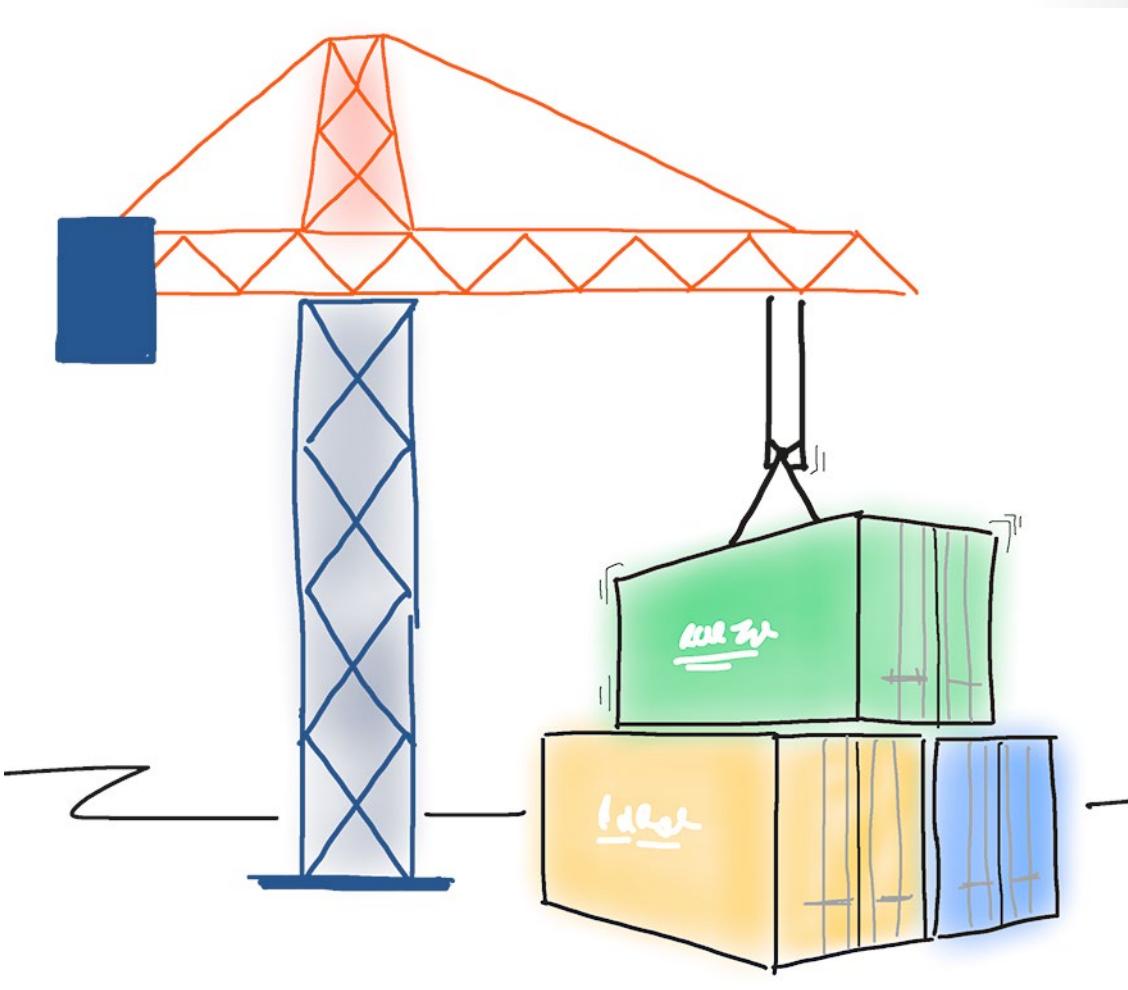
7. Infrastructure as Code (IaC) is a practice which enables the automation and validation of creation and teardown of environments to help with delivering secure and stable application hosting platforms.



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.

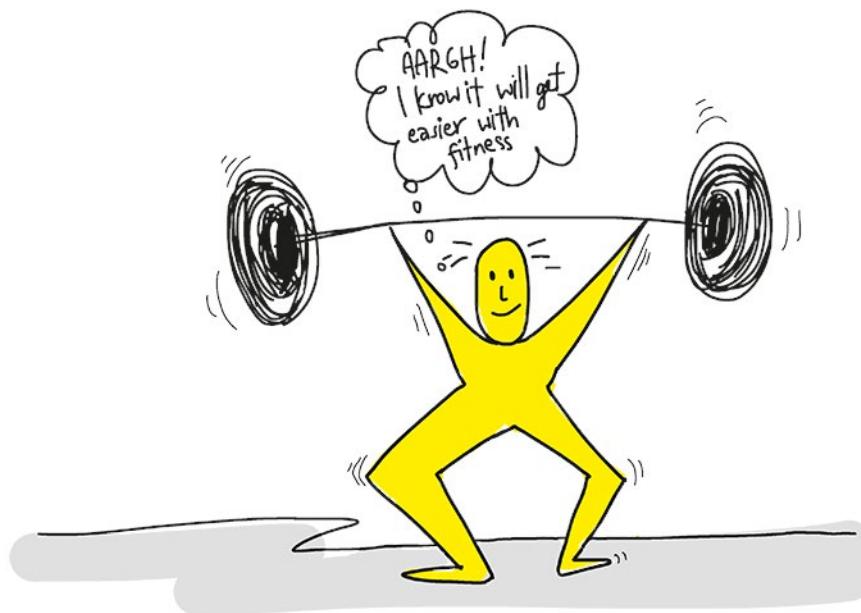


9. Containers are the next evolution in virtualization. They are much more lightweight than virtual machines, allow much faster hydration, and can be easily configured from files.



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.



Note: Source article is **defining DevOps²**.

Separating transformation teams

Unless you are building an entirely new organization, one of the big challenges with any DevOps Transformation Project is that you will take 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 despite the existing organizational processes. They concluded that it only works where a separate team is created to pursue the transformation.

² <https://docs.microsoft.com/en-us/azure/devops/learn/what-is-devops>

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 will 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 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 only for greenfield projects, and that it suits startups best.

However, DevOps can also succeed with brownfield projects. 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 versus 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 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 suits 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.

When choosing Canaries, it is important to find staff members who are both keen to see new features as soon as they are available, and who are also highly tolerant of issues that might arise.

Early adopters have similar characteristics to Canaries but often have work requirements that make them less tolerant to issues and interruptions to their ability to work.

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 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 timeframe.
- 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 key performance indicators (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 the creation of a work item, until it is completed?

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?
- **Service level agreement 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 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 like 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**³ 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**⁴.

From that, they have distilled the **12 Principles Behind the Agile Manifesto**⁵.

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. Businesspeople 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 many 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.

³ <https://www.agilealliance.org/>

⁴ <https://www.agilealliance.org/agile101/the-agile-manifesto/>

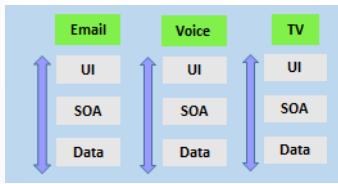
⁵ <https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/>

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:

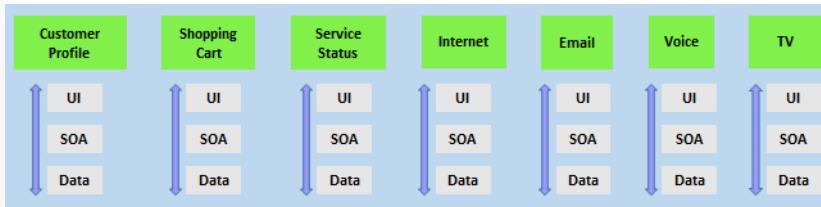


By comparison, vertical team structures span the architecture and are aligned with skill sets or disciplines:



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:



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.

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. At the time of writing, a big shift to working from home has started, initiated as a response to the pandemic. Both situations 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 timeframes, 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

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)⁶ 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.

Slack⁷ A 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 **GitHub⁸** for source code and **DropBox⁹** for document and file storage.

Jira¹⁰ A commonly used tool that allows for planning, tracking, releasing, and reporting.

Asana¹¹ A common tool that's designed to keep details of team plans, progress, and discussions in a single place. It has strong capabilities around timelines and boards.

Glip¹² An offering from Ring Central that provides chat, video, and task management.

Other common tools that include collaboration offerings include 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.

⁶ <https://products.office.com/en-us/microsoft-teams/group-chat-software>

⁷ <https://slack.com/>

⁸ <https://github.com/>

⁹ <https://dropbox.com/>

¹⁰ <https://www.atlassian.com/software/jira>

¹¹ <https://asana.com/>

¹² <https://glip.com/>

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 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 DevOps 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 DevOps, 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, Binfir, 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 DevOps

What can Azure DevOps do?

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 does Azure DevOps provide?

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 / Microsoft .NET 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 organizations building and shipping on the Microsoft technology stack.
- Cloud Agnostic: continuous delivery is supported to AWS and GCP as well as to Azure.

What can GitHub do?

GitHub is a Software as a service (SaaS) platform from Microsoft that provides Git-based repositories and DevOps tooling for developing and deploying software. It has a wide range of integrations with other leading tools.

What does GitHub provide?

GitHub provides a range of services for software development and deployment.

- **Codespaces:** Provides a cloud-hosted development environment (based on Visual Studio Code) that can be operated from within a browser or via external tools. Eases cross-platform development.
- **Repos:** Public and private repositories based upon industry standard Git commands.

- **Actions:** Allows for the creation of automation workflows. These workflows can include environment variables and customized scripts.
- **Artifacts:** The majority of the world's open-source projects are already contained in GitHub repositories. GitHub makes it easy to integrate with this code, and with other third-party offerings.
- **Security:** Provides detailed code scanning and review features, including automated code review assignment.

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.

Tools like Visual Studio and Azure DevOps natively support the use of Microsoft Accounts and AAD. Eclipse can also support this form of authentication if you install a Team Explorer Everywhere plug-in.

Personal access tokens

When you need to use tools like other Git-based repositories, or NuGet or Xcode and want them to integrate directly with Azure DevOps Services but the tools don't directly support Microsoft accounts or AAD accounts for authentication, you can generally 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.

Security groups

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.

Multifactor 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

Both Azure DevOps and GitHub can be integrated with a variety of existing work management tools. As an example, in the Visual Studio Marketplace, Microsoft offers **Trello integration tooling**¹³.

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**¹⁴ 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 DevOps¹⁵

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 JMeter¹⁶ is open-source software written in Java and designed to load test functional behavior and measure performance.

Pester¹⁷ is a tool that can be used to automate the testing of PowerShell code.

SoapUI¹⁸ 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**¹⁹.

¹³ <https://marketplace.visualstudio.com/items?itemName=ms-vsts.services-trello>

¹⁴ <https://marketplace.visualstudio.com/items?itemName=solidify-labs.jira-devops-migration>

¹⁵ <http://www.azurefieldnotes.com/2018/10/01/migrate-your-project-from-jira-to-azure-devops/>

¹⁶ <https://docs.microsoft.com/en-us/azure/devops/test/load-test/get-started-jmeter-test?view=vsts>

¹⁷ <https://marketplace.visualstudio.com/items?itemName=richardfennellIBM.BM-VSTS-PesterRunner-Task>

¹⁸ <https://marketplace.visualstudio.com/items?itemName=AjeetChouksey.soapui>

¹⁹ <https://marketplace.visualstudio.com/search?term=test%20management&target=AzureDevOps&category>All%20categories&sortBy=Relevance>

Designing a license management strategy

For the latest, most up-to-date Azure DevOps pricing information, visit [Azure DevOps Pricing²⁰](#).

For the latest, most up-to-date GitHub pricing information, visit [GitHub Pricing²¹](#).

²⁰ <https://azure.microsoft.com/en-us/pricing/details/devops/azure-devops-services/>

Lab

Agile planning and portfolio management with Azure Boards



Lab overview

In this lab, 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.

Objectives

After you complete this lab, you will be able to:

- Manage teams, areas, and iterations
- Manage work items
- Manage sprints and capacity
- Customize Kanban boards
- Define dashboards
- Customize team process

Lab duration

- Estimated time: **60 minutes**

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- **AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions²²**

²² <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/>

Module review and takeaways

Module review questions

Review Question 1

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

- System of Record
- System of Engagement

Review Question 2

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

Review Question 3

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

- Greenfield project
- Brownfield project

Review Question 4

As a project metric, what is Lead Time measuring?

Review Question 5

What is a cross-functional team?

Answers

Review Question 1

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.

Review Question 2

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.

Review Question 3

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 measures the total time elapsed from the creation of work items to their completion.

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 module, 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

Source 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 source control is today a commonsense 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 - stages 0 to 2

The [2019 State of DevOps report](#)¹, highlights version control in almost all stages of DevOps evolution.

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.

It's also helpful for non-developers in an organization to understand the fundamentals of the discipline as it is so deeply rooted in the daily life of software engineers. This is particularly important if those individuals are making decisions about which version control tools and platforms to use.

Foundational practices of DevOps - stages 3 to 5

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.

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, siloed organizations and hierarchical organization. All those characteristics represent coordination and integration challenges when it comes to merging and deploying code. Even more so in

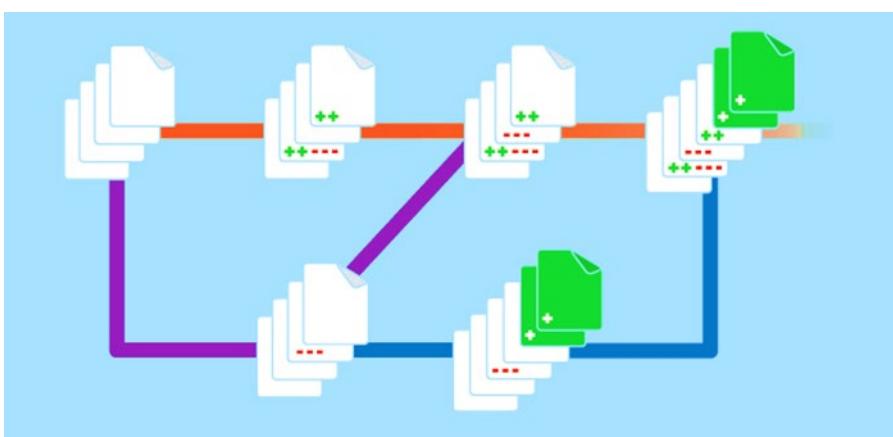
¹ <https://puppet.com/resources/report/state-of-devops-report>

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?

A Source control system (or version control system) allows developers to collaborate on code and track changes. Source control is an essential tool for multi-developer projects.

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.

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.

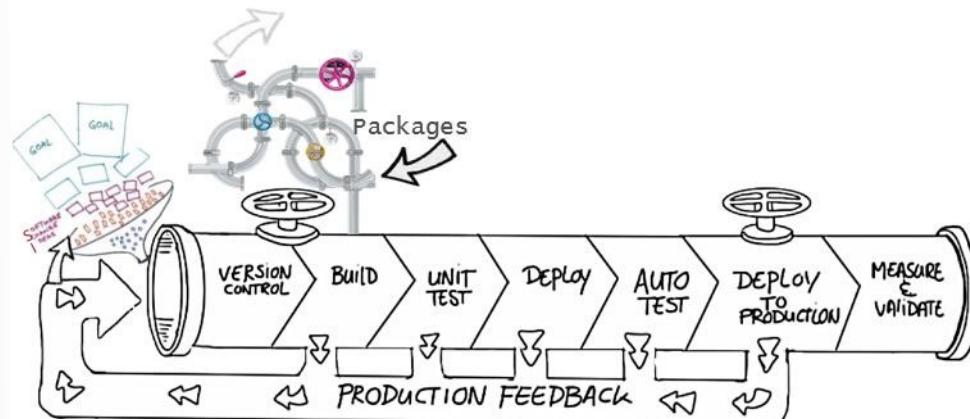
Version control is a solution for managing and saving changes made to any manually created assets. It allows you to go back in time and easily roll back to previously working versions if changes are made to source code. Version control tools allow you to see who made changes, when and what exactly was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

##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 debug-

ging 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 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.

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 source 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 source 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 must 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 source control workflow

When you’re working with a centralized source 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 source 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

Over time, so-called “distributed” source control or version control systems (DVCS for short) have become the most important. 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 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 source control

The act of cloning an entire repository gives distributed source 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 source control

There are almost no disadvantages to using a distributed source 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 development system. Developers can commit each set of changes on their dev machine and perform

version control 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:

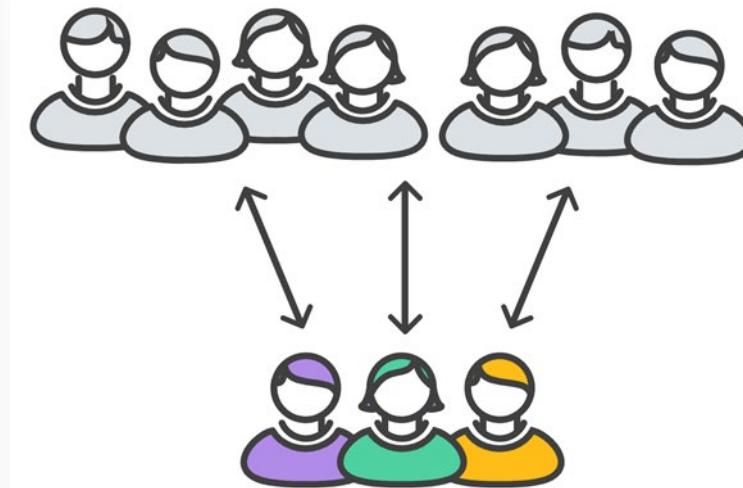
- **Server workspaces** - Before making changes, team members publicly check out files. Most operations require developers to be connected to the server. This system facilitates locking workflows. Other systems that work this way include Visual Source Safe, Perforce, and CVS. With server workspaces, 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

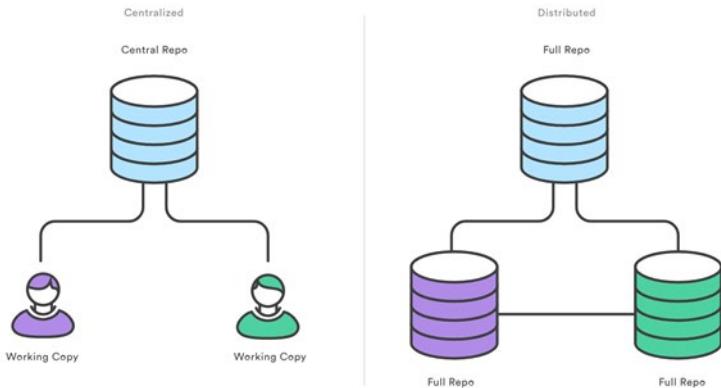
In many circles, Git has come to be the expected version control system for new projects. If your team is using Git, odds are you won't have to train new hires on your workflow, because they'll already be familiar with distributed development.



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.



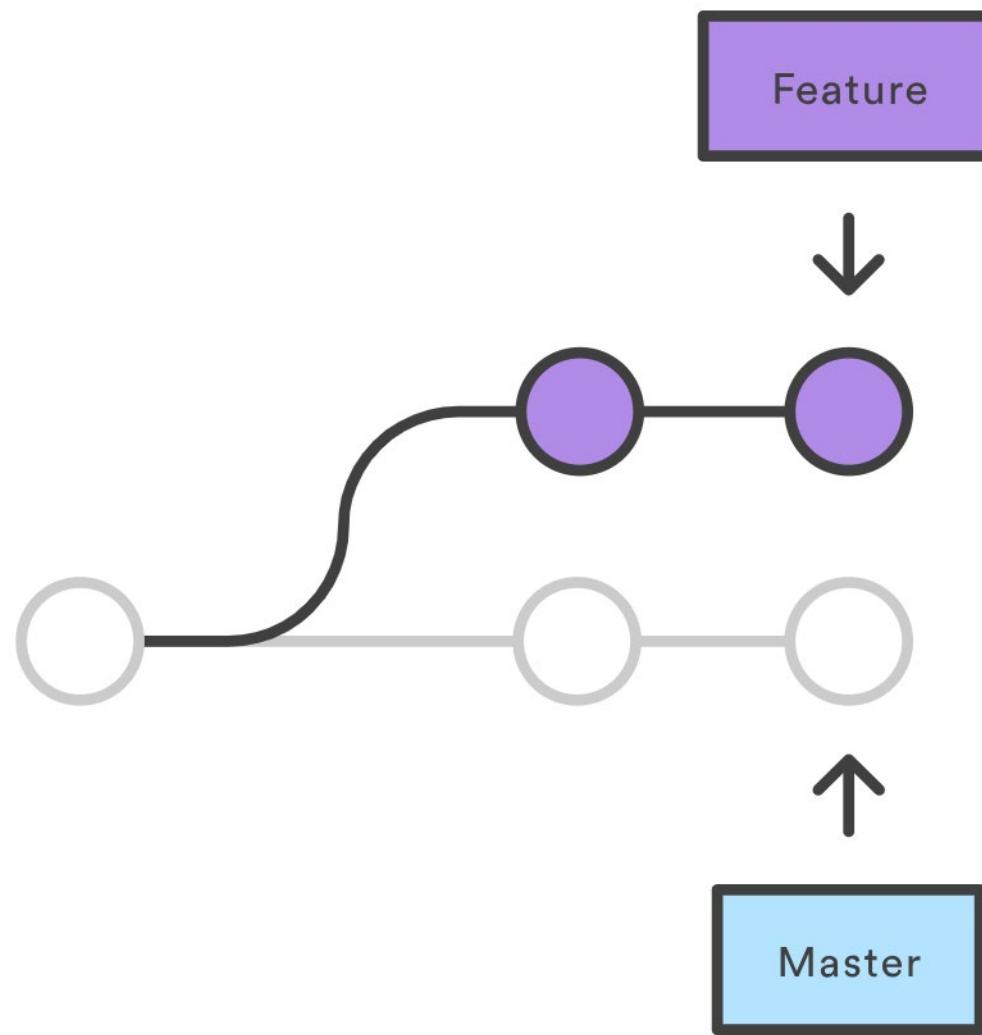
Having a full local history makes Git fast, since it means you don't need a network connection to create commits, inspect previous versions of a file, or perform diffs between commits.

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, like 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.

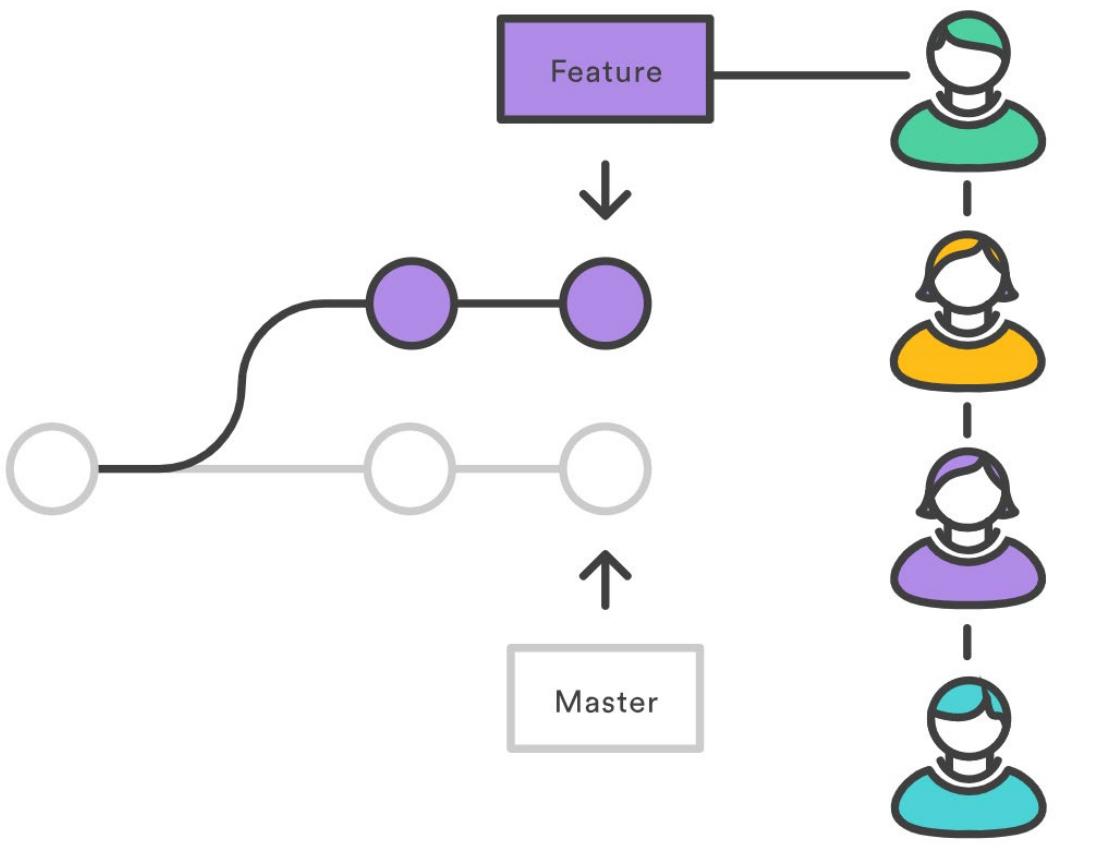


Trunk-based development provide an isolated environment for every change to your codebase. When a developer wants to start working on something—no matter how big or small—they create a new branch. This ensures that the master branch always contains production-quality code.

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

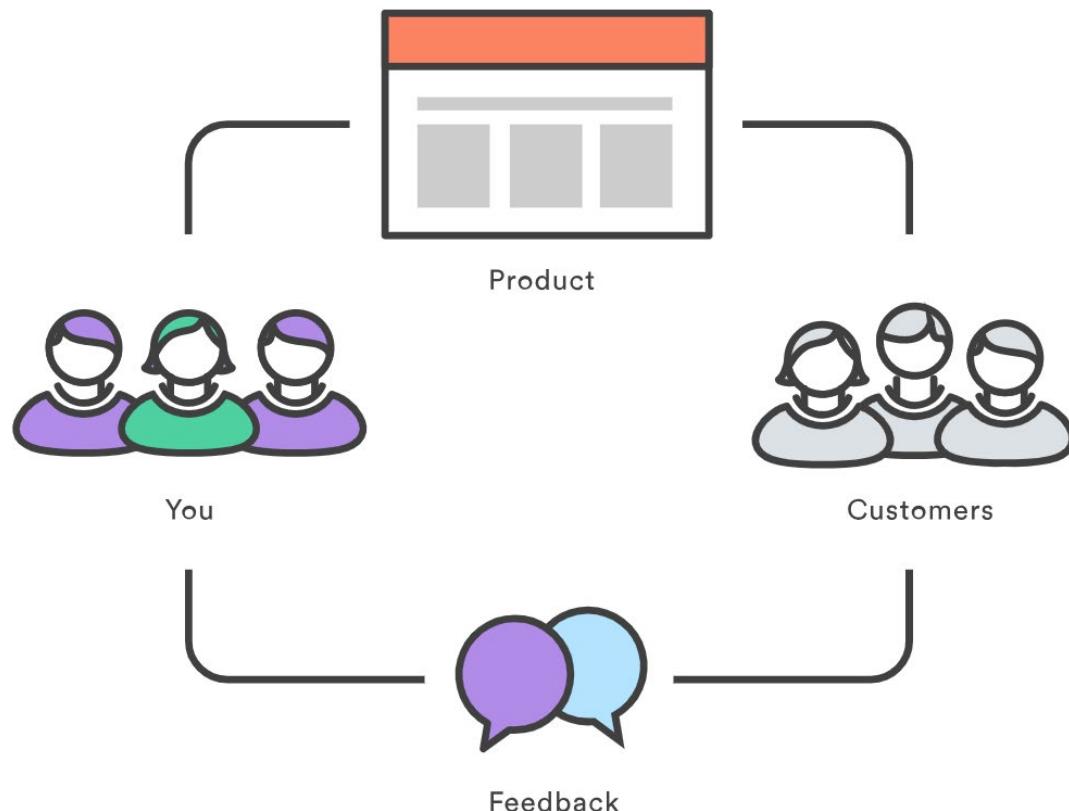
Many source code management tools such as Azure Repos enhance core Git functionality with pull requests. A pull request is a way to ask another developer to merge one of your branches into their repository. This not only makes it easier for project leads to keep track of changes, but also lets developers 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 three common objections I often hear to migrating to Git:

- I can overwrite history.
- I have large files.
- 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 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 because 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 for which you have source code. However, teams that have large files (like 3D models or other assets) can use Git LFS to keep the code repo slim and trim.

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 make the mental shift especially around branches and synchronizing. Once developers understand how Git branches work and get over the fact that they must 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 and 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 .NET⁴](#)

² <https://git-lfs.github.com/>

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

⁴ <https://dotnet.microsoft.com/download>

- Visual Studio Code: [Download Visual Studio Code⁵](https://code.visualstudio.com/Download)
- C# Visual Studio Code extension: [C# for Visual Studio Code⁶](https://marketplace.visualstudio.com/items?itemName=ms-dotnettools.csharp)
- Git: [Git - Downloads⁷](https://git-scm.com/downloads)
- Git for Windows (if you are using Windows): [Git for Windows⁸](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 Lens⁹](#): 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 History¹⁰](#): Brings visualization and interaction capabilities to view the Git log, file history, and compare branches or commits.

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 "John Doe"  
git config --global user.email "john.doe@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 docs¹¹](#).

⁵ <https://code.visualstudio.com/Download>

⁶ <https://marketplace.visualstudio.com/items?itemName=ms-dotnettools.csharp>

⁷ <https://git-scm.com/downloads>

⁸ <https://gitforwindows.org/>

⁹ <https://marketplace.visualstudio.com/items?itemName=eamodio.gitlens>

¹⁰ <https://marketplace.visualstudio.com/items?itemName=donjayamanne.githistory>

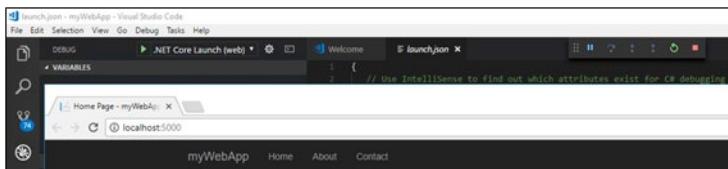
¹¹ <https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-new>

```
dotnet new mvc
```

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

```
code .
```

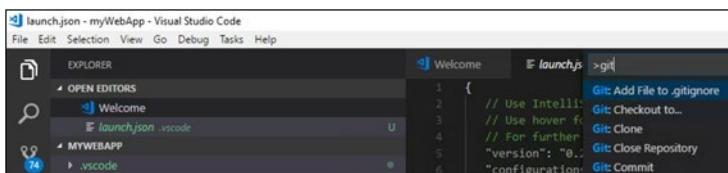
- When the project opens in Visual Studio Code, select **Yes** for the **Required assets to build and debug are missing from 'myWebApp'. Add them?** warning message. Select **Restore** for the **There are unresolved dependencies** info message. Hit **F5** to debug the application, then myWebApp will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



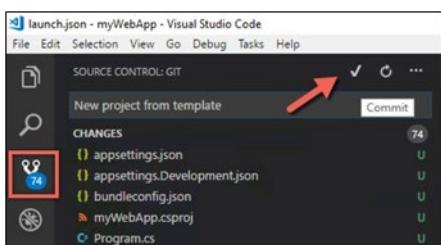
If you prefer to use the commandline, you can run the following commands in the context of the git repository to run the web application.

```
dotnet build  
dotnet run
```

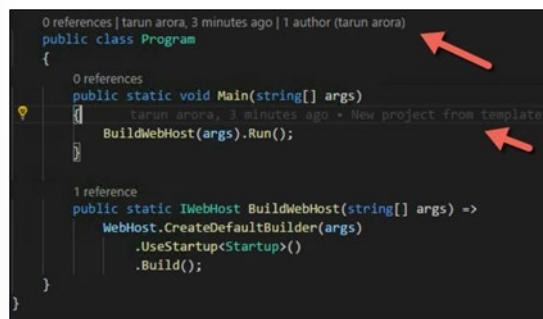
You'll notice the .vscode folder is added to your working folder. To avoid committing this folder into your Git repository, you can include this in the .gitignore file. With the .vscode folder selected, hit F1 to launch the command window in Visual Studio Code, type gitignore, and accept the option to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



7. 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`

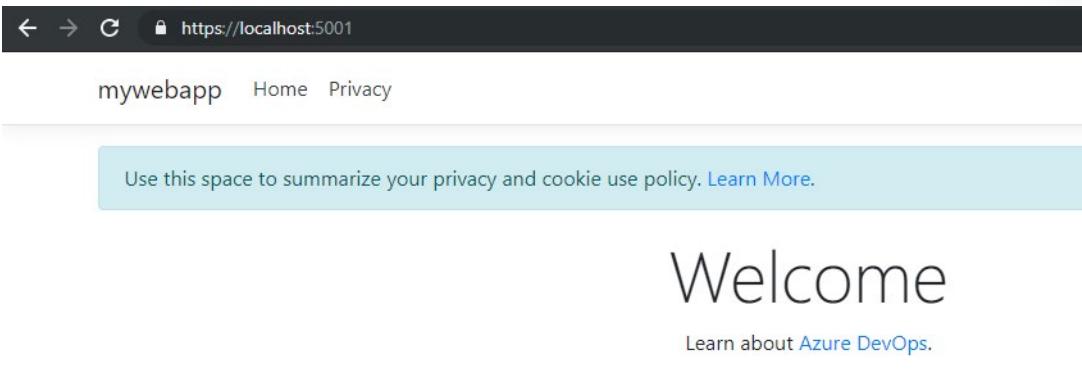
```
git branch feature-devops-home-page  
git checkout feature-devops-home-page  
git branch --list
```

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 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

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 Docs¹²](#)

¹² <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 scans security advisories for popular languages and sends security alerts to maintainers of affected repositories with details so they can remediate risks.
 - Automatically update vulnerabilities: GitHub monitors your project dependencies and automatically opens 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 its 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 [here¹³](#).

Linking GitHub to Azure Boards

Azure Boards has direct integration with Azure Repos, but it can also be integrated with GitHub.

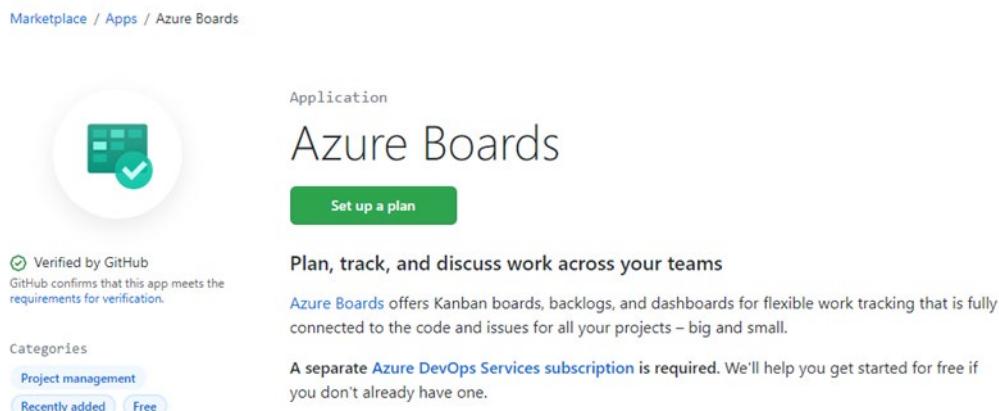
Integrating GitHub with Azure Boards lets you plan and track your work, by linking GitHub commits, pull requests, and issues, directly to work items in Boards.

Azure Boards App

The integration is created by using the Azure Boards App as a bridge between Azure Boards and GitHub.

To install the app, you must be an administrator or owner of the GitHub repository, or the GitHub organization.

The app is installed from the GitHub Marketplace. [Azure Boards App¹⁴](#)



Authenticating to GitHub

Azure Boards needs to be able to connect to GitHub. For GitHub in the cloud, when adding a GitHub connection, the authentication options are:

- Username/Password
- Personal Access Token (PAT)

For a walkthrough on making the connection, see: [Connect Azure Boards to GitHub¹⁵](#)

For details on linking to workitems, see: [Link GitHub commits, pull requests, and issues to work items¹⁶](#)

¹³ <https://lab.github.com/>

¹⁴ <https://github.com/marketplace/azure-boards>

¹⁵ <https://docs.microsoft.com/en-us/azure/devops/boards/github/connect-to-github?view=azure-devops>

¹⁶ <https://docs.microsoft.com/en-us/azure/devops/boards/github/link-to-from-github?view=azure-devops>

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

Migrating from TFVC to Git

Migrating the tip

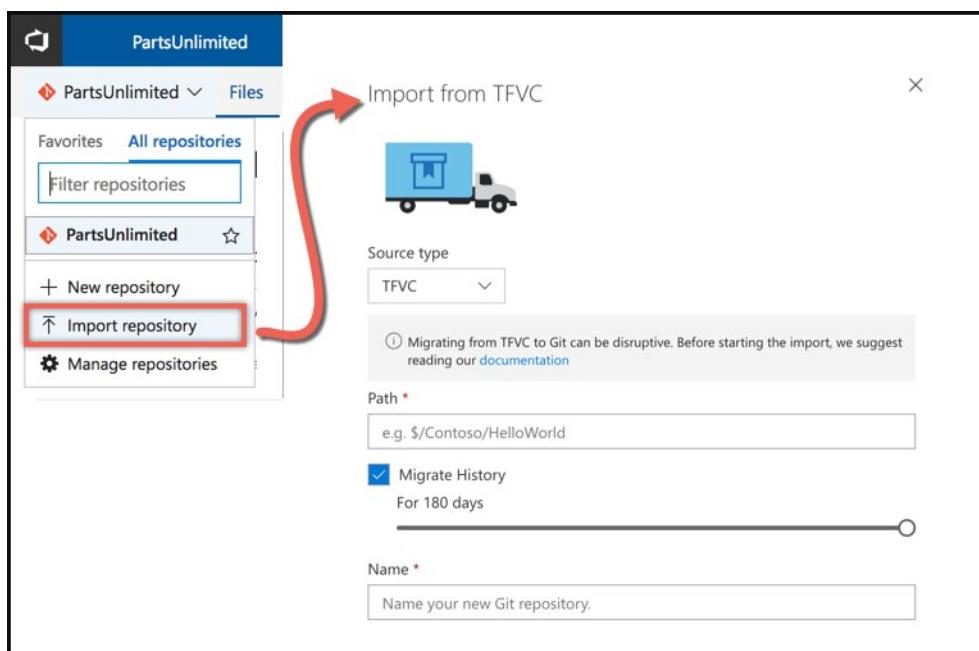
Most teams wish they could reorganize their source control structure - typically the structure the team is using today was set up by a well-meaning developer a decade ago but it's not really optimal. Migrating to Git could be a good opportunity to restructure your repo. In this case, it probably doesn't 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.

Single branch import

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

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.

Using 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 built 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://tfs:8080/tfs/DefaultCollection $/some_project
```

Some of the more advanced use cases of cloning the TFVC repository into git are **documented here¹⁷**.

```
cd some_project
git log
```

¹⁷ <https://github.com/git-tfs/git-tfs/blob/master/doc/commands/clone.md>

Lab

Version controlling with Git in Azure Repos

Lab overview

Azure DevOps supports two types of version control, Git and Team Foundation Version Control (TFVC). Here is a quick overview of the two version control systems:

- **Team Foundation Version Control (TFVC):** 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.
- **Git:** Git is a distributed version control system. Git repositories can live locally (such as on a developer's machine). 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 control operations such as history and compare without a network connection.

Git is the default version control provider for new projects. You should use Git for version control in your projects unless you have a specific need for centralized version control features in TFVC.

In this lab, 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. You will use Visual Studio Code, but the same processes apply for using any Git-compatible client.

Objectives

After you complete this lab, you will be able to:

- Clone an existing repository
- Save work with commits
- Review history of changes
- Work with branches by using Visual Studio Code

Lab duration

- Estimated time: **50 minutes**

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- [AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions¹⁸](https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/)

¹⁸ <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/>

Module review and takeaways

Module review questions

Review Question 1

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

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

Review Question 2

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

Review Question 3

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

Review Question 4

What is source control?

Answers

Review Question 1

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.

Review Question 2

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

Review Question 3

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 Managing technical debt

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. 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.

Learning objectives

After completing this module, students will be able to:

- Manage code quality including technical debt SonarCloud, and other tooling solutions
- Build organizational knowledge on code quality

Identifying technical debt

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 several 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.

Complexity metrics

While there are various quality metrics, a few of the most important ones are listed below.

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 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

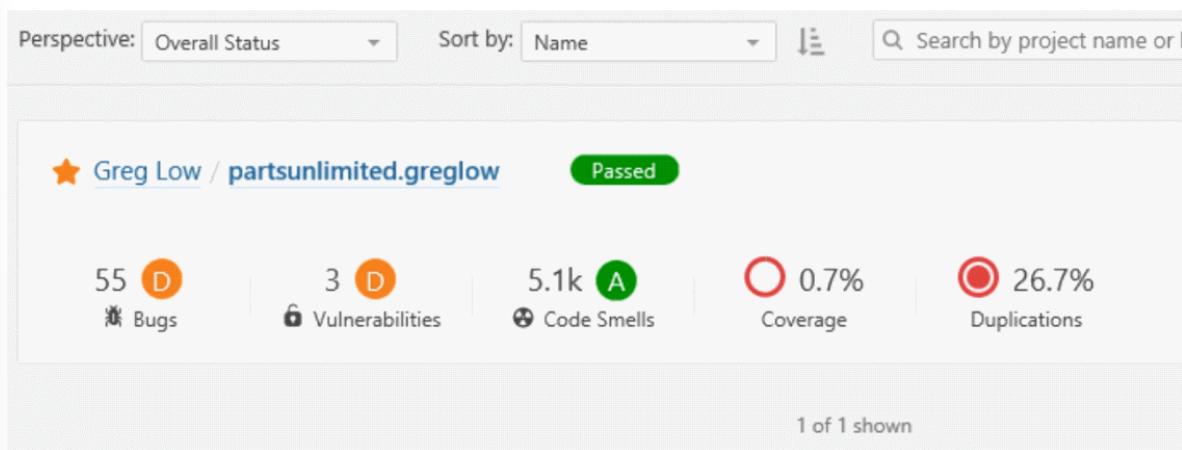
Azure DevOps can be integrated with a wide range of existing tooling that is used for checking code quality during builds. Which code quality tools do you currently use (if any)? What do you like or don't you like about the tools?

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.

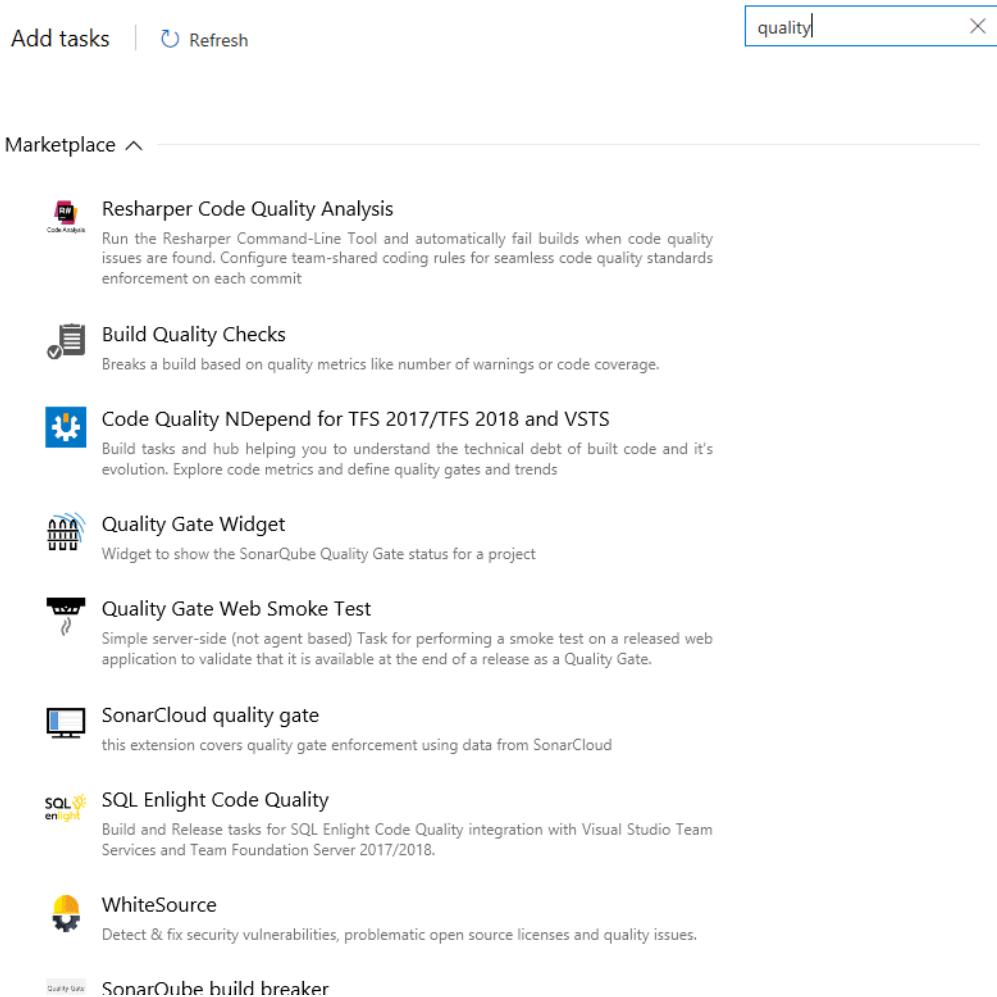
NDepend is a Visual Studio extension that assesses the amount of technical debt that a developer has added during a recent development period, typically in the last hour. With this information, the developer might be able to make the required corrections before ever committing the code. NDepend lets you create code rules that are expressed as C# LINQ queries, but it has many 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** by CodeAnalysis: 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**: Breaks a build based on quality metrics like number of warnings or code coverage.
- Code Quality NDepend for TFS 2017/TFS 2018 and VSTS**: 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**: Widget to show the SonarQube Quality Gate status for a project.
- Quality Gate Web Smoke Test**: 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**: this extension covers quality gate enforcement using data from SonarCloud
- SQL Enlight Code Quality** by SQLenlight: Build and Release tasks for SQL Enlight Code Quality integration with Visual Studio Team Services and Team Foundation Server 2017/2018.
- WhiteSource**: Detect & fix security vulnerabilities, problematic open source licenses and quality issues.
- SonarQube build breaker**

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 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.

Knowledge sharing within teams

Sharing acquired knowledge within development teams

Organizational knowledge builds up within development teams over time. It is important to avoid endlessly relearning the same lessons that the team (and the organization) has learned before.

As staff turnover occurs in teams, it is easy for organizational knowledge to be lost. It is important to record this knowledge to avoid this loss.

A simple example is if the team has developed coding standards, these should be captured and not just exist as word of mouth.

Relearning old lessons is both wasteful and expensive.

Discussion - tools for knowledge sharing

Discussion: Tools for knowledge sharing

Azure DevOps can be used in conjunction with a wide range of existing tooling that is used for sharing knowledge. Which knowledge sharing tools do you currently use (if any)? What do you or don't you like about the tools?

Azure DevOps project wikis

Azure DevOps projects include an option to create a project wiki.

The wiki to share information with your team to understand and contribute to your project.

Wikis are stored in a repository. They need to be created (no wiki is automatically provisioned).

Prerequisites

You must have the permission **Create Repository** to publish code as wiki. While the **Project Administrators** group has this permission by default, it can be assigned to others.

To add or edit wiki pages, you should be a member of the **Contributors** group.

All members of the team project (including stakeholders) can view the wiki.

Creation

The following article includes details on creating a wiki: **Create a Wiki for your project⁵**

Editing the wiki

The following article includes details on how to publish a Git repository to a wiki: **Publish a Git repository to a wiki⁶**

Wiki contents

Azure DevOps Wikis are written in Markdown and can also include file attachments and videos.

Markdown

Markdown is a markup language. Plain text includes formatting syntax. It has become the defacto standard for how project and software documentation is now written.

One key reason for this, is that because it is made up of plain text, it is much easier to merge in much the same way that program code is merged. This allows documents to be managed with the same tools used to create other code in a project.

GitHub Flavored Markdown (GFM)

GFM is a formal specification released by GitHub that added extensions to a base format called CommonMark. GFM is widely used both within GitHub and externally. GFM is rendered in Azure DevOps Wikis.

Mermaid

Mermaid has become an important extension to Markdown because it allows diagrams to be included in the documentation. This overcomes the previous difficulties in merging documentation that includes diagrams represented as binary files.

```
Welcome to the project wiki for **Contoso**. The aim of this wiki is to record key lessons learned by our project team over time.
```

```
::: mermaid
graph LR;
A[Wiki supports Mermaid] --> B[Visit https://mermaidjs.github.io/ for Mermaid syntax];
:::
```

Welcome to the project wiki for **Contoso**. The aim of this wiki is to record key lessons learned by our project team over time.

Wiki supports Mermaid → Visit <https://mermaidjs.github.io/> for Mermaid syntax

Details on Mermaid syntax can be found here: **Mermaid Introduction⁷**

⁵ <https://docs.microsoft.com/en-us/azure/devops/project/wiki/wiki-create-repo?view=azure-devops&tabs=browser>

⁶ <https://docs.microsoft.com/en-us/azure/devops/project/wiki/publish-repo-to-wiki?view=azure-devops&tabs=browser>

⁷ <https://mermaid-js.github.io/mermaid/>

Modernizing development environments with GitHub Codespaces

Developing online with GitHub Codespaces

GitHub Codespaces addresses several issues from which developers regularly suffer.

First, many developers are working with old hardware and software systems, and they aren't refreshed regularly enough.

Second, developers are often tied to individual development systems, and moving from location to location, or system to system, is inconvenient and/or slow to configure.

A problem for the organizations that the developers work for, is that there is also a proliferation of the intellectual property across all these machines.

What is GitHub Codespaces ?

Codespaces is a cloud-based development environment that is hosted by GitHub. It is essentially an online implementation of Visual Studio Code. Codespaces allows developers to work entirely in the cloud. Codespaces even allows developers to contribute from tablets and Chromebooks.

Because it's based on Visual Studio Code, the development environment is still a rich environment with syntax highlighting, autocomplete, integrated debugging, and direct Git integration.

Developers can create a codespace (or multiple codespaces) for a repository. Each codespace is associated with a specific branch of a repository.

Using GitHub Codespaces

While you can do all your work in codespaces within a browser, for an even more responsive experience, you can connect to a codespace from a local copy of Visual Studio Code.

Lab

Sharing team knowledge using Azure project wikis

Lab overview

In this lab, you will create and configure wiki in an Azure DevOps, including managing markdown content and creating a Mermaid diagram.

Objectives

After you complete this lab, you will be able to:

- Create a wiki in an Azure Project
- Add and edit markdown
- Create a Mermaid diagram

Lab duration

- Estimated time: **45 minutes**

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- **AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions⁸**

⁸ <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/>

Module review and takeaways

Module review questions

Review Question 1

What are Mermaid diagrams?

Review Question 2

What are code smells? Give an example of a code smell.

Review Question 3

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?

Review Question 4

Name three attributes of high-quality code.

Review Question 5

You are using Azure Repos for your application source code repository. You want to perform code quality checks. Which tool could you use?

Answers

What are Mermaid diagrams?

Mermaid diagrams are code written in markdown to represent diagrams.

What are 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 4 Working with Git for Enterprise Dev-Ops

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 and 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

Monorepo versus multiple 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, over time it is not uncommon for the code repositories to be bloated with unrelated code and artifacts.

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

The fundamental difference between the monorepo and multiple repos philosophies boils down to a difference about what will allow teams working together on a system to go fastest. The multiple repos view, in extreme form, is that if you let every subteam 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 must 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 must 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 must deal with not only a different code base but potentially with different libraries and tools or even a different workflow. Or maybe you just must ask someone who owns that system to make the change for you and wait for them to get around to it.

The monorepo view, on the other hand, is that that friction, especially when dealing with more complicated dependency graphs, is much more costly than multiple repos 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 monorepo 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 monorepo 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.

Azure DevOps project repositories

In Azure DevOps, a project can contain multiple repositories. It is common to use one repository for each associated solution.

Implementing a change log

The concept of a change log is simple enough: It's a file that has a list of changes made to a project, usually in date order. The typical breakdown is to separate a list of versions, and then within each version, show:

- Added features
- Modified/Improved features
- Deleted features

Some teams will post change logs as blog posts; others will create a CHANGELOG.md file in a GitHub repository.

Automated change log tooling

While change logs can be created and manually maintained, you might want to consider using an automated change log creation tool, at least as a starting point.

Using native GitHub commands

The git log command can be useful for automatically creating content. Example: create a new section per version:

```
git log [options] vX.X.X..vX.X.Y | helper-script > projectchangelogs/X.X.Y
```

gitchangelog

One common tool is **gitchangelog**¹. This tool is based on Python.

GitHub change log generator

Another common tool is called **github-changelog-generator**²

```
$ github_changelog_generator -u github-changelog-generator -p Timer-Trend-3.0
```

This tool is based on Gem.

Should you use auto-generated log-based data?

Preference is to always avoid dumping log entries into a change log. Logs are “noisy” and so it’s easy to generate a mess that is not helpful.

¹ <https://pypi.org/project/gitchangelog/>

² <https://github.com/github-changelog-generator/github-changelog-generator>

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 regarding 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 nvie³**. 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.

³ <https://nvie.com/posts/a-successful-git-branching-model/>

Feature branch 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 will never contain broken code, which is a huge advantage for continuous integration environments.

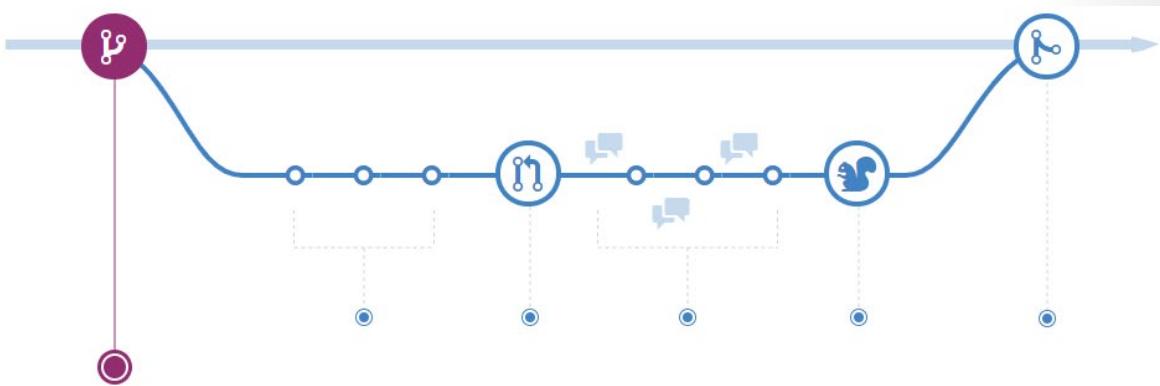
Encapsulating feature development also makes it possible to leverage pull requests, which are a way to initiate discussions around a branch. They give other developers the opportunity to sign off on a feature before it gets integrated into the official project. Or, if you get stuck in the middle of a 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.

Trunk-based development workflow

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



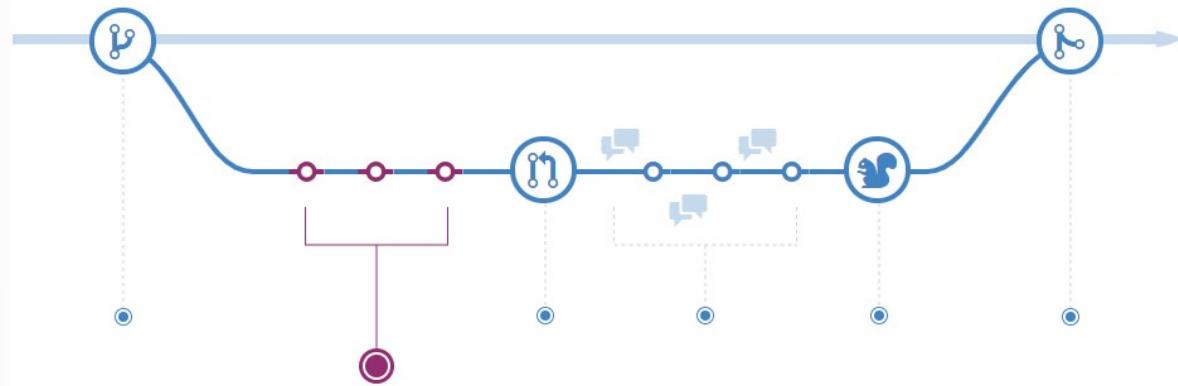
When you're working on a project, you're going to have a bunch of different features or ideas in progress at any given time – some of which are ready to go, and others which are not. Branching exists to help you manage this workflow.

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 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

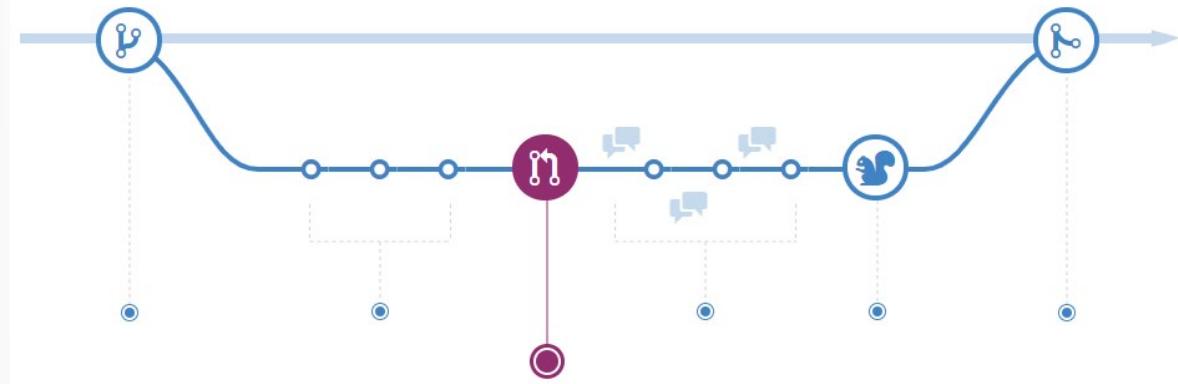


Once your branch has been created, it's time to start making changes. Whenever you add, edit, or delete a file, you're making a commit, and adding them to your branch. This process of adding commits keeps track of your progress as you work on a feature branch.

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



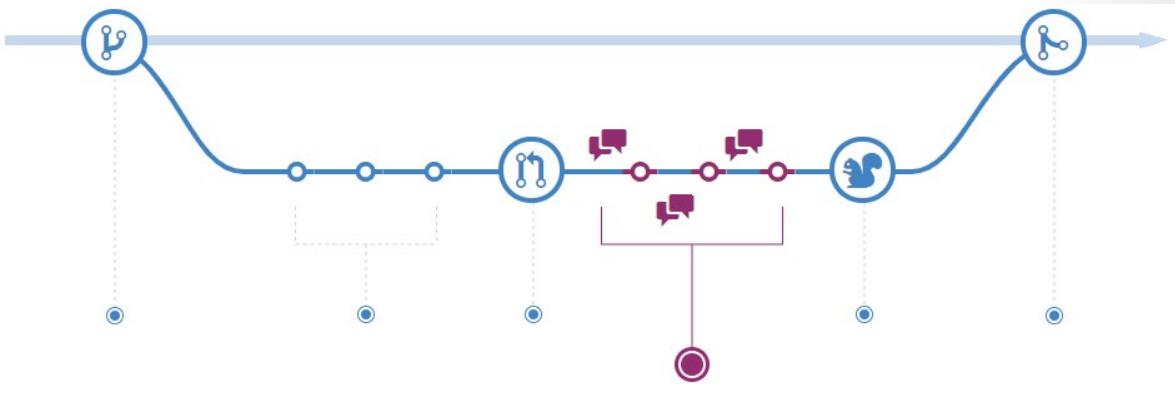
Pull Requests initiate discussion about your commits. Because they're tightly integrated with the underlying Git repository, anyone can see exactly what changes would be merged if they accept your request.

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

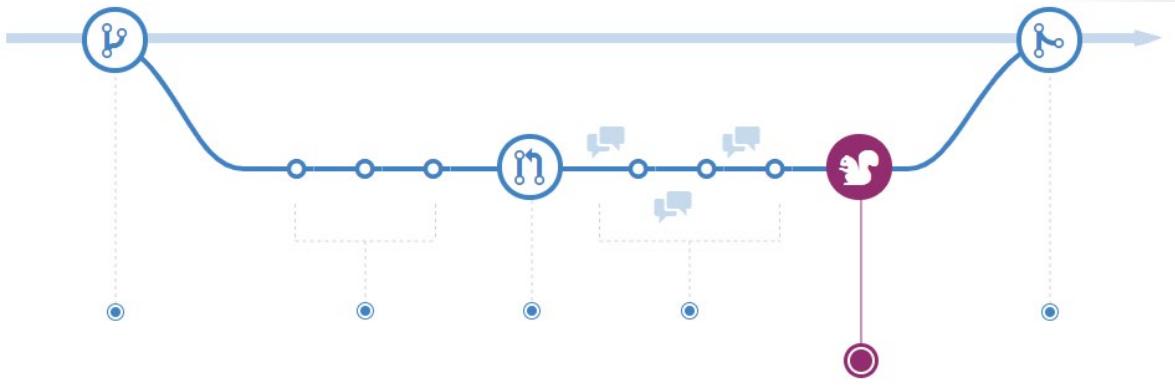


Once a Pull Request has been opened, the person or team reviewing your changes may have questions or comments. Perhaps the coding style doesn't match project guidelines, the change is missing unit tests, or maybe everything looks great and props are in order. Pull Requests are designed to encourage and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

You can also continue to push to your branch considering 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.

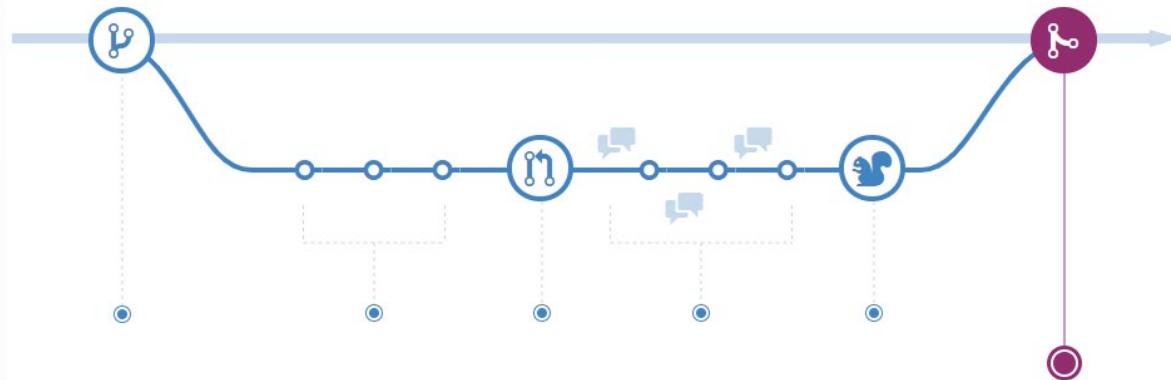
Deploy



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 close.

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 does 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 three very popular extensions from the marketplace:
- Azure CLI (<https://docs.microsoft.com/en-gb/cli/azure/install-azure-cli?view=azure-cli-latest>) is a command line interface for Azure.
 - Azure DevOps CLI (<https://marketplace.visualstudio.com/items?itemName=ms-vsts.cli>): This is an extension for the Azure CLI for working with 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.

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 new1. 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:

The screenshot shows the Azure DevOps repository interface for the 'feature/myFeature-1' branch. The 'History' tab is selected. Two commits are listed:

- 70f67b28 feature 1 added to program.cs
- 47b4370d Vlab for hands on demo

A red arrow points to the message of the first commit: "feature 1 added to program.cs".

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

```
> az repos 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 screenshot shows a GitHub pull request interface. At the top, a green button indicates the pull request is 'COMPLETED'. The title is 'Review Feature-1 before merging to master'. Below the title, it says 'Tarun Arora' merged 'feature/myFeature-1 into master'. There are tabs for 'Overview', 'Files', 'Updates', 'Commits', and 'Conflicts'. On the left, under 'Policies', it shows 'Required' with two items checked: '1 reviewer approved' and 'All comments resolved'. A message on the right states 'Tarun Arora completed the pull request on 2/7/2018 4:14 PM (just now)' and lists the commit 'b2ee4332' and the merge commit 'Merged PR 9: Review Feature-1 before merging to master...'.

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

The screenshot shows a 'Create a tag' dialog box. At the top, it says 'Create a tag'. It has fields for 'Name *' containing 'release_feature1', 'Tag from' set to 'master', and 'Description *' containing 'Tagging the master for feature 1 release'. At the bottom are 'Create' and 'Cancel' buttons.

- 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 comment line in the code that was changed in feature-1

```
public class Program
{
    // Editing the same line (file from feature-2 branch)
    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 pull8. request to merge the changes from feature/myFeature-2 to the master branch:

```
> az repos 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. 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:

```
public class Program
{
    // Editing the same line (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

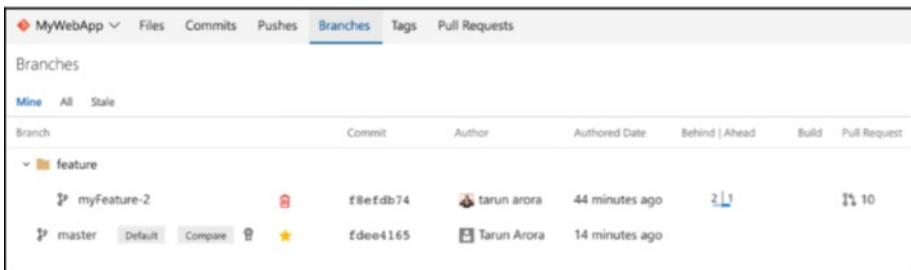
To https://dev.azure.com/Geeks/PartsUnlimited/_git/MyWebApp * [new branch]
  fof/bug-1 -> fof/bug-1 Branch fof/bug-1 set up to track remote branch fof/
  bug-1 from origin
```

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 repos 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.

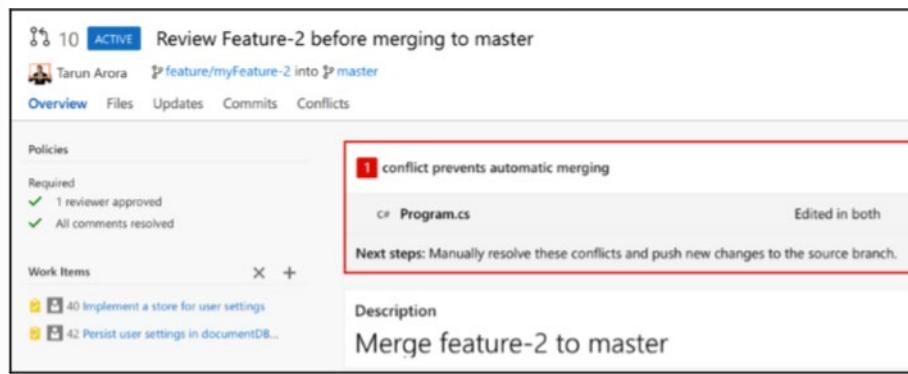
With the critical bug fix out of the way, let's go back to the review of the feature-2 pull request. The branches page makes it clear that the feature/myFeature-2 branch is one change ahead of the master and two changes behind the master:



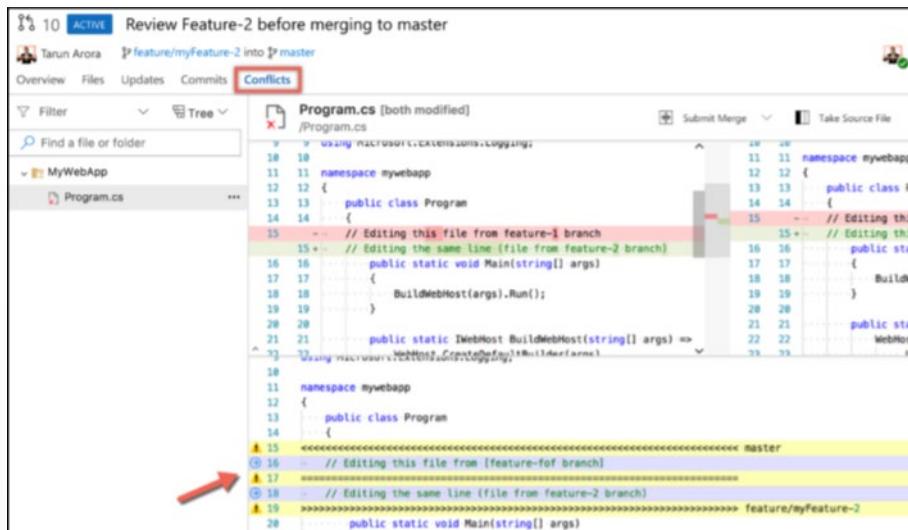
The screenshot shows the Azure DevOps 'Branches' page for the 'MyWebApp' repository. The 'feature' branch is selected. It lists two commits: one from 'myFeature-2' (commit f8eefdb74) and one from 'master' (commit fdee4165). The 'myFeature-2' commit is ahead of 'master' by 1 commit, and behind 'master' by 2 commits.

Branch	Commit	Author	Authored Date	Behind Ahead	Build	Pull Request
feature	f8eefdb74	tarun arora	44 minutes ago	2 1		
master	fdee4165	Tarun Arora	14 minutes ago			

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



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 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.

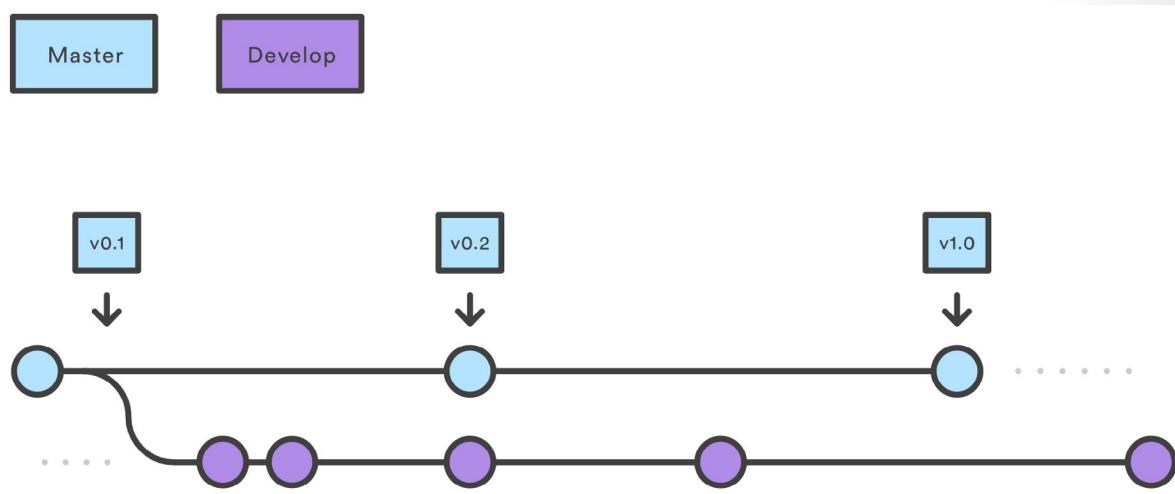
Gitflow is ideally suited for projects that have a scheduled release cycle. This workflow doesn't add any new concepts or commands beyond what's required for the Feature Branch Workflow. Instead, it assigns very specific roles to different branches and defines how and when they should interact. In addition 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 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]
```

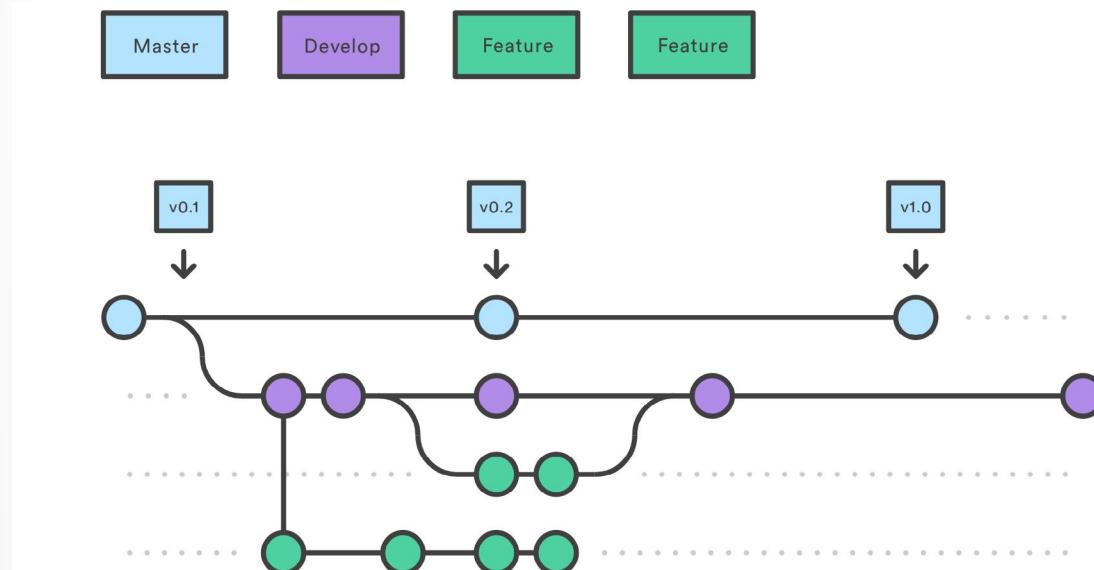
How to name your supporting branch prefixes?

```
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 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.

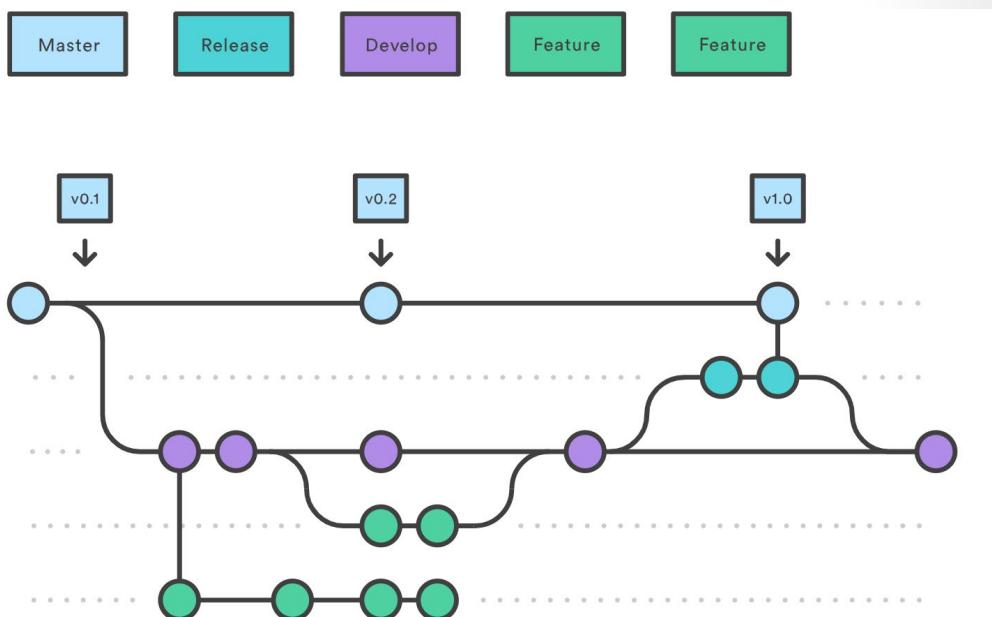
Without the git-flow extensions:

```
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 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 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.

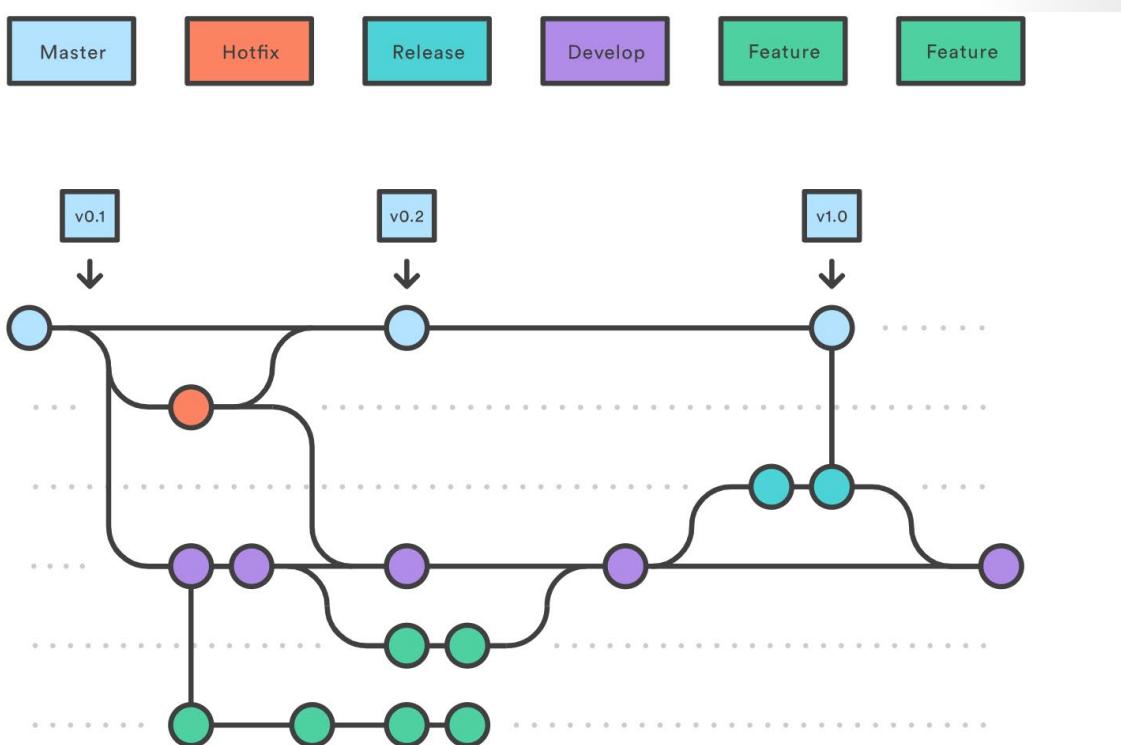
To finish a release branch, use the following methods:

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 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
```

Like 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 can 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.

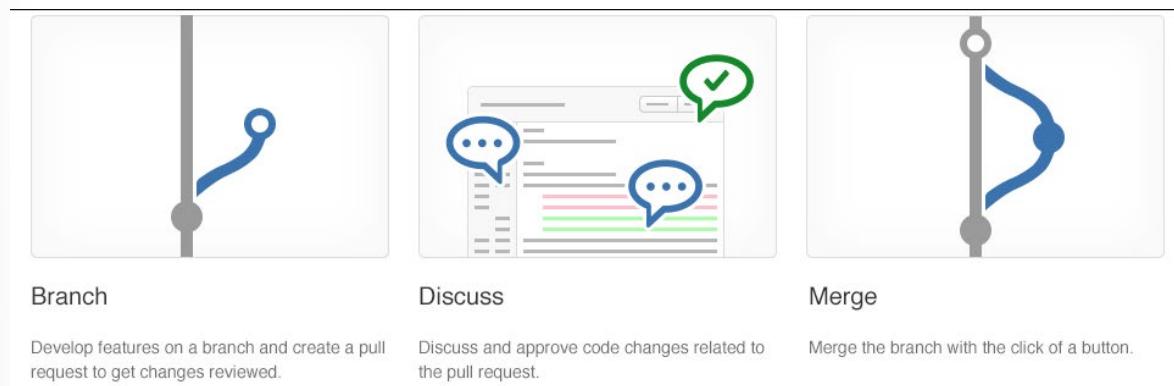
Collaborating with pull requests in Azure Repos

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 try to provide feedback on changes they don't agree with. Identify the issue and give a specific suggestion 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

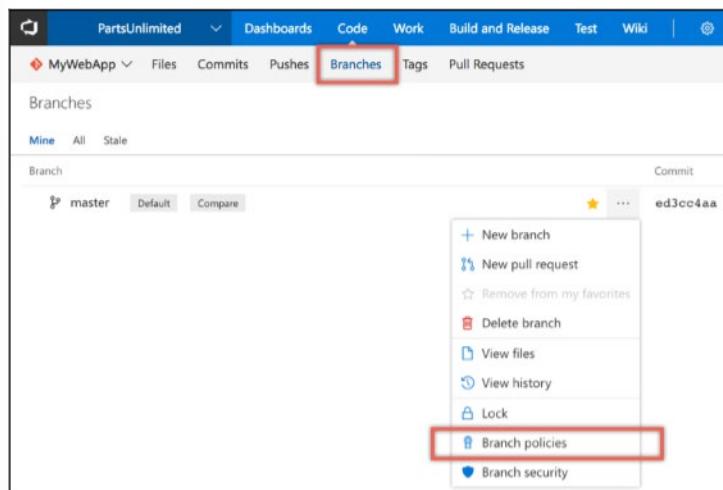
Code issues that are found sooner are both easier and cheaper to fix, therefore development teams strive to push code quality checks as far left into the development process as possible. As the name suggests, branch policies give you a set of out-of-the-box policies that can be applied to the branches 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:

Policies for: PartsUnlimited > MyWebApp > master

Save changes Discard changes

Protect this branch

- Code changes must be submitted via pull request
- This branch cannot be deleted
- Manage permissions for this branch on the [Security page](#)

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:

Require a minimum number of reviewers

Require approval from a specified number of reviewers on pull requests.

Minimum number of reviewers

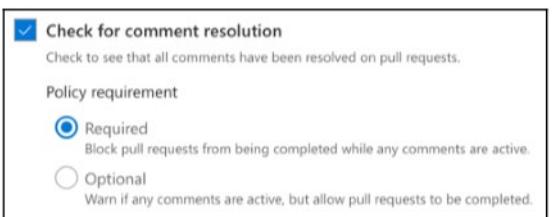
Allow users to approve their own changes.

Allow completion even if some reviewers vote "Waiting" or "Reject".

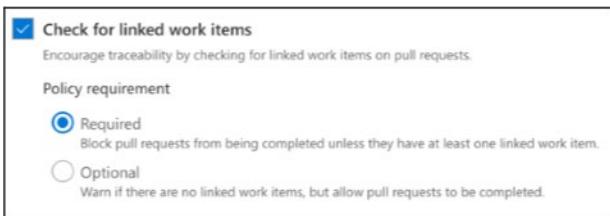
Reset code reviewer votes when there are new changes.

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.

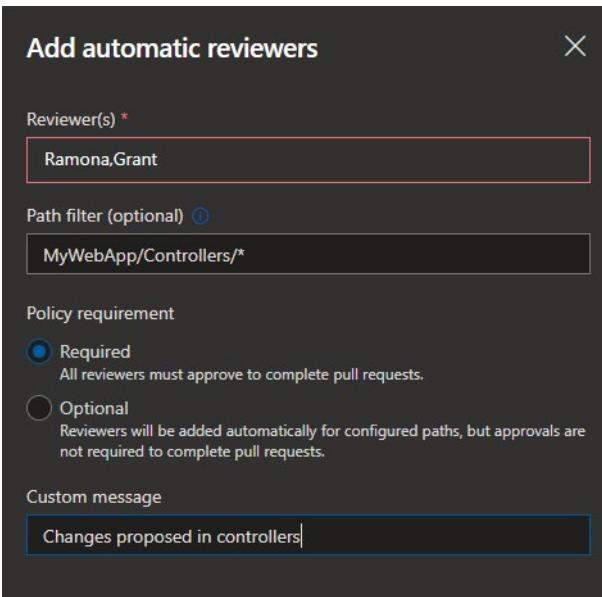
4. Use the review policy in conjunction with the comment-resolution policy. This allows you to enforce that the code review comments are resolved before the changes are accepted. The requester can take the feedback from the comment and create a new work item and resolve the changes, this at least guarantees 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:

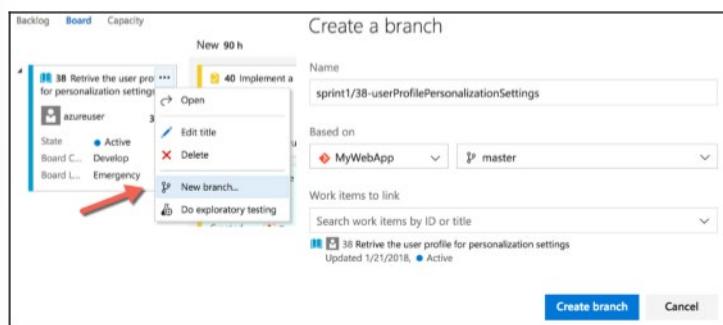


6. Select the option to automatically add code reviewers when a pull request is raised. You can map which reviewers are added based on the area of the code being changed:



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 `sprint1` folder. This is a great way to organise branches in busy environments.

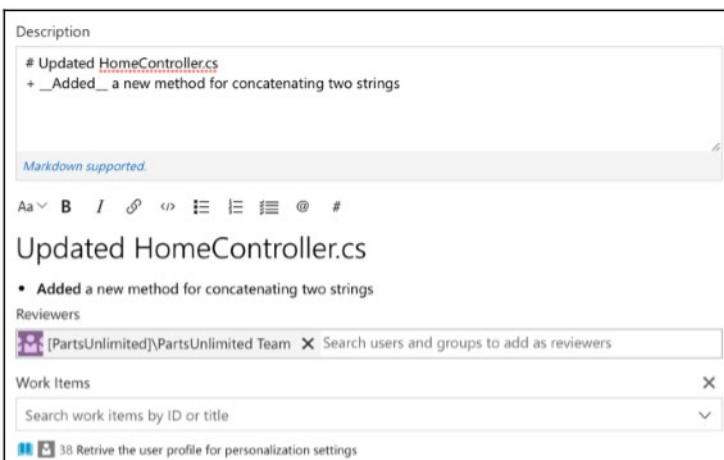
With the newly created branch selected in the web portal, edit the `HomeController.cs` file to include the following code snippet and commit the changes to the branch. In the image below you'll see that after editing the file, you can directly commit the changes by clicking the commit button.

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 drop-down.

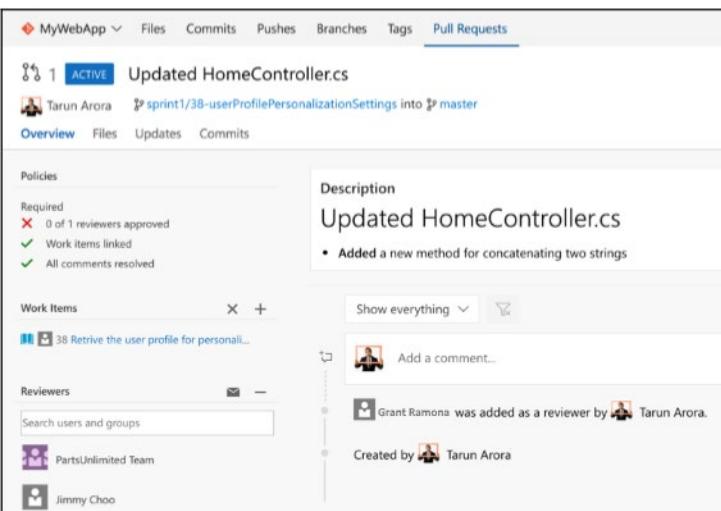


The code editor in web portal has several new features in Azure DevOps Server 2018, such as support for bracket matching and toggle white space. You can load the command palette by pressing `.`. Among many other new options, you can now toggle the file using a file mini-map, collapse and expand, as well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

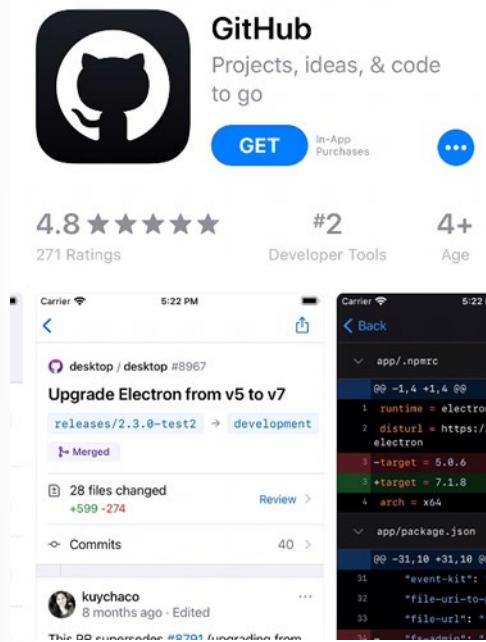
The code comments are persisted in the pull request workflow; the code comments support multiple iterations of reviews and work well with nested responses. The reviewer policy allows for a code review workflow as part of the change acceptance. This is a great way for the team to collaborate on any code 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.

GitHub mobile for pull request approvals



Using a mobile app in combination with Git is a very convenient option, particularly when urgent pull request approvals are required.

- The app can render markdown, images, and PDF files directly on the mobile device.
- Pull requests can be managed within the app, along with marking files as viewed, collapsing files.
- Comments can be added.
- Emoji short codes are rendered.

Why care about Git hooks?

Why care about Git hooks?

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.

Git hooks

Git hooks 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 Git hooks

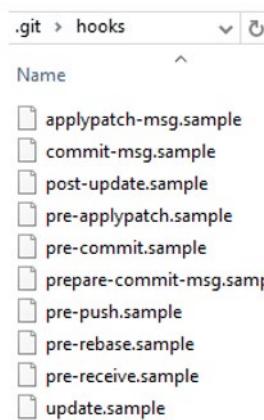
Since Git hooks simply execute the scripts on the specific event type they are called on, you can do pretty much anything with Git hooks. Some examples of where you can use hooks to enforce policies, ensure consistency, and control your environment:

- 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 Git hooks. Navigate to repo.git\hooks directory. You will find that there a bunch of samples, but they are disabled by default. For instance, if you open that folder, you will 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..."

Git hooks. 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 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 Git Hook to scan commit for keywords

How can Git hooks help with security? 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. 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 blocked list 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 can now 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 could do so by using the no-verify switch.

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

Server side service hooks with Azure Repos

So far we have looked at the client side Git Hooks 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 reference⁴](#).

Git hooks in action

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 several sample hook scripts in the repo.git\hooks directory. Since Git snare 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
- 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.

⁴ <https://docs.microsoft.com/en-gb/azure/devops/service-hooks/events?view=vsts#code-pushed>

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 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

How could Git hooks stop you from accidentally leaking 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 blocked list 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 can 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 because 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

Inner source – sometimes called “internal open source” – brings all the benefits of open-source software development inside your firewall. It opens 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 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 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.

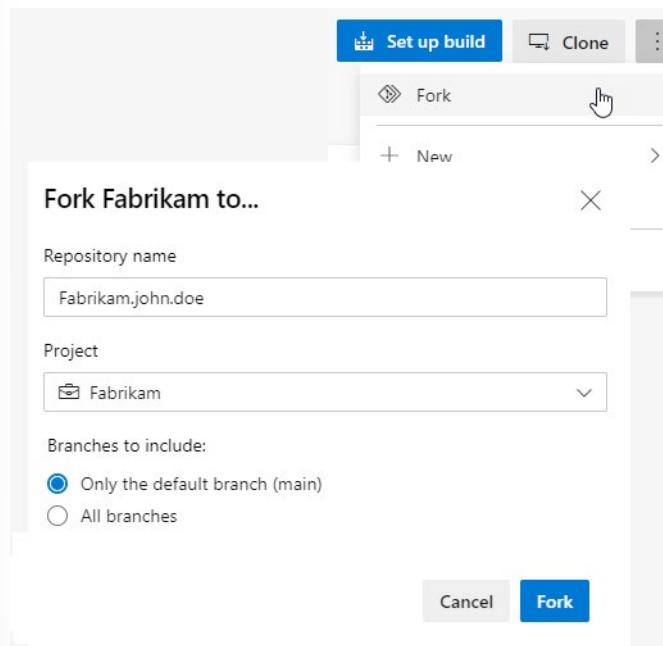
If your repository has many casual or infrequent committers (like an open-source project), we recommend the forking workflow. Typically, only core contributors to your project have direct commit rights into your repository. You should ask collaborators from outside this core set of people to work from 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 the ellipsis, then 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}
```

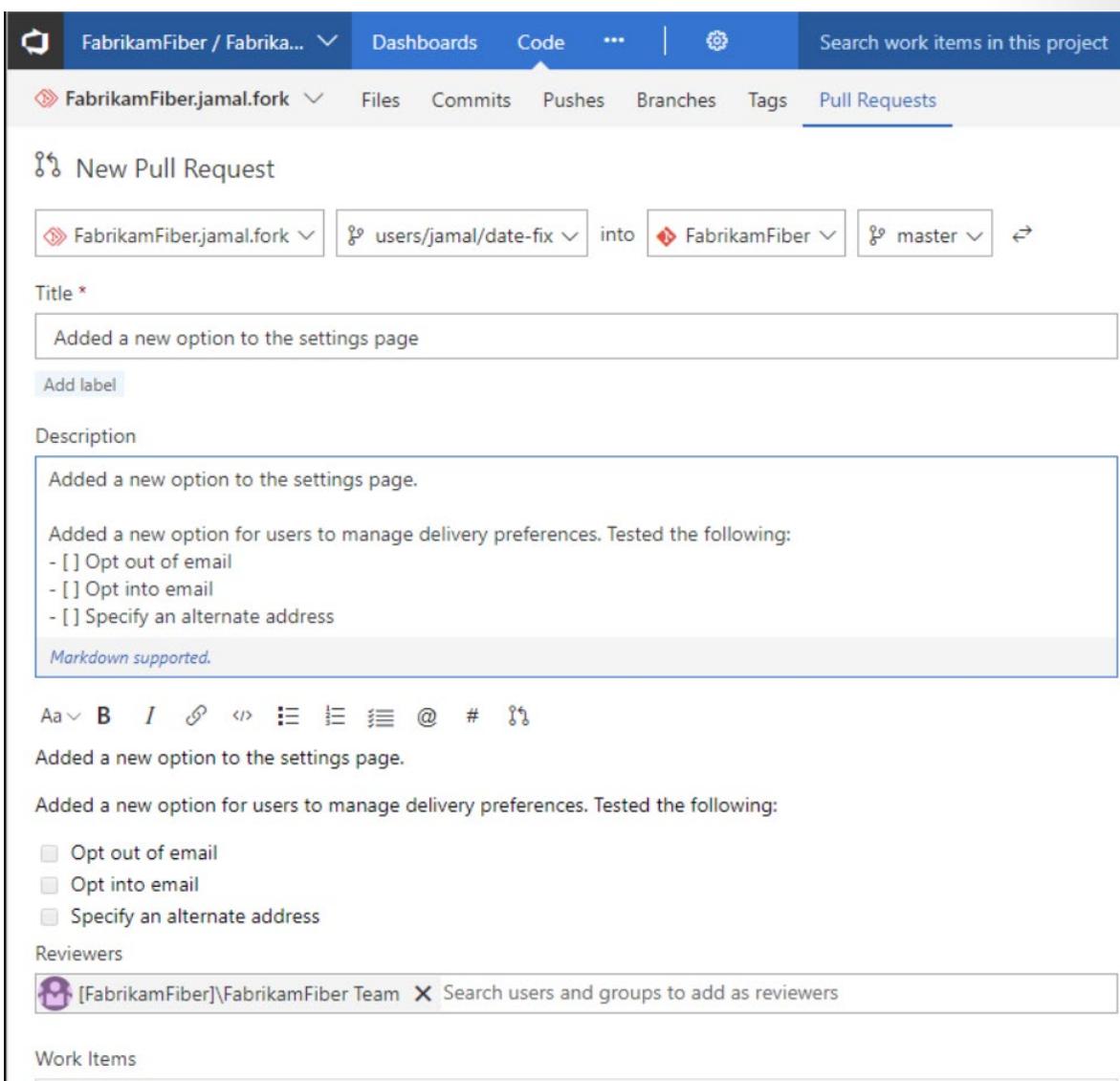
Make and push changes

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.



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 that 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 the code in that repository as a starting point for my own project.

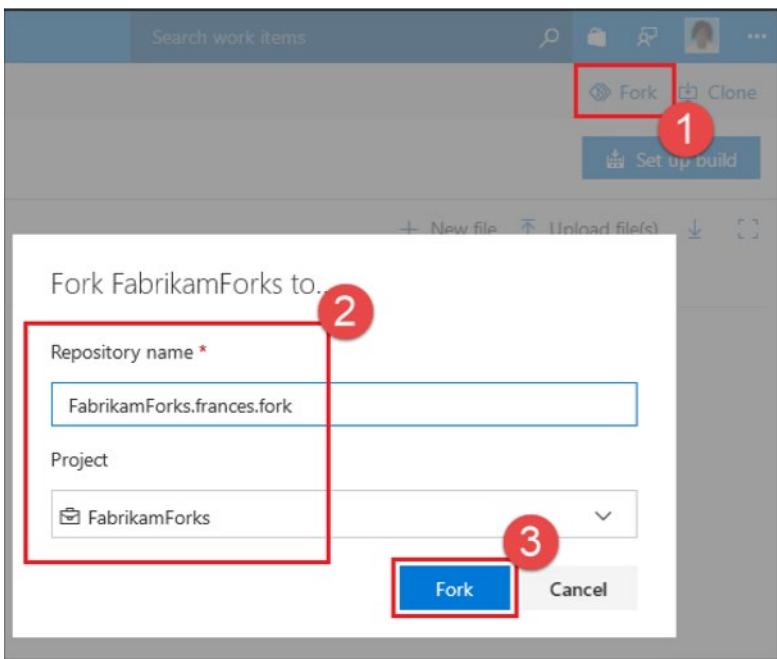
Software teams are encouraged to contribute to all projects internally, not just their own software projects. Forks are a great way to foster a culture of inner open source. Forks are a recent addition to the Azure DevOps Server-hosted Git repositories. In this recipe, we'll learn how to fork an existing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

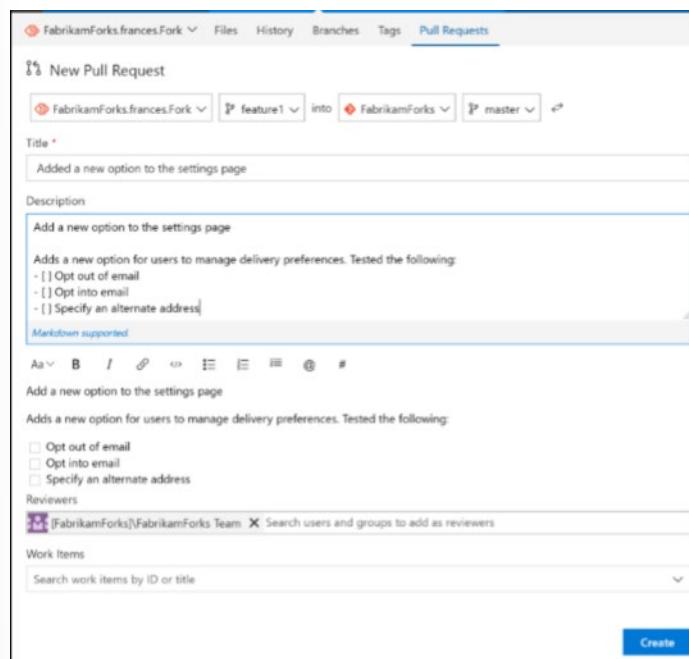
How to do it

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).



- 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}
```
- 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).
- 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 repo⁵](#)
- [Azure Repos Git Tutorial⁶](#)

⁵ <https://docs.microsoft.com/en-us/azure/devops/repos/git/clone?view=azure-devops&tabs=visual-studio>

⁶ <https://docs.microsoft.com/en-us/azure/devops/repos/git/gitworkflow?view=azure-devops>

Managing Git Repositories

Working with large repositories

Git is a great version control system, and widely adopted, but it has limitations. One of the long-standing issues has related to large repositories.

While having a local copy of repositories in a distributed version control system is useful, when large repositories are in place, that can be a significant problem.

As an example, Microsoft discovered this issue when trying to migrate a repository with over 300GB of data from an internal system to Git.

Why repositories become large

There are two basic causes for large repositories:

- Long history
- Large binary files

Shallow clone

If developers do not need all the available history in their local repositories, a good option is to implement a shallow clone. This saves both space on local development systems, and the time it takes to sync.

You can specify the depth of the clone that you want to execute:

```
git clone --depth [depth] [clone-url]
```

You can also achieve a reduced size clone by filtering branches, or by cloning only a single branch.

Git Virtual File System (GVFS)

GVFS helps with large repositories. It requires a GVFS-aware client, and a GVFS-enabled version of Git. Normal Git commands are unaffected but when you need files from the server, the GVFS subsystem works in conjunction with the normal filesystem, to download files that are required, in the background.

The GVFS client was released as open source. The protocol is a straightforward one with four endpoints that are very similar to REST endpoints.

For more information on large repositories, see: [Working with large files⁷](#)

Purging repository data

While one of the benefits of Git is its ability to hold long histories for repositories, very efficiently, there are times when you need to purge data.

The most common situations are where you want to:

- Greatly reduce the size of a repository by removing history
- Remove a large file that was accidentally uploaded
- Remove a sensitive file that should not have been uploaded

⁷ <https://docs.github.com/en/free-pro-team@latest/github/managing-large-files/working-with-large-files>

If you commit sensitive data (e.g., password, key) to Git, it can be removed from history. There are two tools that are commonly used to do this:

Filter-branch

The standard built-in Git method for removing files is to use the git filter-branch command. This command rewrites your repository history.

Note: the SHA hashes for your commits will then also change.

BFG Repo-Cleaner

BFG Repo-Cleaner is a commonly used open-source tool for deleting or “fixing” content in repositories. It is easier to use than the git filter-branch command. For a single file or set of files, use the **--delete-files** option:

```
$ bfg --delete-files file_I_should_not_have_committed
```

To find all the places that a file called passwords.txt exists in the repository, and replace all the text in it, you can execute the **--replace-text** option:

```
$ bfg --replace-text passwords.txt
```

For more information, see:

Removing files from a repository's history⁸

Removing sensitive data from a repository⁹

BFG Repo Cleaner¹⁰

⁸ <https://docs.github.com/en/free-pro-team@latest/github/managing-large-files/removing-files-from-a-repositorys-history>

⁹ <https://docs.github.com/en/free-pro-team@latest/github/authenticating-to-github/removing-sensitive-data-from-a-repository>

¹⁰ <https://rtyley.github.io/bfg-repo-cleaner/>

Lab

Version controlling with Git in Azure Repos

Lab overview

Azure DevOps supports two types of version control, Git and Team Foundation Version Control (TFVC). Here is a quick overview of the two version control systems:

- **Team Foundation Version Control (TFVC):** 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.
- **Git:** Git is a distributed version control system. Git repositories can live locally (such as on a developer's machine). 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 control operations such as history and compare without a network connection.

Git is the default version control provider for new projects. You should use Git for version control in your projects unless you have a specific need for centralized version control features in TFVC.

In this lab, you will learn how to work with branches and repositories in Azure DevOps.

Objectives

After you complete this lab, you will be able to:

- Work with branches in Azure Repos
- Work with repositories in Azure Repos

Lab duration

- Estimated time: **30 minutes**

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- **AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions¹¹**

¹¹ <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/>

Module review and takeaways

Module review questions

Review Question 1

What are the three types of branching? Select three.

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

Review Question 2

What are Git hooks?

Review Question 3

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

Answers

Review Question 1

What are the three types of branching? Select three.

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

What are Git hooks?

A mechanism that allows arbitrary code to be run before, or after, certain Git lifecycle events occur. Use Git hooks 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 5 Configuring Azure Pipelines

Module overview

Azure Pipelines

Azure Pipelines is a fully featured service that is mostly used to create cross platform CI (Continuous Integration) and CD (Continuous Deployment). 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:

- Explain the role of Azure Pipelines and its components
- Configure Agents for use in Azure Pipelines

The concept of pipelines in DevOps

The concept of pipelines in DevOps

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 provide 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 test and build your code and ship it to any target constantly and consistently.

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 Web Services (AWS).

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 macOS machines.
- Integrate with GitHub.
- Work with open-source projects.

Azure Pipelines 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 a build and/or deployment job.

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) (also known as Continuous Deployment) 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. Most jobs run on an agent. A job represents an execution boundary of a set of steps. All 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.

Stage

Stages are the major divisions in a pipeline: "build the app", "run integration tests", and "deploy to user acceptance testing" are good examples of 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 these actions are known as triggers.

Evaluate use of Microsoft-hosted versus self-hosted agents

Microsoft versus self-hosted agents

To build your code or deploy your software you generally 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 agent

If your pipelines are in Azure Pipelines, then you've got a convenient option to build and deploy using a Microsoft-hosted agent. With a Microsoft-hosted agent, maintenance and upgrades are automatically done. Each time a pipeline is run, a fresh virtual machine (instance) is provided. 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.

A Microsoft-hosted agent has job time limits.

Self-hosted agent

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 a self-hosted agent in Azure Pipelines. A self-hosted agent gives you more control to install dependent software needed for your builds and deployments.

You can install the agent on Linux, macOS, Windows machines, or 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.

A self-hosted agent does not have job time limits.

Job types

In Azure DevOps, there are four types of jobs available:

- Agent pool jobs
- Container jobs
- Deployment group jobs
- Agentless jobs

Agent pool jobs

These are the most common types of jobs. The jobs run on an agent that is part of an agent pool.

Container jobs

These jobs are similar to Agent Pool Jobs, but they run in a container on an agent that is part of an agent pool.

Deployment group jobs

These are jobs that run on systems in a deployment group.

Agentless jobs

These are jobs that run directly on the Azure DevOps server. They do not require an agent for execution.
These are also often called Server 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.

Predefined agent pool - Azure Pipelines

Azure Pipelines provides a pre-defined agent pool named **Azure Pipelines** with Microsoft-hosted agents. This will often be an easy way to run jobs without needing to configure build infrastructure. The following virtual machine images are provided by default:

- Hosted VS2019: 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: 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 20.04
- Hosted Ubuntu 18.04
- Hosted Ubuntu 16.04: 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**¹.
- Hosted macOS X Mojave 10.14
- Hosted macOS X Catalina 10.15

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 a Microsoft-hosted pool.

Pools are used to run jobs. Learn about **specifying pools for jobs**².

Note: For the most up-to-date list of Agent Pool Images and the complete list of software installed on these machines, see **Microsoft-hosted agents**³.

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.

¹ <https://docs.microsoft.com/en-us/azure/devops/pipelines/process/container-phases?view=vsts&tabs=yaml>

² <https://docs.microsoft.com/en-us/azure/devops/pipelines/process/phases?view=vsts&tabs=yaml>

³ <https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/hosted?view=azure-devops&tabs=yaml>

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.
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.

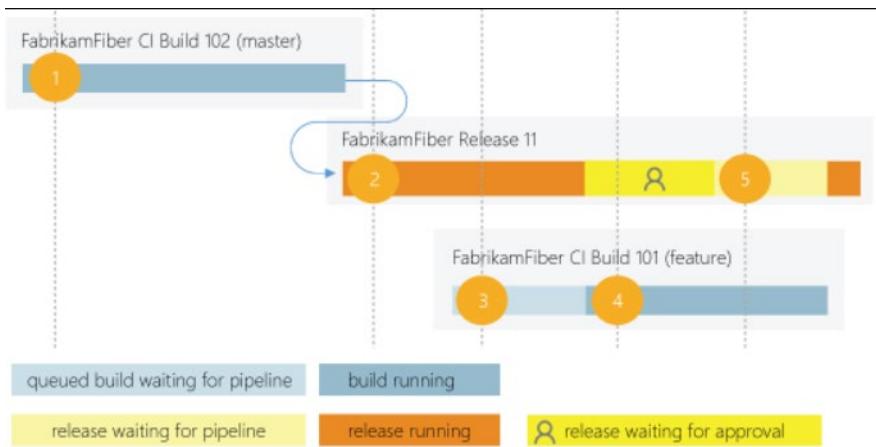
The All agent pools node in the Agent pools tab is used to control the security of all project agent pools in a project. Role memberships for individual project agent pools are automatically inherited from those of the 'All agent pools' node. By default, the following groups are added to the Administrator role of 'All agent pools': Build Administrators, Release Administrators, Project Administrators.

Pipelines and concurrency

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 could 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 requires 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 'Retention and parallel jobs' section of the Azure DevOps Organization Settings. On the left, there's a sidebar with links like 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 has tabs for 'Settings' and 'Parallel jobs', with 'Parallel jobs' being the active tab. It displays two sections: 'Microsoft-hosted' and 'Self-hosted'. Under Microsoft-hosted, it says 'Currently 0/1800 minutes are consumed' and has a 'Purchase parallel jobs' link. Under Self-hosted, it shows 2 parallel jobs, with details for 'Free parallel jobs' (1), 'Visual Studio Enterprise subscribers' (1), and 'Monthly purchases' (0). There's also a 'Change' link for monthly purchases.

Host Type	Parallel Jobs
Microsoft-hosted	Currently 0/1800 minutes are consumed Purchase parallel jobs
Self-hosted	2 Parallel jobs
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 an 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 project⁴**.

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.

⁴ <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 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 Azure Pipelines?

Yes. Visual Studio Enterprise subscribers get 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.

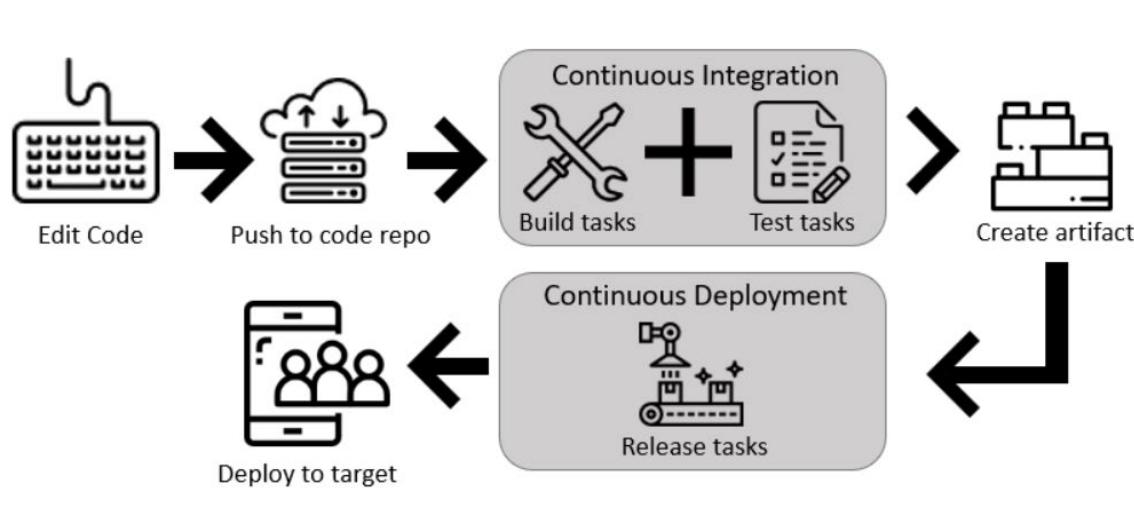
Azure Pipelines YAML versus Visual Designer

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. (These are often now referred to as "Classic Pipelines").

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 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 designer⁵**.

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

⁵ <https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started-designer?view=vsts&tabs=new-nav>

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.

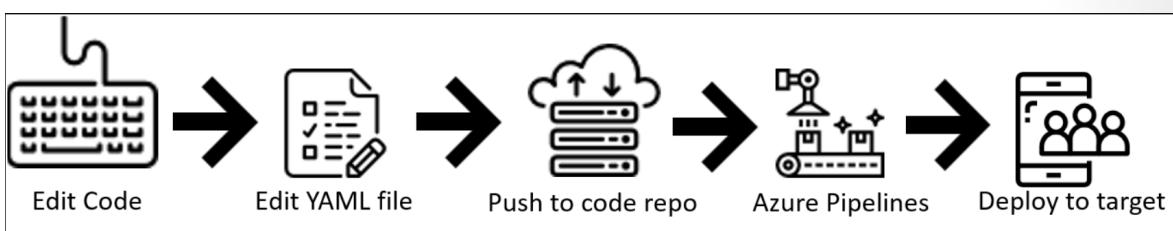
Azure DevOps offers you both experiences, you can either use YAML to define your pipelines or use the visual designer to do the same. You will however find that there are more product level investments being made to enhance the YAML pipeline experience.

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 [YAML⁶](#).

While there is a slightly higher learning curve and a higher-degree of code-orientation when defining pipelines with YAML, it is now the preferred method.

⁶ <https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started-yaml?view=vsts>

Lab

Configuring agent pools and understanding pipeline styles

Lab overview

YAML-based pipelines allow you to fully implement CI/CD as code, in which pipeline definitions reside in the same repository as the code that is part of your Azure DevOps project. YAML-based pipelines support a wide range of features that are part of the classic pipelines, such as pull requests, code reviews, history, branching, and templates.

Regardless of the choice of the pipeline style, to build your code or deploy your solution by using Azure Pipelines, you need an agent. An agent hosts compute resources that runs one job at a time. Jobs can be run directly on the host machine of the agent or in a container. You have an option to run your jobs using Microsoft-hosted agents, which are managed for you, or implementing a self-hosted agent that you set up and manage on your own.

In this lab, you will step through the process of converting a classic pipeline into a YAML-based one and running it first by using a Microsoft-hosted agent and then performing the equivalent task by using a self-hosted agent.

Objectives

After you complete this lab, you will be able to:

- implement YAML-based pipelines
- implement self-hosted agents

Lab duration

- Estimated time: **90 minutes**

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- [AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions⁷](https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/)

⁷ <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/>

Module review and takeaways

Module review questions

Review Question 1

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

Review Question 2

What is a pipeline, and why is it used?

Review Question 3

What are the two types of agents and how are they different?

Review Question 4

What is an agent pool, and why would you use it?

Review Question 5

Name two ways to configure your Azure Pipelines.

Answers

Review Question 1

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

Module 6 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 can be used to create 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.

Learning objectives

After completing this module, students will be able to:

- Explain why continuous integration matters
- Implement continuous integration using Azure Pipelines

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).



Is there “*Continuous*” in your integration?

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 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.

- **Git**¹
- **Apache Subversion**²
- **Team Foundation Version Control**³

A **package management system** is used to install, uninstall, and manage software packages.

- **NuGet**⁴
- **Node Package Manager (NPM)**⁵
- **Chocolatey**⁶
- **HomeBrew**⁷
- **RPM**⁸

A **continuous integration system** merges all developer working copies to a shared mainline several times a day.

- **Azure DevOps**⁹
- **TeamCity**¹⁰
- **Jenkins**¹¹

An **automated build process** creates a software build including compiling, packaging, and running automated tests.

- **Apache Ant**¹²
- **NAnt**¹³
- **Gradle**¹⁴

✓ 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

¹ <https://git-scm.com/>

² <https://subversion.apache.org/>

³ <https://docs.microsoft.com/en-us/azure/devops/repos/tfvc/overview?view=vsts>

⁴ <https://www.nuget.org/>

⁵ <https://www.npmjs.com/>

⁶ <https://chocolatey.org/>

⁷ <https://brew.sh/>

⁸ <http://rpm.org/>

⁹ <https://azure.microsoft.com/en-us/services/devops>

¹⁰ <https://www.jetbrains.com/teamcity/>

¹¹ <https://jenkins.io/>

¹² <http://ant.apache.org/>

¹³ <http://nant.sourceforge.net/>

¹⁴ <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 to have a global view of build results.

For more information, you can see [What is Continuous Integration ?¹⁵](#).

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 bumber formatting

The example shown below is from the build options that were configured by the ASP.NET Web Application build template:

¹⁵ <https://docs.microsoft.com/en-us/azure/devops/learn/what-is-continuous-integration>

The screenshot shows the 'Build properties' section of a pipeline configuration. The 'Build number format' is set to \$(date:yyyyMMdd)\$(rev:r).

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 Options¹⁶](#).

Build authorizations timeouts and badges

Authorization and timeouts

You can configure properties for the build job as shown below:

¹⁶ <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 Options¹⁷](#).

¹⁷ <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:

USER CAPABILITIES

Shows information about user-defined capabilities supported by this host

SYSTEM CAPABILITIES

Shows information about the capabilities provided by this host

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. It includes fields for 'Project collection' (set to 'My organization'), 'Build job timeout in minutes' (set to 60), 'Build job cancel timeout in minutes' (set to 5), and a 'Demands' table. The 'Demands' table has one row: 'HasPaymentService' with condition 'exists'. A '+ Add' button is 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 **Capabilities**¹⁸.

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:

¹⁸ <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. At the top, there are tabs for Tasks, Variables, Triggers, Options, Retention, History, Save & queue, Discard, and Summary. Below the tabs, the pipeline is named 'Pipeline' under 'Build pipeline'. It contains a 'Get sources' task (PartsUnlimited, master branch) and an 'Agent job 1' task (Run on agent). A context menu is open at the end of the pipeline, with a red arrow pointing to the 'Add an agent job' option. Other visible options in the menu include 'Add an agentless job' and 'Learn more about jobs'.

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's 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 screenshot shows the 'Parallel jobs' configuration page. It has two main sections: 'Private projects' and 'Public projects'. Under 'Private projects', there's a Microsoft-hosted section with a 'Free tier' (1 parallel job up to 1800 mins/mo) and a 'Purchase parallel jobs' button. There's also a self-hosted section with 2 parallel jobs. Under 'Public projects', there's a Microsoft-hosted section with 10 parallel jobs and a self-hosted section with 10 parallel jobs. Each section includes a 'View in-progress jobs' link.

The free tier allows for one parallel job of up to 1800 minutes per month. The self-hosted agents have 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

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 run 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 [here¹⁹](#).

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:
```

¹⁹ <https://docs.microsoft.com/en-us/azure/devops/pipelines/build/triggers?view=azure-devops&tabs=yaml>

```
# steps omitted 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 omitted for brevity

- job: B
  dependsOn: none
  steps:
    # steps omitted for brevity
```

If we want to fan out and fan in, we can do that too:

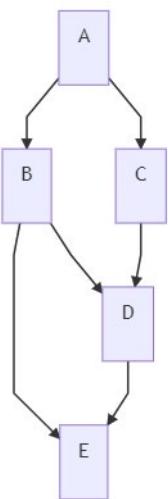
```
jobs:
- job: A
  steps:
    - script: echo 'job A'

- 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 [here²⁰](#).

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.

²⁰ <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 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: VSSBuild@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 }}
```

YAML resources

Resources in YAML represent sources of pipelines, containers, repositories, and types. For more information on Resources, [see here²¹](#).

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/*
```

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
```

²¹ <https://docs.microsoft.com/en-us/azure/devops/pipelines/process/resources?view=azure-devops&tabs=schema>

```
- 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
```

Using multiple repositories in your pipeline

You might 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.

Integrating external source control with Azure Pipelines

Source control types supported by Azure Pipelines

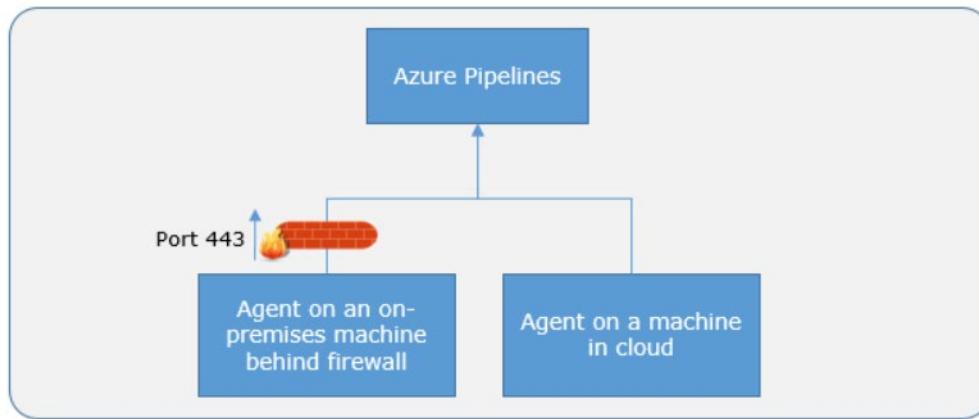
As previously discussed, Azure Pipelines offers both YAML based pipelines and the classic editor. The table below shows the repository types supported by both.

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 self-hosted 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.

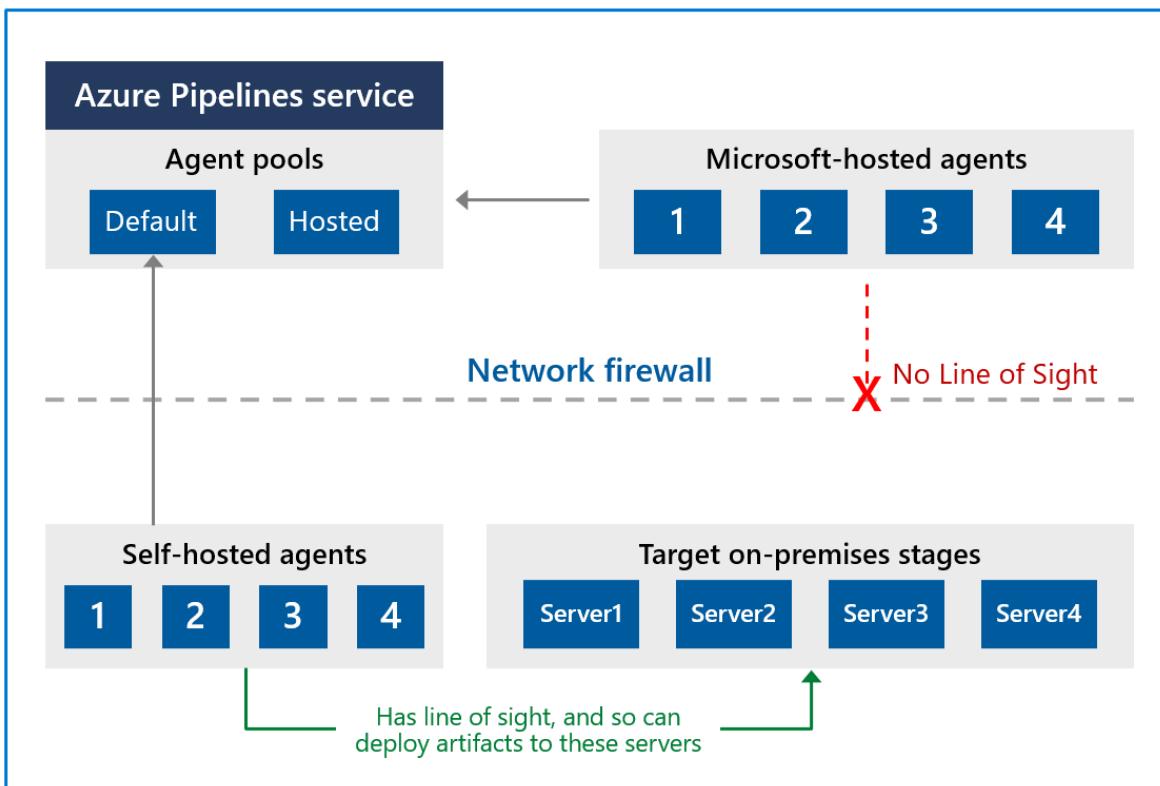
Once the job is completed, the agent discards the job-specific OAuth token and goes back to checking if there is a new job request using the listener OAuth token.

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 diagram.



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 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 versus 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 must 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 self-hosted agents can be found at:

- **Self-hosted Windows agents²²**
- **Run a self-hosted agent behind a web proxy²³**

²² <https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/v2-windows?view=vsts>

²³ <https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/proxy?view=vsts&tabs=windows>

Labs

Enabling continuous integration with Azure Pipelines

Lab overview

In this lab, you will learn how to configure continuous integration (CI) and continuous deployment (CD) for your applications using Build and Release in Azure Pipelines. This scriptable CI/CD system is both web-based and cross-platform, while also providing a modern interface for visualizing sophisticated workflows. Although we won't demonstrate all the cross-platform possibilities in this lab, it is important to point out that you can also build for iOS, Android, Java (using Ant, Maven, or Gradle) and Linux.

Objectives

After you complete this lab, you will be able to:

- Create a basic build pipeline from a template
- Track and review a build
- Invoke a continuous integration build

Lab duration

- Estimated time: **60 minutes**

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions²⁴

Integrating external source control with Azure Pipelines

Lab overview

With the introduction of Azure DevOps, Microsoft is offering developers a new continuous integration/continuous delivery (CI/CD) service called Azure Pipelines that enables you to continuously build, test, and deploy to any platform or cloud. It has cloud-hosted agents for Linux, macOS, and Windows; powerful workflows with native container support; and flexible deployments to Kubernetes, VMs, and serverless environments.

Azure Pipelines provides unlimited CI/CD minutes and 10 parallel jobs to every GitHub open source project for free. All open source projects run on the same infrastructure that our paying customers use. That means you'll have the same fast performance and high quality of service. Many of the top open

²⁴ <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/>

source projects are already using Azure Pipelines for CI/CD, such as Atom, CPython, Pipenv, Tox, Visual Studio Code, and TypeScript—and the list is growing every day.

In this lab, you'll see how easy it is to set up Azure Pipelines with your GitHub projects and how you can start seeing benefits immediately.

Objectives

After you complete this lab, you will be able to:

- Install Azure Pipelines from the GitHub Marketplace
- Integrate a GitHub project with an Azure DevOps pipeline
- Track pull requests through the pipeline

Lab duration

- Estimated time: **60 minutes**

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- [AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions²⁵](https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/)

²⁵ <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/>

Module review and takeaways

Module review questions

Review Question 1

Name the four pillars of continuous integration.

Review Question 2

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?

Review Question 3

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

Name the four pillars of continuous integration.

Continuous Integration relies on four key elements for successful implementation: a Version Control System, Packet 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 7 Managing Application Configuration 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 configuration and secrets
- Integrate Azure Key Vault with a pipeline

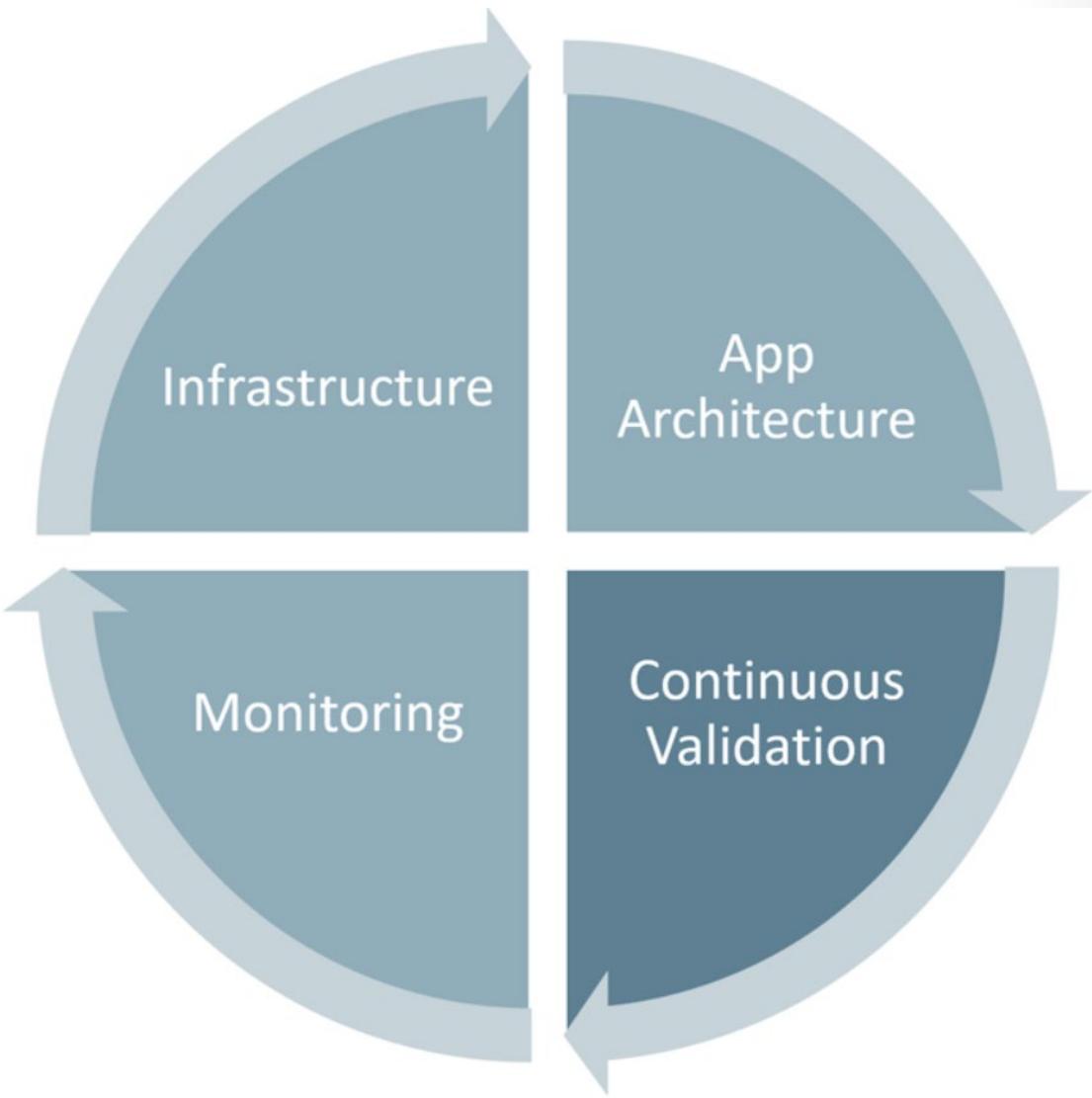
Introduction to security

Introduction to security

While a 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 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.

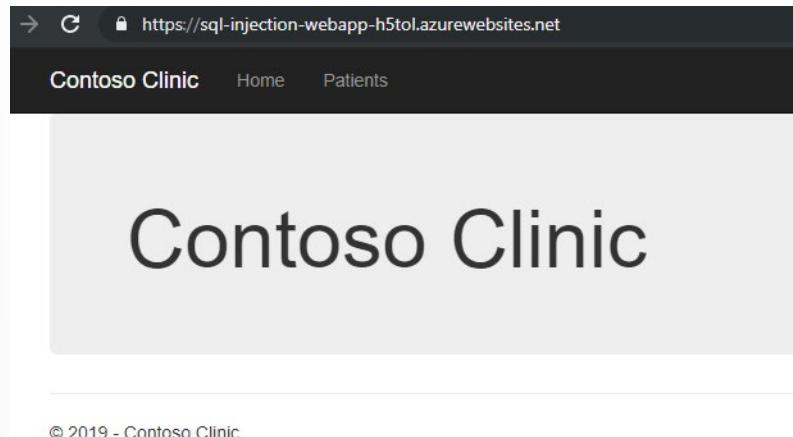
An SQL Injection vulnerability may affect any website or web application that uses an SQL database such as MySQL, Oracle, SQL Server, or others. Criminals may use it to gain unauthorized access to, delete, or alter your sensitive data: customer information, personal data, trade secrets, intellectual property, and more. SQL Injection attacks are one of the oldest, most prevalent, and most dangerous web application

vulnerabilities. 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 will simulate a 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 will 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 a SQL statement 'AND FirstName = 'Kim'-- in the search box. You will 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'--				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

order by SSN--						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

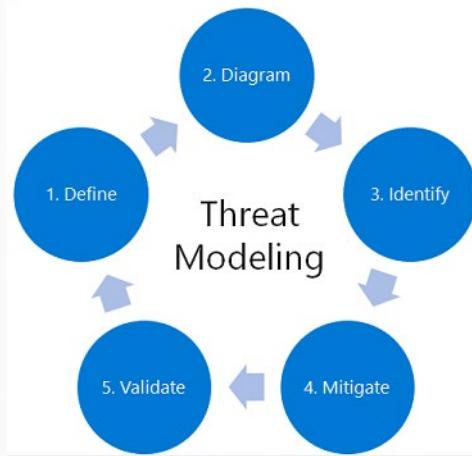
The Azure security center team has other [playbooks](#)² you can look at to learn how vulnerabilities are exploited to trigger a virus attack and a DDoS attack.

² <https://azure.microsoft.com/en-gb/blog/enhance-your-devsecops-practices-with-azure-security-center-s-newest-playbooks/>

Implement a secure development process

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. It also cannot be added at the end of a project. Security must 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 it is 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

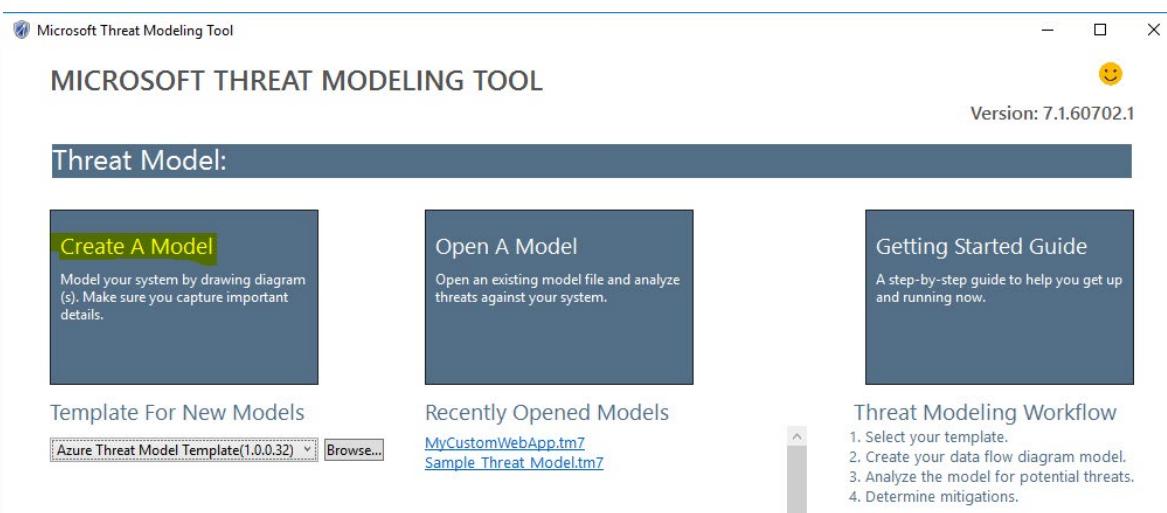
How to do it

1. Launch the Microsoft Threat Modelling Tool and choose the option to Create a Model.

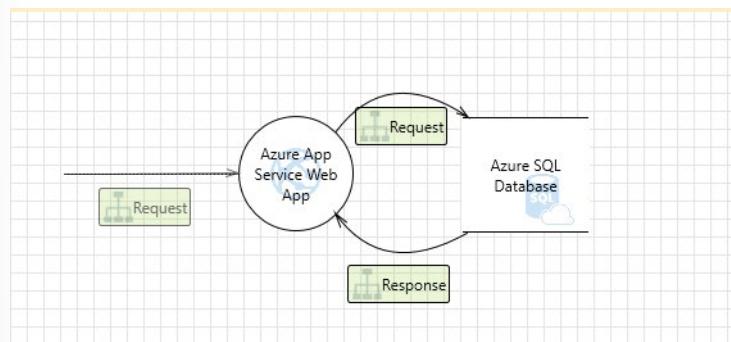
³ <https://docs.microsoft.com/en-us/azure/security/azure-security-threat-modeling-tool-feature-overview>

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.

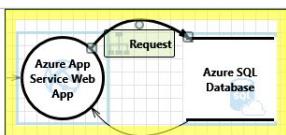


3. From the toolbar menu select View -> Analysis view, the analysis view will show you a full list of threats categorised by severity.

Short Description	Description	Interaction	Possible Mitigation(s)
A user subject gains increased capability or privilege by t...	Due to poorly configured account policies, adversary...	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 S...	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 ca...	An adversary can read confidential data due to we...	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 ca...	An adversary having access to the storage contain...	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 privileg...	Request	It is recommended to review permission and role assignments to ensure the users are granted...
Repudiation threats involve an adversary denying that so...	An adversary can deny actions performed on Azure S...	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...	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...	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...	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. 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 long term persistent access...	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. 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 S...	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. Restrict access to Azure App Service to selected networks (e.g. IP whitelisting, VNET integration).

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

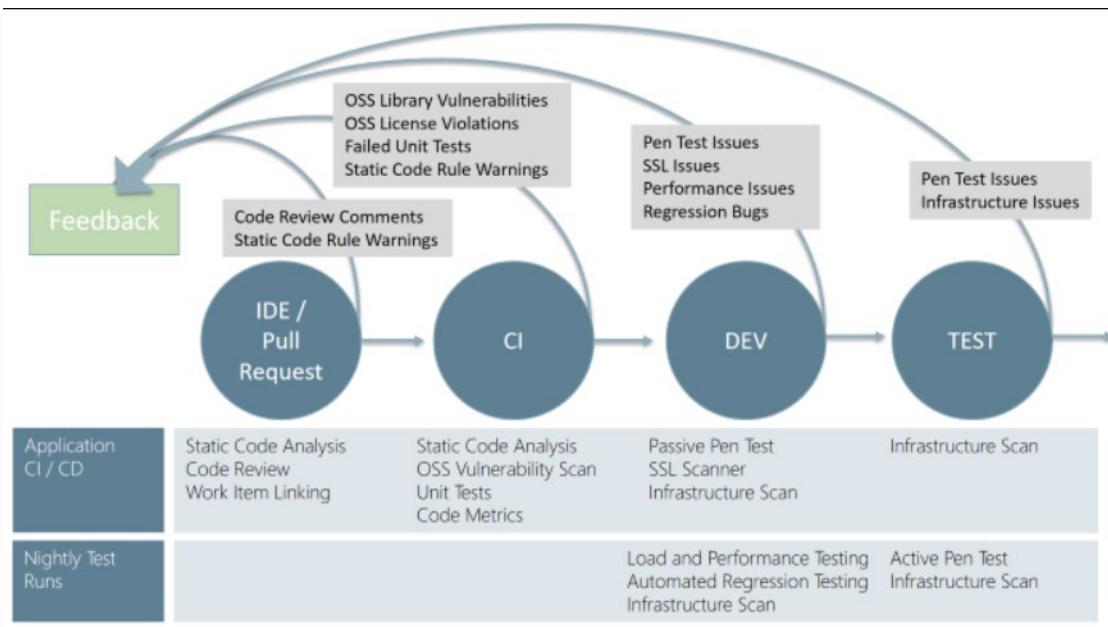
There's more

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.

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.

In a later module, we will discuss the integration of several useful and commonly used security and compliance tools.

Rethinking application configuration data

Rethinking application configuration data

Configuration information in files

Most application runtime environments include configuration information that is 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 is 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.

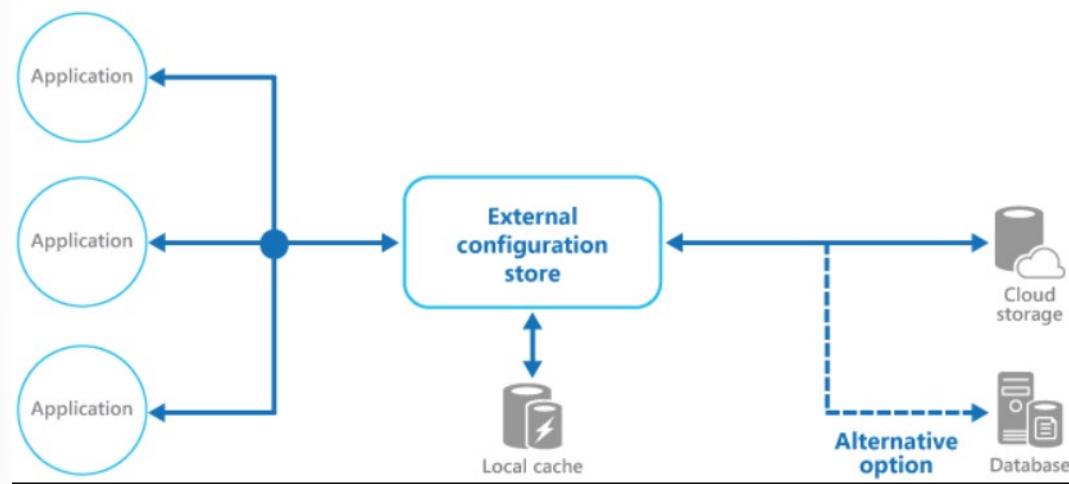
External configuration store patterns

These patterns store the configuration information in an external location 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 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 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 the centrally stored settings.
- 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 Vault⁷](https://docs.microsoft.com/en-us/azure/key-vault/key-vault-overview).

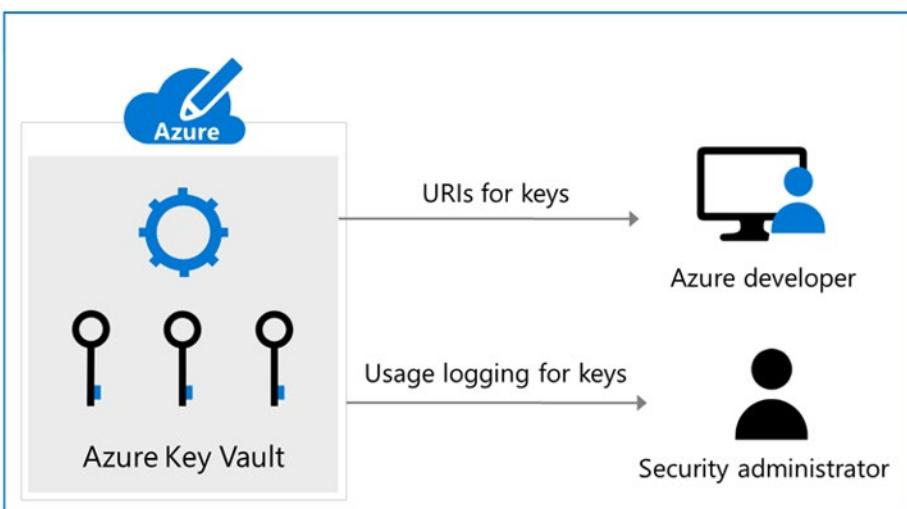
⁷ <https://docs.microsoft.com/en-us/azure/key-vault/key-vault-overview>

Manage secrets, tokens, and certificates

Manage secrets, tokens and 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 can 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 secrets

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 can 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.

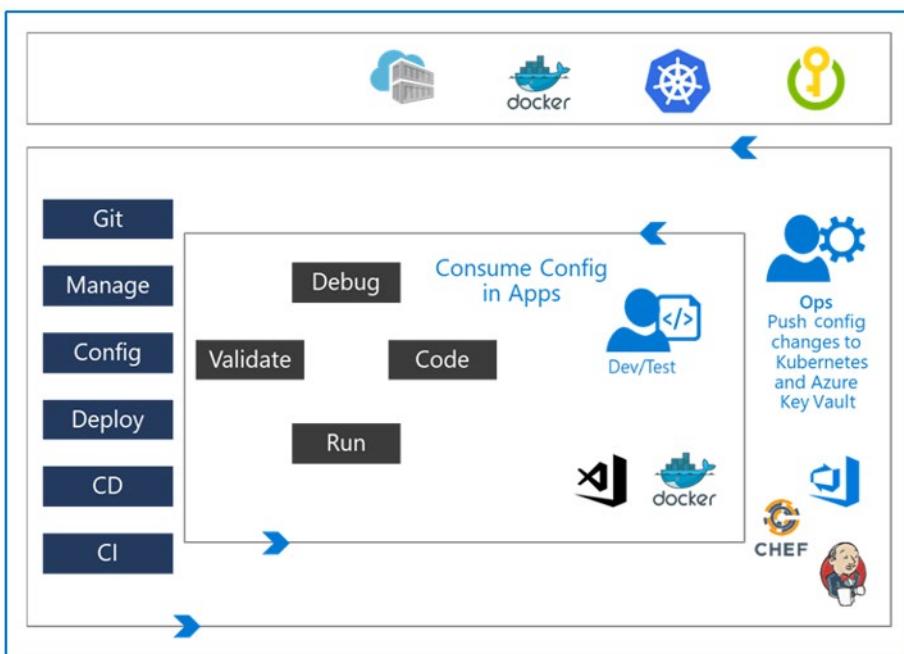
Integrate with other Azure services

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 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.



Integrating with identity management systems

Integrating GitHub with single sign-on (SSO)

To use SSO, you need to connect your identity provider to GitHub at the organization level.

GitHub offers both **SAML** and **SCIM** support.

Provider	Available Support
Active Directory Federation Services (ADFS)	SAML
Azure Active Directory (Azure AD)	SAML and SCIM
Okta	SAML and SCIM
OneLogin	SAML and SCIM
PingOne	SAML
Shibboleth	SAML

For more information, see:

[Enforcing SAML single sign-on for your organization⁸](#)

[About SCIM⁹](#)

Service Principals

Azure AD offers a variety of mechanisms for authentication. In DevOps projects though, one of the most important is the use of Service Principals.

Azure AD applications

Applications are registered with an Azure AD tenant within Azure Active Directory. Registering an application creates an identity configuration. You also determine who can use it:

- Accounts in the same organizational directory
- Accounts in any organizational directory
- Accounts in any organizational directory and Microsoft Accounts (personal)
- Microsoft Accounts (Personal accounts only)

⁸ <https://docs.github.com/en/free-pro-team@latest/github/setting-up-and-managing-organizations-and-teams/enforcing-saml-single-sign-on-for-your-organization>

⁹ <https://docs.github.com/en/free-pro-team@latest/github/setting-up-and-managing-organizations-and-teams/about-scim>

The user-facing display name for this application (this can be changed later).

Supported account types

Who can use this application or access this API?

- Accounts in this organizational directory only (SQL Down Under Pty Ltd only - Single tenant)
 Accounts in any organizational directory (Any Azure AD directory - Multitenant)
 Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)
 Personal Microsoft accounts only

[Help me choose...](#)

Redirect URI (optional)

We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

Web e.g. https://myapp.com/auth

All applications Owned applications

 Start typing a name or Application ID to filter these results

Display name	Application (client) ID	Created on	Certificates & secrets
AC ACMOrders	02744c0e5caf	11/6/2019	<input checked="" type="checkbox"/> Current

Client secret

Once the application is created, you then should create at least one client secret for the application.

Client secrets

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

 New client secret

Description	Expires	Value
ACM Orders Download	12/31/2299	mXL*****

Grant permissions

The application identity can then be granted permissions within services and resources that trust Azure Active Directory.

Service principal

To access resources, an entity must be represented by a security principal. To connect, the entity must know:

- TenantID
- ApplicationID

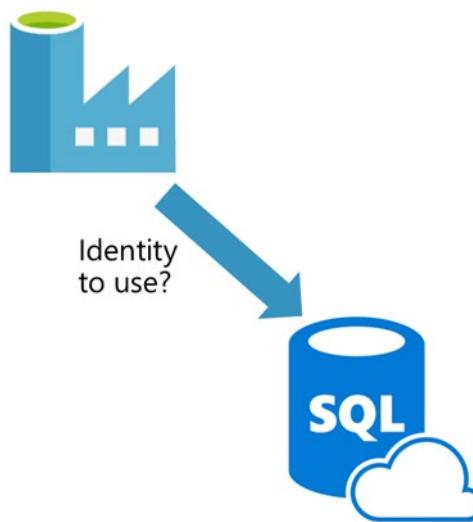
- Client Secret

For more information on Service Principals, see: [App Objects and Service Principals¹⁰](#)

Managed service identities

Another authentication mechanism offered by Azure AD is Managed service identities (MSIs).

Imagine that you need to connect from an Azure Data Factory (ADF) to an Azure SQL Database. What identity should ADF present to the database?



The traditional answer would have been to use SQL Authentication with a username and password, but this leaves yet another credential that needs to be managed on an ongoing basis.

Identity of the service

Instead, many Azure services expose their own identity. This isn't an identity that you need to manage e.g., you don't need to worry about password policies, etc.

You can assign permissions to that identity, as with any other Azure AD identity.

In the ADF example, you can add the ADF MSI as an Azure SQL Database user and add it to roles within the database.

Managed identity types

There are two types of managed identities:

- System-assigned - these are the types of identities described above. Many, but not all, services expose these identities.
- User-assigned - you can create a managed identity as an Azure resource. It can then be assigned to one or more instances of a service.

For more information, see: [What are managed identities for Azure resources?¹¹](#)

¹⁰ <https://docs.microsoft.com/en-us/azure/active-directory/develop/app-objects-and-service-principals>

¹¹ <https://docs.microsoft.com/en-us/azure/active-directory/managed-identities-azure-resources/overview>

Implementing application configuration

Azure App Configuration overview

Azure App Configuration is a service for central management of application settings and feature flags. Modern programs include distributed components, each that need their own settings. This is particularly common with microservice based applications, and with serverless applications.

Distributed configuration settings can lead to hard-to-troubleshoot deployment errors. Azure App Configuration service is used to store all the settings for your application and secure their access in one place.

Azure App Configuration service provides the following features:

- A fully managed service that can be set up in minutes
- Flexible key representations and mappings
- Tagging with labels
- Point-in-time replay of settings
- Dedicated UI for feature flag management
- Comparison of two sets of configurations on custom-defined dimensions
- Enhanced security through Azure-managed identities
- Complete data encryption, at rest or in transit
- Native integration with popular frameworks

App Configuration complements Azure Key Vault, which is used to store application secrets. App Configuration makes it easier to implement the following scenarios:

- Centralize management and distribution of hierarchical configuration data for different environments and geographies
- Dynamically change application settings without the need to redeploy or restart an application
- Control feature availability in real-time

Use App Configuration

The easiest way to add an App Configuration store to your application is through one of the client libraries that Microsoft provides. Based on the programming language and framework, the following best methods are available to you.

Programming language and framework	How to connect
.NET Core and ASP.NET Core	App Configuration provider for .NET Core
.NET Framework and ASP.NET	App Configuration builder for .NET
Java Spring	App Configuration client for Spring Cloud
Others	App Configuration REST API

Key-value pairs

Azure App Configuration stores configuration data as key-value pairs.

Keys

Keys serve as the name for key-value pairs and are used to store and retrieve corresponding values. It's a common practice to organize keys into a hierarchical namespace by using a character delimiter, such as / or :. Use a convention that's best suited for your application. App Configuration treats keys as a whole. It doesn't parse keys to figure out how their names are structured or enforce any rule on them.

Keys stored in App Configuration are case-sensitive, Unicode-based strings. The keys *app1* and *App1* are distinct in an App Configuration store. Keep this in mind when you use configuration settings within an application because some frameworks handle configuration keys case-insensitively.

You can use any Unicode character in key names entered into App Configuration except for *, ,, and \\. These characters are reserved. If you need to include a reserved character, you must escape it by using \{Reserved Character\}. There's a combined size limit of 10,000 characters on a key-value pair. This limit includes all characters in the key, its value, and all associated optional attributes. Within this limit, you can have many hierarchical levels for keys.

Design key namespaces

There are two general approaches to naming keys used for configuration data: flat or hierarchical. These methods are similar from an application usage standpoint, but hierarchical naming offers several advantages:

- Easier to read. Instead of one long sequence of characters, delimiters in a hierarchical key name function as spaces in a sentence.
- Easier to manage. A key name hierarchy represents logical groups of configuration data.
- Easier to use. It's simpler to write a query that pattern-matches keys in a hierarchical structure and retrieves only a portion of configuration data.

Below are some examples of how you can structure your key names into a hierarchy:

- Based on component services

```
AppName:Service1:ApiEndpoint  
AppName:Service2:ApiEndpoint
```

- Based on deployment regions

```
AppName:Region1:DbEndpoint  
AppName:Region2:DbEndpoint
```

Label keys

Key values in App Configuration can optionally have a label attribute. Labels are used to differentiate key values with the same key. A key *app1* with labels *A* and *B* forms two separate keys in an App Configuration store. By default, the label for a key value is empty, or null.

Label provides a convenient way to create variants of a key. A common use of labels is to specify multiple environments for the same key:

```
Key = AppName:DbEndpoint & Label = Test  
Key = AppName:DbEndpoint & Label = Staging  
Key = AppName:DbEndpoint & Label = Production
```

Version key values

App Configuration doesn't version key values automatically as they're modified. Use labels as a way to create multiple versions of a key value. For example, you can input an application version number or a Git commit ID in labels to identify key values associated with a particular software build.

Query key values

Each key value is uniquely identified by its key plus a label that can be `null`. You query an App Configuration store for key values by specifying a pattern. The App Configuration store returns all key values that match the pattern and their corresponding values and attributes.

Values

Values assigned to keys are also Unicode strings. You can use all Unicode characters for values. There's an optional user-defined content type associated with each value. Use this attribute to store information, for example an encoding scheme, about a value that helps your application to process it properly.

Configuration data stored in an App Configuration store, which includes all keys and values, is encrypted at rest and in transit. App Configuration isn't a replacement solution for Azure Key Vault. Don't store application secrets in it.

App configuration feature management

Feature management is a modern software-development practice that decouples feature release from code deployment and enables quick changes to feature availability on demand.

Feature Flags are discussed in a later module in this course, but at this point in the course it is worth noting that Azure App Configuration Service can be used to store and manage feature flags. (These are also known as feature toggles, feature switches, and other names).

Basic concepts

Here are several new terms related to feature management:

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

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

- An application that makes use of feature flags.
- A separate repository that stores the feature flags and their current states.

How these components interact is illustrated in the following examples.

Feature flag usage in code

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

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

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

```
bool featureFlag = true;
```

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

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

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

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

Feature flag declaration

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

When a feature flag has multiple filters, the filter list is traversed in order until one of the filters determines the feature should be enabled. At that point, the feature flag is *on*, and any remaining filter results are skipped. If no filter indicates the feature should be enabled, the feature flag is *off*.

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

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

```
}
```

Feature flag repository

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

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

Lab

Integrating Azure Key Vault with Azure DevOps

Lab overview

Azure Key Vault provides secure storage and management of sensitive data, such as keys, passwords, and certificates. Azure Key Vault includes supports for hardware security modules, as well as a range of encryption algorithms and key lengths. By using Azure Key Vault, you can minimize the possibility of disclosing sensitive data through source code, which is a common mistake made by developers. Access to Azure Key Vault requires proper authentication and authorization, supporting fine grained permissions to its content.

In this lab, you will see how you can integrate Azure Key Vault with an Azure DevOps pipeline by using the following steps:

- create an Azure Key vault to store a MySQL server password as a secret.
- create an Azure service principal to provide access to secrets in the Azure Key vault.
- configure permissions to allow the service principal to read the secret.
- configure pipeline to retrieve the password from the Azure Key vault and pass it on to subsequent tasks.

Objectives

After you complete this lab, you will be able to:

- Create an Azure Active Directory (Azure AD) service principal.
- Create an Azure key vault.
- Track pull requests through the Azure DevOps pipeline.

Lab duration

- Estimated time: **40 minutes**

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- **AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions¹²**

¹² <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/>

Module review and takeaways

Module review questions

Review Question 1

What are the five stages of threat modeling?

Review Question 2

What is the Azure Key Vault and why would you use it?

Answers

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.

What is the Azure Key Vault and why would you 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 8 Implementing Continuous Integration with GitHub Actions

Module overview

Module Overview

GitHub Actions are the primary mechanism for automation within GitHub. They can be used for a wide variety of purposes, but one of the most common is to implement Continuous Integration.

Learning Objectives

After completing this module, students will be able to:

- Create and work with GitHub Actions and Workflows
- Implement Continuous Integration with GitHub Actions

GitHub Actions

What are actions?

Actions are the mechanism used to provide workflow automation within the GitHub environment.

They are often used to build continuous integration (CI) and continuous deployment (CD) solutions. However, they can be used for a wide variety of tasks:

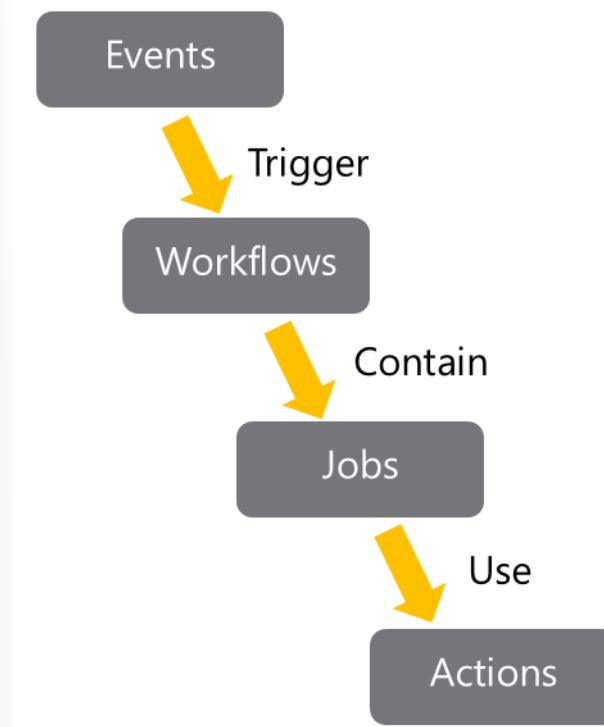
- Automated testing
- Automatically responding to new issues, mentions
- Triggering code reviews
- Handling pull requests
- Branch management

They are defined in YAML and reside within GitHub repositories.

Actions are executed on “runners”, either hosted by GitHub, or self-hosted.

Contributed actions can be found in the GitHub Marketplace, see: [Marketplace Actions¹](#)

Actions flow



GitHub tracks events that occur. Events can trigger the start of workflows. Workflows can also start on cron-based schedules and can be triggered by events outside of GitHub. They can be manually triggered.

Workflows are the unit of automation. They contain Jobs.

¹ <https://github.com/marketplace?type=actions>

Jobs use Actions to get work done.

Workflows

Workflows define the automation required.

They detail the events that should trigger the workflow, and they defined the jobs that should run when the workflow is triggered.

Within the job, they define the location that the actions will run in i.e., which runner to use.

Workflows are written in YAML and live within a GitHub repository, at the location **.github/workflows**

Example workflow:

```
# .github/workflows/build.yml
name: Node Build

on: [push]

jobs:
  mainbuild:
    runs-on: ${{ matrix.os }}

    strategy:
      matrix:
        node-version: [12.x]
        os: [windows-latest]

    steps:
      - uses: actions/checkout@v1
      - name: Run node.js on latest Windows
        uses: actions/setup-node@v1
        with:
          node-version: ${{ matrix.node-version }}
      - name: Install NPM and build
        run:
          npm ci
          npm run build
```

You can find a set of starter workflows here: [Starter Workflows²](#)

You can see the allowable syntax for workflows here: [Workflow syntax for GitHub Actions³](#)

Standard workflow syntax elements

Workflows include several standard syntax elements.

- **name:** is the name of the workflow. This is optional but is highly recommended. It appears in several places within the GitHub UI.
- **on:** is the event or list of events that will trigger the workflow.

² <https://github.com/actions/starter-workflows>

³ <https://docs.github.com/en/free-pro-team@latest/actions/reference/workflow-syntax-for-github-actions>

- **jobs:** is the list of jobs to be executed. Workflows can contain one or more jobs.
- **runs-on:** tells Actions which runner to use.
- **steps:** is the list of steps for the job. Steps within a job execute on the same runner.
- **uses:** tells Actions which predefined action needs to be retrieved. For example, you might have an action that installs node.js.
- **run:** tells the job to execute a command on the runner. For example, you might execute an NPM command.

You can see the allowable syntax for workflows here: [Workflow syntax for GitHub Actions⁴](#)

Events

Events are implemented by the **on** clause in a workflow definition.

There are several types of events that can trigger workflows.

Scheduled events

With this type of trigger, a cron schedule needs to be provided.

```
on:  
  schedule:  
    - cron: '0 8-17 * * 1-5'
```

cron schedules are based on 5 values:

- minute (0 - 59)
- hour (0 - 23)
- day of the month (1 - 31)
- month (1 - 12)
- day of the week (0 - 6)

Aliases for the months are JAN-DEC and for days of the week are SUN-SAT.

A wild card means any. (Note that * is a special value in YAML so the cron string will need to be quoted)

So, in the example above, the schedule would be on the hour 8AM - 5PM Monday to Friday.

Code events

Most actions will be triggered by code events. These occur when an event of interest occurs in the repository.

```
on:  
  pull_request
```

The above event would fire when a pull request occurs.

⁴ <https://docs.github.com/en/free-pro-team@latest/actions/reference/workflow-syntax-for-github-actions>

```
on:  
  [push, pull_request]
```

The above event would fire when either a push or a pull request occurs.

```
on:  
  pull_request:  
    branches:  
      - develop
```

The above event shows how to be specific about the section of the code that is relevant. In this case, it will fire when a pull request is made in the develop branch.

Manual events

There is a special event that is used to manually trigger workflow runs. For this, you should use the **workflow_dispatch** event. To use this, your workflow must be in the default branch for the repository.

Webhook events

Workflows can be executed when a GitHub webhook is called.

```
on:  
  gollum
```

This event would fire when someone updates (or first creates) a Wiki page.

External events

Events can be on **repository_despatch**. That allows events to fire from external systems.

For more information on events, see **Events that trigger workflows**⁵

Jobs

Workflows contain one or more jobs. A job is a set of steps that will be run in order on a runner.

Steps within a job execute on the same runner and share the same filesystem.

The logs produced by jobs are searchable and artifacts produced can be saved.

Jobs with dependencies

By default, if a workflow contains multiple jobs, they run in parallel.

```
jobs:  
  startup:  
    runs-on: ubuntu-latest  
    steps:  
      - run: ./setup_server_configuration.sh  
  build:
```

⁵ <https://docs.github.com/en/free-pro-team@latest/actions/reference/events-that-trigger-workflows>

```
steps:
  - run: ./build_new_server.sh
```

Sometimes you might need one job to wait for another job to complete. You can do that by defining dependencies between the jobs.

```
jobs:
  startup:
    runs-on: ubuntu-latest
    steps:
      - run: ./setup_server_configuration.sh
  build:
    needs: startup
    steps:
      - run: ./build_new_server.sh
```

Note: if the startup job in the example above fails, the build job will not execute.

For more information on job dependencies, see the section **Creating Dependent Jobs at Managing complex workflows⁶**

Runners

When you execute jobs, the steps execute on a Runner. The steps can be the execution of a shell script, or the execution of a predefined Action.

GitHub provides several hosted runners, to avoid you needing to spin up your own infrastructure to run actions.

At present, the maximum duration of a job is 6 hours, and for a workflow is 72 hours.

For JavaScript code, you have implementations of node.js on:

- Windows
- MacOS
- Linux

If you need to use other languages, a Docker container can be used. At present, the Docker container support is only Linux based.

These options allow you to write in whatever language you prefer. JavaScript actions will be faster (no container needs to be used), and the runtime is more versatile. The GitHub UI is also better for working with JavaScript actions.

Self-hosted runners

If you need different configurations to the ones provided, you can create a self-hosted runner.

GitHub has published the source code for self-hosted runners as open-source, and you can find it here: <https://github.com/actions/runner>

This allows you to completely customize the runner, however, you then need to maintain (patch, upgrade) the runner system.

⁶ <https://docs.github.com/en/free-pro-team@latest/actions/learn-github-actions/managing-complex-workflows>

Self-hosted runners can be added at different levels within an enterprise:

- Repository-level (single repository)
- Organizational-level (multiple repositories in an organization)
- Enterprise-level (multiple organizations across an enterprise)

GitHub strongly recommends that you do not use self-hosted runners in public repos.

Doing this would be a significant security risk, as you would allow someone (potentially) to run code on your runner, within your network.

For more information on self-hosted runners, see: [About self-hosted runners⁷](#)

Console output from actions

Actions will often produce console output. You do not need to connect directly into the runners to retrieve that output.

The console output from actions is available directly from within the GitHub UI.

To see the output, select **Actions** on the top repository menu, to see a list of workflows that have executed. Next click on the name of the job to see the output of the steps.

The screenshot shows a GitHub repository page for 'greglow-sdu / hello-github-actions'. The 'Actions' tab is selected. A workflow named 'Update main.yml' is shown, triggered by 'on: push'. The 'Hello world action' step is expanded, showing its four sub-steps: 'Set up job', 'Run actions/checkout@v1', 'Run ./action-a', and 'Complete job', all of which succeeded.

Console output can be helpful in debugging. If it isn't sufficient, you can also enable additional logging. See: [Enabling debug logging⁸](#)

Release management for actions

While you might be happy to just retrieve the latest version of an action, there are many situations where you might want a specific version of an action.

You can request a specific release of an action in several ways:

⁷ <https://docs.github.com/en/free-pro-team@latest/actions/hosting-your-own-runners/about-self-hosted-runners>

⁸ <https://docs.github.com/en/free-pro-team@latest/actions/managing-workflow-runs/enabling-debug-logging>

Tags

Tags allow you to specify the precise versions that you want to work with.

```
steps:  
  - uses: actions/install-timer@v2.0.1
```

SHA based hashes

You can specify a requested SHA based hash for an action. This ensures that the action has not changed. However, the downside to this is that you also won't receive updates to the action automatically either.

```
steps:  
  - uses: actions/install-timer@327239021f7cc39fe7327647b213799853a9eb98
```

Branches

A common way to request actions is to refer to the branch that you want to work with. You'll then get the latest version from that branch. That means you'll benefit from updates, but it also increases the chance of the code breaking.

```
steps:  
  - uses: actions/install-timer@develop
```

Demonstration: testing an action

GitHub offers several learning tools for actions.

It would be useful at this point in the course, to walk students through the most basic actions example:

GitHub Actions: hello world⁹

In this learning lab, you will show a basic example of how to:

- Organize and identify workflow files
- Add executable scripts
- Create workflow and action blocks
- Trigger workflows
- Discover workflow logs

⁹ <https://lab.github.com/githubtraining/github-actions:-hello-world>

Continuous integration with GitHub Actions

Continuous integration with actions

The following is an example of a basic continuous integration workflow created by using actions:

```
name: dotnet Build

on: [push]

jobs:
  build:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        node-version: [10.x]
    steps:
      - uses: actions/checkout@main
      - uses: actions/setup-dotnet@v1
        with:
          dotnet-version: '3.1.x'
      - run: dotnet build awesomeproject
```

- **on** specifies that this will occur when code is pushed
- **jobs** there is a single job called **build**
- **strategy** is being used to specify the Node.js version
- **steps** are performing a checkout of the code and setting up dotnet
- **run** is building the code

Environment variables

When using Actions to create CI or CD workflows, you will typically need to be able to pass variable values to the actions. This is done by using Environment Variables.

Built-in environment variables

GitHub provides a series of built-in environment variables. These all have a GITHUB_ prefix. Note: setting that prefix for your own variables will result in an error.

Examples of built-in environment variables are:

GITHUB_WORKFLOW is the name of the workflow.

GITHUB_ACTION is the unique identifier for the action.

GITHUB_REPOSITORY is the name of the repository (but also includes the name of the owner in owner/repo format)

Using variables in workflows

Variables are set in the YAML workflow files. They are passed to the actions that are in the step.

```
jobs:
  verify-connection:
    steps:
      - name Verify Connection to SQL Server
        run: node testconnection.js
    env:
      PROJECT_SERVER: PH202323V
      PROJECT_DATABASE: HAMaster
```

For more details on environment variables, including a list of built-in environment variables, see: **Environment variables**¹⁰

Passing artifacts between jobs

When using Actions to create CI or CD workflows, you will often need to be able to pass artifacts created by one job to another.

The most common ways to do this are by using the **upload-artifact** and **download-artifact** actions.

Upload-artifact

This action can upload one or more files from your workflow to be shared between jobs.

You can upload a specific file:

```
- uses: actions/upload-artifact
  with:
    name: harness-build-log
    path: bin/output/logs/harness.log
```

You can upload an entire folder:

```
- uses: actions/upload-artifact
  with:
    name: harness-build-logs
    path: bin/output/logs/
```

You can use wildcards:

```
- uses: actions/upload-artifact
  with:
    name: harness-build-logs
    path: bin/output/logs/harness[ab] ?/*
```

You can specify multiple paths:

```
- uses: actions/upload-artifact
  with:
    name: harness-build-logs
    path: |
      bin/output/logs/harness.log
```

¹⁰ <https://docs.github.com/en/free-pro-team@latest/actions/reference/environment-variables>

```
bin/output/logs/harnessbuild.txt
```

For further details on this action, see [upload-artifact¹¹](#)

Download-artifact

There is a corresponding action for downloading (or retrieving) artifacts.

```
- uses: actions/download-artifact
  with:
    name: harness-build-log
```

If no path is specified, it is downloaded to the current directory.

For further details on this action, see [download-artifact¹²](#)

Artifact retention

A default retention period can be set for the repository, organization, or enterprise.

You can set a custom retention period when uploading, but it cannot exceed the defaults for the repository, organization, or enterprise.

```
- uses: actions/upload-artifact
  with:
    name: harness-build-log
    path: bin/output/logs/harness.log
    retention-days: 12
```

Deleting artifacts

You can delete artifacts directly in the GitHub UI.

For details, see: [Removing workflow artifacts¹³](#)

Workflow badges

Badges can be used to show the status of a workflow within a repository.

They show if a workflow is currently passing or failing. While they can appear in several locations, they typically get added to the README.md file for the repository.

Badges are added by using URLs. The URLs are formed as follows:

<https://github.com/AAAAAA/RRRRR/workflows/WWWWW/badge.svg>

where:

- AAAAA is the account name
- RRRRR is the repository name

¹¹ <https://github.com/actions/upload-artifact>

¹² <https://github.com/actions/download-artifact>

¹³ <https://docs.github.com/en/free-pro-team@latest/actions/managing-workflow-runs/removing-workflow-artifacts>

- WWWW is the workflow name



They usually indicate the status of the default branch but can be branch specific. You do this by adding a URL query parameter:

?branch=BBBBB

where:

- BBBB is the branch name.

For more details, see: [Adding a workflow status badge¹⁴](#)

Best practices for creating actions

It's important to follow best practices when creating actions:

- Create chainable actions. Don't create large monolithic actions, instead, create smaller functional actions that can be chained together.
- Version your actions like other code. Others might take dependencies on various versions of your actions. Allow them to specify versions.
- Provide a **latest** label. If others are happy to use the latest version of your action, make sure you provide a **latest** label that they can specify to get it.
- Add appropriate documentation. As with other code, documentation helps others make use of your actions, and can help to avoid surprises about how they function.
- Add details **action.yml** metadata. At the root of your action, you will have an **action.yml** file. Make sure it has been populated with author, icon, any expected inputs and outputs.
- Consider contributing to the marketplace. It's easier for us all to work with actions when we all contribute to the marketplace. Help to avoid people needing to endlessly relearn the same issues.

Marking releases with Git tags

Releases are software iterations that can be packed for release.

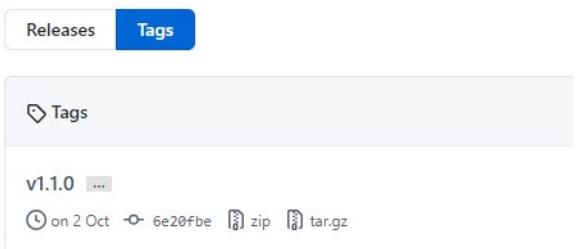
In Git, releases are based on Git tags. These tags mark a point in the history of the repository. Tags are commonly assigned as releases are created.

A screenshot of the GitHub Releases interface. At the top, there are two tabs: 'Releases' (which is selected) and 'Tags'. Below the tabs, there is a search bar with the text 'v1.1.0' and a dropdown menu labeled 'Target: master'. A message says 'Excellent! This tag will be created from the target when you publish this release.' Below that is a text input field containing 'First update after initial release'.

Often these tags will contain version numbers, but they can contain other values.

¹⁴ <https://docs.github.com/en/free-pro-team@latest/actions/managing-workflow-runs/adding-a-workflow-status-badge>

Tags can then be viewed in the history of a repository.



For more information on tags and releases, see: [About releases¹⁵](#)

¹⁵ <https://docs.github.com/en/free-pro-team@latest/github/administering-a-repository/about-releases>

Securing secrets for GitHub Actions

Creating encrypted secrets

Actions often need to be able to use secrets within pipelines. Common examples are passwords or keys.

In GitHub actions, these are called **Secrets**.

Secrets

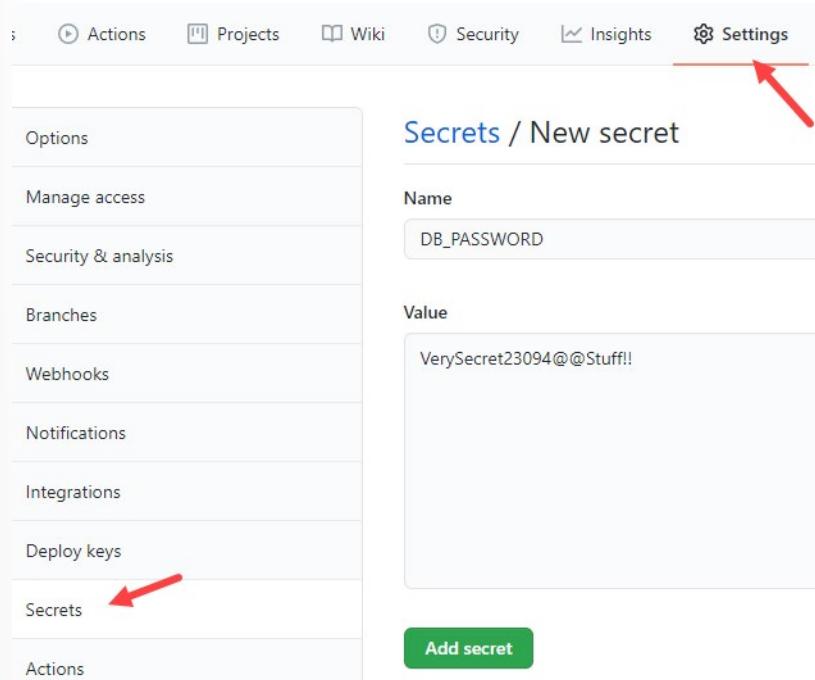
Secrets are similar to environment variables but encrypted. They can be created at two levels:

- Repository
- Organization

If secrets are created at the organization level, access policies can be used to limit the repositories that can use them.

Creating secrets for a repository

To create secrets for a repository, you must be the owner of the repository. From the repository **Settings**, choose **Secrets**, then **New Secret**.



For more information on creating secrets, see: [Encrypted secrets¹⁶](#)

Using secrets in a workflow

Secrets are not passed automatically to the runners when workflows are executed. Instead, when you include an action that requires access to a secret, you use the **secrets** context to provide it.

¹⁶ <https://docs.github.com/en/free-pro-team@latest/actions/reference/encrypted-secrets>

```
steps:  
  - name: Test Database Connectivity  
    with:  
      db_username: ${{ secrets.DBUserName }}  
      db_password: ${{ secrets.DBPassword }}
```

Command line secrets

Secrets should not be passed directly as command line arguments as they may be visible to others. Instead, treat them like environment variables:

```
steps:  
  - shell: pwsh  
    env:  
      DB_PASSWORD: ${{ secrets.DBPassword }}  
    run: |  
      db_test "$env:DB_PASSWORD"
```

Limitations

Workflows can use up to 100 secrets, and they are limited to 64KB in size.

For more information on creating secrets, see: [Encrypted secrets¹⁷](#)

¹⁷ <https://docs.github.com/en/free-pro-team@latest/actions/reference/encrypted-secrets>

Lab

Implementing GitHub Actions by using DevOps Starter

Lab overview

In this lab, you will learn how to implement a GitHub Action workflow that deploys an Azure web app by using DevOps Starter.

Objectives

After you complete this lab, you will be able to:

- Implement a GitHub Action workflow by using DevOps Starter
- Explain the basic characteristics of GitHub Action workflows

Lab duration

- Estimated time: **30 minutes**

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- [AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions¹⁸](https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/)

¹⁸ <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/>

Module review and takeaways

Module review questions

Review Question 1

True or False: Self-hosted runners should be used with public repos.

Review Question 2

Database passwords that are needed in a CI pipeline should be stored where?

Review Question 3

The metadata for an action is held in which file?

Review Question 4

How can the status of a workflow be shown in a repository ?

Answers

True or False: Self-hosted runners should be used with public repos.

False. They should not be used this way.

Database passwords that are needed in a CI pipeline should be stored where?

Encrypted Secrets

The metadata for an action is held in which file?

action.yml

How can the status of a workflow be shown in a repository ?

Using Badges

Module 9 Designing and Implementing 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 many aspects of 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 consider the changes in your dependencies as well. This requires a versioning mechanism for the dependencies so you can be selective of the 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 consumed in other projects and teams. You need to investigate 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 if 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 externalize 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.

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

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

Scanning your codebase for dependencies

There are several 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 must 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 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. Now there are a couple of de facto standard formats for packages.

- **NuGet**

NuGet packages (pronounced “new get”) 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](#)

- **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 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

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

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

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 commonly called a package feed. There are other names in use, such as repository or registry. We will refer to all of these as package 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 has 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 a user interface for the raw tooling. Examples of such tooling are Visual Studio 2017, Visual Studio Code and Eclipse.

Common public 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.

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.

Self-hosted and SaaS based package sources

The following private package sources will give you a starting point for finding the most relevant feeds.

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 a 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

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.

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>

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 can 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

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

Demonstration: pushing a package

For this demonstration, use the full version of **PartsUnlimited**⁶.

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. Build the project PartsUnlimited.Security.
3. Go to Command prompt, use command `dir bin\debug`

Here you see the .nupkg file for PartsUnlimited.Security

4. 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 command line.

- 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.

⁶ <http://microsoft.github.io/PartsUnlimited/>

-
- _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.

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 its 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 TFS⁷](#)

[Use npm to store JavaScript packages in Azure DevOps Services or TFS⁸](#)

[Get started with Maven packages in Azure DevOps Services and TFS⁹](#)

[Get started with Python packages in Azure Artifacts¹⁰](#)

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

⁸ <https://docs.microsoft.com/en-us/azure/devops/artifacts/get-started-npm?view=vsts&tabs=new-nav%2Cwindows>

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

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

Publish and then download a Universal Package¹¹

¹¹ <https://docs.microsoft.com/en-us/azure/devops/artifacts/quickstarts/universal-packages?view=vsts&tabs=azuredevops>

Package security

Securing access to package feeds

Trusted sources

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.

Securing access

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 can 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.

Securing availability

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 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: Can 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 can access the feeds it needs when running tasks. If you changed the build identity to be at project level, you would 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 (xpirit)	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 permissions¹²](#)

Authentication

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

¹² <https://docs.microsoft.com/en-us/azure/devops/artifacts/feeds/feed-permissions>

likely must 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.

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.

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 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 Artifacts¹³](#).

Versioning of artifacts

It is proper software development practice 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 completely 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

¹³ <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 can 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 feeds¹⁴**

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 2.0.0¹⁵**.

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.

¹⁴ <https://docs.microsoft.com/en-us/azure/devops/pipelines/artifacts/nuget#package-versioning>

¹⁵ <https://semver.org/>

Feeds in Azure Artifacts have three different views by default. These views are added when 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 feeds¹⁶**
- **Communicate package quality with prerelease and release views¹⁷**

Promoting packages

Azure Artifacts has the notion of promoting packages to views 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.

¹⁶ <https://docs.microsoft.com/en-us/azure/devops/artifacts/concepts/views>

¹⁷ <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 an 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 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 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 several 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 only reference one feed.
- On package creation, automatically publish packages back to the feed.

It is good to adopt a best practice 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 Artifacts**¹⁸.

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.

¹⁸ <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

Package management with Azure Artifacts

Lab overview

Azure Artifacts facilitate discovery, installation, and publishing NuGet, npm, and Maven packages in Azure DevOps. It's deeply integrated with other Azure DevOps features such as Build, making package management a seamless part of your existing workflows.

In this lab, you will learn how to work with Azure Artifacts by using the following steps:

- create and connect to a feed.
- create and publish a NuGet package.
- import a NuGet package.
- update a NuGet package.

Objectives

After you complete this lab, you will be able to:

- Create an Azure Active Directory (Azure AD) service principal.
- Create an Azure key vault.
- Track pull requests through the Azure DevOps pipeline.

Lab duration

- Estimated time: **40 minutes**

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- **AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions¹⁹**

¹⁹ <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/>

Module Review and Takeaways

Module review questions

Review Question 1

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

- public
- private

Review Question 2

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

- Yes
- No

Review Question 3

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

- NuGet
- NPM
- Maven
- Universal
- Python

Review Question 4

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

Review Question 5

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

Answers

Review Question 1

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

- public
- private

Review Question 2

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

- Yes
- No

Review Question 3

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 10 Designing 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 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 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.

Finally, we 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 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

- Choose a release management tool

Introduction to continuous delivery

Traditional IT development cycle

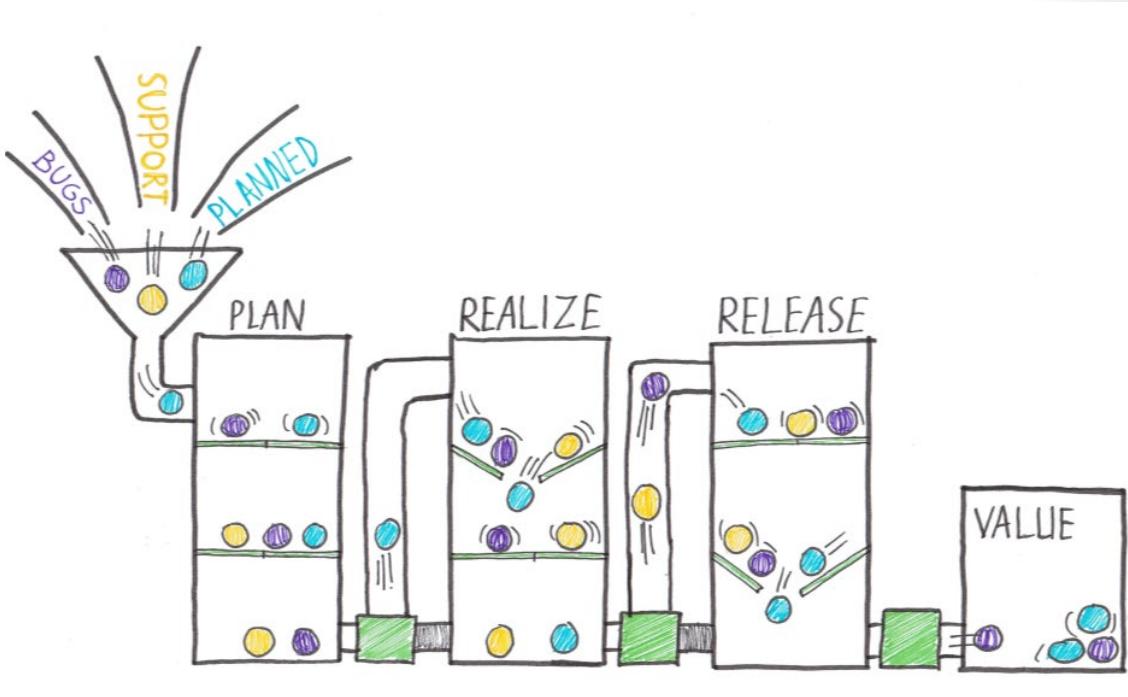
A few years ago, IT was a facilitating department. IT was there to support the business users, and because time had proven that developed software had bad quality by default, software changes were a risk. The resolution for this “quality problem” was to keep changes under strict control. The department 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.

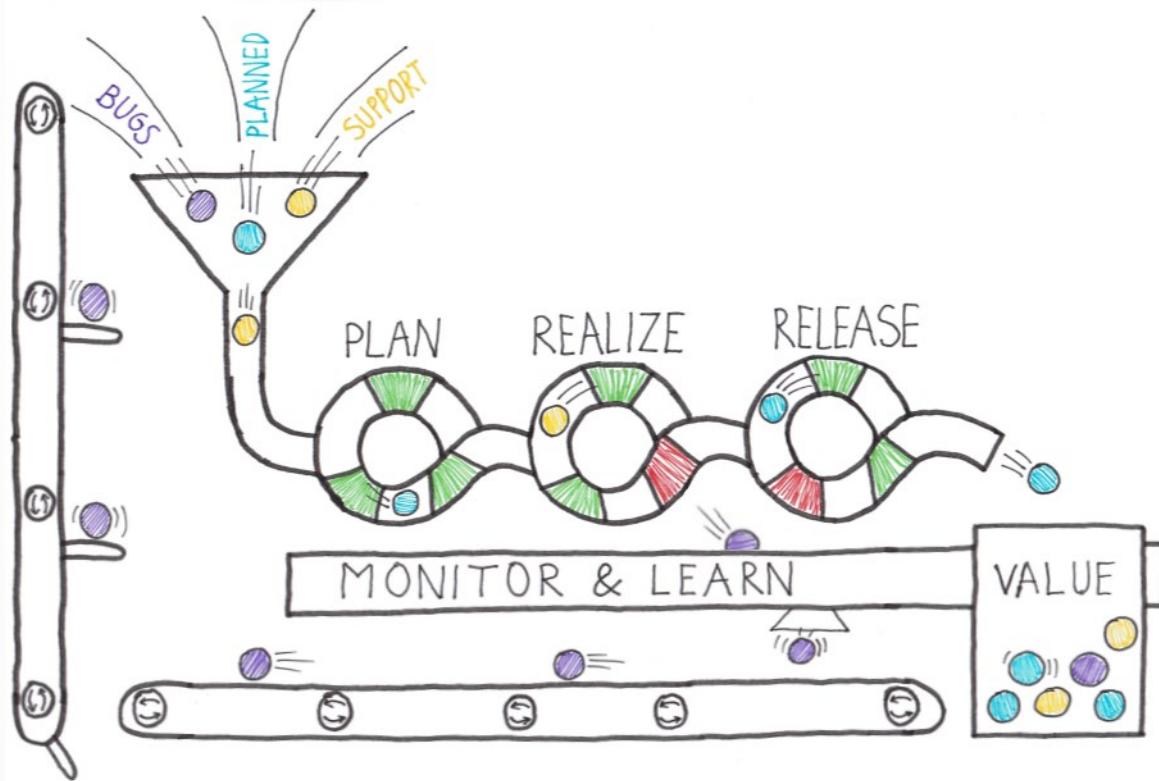
This is what I call silo-based development.



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 want 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 must 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 workflows 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?

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 the 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. 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.

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 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 and a strategy.

Releases and deployments

One of the essential steps in moving software more quickly to production is by changing the way we deliver software to production. In our industry, it is common to have teams that need to do overtime on 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. As soon as we deploy 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 Pipelines¹](#).

Separating technical releases from functional releases

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 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?

¹ <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/releases?view=vsts>

² <https://docs.microsoft.com/en-us/azure/devops/articles/phase-features-with-feature-flags?view=vsts>

If 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: the need for continuous delivery in your organization

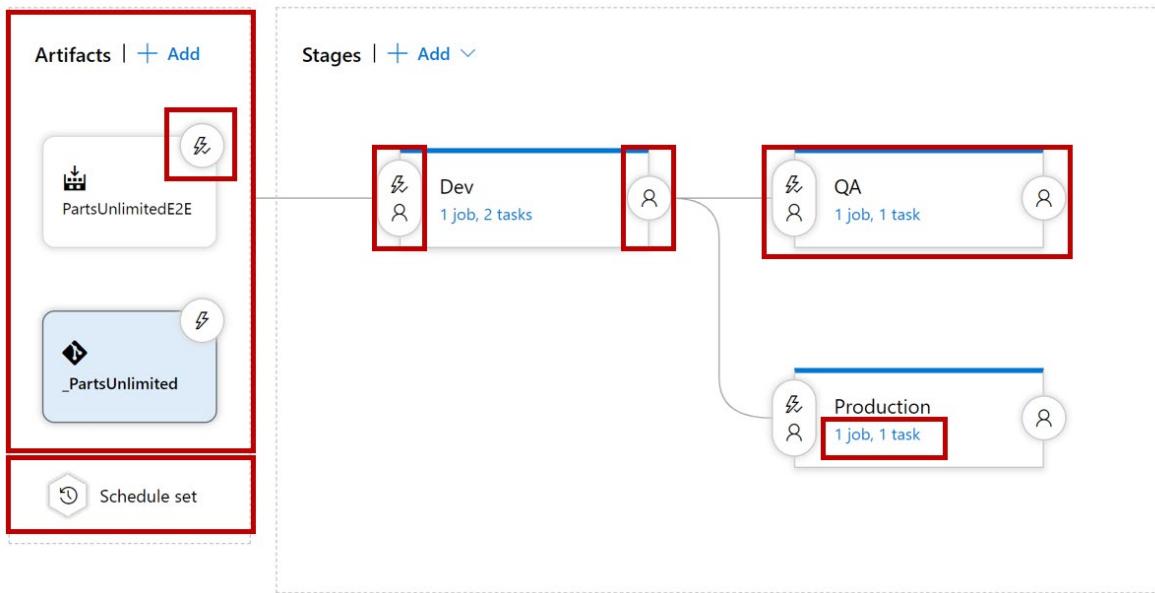
Topics you might want to discuss are:

- 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?

Release strategy recommendations

Release pipelines

A release pipeline takes artifacts and releases them through stages and usually, finally into production.



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.

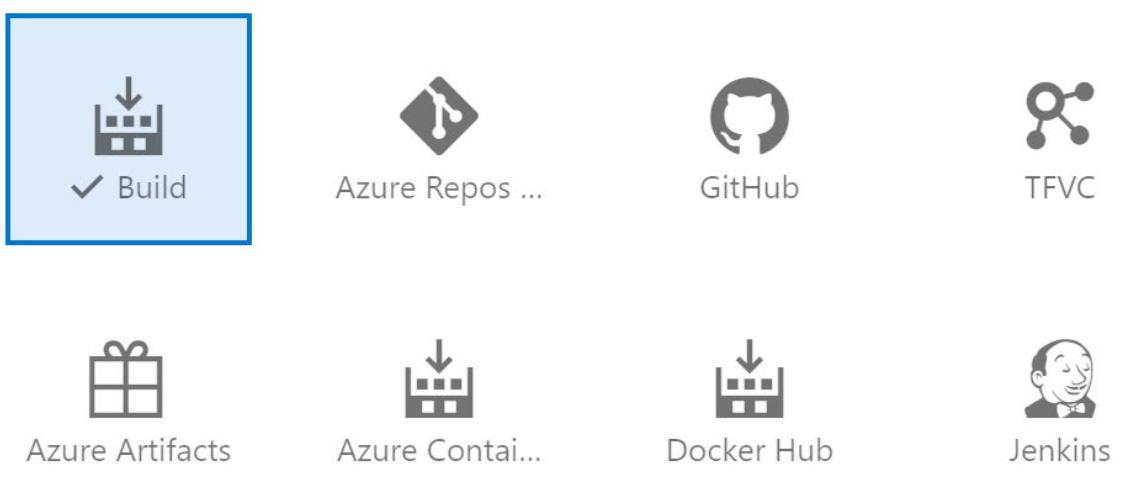
Artifact sources

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 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.

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>

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).

Finally, 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 the same advantages as using a build artifact stored in a safe location.

Considerations for choosing the appropriate 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](#).

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.

⁵ <https://semver.org/>

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 sources**⁶.

Demonstration: selecting an artifact source

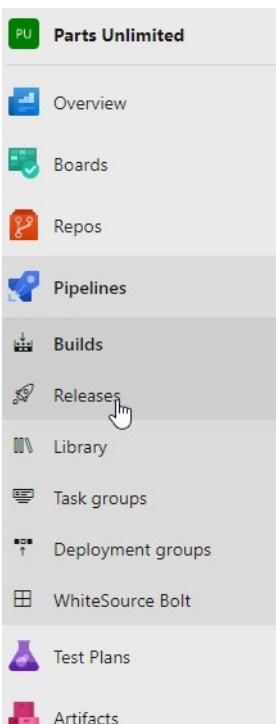
In this demonstration, you will investigate Artifact Sources.

Note: Before starting this demonstration, make sure you have performed the steps in the prerequisites section.

Steps

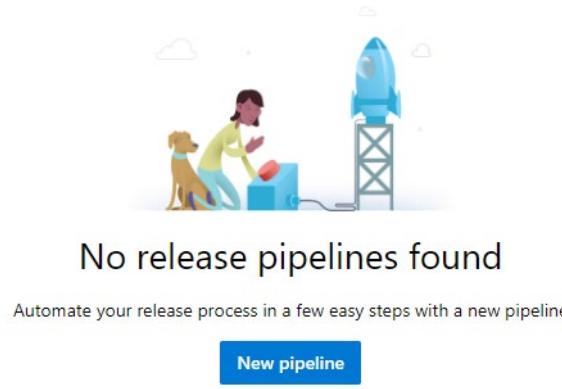
Let's look at how to work with one or more artifact sources in the release pipeline.

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 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 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. Finally, 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.

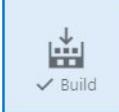
Configuring the build artifact

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

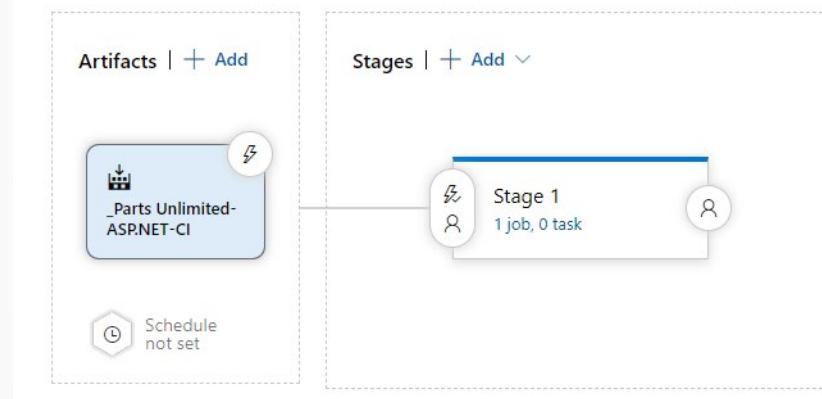
(i) The artifacts published by each version will be available for deployment in release pipelines. The latest successful build of Parts Unlimited-ASP.NET-CI published the following artifacts: drop.

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**.

Considerations for deployment to stages

When you have a clear view of the different stages that you will deploy to, you need to think about when you want to deploy to these stages and when.

As we mentioned in the introduction, Continuous Delivery is not only about deploying multiple times a day, but also 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?

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.

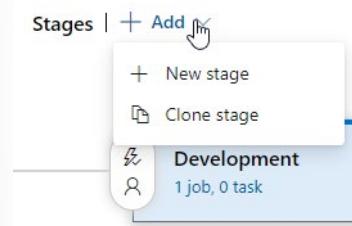
Steps

Let's now 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.

The screenshot shows a 'Stage' properties pane for a 'Development' stage. At the top, there are buttons for 'Delete', 'Move', and an ellipsis. Below that, the stage name is listed as 'Development' and the stage owner is 'Greg Low'. A note at the bottom states: '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.'

- 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

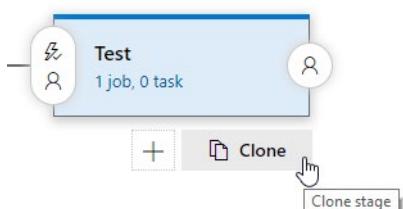
Deploy a Node.js application to an Azure Web App.

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.

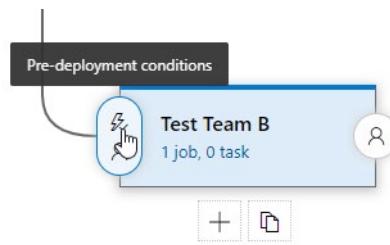
*Note: The same artifact sources move through each of the stages.

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

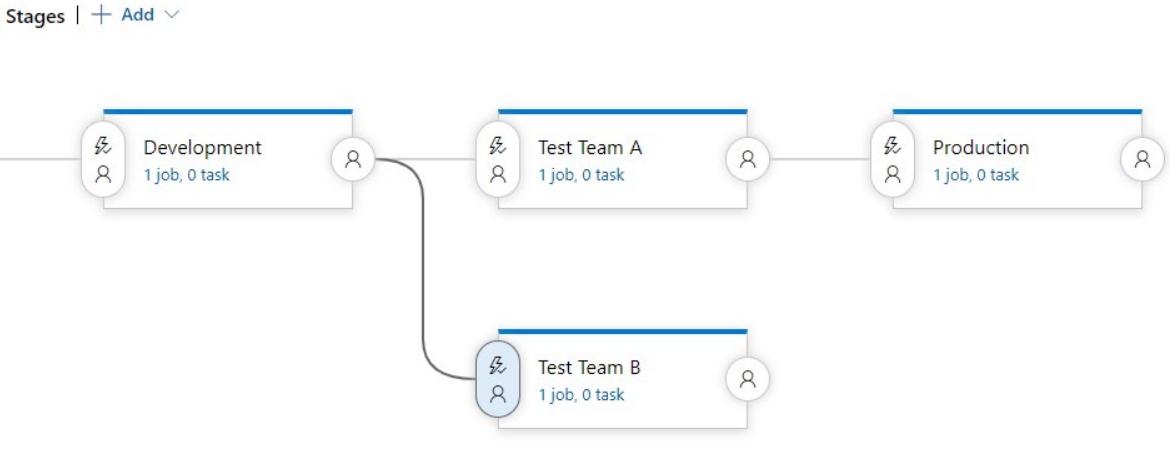
After release After stage (selected) Manual only

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.

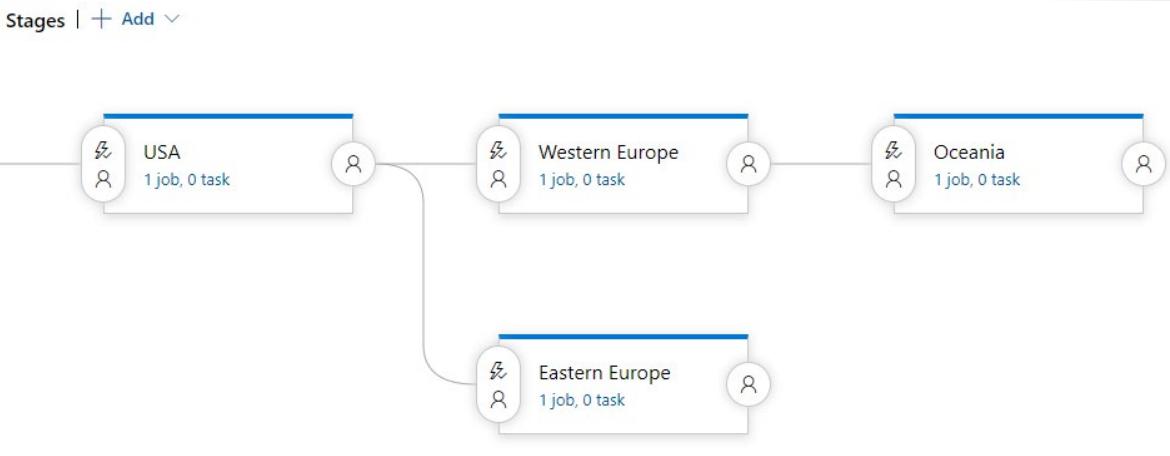
11. From the **Stages** drop down list, chose **Development** and uncheck **Test Team A**, then close the pane. We now have two concurrent Test stages.



Stage versus 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 cadence – three types of triggers

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.

For more information, see also:

- [Release triggers⁷](#)
- [Stage Triggers⁸](#)

Demonstration: Selecting your 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

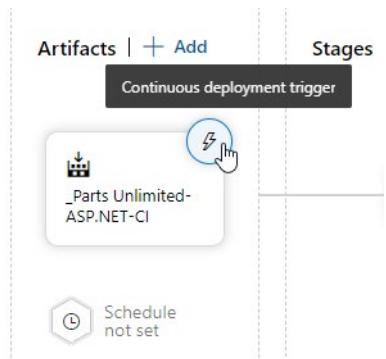
Let's now look at when our release pipeline is used to create deployments. Mostly, this will involve the use of triggers.

⁷ <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/triggers?view=vsts>

⁸ <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/triggers?view=vsts#env-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.



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 ①
No filters added.
+ Add | ▾

Pull request trigger
Build: `_Parts Unlimited-ASP.NET-CI`

Disabled
 ⓘ Enabling this will create a release every time a selected artifact is available as part of a pull request workflow

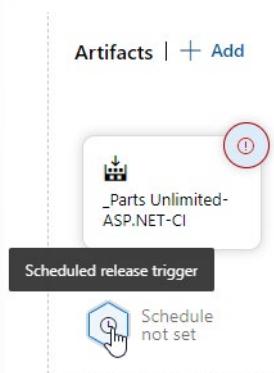
Once this is selected, every time that a build completes, a deployment of the release pipeline will start.

- ✓ 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 (i)

After release

After stage

Manual only

Artifact filters (i) Disabled

Schedule (i) Disabled

Pull request deployment (i) 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 (i)
Define behavior when multiple releases are queued for deployment Disabled

✓ 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.

Release approvals

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 Continous 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.

- **Release approvals and gates overview⁹**

- **Release Approvals¹⁰**

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 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.

⁹ <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/approvals/approvals?view=vsts>

¹⁰ <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/approvals?view=vsts>

Stages | + Add ▾



2. Click the **Disabled** button in the Pre-deployment approvals section to enable it.

Pre-deployment approvals ▾

Select the users who can approve or reject deployments to this stage

Enabled

Approvers [i](#)

Search users and groups for approvers

[① 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 [①](#)

3. In the **Approvers** list, find your own name and select it. Then set the **Timeout** to **1 Days**.

Pre-deployment approvals ▾

Select the users who can approve or reject deployments to this stage

Enabled

Approvers [i](#)

GL 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 [①](#)

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

✓ Release [Release-1](#) has been created

Pipeline Tasks ▾ Variables Retention Options History

10. At this point, an email should have been received, indicating that an approval is required.

Microsoft Azure DevOps

Release to all environments > Release-1

Deployment to Development is waiting for your approval

Parts Unlimited-ASP.NET-CI / 20190901.2 master

Requested for Greg Low

[View approval](#)

Summary

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.

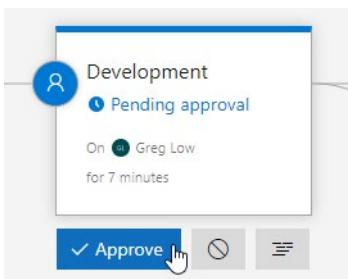
↑ Release to all environments > Release-1 ▾

Pipeline Variables History Deploy Cancel Approve multiple

Release	Stages
<p>Manually triggered by Greg Low 01/09/2019, 16:40</p> <p>Artifacts Parts Unlimited-ASP.N... 20190901.2 master</p>	<p>Development Pending approval</p> <p>On Greg Low for 4 minutes</p>

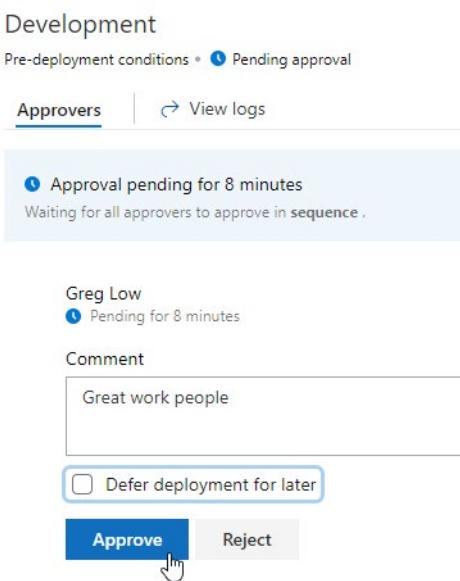
You can see that a release has been manually triggered and that the Development stage is waiting for an approval. As an approver, you can now perform that approval.

12. Hover over the **Development** stage and click the **Approve** icon that appears.

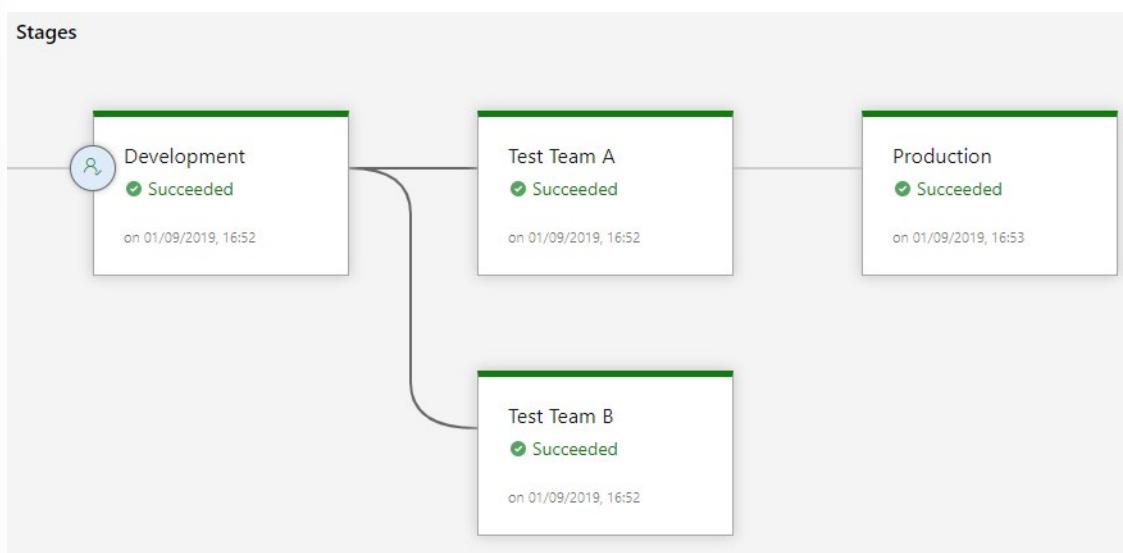


Note: Options to cancel the deployment or to view the logs are also provided at this point

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.



Release gates

Release gates give you additional control over the start and completion of the deployment pipeline. They are often set up as a pre-deployment and post-deployment conditions.

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 overview¹¹](#)
- [Release Approvals¹²](#)
- [Release Gates¹³](#)

Demonstration: Setting up a release gate

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 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.

First we need to make sure that the Release Gates can execute work item queries.

1. On the **Boards > Queries** page, click **All** to see all the queries (not just favorites).
2. Click the ellipsis for **Shared Queries** and choose **Security**.
3. Add a user **ProjectName Build Service (CompanyName)** if they are not already present, and choose **Allow** for Read permissions.

Now let's look at configuring a release gate.

1. Click the lightning icon on the **Development** stage to open the pre-deployment conditions settings.

Stages | + Add ▾

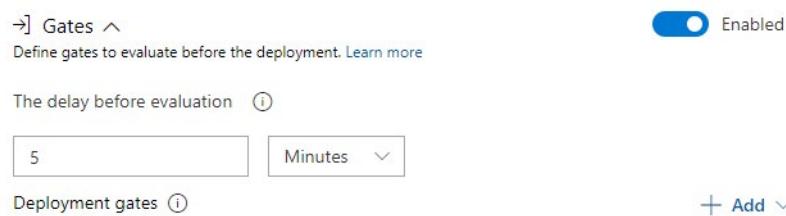


2. In the Pre-deployment conditions pane, click the **Disabled** button beside **Gates** to enable them.

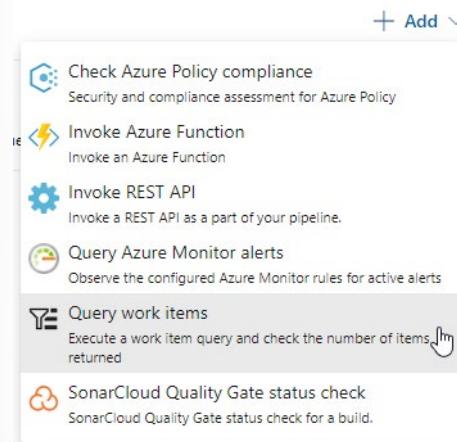
¹¹ <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/approvals/approvals?view=vsts>

¹² <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/approvals/approvals?view=vsts>

¹³ <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/approvals/gates?view=vsts>



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**.

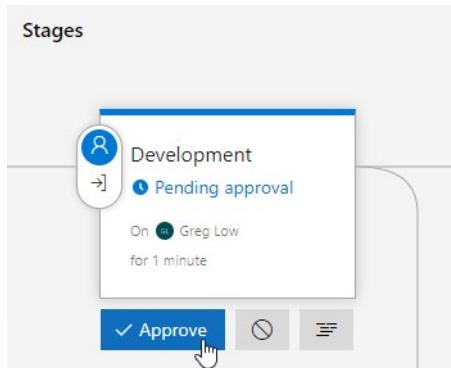


- Click on the release number to see how the release is proceeding.

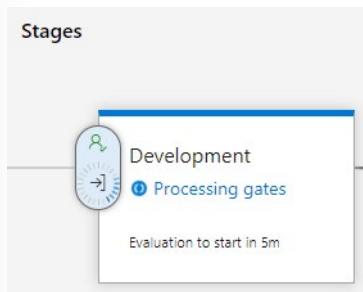
All pipelines >  Release to all environments



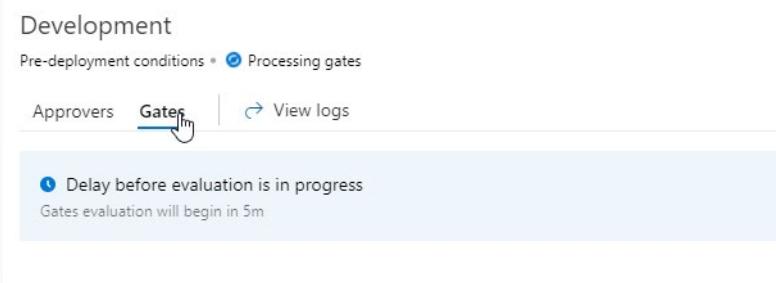
9. If it is waiting for approval, click **Approve** to allow it to continue, and in the **Development** pane, click **Approve**.



After a short while, you should see the release continuing and then entering the phase where it will process the gates.



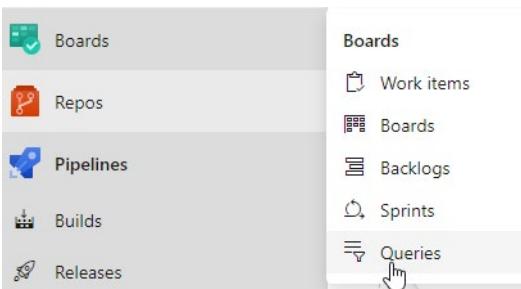
10. In the **Development** pane, click **Gates** to see the status of the release gates.



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.

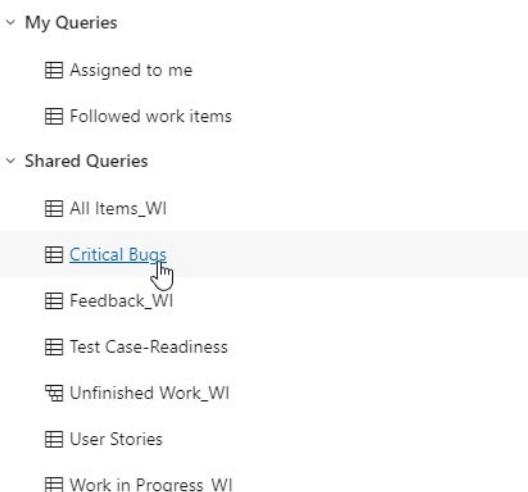
12. In the main menu, click **Boards** then click **Queries**.



13. In the **Queries** window, click **All** to see all the available queries.



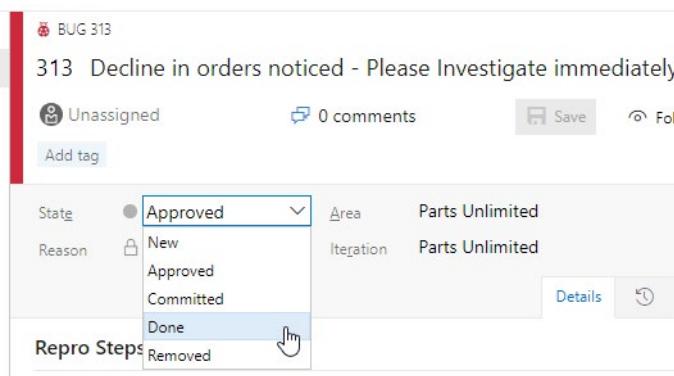
14. From the list of queries, click **Critical bugs**.



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.

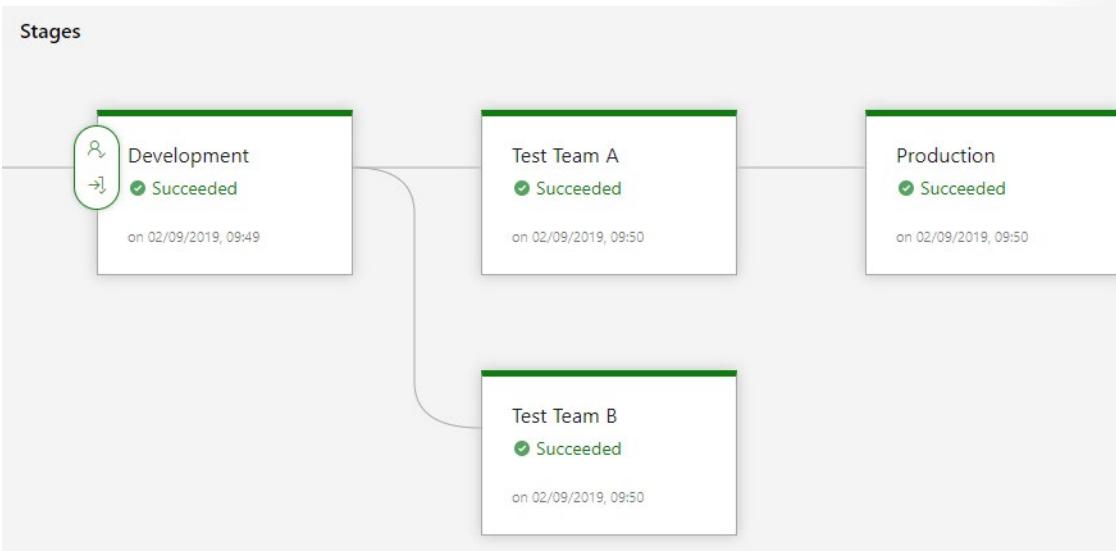
A screenshot of the 'Shared Queries' page in Microsoft Azure DevOps. The query name is 'Critical Bugs'. The results table has columns for 'ID', 'Work Item...', and 'Title'. One result is shown: ID 313, Work Item type 'Bug', Title 'Decline in orders noticed - Please Investigate immediately'. Below the table, a tooltip is shown over the 'Run query' button, which is highlighted with a blue border. The tooltip text is 'Done' with a circular arrow icon.

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 Microsoft Azure DevOps. The page title is 'Release to all environments'. There are tabs for 'Releases', 'Deployments', and 'Analytics', with 'Releases' being the active tab. Below the tabs, there is a section titled 'Releases' with two items: 'Release-2' and 'Release-1'. 'Release-2' is the latest release, indicated by a green circle with 'GL' and a checkmark icon. It was created on '201905.2' and is associated with the 'master' branch. 'Release-1' was created on '20190901.2' and is also associated with the 'master' branch. Both releases have three dots (...) to their right.

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 process versus release

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 must 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%	✓ 100%	▶
Sps.SelfTest	✓ 100%	✓ 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 to protect quality

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 several 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 a feature titled "344 Shopping Cart should be personalized". The card includes sections for Description, Acceptance Criteria, Release Notes, Discussion, Status, Details, Development, and Related Work. The "Release Notes" section contains the placeholder text "Here can be the commercial release notes." The "Development" section notes that development hasn't started yet.

- **Generate Release Notes Build Task¹⁴**
- **WIKI Updater Tasks¹⁵**
- **Atlassian Confluence¹⁶**
- **Azure DevOps Wiki¹⁷**

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.

¹⁴ <https://marketplace.visualstudio.com/items?itemName=richardfennellBM.BM-VSTS-XplatGenerateReleaseNotes>

¹⁵ <https://marketplace.visualstudio.com/items?itemName=richardfennellBM.BM-VSTS-WIKIUpdater-Tasks>

¹⁶ <https://www.atlassian.com/software/confluence>

¹⁷ <https://azure.microsoft.com/en-us/services/devops/wiki/>

Choosing the right release management tool

Considerations for choosing release management tools

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.

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

Finally, the work needs to be done within the pipeline. It is not only about the workflow and orchestration, but the code also 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 Jenkins plugin marketplace interface. On the left, there's a sidebar with navigation links like 'Blog', 'Documentation', 'Plugins', 'Community', 'Sub-projects', and 'About'. Below that is a 'Categories' section with various filters such as 'Available', 'Installed', 'Pending', 'To do', and 'Archived'. The main area displays a grid of 50 Jenkins plugins, each with a thumbnail, name, version, and a brief description. The plugins are arranged in 10 rows and 5 columns. Some notable ones include 'Mailer', 'Credentials', 'SSH Credentials', 'Matrix Project', 'JUnit', 'Script Security', 'SCM API', 'Matrix Authorization Strategy', 'OWASP Markup Formatter', 'Struts', 'Pipeline Step API', 'Token Macro', 'Git client', 'LDAP', 'PAM-Authentication', 'Pipeline: SCM Step', 'Git', 'Display URL API', 'Ant', 'Pipeline: API', 'SSH Slaves', 'bouncycastle API', 'Pipeline: Supporting APIs', 'Plain Credentials', 'Pipeline: Job', 'Folders', and 'Durable Task'. Each plugin entry includes the author's name and a 'GitHub' icon.

Plugin Name	Description	Version	Author	GitHub
Mailer	This plugin allows you to configure small notifications for build results.	2.197.40 Jenkins 2.625.3+>	remshah	Ma
Credentials	This plugin allows you to store credentials in Jenkins.	2.195.61 Jenkins 1.625.3+>	Stephen Connolly	Gf
SSH Credentials	Allows storage of SSH credentials in Jenkins.	2.195.77 Jenkins 1.609++	SG	
Matrix Project	Multi-configuration (matrix) project type.	2.169.57 Jenkins 2.169++	Oliver Gondor	MP
JUnit	Allows JUnit-format test results to be published.	2.165.14 Jenkins 2.73+>	danielbeck	JU
Script Security	Allows Jenkins administrators to control what in-process scripts can be run by users with specific privileges.	2.157.98 Jenkins 2.73+>	Stephen Connolly	SA
SCM API	This plugin provides a new enhanced API for interacting with SCM systems.	2.156.28 Jenkins 2.73+>	SG	
Matrix Authorization Strategy	Offers matrix-based security authorization strategies (global and per-project).	2.150.02 Jenkins 2.60.1+>	danielbeck	MA
OWASP Markup Formatter	Uses the OWASP Java HTML Sanitizer to allow safe replacement of HTML markup to be entered in project descriptions.	1.565.37 Jenkins 1.565.3+>	SG	
Struts	Library plugin for DSGI plugins that need access to Jenkins objects.	2.073.44 Jenkins 2.625.3+>	glick	St
Pipeline Step API	API for asynchronous build step primitives.	2.070.08 Jenkins 2.642.3+>	danielbeck	PS
Token Macro	This plugin adds a more expressive capability for other plug-ins to use.	2.062.79 Jenkins 1.625.3+>	Kohsuke Kawaguchi	TM
Git client	Utility plugin for Git support in Jenkins.	2.021.69 Jenkins 1.462.3+>	Mark Waite	Gc
LDAP	Utility plugin for LDAP authentication to Jenkins.	2.020.92 Jenkins 1.462.3+>	andrewc	LD
PAM-Authentication	Adds Unix PAM-style Authentication Modules (PAM) support to Jenkins.	2.020.02 Jenkins 1.364.2+>	matt	PA
Pipeline: SCM Step	This plugin integrates Git with Jenkins.	2.012.47 Jenkins 2.60+>	SG	
Git	Provides the DisplayURLProvider extension point to provide alternative URLs for use in notifications.	2.007.77 Jenkins 1.40.0+>	James Dumay	DG
Display URL API		2.000.09 Jenkins 1.40.0+>		
Ant	Plugin that defines Pipeline API.	1.975.64 Jenkins 2.32.3+>	America-Premier	An
Pipeline: API	Plugin that defines Pipeline API.	1.998.03 Jenkins 2.32.3+>	Stephen Connolly	PA
SSH Slaves	Allows to launch agents over SSH, using a Java implementation of the SSH protocol.	1.994.02 Kohsuke Kawaguchi (3 other contributors)	SG	
bouncycastle API	This plugin provides an stable API to Bouncy Castle related tasks.	1.964.02 Kohsuke Kawaguchi (3 other contributors)	Albert Laikeen	BA
Pipeline: Supporting APIs	Common utility implementations to build Pipeline Plugin.	1.958.93 Jenkins 2.21.1+>	glick	PS
Plain Credentials	Allows use of plain-strings and files as credentials.	1.950.93 Jenkins 2.40.3+>	PC	
Pipeline: Job	Defines a new job-type for pipelines and provides their generic user interface.	1.952.74 Jenkins 2.31.3+>	SG	
Folders	This plugin allows users to create "folders" to organize jobs. Users can define shared workspace that is not visible to Jenkins yet be monitored.	1.982.07 Jenkins 2.30.4+>	Stephen Connolly	Fo
Durable Task		1.984.17 Jenkins 2.7.3+>	SG	
Subversion	jQuery UI Libs: jQuery bundles (jQuery, Installs: 156.89P Jenkins 2.7.3+> SCMs, etc.)	1.952.07 Kohsuke Kawaguchi (3 other contributors)	SU	
JavaScript GUI Libs: jQuery bundle (jQuery, Installs: 152.20P Jenkins 2.7.3+> SCMs, etc.)	JavaScript GUI Libs: jQuery bundle (jQuery and jQuery UI) plugin.	1.944.77 Kohsuke Kawaguchi (3 other contributors)	SG	
JavaScript GUI Libs: ACE Editor bundle	JavaScript GUI Libs: ACE Editor bundle (jsGedit).	1.944.77 Kohsuke Kawaguchi (3 other contributors)	JG	
Pipeline: Nodes and Processes	Pipeline stage, locking agents and workspaces, and running external processes that may survive a Jenkins restart.	1.933.72 Jenkins 2.21.1+>	danielbeck	PP
Pipeline: Groovy	Pipeline execution engine based on continuation-passing style transformation of Groovy scripts.	1.932.72 Jenkins 2.21.1+>	SG	
Git server	Allows Jenkins to act as a Git server.	1.932.14 Jenkins 2.20.1+>	kohsuke	Gs
Branch API	This plugin provides an API for managing branch based projects.	1.929.00 Jenkins 2.20.1+>	Stephen Connolly	BA
Pipeline: Basic Steps	Commonly used steps for Pipelines.	1.909.89 Jenkins 2.15.3+>	SG	
Credentials Binding	Allows credential to be bound to environment variables for use from miscellaneous build steps.	1.957.74 Jenkins 2.15.3+> Buildsteps	glick	CB

The screenshot shows the Visual Studio Marketplace interface. At the top, there are tabs for Visual Studio, Visual Studio Code, Azure DevOps (which is selected), Subscriptions, and links for Build your own and Publish extensions. A search bar at the top is labeled "Search Azure DevOps extensions". Below the search bar, it says "864 Results". There are filters for Showing: All categories, Hosted On: Any, Price: Any, and Sort By: Downloads. The main area displays a grid of 24 extension cards, each with a title, icon, developer, download count, description, rating, and status (e.g., FREE, FREE TRIAL).

Extension	Developer	Downloads	Description	Rating	Status
Code Search	Microsoft	91.1K	Code Search provides fast, flexible and accurate search across all your code.	★★★★★	FREE
Test & Feedback	Microsoft	73.4K	Now everyone on the team can own quality. Capture findings, create issues, and...	★★★★★	FREE
VSTS Open in Excel	Microsoft DevLabs	47.5K	This extension opens work items and query results in Excel from Visual Studio Team...	★★★★★	FREE
Azure Artifacts	Microsoft	31.4K	Create, host, and share packages with your team	★★★★★	FREE TRIAL
Folder Management	Microsoft DevLabs	29.4K	Create a folder in your source repositories from the web. No need to clone the repository...	★★★★★	FREE
Slack Integration	Microsoft	26K	Keep up to date on your Azure DevOps projects from Slack	★★★★★	FREE
Analytics	Microsoft	23.7K	Gain insights into the health and status of your Azure DevOps and Azure DevOps...	★★★★★	FREE TRIAL
Work Item Visualization	Microsoft DevLabs	23.5K	Visualize relationships between work items from within the work item form.	★★★★★	FREE
Microsoft Teams Integration	Microsoft	21.1K	Microsoft Teams makes collaborating on software projects a breeze - from ide...	★★★★★	FREE
SonarQube	SonarSource	20.2K	Detect bugs, vulnerabilities and code smells across project branches and pull...	★★★★★	FREE
Delivery Plans	Microsoft	19.6K	Manage your portfolio of work with a calendar based view across teams and...	★★★★★	FREE
IIS Web App Deploy	Microsoft	18.7K	Using WinRM connect to the host Computer, to deploy a Web project using Web...	★★★★★	FREE
Test Manager	Microsoft	17.4K	Integrated test management system for all your manual, exploratory and user...	★★★★★	FREE TRIAL
Team Calendar	Microsoft DevLabs	15.9K	Track events important to your team, view and manage days off, quickly see when...	★★★★★	FREE
HockeyApp	Microsoft	13.7K	Distribute your builds, collect crash reports, and get feedback from your users.	★★★★★	FREE
CatLight	Catlight.io	12.6K	Build & task status notifications in your tray	★★★★★	FREE
Replace Tokens	Guillaume Rouchon	12.4K	Task to replace tokens in files.	★★★★★	FREE
Docker Integration	Microsoft	11.5K	Build, push, run or deploy Docker images and multi-container Docker applications.	★★★★★	FREE
Octopus Deploy Integration	Octopus Deploy	10.2K	Build and Release tasks and other features for integrating with Octopus Deploy....	★★★★★	FREE
Test Case Explorer	Microsoft DevLabs	10K	Manage your test cases better. Find, filter, analyze usage of test cases, and more.	★★★★★	FREE
Release Management	Microsoft DevLabs	9.4K	Utility tasks for Release Management	★★★★★	FREE
Branch Visualization	Microsoft DevLabs	9.1K	Visualize your TFVC branch hierarchies	★★★★★	FREE
AWS Tools for Microsoft	Amazon Web Services	8.7K	Tasks for Amazon S3, AWS Elastic Beanstalk, AWS CodeDeploy, AWS Lambda...	★★★★★	FREE
Estimate	Microsoft DevLabs	7.8K	Planning Poker in Visual Studio Team Services.	★★★★★	FREE

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?

Common 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

- [Jenkins¹⁸](#)
- [Tutorial: Jenkins CI/CD to deploy an ASP.NET Core application to Azure Web App service¹⁹](#)
- [Azure Friday - Jenkins CI/CD with Service Fabric²⁰](#)

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/²¹](#)
- [How to get started on CircleCI 2.0: CircleCI 2.0 Demo²²](#)

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

¹⁸ <https://jenkins.io/>

¹⁹ <https://cloudblogs.microsoft.com/opensource/2018/09/21/configure-jenkins-cicd-pipeline-deploy-asp-net-core-application/>

²⁰ <https://www.youtube.com/watch?v=5RYmoolZqS4>

²¹ <https://circleci.com/>

²² <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 Pipelines²³](#)
- [Building and Deploying your Code with Azure Pipelines²⁴](#)

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.

[GitLab²⁵](#)

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 Bamboo²⁶](#)

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 - XebiaLabs²⁷](#)

²³ <https://azure.microsoft.com/en-us/services/devops/pipelines/>

²⁴ <https://www.youtube.com/watch?v=NuYDAs3kNV8>

²⁵ <https://about.gitlab.com/stages-devops-lifecycle/release/>

²⁶ <https://www.atlassian.com/software/bamboo/features>

²⁷ <https://xebialabs.com/products/xl-release/>

Labs

Controlling deployments using Release Gates

Lab overview

This lab covers the configuration of the deployment gates and details how to use them to control execution of Azure pipelines. To illustrate their implementation, 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 Application Insights of Azure Monitor.

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 jobs 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 of the initial set of users.

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.

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.

As an example:

- Pre-deployment gates ensure there are no active issues in the work item or problem management system before deploying a build to an environment.
- Post-deployment gates ensure there are no incidents from the monitoring or incident management system for the app after it's been deployed, before promoting the release to the next environment.

There are 4 types of gates included by default in every account.

- Invoke Azure function: Triggers execution of an Azure function and ensures a successful completion.
- Query Azure monitor alerts: Observes the configured Azure monitor alert rules for active alerts.
- Invoke REST API: Makes a call to a REST API and continues if it returns a successful response.
- Query Workitems: Ensures the number of matching work items returned from a query is within a threshold.

Objectives

After you complete this lab, you will be able to:

- Configure release pipelines
- Configure release gates
- Test release gates

Lab duration

- Estimated time: **75 minutes**

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- **AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions²⁸**

Creating a release dashboard

Lab overview

In this lab, you will step through creation of a release dashboard and the use of REST API to retrieve Azure DevOps release data, which you can make this way available to your custom applications or dashboards.

The lab leverages the Azure DevOps Starter resource, which automatically creates an Azure DevOps project that builds and deploys an application into Azure.

Objectives

After you complete this lab, you will be able to:

- create a release dashboard
- use REST API to query release information

Lab duration

- Estimated time: **45 minutes**

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- **AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions²⁹**

²⁸ <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/>

²⁹ <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/>

Module review and takeaways

Module review questions

Review Question 1

When you want to change an immutable object of any type, what do you do?

Review Question 2

What can you use to prevent a deployment in Azure DevOps when a security testing tool finds a compliance problem?

Review Question 3

Even if you create exactly what a user requested at the start of the project, the solution will often be unsuitable for the same user. Why?

Answers

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

Even if you create exactly what a user requested at the start of the project, the solution will often be unsuitable for the same user. Why?

Their needs will have changed during the project

Module 11 Implementing Continuous Deployment using Azure Pipelines

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 several 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 several 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
- 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
- List the different ways to inspect the health of your pipeline and release by using alerts, service hooks, and reports

Create a release pipeline

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 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.

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 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](#)

Release jobs

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.

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/addons/app/bamboo/trending>

⁴ <https://plugins.jenkins.io/>

⁵ <https://marketplace.visualstudio.com/>

The screenshot shows the 'Release Jobs Picture' section of the Azure Pipelines interface. At the top, there are tabs for Pipeline, Tasks (which is selected), Variables, Retention, Options, and History. Below the tabs, a 'Development Deployment process' is listed. The main area displays several job configurations:

- macOS Job**: Run on agent. This is the currently selected job, indicated by a blue background.
- 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
- Release n/a to internal**: Google Play - Release

Each job entry includes a '+' icon to add more instances and a three-dot menu icon for additional options.

For more information, see [Jobs in Azure Pipelines and TFS⁶](#).

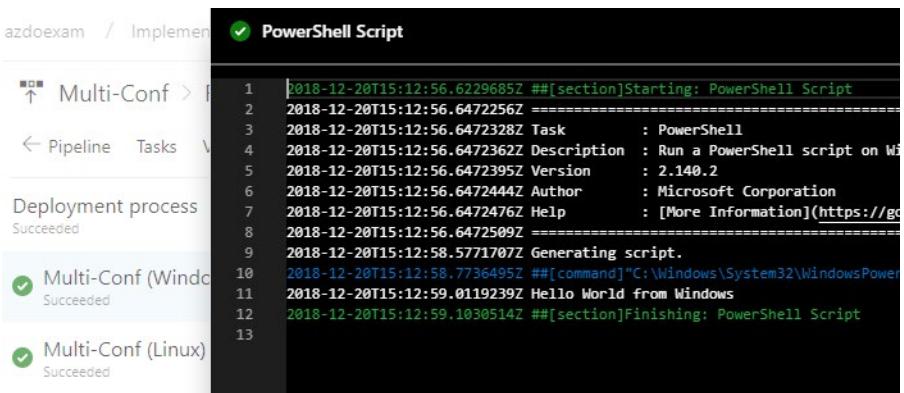
Multi-configuration and multi-agent

You might need 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.

⁶ <https://docs.microsoft.com/en-us/azure/devops/pipelines/process/phases?view=vsts&tabs=yaml>



The screenshot shows the Azure DevOps Pipeline interface. On the left, there's a sidebar with 'azdoexam / Implement' and a 'Deployment process' section showing two successful runs: 'Multi-Conf (Windows)' and 'Multi-Conf (Linux)'. 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  2018-12-20T15:12:56.6472328Z Task      : PowerShell
4  2018-12-20T15:12:56.6472362Z Description : Run a PowerShell script on Windows
5  2018-12-20T15:12:56.6472395Z 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=865931)
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 -NonInteractive -Command ".\HelloWorld.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 pipeline](#)⁷.

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?

⁷ <https://docs.microsoft.com/en-us/azure/devops/pipelines/process/phases?view=vsts&tabs=designer#multi-configuration>

Provision and configure environments

Provision and configure 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, Azure SQL Database in both PaaS and serverless options
- Clusters
- Service Connections

Let us dive a bit further into these different target environments and connections.

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 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 into 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 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

Finally, 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.

Service connections

In addition to the environments, when a pipeline needs access to resources, you will often need to provision service connections.

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 Cloudformation⁸**

⁸ <https://aws.amazon.com/cloudformation/>

- **Terraform**⁹
- **Powershell DSC**¹⁰
- **AWS Lambda**¹¹
- **Azure Functions**¹²
- **Chef**¹³
- **Puppet**¹⁴
- **Azure Resource Manager /ARM**¹⁵

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.

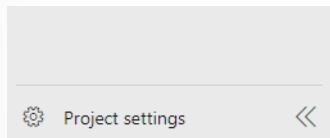
To follow along with this walkthrough, you will need to have an existing Azure subscription, that contains an existing storage account.

Steps

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 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**.

⁹ <https://www.terraform.io/>

¹⁰ <https://docs.microsoft.com/en-us/powershell/scripting/dsc/overview/overview?view=powershell-7>

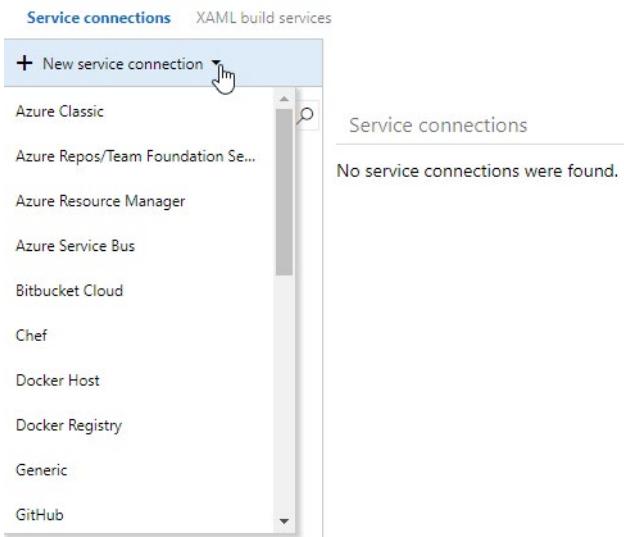
¹¹ <https://docs.aws.amazon.com/lambda/latest/dg/welcome.html>

¹² <https://azure.microsoft.com/en-us/services/functions>

¹³ <https://www.chef.io/chef/>

¹⁴ <https://puppet.com/>

¹⁵ <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.

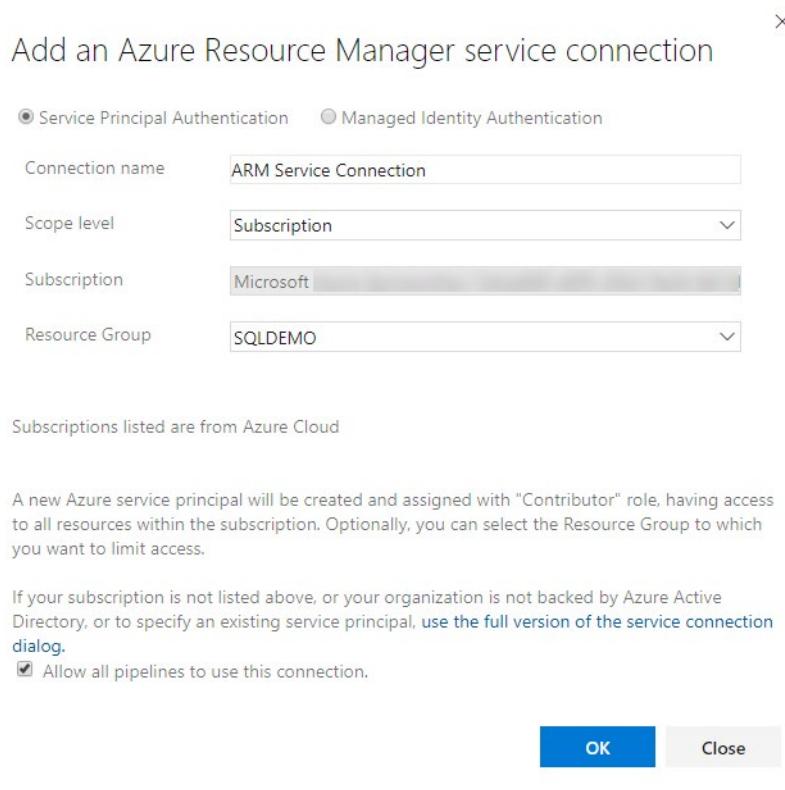
If your subscription is not listed above, or your organization is not backed by Azure Active Directory, or to specify an existing service principal, [use the full version of the service connection dialog.](#)

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.



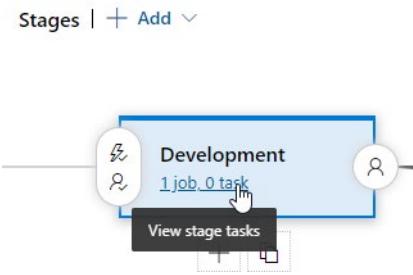
Notice that what we are creating is a **Service Principal**. We will be using the Service Principal as a means of authenticating to Azure. At the top of the window, there is also an option to set up Managed Identity Authentication instead.

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.

The screenshot shows the 'Service connections' list. On the left, there is a sidebar with a '+ New service connection' button and a 'Filter connections...' search bar. The main area displays a single service connection entry: 'Service connection: ARM Service Connection'. Below this, there are tabs for 'Details', 'Roles', 'Request history', and 'Policies'. Under the 'INFORMATION' section, it says 'Type: Azure Resource Manager', 'Created by Greg Low', and 'Connected to service using Service Principal'. Under the 'ACTIONS' section, there is a list of actions: 'Update service connection', 'Manage service connection roles', 'Manage Service Principal', and '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 yet. Each stage can execute many tasks.

The screenshot shows the 'Tasks' tab for the 'Development' stage. The stage is described as a 'Deployment process'. Below the stage, there is a list containing one item: 'Agent job' with the sub-note 'Run on agent'. To the right of this list is a blue plus sign button.

7. Click the + sign to the right of **Agent job** to add a new task. Note the available list of task types.

The screenshot shows the 'Add tasks' dialog. At the top, there are buttons for 'Add tasks' and 'Refresh', and a search bar with the placeholder 'Search'. Below these are tabs for 'All', 'Build', 'Utility', 'Test', 'Package', 'Deploy', 'Tool', and 'Marketplace'. The 'All' tab is selected. A list of tasks is displayed:

- .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 Pipelines 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 Pipelines 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 has a single task: 'Run on agent'. The 'File Copy' stage has a single task: 'File Copy', which is highlighted with a red border and has a warning message: 'Some settings need attention'.

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 * ① This setting is required.

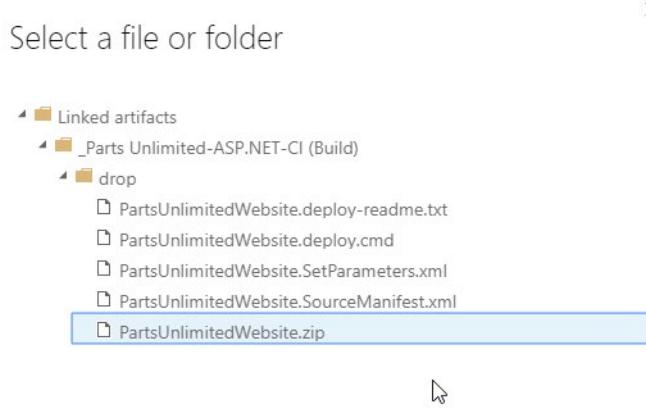
Azure Connection Type Azure Resource Manager

Azure Subscription * ① | Manage ② This setting is required.

Destination Type * ① This setting is required.

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

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 releases¹⁶](#).

Note: Task Groups are not currently supported in YAML. Use templates instead. See [Template References¹⁷](#)

Demonstration: Creating and managing task groups

In this demonstration, you will investigate Task Groups.

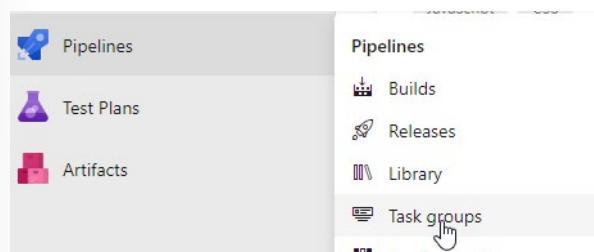
Note: Before starting this walkthrough, make sure you have performed the steps in the prerequisites section and the previous walkthroughs.

Steps

Let's now 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.

¹⁶ <https://docs.microsoft.com/en-us/azure/devops/pipelines/library/task-groups?view=vsts>

¹⁷ <https://docs.microsoft.com/en-us/azure/devops/pipelines/yaml-schema?view=azure-devops&tabs=schema%2Cparameter-schema#template-references>

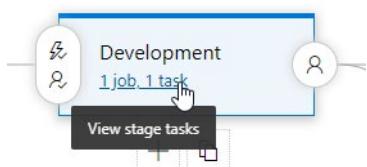
Task groups | Import | Security

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.

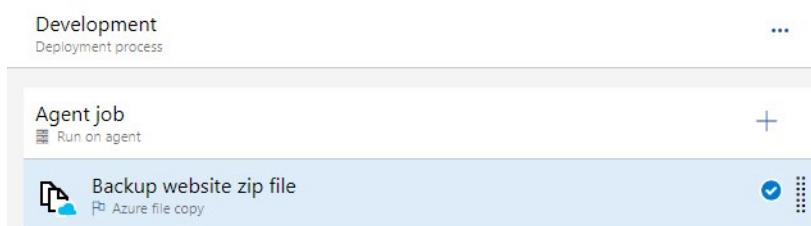
2. In the main menu, click **Pipelines** then click **Releases**, and click **Edit** to open the pipeline that we have been working on.



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.



4. Click the + sign to the right of the **Agent job** line to add a new task. In the **Search** box, type **data-base**.

The screenshot shows a search interface for Azure Pipelines tasks. At the top, there are buttons for 'Add tasks' and 'Refresh'. A search bar contains the text 'database'. Below the search bar, a list of tasks is displayed:

- 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.

The screenshot shows the 'Azure SQL DacpacTask' settings pane. It includes fields for 'Display name' (set to 'Azure SQL DacpacTask'), 'Azure Service Connection Type' (set to 'Azure Resource Manager'), and 'Azure Subscription' (dropdown menu). A note at the bottom states: '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 can reuse our service connection here

The screenshot shows the 'Deploy devopslog database' settings pane. It includes fields for 'Display name' (set to 'Deploy devopslog database'), 'Azure Service Connection Type' (set to 'Azure Resource Manager'), and 'Azure Subscription' (dropdown menu set to 'ARM Service Connection'). A note at the bottom states: '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]

- 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

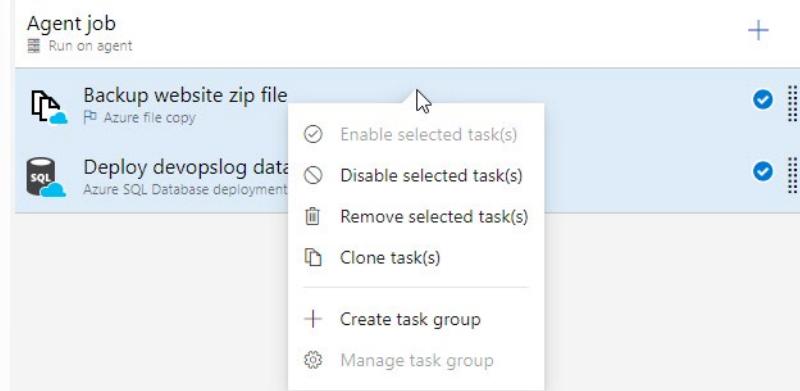
Inline SQL Script * ⓘ

`CREATE TABLE dbo.TrackingLog
(
 TrackingLogID int IDENTITY(1,1) PRIMARY KEY,
 TrackingDetails nvarchar(max)
);`

- Click **Save** then **OK** to save the work.

Now that we have two tasks, let's use them to create a task group.

- 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**.

Create task group X

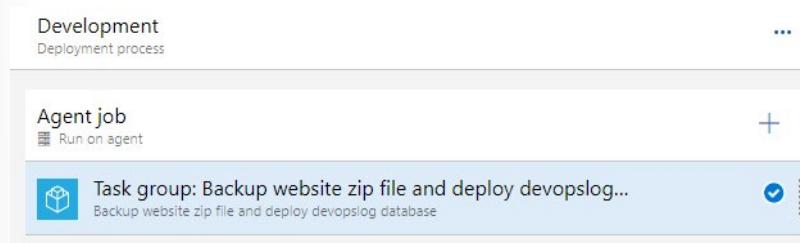
Name *

Description

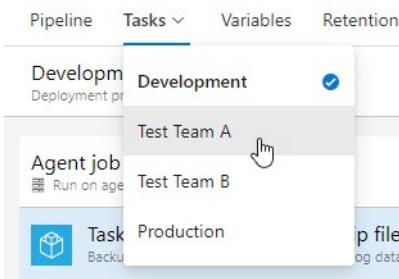
Category (i)

Create Cancel

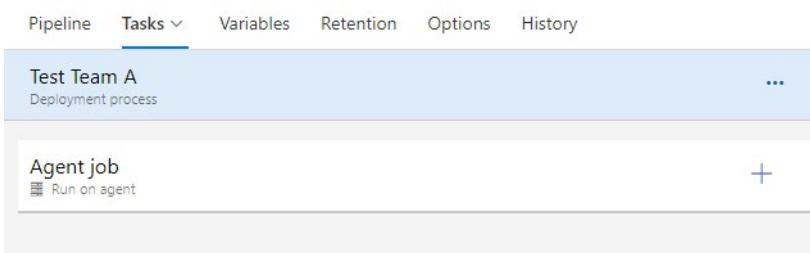
In the list of tasks, the individual tasks have now disappeared, and the new task group appears instead.



12. From the **Task** drop down list, select the **Test Team A** stage.

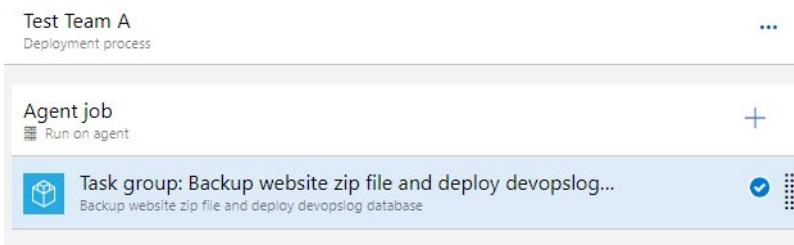


There are currently no tasks in the stage.



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.

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**.

Variables in release pipelines

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 debugging¹⁸**.

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 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 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.

¹⁸ <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/variables?view=vsts&tabs=batch>

The screenshot shows the 'Variables' section of 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. To the right of the search bar are 'List' and 'Grid' view options. On the far right, there's a 'Scope' dropdown menu with four items: 'Release', 'Dev', 'Test', and 'Release' again. The main area displays a table of variables:

Name	Value
Prefix	Demo
ServerName	DevServer
ServerName	TestServer
Password	*****

Below the table are icons for 'Edit', 'Delete', and a lock symbol.

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 TFS¹⁹](#).

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 look at how a release pipeline can make use of predefined sets of variables, called Variable Groups.

Like 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.

¹⁹ <https://docs.microsoft.com/en-us/azure/devops/pipelines/library/variable-groups?view=vsts>

Library

Variable groups Secure files | + Variable group Security Help

2. Click **+ Variable group** to commence creating a variable group. Set **Variable group name** to **Website Test Product Details**.

Library > Website Test Product Details*

Variable group Save Clone Security Help

Properties

Variable group name
Website Test Product Details

Description

Allow access to all pipelines

Link secrets from an Azure key vault as variables ⓘ

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.

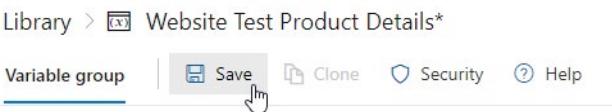
4. Add another variable called **Quantity** with a value of **12**.
5. Add another variable called **SalesUnit** with a value of **Each**.

Variables

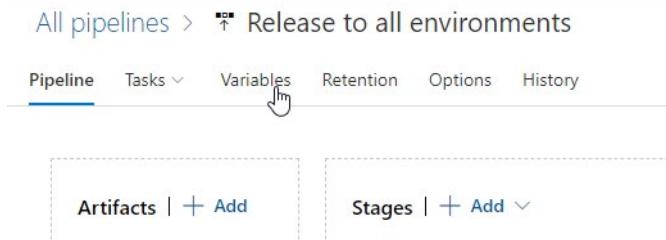
Name ↑	Value	⋮
ProductCode	REDPOLOXL	
Quantity	12	
SalesUnit	Each	

[+ Add](#)

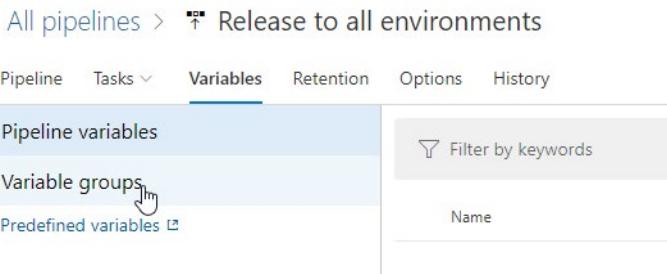
6. Click **Save** to save the new variable group.



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**.



8. In the left-hand pane, click **Variable Groups**.



Variable groups are linked to pipelines, rather than being directly added to them.

9. 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.

The screenshot shows a 'Link variable group' dialog box. At the top right is a search bar with a magnifying glass icon. Below it is a list box containing 'Website Test Product Details (3)' with a checked checkbox. The main body of the dialog is currently empty.

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 were 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 pipeline editor. The table lists variables under 'Variable groups': 'Website Test Product Details (3)' with a value of 'Scopes: Development,Test Team A,Test Team B'. At the bottom of the table is a 'Link variable group' button.

Variable group	Name	Value
Website Test Product Details (3)	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.

Custom build/release 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 task²⁰**.

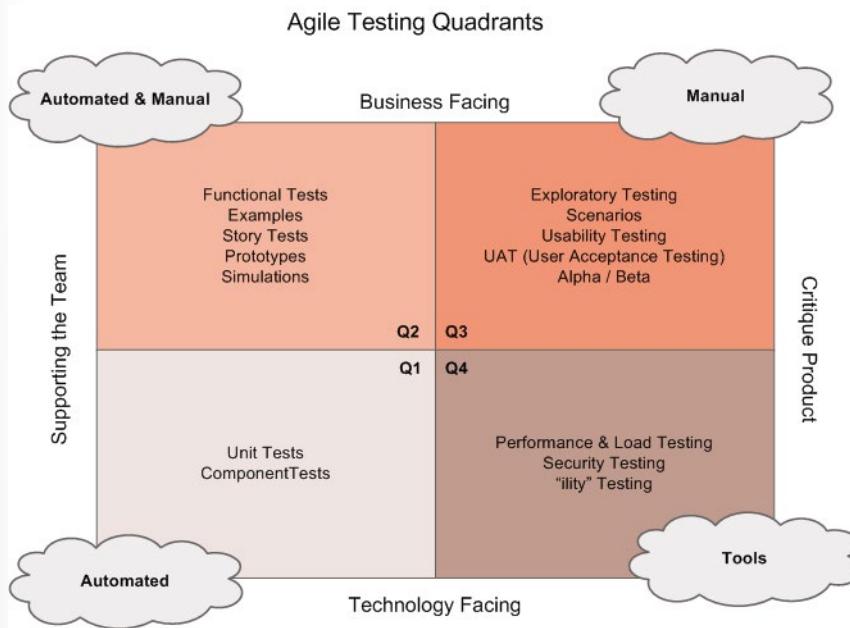
²⁰ <https://docs.microsoft.com/en-us/azure/devops/extend/develop/add-build-task?view=vsts>

Configure automated integration and functional test automation

Configure automated integration and functional test automation

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 - 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 - 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 - the tests are rather technical and non-meaningful to businesspeople. They are typical tests written and executed by the developers in a development team.
- Critique Product - 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 many 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 Tests²¹](#).

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 DevOps²²](#).

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 several HTTP calls that are executed in sequence.

For more information, see also:

- [Creating an Application Insights Web Test and Alert Programmatically²³](#)
- [Monitor the availability of any website²⁴](#)

²¹ <https://docs.microsoft.com/en-us/azure/devops/pipelines/test/set-up-continuous-test-environments-builds?view=vsts>

²² <https://docs.microsoft.com/en-us/azure/devops/test/load-test/overview?view=vsts>

²³ <https://azure.microsoft.com/nl-nl/blog/creating-a-web-test-alert-programmatically-with-application-insights/>

²⁴ <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. Release gates are discussed in an upcoming module.

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, & widgets²⁵](#).

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 repeatedly.

²⁵ <https://docs.microsoft.com/en-us/azure/devops/report/dashboards/overview?view=vsts>

The ability to receive Alerts and notifications is a powerful mechanism to get notified about certain events in your system when 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.

Alerts

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.

Target audience and delivery mechanism

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.

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 notifications²⁶**
- **Events, subscriptions, and notifications²⁷**

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.

Azure DevOps includes built-in support for the following Service Hooks:

Build and release	Collaborate	Customer support	Plan and track	Integrate
AppVeyor	Campfire	UserVoice	Trello	Azure Service Bus

²⁶ <https://docs.microsoft.com/en-us/azure/devops/notifications/index?view=vsts>

²⁷ <https://docs.microsoft.com/en-us/azure/devops/notifications/concepts-events-and-notifications?view=vsts>

Build and release	Collaborate	Customer support	Plan and track	Integrate
Bamboo	Flowdock	Zendesk		Azure Storage
Jenkins	HipChat			Web Hooks
MyGet	Hubot			Zapier
Slack				

This list will change over time.

To learn more about service hooks and how to use and create them, read [Service Hooks in Azure DevOps](#)²⁸.

Demonstration: Setting up service hooks to monitor the pipeline

In this demonstration, you will investigate Service Hooks.

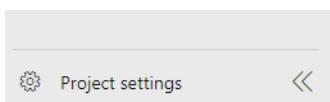
Note: Before starting this walkthrough, make sure you have performed the steps in the prerequisites section and the previous walkthroughs.

Steps

Let's now look at how a release pipeline can communicate with other services by using service hooks.

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 look.

1. Below the main menu for the **Parts Unlimited** project, click **Project settings**.



2. In the **Project settings** menu, click **Service hooks**.

The screenshot shows the 'Service hooks' page in the Azure DevOps 'Project Settings'. On the left, a sidebar lists various project settings like General, Overview, Teams, Security, Notifications, Service hooks (which is highlighted), and Dashboards. The main content area is titled 'Service Hooks' and contains the text: 'Integrate with your favorite services by notifying them when events happen in your project.' Below this is a blue button labeled '+ Create subscription'.

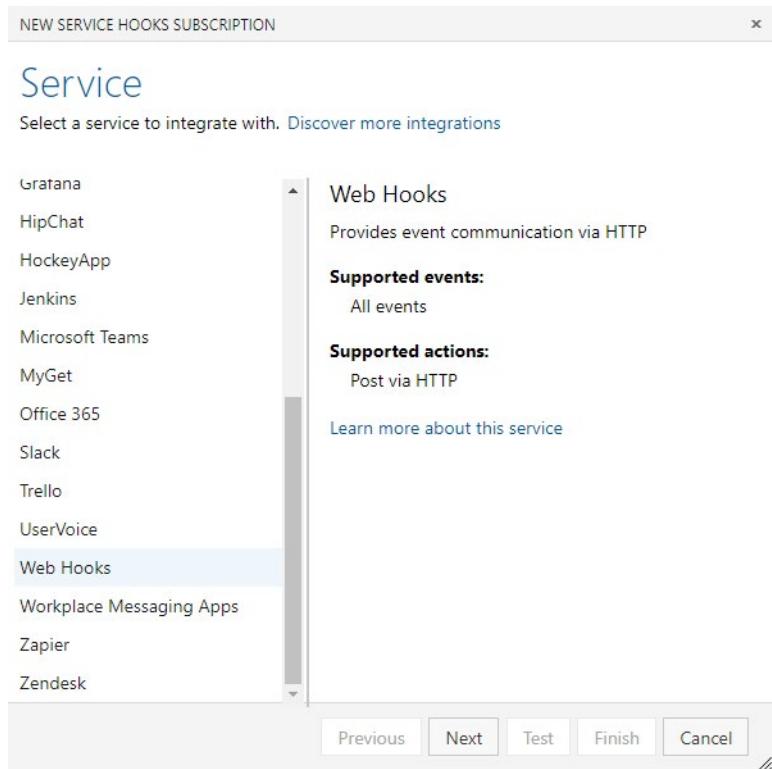
3. Click **+Create subscription**.

²⁸ <https://docs.microsoft.com/en-us/azure/devops/service-hooks/overview?view=vsts>

The screenshot shows a dialog box titled "NEW SERVICE HOOKS SUBSCRIPTION". The main title is "Service". Below it is a sub-header "Select a service to integrate with. Discover more integrations". A vertical list of services is on the left, and detailed information about the selected service is on the right. The selected service is "App Center".
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](#)
At the bottom are buttons: 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**.



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**.

NEW SERVICE HOOKS SUBSCRIPTION

Service

Select a service to integrate with. Discover more integrations

Azure Storage

Microsoft Azure Storage is a service for storing large numbers of messages that can be accessed from anywhere in the world. It is useful for creating a backlog of work to process asynchronously.

Supported events:

All events

Supported actions:

Insert a message in a Storage Queue

Learn more about this service

App Center
AppVeyor
Azuqua
Azure App Service
Azure Service Bus
Azure Storage
Bamboo
Campfire
Flowdock
Grafana
HipChat
HockeyApp
Jenkins
Microsoft Teams

Previous Next Test Finish 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.

Trigger on this type of event

Build completed

Build completed

Code pushed

Pull request commented on

Pull request created

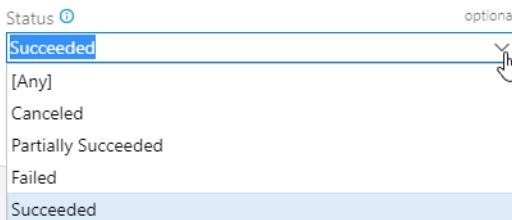
Pull request merge attempted

Pull request updated

Release abandoned

Release created

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 ✓

OPTIONS

Release pipeline name optional
devopsoutput ✓

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/vY...

Connection string
DefaultEndpointsProtocol=https;AccountName=devopsoutput;AccountKey=ApxxJ9Dvs2dguErzh/...

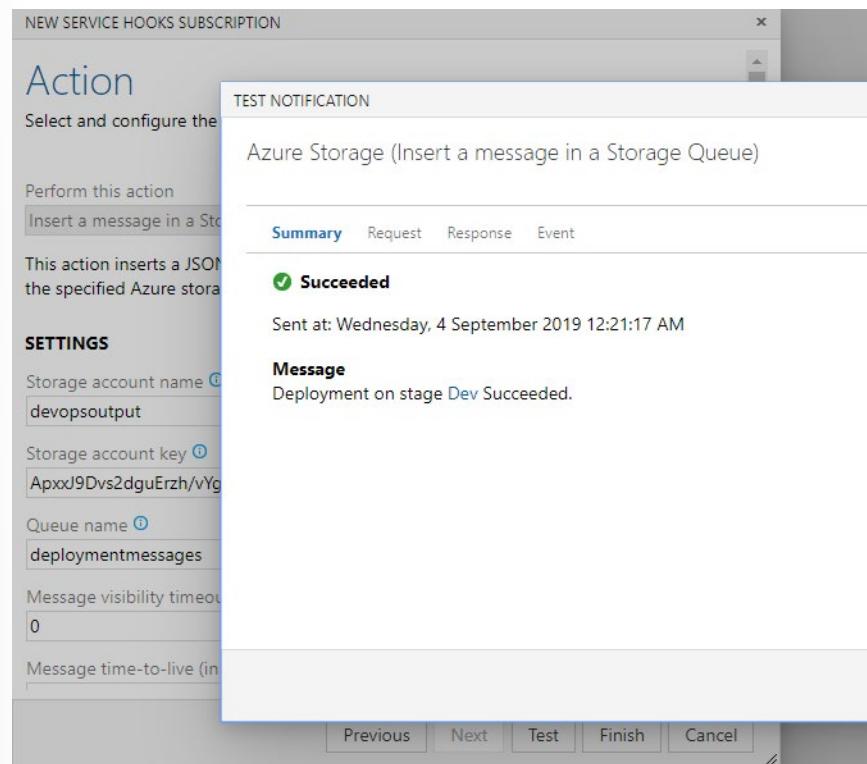
11. Back in the **Action** page in Azure DevOps, paste in the key.

SETTINGS

Storage account name required
 ✓

Storage account key required
 ✓

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.

Consumer ↑	Event ↑	Action
Azure Storage	... Release deployment c... Release Release to all environm... Insert a message in a S...	Account

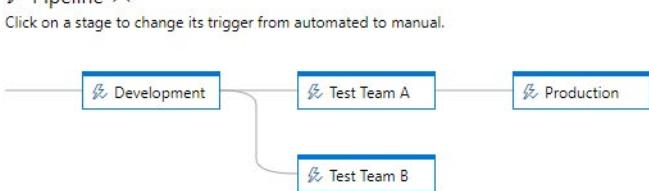
Create a release to test the service hook

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



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

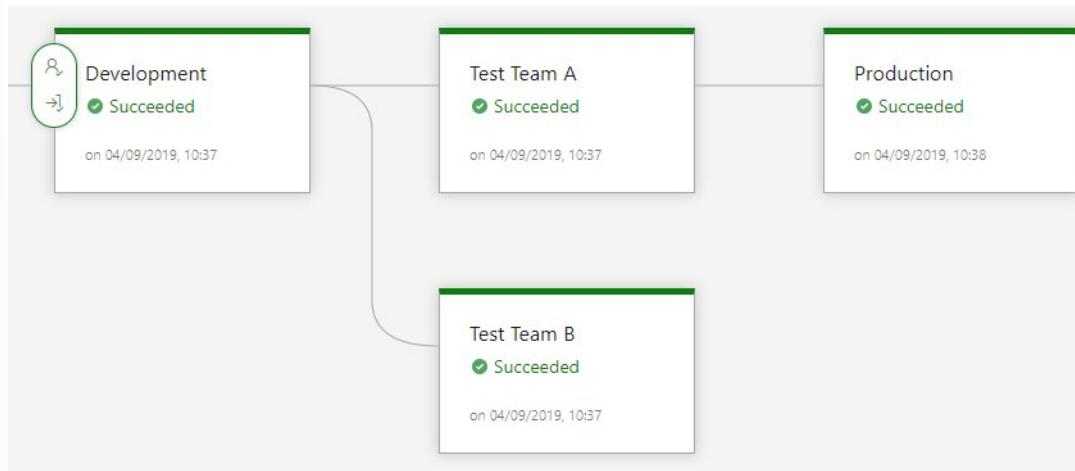
Test the queue service hook

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.

The screenshot shows the Azure Storage Queues blade for the **devopsoutput** storage account. The left sidebar lists services: File service, Tables, Queue service, and Queues. The Queue service is selected. The main area shows the **deploymentmessages** queue with its URL: <https://devopsoutput.queue.core.windows.net/deploymentmessages>.

18. Click to open the **deploymentmessages** queue.

The screenshot shows the details of the **deploymentmessages** queue. The left sidebar includes tabs for Overview, Access Control (IAM), Settings, Access policy, and Metadata. The Overview tab is selected. The main area displays the queue's properties: Refresh, Add message, Dequeue message, and Clear queue. It also shows the Authentication method: Access key (Switch to Azure AD User Account). The queue contains two messages:

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

Lab overview

Many teams prefer to define their build and release pipelines using YAML. This allows them to access the same pipeline features as those using the visual designer, but with a markup file that can be managed like any other source file. YAML build definitions can be added to a project by simply adding the corresponding files to the root of the repository. Azure DevOps also provides default templates for popular project types, as well as a YAML designer to simplify the process of defining build and release tasks.

Objectives

After you complete this lab, you will be able to:

- configure CI/CD pipelines as code with YAML in Azure DevOps

Lab duration

- Estimated time: **60 minutes**

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- [AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions²⁹](https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions)

Setting up and running functional tests

Lab overview

Selenium³⁰ 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, you will learn how to execute Selenium test cases on a C# web application, as part of the Azure DevOps Release pipeline.

Objectives

After you complete this lab, you will be able to:

- Configure a self-hosted Azure DevOps agent
- Configure release pipeline
- Trigger build and release
- Run tests in Chrome and Firefox

²⁹ <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions>/

³⁰ <http://www.seleniumhq.org/>

Lab duration

- Estimated time: **60 minutes**

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- **AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions³¹**

³¹ <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/>

Module review and takeaways

Module review questions

Review Question 1

How many deployment jobs can be run concurrently by a single agent?

Review Question 2

What should you create to store values that you want to make available across multiple build and release pipelines?

Review Question 3

How can you provision the agents for deployment groups in each of your VMs?

Review Question 4

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

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 Implementing an Appropriate Deployment Pattern

Module overview

Module overview

This module is about implementing an appropriate deployment pattern.

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

Introduction to continuous delivery and continuous deployment

Continuous Delivery is an extension of continuous integration. It's all about getting changes to customers quickly and using methods that are sustainable. Continuous delivery goes further, and changes that pass through production pipelines are released to customers.

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 change the code quickly and reliably. Writing high-quality software and high-quality tests are, therefore, an essential part of Continuous Delivery.

Architecture

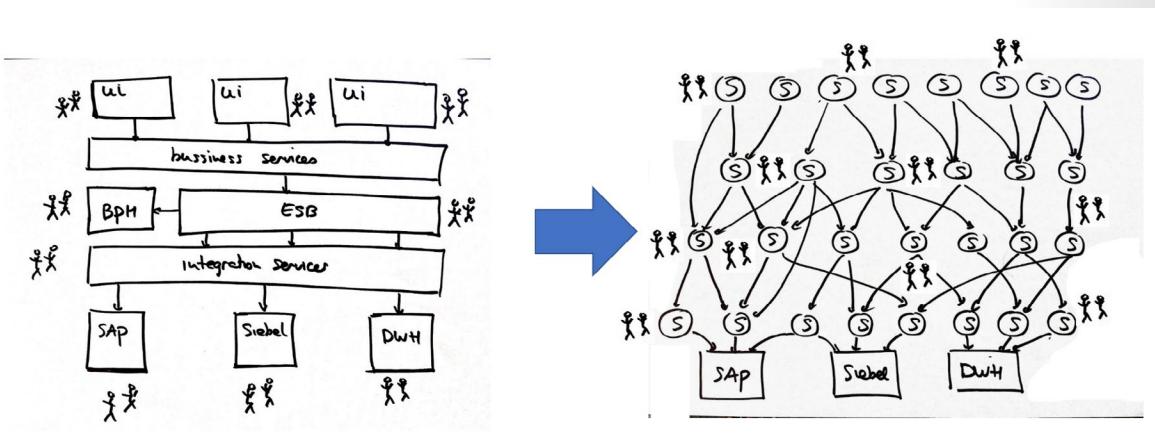
The architecture of your application is always significant. But when implementing Continuous Delivery, it is maybe even more so. If your software is a monolith with a lot of tight coupling between the various components, it is difficult to deliver your software continuously. Every part that is changed 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. One approach to solving these issues is to implement microservices.

Microservices architecture

Today, you will 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.

Classical 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 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. Despite 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.

Modern deployment patterns

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 releases
- Dark launching
- A/B testing
- Progressive exposure or ring-based deployment
- Feature toggles

We will talk about the different patterns in the upcoming lessons.

Discussion: A critical look at your architecture

Are your architecture and the current state of your software ready for Continuous Delivery?

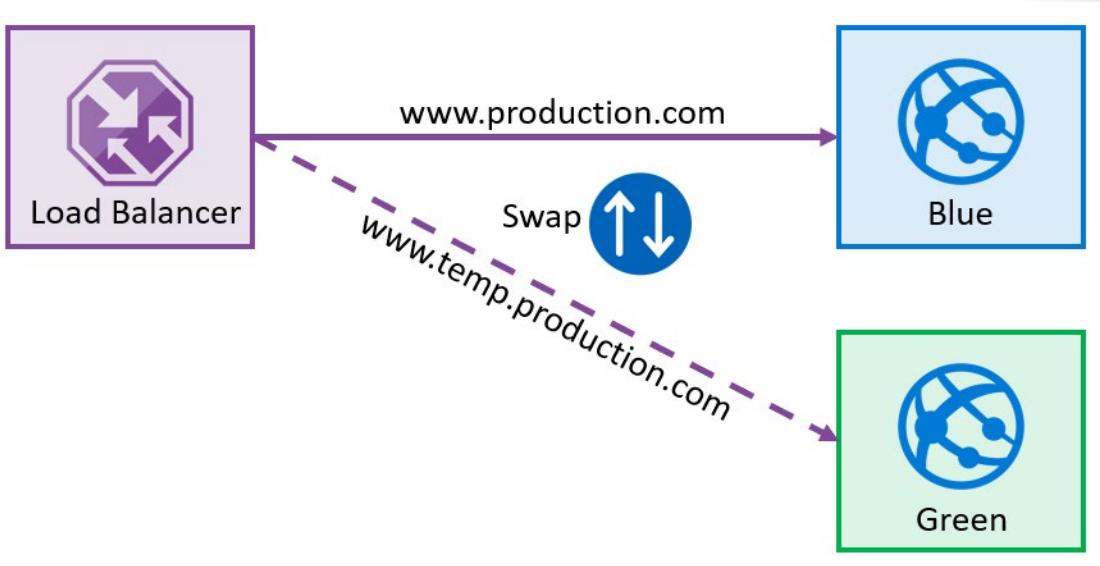
Topics you might want to consider are:

- 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 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 with a blue header bar containing 'Choose a template'. Below the header, there are five project templates listed:

- Tailwind Traders**: An ASP.NET & React application using Azure App Service, AKS, Cosmos DB, Logic App, and Function App. Tags: Agile, Data and AI, React.
- SmartHotel360**: A template for the public web site of SmartHotel360, an E2E reference sample app with several consumer and line-of-business apps and an Azure backend. Tags: scrum, aspnetcore, 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. The template also includes pipeline definition to build and deploy the web app to Azure App Service. Tags: scrum, aspnetcore, azureappservice.
- PartsUnlimited**: A scrum based team project using Azure App Service, ASP.NET, and MySQL. Tags: scrum, aspnetcore, azureappservice.
- PartsUnlimited-YAML**: A scrum based team project using Azure App Service, ASP.NET, and MySQL. Tags: scrum, aspnetcore, azureappservice.
- MyShuttle**: A Java application using Azure App Service, MySQL, and Application Insights. Tags: scrum, java, application, azure web app, mysql.

At the bottom right of the window 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

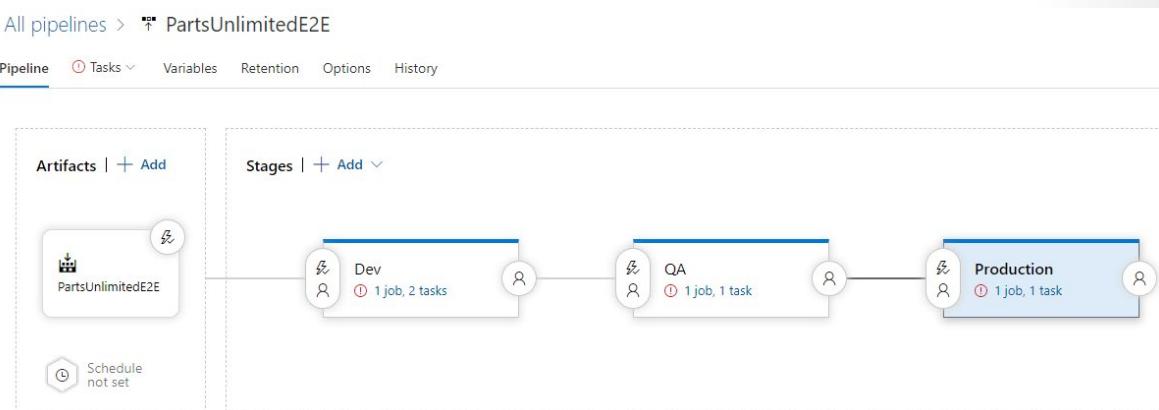
Logs Summary Tests WhiteSource Bolt Build Report

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 "Action" is set to "Create or update resource group". The "Resource group" is set to "\${ResourceGroupName}" and the "Location" is set to "East US" (both with red boxes around them). Under "Template", the "Template location" is set to "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 ⌂

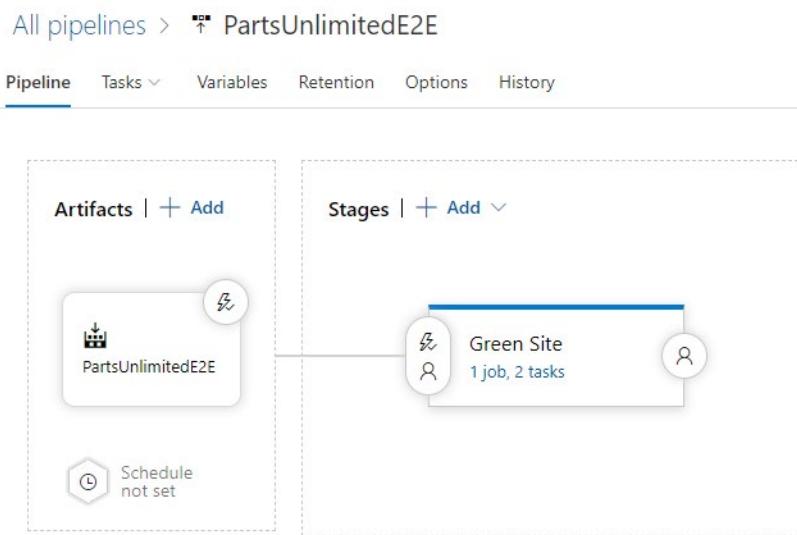
Agent pool ⓘ | Pool information | Manage ⓘ

Azure Pipelines

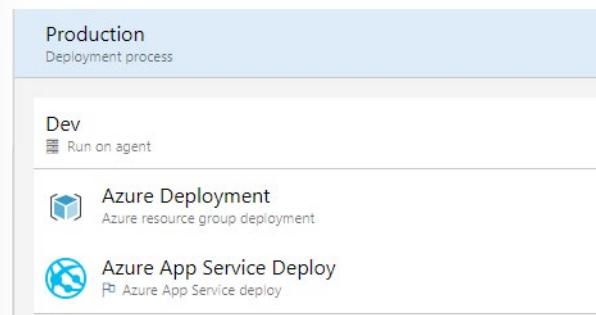
Agent Specification *

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**.

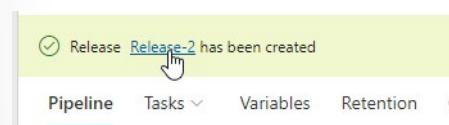


12. Click the **Azure App Service Deploy** task and uncheck the **Deploy to slot** option. Click **Save** and **OK**.

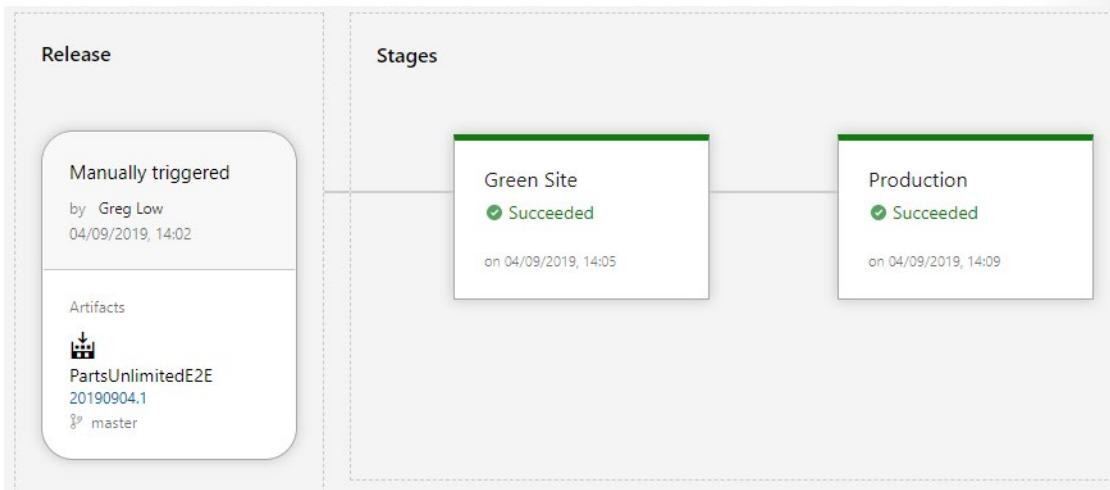
The screenshot shows the configuration dialog for the 'Azure App Service Deploy' task. It includes fields for 'App Service name' (set to '\$(WebsiteName)'), 'Virtual application' (empty), and a checkbox for 'Deploy to slot' (unchecked). There is also an 'OK' button at the bottom right.

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.

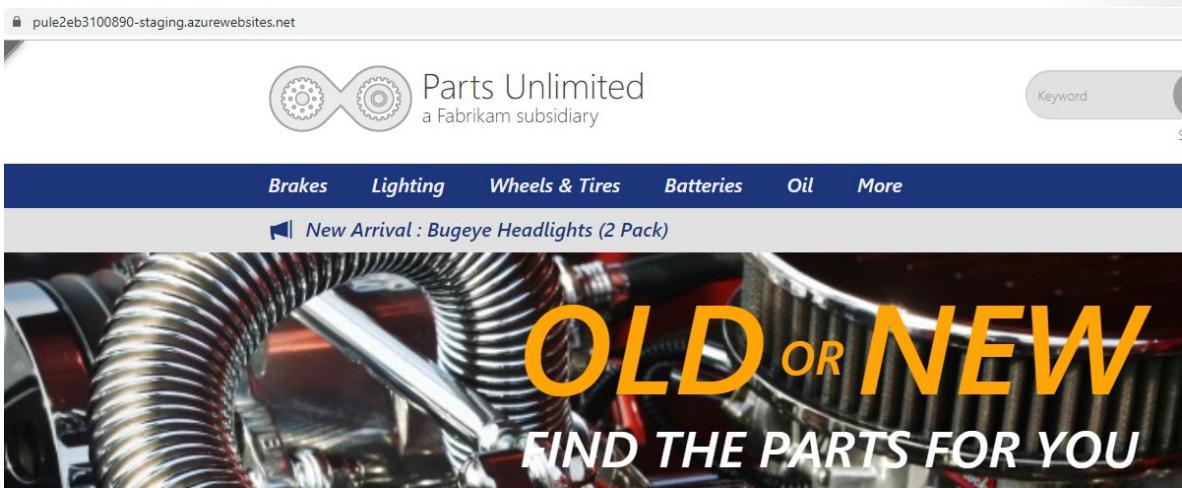


Test the green site and the production site

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.

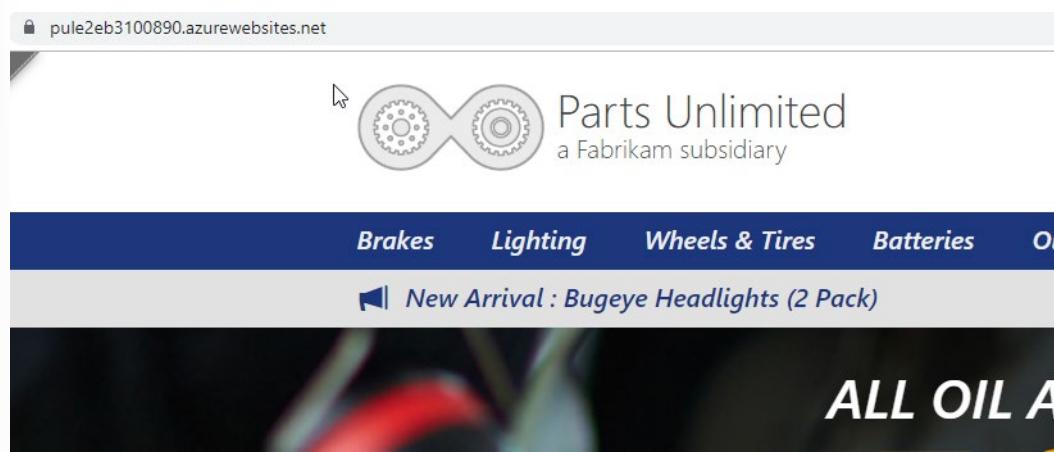
Resource group (change) : ASPDOTNET	URL : https://pule2eb3100890-staging.azurewebsites.net
Status : Running	App Service Plan : pule2e (S1: 1)
Location : East US	https://pule2eb3100890-staging.azurewebsites.net

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	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**.

The screenshot shows the Azure DevOps task catalog interface. At the top, there is a search bar with the text 'cli'. Below the search bar, there are three task templates listed:

- 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.

The screenshot shows the configuration pane for the Azure CLI task. The task is named 'Azure CLI' and is set to version 1.*. The configuration fields include:

- Display name ***: Azure CLI
- Azure subscription ***: A dropdown menu showing a single item.
- Script Location ***: A dropdown menu showing 'Script path'.
- Script Path ***: An input field containing 'az webapp deployment slot swap -g \$(ResourceGroupName) -n \$(WebsiteName) --slot Staging --target-slot production'.
- Arguments**: An input field.

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
```

Azure CLI ⓘ

Task version 1.*

Display name *

Azure CLI

Azure subscription * ⓘ | Manage ⓘ

Microsoft

Scoped to subscription 'Microsoft Azure Sponsorship'

Script Location * ⓘ

Inline script

Inline Script * ⓘ

```
az webapp deployment slot swap -g $(ResourceGroupName) -n $(WebsiteName) --slot Staging --target-slot production
```

23. From the menu above the task list, click **Pipeline**. Click the **Pre-deployment conditions** icon for the **Swap Blue-Green** stage, 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.

master PartsUnlimited / PartsUnlimited-aspNet45 / src / PartsUnlimitedWebsite / Views / Home / Index.cshtml

Contents History Compare Blame Edit Rename Delete Download

```
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.

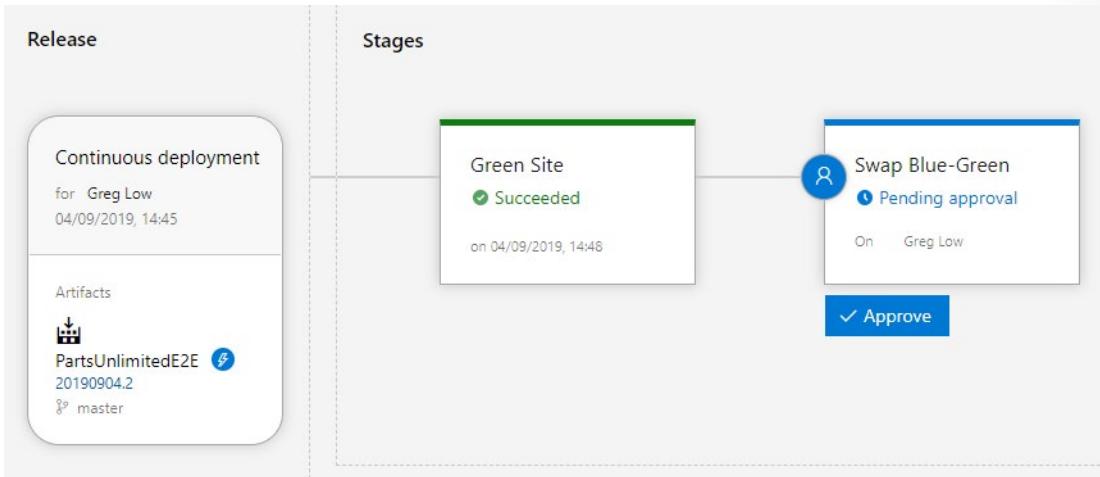
```
<div class="item" style="background-image: url('/Images/hero_image.jpg')>
  <a href="@Url.Action("Browse", "Store", new { categoryId = 3 })">
    <div class="container">
      <p>New tyres</p>
      <ul>
        <li>Improved fuel efficiency</li>
        <li>Superior wet weather breaking</li>
        <li>Added durability</li>
      </ul>
    </div>
  </a>
</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
GL	Updated Index.cshtml CI build for Greg Low		✓ 20190904.2	master
GL	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.

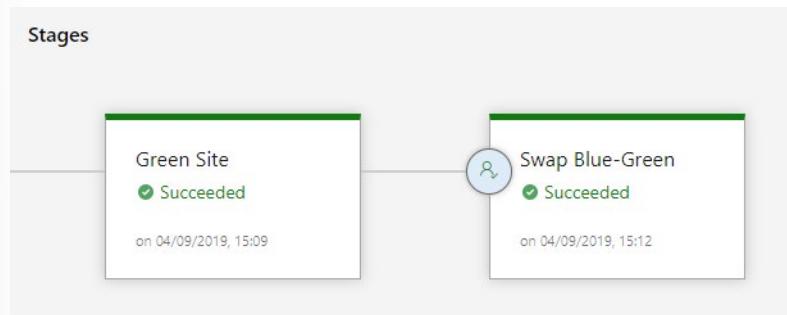
29. Refresh the Green site (i.e., Staging slot) browser tab and see if your change has appeared. It now shows the altered word.



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. 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?

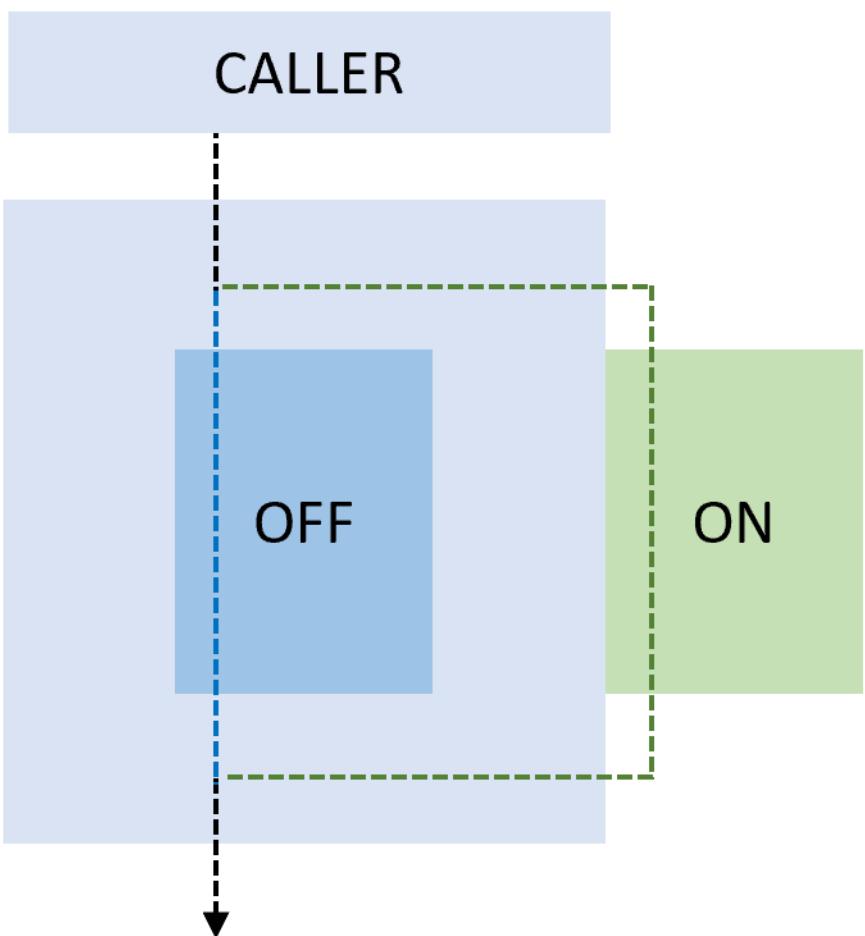
Feature toggles are also known as feature flippers, feature flags, feature switches, conditional features, etc.

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

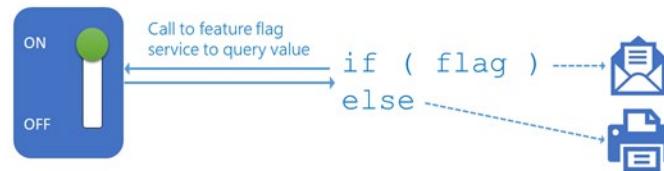
A feature toggle is just code. And to be more specific, conditional code. It adds complexity to the code and increases the technical debt. Be aware of that when you write them, and clean up when you do not need them anymore.

While feature flags can be useful, they can also introduce many issues of their own.

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.

Planning feature flag lifecycles



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.

As soon as you introduce a feature flag, you have 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.

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 that they thought was no longer part of the code, for a new purpose.

³ <https://docs.microsoft.com/en-us/azure/devops/articles/phase-features-with-feature-flags?view=vsts>

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.

Azure App Configuration offers a Feature Manager. See [Azure App Configuration Feature Manager⁴](#)

⁴ <https://docs.microsoft.com/en-us/azure/azure-app-configuration/manage-feature-flags>

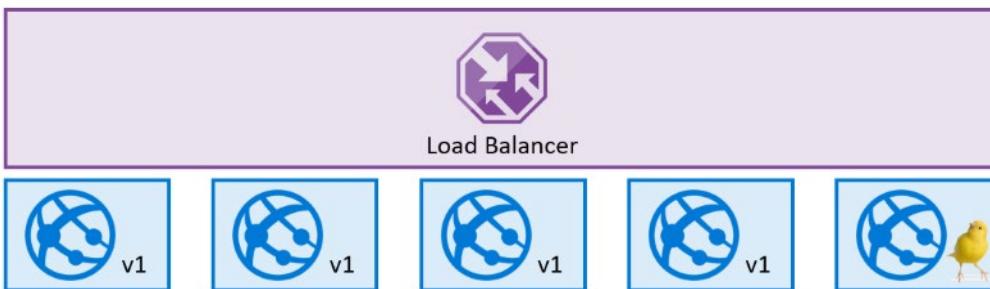
Canary releases

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.



Canary releases can be implemented using a combination of feature toggles, traffic routing, and deployment slots.

- You can route a percentage of traffic to a deployment slot with the new feature enabled.
- You can target a specific user segment by using feature toggles.

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 are 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.

While the available options can change over time, Traffic manager currently provides six options to distribute traffic:

- **Priority:** Select Priority when you want to use a primary service endpoint for all traffic and provide backups in case the primary or the backup endpoints are unavailable.
- **Weighted:** Select Weighted when you want to distribute traffic across a set of endpoints, either evenly or according to weights, which you define.
- **Performance:** Select Performance when you have endpoints in different geographic locations, and you want end users to use the “closest” endpoint in terms of the lowest network latency.
- **Geographic:** Select Geographic so that users are directed to specific endpoints (Azure, External, or Nested) based on which geographic location their DNS query originates from. This empowers Traffic Manager customers to enable scenarios where knowing a user’s geographic region and routing them based on that is important. Examples include complying with data sovereignty mandates, localization of content & user experience and measuring traffic from different regions.
- **Multivalue:** Select MultiValue for Traffic Manager profiles that can only have IPv4/IPv6 addresses as endpoints. When a query is received for this profile, all healthy endpoints are returned.
- **Subnet:** Select Subnet traffic-routing method to map sets of end-user IP address ranges to a specific endpoint within a Traffic Manager profile. When a request is received, the endpoint returned will be the one mapped for that request’s source IP address.

When we look at the options the traffic manager offers, the most used option for Continuous Delivery is the option to route traffic based on weights. (Note: traffic is only routed to endpoints that are currently available).

For more information, see also:

- [What is Traffic Manager?](#)⁵
- [How Traffic Manager works](#)⁶
- [Traffic Manager Routing Methods](#)⁷

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.

⁵ <https://docs.microsoft.com/en-us/azure/traffic-manager/traffic-manager-overview>

⁶ <https://docs.microsoft.com/en-us/azure/traffic-manager/traffic-manager-how-it-works>

⁷ <https://docs.microsoft.com/en-us/azure/traffic-manager/traffic-manager-routing-methods>

Dark launching

Dark launching

Dark launching is in many ways like 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.

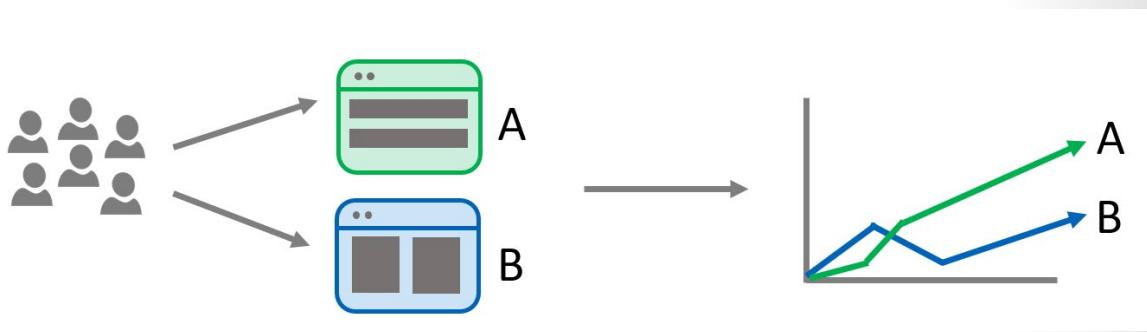
How to implement dark launching

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.

A/B 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 deliver MVP's to a production environment and your end-users.

Common aims are to experiment with new features, often to see if they improve conversion rates. Experiments are often continuous, and the impact of change is measured.

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.

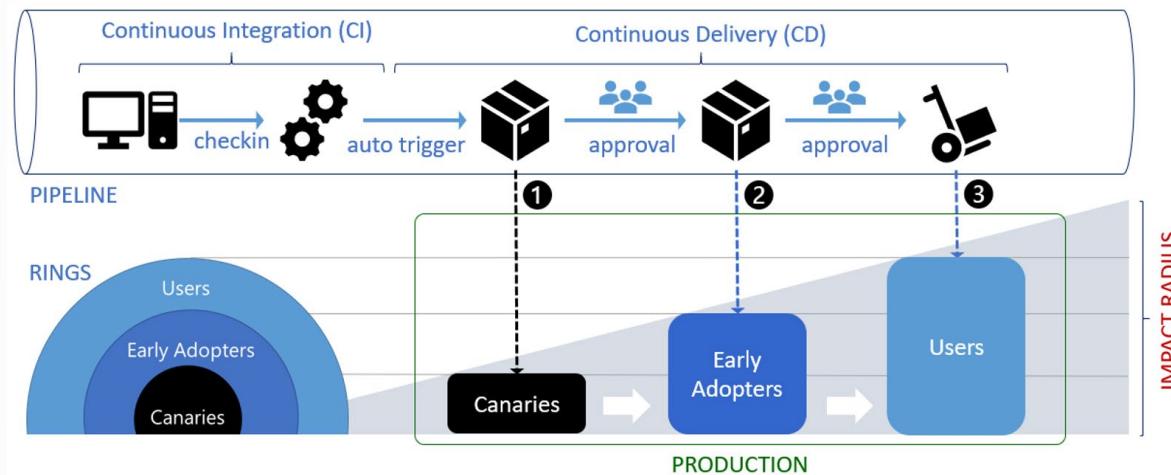
Progressive exposure deployment

CI/CD with deployment rings

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.

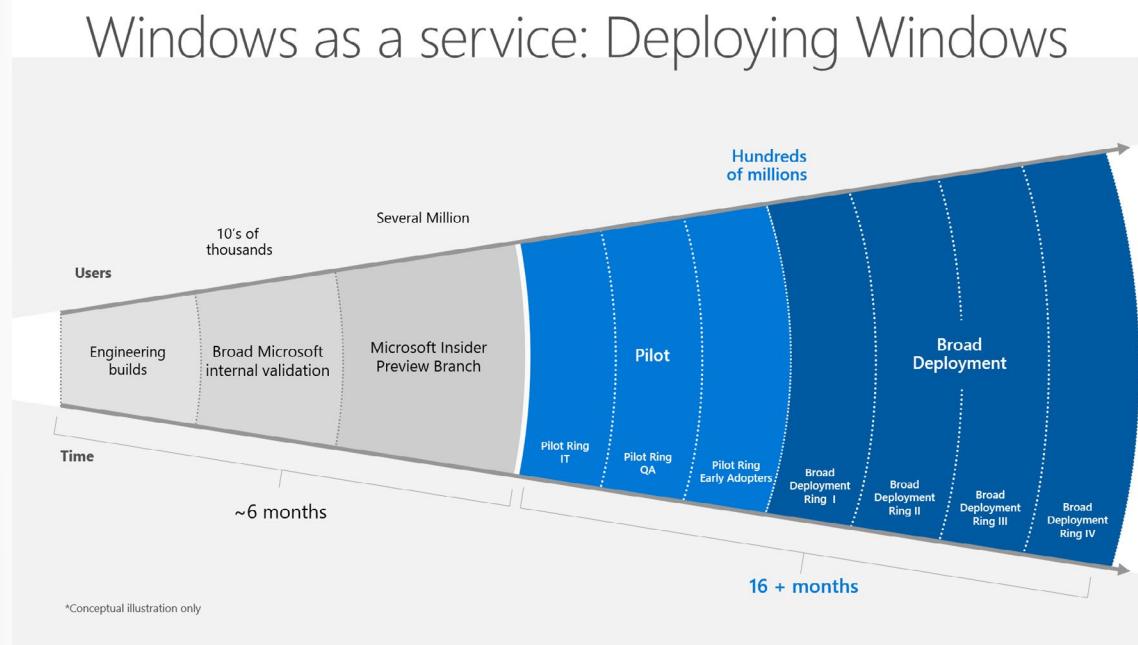
In DevOps, rings are typically modeled as stages.

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.



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 users⁸](#).

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 look at how a release pipeline can be used to stage features by using ring-based deployments.

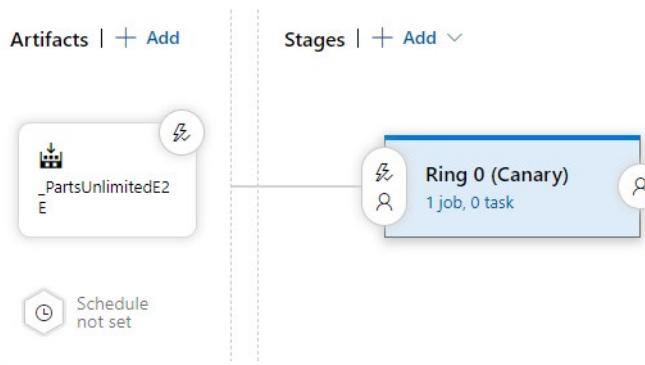
When I have a new feature, I might want to just release it to a small number of users at first, just in case something goes wrong. In authenticated systems, I could do this by having those users as members of a security group and letting members of that group use the new features.

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)**.

⁸ <https://docs.microsoft.com/en-us/azure/devops/articles/phase-rollout-with-rings?view=vsts>



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.

The screenshot shows the configuration pane for the Azure CLI task. At the top, it says 'Azure CLI' with a help icon, and 'View YAML' and 'Remove' buttons. Below that is a dropdown for 'Task version' set to '1.*'. The 'Display name *' field contains 'Azure CLI'. Under 'Azure subscription *', there is a dropdown menu with a placeholder 'Select an Azure subscription...' and a note '(1) This setting is required.' Below that is a 'Script Location *' section with a dropdown for 'Script path' and a note '(1)'. Further down is a 'Script Path *' section with a dropdown and a note 'This setting is required.'.

6. 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
```

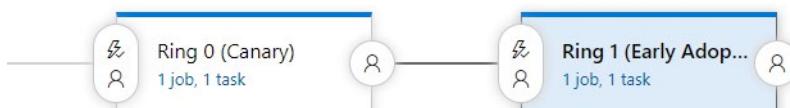
This distribution setting will cause 10% of the web traffic to be sent to the new feature Site (i.e., currently the staging slot).

7. 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

8. 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 ▾



9. 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
```

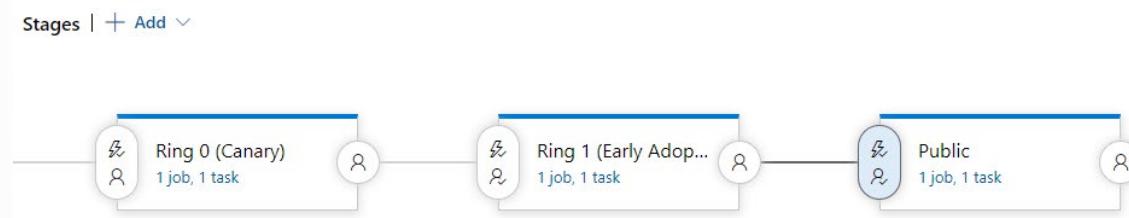
This allows us to move the new feature into wider distribution if it was working ok in the smaller set of users.

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**.



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
```

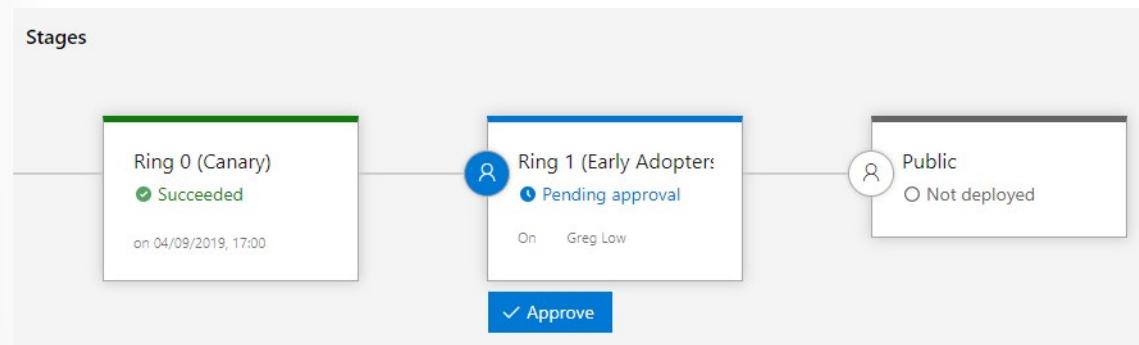
14. Click **Save** then **OK** to save your work.

15. Click **Create release** and **Create** to start a release. Click the release link to see the progress.

All pipelines > ⚡ Ring-based Deployment



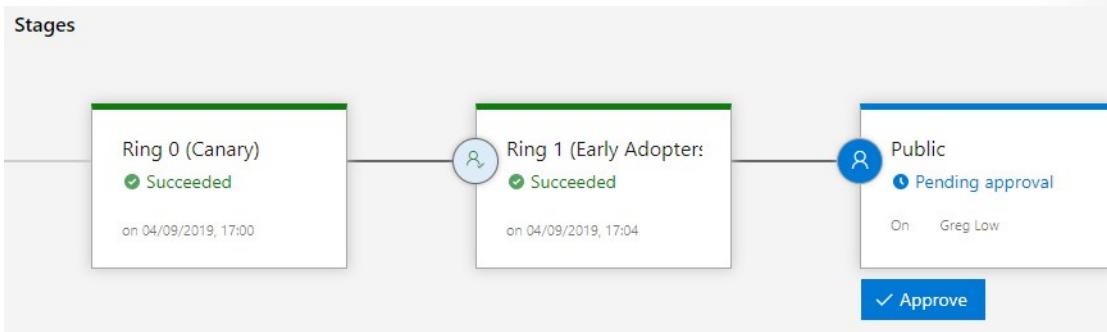
Wait until Ring 0 (Canary) has succeeded.



At this point, 10% of the traffic will be going to the new feature site.

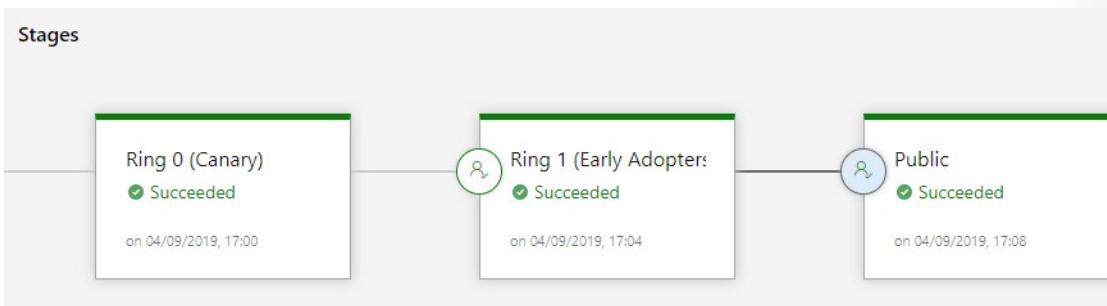
16. Click **Approve** on the **Ring 1 (Early Adopters)** stage, and then **Approve**.

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 the traffic will now be going to the swapped production site, running the new code.



The new feature has been fully deployed.

Lab

Feature flag management with LaunchDarkly and Azure DevOps

Lab overview

LaunchDarkly⁹ is a continuous delivery platform that provides feature flags as a service. LaunchDarkly gives you the power to separate feature rollout from code deployment and manage feature flags at scale. Integration of LaunchDarkly with Azure DevOps minimizes potential risks associated with frequent releases. To further integrate releases with your development process, you can link feature flag roll-outs to Azure DevOps work items.

In this lab, you will learn how to optimize management of feature flags in Azure DevOps by leveraging LaunchDarkly.

Objectives

After you complete this lab, you will be able to:

- Create feature flags in LaunchDarkly
- Integrate LaunchDarkly with Web applications
- Automatically roll-out LaunchDarkly feature flags as part of Azure DevOps release pipelines

Lab duration

- Estimated time: **60 minutes**

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- **AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions**¹⁰

⁹ <https://launchdarkly.com/>

¹⁰ <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/>

Module review and takeaways

Module review questions

Review Question 1

What is the easiest way to create a staging environment for an Azure webapp?

Review Question 2

What Azure-based tool can you use to divert a percentage of your web traffic to a newer version of an Azure website?

Review Question 3

What characteristics make users suitable for working with canary deployments?

Review Question 4

What is a potential disadvantage of using canary deployments?

Review Question 5

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 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 Managing Infrastructure and Configuration using 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

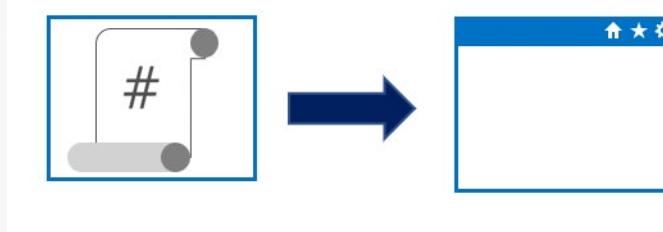
The following list are benefits of configuration management:

- 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

Imperative versus declarative configuration

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.

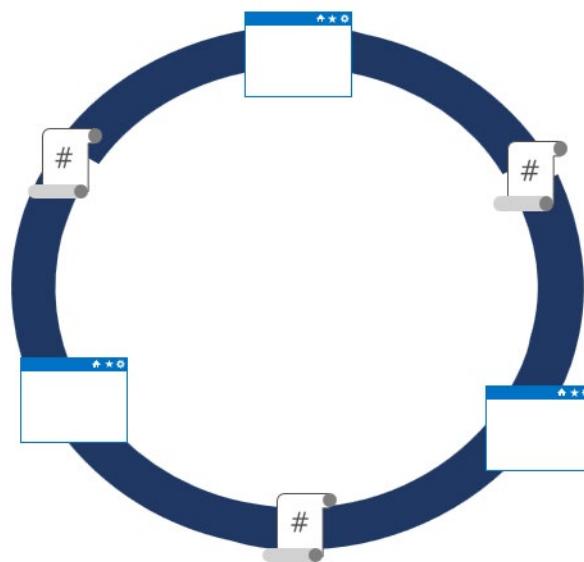
There is no absolute on which is the best approach to take, and individual tools may be able to be used in either *declarative* or *imperative* forms. The best approach for you to take will depend on your needs.

Idempotent configuration

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 Queues¹](https://www.wintellect.com/idempotency-for-windows-azure-message-queues).

¹ <https://www.wintellect.com/idempotency-for-windows-azure-message-queues/>

Create Azure resources using ARM templates

Why use ARM templates?

Using Resource Manager templates will make your deployments faster and more repeatable. For example, you no longer must 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 like 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 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. 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.

Azure provides many QuickStart templates. You might use these as a base for your work.

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."  

```

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. Like 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]"
    }
}
```

Manage dependencies

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 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://linkedtemplateeek1store.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:

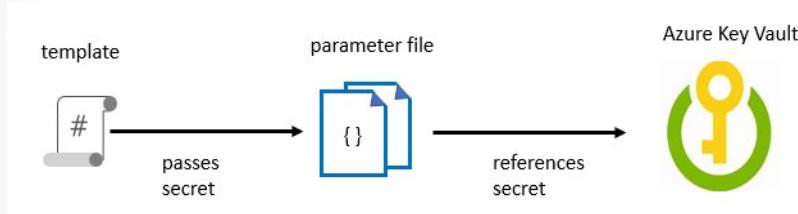
```
keyVaultName='{your-unique-vault-name}'  
resourceGroupName='{your-resource-group-name}'  
location='centralus'  
userPrincipalName='{your-email-address-associated-with-your-subscription}'  
  
# Create a resource group  
az group create --name $resourceGroupName --location $location  
  
# Create 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.json²](#)) 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:

² <https://github.com/Azure/azure-docs-json-samples/blob/master/azure-resource-manager/keyvaultparameter/sqlserver.json>

```
{  
    "$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 name of the secret. In the following parameter file (also available at [GitHub - keyvaultparameter³](#)), 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": {  
            "value": "P@ssw0rd!"  
        }  
    }  
}
```

³ <https://github.com/Azure/azure-docs-json-samples/blob/master/azure-resource-manager/keyvaultparameter/sqlserver.parameters.json>

```
    "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 deployment⁴** for more details. There also are details available on this webpage for how to reference a secret with a dynamic ID as well.

⁴ <https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-manager-keyvault-parameter>

Create Azure resources by 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.

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.

So, how do you find the 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 blob
```

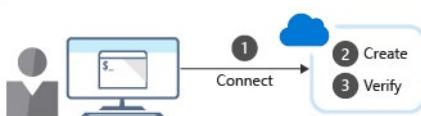
If you already know the name of the command you want, the help argument for that command will get you more detailed information on the command, and for a command group, a list of the available subcommands. For example, here's how you would get a list of the subgroups and commands for managing blob storage:

```
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**⁵ 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>']"
```

⁵ <https://aka.ms/devicelogin>

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/>⁶

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
```

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

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 today](#)⁷ 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 deployment group 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
```

⁶ <http://jmespath.org/>

⁷ <https://azure.microsoft.com/en-us/free/>

4. Deploy the resource by running the following command, substituting the same values as earlier:

```
az deployment group 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:

```
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.

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 Automation⁸**.



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 Overview⁹**.
- 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 Azure¹⁰**.
- 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 Automation¹¹**.

⁸ <https://azure.microsoft.com/en-us/documentation/articles/automation-intro/>

⁹ <https://docs.microsoft.com/en-us/azure/automation/automation-dsc-overview>

¹⁰ <https://docs.microsoft.com/en-us/azure/automation/automation-update-management>

¹¹ <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**¹²
- 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**¹³.
- 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**¹⁴ from within the Azure portal. Steps to create an Azure Automation account are available on the **Create an Azure Automation account**¹⁵ page.

Automation accounts are like 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.

¹² <https://docs.microsoft.com/en-us/azure/automation/source-control-integration>

¹³ <https://docs.microsoft.com/en-us/azure/automation/automation-config-aws-account>

¹⁴ <https://azure.microsoft.com/en-us/documentation/articles/automation-security-overview/>

¹⁵ <https://docs.microsoft.com/en-us/azure/automation/automation-quickstart-create-account>

* Name [i](#)
azautoac01

* Subscription
Pay-As-You-Go

* Resource group
(New) az-auto-rg

Create new

* Location
Japan East

* Create Azure Run As account [i](#)

Yes No

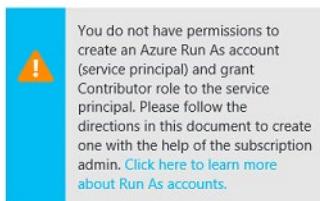
The Run As account feature will create a Run As account and a Classic Run As account. [Click here to learn more about Run As accounts.](#)

Learn more about Automation [i](#)

Create

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:

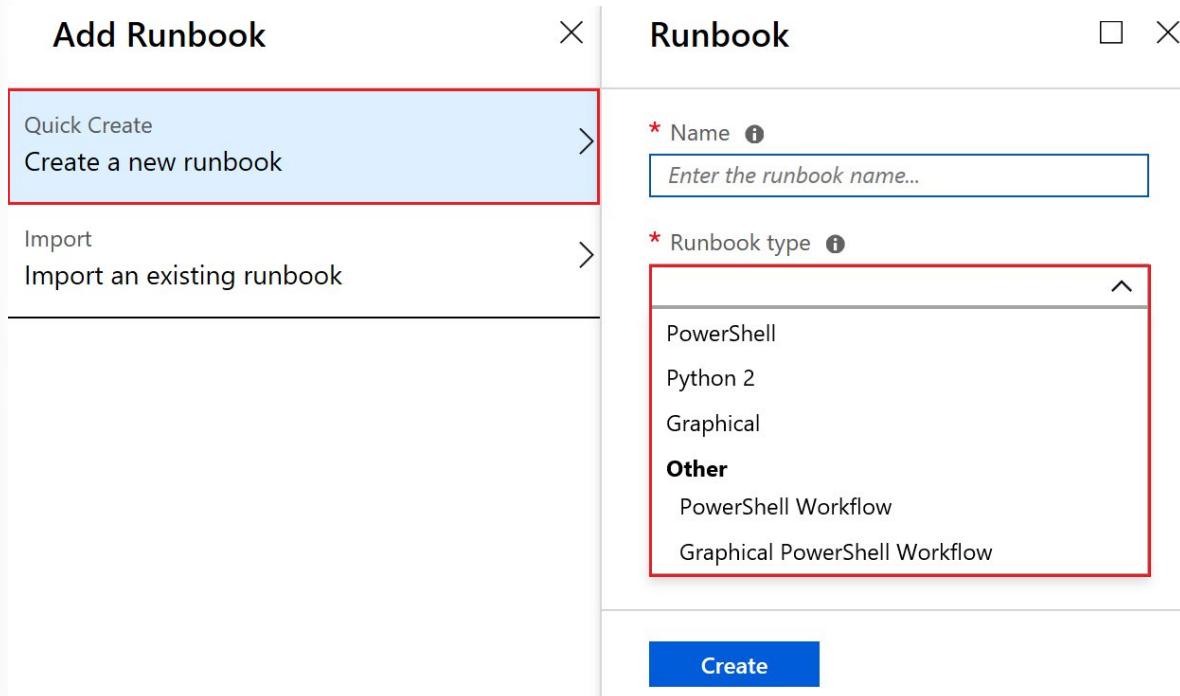


To use Azure Automation, you will need at least one Azure Automation account. However, as a best practice you should create multiple automation accounts to segregate and limit the scope of access and minimize any risk to your organization. For example, you might use one account for development, another for production, and another for your on-premises environment. You can have up to 30 Automation accounts.

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 Automation](#)¹⁶.
- 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 Automation](#)¹⁷ 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.

¹⁶ <https://docs.microsoft.com/en-us/azure/automation/automation-creating-importing-runbook>

¹⁷ <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.

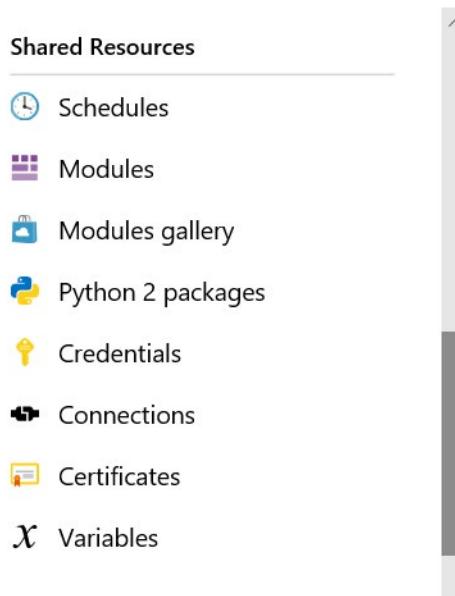
For more information on the different types of runbooks, visit [Azure Automation runbook types¹⁸](#).

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.

¹⁸ <https://azure.microsoft.com/en-us/documentation/articles/automation-runbook-types>



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.

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 professionals¹⁹](#) 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 module²⁰](#).

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

¹⁹ [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)

²⁰ <https://docs.microsoft.com/en-us/powershell/azure/new-azurmps-module-az?view=azps-2.7.0>

description, ratings, and questions and answers. For more information, refer to **Script resources for IT professionals²¹**.

The screenshot shows the Azure Runbook gallery page for the 'Start Azure V2 VMs' runbook. At the top, there's a title bar with a 'Start Azure V2 VMs' icon, a 'View Source' link, and close/minimize buttons. Below the title is a large red-bordered 'Import' button. The main content area contains a brief description: 'This Graphical PowerShell runbook connects to Azure using an Automation Run As account and starts all V2 VMs in an Azure subscription or in a resource group or a single named V2 VM. You can attach a recurring schedule to this runbook to run it at a specific time. The asso'. It also lists the creator as 'Azure Automation Product Team - Microsoft', tags like 'Azure Virtual Machines, Start VM, GraphicalPS', and metrics: 'Ratings: 4.43 of 5', '75,721 downloads', and 'Last updated: 10/22/2016'. A 'View Source Project' link is also red-bordered. Below this is a graphical flowchart:

```

graph TD
    A[Get Run As Connection] --> B[Connect to Azure]
    B --> C[Get single VM]
    B --> D[Get all VMs in RG]
    B --> E[Get all VMs in Sub]
    C -.-> D
    C -.-> E
    D -.-> E
  
```

The flowchart starts with 'Get Run As Connection', which leads to 'Connect to Azure'. From 'Connect to Azure', three parallel paths branch out: 'Get single VM', 'Get all VMs in RG', and 'Get all VMs in Sub'. There are dashed arrows indicating dependencies between the parallel steps: 'Get single VM' has a dashed arrow pointing to both 'Get all VMs in RG' and 'Get all VMs in Sub'; 'Get all VMs in RG' and 'Get all VMs in Sub' also have dashed arrows pointing to each other.

ATTENTION 1:1 Each runbook is licensed to you under a license agreement by its owner, not Microsoft. Microsoft is not responsible for runbooks provided & licensed by the community members and does not screen for security, compatibility or performance. The runbooks are not supported under any Microsoft support program or service. The runbooks are provided AS IS without warranty of any kind.

- ✓ 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.

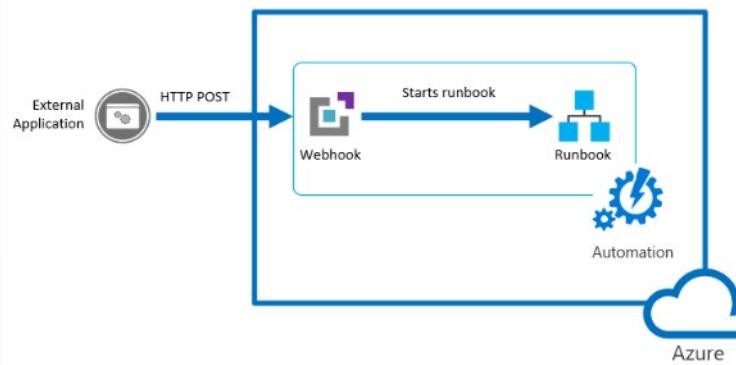
Webhooks

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 webhook²²](#).)

²¹ <https://gallery.technet.microsoft.com/scriptcenter>

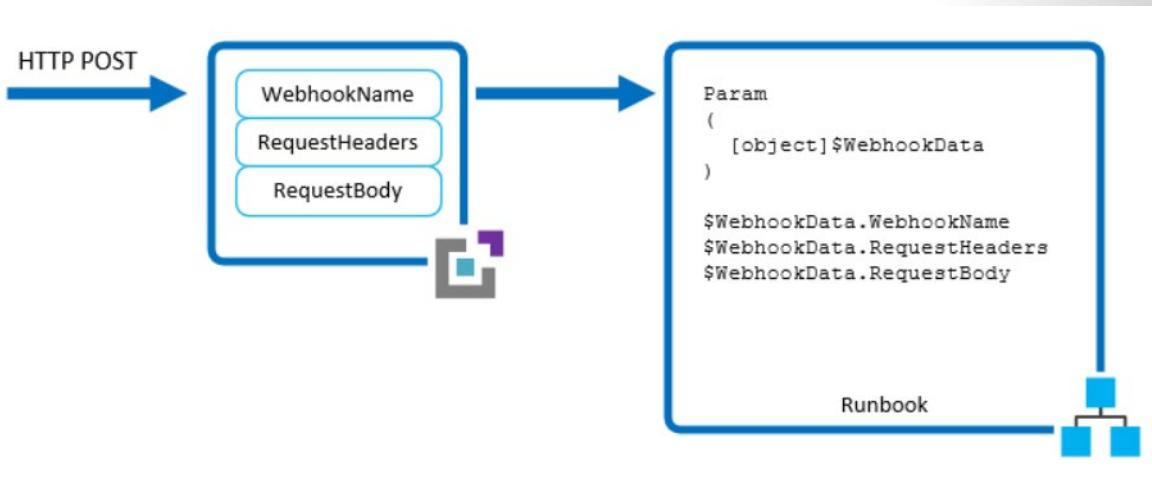
²² <https://docs.microsoft.com/en-us/azure/automation/automation-webhooks>



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 can 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.

The screenshot shows two side-by-side dialog boxes. The left box is titled 'Add Webhook' and has a sub-section 'Start a runbook via a simple HTTP POST to a URL'. It contains two options: 'Webhook' and 'Create new webhook'. Below these are 'Parameters and run settings' and 'Modify run settings (Default: Azure)'. At the bottom is a 'Create' button. The right box is titled 'Create a new webhook' and contains a warning message: 'For security, after creating a webhook its URL can't be viewed. Make sure to copy it before pressing "OK", and to store it securely. Learn more'. It has fields for 'Name' (with placeholder 'Enter the webhook name...'), 'Enabled' (with radio buttons for 'Yes' and 'No', where 'Yes' is selected), 'Expires' (set to 2020-01-11 at 6:00:23 AM), and a 'URL' field containing 'https://s2events.azure-automation.net/'. A blue 'OK' button is at the bottom of this box.

- 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 webhook²³](#) 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 collaborate with your team more easily, 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.

²³ <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

↻ Start Sync Save Delete ✖ Discard

* Source control name
gituh repo ✓

* Source control type
GitHub ▼

Authorization successful.
Authenticate

* Repository
PartsUnlimited ▼

* Branch
master ▼

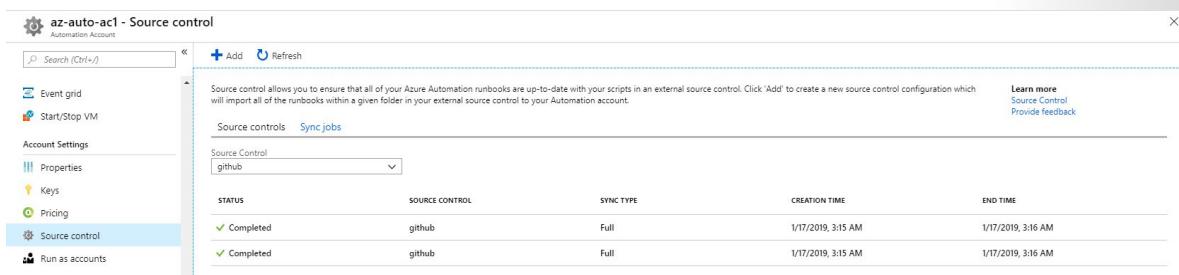
* Folder path
\Labfiles\Devops200.2x-InfrastructureasCode\Mod01 ✓

Auto Sync On Off

Publish Runbook 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 'az-auto-ac1 - Source control' page in the Azure portal. On the left, there's a sidebar with options like Event grid, Start/Stop VM, Account Settings, Keys, Pricing, Source control (which is selected and highlighted in blue), and Run as accounts. The main content area has a search bar at the top. Below it, there's a section about source control and a table listing sync jobs. The table has columns for STATUS, SOURCE CONTROL, SYNC TYPE, CREATION TIME, and END TIME. It shows two entries, both of which are 'Completed' and use 'github' as the source control.

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

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 4²⁴** 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.

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.

²⁴ [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))

- 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 runbook²⁵](#).

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

{
    ...
}
```

²⁵ <https://azure.microsoft.com/en-us/documentation/articles/automation-first-runbook-textual/>

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 today²⁶](#) webpage.

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.

²⁶ https://azure.microsoft.com/en-us/free/?ref=microsoft.com&utm_source=microsoft.com&utm_medium=docs&utm_campaign=visualstudio

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"
}
```

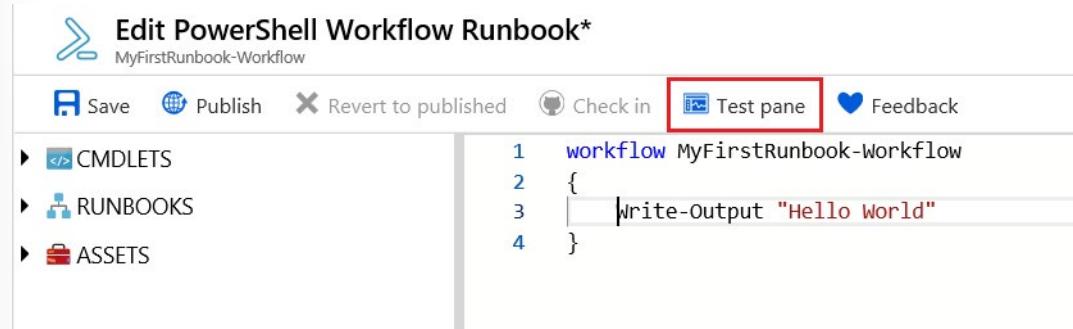
2. Save the runbook by selecting **Save**.



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.



2. Select **Start** to start the test. This should be the only enabled option.

The screenshot shows the Azure portal's Test pane for a runbook titled "Test". The runbook is identified as "MyFirstRunbook-Workflow". The top navigation bar includes buttons for "Start" (highlighted with a red border), "Stop", "Suspend", "Resume", "View last test", and "Refresh job streams". Below the navigation are sections for "Parameters" (No input parameters) and "Run Settings" (Run on Azure). A callout box provides information about using a hybrid runbook worker to increase test performance. The main pane displays a message: "Click 'Start' to begin the test run. Streams will display when the test completes." On the left, there is a section for "Activity-level tracing" with a note that it is available only for graphical runbooks, and a "Trace level" selector with options "None", "Basic", and "Detailed".

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 starts running. When the runbook job completes, its output displays. In your case, you should see Hello World.

- When the runbook job finishes, close the **Test pane**.

The screenshot shows the Azure portal's Test pane for the same runbook. The "Start" button is now highlighted with a blue border. The main pane displays the word "Completed" in green. Below it, the text "Hello World" is shown in white, indicating the runbook's output. The rest of the interface is identical to the first screenshot, including the "Parameters", "Run Settings", and "Activity-level tracing" sections.

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 sections for Overview, Activity log, Tags, and Diagnose and solve problems. The main content area displays the runbook's properties: Resource group (az-auto-rq), Account (az-auto-ac-1), Subscription (Pay-As-You-Go), Subscription ID (974e6e39-73eb-48b0-9226-dae31425c367), Runbook type (PowerShell Workflow Runbook), Location (West Europe), Last modified (1/11/2019 10:14 AM), and Tags (with a link to change). Below this is a "Recent Jobs" table with columns STATUS, CREATED, and LAST UPDATED, showing "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' run on 1/11/2019 at 10:16 AM. The job status is 'Completed'. The overview section displays metrics: 0 Input, 0 Errors, and 0 Warnings. It also includes a link to 'All Logs'.

Essentials	Value
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
Ran on	Azure
Source snapshot	View source snapshot

Overview	
Input	0
Output	Output
All Logs	
Errors	0
Warnings	0
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 [checkpoint²⁷](#) 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.

²⁷ <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 processing²⁸](#) 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
{
```

²⁸ <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."
}
```

Desired State Configuration (DSC)

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 involve 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**²⁹.
- Azure Policy. Use Azure Policy to enforce policies and compliance standards for Azure resources. For more information about Azure Policy, go to: **Azure Policy**³⁰.

There are also other third-party solutions that you can integrate with Azure.

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.

²⁹ <https://docs.microsoft.com/en-us/powershell/scripting/dsc/overview/overview?view=powershell-7>

³⁰ <https://azure.microsoft.com/en-us/services/azure-policy/>

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 Overview³¹](#).

DSC components

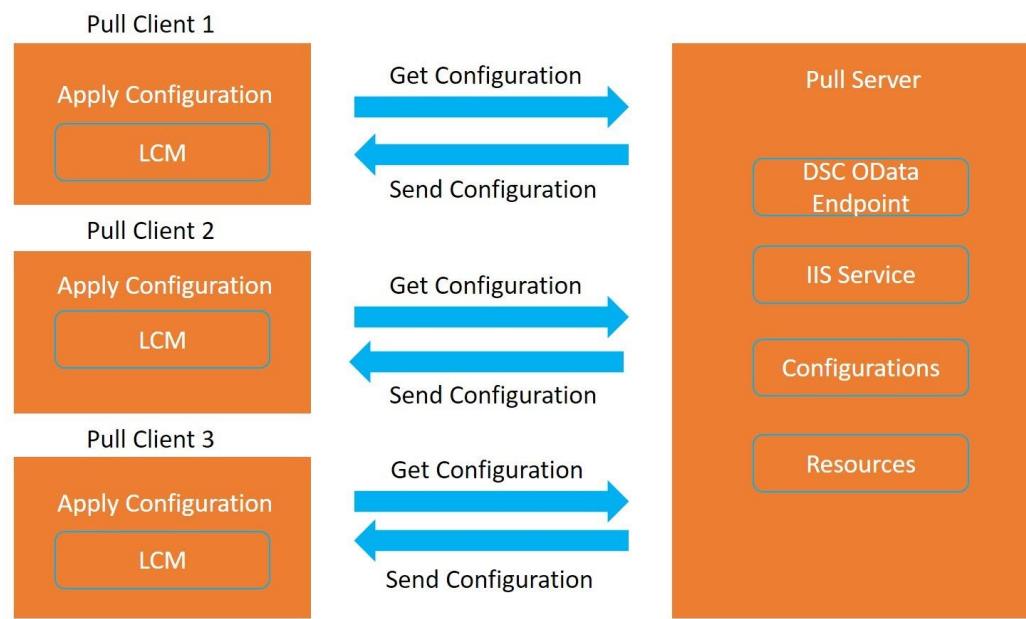
DSC consists of three primary components:

- 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 to 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.

³¹ <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 Configuration³²](#). 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:

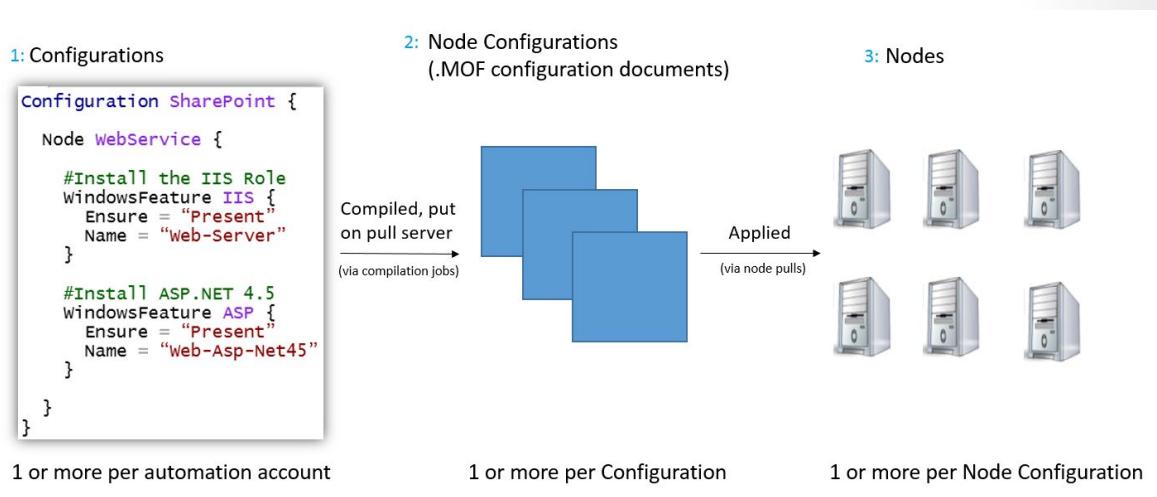
- Built-in pull server. Provides a DSC pull server like the Windows Feature DSC-service so that target nodes automatically receive configurations, conform to the desired state, and report back on their compliance. The built-in pull server in Azure Automation eliminates the need to set up and maintain 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.

³² <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)** file³³ 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**³⁴ 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
        }
    }
}
```

³³ [https://msdn.microsoft.com/en-us/library/aa823192\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/aa823192(v=vs.85).aspx)

³⁴ <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**³⁵.

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**³⁶.

³⁵ <https://docs.microsoft.com/en-us/powershell/scripting/dsc/resources/resources?view=powershell-7>

³⁶ <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:

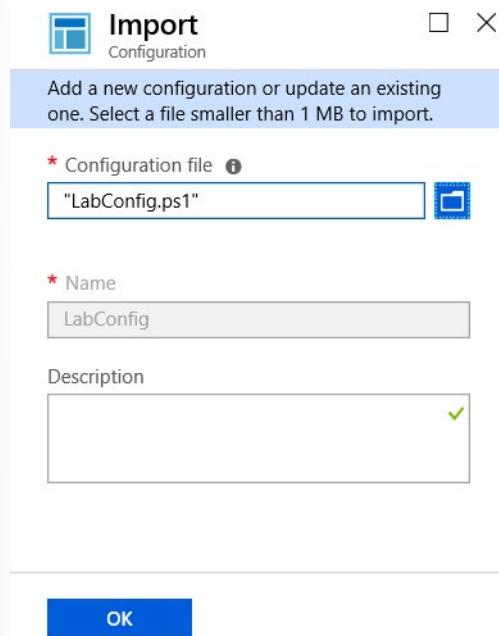
```
configuration LabConfig
{
    Node WebServer
    {
        WindowsFeature IIS
        {
            Ensure = 'Present'
            Name = 'Web-Server'
            IncludeAllSubFeature = $true
        }
    }
}
```

2. In Azure Automation, account under **Configuration Management > State configuration (DSC)**, select the **Configurations** tab, and then select **+Add**.

The screenshot shows the Azure Automation blade for an account named 'az-auto-ac-1'. The 'State configuration (DSC)' section is selected. The 'Configurations' tab is active and highlighted with a red box. The 'Nodes' tab is also highlighted with a red box. The 'Compose configuration' button is visible above the tabs. The main pane displays a table with columns: CONFIGURATION, COMPILED CONFIGURATION COUNT, and LAST MODIFIED. A message 'No data' is shown.

CONFIGURATION	COMPILED CONFIGURATION COUNT	LAST MODIFIED
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 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" which contains 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 PowerShell³⁷](#).

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/AWS³⁸](#) 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.)

³⁷ <https://azure.microsoft.com/en-us/documentation/articles/automation-dsc-compile/#compiling-a-dsc-configuration-with-windows-powershell>

³⁸ <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 under the 'az-auto-ac-1' subscription. It lists two VMs: 'SimpleWinVM' and 'SimpleWinVM'. Both are in the 'Pay-As-You-Go' subscription, belong to the 'simplewinvm' resource group, and are located in 'West Europe'. On the right is a detailed view of the 'SimpleWinVM' virtual machine. It shows the 'SimpleWinVM' icon, its status as 'Not connected', and its configuration: 'POWER STATE' is 'VM running', 'OS' is 'Windows', and 'STATUS' is 'Not connected'. There are 'Connect', 'Refresh', and 'Learn more' buttons at the top.

5. In the resultant Registration pane, configure the following settings, and then select **OK**.

The screenshot shows the 'Registration' pane. At the top, there's a note about a required 'Registration key' with tabs for 'Primary key' (which is selected) and 'Secondary key'. Below that, the 'Node configuration name' is set to 'LabConfig.WebServer'. Under 'Refresh Frequency', the value is '30'. Under 'Configuration Mode Frequency', the value is '15'. Under 'Configuration Mode', the dropdown is set to 'ApplyAndMonitor'. There are two checkboxes: 'Allow Module Override' (unchecked) and 'Reboot Node if Needed' (unchecked). Finally, under 'Action after Reboot', the dropdown is set to 'ContinueConfiguration'.

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.

Property	Description
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 of the Azure Automation DSC configuration. At the top, there are buttons for 'Add', 'Refresh', 'Reset filters', and 'Enable Log Search'. Below the tabs, a summary chart shows the following status counts:

Configuration status	Count
Failed	0
Not compliant	0
Unresponsive	0
Pending	0
In progress	0
Compliant	1

Below the summary, there are search and filter fields for 'Nodes', 'Status', 'Node configuration', and 'VM DSC extension version'. A table at the bottom lists the node details:

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 Configuration³⁹](#)
- [Configuring the Local Configuration Manager⁴⁰](#)

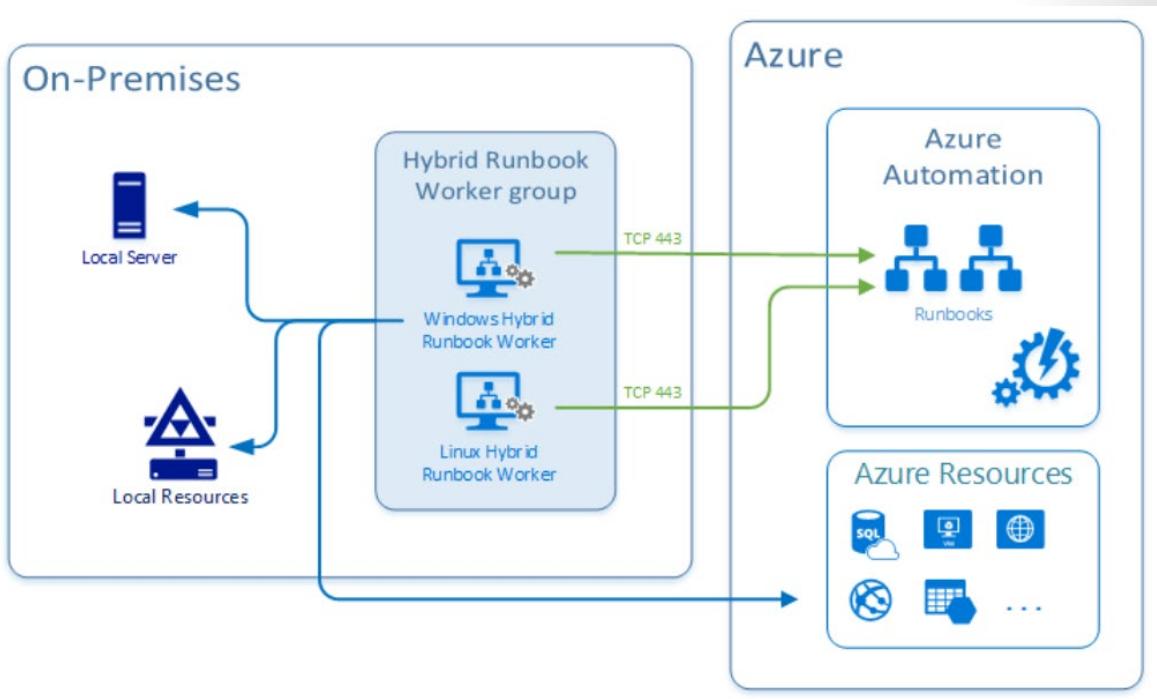
Hybrid management

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:

³⁹ <https://docs.microsoft.com/en-us/azure/automation/automation-dsc-onboarding>

⁴⁰ <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 Configuration⁴¹**.

For more information on installing and removing Hybrid Runbook Workers and groups, see:

- **Automate resources in your datacenter or cloud by using Hybrid Runbook Worker⁴²**
- **Hybrid Management in Azure Automation⁴³**

41 <https://docs.microsoft.com/en-us/azure/automation/automation-dsc-onboarding>

42 <https://docs.microsoft.com/en-us/azure/automation/automation-hybrid-runbook-worker#installing-hybrid-runbook-worker>

43 <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, 16.04 LTS, 18.04 (x86/x64)

More specific and up to date details are available at: **Get started with Desired State Configuration (DSC) for Linux⁴⁴**

⁴⁴ <https://docs.microsoft.com/en-us/powershell/scripting/dsc/getting-started/lxngettingstarted?view=powershell-7.1>

Lab

Deployments using Azure Resource Manager templates

Lab overview

In this lab, you will create an Azure Resource manager template and modularize it by using a linked template. You will then modify the main deployment template to call the linked template and updated dependencies, and finally deploy the templates to Azure.

Objectives

After you complete this lab, you will be able to:

- Create Resource Manager template
- Create a Linked template for storage resources
- Upload Linked Template to Azure Blob Storage and generate SAS token
- Modify the main template to call Linked template
- Modify main template to update dependencies
- Deploy resources to Azure using linked templates

Lab duration

- Estimated time: **60 minutes**

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- **AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions⁴⁵**

⁴⁵ <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/>

Module review and takeaways

Module review questions

Review Question 1

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

Review Question 2

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-oriented
- 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

Review Question 4

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

Review Question 5

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

Answers

Review Question 1

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.

Review Question 2

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-oriented
- 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-oriented 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.

Review Question 4

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.

Review Question 5

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 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.

Module 14 Using 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

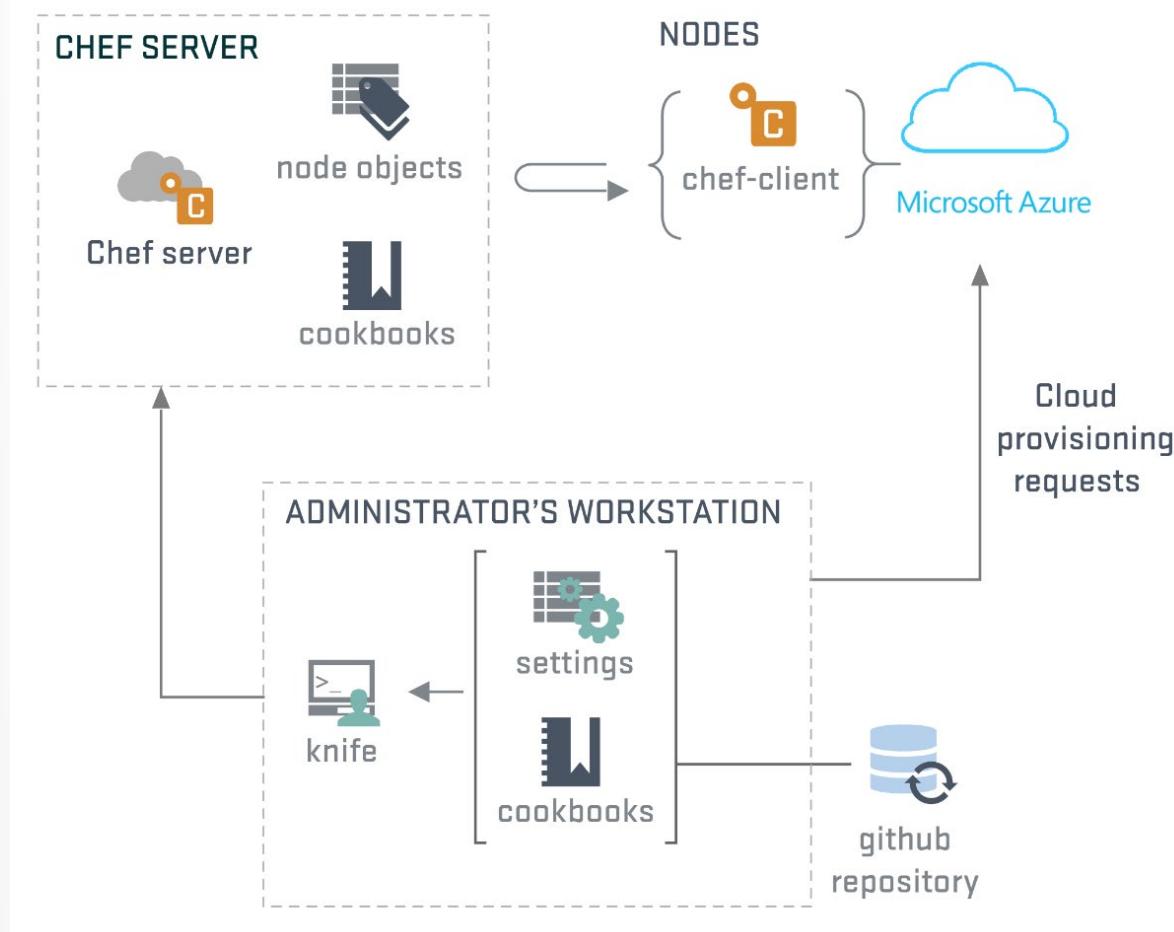
Chef

What is Chef?

Chef Infra 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 Infra 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 Infra.

The following diagram is of the high-level Chef Infra architecture:



Chef Infra components

Chef Infra 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 Infra also uses concepts called *cookbooks* and *recipes*. Chef Infra cookbooks and recipes are essentially the policies that you define and apply to your servers.

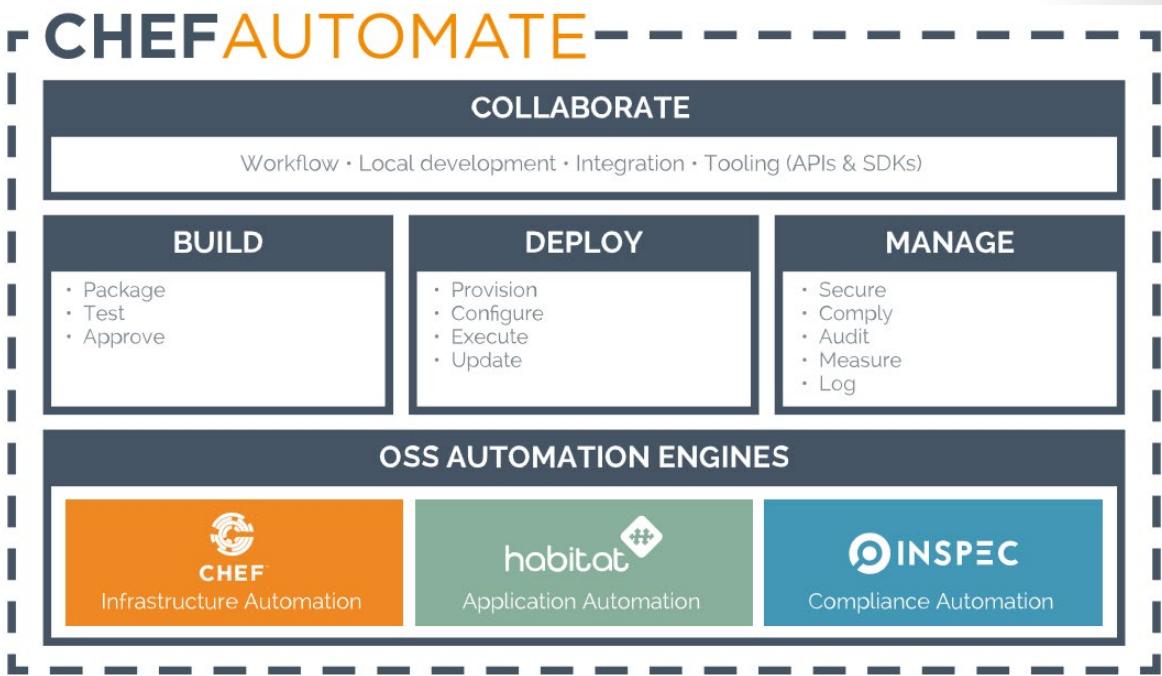
Chef Automate

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?

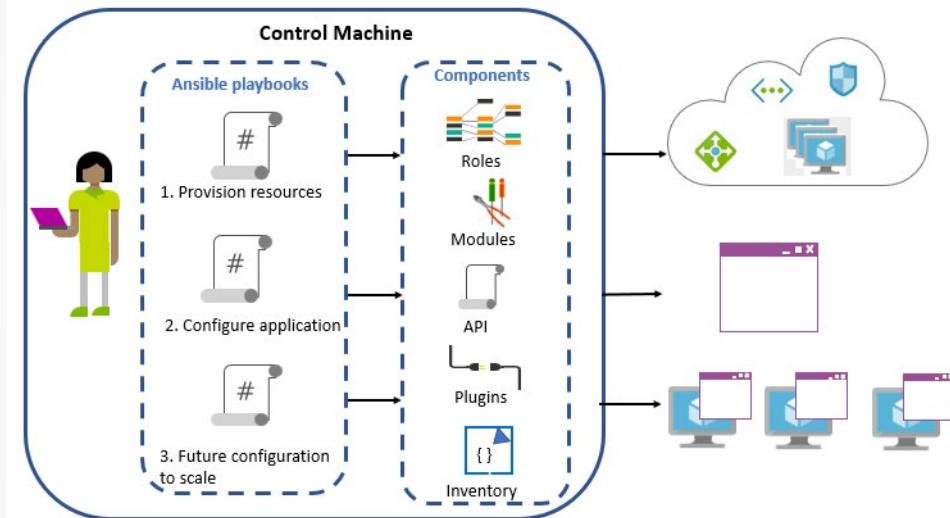
Ansible is an open-source platform by Red Hat that automates cloud provisioning, configuration management, and application deployments. Using Ansible, you can provision VMs, containers, and your entire cloud infrastructure. In addition to provisioning and configuring applications and their environments, 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 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 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 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 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 by 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
```

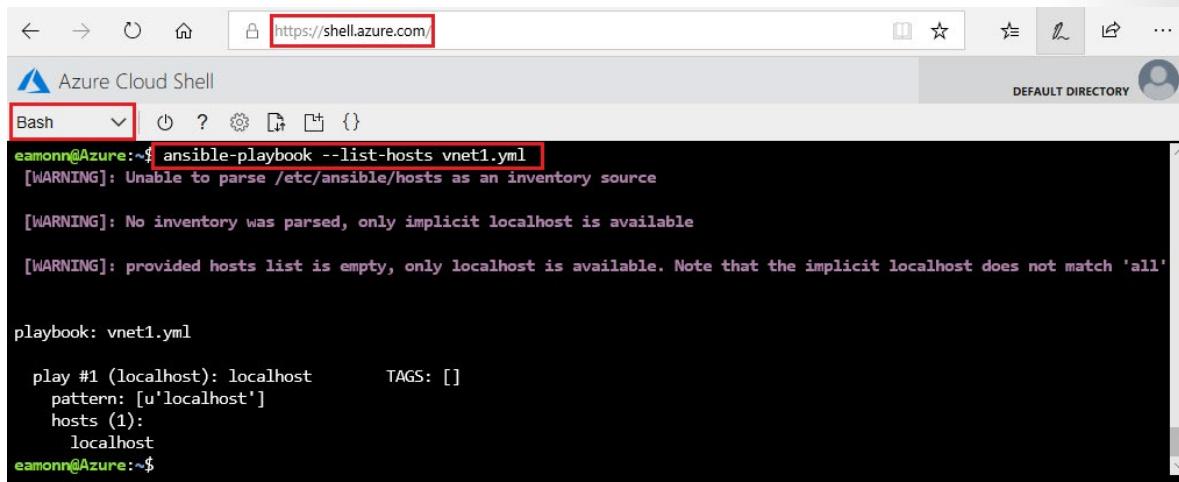
✓ Note: Ansible Playbook samples for Azure are available on GitHub on the [Ansible Playbook Samples for Azure](#)¹² page.

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 logo, a dropdown menu set to 'Bash', and a 'DEFAULT DIRECTORY' indicator. The terminal window shows the command `ansible-playbook --list-hosts vnet1.yml` being run. The output indicates that the command failed due to parsing errors and a missing inventory source. The terminal window has a dark background with white text and a red border around the command line.

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', 'azure', 'playbook.yml', etc. The terminal content is a YAML file named 'vnet1.yml' containing Ansible code to create a resource group and a virtual network. The code includes variables for resource group name ('ansible_rg5'), location ('westus'), and subnet details ('myVnet', '10.0.0.0/16', 'mySubnet', '10.0.1.0/24'). The terminal shows the command '#! /bin/bash' being typed.

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 like 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.

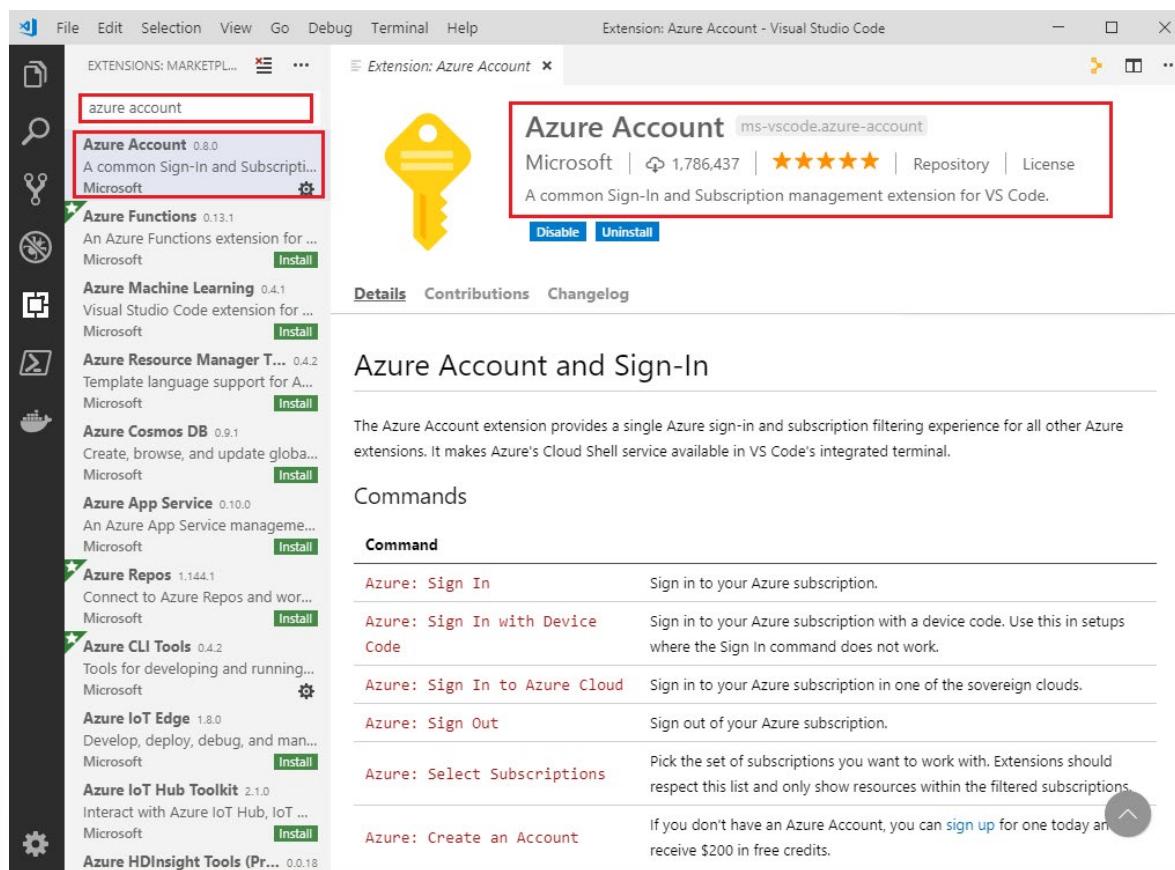
Demonstration: Run Ansible in Visual Studio Code

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

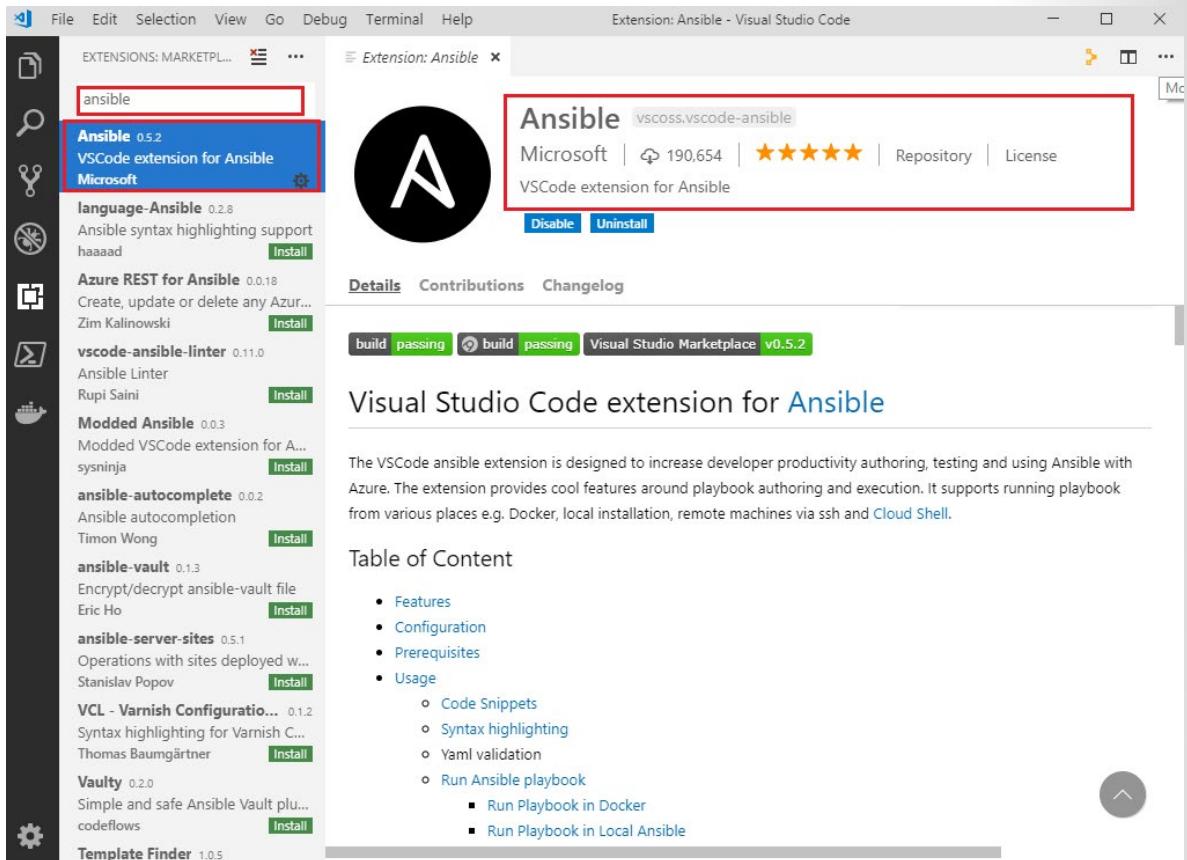
Complete the following steps to create network resources in Azure using Visual Studio Code:

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**.



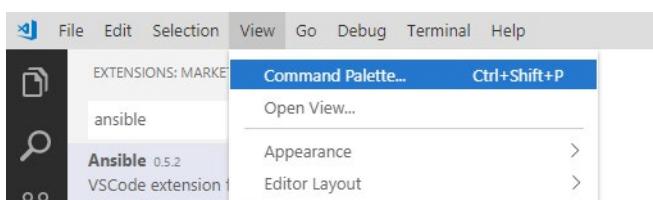
4. Search for and install the extension **Ansible**.

¹⁴ <https://code.visualstudio.com/>



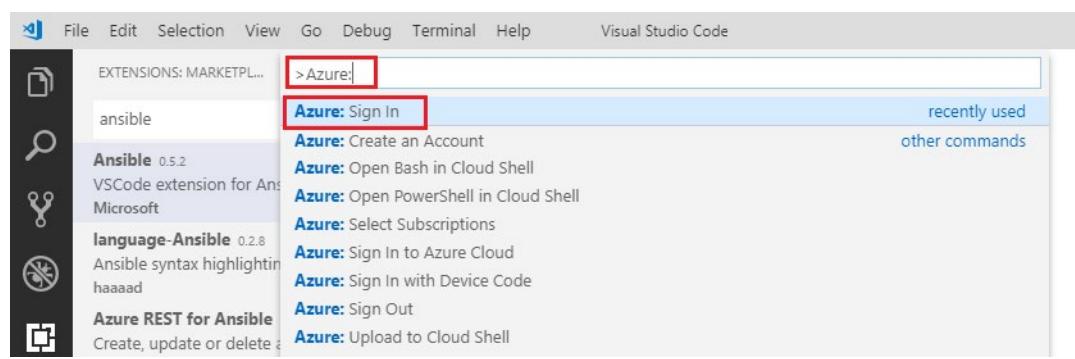
You can also view details of this extension on the Visual Studio Marketplace [Ansible](#)¹⁵ page.

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**.

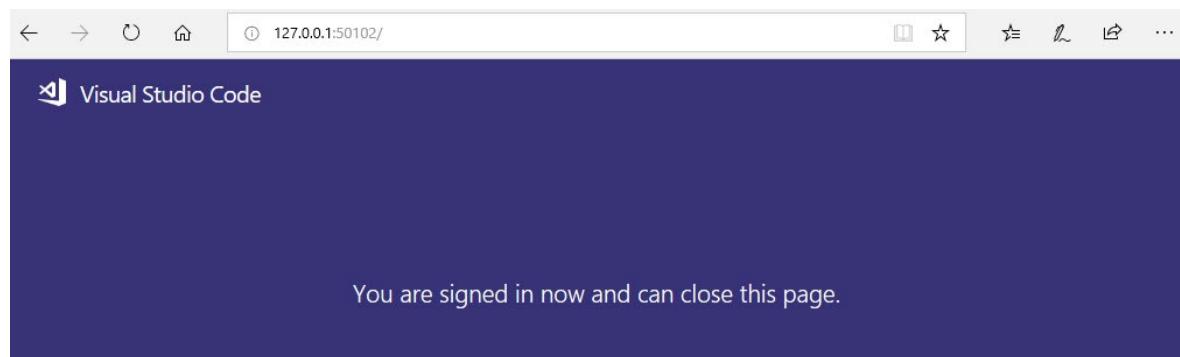


6. In the Command Palette, Type **Azure:**, select **Azure:Sign in**.

¹⁵ https://marketplace.visualstudio.com/items?itemName=vsco.vscos.vscod-ansible&ocid=AID754288&wt.mc_id=CFID0352



- 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.



- Verify that your Azure account now displays at the bottom of the Visual Studio Code window.
- 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
```

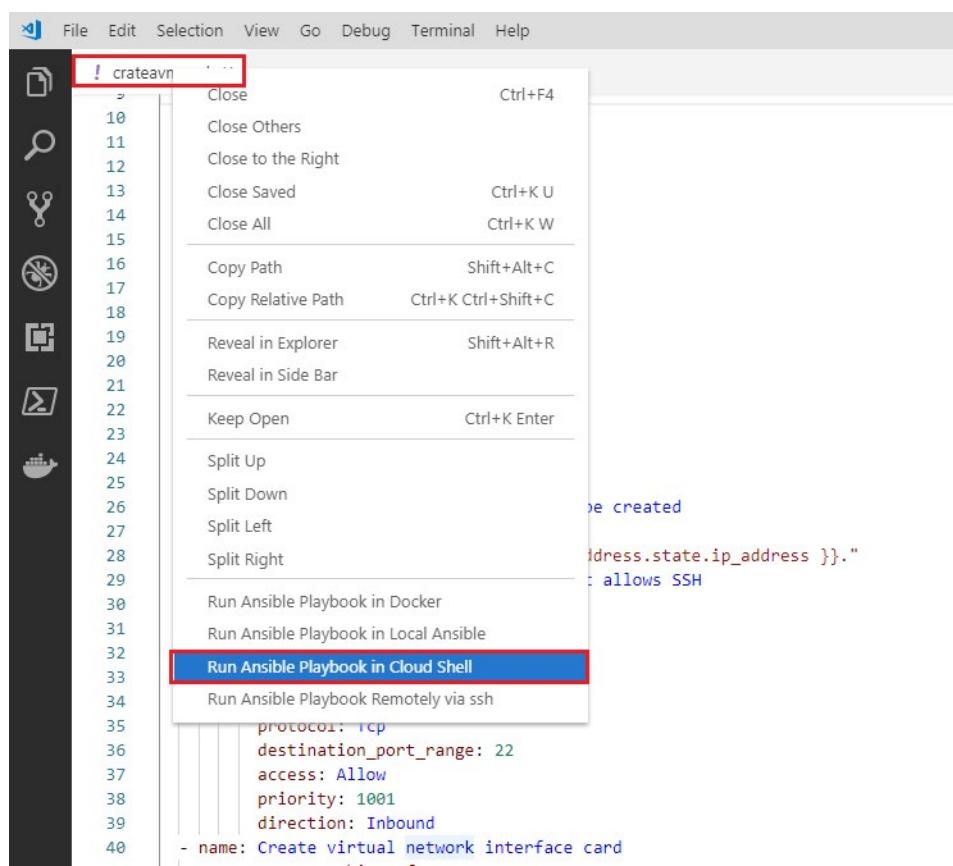
```
        allocation_method: Static
        name: myPublicIP
    register: output_ip_address
- name: Dump public IP for VM which will be created
  debug:
    msg: "The public IP is {{ output_ip_address.state.ip_address }}."
- name: Create Network Security Group that allows SSH
  azure_rm_securitygroup:
    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**.

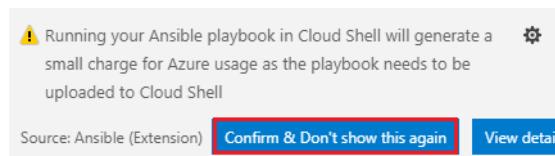
11. Right-click on the file name in the tab at the top of Visual Studio Code, and review the available options available to run the Ansible playbook:

- Run Ansible Playbook in Docker
- Run Ansible Playbook in Local Ansible
- Run Ansible Playbook Cloud Shell
- Run Ansible Playbook Remotely via ssh

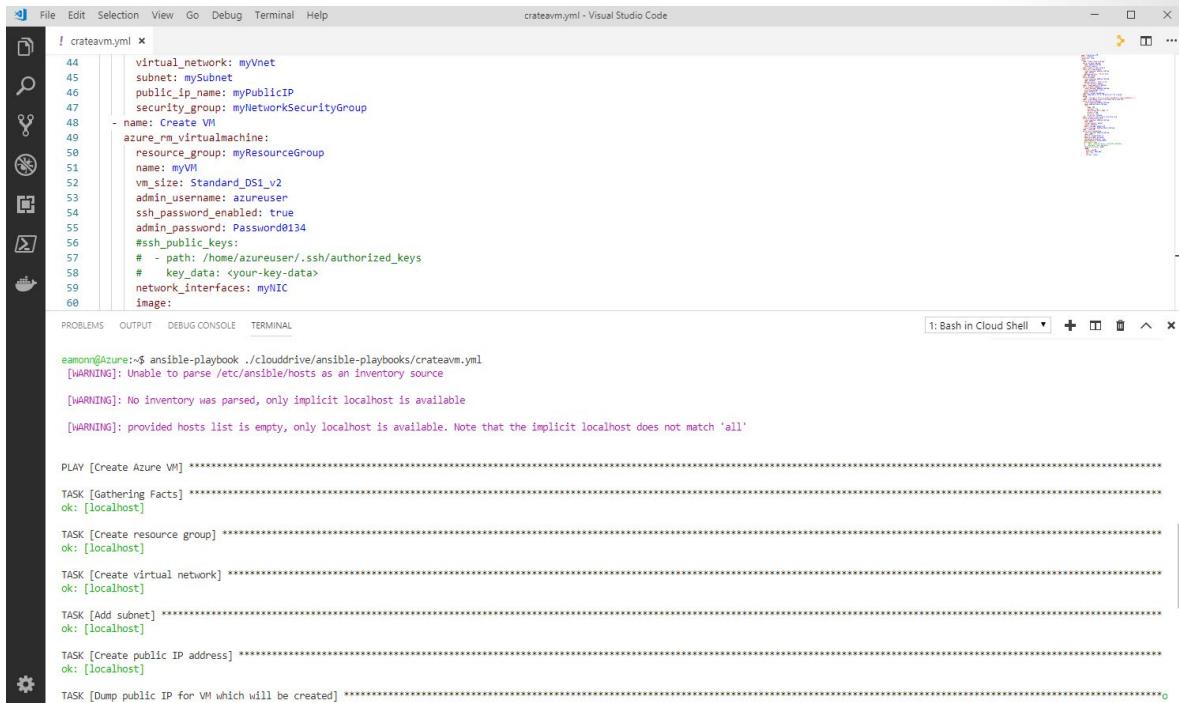
12. Select the third option, **Run Ansible Playbook Cloud Shell**.



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
cratevm.yml x
1 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:
61
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]

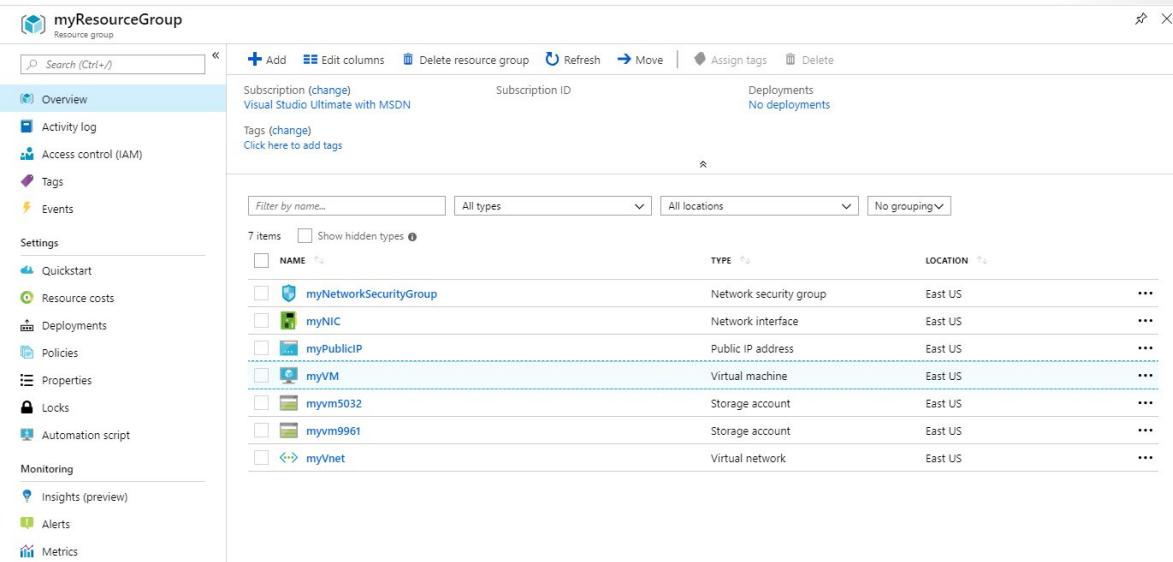
TASK [Add subnet] ****
ok: [localhost]

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 username 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: ~
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
azureuser@Terraform:~$
```

Authenticating Terraform with Azure

Terraform supports several 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}"
    }

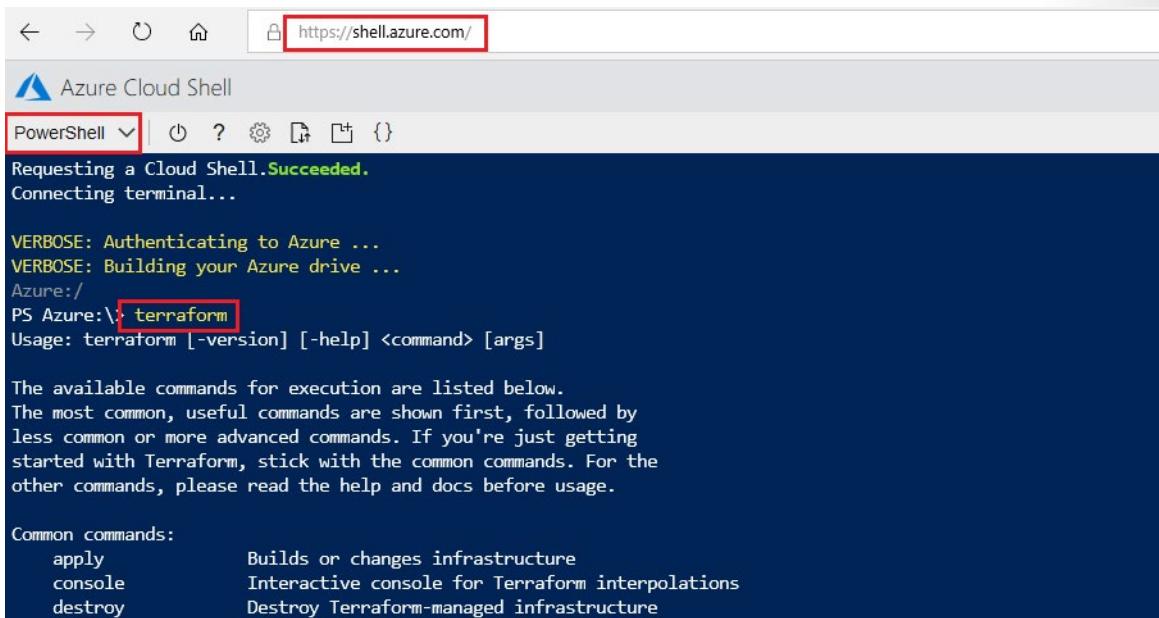
    tags {
```

```
        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.



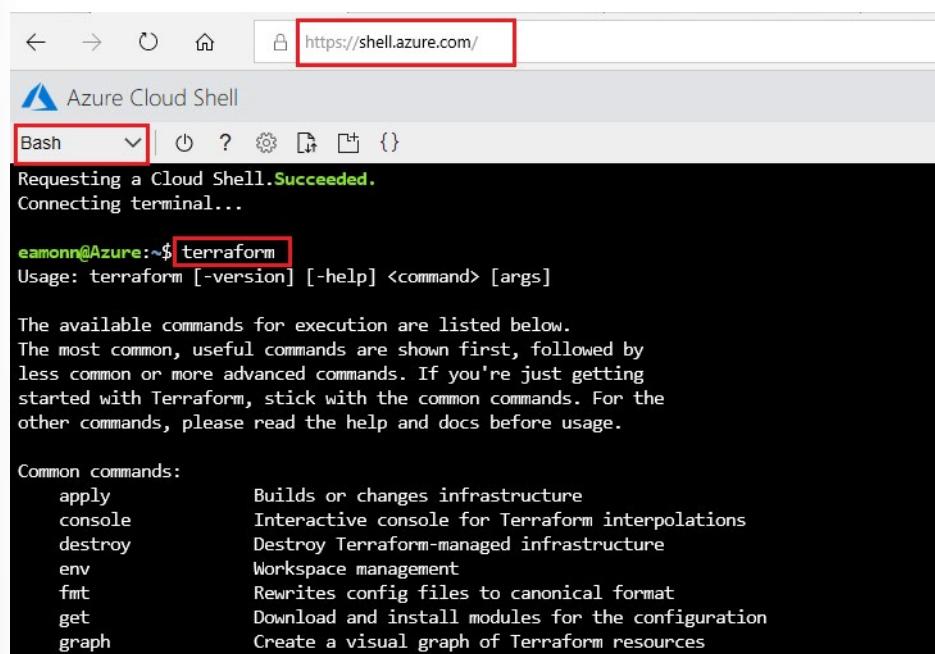
A screenshot of the Azure Cloud Shell interface. The browser address bar shows <https://shell.azure.com/>. The shell interface is labeled "Azure Cloud Shell" and "PowerShell". A red box highlights the "PowerShell" dropdown menu. The terminal window shows the following output:

```
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 address bar at the top displays the URL <https://shell.azure.com/>. Below the address bar, the title "Azure Cloud Shell" is visible, followed by a toolbar with icons for back, forward, refresh, and other shell functions. A dropdown menu labeled "Bash" is open, with a red box highlighting it. The main terminal window shows the output of a Terraform command. It starts with "Requesting a Cloud Shell. Succeeded." followed by "Connecting terminal...". The user then runs the command `terraform`, which outputs usage information: "Usage: terraform [-version] [-help] <command> [args]". It then lists common commands with their descriptions:

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
subscriptionterraform.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"
}
```

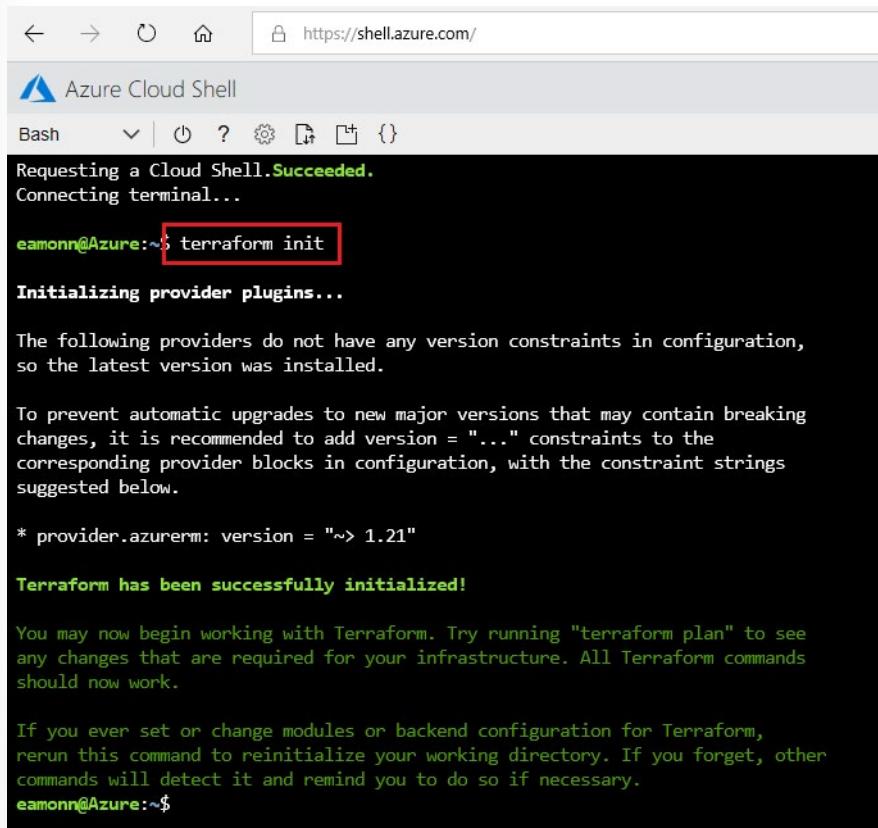
7. Exit **insert** mode by selecting the **Esc** key.
8. Save the file and exit the **vi** editor by entering the following command:

```
:wq
```

9. Use the following command to initialize Terraform:

```
terraform init
```

You should receive a message saying Terraform was successfully initiated.



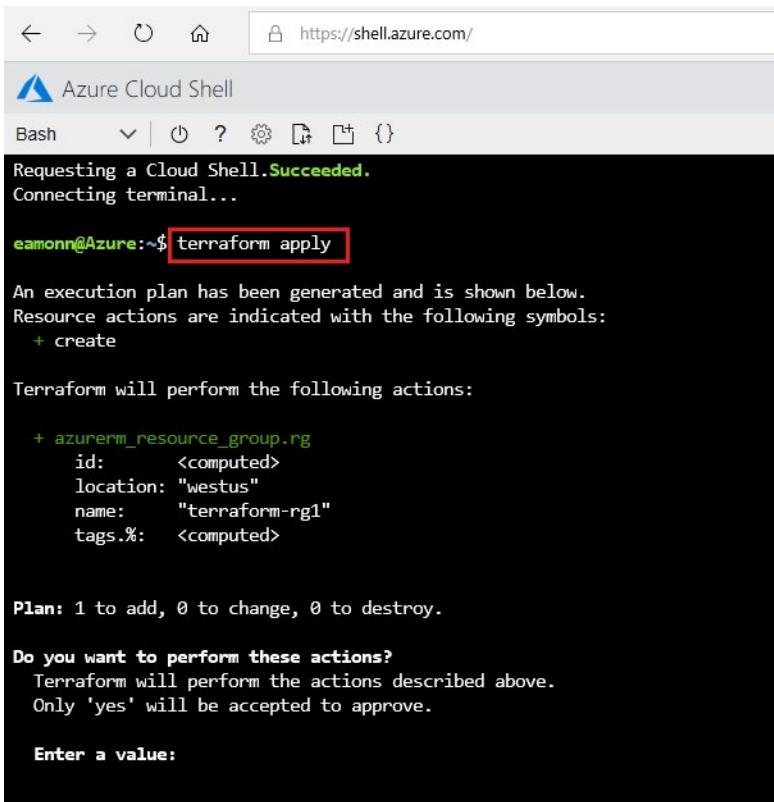
The screenshot shows a terminal window in the Azure Cloud Shell. The URL bar at the top shows <https://shell.azure.com/>. The terminal window has a title bar with the Azure Cloud Shell logo and the word 'Bash'. Below the title bar are several icons. The main area of the terminal shows the output of the 'terraform init' command. The output includes:

- A message indicating the Cloud Shell is requesting a connection: "Requesting a Cloud Shell. **Succeeded.**"
- A message indicating the terminal is connecting: "Connecting terminal..."
- The command being run: "eamonn@Azure:~\$ terraform init"
- A message indicating provider plugins are being initialized: "Initializing provider plugins..."
- A note about provider version constraints: "The following providers do not have any version constraints in configuration, so the latest version was installed."
- A note about preventing automatic upgrades: "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."
- A specific constraint suggestion: "* provider_azurerm: version = \"~> 1.21\""
- A confirmation message: "**Terraform has been successfully initialized!**"
- A note about working with Terraform: "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."
- A final note: "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."
- The command prompt again: "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. At the top, it says "Requesting a Cloud Shell. Succeeded. Connecting terminal...". Below that, a command line window shows the user "eamonn@Azure:~\$ terraform apply". A red box highlights this command. The output shows an execution plan: "An execution plan has been generated and is shown below. Resource actions are indicated with the following symbols: + create". It lists a single resource: "+ azurerm_resource_group.rg" with details: id: <computed>, location: "westus", name: "terraform-rg1", tags.%: <computed>. Below this, it says "Plan: 1 to add, 0 to change, 0 to destroy." and asks "Do you want to perform these actions?". The message continues: "Terraform will perform the actions described above. Only 'yes' will be accepted to approve." Finally, it prompts "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 a terminal window. The user types "Enter a value: yes" and presses Enter. The output shows the creation of a resource group: "azurerm_resource_group.rg: Creating...", followed by its configuration details: "location: "" => "westus", name: "" => "terraform-rg1", tags.%: "" => "<computed>". It then says "azurerm_resource_group.rg: Creation complete after 2s (ID: /subscriptions/subscriptionId/resourceGroups/terraform-rg1)". Below this, it says "Apply complete! Resources: 1 added, 0 changed, 0 destroyed." and ends with "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**.

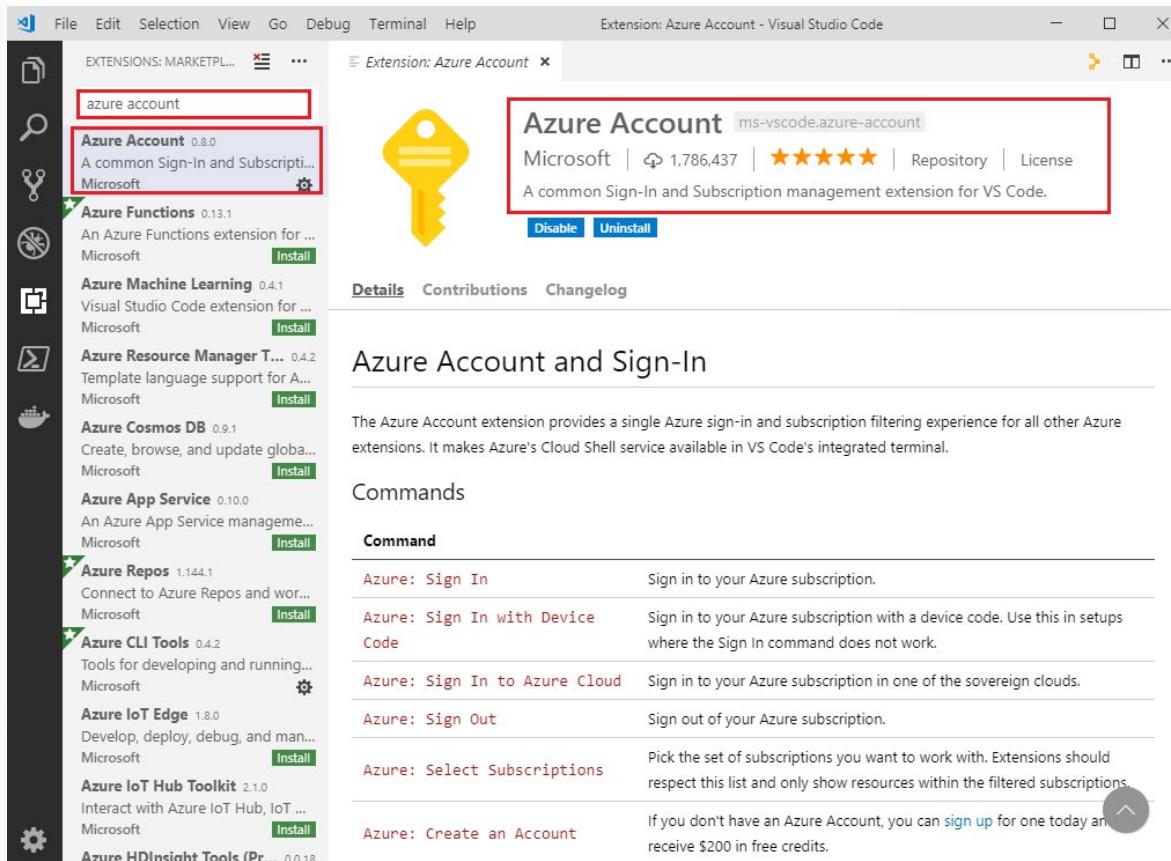
In this walkthrough you will create a VM in Visual Studio Code using Terraform

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

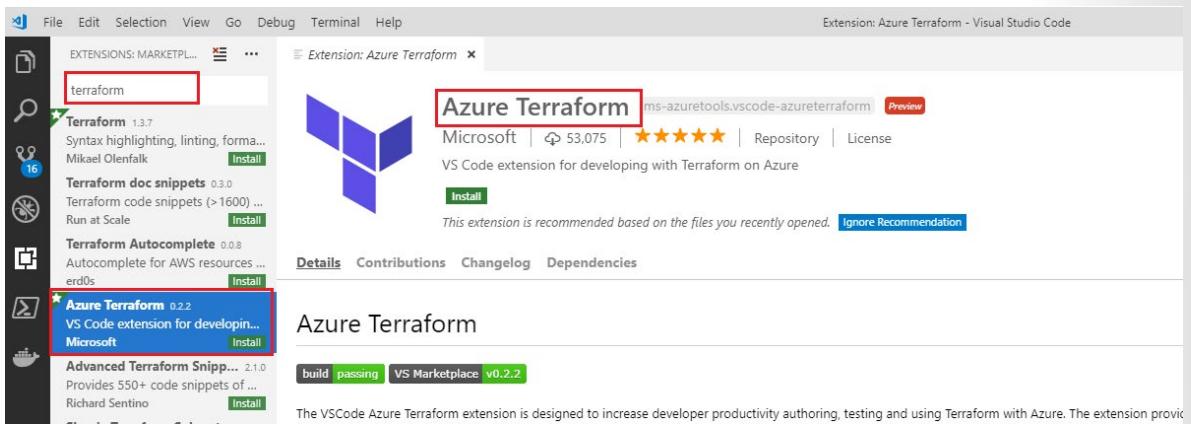
- Launch the Visual Studio Code editor.
- The two Visual Studio Code extensions *Azure Account* and *Azure Terraform* must be installed. To install the first extension, from inside Visual Studio Code, select **File > Preferences > Extensions**.
- Search for and install the extension **Azure Account**.



- 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.

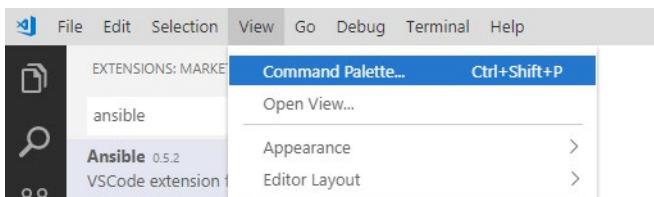
²⁰ <https://code.visualstudio.com/>

²¹ https://azure.microsoft.com/en-us/free/?ref=microsoft.com&utm_source=microsoft.com&utm_medium=docs&utm_campaign=visualstudio

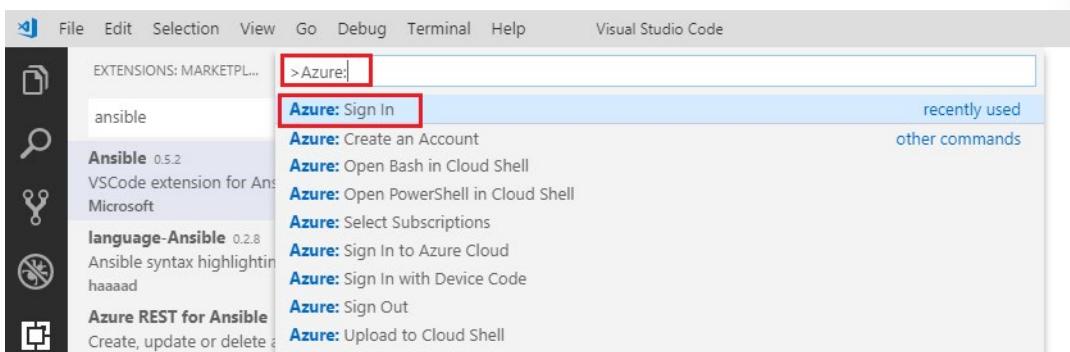


You can view more details of this extension at the Visual Studio Marketplace on the [Azure Terraform²²](#) page.

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**.

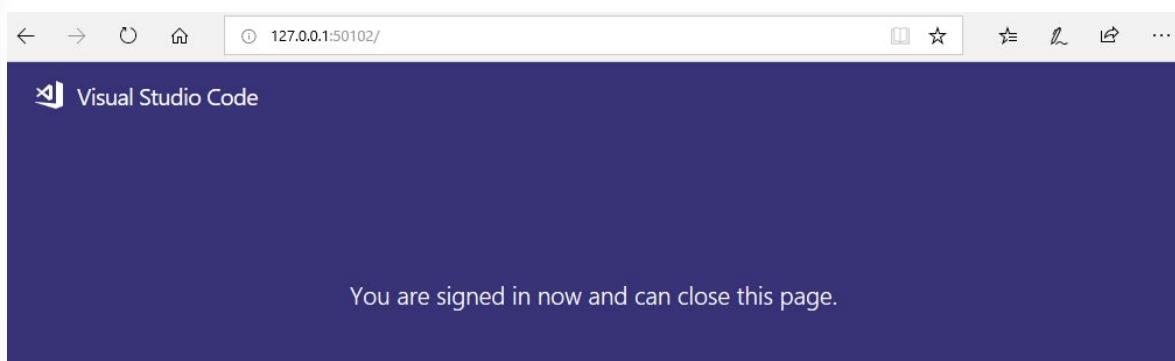


6. In the Command Palette search field, type **Azure:**, and from the results, select Azure: Sign In.



7. When a browser launches and prompts you to sign into 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}"
}

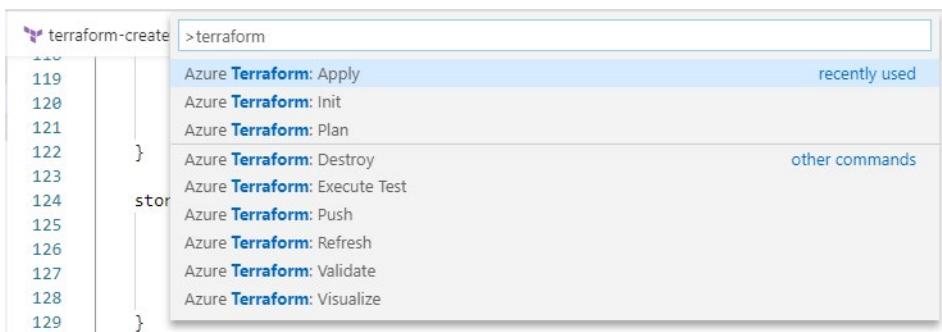
tags {
    environment = "Terraform Demo"
}
}

```

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**.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files like `ansiblelocalhostping.yml`, `azuredeploy.json`, `createavm.json`, `rg1.yml`, and `terraform-createvm.tf`.
- Editor:** Displays the Terraform configuration file `terraform-createvm.tf` with code snippets for creating a storage OS disk, specifying a storage image reference, defining an OS profile, and configuring Linux settings.
- Output:** Shows the results of the Terraform plan command, indicating 1 resource to add, 0 to change, and 0 to destroy.
- Terminal:** Shows the command `Enter a value: yes` being typed into the terminal.
- Bottom Status Bar:** Shows "1: Bash in Cloud Shell".

14. After the command completes successfully, review the list of resources created.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files like `ansiblelocalhostping.yml`, `azuredeploy.json`, `createavm.json`, `rg1.yml`, and `terraform-createvm.tf`.
- Output:** Displays the output of a Terraform apply command, showing the creation of various resources including a storage OS disk and an Azure VM.
- Terminal:** Shows the command `Apply complete! Resources: 1 added, 0 changed, 0 destroyed. eamonn@Azure:~/clouddrive/linked\$`.
- Bottom Status Bar:** Shows "1: Bash in Cloud Shell".

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 username 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 username 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

Ansible with Azure

Lab overview

In this lab we will deploy, configure, and manage Azure resources by using Ansible.

Ansible is declarative configuration management software. It relies on a description of the intended configuration applicable to managed computers in the form of playbooks. Ansible automatically applies that configuration and maintains it going forward, addressing any potential discrepancies. Playbooks are formatted by using YAML.

Unlike majority of other configuration management tools, such as Puppet or Chef, Ansible is agentless, which means that it does not require the installation of any software in the managed machines. Ansible uses SSH to manage Linux servers and Powershell Remoting to manage Windows servers and clients.

In order to interact with resources other than operating systems (such as, for example, Azure resources accessible via Azure Resource Manager), Ansible supports extensions called modules. Ansible is written in Python so, effectively, the modules are implemented as Python libraries. In order to manage Azure resources, Ansible relies on **GitHub-hosted modules²³**.

Ansible requires that the managed resources are specified in a designated host inventory. Ansible supports dynamic inventories for some systems, including Azure, so that the host inventory is dynamically generated at runtime.

The lab will consist of the following high-level steps:

- Installing and configuring Ansible on the Azure VM
- Downloading Ansible configuration and sample playbook files
- Creating and configuring a managed identity in Azure AD
- Configuring Azure AD credentials and SSH for use with Ansible
- Deploying an Azure VM by using an Ansible playbook
- Configuring an Azure VM by using an Ansible playbook

Objectives

After you complete this lab, you will be able to:

- Install and configure Ansible on Azure VM
- Download Ansible configuration and sample playbook files
- Create and configure Azure Active Directory managed identity
- Configure Azure AD credentials and SSH for use with Ansible
- Deploy an Azure VM by using an Ansible playbook
- Configure an Azure VM by using an Ansible playbook

²³ <https://github.com/ansible-collections/azure>

Lab duration

- Estimated time: **90 minutes**

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- [AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions²⁴](#)

Automating infrastructure deployments in the cloud with Terraform and Azure Pipelines

Lab overview

Terraform²⁵ is a tool for building, changing and versioning infrastructure. Terraform can manage existing and popular cloud service providers as well as custom in-house solutions.

Terraform configuration files describe 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 to execute.

In this lab, you will learn how to incorporate Terraform into Azure Pipeline for implementing Infrastructure as Code.

Objectives

After you complete this lab, you will be able to:

- Use Terraform to implement Infrastructure as Code
- Automate infrastructure deployments in Azure with Terraform and Azure Pipelines

Lab duration

- Estimated time: **60 minutes**

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- [AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions²⁶](#)

²⁴ <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/>

²⁵ <https://www.terraform.io/intro/index.html>

²⁶ <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/>

Module review and takeaways

Module review questions

Review Question 1

*Which of the following are main architectural components of Chef?
(choose all that apply)*

- Chef Server
- Chef Facts
- Chef Client
- Chef Workstation

Review Question 2

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

Review Question 3

*Which of the following are core components of the Puppet automation platform?
(choose all that apply)*

- Master
- Agent
- Facts
- Habitat

Review Question 4

Complete the following sentence.

The main elements of a Puppet Program (PP) Manifest file are class, resource and _____.

- Module
- Habitat
- InSpec
- Cookbooks

Review Question 5

*Which of the following platforms use agents to communicate with target machines?
(choose all that apply)*

- Puppet
- Chef
- Ansible

Review Question 6

True or false: The control machine in Ansible must have Python installed?

- True
- False

Review Question 7

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.

Review Question 8

True or false: Terraform ONLY supports configuration files with the file extension .tf.

- True
- False

Review Question 9

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

Review Question 1

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.

Review Question 2

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.

Review Question 3

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.

Review Question 4

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.

Review Question 5

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.

Review Question 6

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.

Review Question 7

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 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.

Review Question 8

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.

Review Question 9

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 15 Managing Containers using Docker

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
- Implement Docker multi-stage builds

Implementing a container build strategy

Why Containers?

Container

Unlike a VM which provides hardware virtualization, a container provides operating-system-level virtualization by abstracting the “user space”, not the entire operating system.

The operating system level architecture is being shared across containers. This is what makes containers so lightweight.

What other benefits do containers offer?

Containers are **portable**. A container will run wherever Docker is supported.

Containers allow you to have a **consistent** development environment. For example, a SQL Server 2019 CU2 container that one developer is working with, will be identical to the one that another developer is working with.

Containers can be extremely **lightweight**. 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.

Containers can be very **efficient**: fast to deploy, fast to boot, fast to patch, fast to update.

Structure 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?

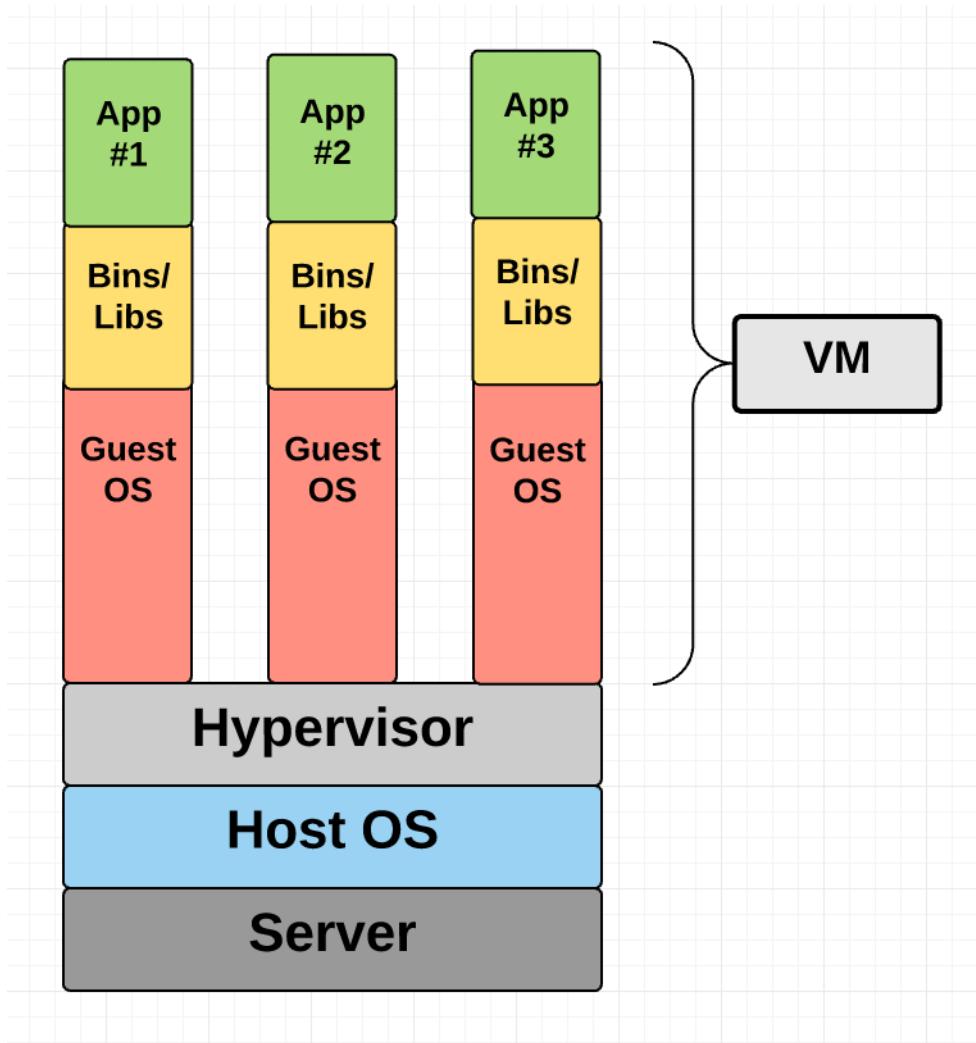
Containers and VMs are similar in their goals: to isolate an application and its dependencies into a self-contained unit that can run anywhere.

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.

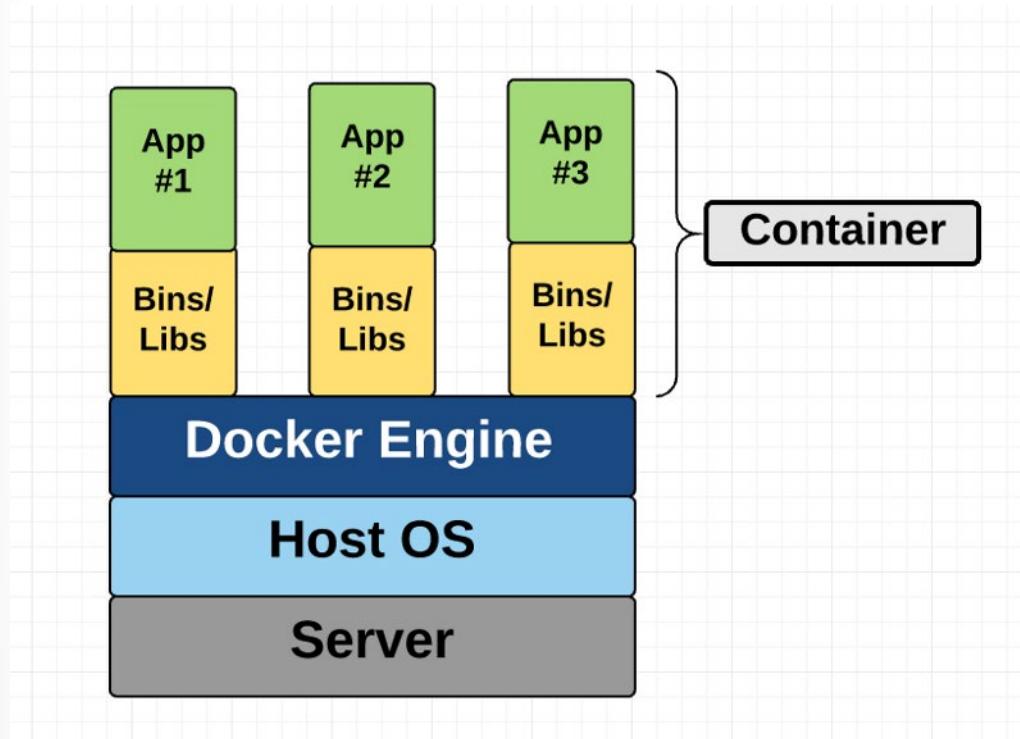
Virtual Machines

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.



Discussion - Containers versus Virtual Machines

Discussion: Containers versus 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, but 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**

Working with Docker containers

Container Lifecycle

The standard steps when working with containers are:

docker build - You create an image by executing a Dockerfile.

docker pull - You retrieve the image, likely from a container registry.

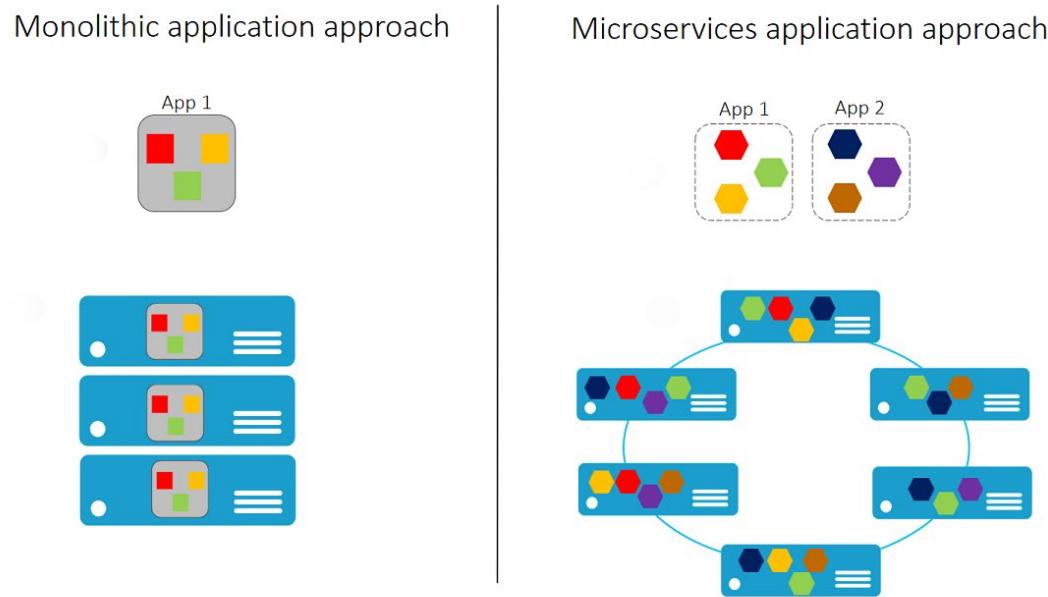
docker run - You execute the container. An instance is created of the image.

¹ <https://goto.docker.com/docker-for-the-virtualization-admin.html>

² <https://azure.microsoft.com/en-us/blog/containers-docker-windows-and-trends/>

Note that you can often just execute docker run without needing to first perform docker pull. In that case, Docker will pull the image and then run it. Next time, it won't need to pull it again.

Microservices and containers



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. Containers lend themselves well to this style of development.

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 programming, 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 service 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 provides 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

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/>

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

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.

Docker container registries

Both public and private container registries are available.

Public

Common public container registries are:

Docker Hub⁸

Red Hat Container Catalog⁹

Microsoft Container Registry¹⁰

While Microsoft Container Registry offers a public registry, you can also create your own private registries using it.

Private

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.

Demonstration: Create an Azure Container Registry

The 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.

⁷ <https://azure.microsoft.com/en-us/services/app-service/>

⁸ <https://hub.docker.com>

⁹ <https://access.redhat.com/containers>

¹⁰ <https://mcr.microsoft.com>

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.

```
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/resource-
Groups/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
```

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 reference¹¹](#)

¹¹ <https://docs.docker.com/engine/reference/builder/>

Implementing Docker multi-stage builds

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.

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 (i.e., 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 gregimages/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.

Multi-stage Dockerfiles

What are multi-stage Dockerfiles?

Multi-stage builds give the benefits of the builder pattern without the hassle of maintaining three separate files. Let's look at a multi-stage Dockerfile.

```
FROM mcr.microsoft.com/dotnet/core/aspnetcore:3.1 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/core/sdk:3.1 AS build
WORKDIR /src
COPY ["WebApplication1.csproj", ""]
RUN dotnet restore "./WebApplication1.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "WebApplication1.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "WebApplication1.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
```

```
ENTRYPOINT ["dotnet", "WebApplication1.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 build AS publish
```

build 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 (`mcr.microsoft.com/dotnet/core/sdk:3.1`), while creating a production image, based on an optimized runtime image (`mcr.microsoft.com/dotnet/core/aspnet:3.1`).

Notice the line

```
COPY --from=publish /app/publish .
```

This takes the `/app/publish` 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 `mcr.microsoft.com/dotnet/core/aspnet:3.1`. 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. `mcr.microsoft.com/dotnet/core/sdk:3.1` 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 build stage. It takes the compiled output and "publishes" them, in .NET terms. Publishing simply means take all the output required to deploy your "app/publish/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 build 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.

Considerations for multiple stage builds

Adopt container modularity

Try to avoid creating overly complex container images that couple together several 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 builds¹²](#).

Example builder pattern

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.

¹² <https://docs.docker.com/develop/develop-images/multistage-build/>

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 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.

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 repo¹³](#) 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 projects 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
```

¹³ <https://github.com/SteveLasker/AspNetCoreMultiProject>

```
[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

```
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.

Demonstration: 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 application¹⁴**.

¹⁴ <https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/docker/visual-studio-tools-for-docker?view=aspnetcore-3.1>

Lab

Deploying Docker Containers to Azure App Service web apps

Lab overview

In this lab, you will learn how to use an Azure DevOps CI/CD pipeline to build a custom Docker image, push it to Azure Container Registry, and deploy it as a container to Azure App Service.

Objectives

After you complete this lab, you will be able to:

- Build a custom Docker image by using an Microsoft hosted Linux agent
- Push an image to Azure Container Registry
- Deploy a Docker image as a container to Azure App Service by using Azure DevOps

Lab duration

- Estimated time: **60 minutes**

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions¹⁵

Modernizing your existing ASP.NET apps with Azure

Lab overview

If you decide to modernize your web applications as part of migration to Azure, you don't necessarily have to entirely re-architect them. Re-architecting an application by using a more advanced approach, such as micro-services, isn't always an option, because of cost and time restraints. However, you might be able to modernize your apps by leveraging containerization and Azure PaaS services. For example, migrating the data tier of your apps to a managed service offering such as Azure's SQL Database could be limited to an update to connection strings, without the need to change the underlying code.

In this lab, you will use the Nerd Dinner Application to illustrate this approach. Nerd Dinner is an open source ASP.NET MVC project. You will move the application DB to Azure SQL instance and add the Docker support to the application to run the application in Azure Container Instances.

¹⁵ <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/>

Objectives

After you complete this lab, you will be able to:

- Migrate LocalDB to SQL Server in Azure
- Use Docker tools in Visual Studio and add Docker support for applications
- Publish Docker images to Azure Container Registry (ACR)
- Push Docker images from ACR to Azure Container Instances (ACI)

Lab duration

- Estimated time: **60 minutes**

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- **AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions¹⁶**

¹⁶ <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/>

Module review and takeaways

Module review questions

Review Question 1

You are reviewing an existing Dockerfile. How would you know if it's a multi-stage Dockerfile?

Review Question 2

You are designing a multi-stage Dockerfile. How can one stage refer to another stage within the Dockerfile?

Review Question 3

What is the line continuation character in Dockerfiles?

Review Question 4

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.

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 16 Creating and Managing 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. Today, it is open source but it is owned by the Cloud Native Computing Foundation.

Because the Kubernetes platform is open-source and has so many supporters, it is growing rapidly through contributions. 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. It is currently owned by the Cloud Native Computing Foundation and it is open source.

Kubernetes is often referred to as *k8s*, it's a 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. Today though, all the industry interest appears to be in Kubernetes.

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, 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.

If you were manually deploying Kubernetes, you would need to pay for the resources for the master nodes.

¹ <https://mesosphere.com/product/>

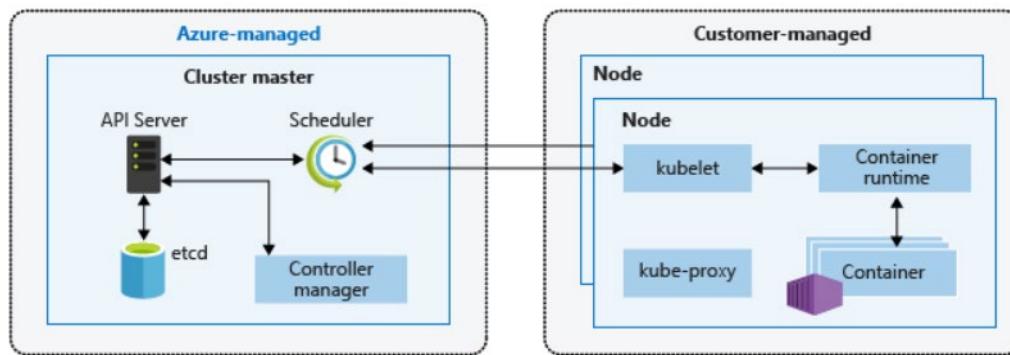
² <https://www.docker.com/products/orchestration>

³ <https://kubernetes.io/>

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.



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 several 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. Therefore, 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.

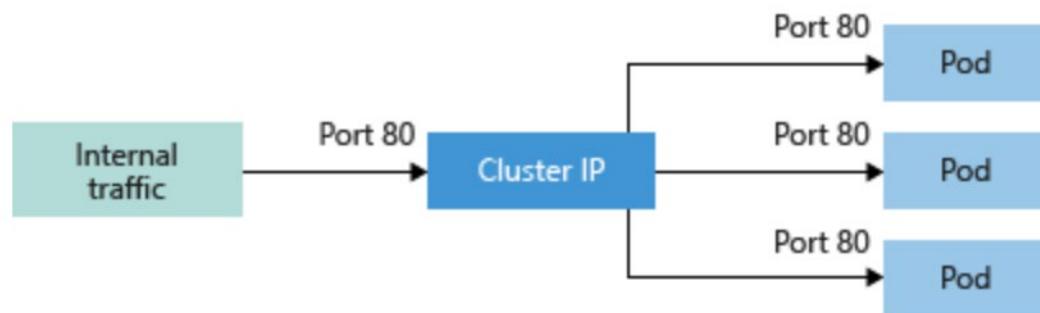
Kubernetes Service is an abstraction that defines a logical set of pods, combined with a policy that describes how to access them. Where pods have a shorter lifecycle, services are usually more stable and are not affected by container updates. This means that you can safely configure applications to interact with pods by using 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.

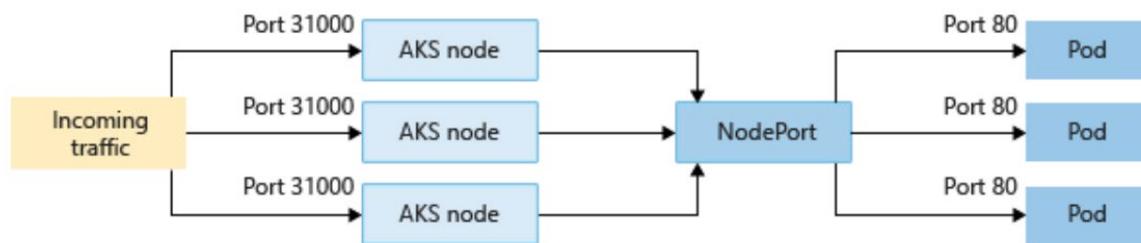
Services

The following Service types are available:

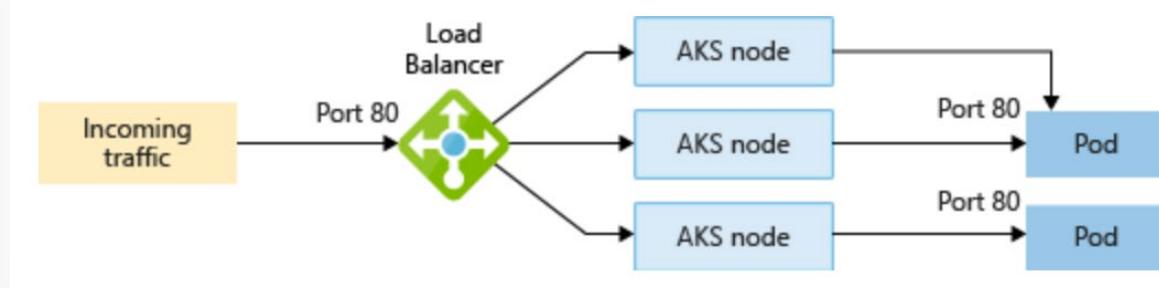
- **ClusterIP** This service creates an internal IP address for use within the AKS cluster. However, it's good for internal-only applications that support other workloads within the cluster.



- **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.



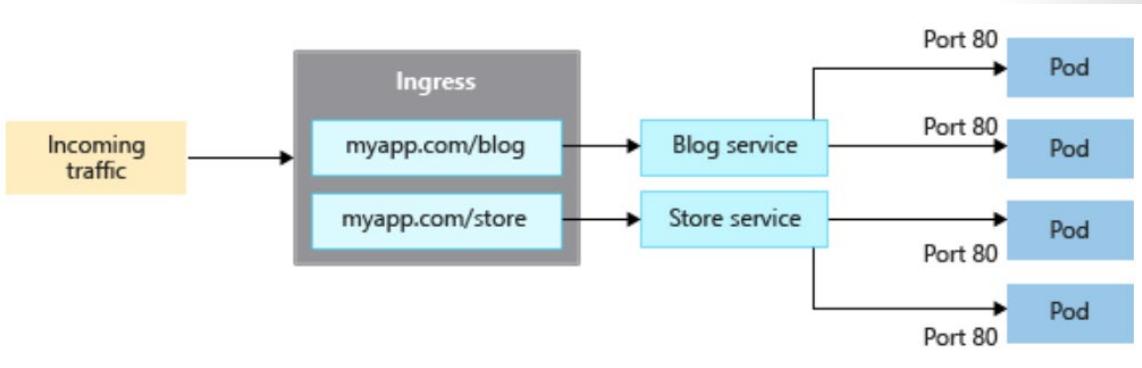
- **LoadBalancer** This service creates an Azure Load Balancer resource, configures an external IP address, and connects the requested pods to the load balancer backend pool. To allow customers traffic to reach the application, load balancing rules are created on the desired ports.



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 units

`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

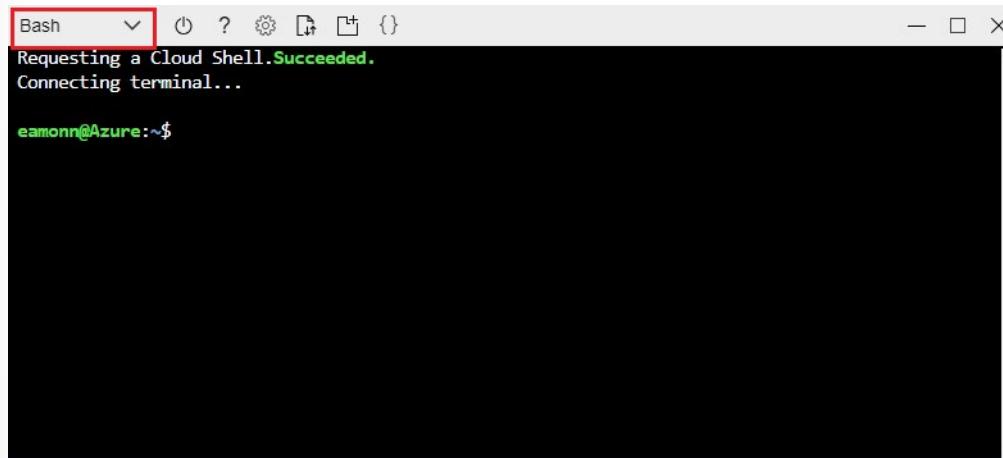
This walkthrough shows how to deploy an AKS cluster using the Azure CLI. A multi-container application that includes a web front end and a Redis Cache instance is run in the cluster. You then see how to monitor the health of the cluster and pods that run your application.

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
```

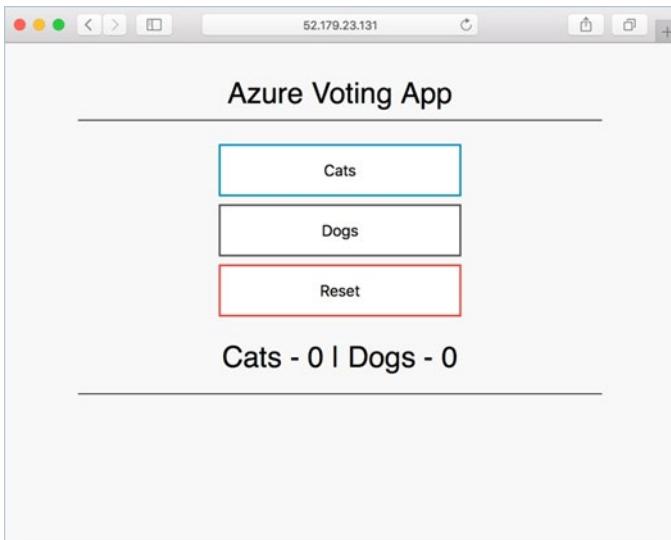
10. Initially the **EXTERNAL-IP** for the azure-vote-front service is shown as pending.

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:

Container: azure-vote-back

- View container logs (preview)
- View container logs
- Container Name: azure-vote-back
- Container ID: 062803a24018e2bda9bf4353bf923934889ec8562e94f43b6b8b30d213044
- 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 m
- CPU Request: 100 m
- 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.

```
let startDateTime = datetime('2018-12-18T12:45:00.000Z');
let endDateTime = datetime('2018-12-18T18:50:34.378Z');
let ContainerIDList = KubepodInventory
| where TimeGenerated >= startDateTime and TimeGenerated < endDateTime
| where ContainerName == '37d12cec-02f9-11e9-8d50-8e4fb98784de/azure-vote-back'
| distinct ContainerID;
ContainerLog
| where TimeGenerated >= startDateTime and TimeGenerated < endDateTime
| where ContainerID in (ContainerIDList)
| project LogEntrySource, LogEntry, TimeGenerated, Computer, Image, Name, ContainerID
| order by TimeGenerated desc
| render table
```

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-nodepool1-31718369-0	redis:latest
> stdout	1:M 18 Dec 2018 18:54:08.774 * Ready to accept connections	2018-12-18T18:54:08.774	aks-nodepool1-31718369-0	redis:latest
> stdout	1:M 18 Dec 2018 18:54:08.774 # WARNING you have Transparent Hu...	2018-12-18T18:54:08.774	aks-nodepool1-31718369-0	redis:latest
> stdout	1:M 18 Dec 2018 18:54:08.774 # WARNING: The TCP backlog setting ...	2018-12-18T18:54:08.774	aks-nodepool1-31718369-0	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-nodepool1-31718369-0	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-nodepool1-31718369-0	redis:latest
> stdout	1:C 18 Dec 2018 18:54:08.773 # o0000000000 Redis is starting o...	2018-12-18T18:54:08.773	aks-nodepool1-31718369-0	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
```

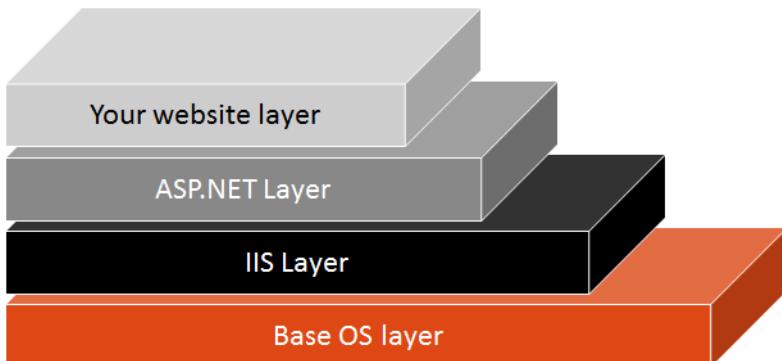
The YAML file contains a specification of the deployment. The **apply** command is convenient because it makes no difference whether the deployment was already on the cluster. This means that you can always use the exact same steps regardless of whether you are doing an initial deployment or an update 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, considering 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

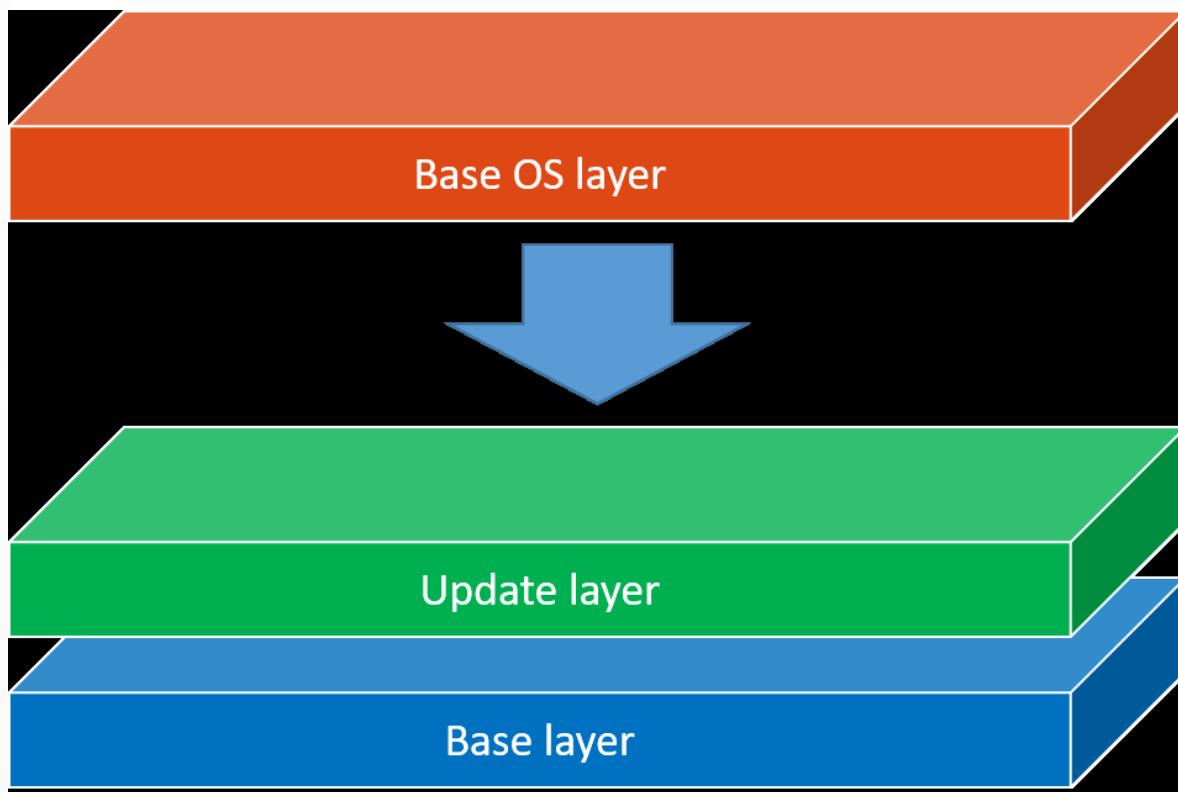
After you've successfully containerized your application, you'll need to ensure that you update your image regularly. This entails creating a new image for every change you make in your own code and ensuring that all layers receive regular patching.

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 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/windows-servercore`.

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 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.

Kubernetes tooling

kubectl

kubectl is a command-line tool for running commands against Kubernetes clusters. Deploy applications, manage cluster resources.

Some common commands for kubectl are shown below:

| Common Commands | |
|-----------------|--|
| annotate | Add/update annotations for resources |
| apply | Apply configuration changes |
| autoscale | Scale pods managed by a replication controller |
| certificate | Modify certificate resources |
| cluster-info | Display endpoint information about master and services |
| config | Modify kubeconfig files |
| cp | Copies files to/from containers |
| describe | Show detailed state about resources |
| exec | Execute a command against a container |
| label | Add/update labels for resources |
| logs | Print the logs for a container |
| run | Run an image on a container |

For more information on kubectl commands and resource types, see: [Overview of kubectl](#)⁵

Helm

Helm is a package manager for Kubernetes. It makes it easier to package, configure, and deploy applications and services.

helm (lower-case) is the command line tool that provides a user-interface to the functionality of Helm.

tiller was a server-side component that executed Helm packages. From Helm 3 onwards, tiller will no longer be required.

Helm packages are called **charts** and are implemented in YAML. There are public and private repositories for Helm charts.

For more information on Helm, see: [Helm](#)⁶

Kubernetes extension for Visual Studio Code

The **Kubernetes Tools extension for Visual Studio Code** lets you quickly develop Kubernetes manifests and helps with the creation of Helm charts.

⁵ <https://kubernetes.io/docs/reference/kubectl/overview/>

⁶ <https://helm.sh/>



You can deploy containerized micro-service-based applications to local minikube clusters or to Azure Kubernetes clusters and debug live applications running in containers on the clusters.

The extension allows you to browse and manage Kubernetes clusters from within VS Code and helps to streamline Kubernetes development.

For more information on the extension, see: [Working with Kubernetes in VS Code](#)⁷

For more information on local minikube clusters, see: [Using Minikube to Create a Cluster](#)⁸

⁷ <https://code.visualstudio.com/docs/azure/kubernetes>

⁸ <https://kubernetes.io/docs/tutorials/kubernetes-basics/create-cluster/cluster-intro/>

Integrating AKS with Pipelines

Integrating AKS with pipelines

Pipelines will need to deploy containers from images. For this, you should create a AKS cluster and an AKS container registry.

The AKS cluster will need to retrieve the image from the container registry. You'll need to grant the AKS service principal permission to access the new container registry.

In the pipeline, useful tasks will be:

- Create Deployments & Services in AKS
- Update image in AKS

You will see an example of implementing these tasks in the upcoming lab.

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.

Responsibilities

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.

Integration

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
```

Readiness, startup and liveness probes

You might need to wait for a container to be ready to serve clients when it first starts, or you might want to check if a container is still alive. For these purposes, you will need readiness and/or liveness probes.

Kubernetes offers a variety of readiness probes. In the configuration YAML file, you can configure the type of probe:

- startupProbe
- livenessProbe
- readinessProbe

You can also configure the initialDelaySeconds (how long after start before checking), and periodSeconds (how often to check).

Restarting an unresponsive container can be achieved by configuring the restartPolicy setting in the configuration file.

An example is:

```
ports:  
- name: liveness-port  
  containerPort: 8080  
  hostPort: 8080  
  
livenessProbe:  
  httpGet:  
    path: /health-check  
    port: liveness-port  
  failureThreshold: 2  
  periodSeconds: 15  
  
startupProbe:  
  httpGet:  
    path: /health-check  
    port: liveness-port  
  failureThreshold: 40  
  periodSeconds: 15
```

For more information on Kubernetes readiness probes, see here: **Configure Liveness, Readiness and Startup Probes⁹**

⁹ <https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/>

Lab

Deploying a multi-container application to Azure Kubernetes Services

Lab overview

Azure Kubernetes Service (AKS)¹⁰ is the quickest way to use Kubernetes on Azure. **Azure Kubernetes Service (AKS)** manages your hosted Kubernetes environment, making it straightforward to deploy and manage containerized applications without requiring container orchestration expertise. It also enhances agility, scalability, and availability of your containerized workloads. Azure DevOps further streamlines AKS operations by providing continuous build and deployment capabilities.

In this lab, you will use Azure DevOps to deploy a containerized ASP.NET Core web application **My-HealthClinic** (MHC) to an AKS cluster.

Objectives

After you complete this lab, you will be able to:

- Create an Azure DevOps team project with a .NET Core application using the Azure DevOps Demo Generator tool.
- Use Azure CLI to create an Azure Container registry (ACR), an AKS cluster and an Azure SQL database
- Configure containerized application and database deployment by using Azure DevOps
- Use Azure DevOps pipelines to build to automatically deploy containerized applications

Lab duration

- Estimated time: **60 minutes**

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- [AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions¹¹](https://microsoftlearning.github.io/AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions/)

¹⁰ <https://azure.microsoft.com/en-us/services/kubernetes-service/>

¹¹ <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/>

Module review and takeaways

Module review questions

Review Question 1

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

Review Question 2

Kubernetes CLI is called _____.

- HELM
- ACI
- AKS
- KUBECTL

Review Question 3

For workloads running in AKS Kubernetes Web Dashboard allows you to view _____. Select all that apply.

- Config Map & Secrets
- Logs
- Storage
- Azure Batch Metrics

Review Question 4

Pods can be described using which of the following languages? Select all that apply.

- JSON
- XML
- PowerShell
- YAML

Answers

Review Question 1

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

Review Question 2

Kubernetes CLI is called _____.

- HELM
- ACI
- AKS
- KUBECTL

Review Question 3

For workloads running in AKS Kubernetes Web Dashboard allows you to view _____.

Select all that apply.

- Config Map & Secrets
- Logs
- Storage
- Azure Batch Metrics

Review Question 4

Pods can be described using which of the following languages? Select all that apply.

- JSON
- XML
- PowerShell
- YAML

Module 17 Implementing Feedback for 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 must 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:

- 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

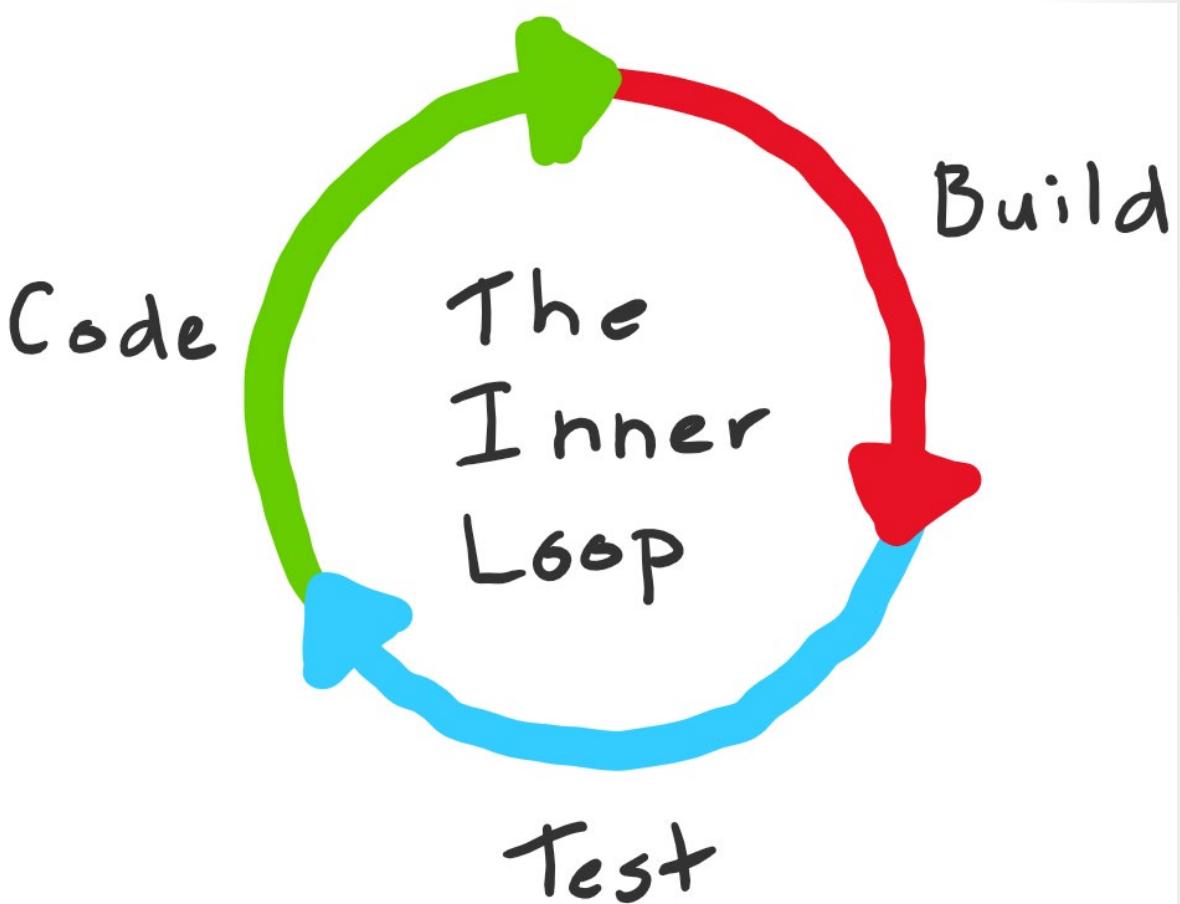
Implement tools to track system usage, feature usage, and flow

The inner loop

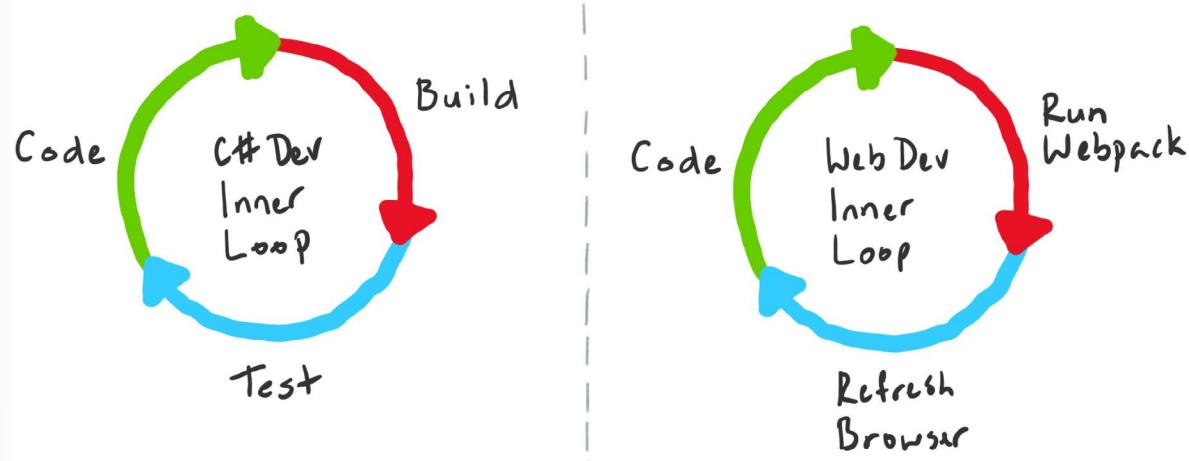
It's not clear who coined the term "inner loop" in the context of software engineering, but within Microsoft at least the term seems to have stuck. Many of the internal teams that I work with see it as something that they want to keep as short as possible - but what is the inner loop?

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 were 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 were 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).



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.

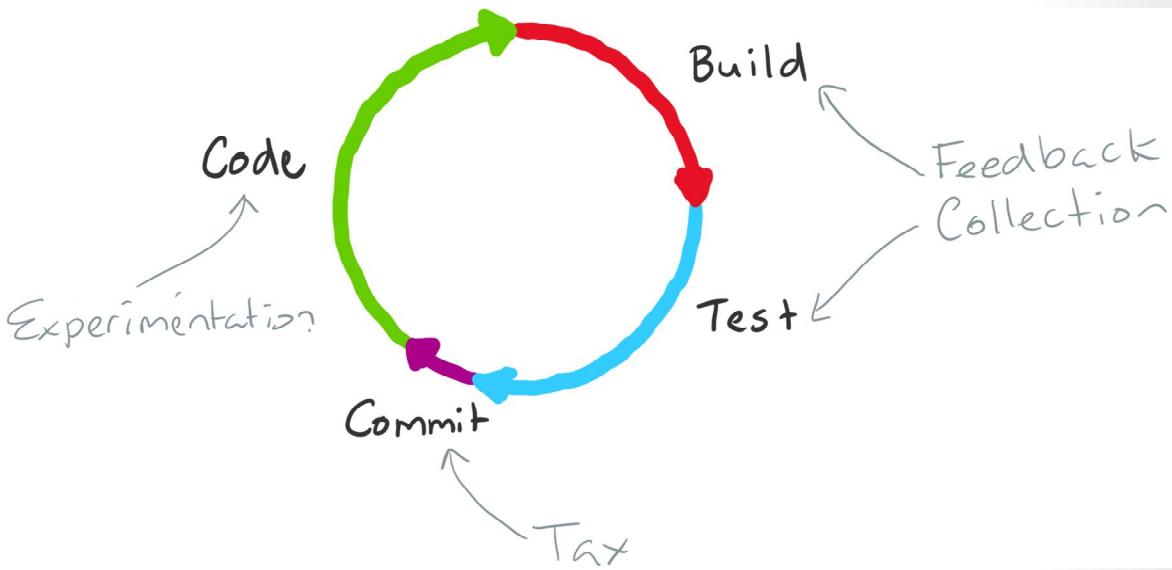
Understanding the loop

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 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, nor provide feedback. Tax is necessary work. If it's 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 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 several 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 codebase into small units and share binaries.

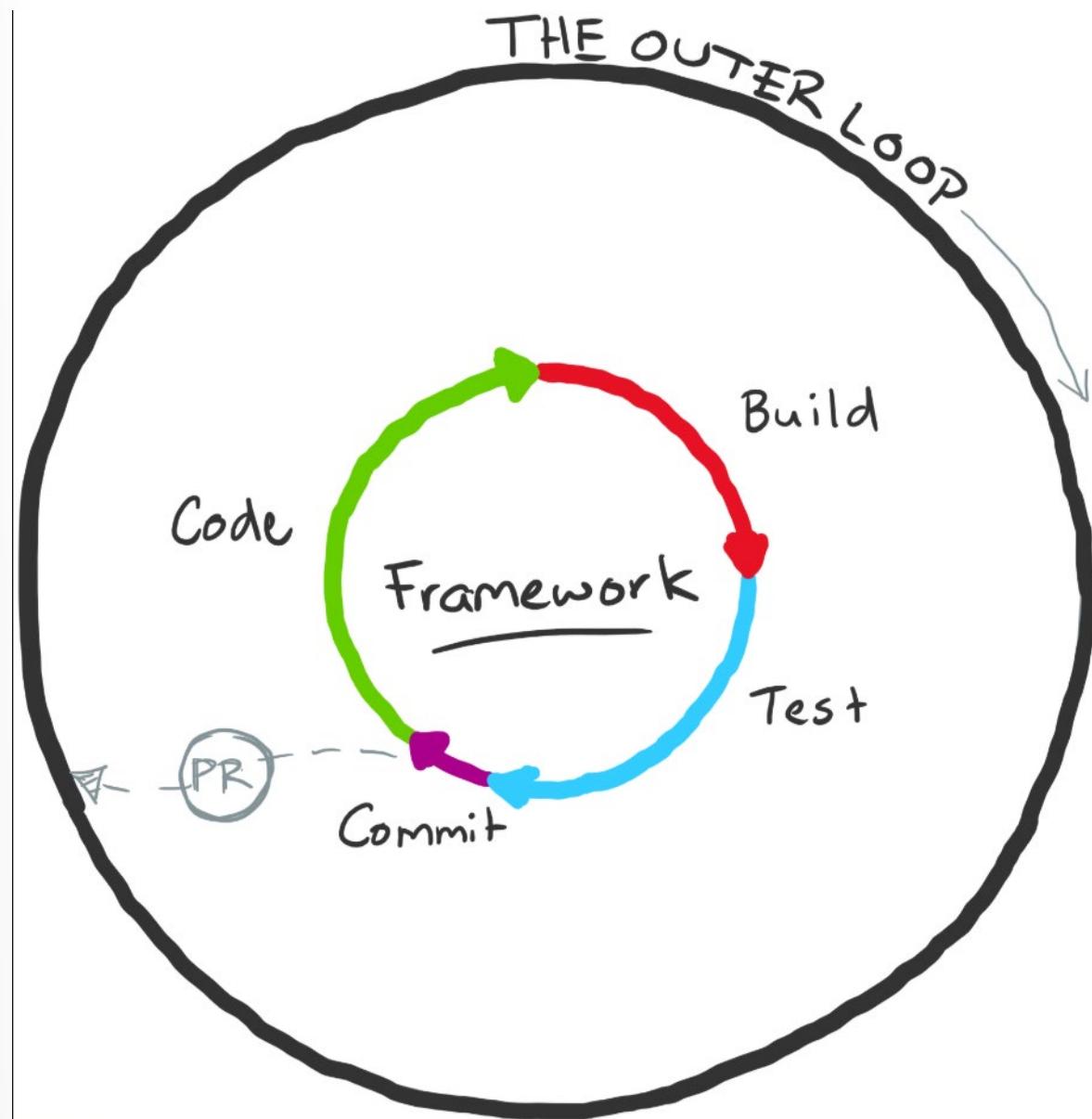
How you tackle each one of those is probably a blog post. At Microsoft, for some of our truly massive monolithic codebases we are investing quite heavily in #1 and #2 - but #3 requires a special mention because it can be a double-edged sword and if done incorrectly and can have the opposite of the desired 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.

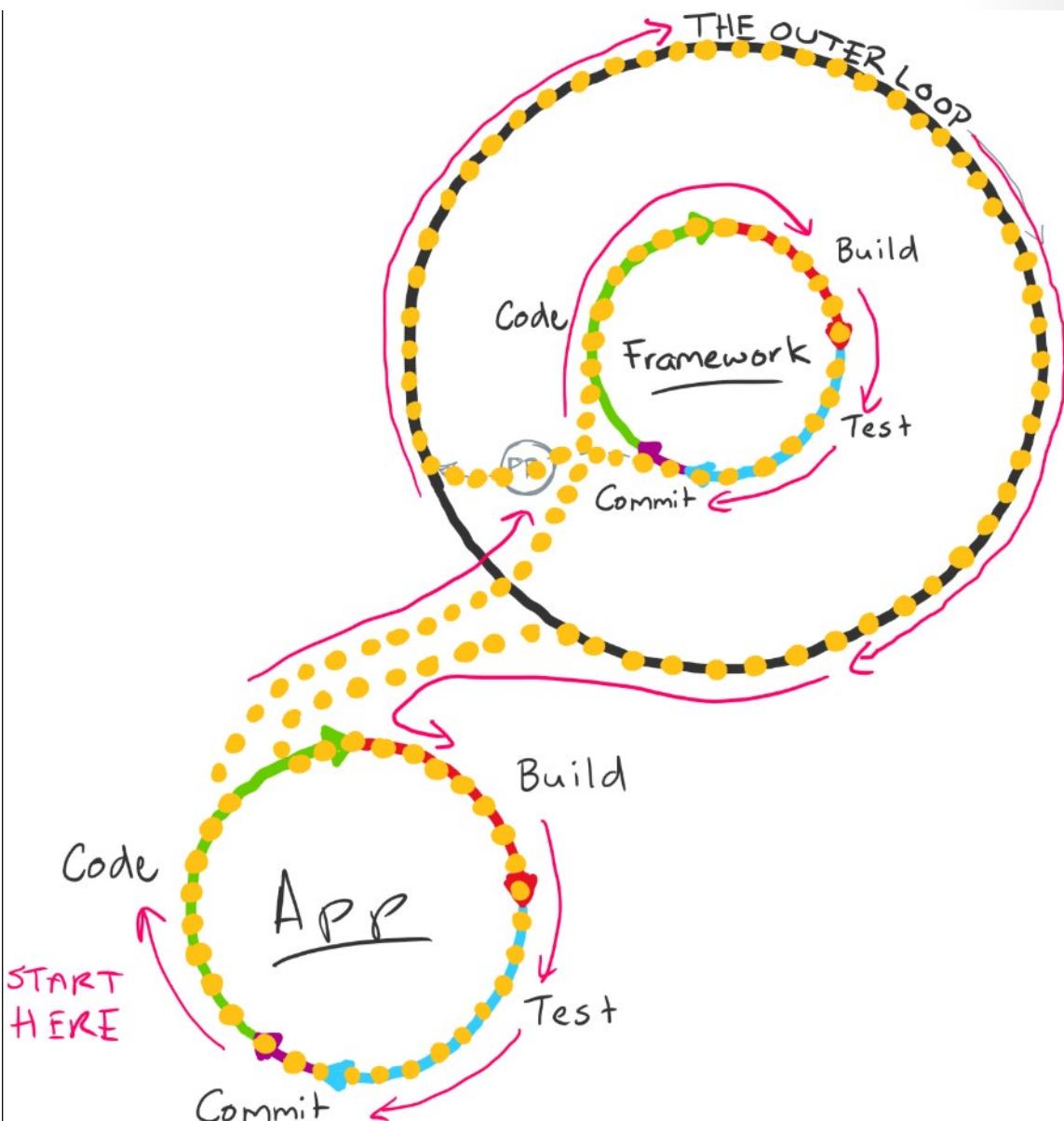
When someone needs to change this framework code, they clone down the repository, make their changes (a separate inner loop) and submit a PR which is the transition of the workflow from the inner loop to the outer loop. The framework package would then be available to be pulled into dependent applications (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 a 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 messy quick and you must pay that outer loop tax at some point.

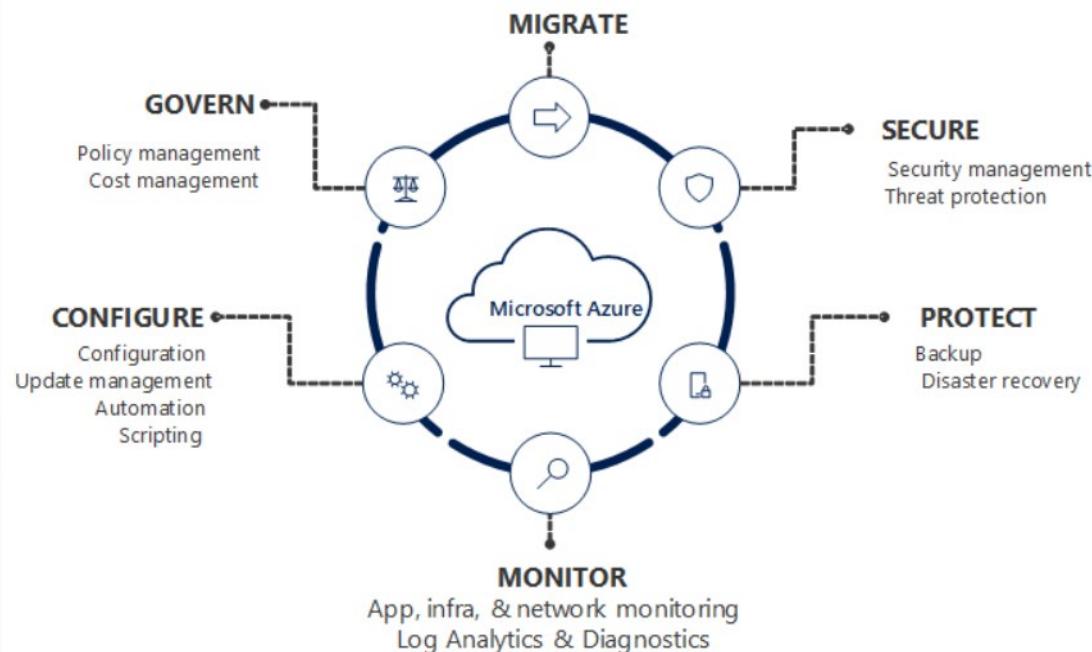
This isn't to say that breaking code up into separate packages is an inherently bad thing - it can work brilliantly; you just need to make those incisions carefully.

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.

Introduction to continuous monitoring



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.

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>

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.

Enable monitoring for all your applications⁴

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 Projects**⁵ 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 pipeline**⁶ allows you to gate or rollback your deployment based on monitoring data.
- **Status Monitor**⁷ 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 Insights**⁸ by installing the Azure Monitor Application Insights SDK for **.NET**⁹, **Java**¹⁰, **Node.js**¹¹, or **any other programming languages**¹². 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 logs**¹⁴ from most of your Azure resources with no configuration.
- Enable deeper monitoring for VMs with **Azure Monitor for VMs**¹⁵.
- Enable deeper monitoring for AKS clusters with **Azure Monitor for containers**¹⁶.
- Add **monitoring solutions**¹⁷ for different applications and services in your environment.

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>

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 templates**¹⁹ to enable monitoring and configure alerts over a large set of resources.
- 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.

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>

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>

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.
- 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.

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>

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/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>

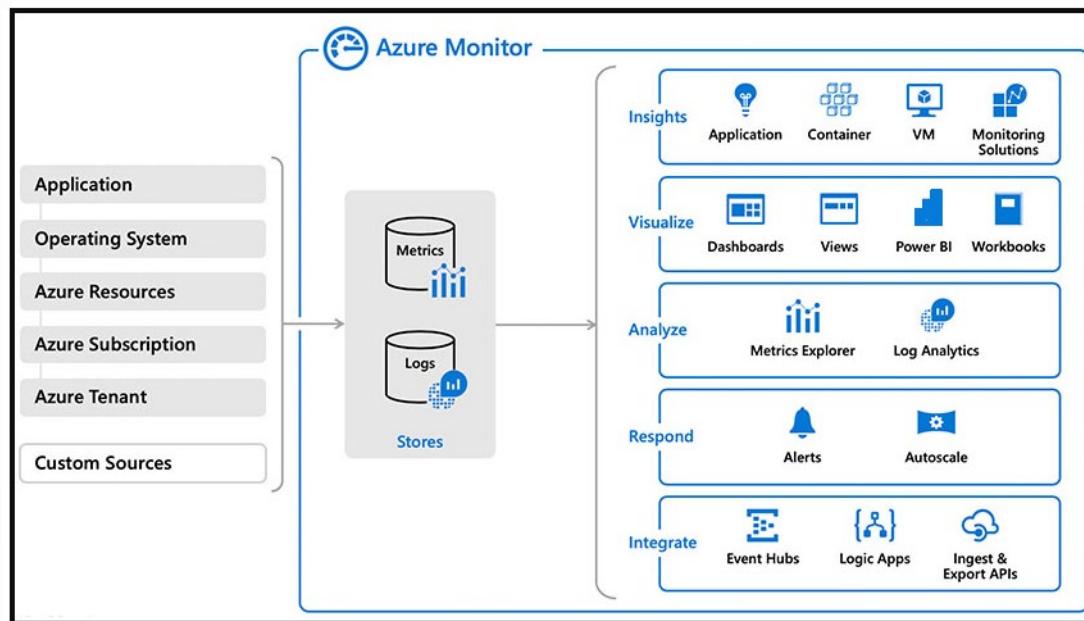
⁴⁰ <https://docs.microsoft.com/en-us/azure/azure-monitor/continuous-monitoring#continuously-optimize>

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>

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.



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.
- Analyze 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"
```

⁴³ <http://shell.azure.com/powershell>

```
# 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
}

# 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 ++
    }
}

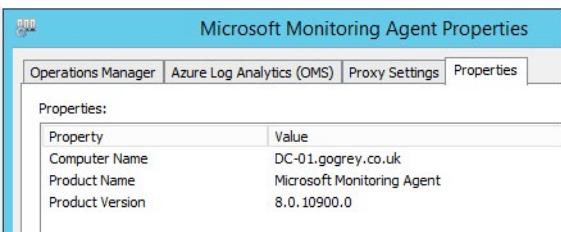
AddPerfCounters -PerfObject $ObjectLogicalDisk -PerfCounter $CounterLogicalDisk -Instance $InstanceNameAll
AddPerfCounters -PerfObject $ObjectNetworkAdapter -PerfCounter $CounterNetworkAdapter -Instance $InstanceNameAll
AddPerfCounters -PerfObject $ObjectNetworkInterface -PerfCounter $CounterNetworkInterface -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 -In-  
stance $InstanceNameAll
```

6. 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.

How it works

1. 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.
2. 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.



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.

In the next lessons, we will continue by querying the log analytics data.

Kusto Query Language (KQL) Walkthrough

Kusto is the primary way to query Log Analytics. It provides both a query language and a set of control commands.

Kusto can be used directly within Azure Data Explorer.

Azure Data Studio also offers a Kusto query experience and supports the creation of Jupiter-style notebooks for Kusto queries.

See [Getting Started with Kusto Queries](#)⁴⁵

⁴⁴ <https://www.jam-software.com/heavyload/>

⁴⁵ <https://docs.microsoft.com/en-us/azure/data-explorer/kusto/concepts/>

Walkthrough

Note: This walkthrough continues the previous lesson on Azure Log Analytics and the walkthrough started within it.

1. Log into **Azure Portal⁴⁶** and navigate to the log analytics workspace. From the left blade in the log analytics workspace click Logs. This will open 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
```

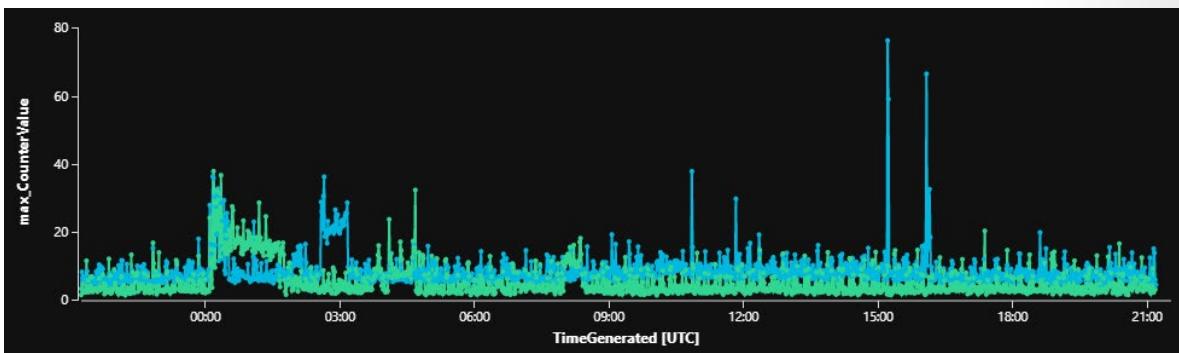
4. Show a count of the data points collected in the last 24 hours. In the result below, you can see we have 66M data points that we are able to query against in near real time to analyse and correlate insights.

The screenshot shows the Azure Log Analytics interface. In the search bar at the top, the query "search '*' | count" is entered. Below the search bar, a message says "Completed. Showing results from the last 24 hours." Underneath, there are three tabs: TABLE, CHART, and Columns. A tooltip "Drag a column header and drop it here to group by that column" is visible. A table is displayed with one row, showing the value "88,685,260" in yellow. The entire interface has a dark theme.

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.

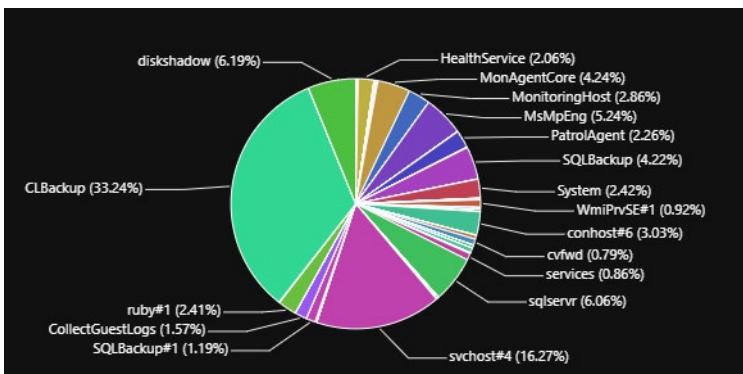
```
Perf  
| where ObjectName == "Processor" and InstanceName == "_Total"  
| summarize max(CounterValue) by Computer, bin(TimeGenerated, 1m)  
| render timechart
```

⁴⁶ <https://portal.azure.com>



6. 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
```



There's more

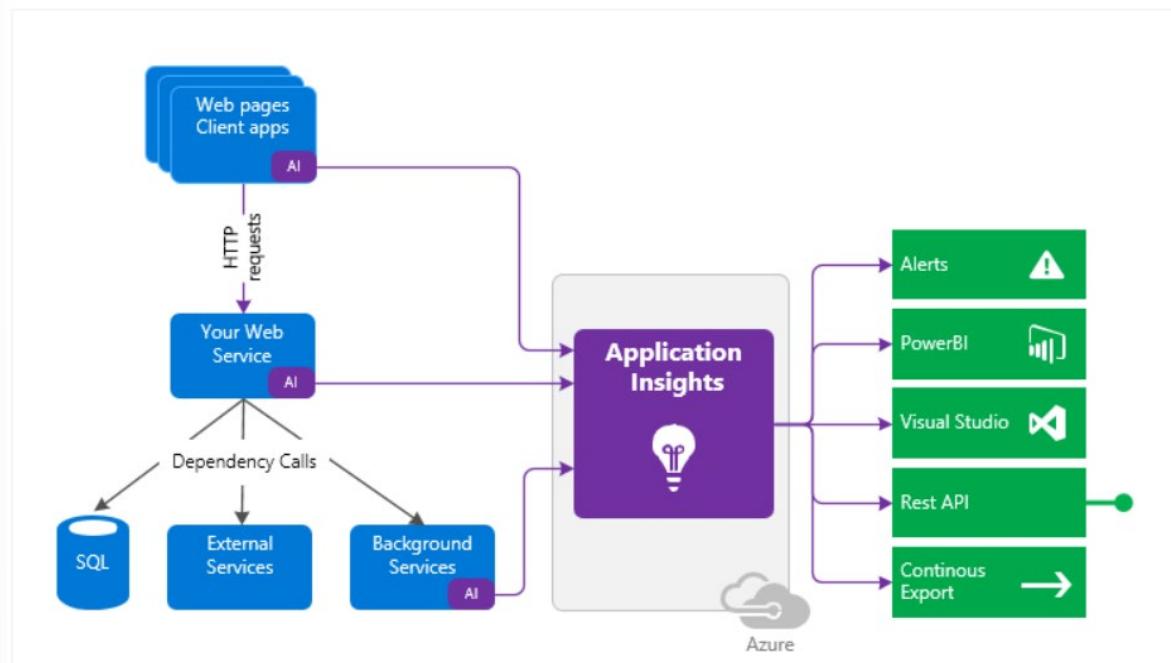
This tutorial has introduced you to the basic concepts of Log Analytics and how to get started with the basics. We've only scratched the surface of what's possible with Log Analytics. I would encourage you to try out the advanced tutorials available for Log Analytics on [Microsoft Docs](#)⁴⁷

Application Insights

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.

⁴⁷ <https://docs.microsoft.com/en-us/azure/azure-monitor/>



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.

What's the overhead?⁴⁸

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?⁴⁹

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 - Analyze 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.

⁴⁸ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-overview?toc=/azure/azure-monitor/toc.json#whats-the-overhead>

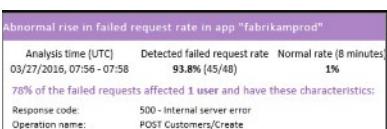
⁴⁹ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-overview?toc=/azure/azure-monitor/toc.json#what-does-application-insights-monitor>

- 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?

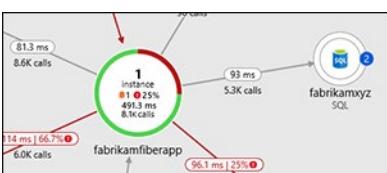
There are plenty of ways to explore your data. Check out this article for more information - **Smart detection and manual alerts**⁵⁰

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 alerts**⁵¹ on levels of custom or standard metrics.



Application map⁵²

The components of your app, with key metrics and alerts.



Profiler⁵³

Inspect the execution profiles of sampled requests.



⁵⁰ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-proactive-diagnostics>

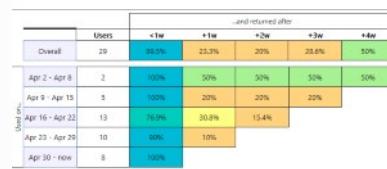
⁵¹ <https://docs.microsoft.com/en-us/azure/azure-monitor/app/alerts>

⁵² <https://docs.microsoft.com/en-us/azure/azure-monitor/app/app-map>

⁵³ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-profiler>

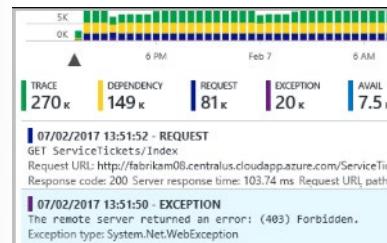
Usage analysis⁵⁴

Analyze user segmentation and retention.



Diagnostic search for instance data⁵⁵

Search and filter events such as requests, exceptions, dependency calls, log traces, and page views.



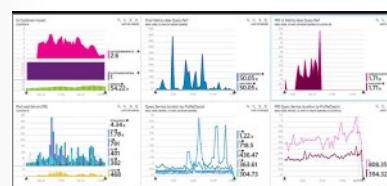
Metrics Explorer for aggregated data

Explore, filter, and segment aggregated data such as rates of requests, failures, and exceptions, response times, page load times.



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.

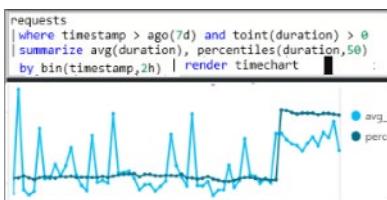
⁵⁴ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-usage-overview>

⁵⁵ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-diagnostic-search>



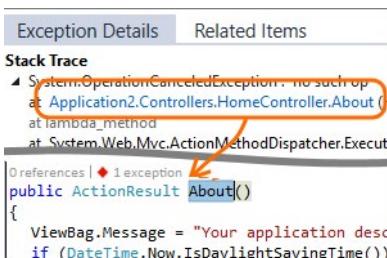
Analytics

Answer tough questions about your app's performance and usage by using this powerful query language.



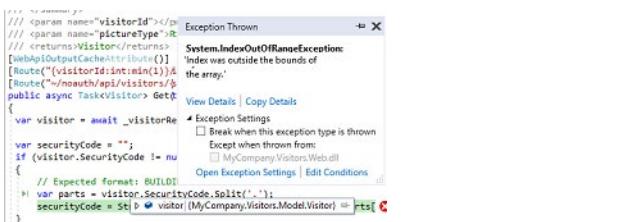
Visual Studio

See performance data in the code. Go to code from stack traces.



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**⁵⁶, and:

- Set up a **dashboard**⁵⁷ 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**⁵⁸ 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.

⁵⁶ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-monitor-web-app-availability>

⁵⁷ <https://docs.microsoft.com/en-us/azure/azure-monitor/app/app-insights-dashboards>

⁵⁸ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-live-stream>

- Examine profiler, snapshots, stack dumps, and trace logs.

Build, Measure, Learn

Measure the effectiveness⁵⁹ 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**⁶⁰. It's free to sign up, and if you choose the basic **pricing plan**⁶¹ 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.

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**⁶²
- **Azure web app or VM**⁶³
- **J2EE**⁶⁴

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**⁶⁵ 2013 update 2 or later.
- **Java**⁶⁶
- Node.js
- **Other platforms**⁶⁷
- **Instrument your web pages**⁶⁸ for page view, AJAX, and other client-side telemetry.

⁵⁹ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-usage-overview>

⁶⁰ <https://azure.com/>

⁶¹ <https://azure.microsoft.com/pricing/details/application-insights/>

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-monitor-performance-live-website-now>

⁶⁴ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-java-live>

⁶⁵ <https://docs.microsoft.com/en-us/azure/azure-monitor/app/asp-net>

⁶⁶ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-java-get-started>

⁶⁷ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-platforms>

⁶⁸ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-javascript>

- **Analyze mobile app usage⁶⁹** by integrating with Visual Studio App Center.
- **Availability tests⁷⁰** - ping your website regularly from our servers.

Demonstration: Adding Application Insights to an ASP.NET core application

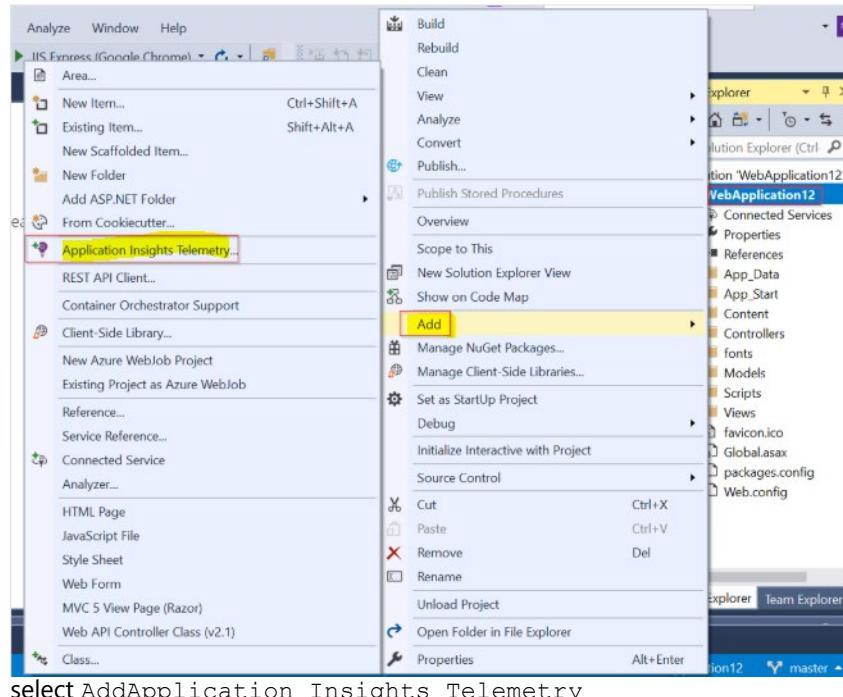
Application performance management (APM) is a discipline that includes all the tools and activities involved in observing how software and hardware are performing. These tools present that performance information in a form product owners and software development teams can use to make decisions. Application 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:
 - 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



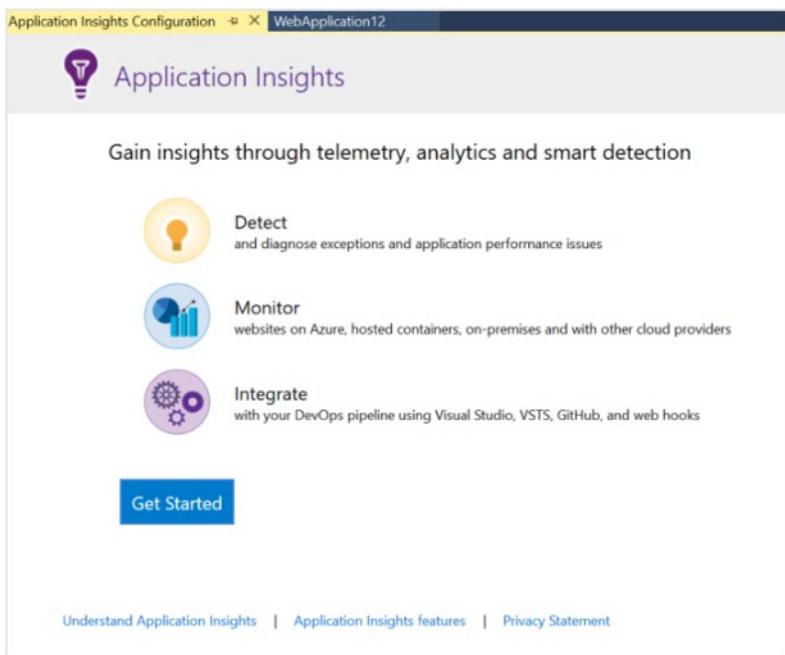
select AddApplication Insights Telemetry

⁶⁹ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-mobile-center-quickstart>

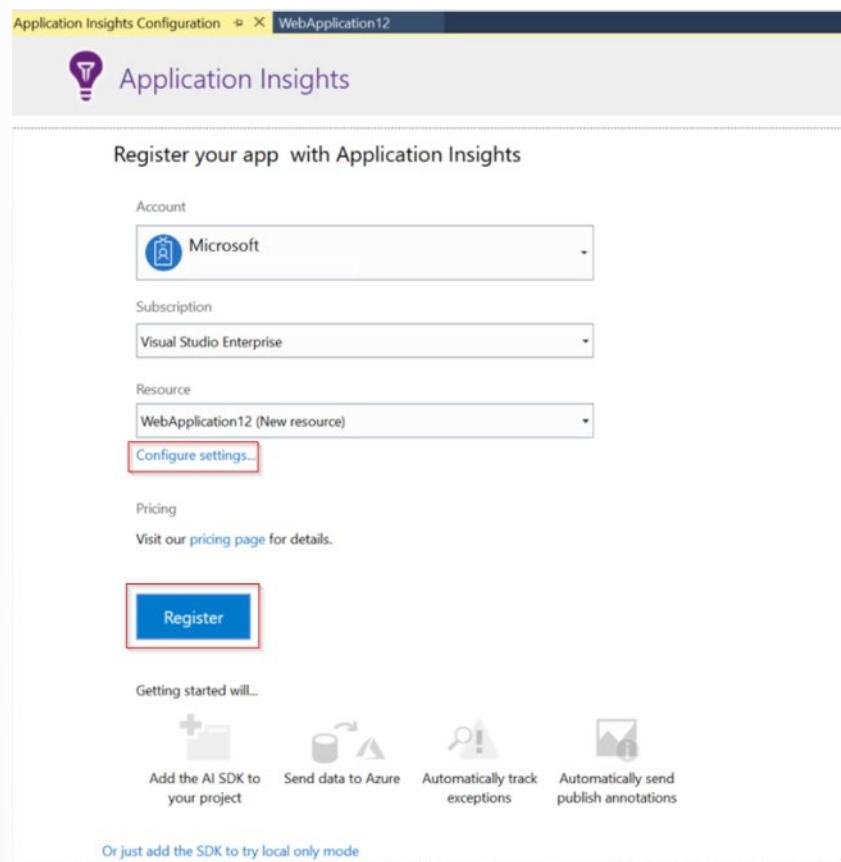
⁷⁰ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-monitor-web-app-availability>

(Depending on your Application Insights SDK version you may be prompted to upgrade to the latest SDK release. If prompted, select Update SDK.)

- From the Application Insights configuration screen, click Get started to start setting up App Insights.



- Choose to set up a new resource group and select the location where you want the telemetry data to be persisted.

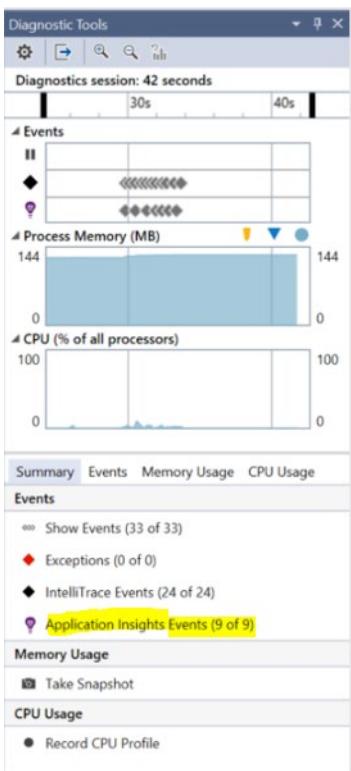


Summary

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.

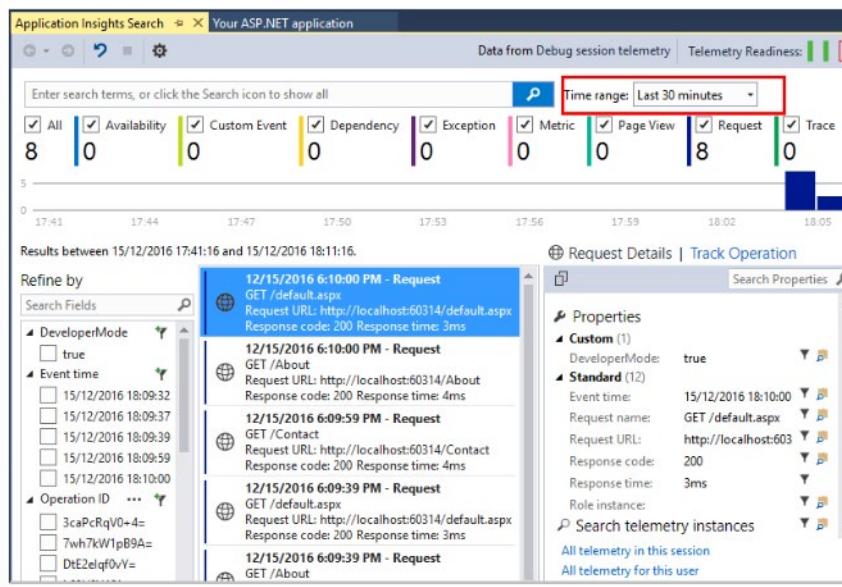
How to do it

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.

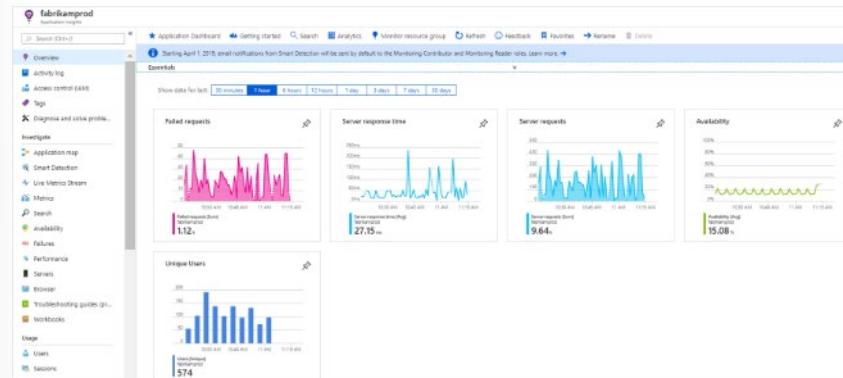
In the Visual Studio Application Insights Search window, you will see the data from your application for telemetry generated in the server side of your app. Experiment with the filters, and click any event to see more detail.



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 Application⁷¹**.

⁷¹ <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

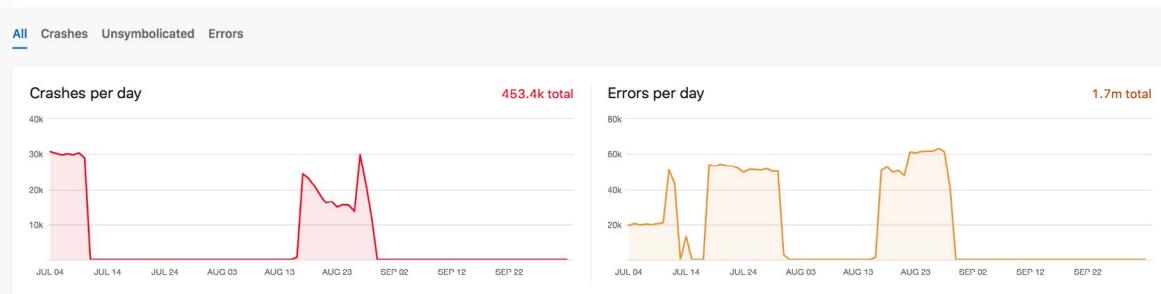
Crashes are what happens when a runtime exception occurs from an unexpected event that terminates the app. These are errors not handled by a try/catch block. When a crash occurs, App Center Crashes records the state of the app and device and automatically generates a crash log. These logs contain valuable 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: CRLCrashROPage.m line 42
SIGBUS | | ↳ 15 | ↳ 4 | 5.0.0 (60) | Open | 4 days ago |
| <input type="checkbox"/> #268: CRLCrashNULL.m line 37
SIGSEGV | ☰ | ↳ 7 | ↳ 2 | 5.0.0 (60) | Open | 4 days ago |
| <input type="checkbox"/> #270: CRLCrashStackGuard.m line 38
SIGBUS | | ↳ 3 | ↳ 1 | 5.0.0 (60) | Open | 5 days ago |
| <input type="checkbox"/> #304: CRLCrashNSLog.m line 41
SIGSEGV | | ↳ 1 | ↳ 1 | 5.0.0 (1529...) | Open | 1 week ago |
| <input type="checkbox"/> #303: CRLCrashROPage.m line 42
SIGBUS | | ↳ 1 | ↳ 1 | 5.0.0 (1529...) | Open | 1 week ago |
| <input type="checkbox"/> #286: CRLCrashPrivInst.m line 52
SIGILL | | ↳ 9 | ↳ 3 | 5.0.0 (60) | Open | 2 weeks ago |
| <input type="checkbox"/> #288: CRLCrashReleasedObject.m line 51
SIGSEGV | | ↳ 7 | ↳ 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 [Android⁷²](#), [iOS⁷³](#), [macOS⁷⁴](#), [React Native⁷⁵](#), [Xamarin⁷⁶](#), and [Apache Cordova⁷⁷](#) apps.

To view and download the attachments, select a crash group, a specific device report and then click on the attachments tab.

⁷² <https://docs.microsoft.com/en-us/appcenter/sdk/crashes/android#add-attachments-to-a-crash-report>

⁷³ <https://docs.microsoft.com/en-us/appcenter/sdk/crashes/ios#add-attachments-to-a-crash-report>

⁷⁴ <https://docs.microsoft.com/en-us/appcenter/sdk/crashes/macos#add-attachments-to-a-crash-report>

⁷⁵ <https://docs.microsoft.com/en-us/appcenter/sdk/crashes/react-native#add-attachments-to-a-crash-report>

⁷⁶ <https://docs.microsoft.com/en-us/appcenter/sdk/crashes/xamarin#add-attachments-to-a-crash-report>

⁷⁷ <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 the device as "iPhone 8 Plus (A1864/A1898/A1899)" running iOS 11.4 (15F79). Below this, a timeline lists several events from "iPhone 8 Plus (A1864/A1898/A1899)" occurring "2w ago". The events include: "-[CRLCrashROPage crash]", "CRLCrashROPage.m - line 42", and "SIGBUS". The interface includes tabs for OVERVIEW, THREADS, EVENTS, and ATTACHMENTS (which is selected), and a RAW link. Under ATTACHMENTS, there is a section for ATTACHED BINARY containing a file named "144x144.png" (10.52KB). Another section for ATTACHED TEXT contains detailed instructions for using CrashProbe, including steps for cloning the repository, integrating the SDK, and symbolicating crash reports.

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 Documentation⁷⁸** for **Android⁷⁹**, **iOS⁸⁰**, **React Native⁸¹**, **Xamarin⁸²**, **UWP⁸³** and **macOS⁸⁴**.

To view events before a crash, select a crash group, a specific device report, and then click on the events tab.

⁷⁸ <https://docs.microsoft.com/en-us/appcenter/sdk/index>

⁷⁹ <https://docs.microsoft.com/en-us/appcenter/sdk/analytics/android>

⁸⁰ <https://docs.microsoft.com/en-us/appcenter/sdk/analytics/ios>

⁸¹ <https://docs.microsoft.com/en-us/appcenter/sdk/analytics/react-native>

⁸² <https://docs.microsoft.com/en-us/appcenter/sdk/analytics/xamarin>

⁸³ <https://docs.microsoft.com/en-us/appcenter/sdk/getting-started/uwp>

⁸⁴ <https://docs.microsoft.com/en-us/appcenter/sdk/analytics/macos>

The screenshot shows the Microsoft App Center interface. On the left, there's a sidebar with various navigation options like CrashProbe, Getting Started, Build, Test, Distribute, Diagnostics (Crashes, Symbols), Analytics, Push, and Settings. The main area is titled 'Crash Group #286' and shows a list of devices and their crash details. At the top right, it says 'iPhone X (A1901) iOS 11.3.1 (15E302)' with 'VERSION 5.0.0 (80)' and 'OCCURRENCE Jun 19, 6:56 AM'. There are 'Download' and 'X' buttons. Below this, there are tabs for 'OVERVIEW', 'THREADS', 'EVENTS' (which is selected), and 'ATTACHMENTS'. Under 'EVENTS', it says '5 EVENTS' and lists several crash events with their timestamps: 'App crashed' (12 sec), 'Trigger Crash' (11 sec), 'Page Execute a privileged instruction' (10 sec), 'Main Page' (5 sec), 'Main Page' (0 sec), and 'Session started' (0 sec). A search bar is also present.

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](#)⁸⁵ 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:

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.

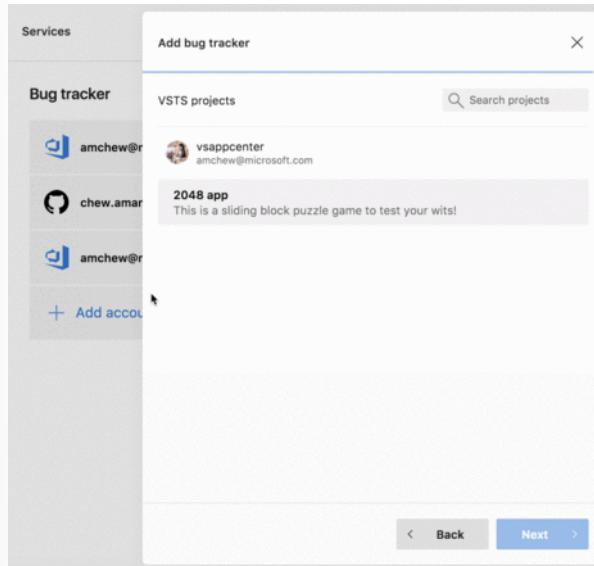
⁸⁵ <https://docs.microsoft.com/en-us/appcenter/dashboard/bugtracker/index>

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 API⁸⁶** for additional information.



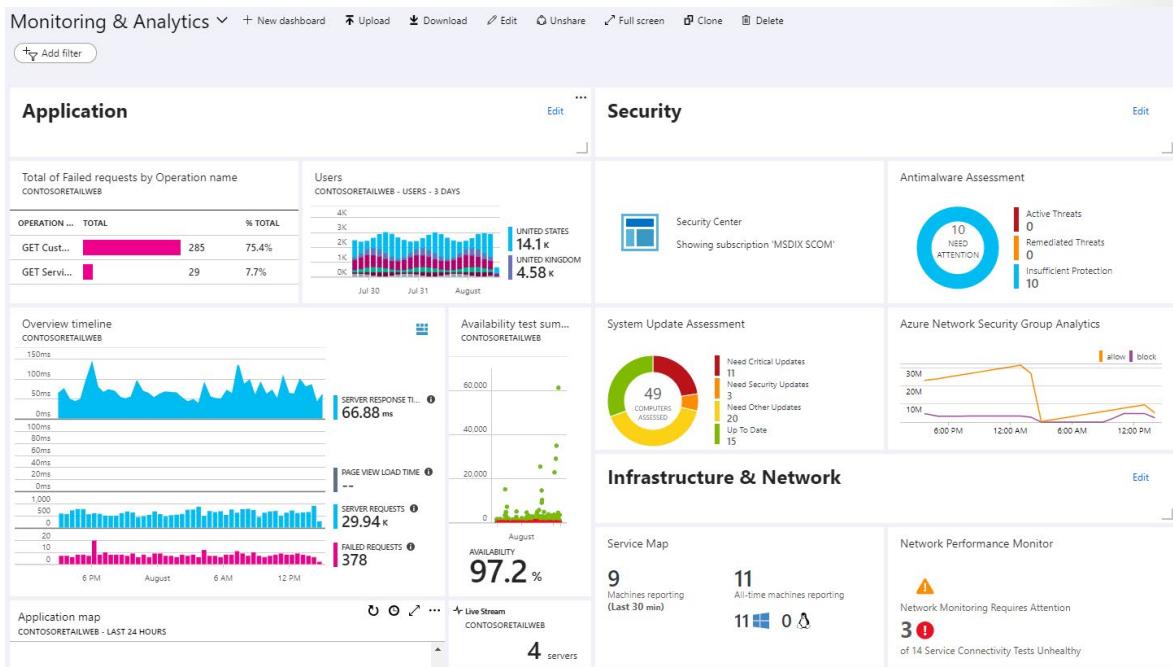
⁸⁶ <https://docs.microsoft.com/en-us/rest/api/vsts/wit/work%20item%20types>

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 dashboards⁸⁷ 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 explorer**⁸⁸, **Log Analytics queries**⁸⁹, and **maps**⁹⁰ and **availability**⁹¹ in Application Insights.
- Option for personal or shared dashboards. Integrated with Azure **role-based authentication (RBAC)**⁹².
- Automatic refresh. Metrics refresh depends on time range with minimum of five minutes. Logs refresh at one minute.

⁸⁷ <https://docs.microsoft.com/en-us/azure/azure-portal/azure-portal-dashboards>

⁸⁸ <https://docs.microsoft.com/en-us/azure/azure-monitor/platform/metrics-charts>

⁸⁹ <https://docs.microsoft.com/en-us/azure/azure-monitor/log-query/log-query-overview>

⁹⁰ <https://docs.microsoft.com/en-us/azure/azure-monitor/app/app-map>

⁹¹ <https://docs.microsoft.com/en-us/azure/azure-monitor/visualizations?toc=/azure/azure-monitor/toc.json>

⁹² <https://docs.microsoft.com/en-us/azure/role-based-access-control/overview>

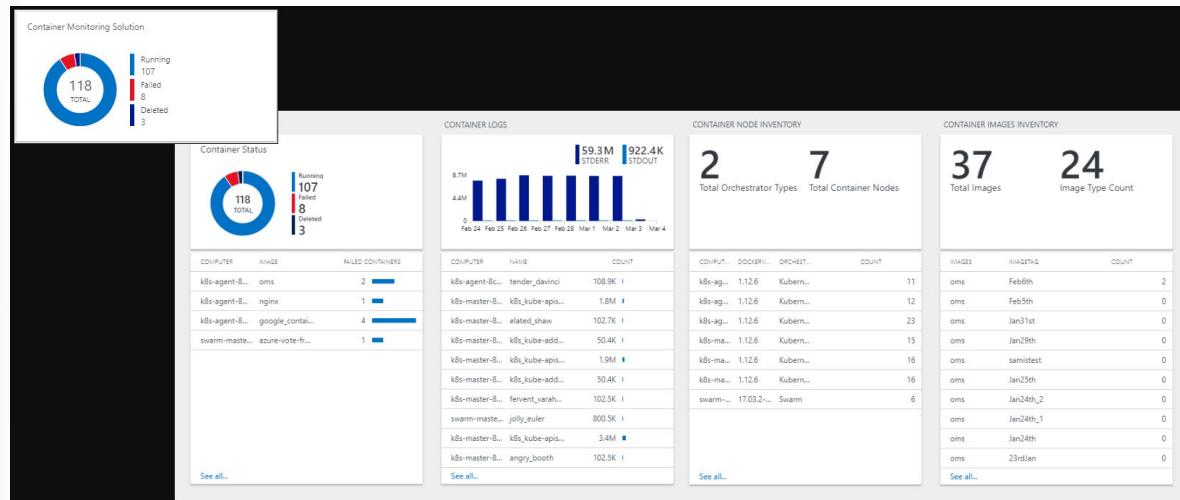
- Parametrized metrics dashboards with timestamp and custom parameters.
- Flexible layout options.
- Full screen mode.

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 Monitor⁹³ allow you to create custom visualizations with log data. They are used by **monitoring solutions**⁹⁴ 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 Analytic management model with workspaces and monitoring solutions.
- **Filters**⁹⁵ for custom parameters.
- Interactive, supports multi-level drill-in (view that drills into another view)

⁹³ <https://docs.microsoft.com/en-us/azure/log-analytics/log-analytics-view-designer>

⁹⁴ <https://docs.microsoft.com/en-us/azure/azure-monitor/insights/solutions>

⁹⁵ <https://docs.microsoft.com/en-us/azure/azure-monitor/platform/view-designer-filters>

Limitations

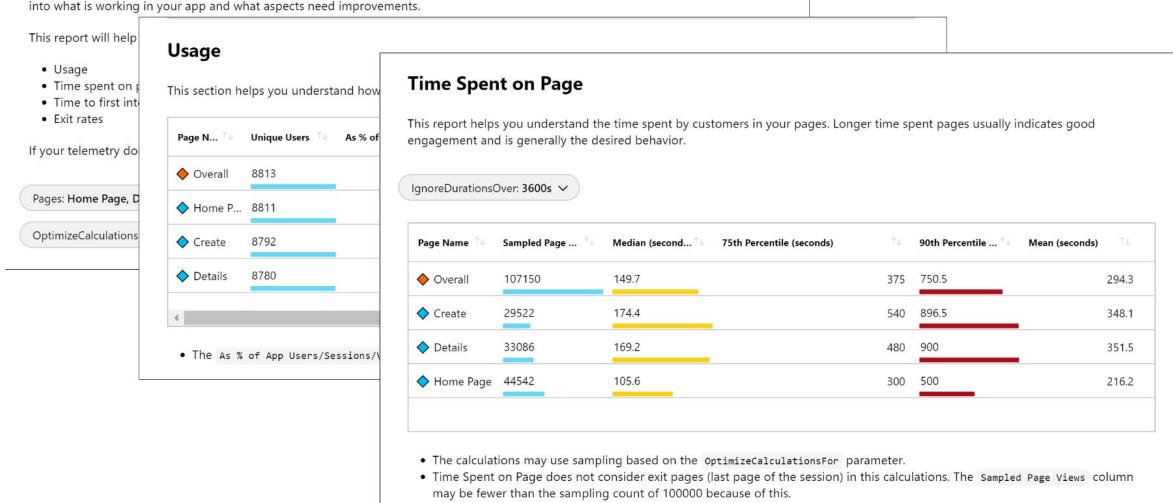
- Supports logs but not metrics.
- No personal views. Available to all users with access to the workspace.
- No automatic refresh.
- Limited layout options.
- No support for querying across multiple workspaces or Application Insights applications.
- Queries are limited in response size to 8MB and query execution time of 110 seconds.

Azure Monitor workbooks

Workbooks⁹⁶ 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.

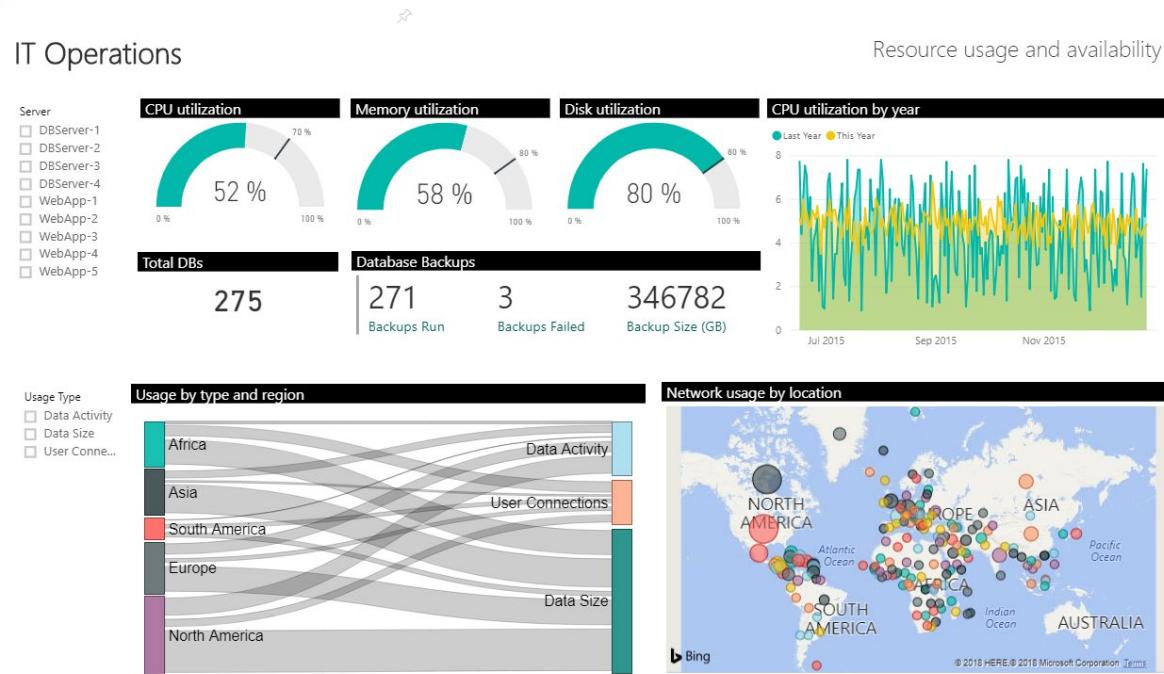
⁹⁶ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-usage-workbooks>

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 BI⁹⁷ 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 query⁹⁸ 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 RM integration. Can't manage dashboards and models through Azure Resource Manager

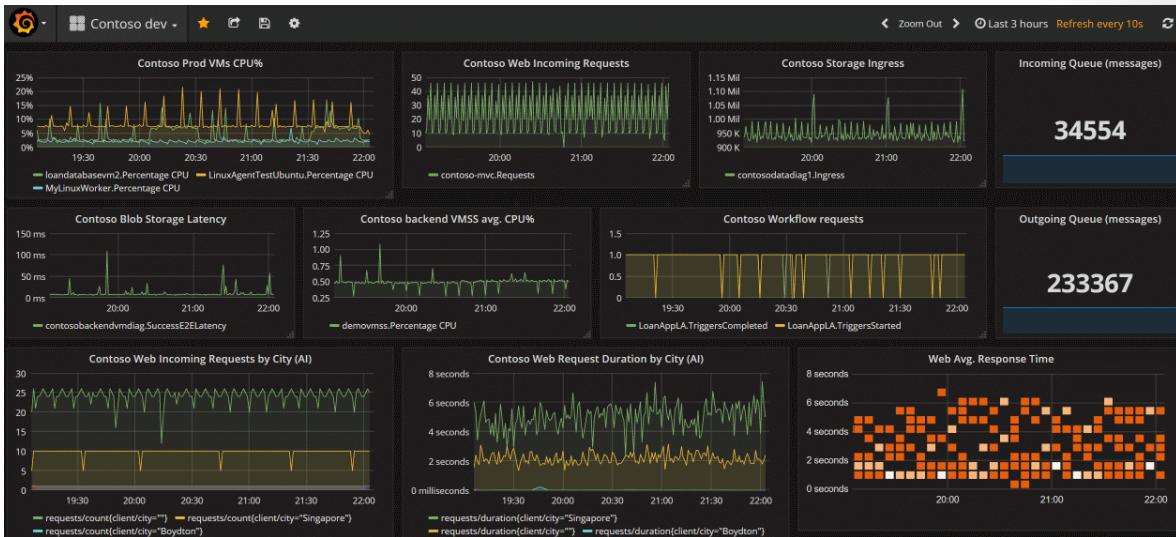
⁹⁷ <https://powerbi.microsoft.com/documentation/powerbi-service-get-started/>

⁹⁸ <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 for Pro licenses (currently 48 for Premium)

Grafana

Grafana⁹⁹ 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**¹⁰⁰ to your Azure subscription to have it visualize your Azure metrics data.



Advantages

- Rich visualizations
- Rich ecosystem of data sources
- 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.

⁹⁹ <https://grafana.com/>

¹⁰⁰ <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 set up ITSMC by following this overview:

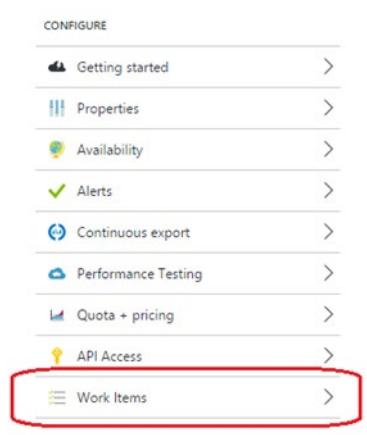
[ITSMC Overview¹⁰¹](#)

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).

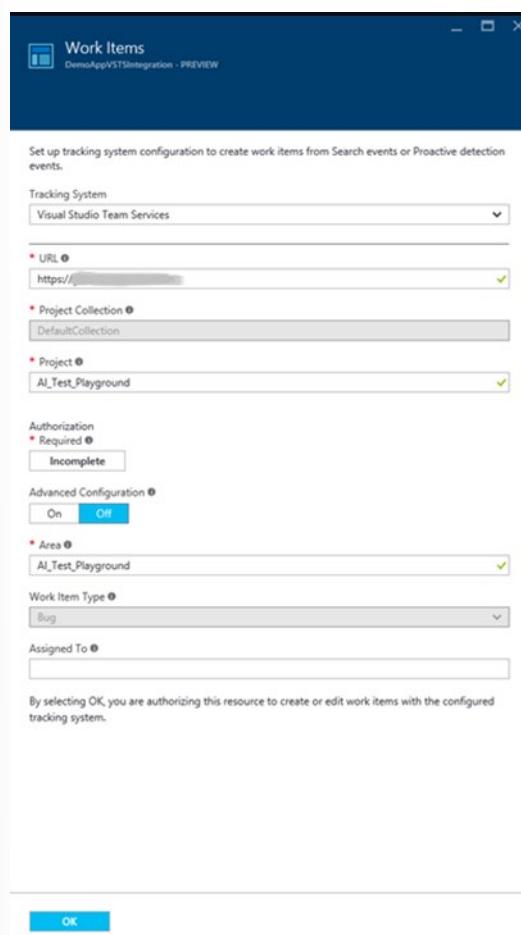
Configuring work item integration

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."



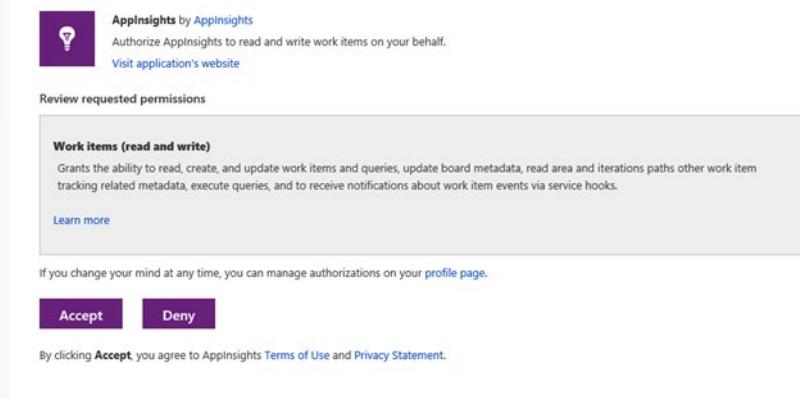
Click on this, and the configuration blade for work items will open. All you need to do is fill out the information about the VSTS system to which you want to connect, along with the project where you want to write your work items:

¹⁰¹ <https://docs.microsoft.com/en-us/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

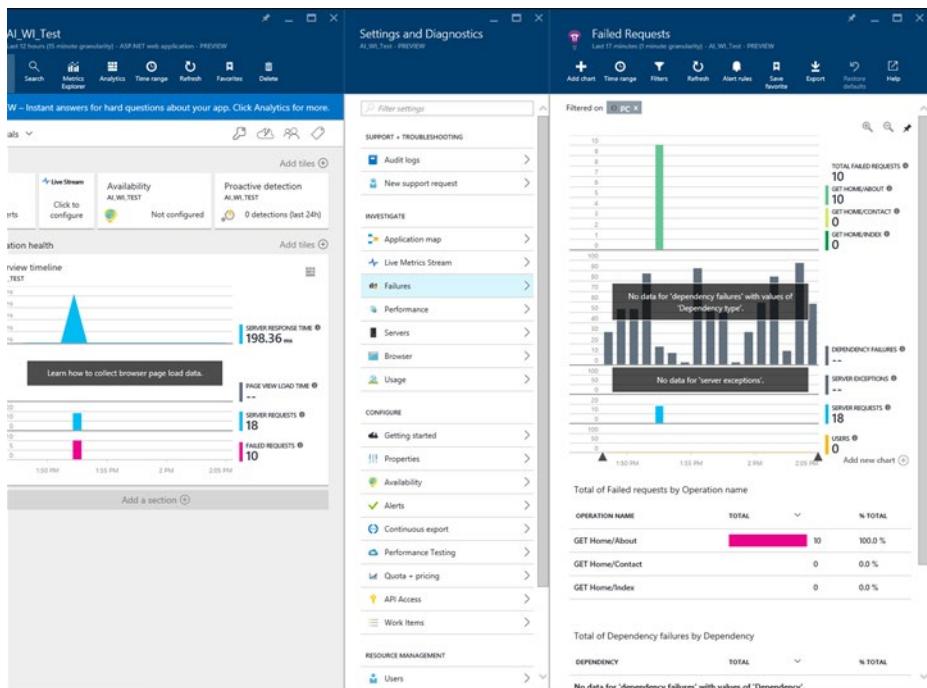


Once you've completed the authorization process, you can set defaults for "area path" and "assigned to." Only area path is required (if you haven't set up specific area paths in your project, that's ok. Just use the name of the project, as it is the top-level area path. Click OK, and assuming you've 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!

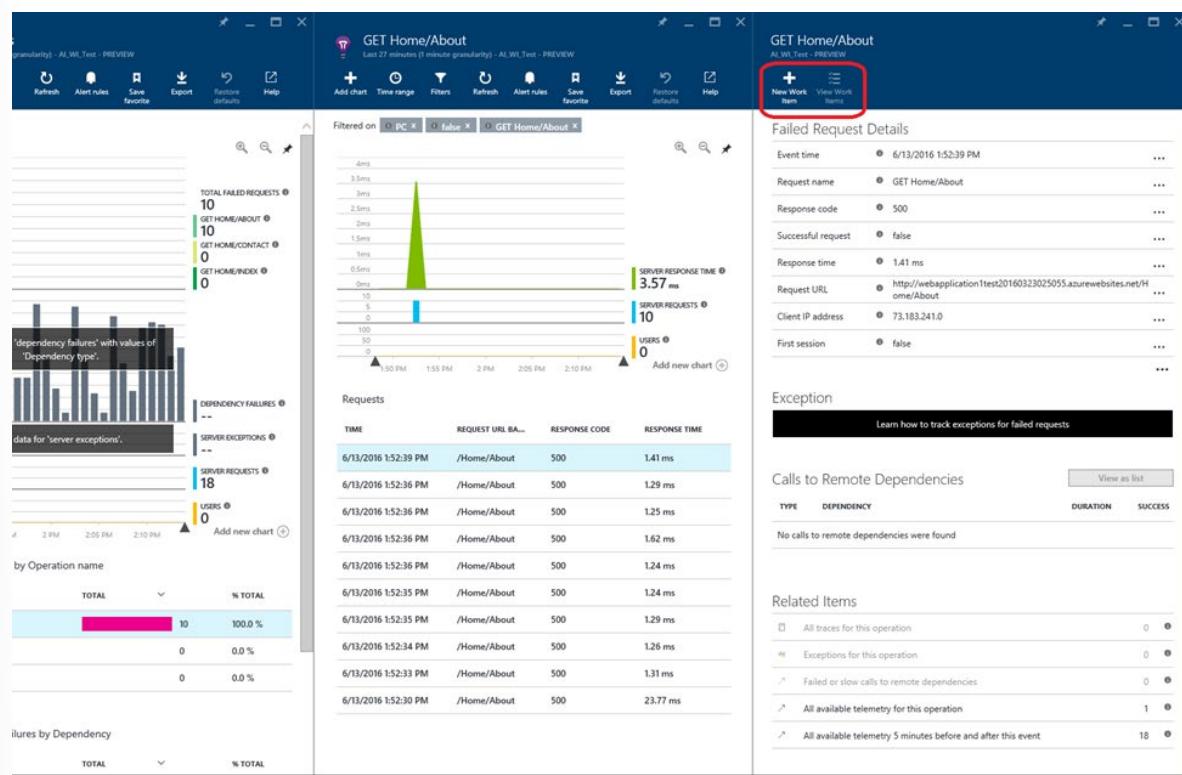
Creating work items

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 several 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 the Application Insights AI_WI_Test - PREVIEW interface. It includes sections for Failed Request Details, Exception, Calls to Remote Dependencies, and Related Items. On the right, a New Work Item dialog is open, showing fields for Title, Area, Work Item Type, Assigned To, and Details (which contains captured traffic information like IP address and device type). An OK button is at the bottom of the dialog.

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 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 side-by-side blades in VSTS. The left blade, titled 'GET Home/About AI_WI_Test - PREVIEW', displays 'Failed Request Details' for a failed request. It lists various details such as Event time (6/13/2016 1:52:39 PM), Request name (GET Home/About), Response code (500), and Client IP address (73.183.241.0). The right blade, also titled 'AI_WI_Test - PREVIEW', is titled 'View Work Items' and shows a list of work items with IDs 21 and 22, both titled 'Issue with GET Home/About'.

If you click the link for the work item that you want to view, it will open in VSTS:

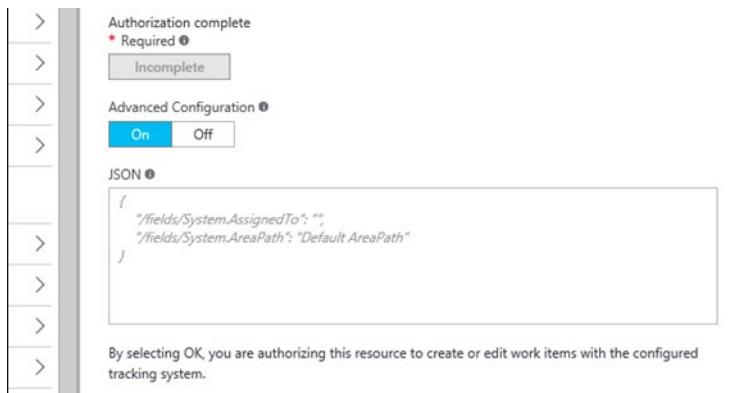
The screenshot shows the 'Queries' blade in VSTS with a query named 'Bug 21'. The results show a single work item titled 'BUG 21 21 Issue with GET Home/About'. The work item details are as follows:

| Area | Iteration | Details |
|---|---|--|
| AI_Test_Playground | AI_Test_Playground | Updated by Michael Gresley
2 minutes ago |
| Repro Steps | Status | Details |
| Authenticated or anonymous traffic: Anonymous user traffic
City: Houston
Client IP address: 73.183.241.0
Continent: North America
Country or region: United States
Device type: PC
Event time: 6/13/2016 1:52:39 PM
First session: false
Operation Id: T4mUOHAHM/0=
Operation name: GET Home/About
Real or synthetic traffic: Real user traffic
Request count: 1 | New
Reason: New defect reported
Build: Found in Build
System Info: Integrated in Build | Priority: 2
Severity: 3 - Medium
Effort: Remaining Work
Activity: |

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 the settings/mappings that you might need to handle modifications to your VSTS project. Some sample text is filled in for you. Soon, 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 kinds 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 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 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 (like 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

Lab overview

Application Insights is an extensible Application Performance Management (APM) service for web developers on multiple platforms. You can use it to monitor your live web applications. It automatically detects performance anomalies, includes powerful analytics tools to help you diagnose issues, and helps 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 with connection points available in a variety of development tools. It also allows you to monitor and analyze telemetry from mobile apps through integration with Visual Studio App Center.

In this lab, you'll learn about how you can add Application Insights to an existing web application, as well as how to monitor the application via the Azure portal.

Objectives

After you complete this lab, you will be able to:

- Deploy Azure App Service web apps
- Generate and monitor Azure web app application traffic by using Application Insights
- Investigate Azure web app performance by using Application Insights
- Track Azure web app usage by using Application Insights
- Create Azure web app alerts by using Application Insights

Lab duration

- Estimated time: **60 minutes**

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions¹⁰²

¹⁰² <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/>

Module Review and Takeaways

Module review questions

Review Question 1

Does Azure Monitor allow you to create alerts from log queries?

- yes
- no

Suggested answer

Review Question 2

What features are provided by Azure Monitor?

Review Question 3

What query language can you use to query Azure Log Analytics?

Review Question 4

What platform integration does Azure Monitor provide to visualize your logs in real time?

Answers

Review Question 1

Does Azure Monitor allow you to create alerts from log queries?

- 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 18 Implementing System Feedback Mechanisms

Module overview

Module overview

The 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:

- Define site reliability engineering
- Design processes to measure end-user satisfaction and analyze user feedback
- Design processes to automate application analytics
- Manage alerts and reduce meaningless and non-actionable alerts
- Carry out blameless retrospectives and create a just culture

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.

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 significantly 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 versus 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 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 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 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.

Design practices to measure end-user satisfaction

Capturing 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 happening; Measuring customer satisfaction doesn't have to be complicated or expensive. In fact, it's 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 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 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.

Capturing feedback in product

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 can access it 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. Despite 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 my feedback in one place. | |

Feedback from 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 [here¹](#).

| 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. |
| 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 acting on their suggestions. | |

¹ https://dev.azure.com/mseng/Azure%20DevOps%20Roadmap/_workitems/recentlyupdated

Feature request board

A massive part of build a product is identifying new features that customers desire. What is 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** ².

² <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:

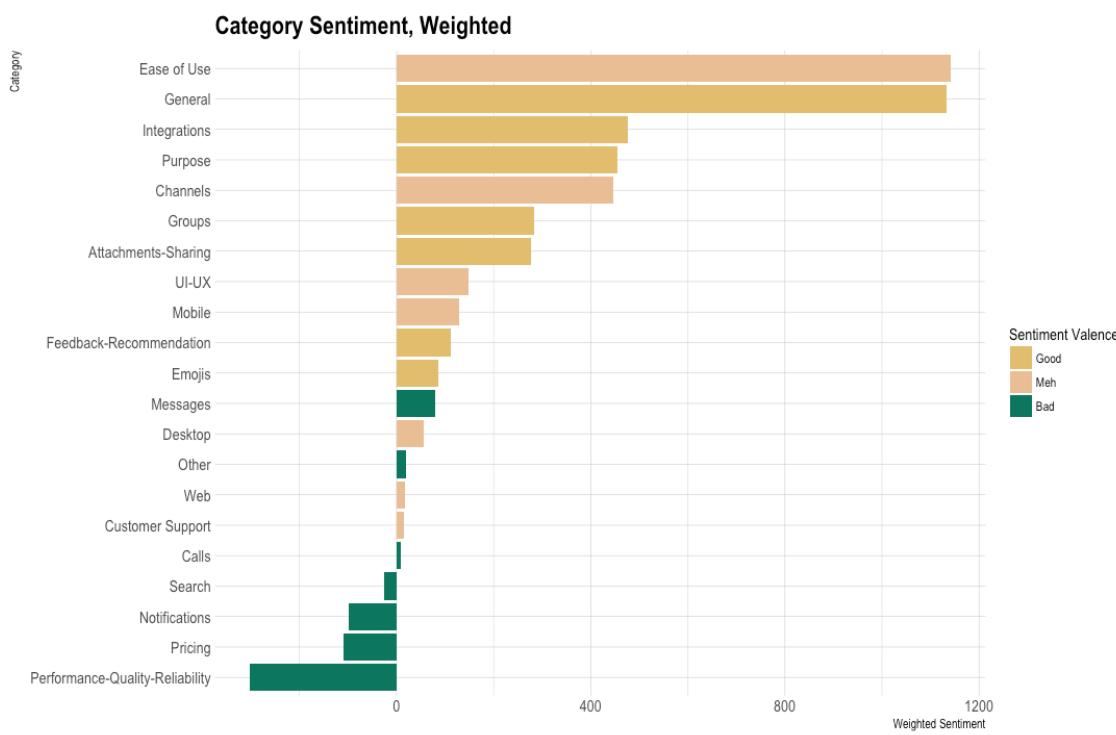
| Pros | Cons |
|--|--|
| No burden on the customer to complete a survey. In their natural environment. | Difficult to measure and quantify which makes it nearly impossible to track performance over time. |
| Get a true measure of what customers think about you as this method is entirely organic. | Challenging 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 must 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.

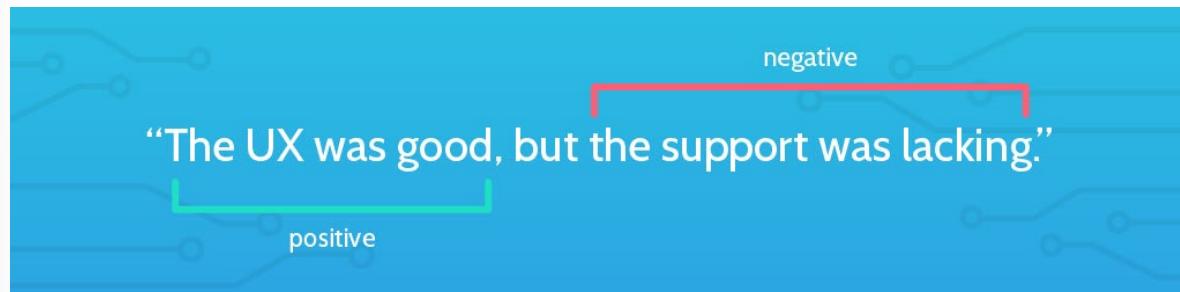
Startups thrive on feedback as their 'North star' and are constantly evolving based on what their customers request, break, and complain about. Enterprises also can't overlook the fact that customers are what make any company tick and must struggle harder than startups to stay relevant and innovate.

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 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 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.

Twitter sentiment release gate

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.

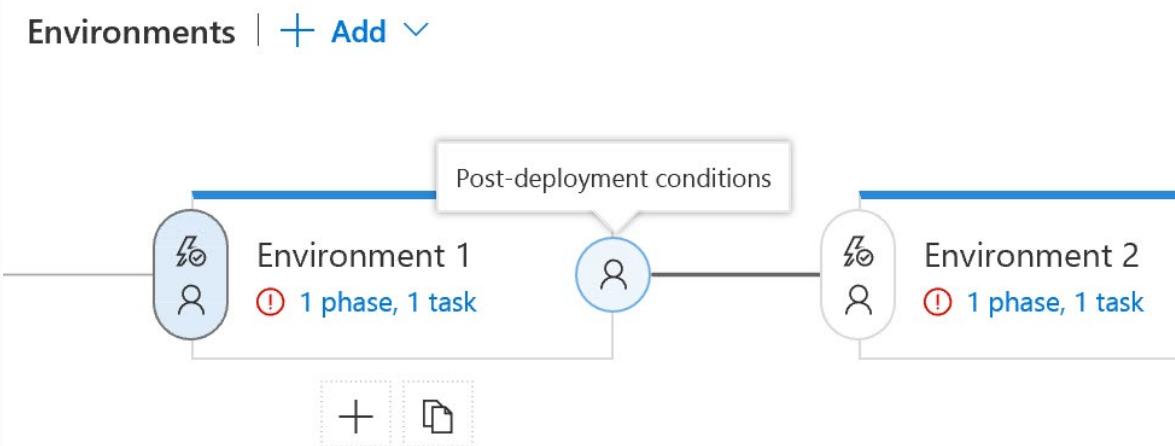
However, one of the things you’ll quickly learn is that no amount of monitoring will catch every single problem and, particularly, if you have a popular application, your users will know within seconds and turn very quickly to Twitter to start asking about the problem. Twitter can be a wonderful “alert” to let you know something is wrong with your app.

The Twitter sentiment release gate that can be downloaded from the [Visual Studio Marketplace³](#) 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.

³ <https://marketplace.visualstudio.com/items?itemName=ms-devlabs.vss-services-twittersentimentanalysis>



Then enable the release gates.

Gates* ^ Enabled

Define gates to evaluate before the deployment. [Learn more](#)

Then choose the Twitter Sentiment item and configure it.

--

Pre-deployment approvals
Select the users who can approve your releases.

Gates* ^
Define gates to evaluate before the deployment.

Delay before evaluation *
15

Azure Monitor
Observe the configured Azure monitor rules for active alerts.

Get Twitter Sentiment
Gate your releases based on average sentiment of tweets for a hashtag.

Invoke Azure Function
Invoke Azure Function as a part of your process.

Invoke REST API
Invoke REST API as a part of your process.

Query Work Items
Executes a work item query and checks for the number of items returned.

Deployment gates + Add

You can read up more about release gates at [Release deployment control using gates⁴](#).

⁴ <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/approvals/gates?view=vsts>

Demonstration: Deploy with release management green light capabilities

A release management (RM) process comprises of more than just deployment. Therefore, a release pipeline needs more than just steps for application deployment. There is 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.

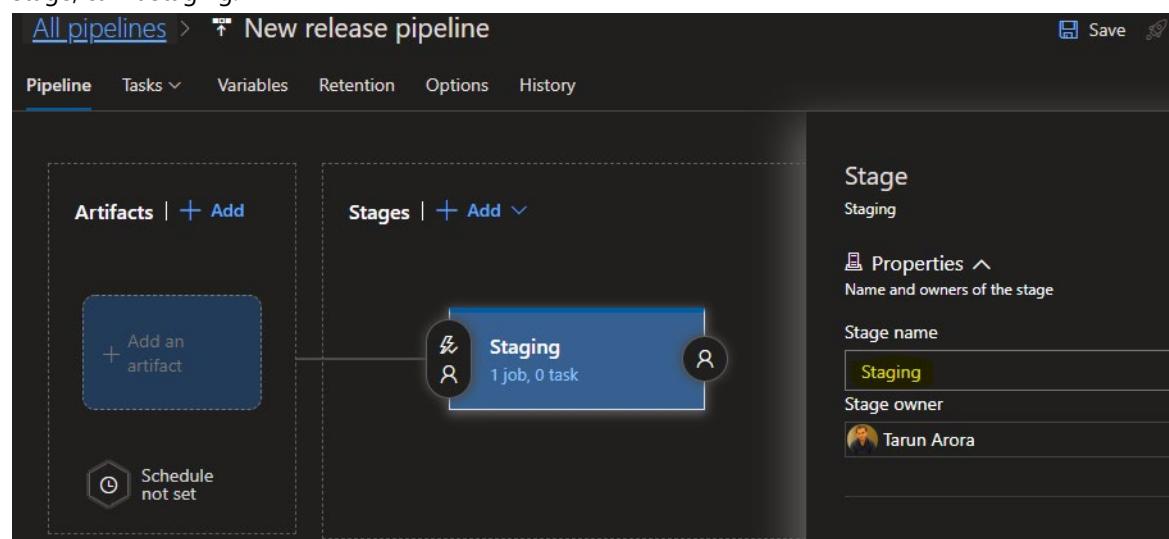
| Type | Title | Description | Assigned To |
|------------|---|---|----------------|
| Work Items | PR banner wording with compliance | An issue in staging after recreating the auth workflow f... | Anuradha Arora |
| Work Items | Create the issue in staging after recreating the auth workflow f... | Anuradha Arora | |
| Backlogs | ture flag to test workflow with customers | Tarun Arora | |
| Sprints | ferral workflow | Tarun Arora | |
| Plans | d for review | Tarun Arora | |

2. Click to create a new Query, add filter to return work item types of Bug of status New with priority 1

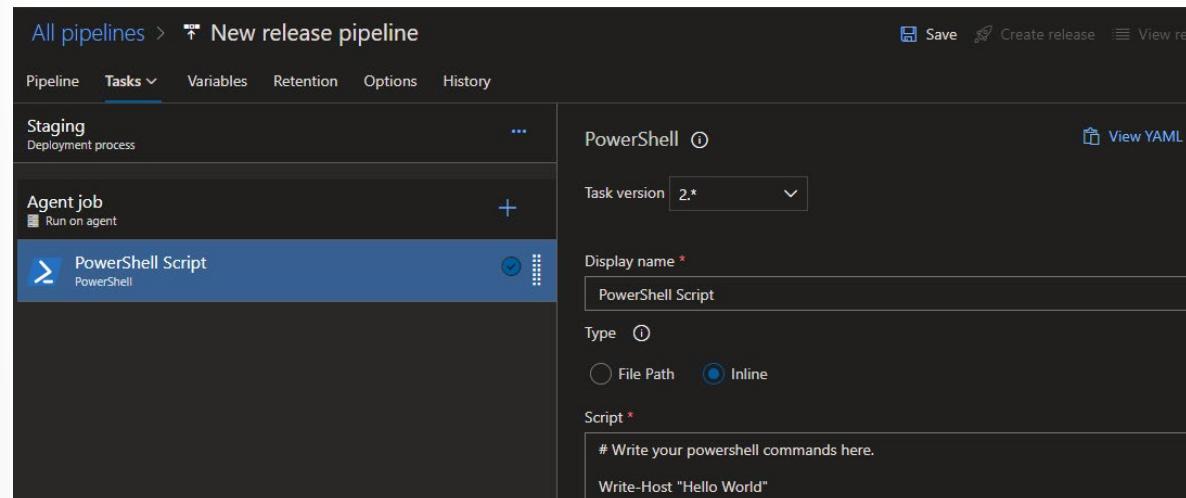
| Field | Operator | Value |
|----------------|----------|--------------|
| Work Item Type | = | Bug |
| State | = | New |
| Priority | = | 1 |
| Severity | = | 1 - Critical |

and severity Critical. Save the query in shared queries as Untriaged Critical Bugs

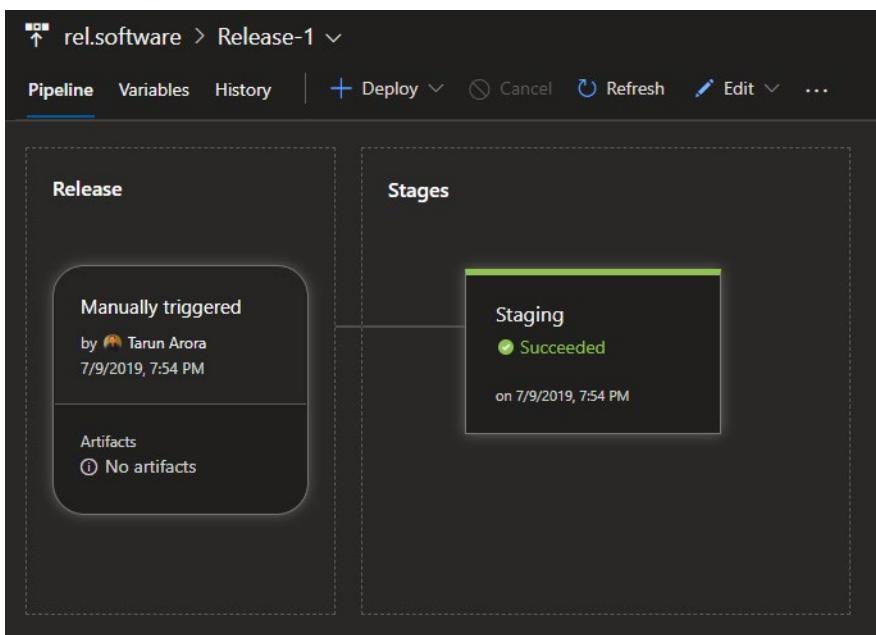
3. In the same team project, navigate to Azure Pipelines. Create an empty release pipeline with one stage, call it staging.



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.

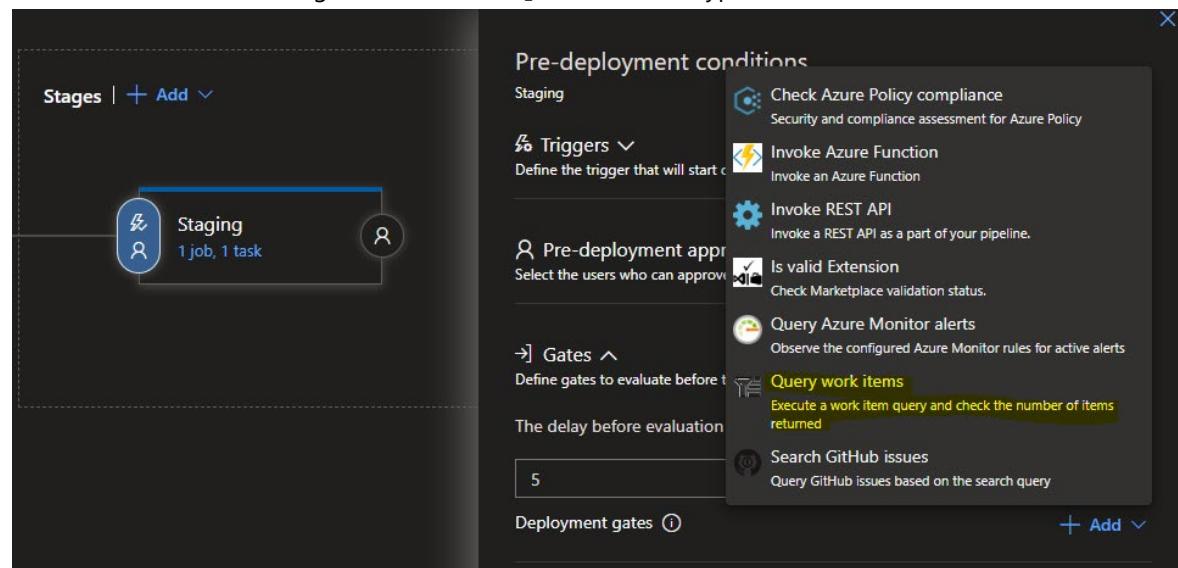


How to do it

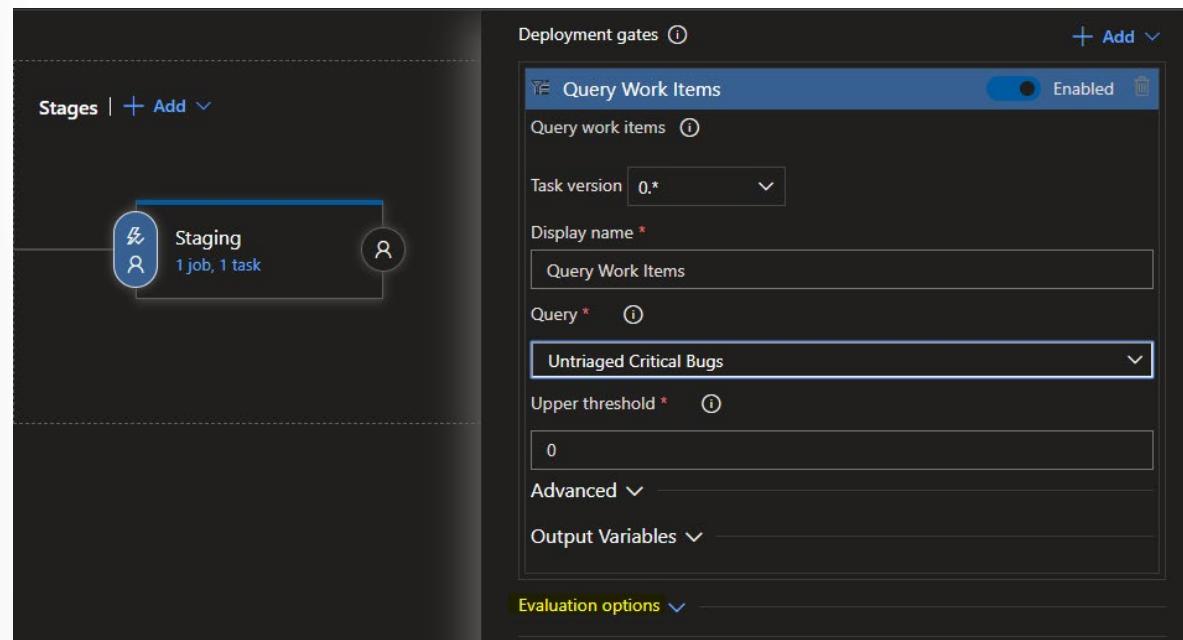
1. Edit the release pipeline, click on pre-deployment condition and from the right pane enable gates.

A screenshot of the Azure DevOps Pipeline editor. The left side shows the pipeline structure with 'Artifacts' and 'Stages'. The 'Staging' stage is selected. On the right, the 'Pre-deployment conditions' section is open. It includes a 'Triggers' section with a dropdown menu, a 'Pre-deployment approvals' section with a toggle switch set to 'Disabled', and a 'Gates' section with a toggle switch set to '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.

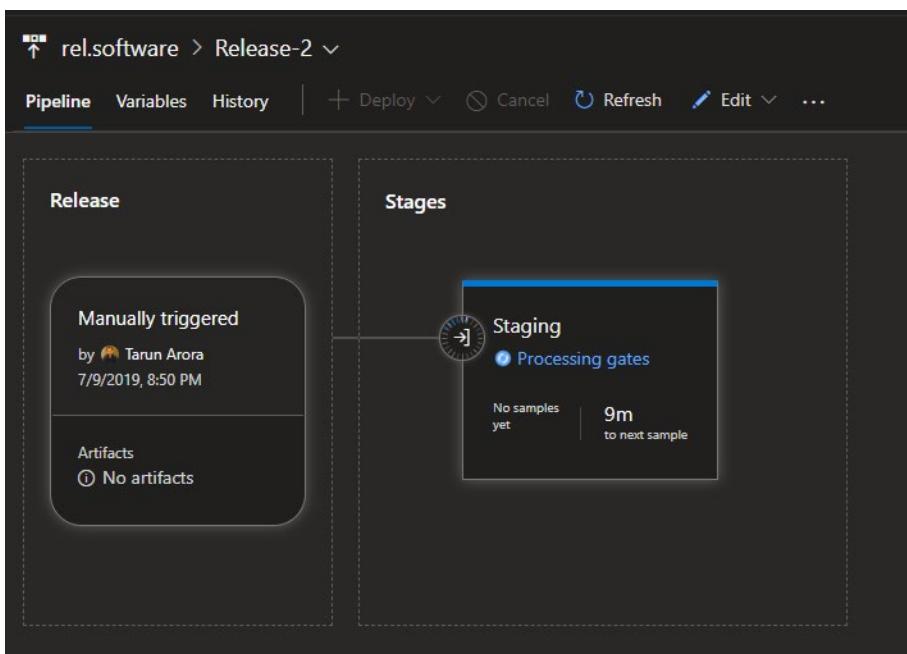
2. Click on Add to add a new gate, choose Query Work Item type Gate.



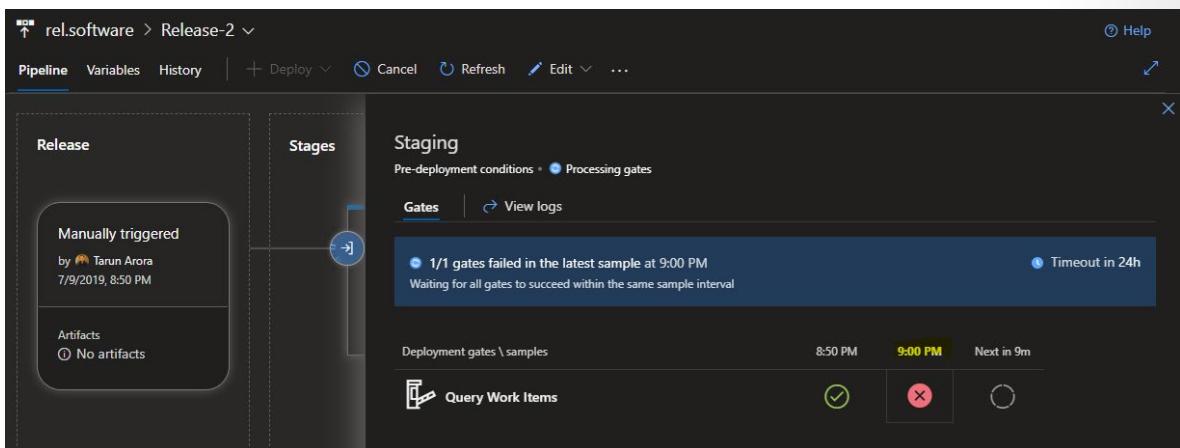
3. From the query work item drop down select the shared query 'untriaged critical bugs' created earlier and set the upper threshold to 0. Set the delay between evaluation period to 1 minute. Expand the evaluation options section and ensure that the time between evaluation of gates is set to 10 minutes and the Minimum duration for steady results results after a successful gates evaluation is configured to 15 minutes.



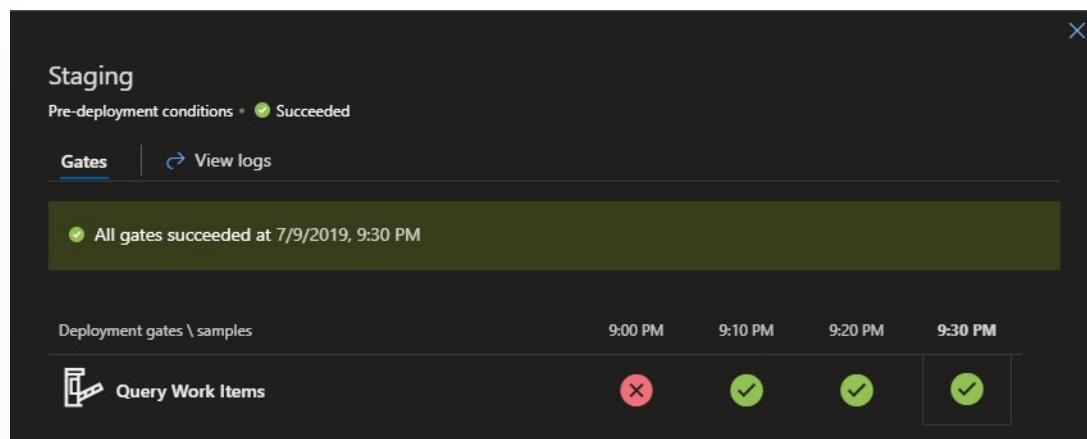
4. Save and queue a release. You'll see that the gate gets evaluated right away.



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" and the status is "Sample at 9:00 PM - 1/1 gates Failed". The log shows a JSON configuration for querying work items, including fields for Work Item Type, Title, Assigned To, State, and Tags. It also lists a single work item with ID 1418. A note at the bottom states: "Evaluation of expression 'xor(and(or(eq(root['queryType'], 'oneHop'), eq(root['queryType'], 'tree'))), and(le(count(roo

- 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 [here⁵](#)

⁵ <https://www.visualstudiogeeks.com/DevOps/IntegratingServiceNowWithVstsReleaseManagementUsingDeploymentGate>

Design processes to automate application analytics

Rapid responses and augmented search

In an Agile environment, you may typically find multiple development teams that work simultaneously, introducing new code or code changes daily, 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 must 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 using 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 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 using 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-paced 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 make 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, acquaint-

ance 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 many 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 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.

Integrating telemetry

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.

Recommending 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-premises 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 must 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.

Managing alerts

When would I get a notification?

Application Insights⁶ 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**⁷, **Java**⁸, or Node.js, and in **web page code**⁹). It is active when your app generates enough telemetry.

When would I get a smart detection notification?¹⁰

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 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 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 have a problem?

No, a notification doesn't mean that your app 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:

⁶ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-overview>

⁷ <https://docs.microsoft.com/en-us/azure/azure-monitor/app/asp-net>

⁸ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-java-get-started>

⁹ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-javascript>

¹⁰ <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 (s) | 90th Percentile (s) |
|--------|-----------------------|---------------------|
| Mar 16 | 0.5 | 1.5 |
| Mar 17 | 0.5 | 1.2 |
| Mar 18 | 0.5 | 1.5 |
| Mar 19 | 0.5 | 1.2 |
| Mar 20 | 0.5 | 1.3 |
| Mar 21 | 0.5 | 1.5 |
| Mar 22 | 1.6 | 2.8 |

Server requests

| Date | Server Requests (K) |
|--------|---------------------|
| Mar 16 | 2.0 |
| Mar 17 | 2.0 |
| Mar 18 | 2.0 |
| Mar 19 | 2.0 |
| Mar 20 | 2.0 |
| Mar 21 | 2.0 |
| Mar 22 | 2.0 |

Degradation in related dependency duration

| Date | Degradation (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**¹¹ data. If exceptions are thrown, you can also try the **snapshot debugger**¹².

¹¹ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-profiler>

¹² <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-snapshot-debugger>

Smart detection notifications

1. Smart detection notifications are enabled by default and sent to those who have **owners, contributors, and readers access to the Application Insights resource**¹³. To change this, either click Configure in the email notification, or open Smart Detection settings in Application Insights.

The screenshot shows the 'Smart Detection settings' page for the 'fabrikamprod' Application Insights resource. The left sidebar includes links for Overview, Activity log, Access control (IAM), Configure (Getting started and Properties), Alerts (selected), and Smart Detection settings (highlighted with a red box). The main content area displays a table of detected anomalies:

| 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.

Triage¹⁴

- 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 tests**¹⁵ including that region: there might simply be network issues in that area.

Diagnose slow page loads¹⁶

- 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

¹³ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-resources-roles-access-control>

¹⁴ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-proactive-performance-diagnostics#triage>

¹⁵ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-monitor-web-app-availability>

¹⁶ <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**¹⁷ to investigate response times.
- Set up **dependency tracking**¹⁸ 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**¹⁹ 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 pages²⁰

- 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 another 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.

Example: 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.

¹⁷ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-web-monitor-performance#metrics>

¹⁸ <https://docs.microsoft.com/en-us/azure/azure-monitor/app/asp-net-dependencies>

¹⁹ <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-monitor-web-app-availability>

²⁰ <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.

Reducing meaningless and non-actionable alerts

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 can't 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.

Blameless retrospectives and a just culture

Discussion: Blameless retrospective

What does it mean to have 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
- 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 must 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 acts 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 Hollnagel²¹** 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.

Developing a just culture

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, but 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, 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.

²¹ <http://www.erikhollnagel.com/>

- 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.
- Accept that the **Hindsight Bias**²² will continue to cloud our assessment of past events and work hard to eliminate it.
- Accept that the **Fundamental Attribution Error**²³ 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 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. To understand how failures occur, we first must 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 happened, treat the engineers involved with respect, and *learn* from the event.

For more information, see also:

- **Brian Harry's Blog - A good incident postmortem**²⁴

²² <http://en.wikipedia.org/wiki/Hindsight>

²³ http://en.wikipedia.org/wiki/Fundamental_attribution_error

²⁴ <https://blogs.msdn.microsoft.com/bharry/2018/03/02/a-good-incident-postmortem/>

Lab

Integration between Azure DevOps and Microsoft Teams

Lab overview

Microsoft Teams²⁵ is a hub for teamwork in Office 365. It allows you to manage and use all your team's chats, meetings, files, and apps together in one place. It provides software development teams with a hub for teams, conversations, content and tools from across Office 365 and Azure DevOps.

In this lab, you will implement integration scenarios between Azure DevOps services and Microsoft Teams.

Note: **Azure DevOps Services** integration with Microsoft Teams provides a comprehensive chat and collaborative experience across the development cycle. Teams can easily stay informed of important activities in your Azure DevOps team projects with notifications and alerts on work items, pull requests, code commits, as well as build and release events.

Objectives

After you complete this lab, you will be able to:

- Integrate Microsoft Teams with Azure DevOps
- Integrate Azure DevOps Kanban boards and Dashboards in Teams
- Integrate Azure Pipelines with Microsoft Teams
- Install the Azure Pipelines app in Microsoft Teams
- Subscribe for Azure Pipelines notifications

Lab duration

- Estimated time: **60 minutes**

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- **AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions**²⁶

²⁵ <https://teams.microsoft.com/start>

²⁶ <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/>

Module review and takeaways

Module review questions

Review Question 1

What are some of the ways to measure end user satisfaction for your product?

- CSAT
- CES
- STAR
- NPM

Review Question 2

True or False: Azure DevOps has a feature request board.

- True
- False

Review Question 3

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

Review Question 1

What are some of the ways to measure end user satisfaction for your product?

- CSAT
- CES
- STAR
- NPM

Review Question 2

True or False: Azure DevOps has a feature request board.

- True
- False

Review Question 3

True or False: Application Insights analyses the traffic from your website against historic trends and sends you smart detection notifications on degradation?

- True
- False

Module 19 Implementing Security in DevOps Projects

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 in the Pipeline

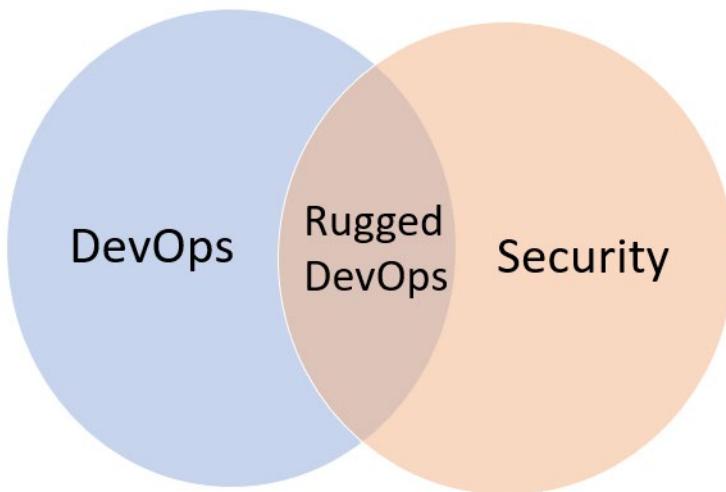
What is rugged DevOps ?

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.

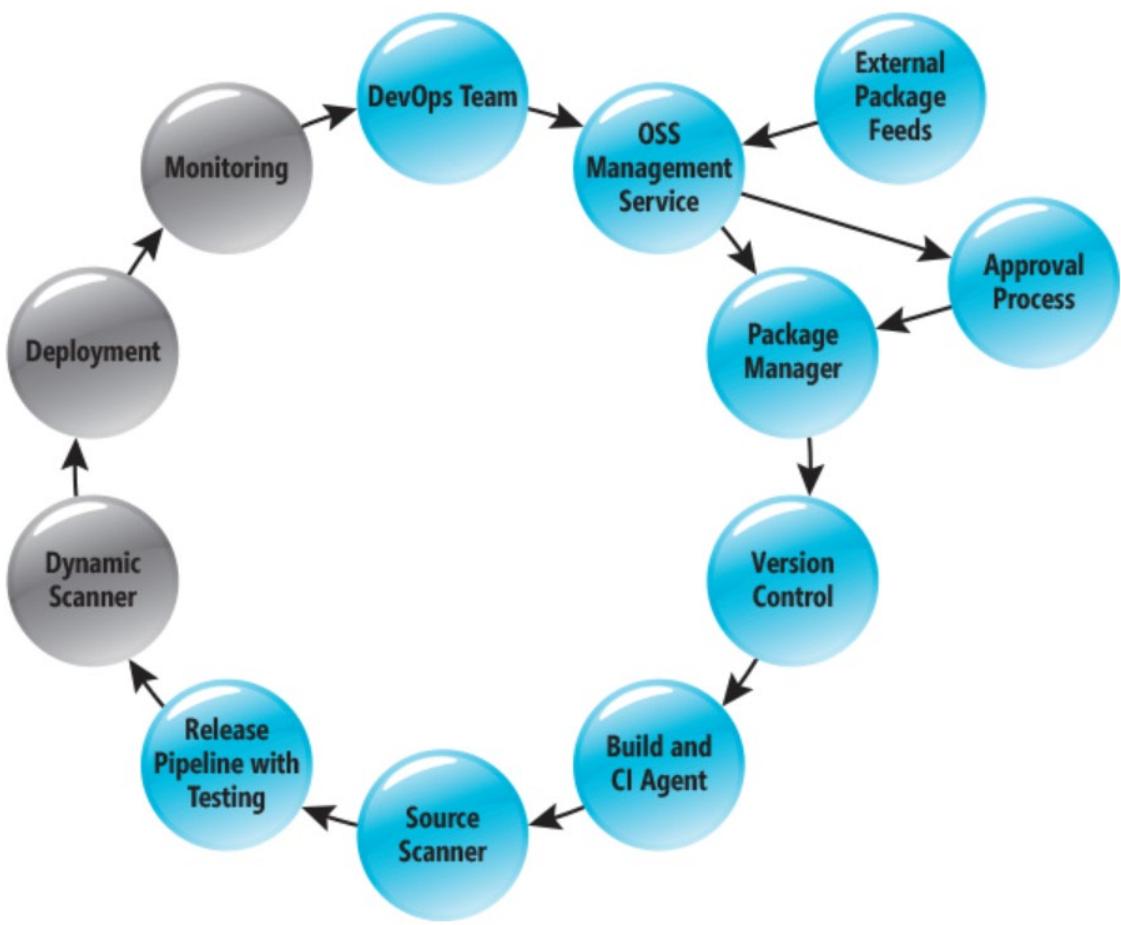
Rugged DevOps addresses broader questions, such as:

- 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. On the left, there is a sidebar with the following items:

- Overview
- Boards
- Repos
- Pipelines
- Test Plans

The "Artifacts" item is highlighted with a red box.

The main content area has a title "Connect to feed" and a list of supported feeds:

- NuGet
- npm
- Maven
- Gradle
- Universal
- Python

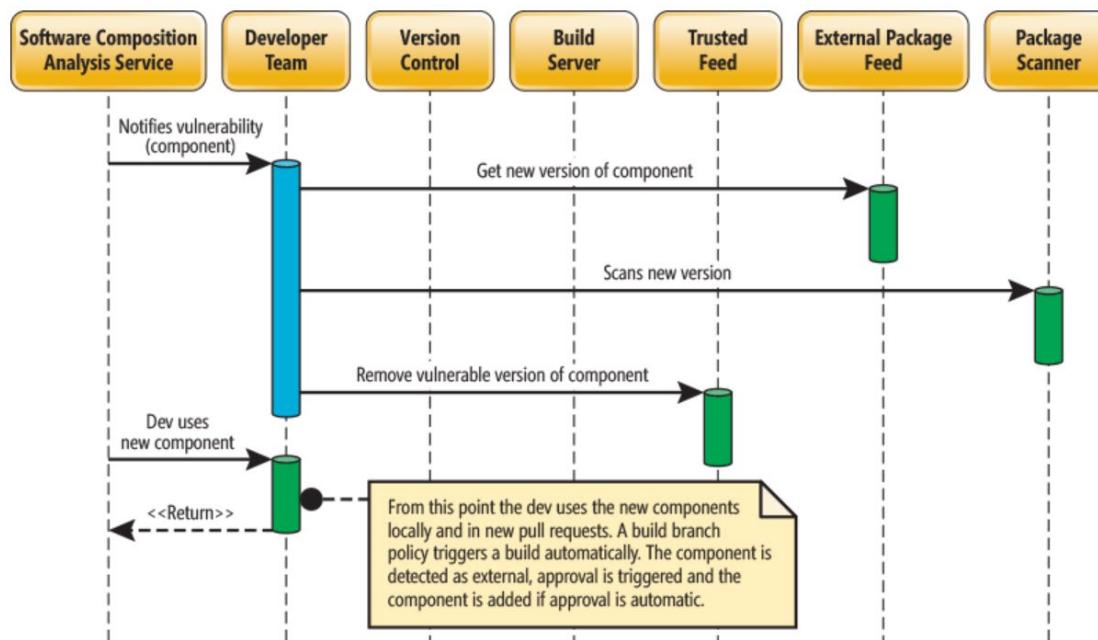
Note: After you publish a particular version of a package to a feed, that version number is permanently reserved. You cannot upload a newer revision package with that same version number or delete that version and upload a new package with the same version number. The published version is immutable.

The Role of OSS components

Development work is more productive because 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, 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 | |
| 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

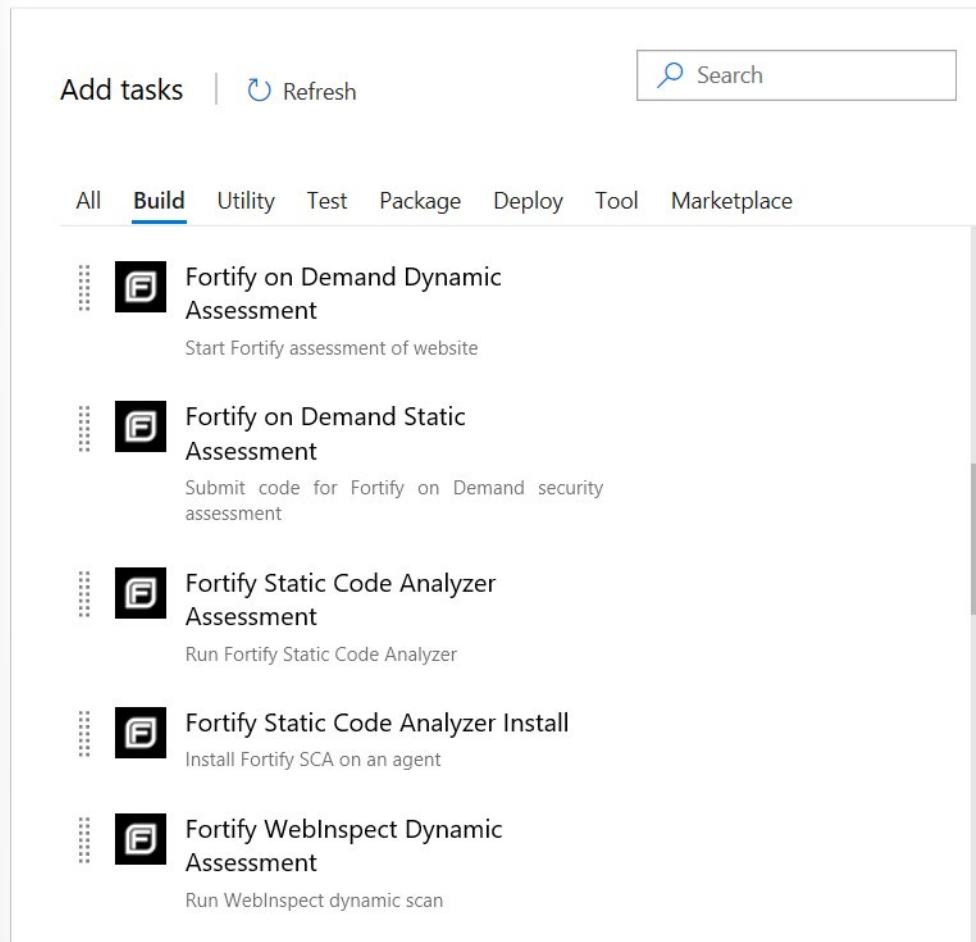
According to a company's policies, WhiteSource automatically approves, rejects, or triggers a manual approval process every time a new open-source component is added to a build. Developers can set up policies based on parameters such as security-vulnerability severity, license type, or library age. When 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 Pipelines

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.



The screenshot shows the Azure DevOps Marketplace interface. At the top, there are buttons for 'Add tasks' and 'Refresh', and a search bar. Below the search bar is a navigation bar with tabs: All, Build (which is selected), Utility, Test, Package, Deploy, Tool, and Marketplace. The main area displays a list of five build tasks provided by Micro Focus Fortify:

- Fortify on Demand Dynamic Assessment: Start Fortify assessment of website.
- Fortify on Demand Static Assessment: Submit code for Fortify on Demand security assessment.
- Fortify Static Code Analyzer Assessment: Run Fortify Static Code Analyzer.
- Fortify Static Code Analyzer Install: Install Fortify SCA on an agent.
- Fortify WebInspect Dynamic Assessment: Run WebInspect dynamic scan.

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.

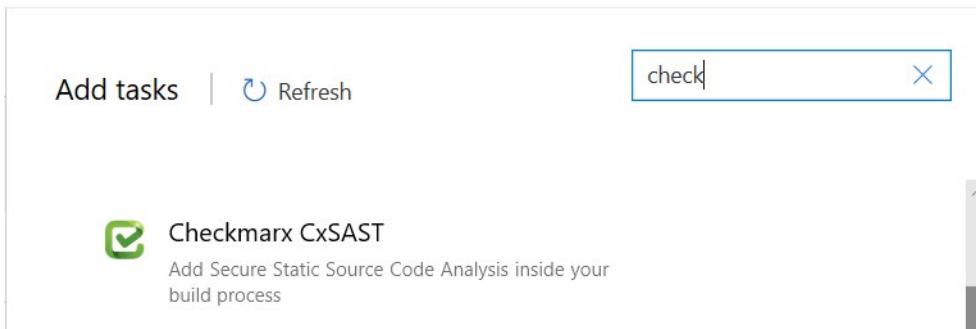
⁶ <https://marketplace.visualstudio.com/items?itemName=fortifyvsts.hpe-security-fortify-vsts>

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:

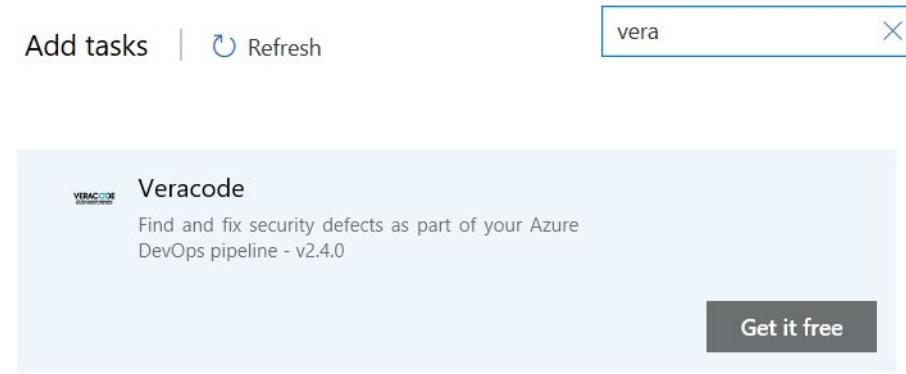
- Best fix location. Checkmarx highlights the best place to fix your code to minimize the time required to remediate the issue. A visual chart of the data flow graph indicates the ideal location in the code to address multiple vulnerabilities within the data flow using a single line of code.
- Quick 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 software composition analysis 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.

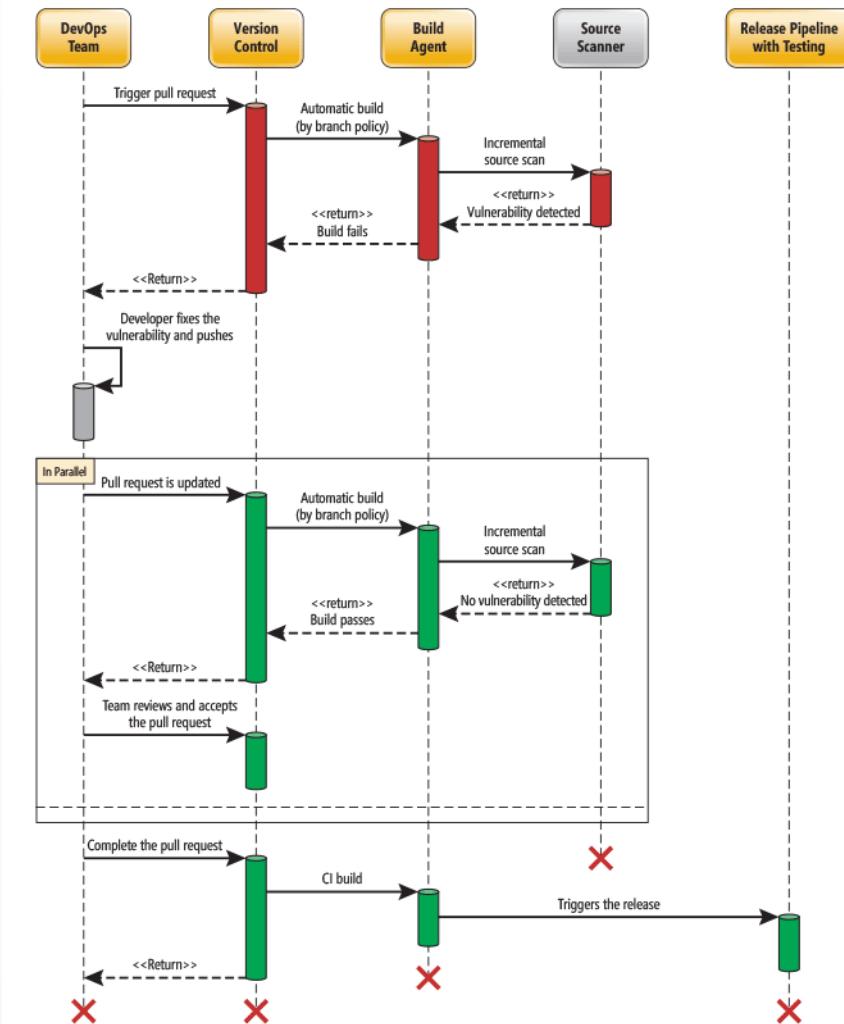
⁹ <https://www.whitesourcesoftware.com/>

¹⁰ <https://www.checkmarx.com/>

¹¹ <https://www.veracode.com/>

¹² <https://www.blackducksoftware.com/>

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. 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.



Implementing pipeline security

In addition to protecting your code, it's essential to protect credentials and secrets. 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

¹³ <http://aka.ms/jea>

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 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 Center¹⁵** 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 Azure¹⁶** (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. Like 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

¹⁴ <https://www.owasp.org>

¹⁵ <https://azure.microsoft.com/en-us/services/security-center/>

¹⁶ <https://github.com/azsk/DevOpsKit-docs>

security will be maintained despite changes to the state of your systems, by using a combination of tools such as automation runbooks and schedules.

- 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 can 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

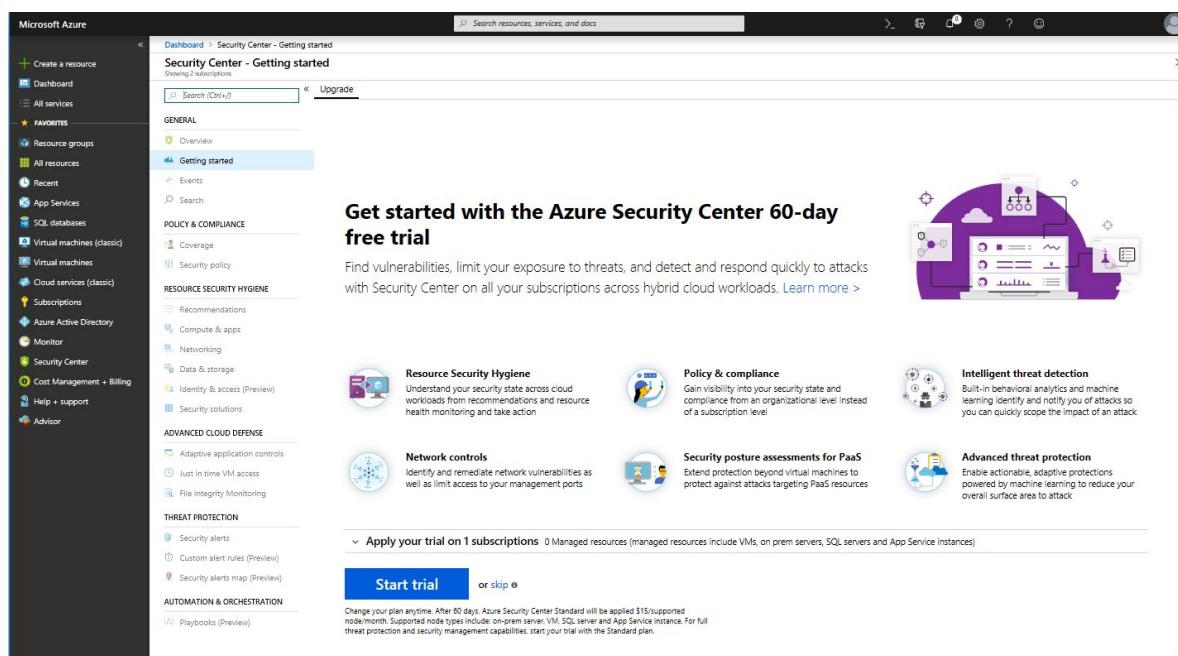
Azure Security Center supports both Windows and Linux operating systems. It can also provide security to features in both IaaS and platform as a service (PaaS) scenarios.

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 prominent "Get started with the Azure Security Center 60-day free trial" section. Below this, there are several cards describing different security features:

- Resource Security Hygiene**: Understand your security state across cloud workloads from recommendations and resource health monitoring and take action.
- Policy & compliance**: Gain visibility into your security state and compliance from an organizational level instead of a subscription level.
- Intelligent threat detection**: Built-in behavioral analytics and machine learning identify and notify you of attacks so you can quickly scope the impact of an attack.
- Network controls**: Identify and remediate network vulnerabilities as well as limit access to your management ports.
- Security posture assessments for PaaS**: Extend protection beyond virtual machines to protect against attacks targeting PaaS resources.
- Advanced threat protection**: Enable actionable, adaptive protections powered by machine learning to reduce your overall surface area to attack.

At the bottom, there's a "Start trial" button and a link to "Apply your trial on 1 subscriptions". A note at the bottom states: "Change your plan anytime. After 60 days, Azure Security Center Standard will be applied \$15-supported node/month. Supported node types include on-prem server, VM, SQL server and App Service instance. For full threat protection and security management capabilities, start your trial with the Standard plan."

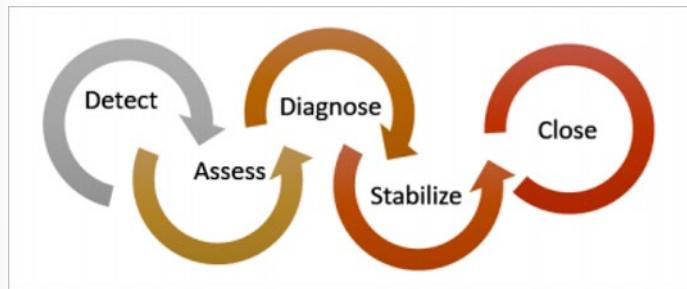
You can read more about Azure Security Center at [Azure Security Center¹⁸](https://azure.microsoft.com/en-us/services/security-center/).

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.
- 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.

¹⁸ <https://azure.microsoft.com/en-us/services/security-center/>

- 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",  
                        "in": ["West US", "East US"]  
                    }  
                }  
            }  
        }  
    }  
}
```

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 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](#).

Resource 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 changes²⁵](#) 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 develop-

²⁴ <https://docs.microsoft.com/en-us/azure/governance/policy/concepts/definition-structure>

²⁵ <https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-lock-resources>

ment 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.



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 Blueprints²⁶](#).

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 organi-

²⁶ <https://azure.microsoft.com/services/blueprints/>

zation. Azure ATP is capable of detecting known malicious attacks and techniques and can help you investigate security issues and network vulnerabilities.

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 into the Azure ATP portal at <https://portal.atp.azure.com>²⁷. 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.

²⁷ <https://portal.atp.azure.com>

The screenshot shows the Azure Advanced Threat Protection Timeline for the contoso-corp subscription. 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.

Cryptocurrency mining and other advanced attacks

Azure Defender for container registries can be enabled at the subscription level. Once it is enabled, Security Center will then scan images that are pushed to the registry, imported into the registry, or any images pulled within the last 30 days. There is a per-image charge for this feature.

When issues are found, a notification appears in the Security Center dashboard.

There are three triggers for image scans:

- On push (a new image is pushed to the registry)
- Recently pulled (any image pulled in the last 30 days)
- On import (when an image is imported from other locations like Docker Hub)

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 options²⁸** 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 Protection²⁹** webpage.

²⁸ <https://www.microsoft.com/en-ie/cloud-platform/enterprise-mobility-security-pricing>

²⁹ <https://azure.microsoft.com/en-us/features/azure-advanced-threat-protection/>

Lab

Implement security and compliance in Azure DevOps Pipelines

Lab overview

In this lab, we will create a new Azure DevOps project, populate the project repository with a sample application code, create a build pipeline. Next, we will install WhiteSource Bolt from the Azure DevOps Marketplace to make it available as a build task, activate it, add it to the build pipeline, use it to scan the project code for security vulnerabilities and licensing compliance issues, and finally view the resulting report.

Objectives

After you complete this lab, you will be able to:

- Create a Build pipeline
- Install WhiteSource Bolt from the Azure DevOps marketplace and activate it
- Add WhiteSource Bolt as a build task in a build pipeline
- Run build pipeline and view WhiteSource security and compliance report

Lab duration

- Estimated time: **60 minutes**

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- **AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions³⁰**

³⁰ <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/>

Module review and takeaways

Module review questions

Review Question 1

Rugged DevOps combines which two elements? Choose two.

- DevOps
- Cost management
- Microservice Architecture
- Security
- Hackathons

Review Question 2

Which term broadly defines what security means in rugged DevOps?

- Access control
- Application server hardening
- perimeter protection
- Securing the pipeline

Review Question 3

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

Review Question 4

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 criterion to use in your pipeline
- Analyzing open-source software after it has been deployed to production to identify security vulnerabilities

Review Question 5

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

Review Question 6

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

Review Question 7

Which Azure service from the below list is a monitoring service that can provide threat protection and security recommendations across all your services both in Azure and on-premises? <<

- Azure Policy
- Azure Security Center
- Azure Key vault
- Role-based access control

Review Question 8

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

Review Question 9

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

Review Question 1

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.

Review Question 2

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, 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)?

Review Question 3

What component in Azure DevOps can you use to store, organize, and share access to packages, and integrate those packages 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.

Review Question 4

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 criterion 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 criterion 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:

Review Question 5

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.

Review Question 6

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 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.

Review Question 7

Which Azure service from the below list is a monitoring service that can provide threat protection and security recommendations across all 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 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 your services both in Azure and on-premises.

Review Question 8

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:

Review Question 9

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 20 Validating Code Bases for Compliance

Module overview

Module overview

In the last module we saw how “Rugged” DevOps (or DevSecOps) has become a critical part of software development.

Open-source software is now also commonly used and as well as all the benefits it has brought, there are many potential downsides. These need to be managed.

Organizations will have security and compliance policies that need to be adhered to. This is best done by integrating license and vulnerability scans as part of the build and deployment processes.

Learning objectives

After completing this module, students will be able to:

- Describe the potential challenges with integrating open-source software
- Inspect open-source software packages for security and license compliance
- Manage organizational security and compliance policies
- Integrate license and vulnerability scans into build and deployment pipelines
- Configure build pipelines to access package security and license ratings

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.

Software today 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 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 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 website¹](http://www.dotnetfoundation.org).

Corporate concerns with open-source software components

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 built 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 must accept the risks involved with it. The concerns are that source code components:

- **Are 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 come 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 ones that carry out the actual changes.

Open-source licenses

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.

¹ <http://www.dotnetfoundation.org>

According to the open-source definition of OpenSource.org a license should not:

- Discriminate against persons or groups
- Discriminate against fields of endeavor
- Be specific to a product
- Restrict other software
- and more - See the **Open Source Definition²**

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.

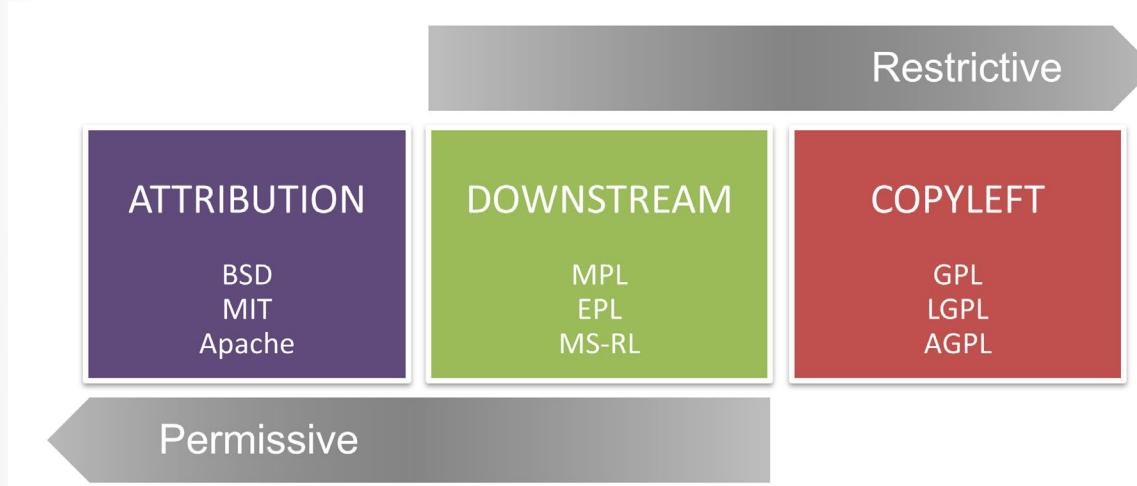
Since the source code is open-source, people with malicious intent can also inspect the code for vulnerabilities and exploit these when possible. In that regard, it is both a blessing and a curse that open-source software has source code available for others.

Common open-source licenses

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

² <http://opensource.org/osd>

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.

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 ratings

Any form of software that uses code or components must consider 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, like 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.

Managing security and compliance 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)**³ 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 can make informed decisions. They offer impartial and practical advice.

³ <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**⁴. It includes vulnerabilities from all the **OWASP Top 10**⁵.

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**⁶, 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**⁷
- **OWASP Secure Coding Practices Quick Reference Guide**⁸
- **OWASP Code Review guide**⁹
- **OWASP Top Ten**¹⁰

⁴ https://www.owasp.org/index.php/OWASP_Juice_Shop_Project

⁵ https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

⁶ <https://www.booktopia.com.au/ebooks/writing-secure-code-david-leblanc/prod2370006179962.html>

⁷ <http://owasp.org>

⁸ https://www.owasp.org/images/0/08/OWASP SCP_Quick_Reference_Guide_v2.pdf

⁹ https://www.owasp.org/images/2/2e/OWASP_Code_Review_Guide-V1_1.pdf

¹⁰ https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Inspecting and validating code bases for compliance

Automated code security analysis

We have previously discussed the use of Fortify, and Checkmarx CxSAST but there are many other tools for automating code security analysis. They include the following:

BinSkim¹¹ is a static analysis tool that scans binary files. BinSkim replaces an earlier Microsoft tool called BinScope. It checks that the executable produced (ie: a Windows PE formatted file) has opted into all 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.

Discussion - Security policy tooling

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?

¹¹ <https://blogs.msdn.microsoft.com/secdevblog/2016/08/17/introducing-binskim/>

Integrating license and vulnerability scans

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 in the release cycle gives you an advanced warning and allows you enough time to fix the issues.

Importantly, the cost to rectify issues is lower the earlier in the project that the issue is discovered.

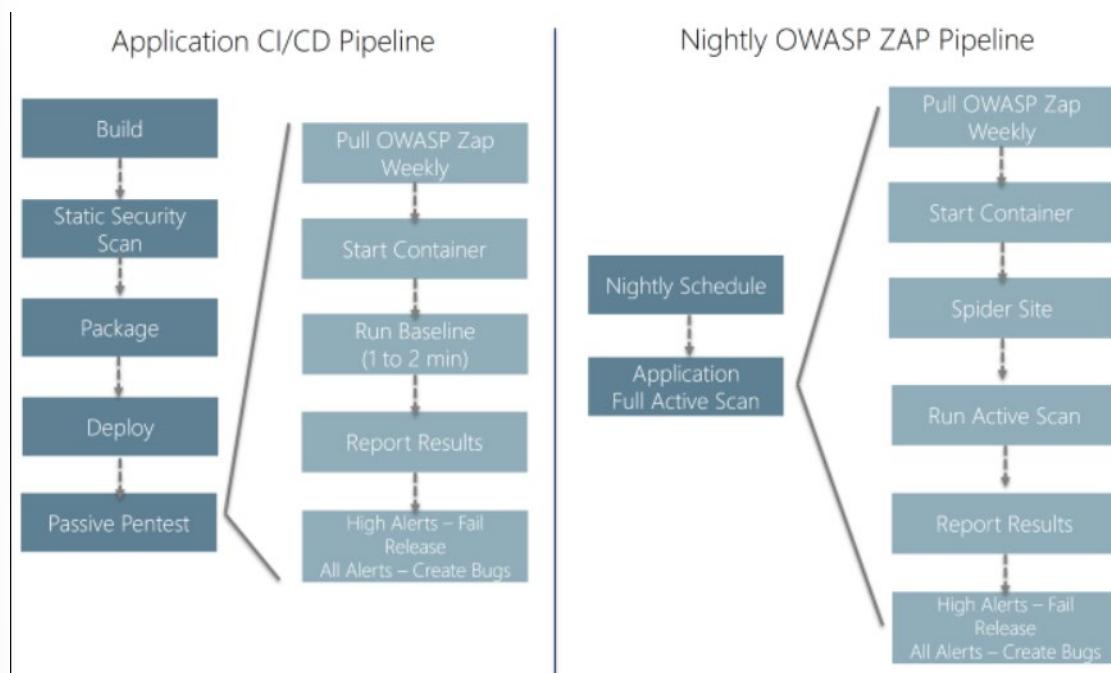
There are many available tools that can scan for these vulnerabilities within the build and release pipelines.

OWASP ZAP penetration testing

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 extension**¹² 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 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.

¹² <https://github.com/deliveron/owasp-zap-vsts-extension>



OWASP ZAP 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 DevOps Board view for the "OWASP ZAP Nightly / Release-40" release. The board has three columns: Backlog, Board, and Capacity.

Backlog: An Unparented item.

Board: A Kanban board with the following items:

| Item | Type | State | Created |
|--|------|-------|------------|
| 207810 Incomplete or No Cache-control and Pragma HTTP Header Set | Bug | New | 61.5 h ago |
| 207811 Cookie No HttpOnly Flag | Bug | New | 11 h ago |
| 207813 Web Browser XSS Protection Not Enabled | Bug | New | 11 h ago |
| 207812 Cookie Without Secure Flag | Bug | New | 11 h ago |
| 207815 X-Frame-Options Header Not Set | Bug | New | 11 h ago |
| 207814 X-Content-Type-Options Header Missing | Bug | New | 11 h ago |

Capacity: Shows 61.5 hours available.

Summary:

- Total tests: 6
- Passed: 0
- Failed: 6
- Others: 0
- Pass percentage: 0%
- Run duration: 0s

Test Results:

- 0/6 Passed - OWASP ZAP Security Tests
- ✗ 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 several tools that provide detection, prevention, and alerting using rules, such as **OWASP Top 10**¹³ 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.

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 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 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.

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

¹³ <https://owasp.org/www-project-top-ten/>

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.

Interpret alerts from scanning tools

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 results of the tooling scan 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.

CodeQL in GitHub

CodeQL is used by developers to automate security checks. CodeQL treats code like data that can be queried. GitHub researchers and community researchers have contributed standard CodeQL queries, and you can write your own.

A CodeQL analysis consists of three phases:

- Creating a CodeQL database (based upon the code)
- Run CodeQL queries against the database
- Interpret the results

CodeQL is available as a command line interpreter but also as an extension for Visual Studio Code.

For an overview of CodeQL, see: [CodeQL overview¹⁴](#)

For the available tools, see: [CodeQL tools¹⁵](#)

GitHub Dependabot alerts and security updates

Alerts

GitHub Dependabot detects vulnerable dependencies and sends Dependabot alerts about them in several situations:

- A new vulnerability is added to the GitHub Advisory database.
- New vulnerability data from Whitesource is processed.
- Dependency graph for a repository changes.

Alerts are detected in public repositories by default but can be enabled for other repositories.

Notifications can be sent via standard GitHub notification mechanisms.

For more information on Dependabot Alerts, see [About alerts for vulnerable dependencies¹⁶](#)

For details on the supported package ecosystems that alerts can be generated from, see [Supported package ecosystems¹⁷](#)

For notification details, see: [Configuring notifications¹⁸](#)

Security updates

A key advantage of Dependabot security updates, is that they can automatically create pull requests.

A developer can then review the suggested update, and triage what is required to incorporate it.

For more information on automatic security updates, see [About GitHub Dependabot security updates¹⁹](#)

¹⁴ <https://help.semmler.com/codeql/codeql-overview.html>

¹⁵ <https://help.semmler.com/codeql/codeql-tools.html>

¹⁶ <https://docs.github.com/en/free-pro-team@latest/github/managing-security-vulnerabilities/about-alerts-for-vulnerable-dependencies>

¹⁷ <https://docs.github.com/en/free-pro-team@latest/github/visualizing-repository-data-with-graphs/about-the-dependency-graph#supported-package-ecosystems>

¹⁸ <https://docs.github.com/en/free-pro-team@latest/github/managing-subscriptions-and-notifications-on-github/configuring-notifications#github-dependabot-alerts-notification-options>

¹⁹ <https://docs.github.com/en/free-pro-team@latest/github/managing-security-vulnerabilities/about-github-dependabot-security-updates>

Lab

Managing technical debt with SonarQube and Azure DevOps

Lab overview

In the context of Azure DevOps, the term *technical debt* represents suboptimal means of reaching tactical goals, which affect negatively the ability to reach strategic objectives in the area of software development and deployment. Technical debt affects productivity by making code hard to understand, prone to failures, time-consuming to change, and difficult to validate. Without proper oversight and management, technical debt can accumulate over time and significantly impact the overall quality of the software and the productivity of development teams in the longer term.

SonarQube²⁰ is an open source platform for continuous inspection of code quality that facilitates automatic reviews with static analysis of code to improve its quality by detecting bugs, code smells, and security vulnerabilities.

In this lab, you will learn how to setup SonarQube on Azure and integrate it with Azure DevOps.

Objectives

After you complete this lab, you will be able to:

- Provision SonarQube server as an **Azure Container Instance²¹** from the SonarQube Docker image
- Setup a SonarQube project
- Provision an Azure DevOps Project and configure CI pipeline to integrate with SonarQube
- Analyze SonarQube reports

Lab duration

- Estimated time: **60 minutes**

Lab updates

The labs are updated on a regular basis. For the latest information please visit:

- **AZ400-DesigningAndImplementingMicrosoftDevOpsSolutions²²**

²⁰ <https://www.sonarqube.org/>

²¹ <https://docs.microsoft.com/en-in/azure/container-instances/>

²² <https://microsoftlearning.github.io/AZ400-DesigningandImplementingMicrosoftDevOpsSolutions/>

Module review and takeaways

Module review questions

Review Question 1

What issues are often associated with the use of open source libraries?

Review Question 2

How can an open source library cause licensing issues if it is free to download?

Review Question 3

What is open source software?

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 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.