Microsoft

Microsoft Official Course

AZ-400T01

Implementing DevOps Development Processes

AZ-400T01
Implementing DevOps Development Processes

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.

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.**
   a) "Authorized Learning Center" means a Microsoft IT Academy Program Member, Microsoft Learning Competency Member, or such other entity as Microsoft may designate from time to time.
   b) "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.
   c) "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.
   d) "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 a MPN Member, or (iii) a Microsoft full-time employee.
   e) "Licensed Content" means the content accompanying this agreement which may include the Microsoft Instructor-Led Courseware or Trainer Content.
   f) "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.
   g) "Microsoft Instructor-Led Courseware" means the Microsoft-branded instructor-led training course that educates IT professionals and developers on Microsoft technologies. A Microsoft Instructor-Led Courseware title may be branded as MOC, Microsoft Dynamics or Microsoft Business Group courseware.
   h) "Microsoft IT Academy Program Member" means an active member of the Microsoft IT Academy Program.
   i) "Microsoft Learning Competency Member" means an active member of the Microsoft Partner Network program in good standing that currently holds the Learning Competency status.
   j) "MOC" means the "Official Microsoft Learning Product" instructor-led courseware known as Microsoft Official Course that educates IT professionals and developers on Microsoft technologies.
   k) "MPN Member" means an active Microsoft Partner Network program member in good standing.
   l) "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.

m) "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.

n) Trainer" means (i) an academically accredited educator engaged by a Microsoft IT Academy Program Member to teach an Authorized Training Session, and/or (ii) a MCT.

o) "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.

a) **If you are a Microsoft IT Academy Program Member:**

i) 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.

ii) 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,

provided you comply with the following:

iii) you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,

iv) 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,

v) 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,

vi) 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,

vii) 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,

viii) you will only deliver a maximum of 15 hours of training per week for each Authorized Training Session that uses a MOC title, and

ix) you acknowledge that Trainers that are not MCTs will not have access to all of the trainer resources for the Microsoft Instructor-Led Courseware.

b) **If you are a Microsoft Learning Competency Member:**

i) 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.

ii) 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,

provided you comply with the following:

iii) you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,

iv) 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,

v) you will ensure that each End User provided with a hard-copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,

vi) 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,

vii) 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,

viii) you will only provide access to the Microsoft Instructor-Led Courseware to End Users, and

ix) you will only provide access to the Trainer Content to MCTs.

c) **If you are a MPN Member:**

i) 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.

ii) 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,

provided you comply with the following:

iii) you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,

iv) 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,

v) you will ensure that each End User provided with a hard copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,

vi) 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,

vii) 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,

viii) 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,

ix) you will only provide access to the Microsoft Instructor-Led Courseware to End Users, and

x) you will only provide access to the Trainer Content to Trainers.

d) **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.

e) **If you are a Trainer.**

i) 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.

ii) 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. 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:

a) **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.

b) **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.

c) **Pre-release Term.** If you are an Microsoft IT Academy Program Member, Microsoft Learning Competency Member, MPN Member 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 "as is", we may not 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.**

    a) 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.

    b) 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 November 2014

Contents

Module 0 Welcome 1
Start Here 1

Module 1 Getting Started with Source Control 7
What is Source Control 7
Benefits of Source Control 11
Types of Source Control Systems 13
Introduction to Azure Repos 27
Migrating from TFVC to Git 31
Authenticating to Your Git Repos 36
Module 1 Review Questions 38

Module 2 Scaling Git for Enterprise DevOps 39
How to Structure Your Git Repo 39
Git Branching Workflows 41
Collaborating with Pull Requests 58
Why Care About GitHooks 65
Fostering Internal Open Source 70
Git Version 77
Public Projects 85
Files in Git 86
Module 2 Review Questions 88

Module 3 Implement & Manage Build Infrastructure 89
The Concept of Pipelines in DevOps 89
Azure Pipelines 91
Evaluate Use of Hosted vs Private Agents 94
Agent Pools 95
Pipelines & Concurrency 99
Azure DevOps and Open Source Projects 102
Azure Pipelines YAML vs Visual Designer 103
Setup Private Agents 105
Integrate Jenkins with Azure Pipelines 109
Integration External Source Control with Azure Pipelines 110
Analyze & Integrate Docker Multi-stage Builds 111
Module 3 Review Questions 116

Module 4 Managing Application Config and Secrets 117
Introduction to Security 117
Implement Secure & Compliant Development Processes 121
Rethinking Application Config Data 129
Manage Secrets, Tokens & Certificates 133
Implement Tools for Managing Security and Compliance 139
Module 4 Review Questions 149

Module 5 Implement a Mobile DevOps Strategy 151
Introduction to Mobile DevOps 151
Introduction to Visual Studio App Center 153
Manage Mobile Target Device Sets and Distribution Groups 157
Manage Target UI Test Device Sets 163
Provision Tester Devices for Deployment 164
Create Public and Private Distribution Groups 166
Module 5 Review Questions 168

Module 6 Course Completion 169
Final Exam 169

Module 0 Welcome

Start Here

Azure DevOps Curriculum

Welcome to the **Implementing DevOps Development Processes** course. This course is part of a series of courses to help you prepare for the AZ-400, **Microsoft Azure DevOps Solutions**[1] certification exam.

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

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

AZ-400 Study Areas	Weights
Implement DevOps Development Processes	**20-25%**
Implement Continuous Integration	10-15%
Implement Continuous Delivery	10-15%
Implement Dependency Management	5 -10%
Implement Application Infrastructure	15-20%
Implement Continuous Feedback	10-15%
Design a DevOps Strategy	20-25%

There are seven exam study areas. Each study area has a corresponding course. While it is not required that you have completed any of the other courses in the DevOps series before taking this course, it is

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

highly recommended that you start with the first course in the series, and progress through the courses in order.

✓ This course will focus on preparing you for the **Implement DevOps Development Processes** area of the AZ-400 certification exam.

About this Course

Course Description

This course provides the knowledge and skills to implement DevOps processes. Students will learn how to use source control, scale Git for an enterprise, and implement and manage build infrastructure.

Level: Intermediate

Audience

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

Prerequisites

- Students should have fundamental knowledge about Azure, version control, Agile software development, and core software development principles. It would be helpful to have experience in an organization that delivers software.
- It is recommended that you have experience working in an IDE, as well as some knowledge of the Azure portal. However, students who may not have a technical background in these technologies, but who are curious about DevOps practices as a culture shift, should be able to follow the procedural and expository explanations of continuous integration regardless.

Expected learning objectives

- How to implement source control strategies and repositories
- How to scale Git for Enterprise DevOps
- How to implement and manage a build infrastructure
- How to manage application configuration and secrets
- How to implement a mobile DevOps Strategy

Syllabus

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

Module 1 – Getting Started with Source Control

In this module, you will learn about source control principles and source control systems. You will also learn about Azure repositories, migrating strategies and authentication options.

- Lesson: What is Source Control
- Lesson: Benefits of Source Control
- Lesson: Types of Source Control Systems
- Lesson: Introduction to Azure Repos
- Lesson: Migrating from TFVC to Git

- Lesson: Authenticating to Your Git Repos
- Lab: Version Controlling with Git

Module 2 – Scaling Git for Enterprise DevOps

In this module, you will learn how to structure your Git repository and how to use branching workflows. You will also learn about pull requests, GitHooks, internal open source, Git versioning, and file handing.

- Lesson: How to Structure Your Git Repo
- Lesson: Git Branching Workflows
- Lesson: Collaborating with Pull Requests
- Lesson: Why Care about GitHooks
- Lesson: Fostering Internal Open Source
- Lesson: Git Version
- Lesson: Files in Git
- Lab: Code Review with Pull Requests

Module 3 – Implement and Manage Build Infrastructure

In this module, you will learn about Azure pipelines, agents, agent pools, concurrency, integration with Jenkins, and Docker multiple stage builds.

- Lesson: The Concept of Pipelines in DevOps
- Lesson: Azure Pipelines
- Lesson: Evaluate Use of Hosted vs Private Agents
- Lesson: Agent Pools
- Lesson: Pipelines and Concurrency
- Lesson: Azure DevOps and Open Source Projects
- Lesson: Azure Pipelines YAML vs Visual Designer
- Lesson: Setup Private Agents
- Lesson: Integrate Jenkins with Azure Pipelines
- Lesson: Integration External Source Control with Azure Pipelines
- Lesson: Analyze and Integrate Docker Multi-stage Builds
- Lab: Integrate Jenkins with Azure Pipelines
- Lab: Integration External Source Control with Azure Pipelines
- Lab: Deploying a Multi-Container Application to Azure Kubernetes Service

Module 4 - Managing Application Config and Secrets

In this module, you will learn about security and compliance including secrets, tokens, certifications, configuration data, and tooling.

- Lesson: Introduction to security
- Lesson: Implement Secure and Compliant Development Processes
- Lesson: Rethinking Application Config Data
- Lesson: Manage Secrets, Tokens and Certificates

- Lesson: Implement Tools for Managing Security and Compliance
- Lab: SonarCloud
- Lab: WhiteSource

Module 5 - Implement a Mobile DevOps Strategy

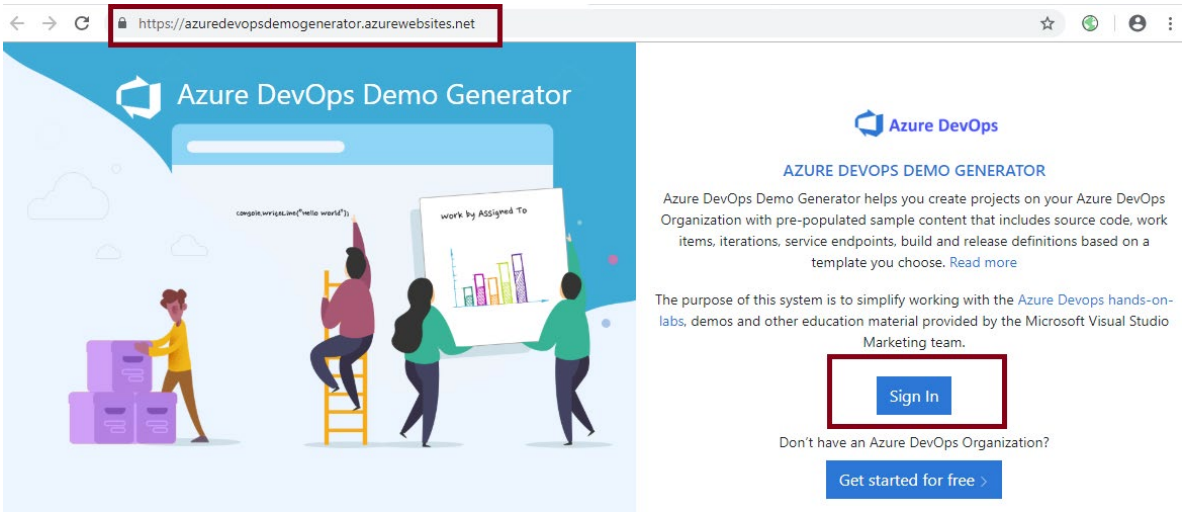
In this module, you will learn about mobile DevOps strategies using the App Center, Device Sets, and Distribution Groups.

- Lesson: Introduction to Mobile DevOps
- Lesson: Introdcution to Visual Studio App Center
- Lesson: Manage Mobile Target Device Sets and Distribution Groups
- Lesson: Manage Target UI Test Device Sets
- Lesson: Create Public and Private Distribution Groups

Lab Environment Setup

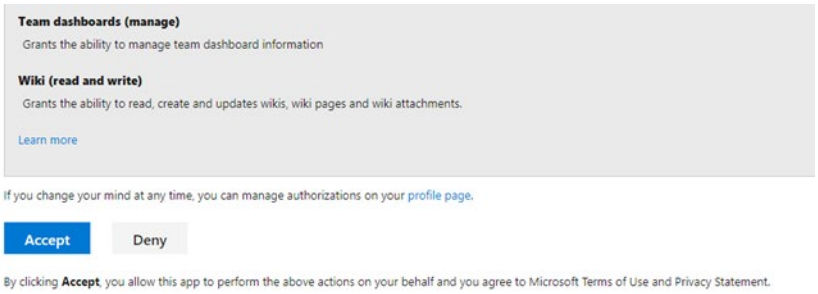
We highly recommend that you complete the assigned hands-on lab work.To do the hands-on labs, you will need to complete the following steps.

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

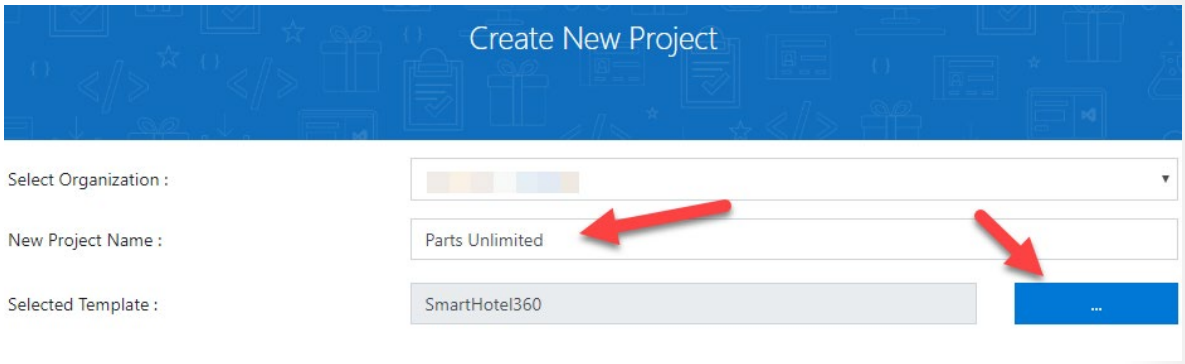


4. You will then be asked to confirm that the generator site can have permission to create objects in your Azure DevOps account.

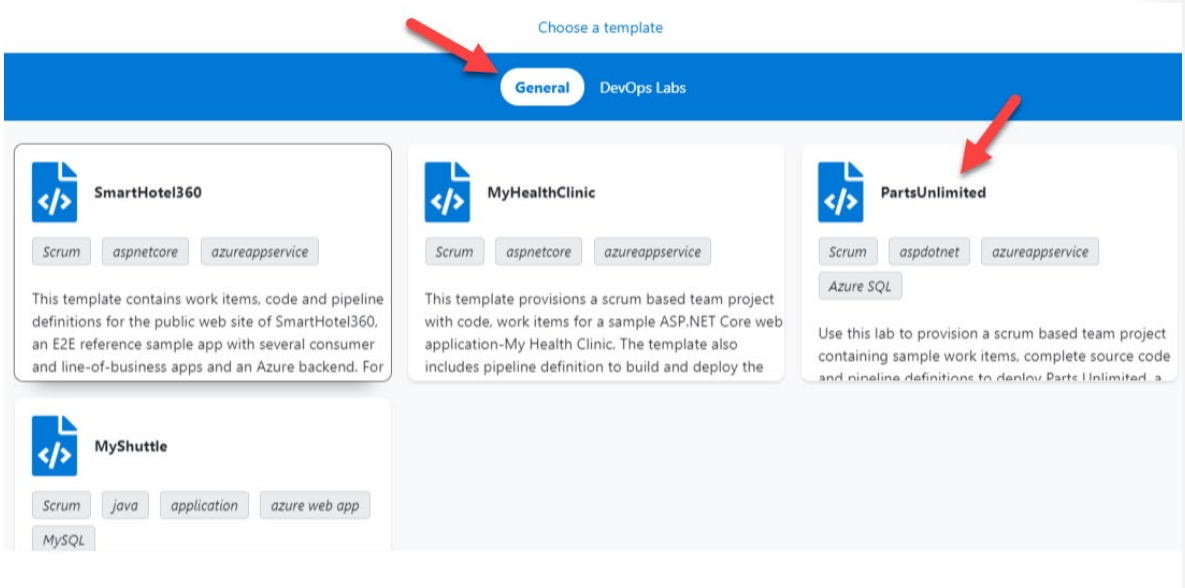
2 https://www.azuredevopslabs.com/
3 https://azure.microsoft.com/en-us/free/
4 https://azuredevopsdemogenerator.azurewebsites.net/



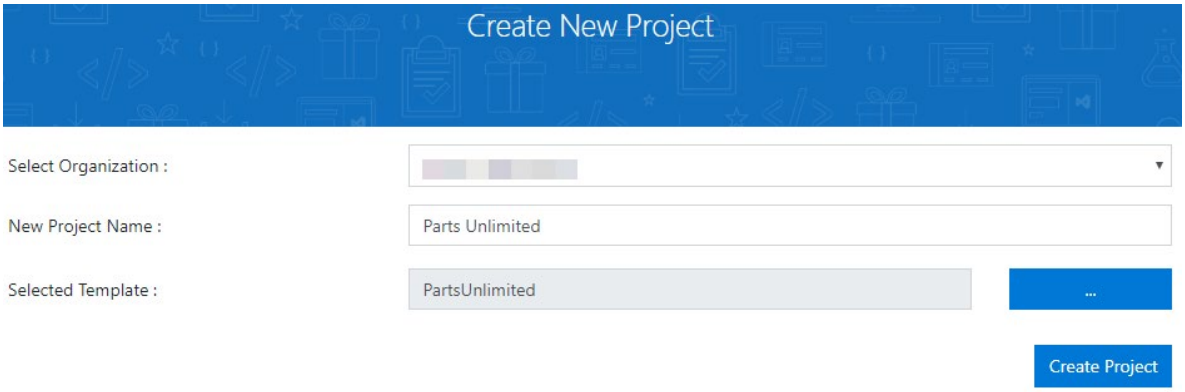
5. If you agree, click the **Accept** button and you should be greeted by the Create New Project screen:



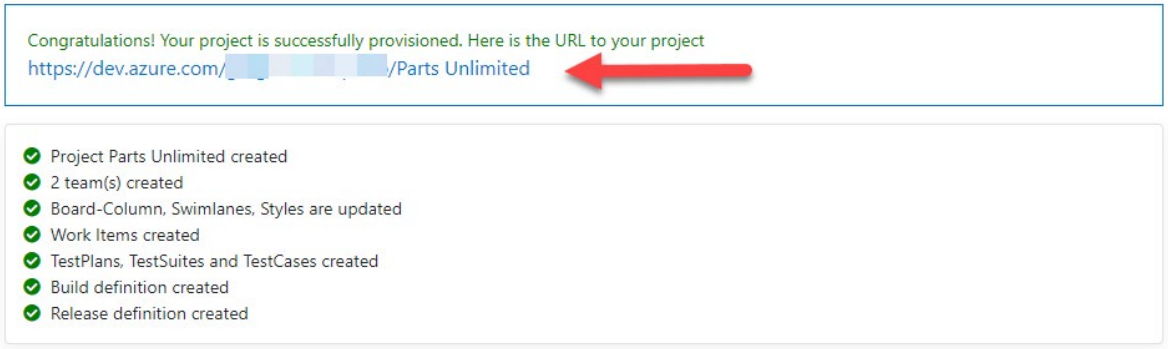
6. Select the appropriate organization (if you have more than one) and enter **Parts Unlimited** as the **New Project Name**, then click the ellipsis to view the available templates. These will change over time but you should see a screen similar to the following:



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



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



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

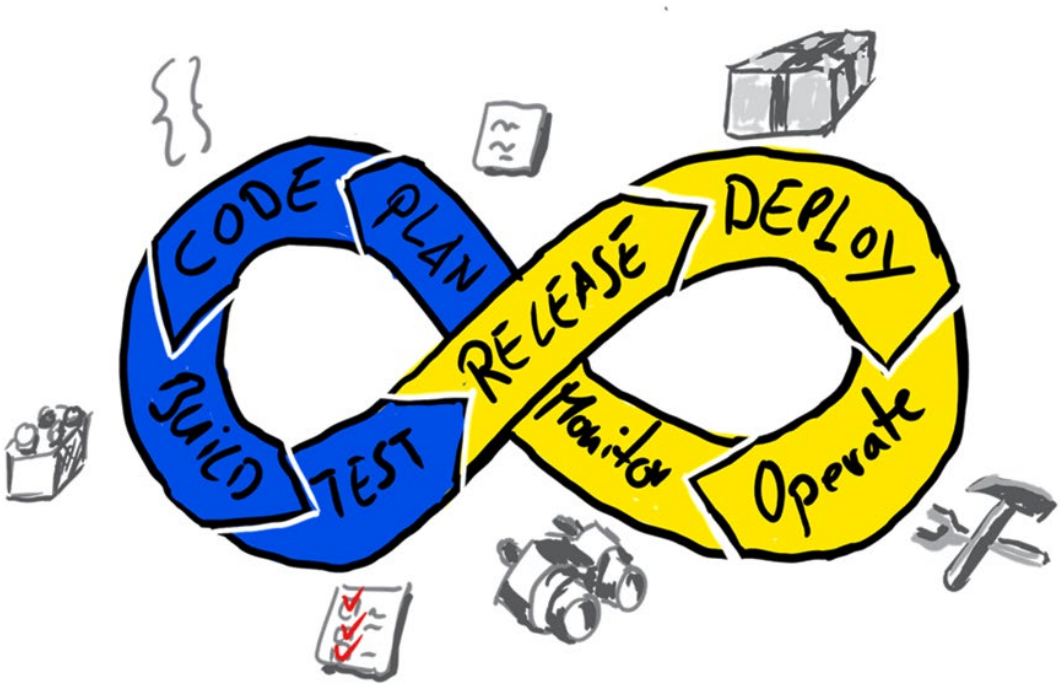
✓ Note that because Azure DevOps was previously called VSTS (Visual Studio Team Services), some of the existing hands-on labs might refer to VSTS rather than Azure DevOps.

Module 1 Getting Started with Source Control

What is Source Control

Introduction to Source Control

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



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

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

Foundational Practices of DevOps

In an effort to eliminate the enigma that comes with any new business practice… the **2018 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	• 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	• **Application development teams use version control.** • **Teams deploy on a standard set of operating systems.**	• 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	• **Build on a standard set of technology.** • **Teams deploy on a single standard operating system.**	• Deployment patterns for building applications and services are reused. • Rearchitect applications based on business needs. • Put system configurations in version control.
Stage 3	• **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.	• 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	• **System configurations are automated.** • **Provisioning is automated.** • Application configurations are in version control. • Infrastructure teams use version control.	• Security policy configurations are automated. • Resources made available via self-service.
Stage 5	• **Incident responses are automated.** • **Resources available via self-service.** • Rearchitect applications based on business needs. • Security teams are involved in technology design and deployment.	• Security policy configurations are automated. • Application developers deploy testing environments on their own. • Success metrics for projects are visible. • Provisioning is automated.

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

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

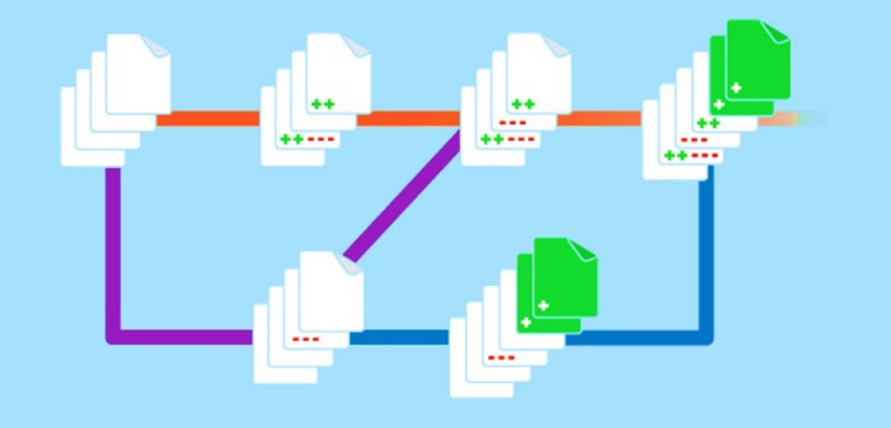
Version control is important for all software development projects and is particularly vital at large businesses and enterprises. Enterprises have many stakeholders, distributed teams, strict processes and workflows, silo'ed organizations and hierarchical organization. All of those characteristics represent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

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

regulations, need a practical way to ensure that all standards are being met appropriately and risk is mitigated.

What is Source Control

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



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

Common Software Development Values

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

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

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

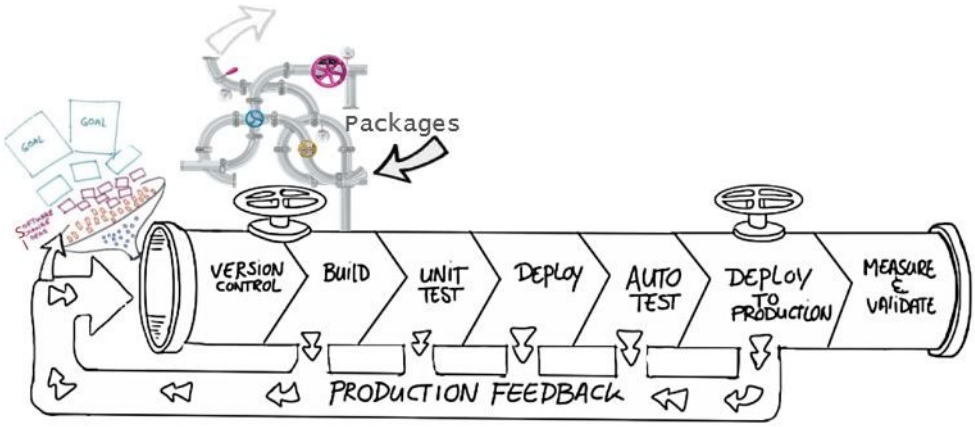
Version control is a solution for managing and saving changes made to any manually created assets. It allows you to go back in time and easily roll back to previously working versions if changes are made to source code. Version control tools allow you to see who made changes, when and what exactly was changed. Version control also makes experimenting easy and most importantly makes collaboration possible. Without version control, collaborating over source code would be a painful operation.

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

Benefits of Source Control

Benefits of Source Control

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



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

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

Best Practices for Source Control

Best Practices for Version Control Users

- **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 accidently 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 to why it was created or changed by providing traceability across requirements and code changes.
- **No matter your background or preferences, be a team player and follow agreed conventions and workflows**. Consistency is important and helps ensure quality making it easier for team members to pick up where you left off, to review your code, to debug, etc.

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

Types of Source Control Systems

Centralized Version Control

Centralized Version Control

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

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

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

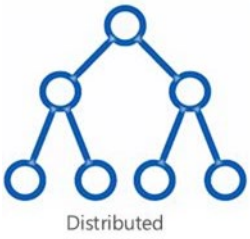
A Typical Centralized Version Control Workflow

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

- Pull down any changes other people have made from the central server.
- Make your changes, and make sure they work properly.
- Commit your changes to the central server, so other programmers can see them.

Distributed Version Control

Distributed Version Control



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

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

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

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

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

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

Advantages Over Centralized Version Control

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

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

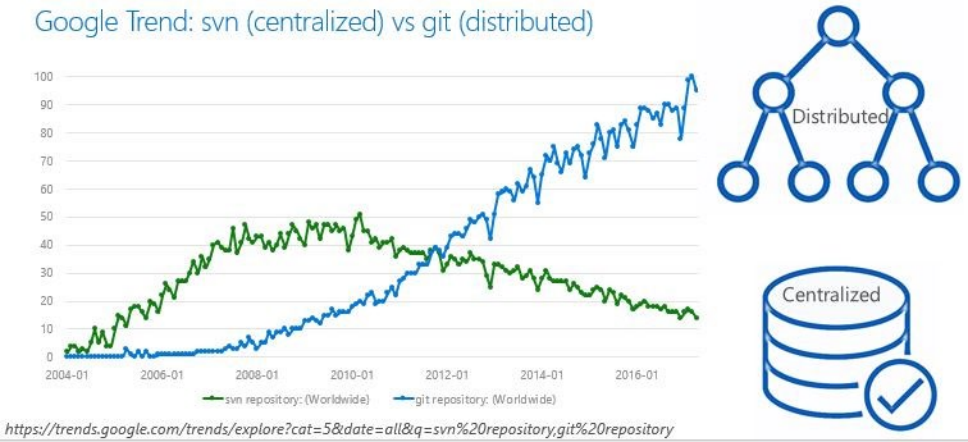
Disadvantages Compared to Centralized Version Control

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

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

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

Git and TFVC



Git (distributed)

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

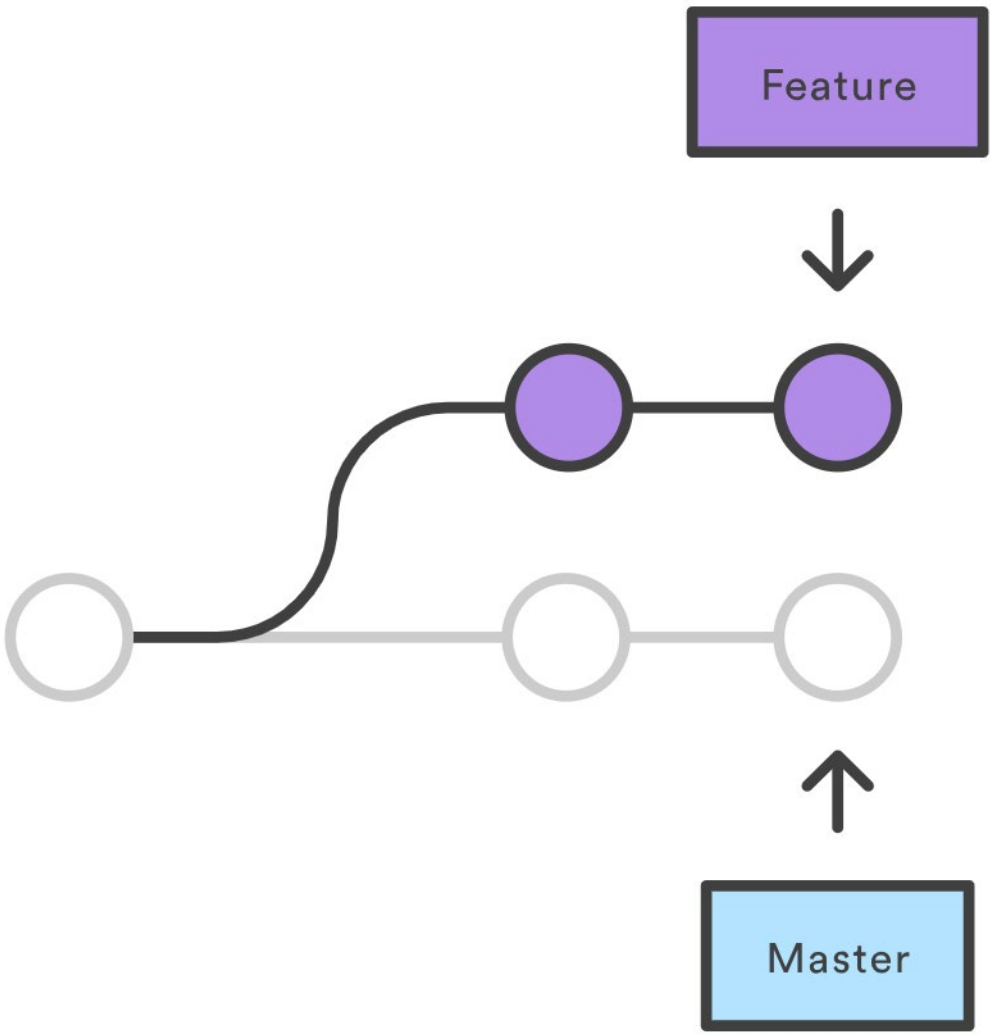
- **Server workspaces** - Before making changes, team members publicly check out files. Most operations require developers to be connected to the server. This system facilitates locking workflows. Other systems that work this way include Visual Source Safe, Perforce, and CVS. With server workspaces,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

Feature Branch Workflow

One of the biggest advantages of Git is its branching capabilities. Unlike centralized version control systems, Git branches are cheap and easy to merge. This facilitates the feature branch workflow popular with many Git users.

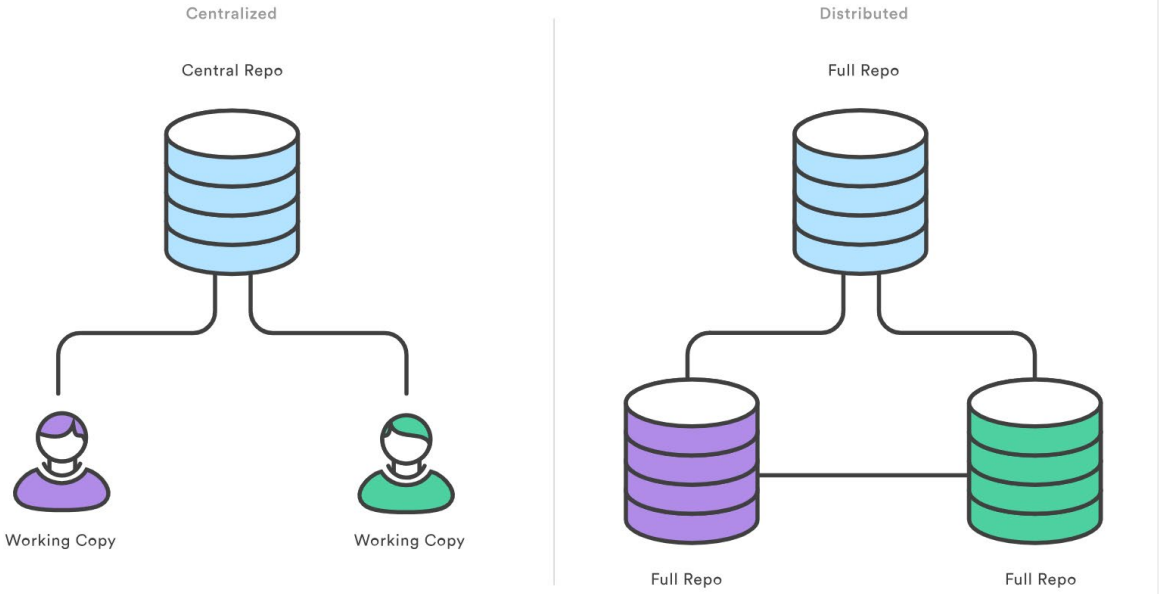


Feature branches provide an isolated environment for every change to your codebase. When a developer wants to start working on something—no matter how big or small—they create a new branch. This ensures that the master branch always contains production-quality code.

Using feature branches 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.

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.



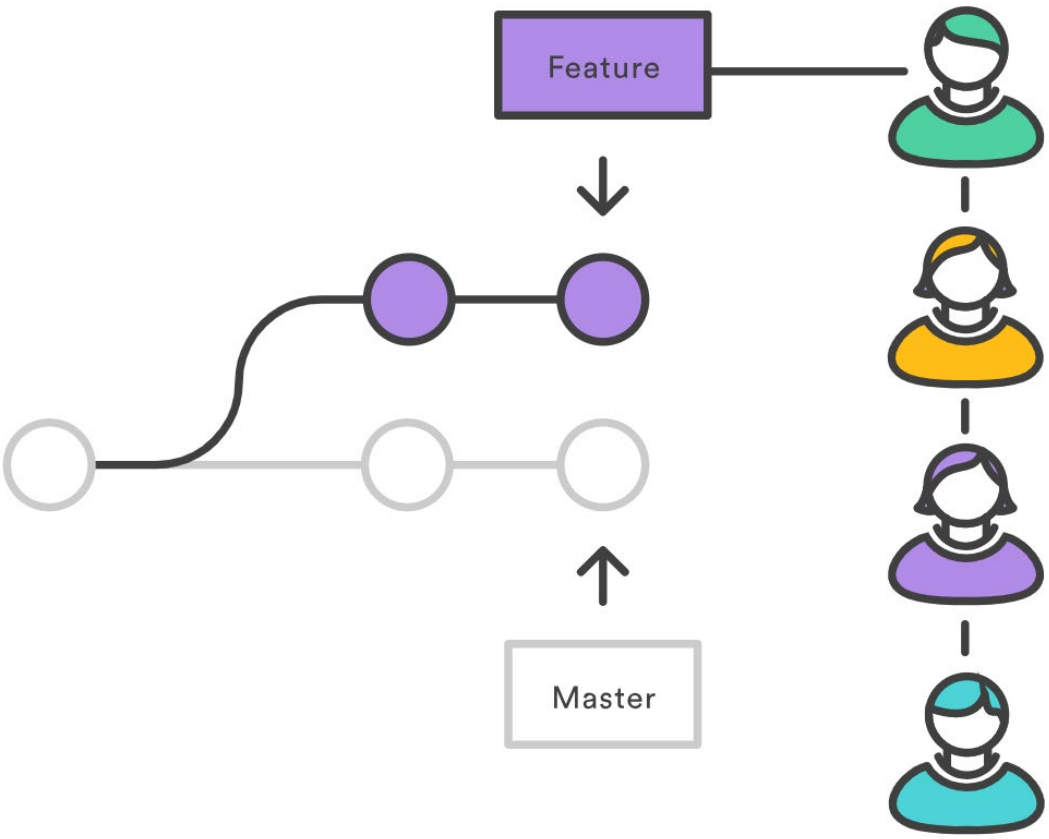
Having a full local history makes Git fast, since it means you don't need a network connection to create commits, inspect previous versions of a file, or perform diffs between commits.

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

Pull Requests

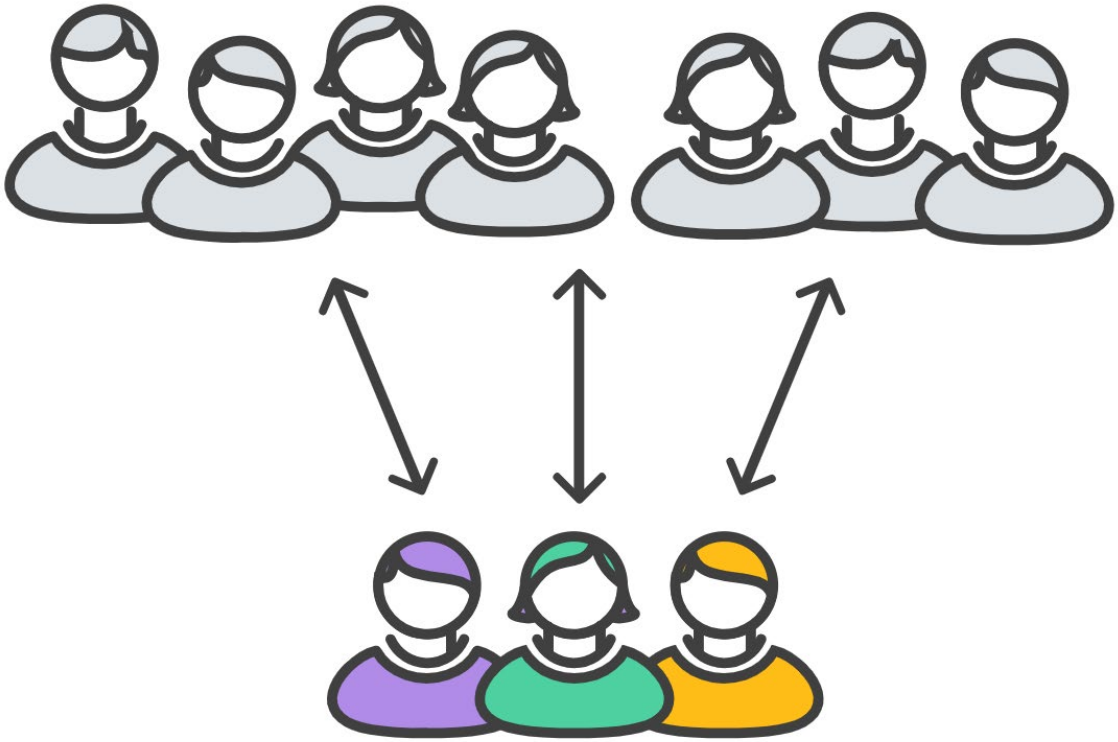
Many source code management tools such as Azure Repos enhance core Git functionality with pull requests. A pull request is a way to ask another developer to merge one of your branches into their repository. This not only makes it easier for project leads to keep track of changes, but also lets developers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

Community

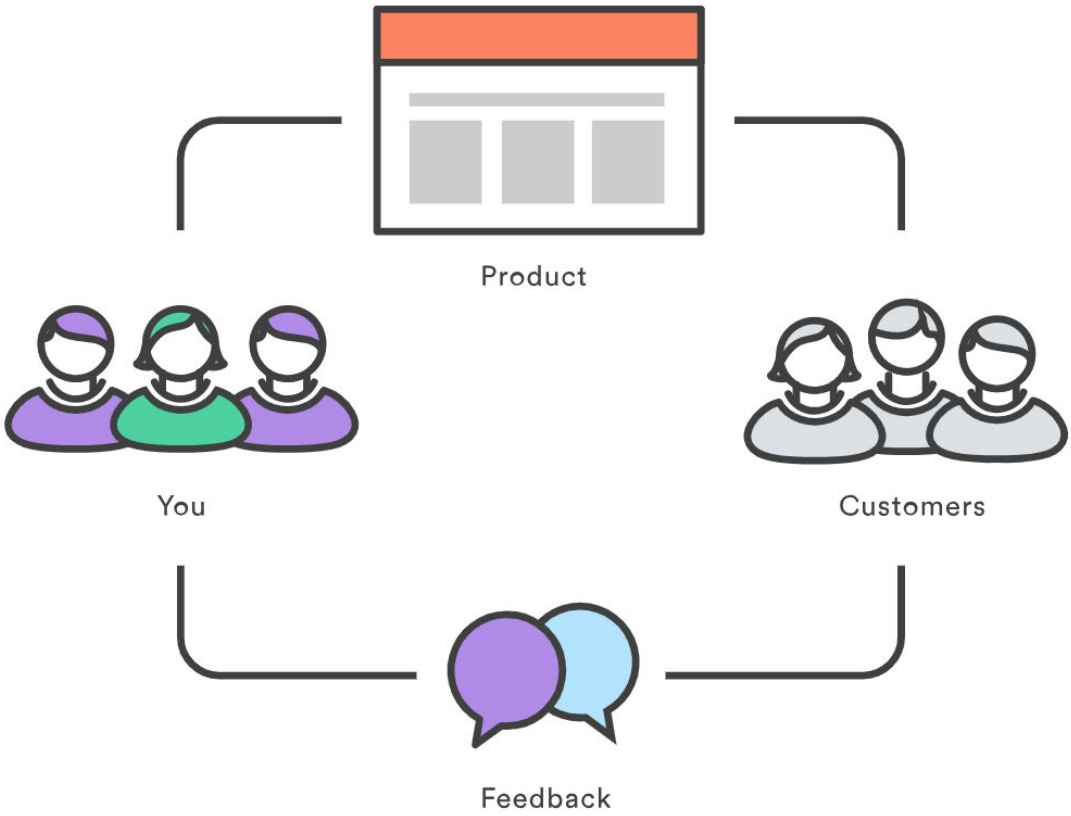
In many circles, Git has come to be the expected version control system for new projects. If your team is using Git, odds are you won't have to train new hires on your workflow, because they'll already be familiar with distributed development.



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.

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

Common Objections to using Git

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

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

Overwriting History

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

Large Files

Git works best with repos that are small and that do not contain large files (or binaries). Every time you (or your build machines) clone the repo, they get the entire repo with all its history from the first commit. This is great for most situations, but can be frustrating if you have large files. Binary files are even worse since Git just can't optimize how they are stored. That's why **Git LFS**[2] 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 Artifacts**[3] or some other package management tool to store binaries you have source code for. However, teams that have large files (like 3D models or other assets) you can use Git LFS to keep your code repo slim and trim.

Large Repos

This used to be a blocker - but fortunately the engineers at Microsoft have been on a multi-year journey to convert all of Microsoft's source code to Git. The Windows team has a repo that's over 300GB in size, and they use Git for source control! How? They invented **Virtual File System (VFS) for Git**[4]. VFS for Git is a client plugin that lets Git think it has the entire repo - but only fetches files from the upstream repo when a file is touched. This means you can clone your giant repo in a few seconds, and only when you touch files does Git fetch them down locally. In this way, the Windows team is able to use Git even for their giant repo.

Git? GitHub?

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

Learning Curve

There is a learning curve - if you've never used source control before you're probably better off when learning Git. I've found that users of centralized source control (TFVC or SubVersion) battle initially to make the mental shift especially around branches and synchronizing. Once developers grok how Git branches work and get over the fact that they have to commit and then push, they have all the basics they need to be successful in Git.

2 https://git-lfs.github.com/
3 https://azure.microsoft.com/en-us/services/devops/artifacts/
4 https://github.com/Microsoft/VFSForGit

Working with Git Locally

Git and Continuous Delivery is one of those delicious chocolate & peanut butter combinations we occasionally find in the software world, two great tastes that taste great together! Continuous Delivery of software demands a significant level of automation. It's hard to deliver continuously if you don't have a quality codebase. Git provides you with the building blocks to really take charge of quality in your codebase; it gives you the ability to automate most of the checks in your codebase even before committing the code into you 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 2.0 SDK or later: https://www.microsoft.com/net/download/macos
- Visual Studio Code: https://code.visualstudio.com/download
- C# Visual Studio Code extension: https://marketplace.visualstudio.com/items?itemName=ms-vscode.csharp
- Git: https://git-scm.com/downloads
- Git for Windows (if you are using Windows): 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 Lens (https://marketplace.visualstudio.com/items?itemName=eamodio. gitlens): 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 History (https://marketplace.visualstudio.com/items?itemName= donjayamanne.githistory): 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 "Tarun Arora"
git config --global user.email "tarun.arora@contoso.com"
```

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

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

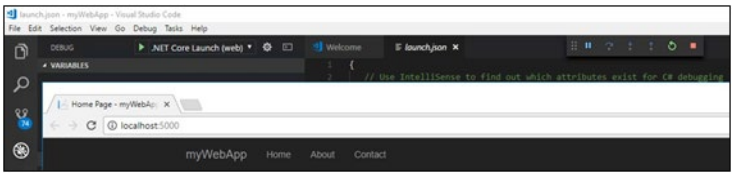
4. Create a new ASP.NET core application. The new command offers a collection of switches that can be used for language, authentication, and framework selection (more details can be found on Microsoft docs: https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-new?tabs=netcore2x):

```
dotnet new mvc
```

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

```
code .
```

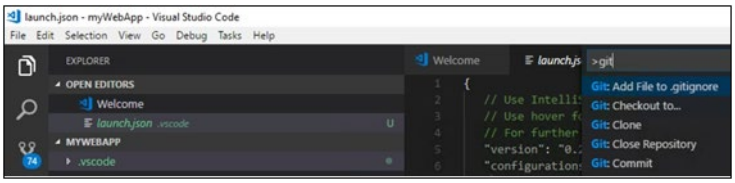
5. When the project opens up in Visual Studio Code, select Yes for the Required assets to build and debug are missing from 'myWebApp'. Add them? warning message. Select Restore for the There are unresolved dependencies info message. Hit F5 to debug the application, then myWebApp will load in the browser,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



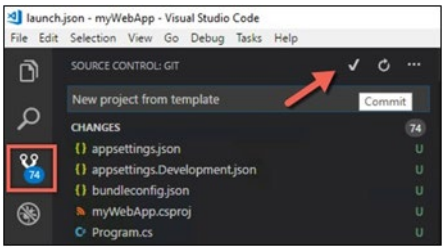
If you prefer to use the commandline you can run the following commands in the context of the git repository to run the web application.

```
dotnet build
dotnet run
```

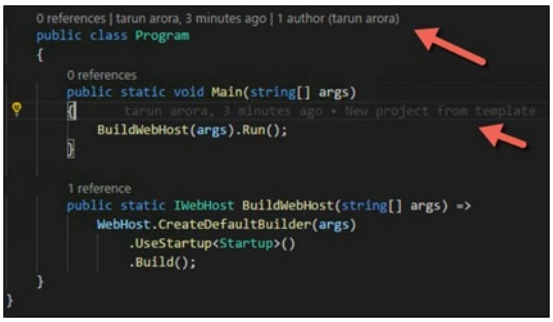
You'll notice the .vscode folder is added to your working folder. To avoid committing this folder into your Git repository, you can include this in the .gitignore file. With the .vscode folder selected, hit F1 to launch the command window in Visual Studio Code, type gitIgnore, and accept the option to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



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



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



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.

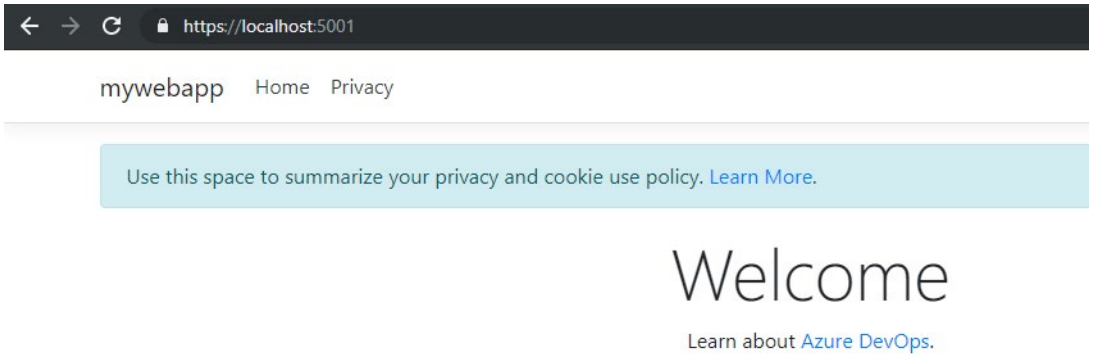
8. Now navigate to the file `~\Views\Home\Index.cshtml` and replace the contents with the text below...

```
@{
    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 .

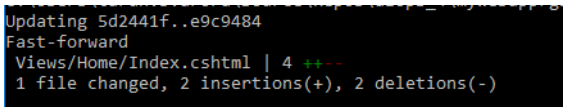
git commit -m "updated welcome page"

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
```



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 into the repository.
2. To see the history of changes in the master branch run the command `git log -v`

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

    updated welcome page

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

    project init
```

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

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

    updated welcome page

diff --git a/Views/Home/Index.cshtml b/Views/Home/Index.cshtml
index d2d19bd..6d8ad94 100644
--- a/Views/Home/Index.cshtml
+++ b/Views/Home/Index.cshtml
@@ -4,5 +4,5 @@

 <div class="text-center">
     <h1 class="display-4">Welcome</h1>
-    <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">building Web apps with ASP.NET Core</a>.</p>
-</div>
+    <p>Learn about <a href="https://azure.microsoft.com/en-gb/services/devops/">Azure DevOps</a>.</p>
+</div>
\ No newline at end of file
```

There is more...

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

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

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

Introduction to Azure Repos

Azure Repos with Azure DevOps CLI

The Azure DevOps extension for the Azure CLI allows you to experience Azure DevOps from the command line, bringing the capability to manage Azure DevOps right to your fingertips! This allows you to work in a streamlined task/command oriented manner without having to worry about the GUI flows, providing you a faster and flexible interaction canvas. The Azure DevOps Extension for Azure CLI adds Pipelines, Boards, Repos, Artifacts and DevOps commands to the Azure CLI.

In this tutorial, we'll look at how to add the Azure DevOps extension to the Azure CLI. We'll also learn how to administer, manage, report and run some basic DevOps operations using the CLI.

Getting ready

1. Since Azure DevOps is an extension of Azure CLI, you'll need to first install Azure CLI. You must have at least v2.0.49, which you can verify with `az --version` command.
2. If you are on Windows, you can download and install the Azure DevOps extension from the **marketplace**[5]
3. Next add the Azure DevOps Extension to your Azure CLI. Launch powershell or cmd and run the command, `az extension add --name azure-devops`
4. To initialise Azure CLI, run the command `az login`

If the CLI can open your default browser, it will do so and load a sign-in page. Otherwise, you need to open a browser page and follow the instructions on the command line to enter an authorization code after navigating to https://aka.ms/devicelogin in your browser.

5. To interact with the Azure DevOps CLI, you will need to login to the orgnization you want to interact with. When you run the first command, you'll need to specify the PAT token for the org. If you have multiple organisations mapped out, run the second command to configure the default org and (or) project.

```
az devops login --org https://dev.azure.com/geeks

az devops configure --defaults organization=https://dev.azure.com/geeks
```

The az devops extension can output the results in multiple formats including json, jsonc, table, yaml, etc. A default output format can be configured using the default command, alternatively this can be specified with each call.

How to do it...

1. To see a full list of actions you can perform with this CLI, run the command `az devops -h`

5 https://docs.microsoft.com/en-gb/cli/azure/install-azure-cli?view=azure-cli-latest

```
C:\Users\tarun.b.arora>az devops -h

Group
    az devops : Manage Azure DevOps organization level operations.
        Related Groups
        az pipelines: Manage Azure Pipelines
        az boards: Manage Azure Boards
        az repos: Manage Azure Repos
        az artifacts: Manage Azure Artifacts.

Subgroups:
    admin            : Manage administration operations.
    extension        : Manage extensions.
    project          : Manage team projects.
    security         : Manage security related operations.
    service-endpoint : Manage service endpoints/service connections.
    team             : Manage teams.
    user             : Manage users.
    wiki             : Manage wikis.

Commands:
    configure        : Configure the Azure DevOps CLI or view your configuration.
    feedback         : Displays information on how to provide feedback to the Azure DevOps CLI team.
    invoke           : This command will invoke request for any DevOps area and resource. Please use
                       only json output as the response of this command is not fixed. Helpful docs -
                       https://docs.microsoft.com/en-us/rest/api/azure/devops/.
    login            : Set the credential (PAT) to use for a particular organization.
    logout           : Clear the credential for all or a particular organization.
```

2. To see a list of all the projects in the geeks organization

```
az devops project list -o table
```

```
ID                                    Name                  Visibility
------------------------------------  --------------------  ------------
fd93bce7-3d3a-4900-b7f4-b8625807db51  AppVeyorExtension     Private
e2996175-ced8-40ca-81b2-335ddf68a62c  AvanadeExtensions     Private
f0d0fa34-5a53-4576-be8e-cbe1163b695a  BuildActivities       Private
4e277867-2f5d-4185-9a20-df5703a8d6bf  BuildGuru             Private
88535fd9-7614-4283-a912-7117b0d585f0  Bureau Data           Private
f601713e-fd20-47a2-8e30-de59d5c02e60  CityStyle             Private
ada4821b-1239-4978-bf88-2a6101d062c4  Dashboard             Private
d453c4c1-3f1d-4026-b2e1-8911ff8d4319  Demand Management     Private
0b1a9778-5d09-405b-b5d8-7a6fc51d5efc  DevOpsContent         Public
5cdb4761-98d6-4642-b652-8cf567bdaafa  EnvOps                Private
ab2c893c-5151-4980-b20a-cc8ddd7260c7  EnvOps-Temp           Private
121b6d1f-3025-453d-8eb2-e4c720b7aa71  Event                 Private
03a1b747-70d4-4a81-a07a-314b6820e4fd  Fabrikam              Private
150b6e3c-c924-4796-bbd1-3c2885052289  FabrikamTFVC1         Private
22467a2f-0553-4030-b133-f96150c1a24b  GeeksWorkshop         Private
3c9a9d12-b289-419c-9987-c7e8984c7d20  LoadTestingWithAzure  Private
360d2199-ea66-4101-83a2-ae5c931722e1  Mint                  Public
```

3. To see a list of all the pipelines in the partsunlimited project

```
az pipelines list -p partsunlimited -o table
```

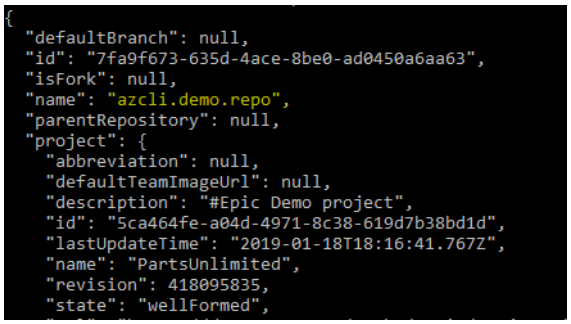
```
ID    Name                          Status    Default Queue
----  ----------------------------  --------  --------------
82    partsunlimited.web.framework  enabled   Hosted VS2017
83    partsunlimited.auth           enabled   Hosted VS2017
86    foo.codeanalysis              enabled   Hosted VS2017
87    packagescan                   enabled   Hosted VS2017
91    foo.ci                        enabled   Hosted VS2017
92    partsunlimited.auth2          enabled   Hosted VS2017
93    PartsUnlimited.kv.demo        enabled   Hosted VS2017
95    partsunlimited.arm.scanner    enabled   Hosted VS2017
96    deepdive.ci                   enabled   Hosted VS2017
97    deepdive.provision.agents     enabled   Hosted VS2017
98    deepdive.infratemplate.scan   enabled   Hosted VS2017
```

To see the build definition for 'deepdive.ci'

```
az pipelines build definition show --id 96 -p partsunlimited
```

4. So far we've looked at read operations, next, we'll learn how to create a new empty repository of type git in an existing team project

```
az repos create --name azcli.demo.repo -p partsunlimited
```



How it works...

The Azure DevOps CLI is a wrapper over the Azure DevOps REST API which abstracts the complexity of API's away by giving you purposeful switches to work across one or multiple Azure DevOps instances. The command syntax follows the structure outlined below, making it really easy to query what you are interested in.

```
az [group] [subgroup] [command] {parameters}
```

There is more

Refer to this example here to learn how to leverage the Azure DevOps CLI to publish a banner message for all the users of the **Azure DevOps organization**[6]

Lab - Version Controlling with Git

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.

Activities to perform,

- Exercise 1: Cloning an existing repository
- Exercise 2: Saving work with commits
- Exercise 3: Reviewing history
- Exercise 4: Managing branches from Visual Studio
- Exercise 5: Managing branches from Azure DevOps

6 https://www.visualstudiogeeks.com/azure/azure-devops-create-info-banner-for-all-users

Version Controlling with Git in Azure Repos - **https://www.azuredevopslabs.com/labs/azuredevops/git/**

Migrating from TFVC to Git

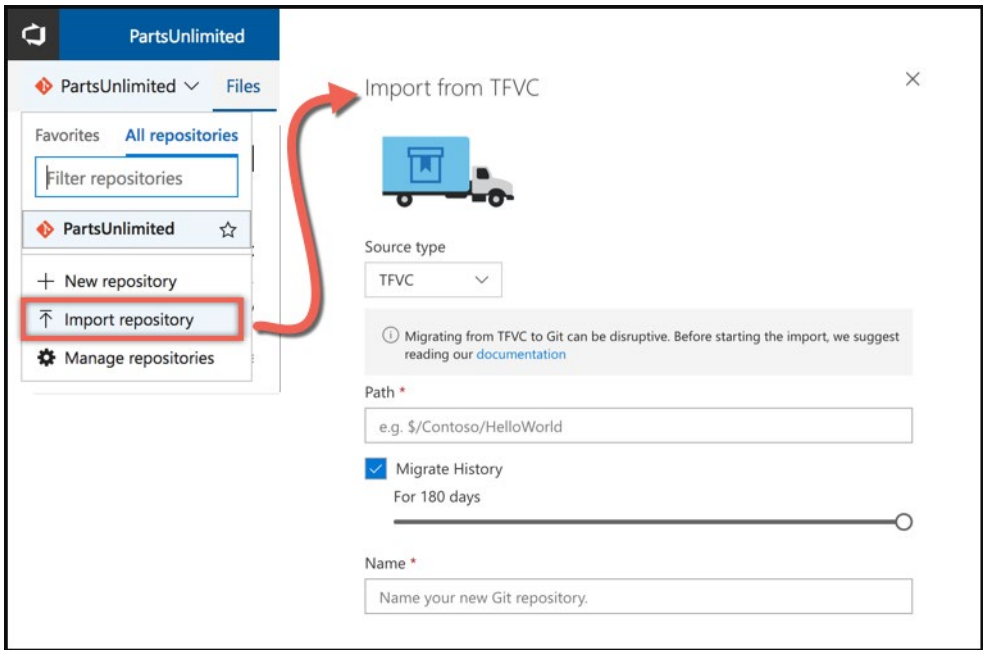
Single Branch Import

Migrating the Tip

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

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. Just click on "Import repository" from the Azure Repos top level drop-down menu to pop open the dialog. Then enter the path to the branch you're migrating (yes, you can only choose one branch) and if you want history or not (up to 180 days). Then add in a name for the repo and the import will be triggered.



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

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

Git TFS

Git-tfs

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

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

Migrating from TFVC to Git Using Git-tfs

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

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

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

Cloning
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 here**[7]

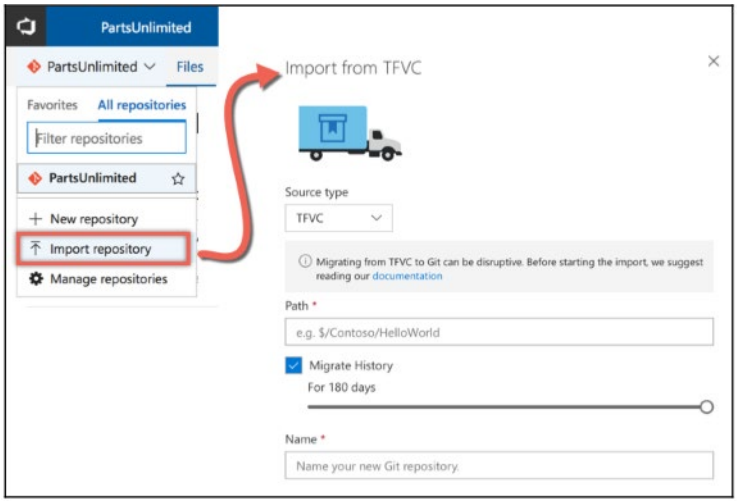
Working

```
cd some_project
git log
```

Import Repository

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

7 https://github.com/git-tfs/git-tfs/blob/master/doc/commands/clone.md



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

You may also come across use cases where you need to migrate from the TFVC repository that is hosted in an Azure DevOps server that your new Azure DevOps server doesn't have direct access to through the network. In this recipe, we'll learn how to use the open source command-line git-tf to migrate your TFVC projects with complete history into Git, and then publish the local Git repository into a new Git repository in Azure DevOps Server.

Getting ready

In this section, we'll cover how to download and set up git-tf to prepare for the migration:

- Download the git-tf command-line tools from the **Microsoft Download Center**[8], then extract the ZIP file into the C:\git-tf folder.
- To access git-tf directly from the command line, add the path C:\git-tf path to2. your path environment variable.
- Create a folder, C:\migrated, to store the migrated repositories. In this example, we'll assume that the host TFVC repository that needs to be migrated is hosted on the http://myOldAzure DevOps Server-Server/Azure DevOps Server/DefaultCollection Azure DevOps Server server in the $/OldTeamProject/App2BeMigrated folder.

How to do it...

In this section we'll cover the steps for migrating the code from TFVC to git with history:

1. Launch the command line and run the following command; –deep is used to1. extract the entire history from this repository. This operation may take longer to complete, depending on the size and depth of history of the source repository:

```
git-tf clone --deep http://myOldAzure DevOps ServerServer/Azure DevOps
Server/DefaultCollection $/OldTeamProject/App2BeMigrated C:\migrated\
```

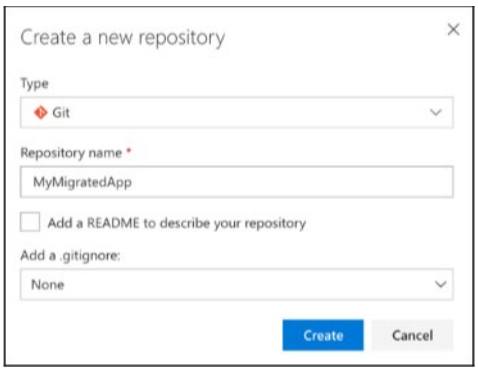
8 https://www.microsoft.com/en-us/download/details.aspx?id=30474

```
App2BeMigrated
```

2. In the command line, change the directory to C:\migrated\App2BeMigrate and run the following command. This will clean the Git metadata from the commit messages:

```
git filter-branch -f --msg-filter "sed 's/^git-Azure DevOps Serverid:.*;C\
([0-9]*\)$/Changeset:\1/g'" -- --all
```

Delete the .git/refs/original folder in C:\migrated\App2BeMigrated to delete the old branches as they are not needed anymore. To publish the local migrated Git repository in Azure DevOps Server, you'll need to create a new Git repository in Azure DevOps Server. To do this, navigate to the parts unlimited team project in the team portal and create a new Git code repository, MyMigratedApp:



4. Run the following command to add the newly-created Git repository as an origin to the migrated Git repository:
   git remote add origin http://Azure DevOps Server2018/Azure DevOps Server/DefaultCollection/PartsUnlimited/_git/MyMigratedApp Run the following command to push the new Git repository to the remote origin:5.

```
git push -u origin –all
```

How it works…

While most of the other commands are pretty self explanatory, the emphasis here is on the –deep switch:

1. By including the –deep switch, the entire history of the TFVC repository is1. consumed during the migration process. If this keyword is left out, only the most recent changeset will be fetched, which you wouldn't want in the scenario of a full export.

There's more…

Another situation you may keep running into is that the committer names are different on Azure DevOps Server and Git. As a rule, Git recognizes committers by their designed email address, while Azure DevOps Server ordinarily utilizes your Windows character. Accordingly, a similar individual may be spoken to by two different committers on the Git store. Use the Azure DevOps Server username for the import and the genuine Git client for new submits that are made on the Git storehouse. Utilize the below command to remap the names:

```
git filter-branch -f --commit-filter "
if [ "$GIT_COMMITTER_NAME" = "<old Azure DevOps Server user>" ];
```

```
then GIT_COMMITTER_NAME="<new name>";
GIT_AUTHOR_NAME="<new name>";                                    GIT_COM-
MITTER_EMAIL="<new - email>";                                   GIT_AU-
THOR_EMAIL="<new - email>";
git commit-tree "$@";
else
git commit-tree "$@";
fi" HEAD
```

Authenticating to Your Git Repos

Authenticating to Your Git Repos

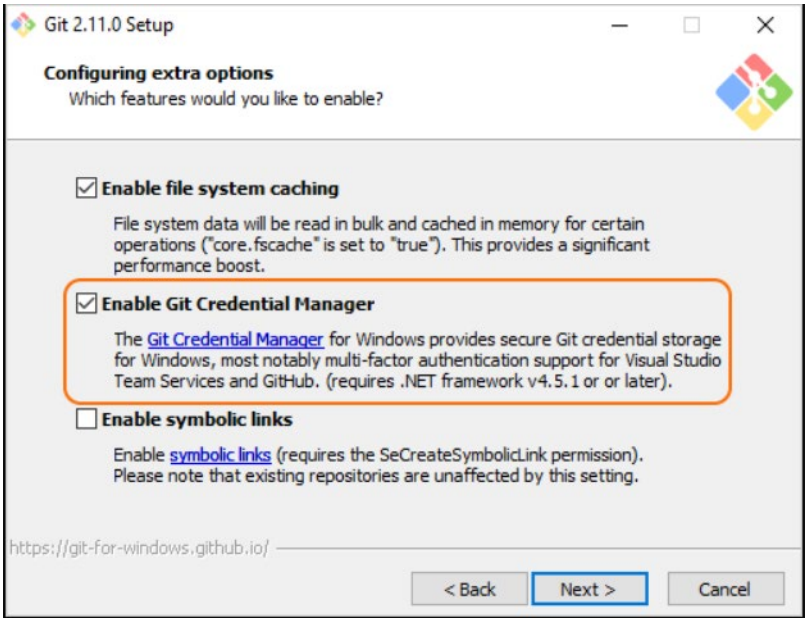
Git Credential Managers simplify authentication with your Azure DevOps Services/TFS Git repos. Credential Managers let you use the same credentials that you use for the Azure DevOps Services/TFS web portal and support multi-factor authentication through Microsoft Account (MSA) or Azure Active Directory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

Azure DevOps Services provides IDE support for MSA and Azure AD authentication through Team Explorer in Visual Studio, IntelliJ and Android Studio with the Azure Repos Plugin for IntelliJ, and Eclipse (with the Team Explorer Everywhere plug-in). If your environment doesn't have an integration available, configure your IDE with a **Personal Access Token**[9] or **SSH**[10] to connect with your repos.

Install the Git Credential Manager

Windows

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



macOS and Linux

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

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

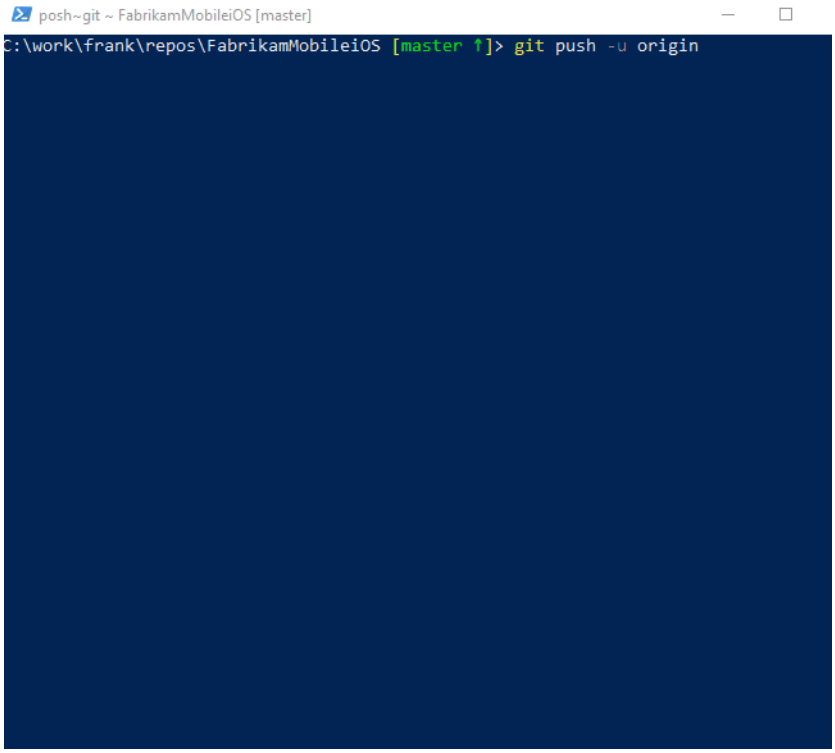
```
> git-credential-manager install
```

9 https://docs.microsoft.com/en-us/azure/devops/organizations/accounts/use-personal-access-tokens-to-authenticate?view=vsts
10 https://docs.microsoft.com/en-us/azure/devops/repos/git/use-ssh-keys-to-authenticate?view=vsts

Demonstration - Git Credential Manager

Using the Git Credential Manager

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



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

Module 1 Review Questions

Module 1 Review Questions

Source Control

What is source control and what are the benefits of source control?

Suggested Answer

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.

Source Control Best Practices

What are some best practices for source control?

Suggested Answer

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

Source Control Types

What are the two basic types of source control and how do they work? When are the benefits and usage cases for each?

Suggested Answer

Distributed and Centralized. Distributed - Every developer clones a copy of a repository and has the full history of the project. Distributed strengths - cross platform, open source, offline support, and history. Distributed is best used for small codebases, evolving open course, distributed teams, and Greenfield projects. Centralized - There is a single central copy of your project and programmers commit their changes to this central copy. Centralized strengths - scaling, permissions, monitoring, file locking. Centralized is best used for large codebases, audit and access, and hard to merge file types.

Module 2 Scaling Git for Enterprise DevOps

How to Structure Your Git Repo

Mono vs Multi Repos

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

Mono-Repo or Multi-Repo?

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

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

The mono-repo view, on the other hand, is that that friction, especially when dealing with more complicated dependency graphs, is much more costly than multi-repo advocates recognize and that the productivity gains to be had by letting different teams go their own way aren't really all that significant: While it may be the case that some teams will find a locally optimal way of working, it is also likely that their gains will be offset by other teams choosing a sub-optimal way of working. By putting all your eggs

in the one basket of the mono-repo you can then afford to invest in watching that basket carefully. Clearly the friction of having to make changes in other repos or, worse, having to wait for other teams to make changes for you, is largely avoided in a mono-repo because anyone can (and is in fact encouraged) to change anything. If you find a bug in a library, you can fix it and get on with your life with no more friction than if you had found a bug in your own code.

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

Git Branching Workflows

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.

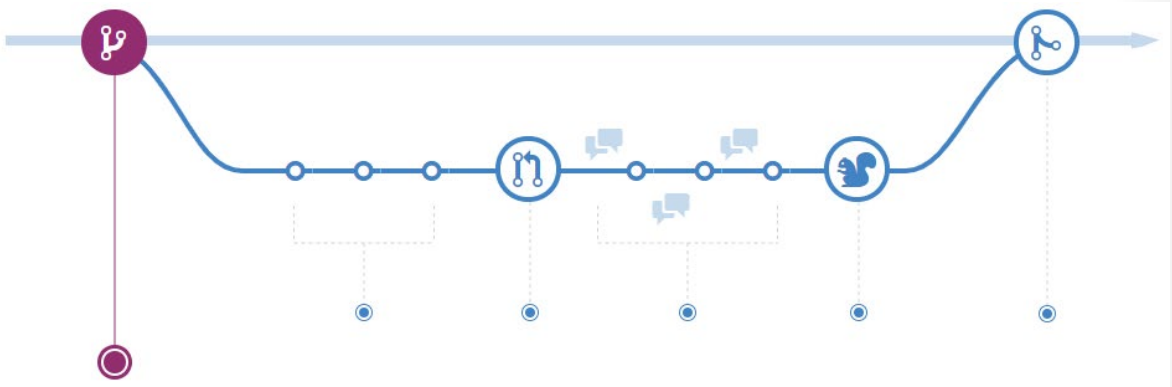
Encapsulating feature development also makes it possible to leverage pull requests, which are a way to initiate discussions around a branch. They give other developers the opportunity to sign off on a feature before it gets integrated into the official project. Or, if you get stuck in the middle of a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

How it works

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

Create a branch



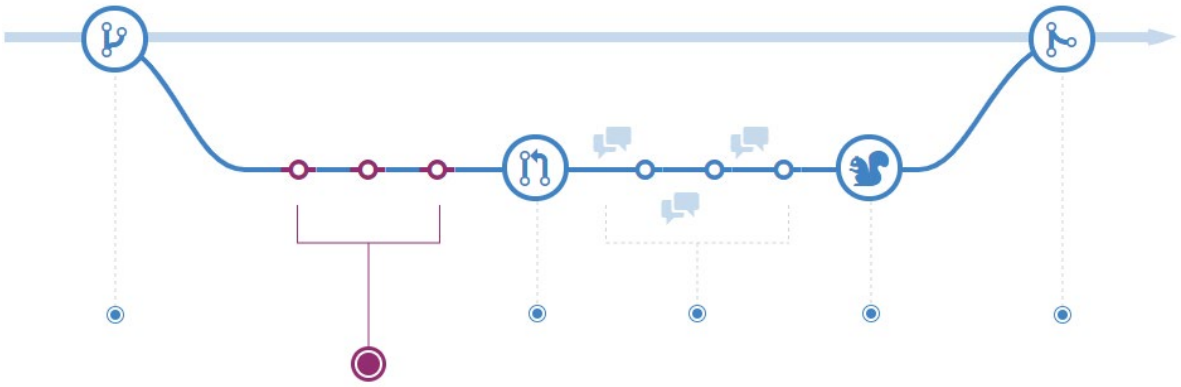
When you're working on a project, you're going to have a bunch of different features or ideas in progress at any given time – some of which are ready to go, and others which are not. Branching exists to help you manage this workflow.

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

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

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

Add commits

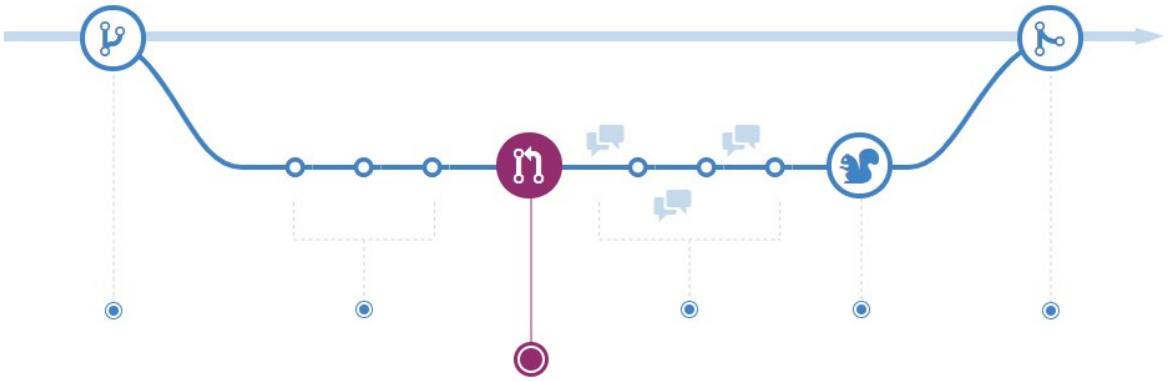


Once your branch has been created, it's time to start making changes. Whenever you add, edit, or delete a file, you're making a commit, and adding them to your branch. This process of adding commits keeps track of your progress as you work on a feature branch.

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

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

Open a Pull Request

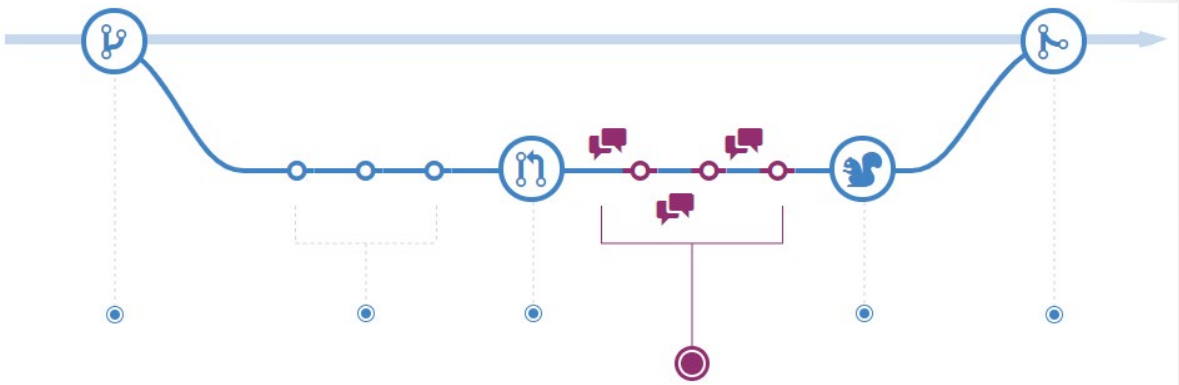


Pull Requests initiate discussion about your commits. Because they're tightly integrated with the underlying Git repository, anyone can see exactly what changes would be merged if they accept your request.

You can open a Pull Request at any point during the development process: when you have little or no code but want to share some screenshots or general ideas, when you're stuck and need help or advice, or when you're ready for someone to review your work. By using @mention system in your Pull Request message, you can ask for feedback from specific people or teams, whether they're down the hall or ten time zones away.

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

Discuss and review your code

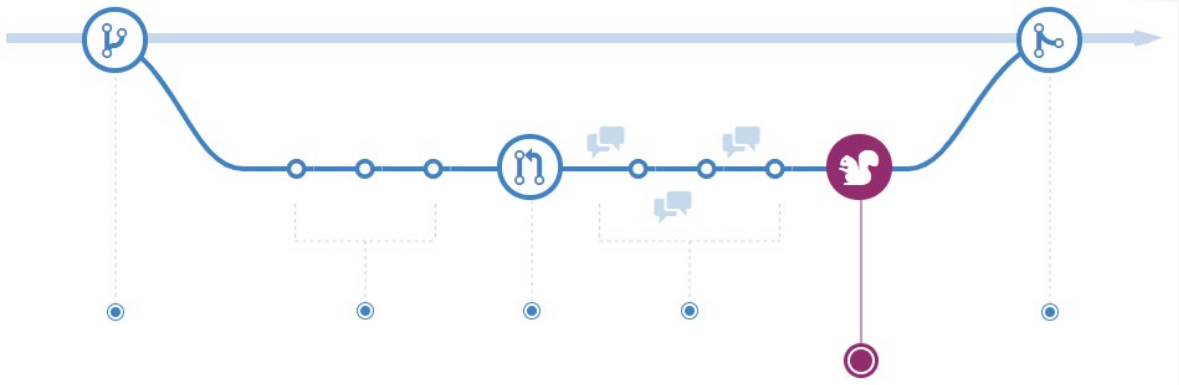


Once a Pull Request has been opened, the person or team reviewing your changes may have questions or comments. Perhaps the coding style doesn't match project guidelines, the change is missing unit tests, or maybe everything looks great and props are in order. Pull Requests are designed to encourage and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

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

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

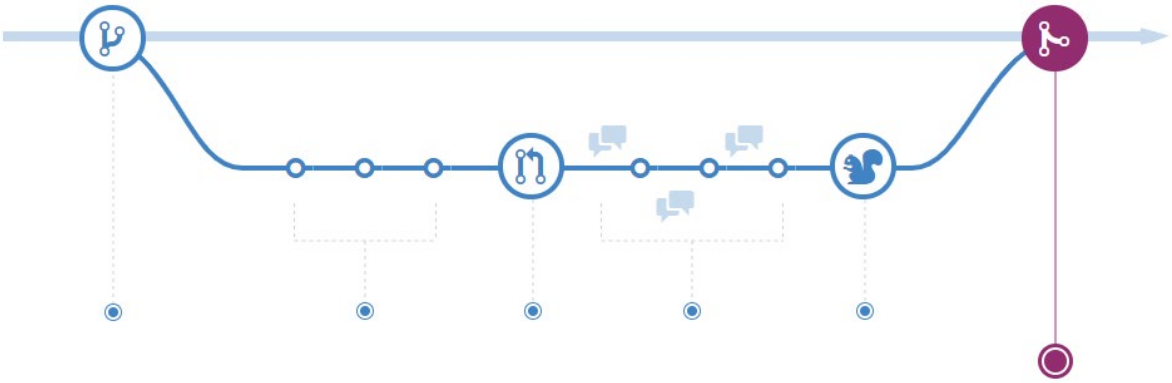
Deploy



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

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

Merge



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

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

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

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

Branching Workflow Types

What is a successful Git branch workflow?

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

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

Common branch workflows

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

Feature branching

Feature Branching 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 nvie**[1]. 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.

Git-Branching model for continuous delivery

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

Getting ready

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

- The master branch:
  - The master branch is the only way to release anything to production.
  - The master branch should always be in a ready-to-release state.
  - Protect the master branch with branch policies.
  - Any changes to the master branch flow through pull requests only.
  - Tag all releases in the master branch with Git tags.
- The feature branch:
  - Use feature branches for all new features and bug fixes.
  - Use feature flags to manage long-running feature branches.
  - Changes from feature branches to the master only flow through pull requests.
  - Name your feature to reflect their purpose.

List of branches...

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

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

```
hotfix/description
```

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

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

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

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

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

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

How to do it...

1. After you've cloned the master branch into a local repository, create a 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

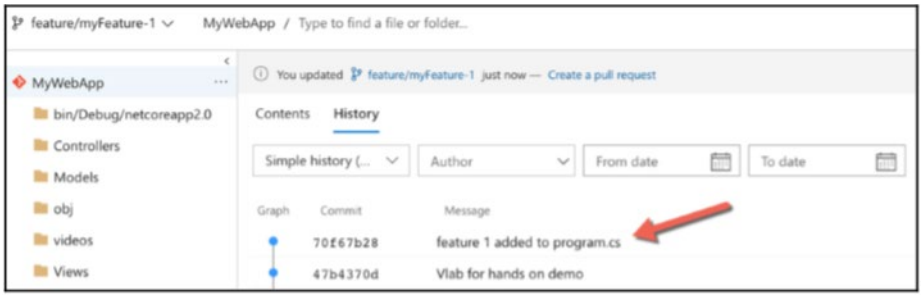

myWebApp> git add .
myWebApp> git commit -m "Feature 1 added to Program.cs"

[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:

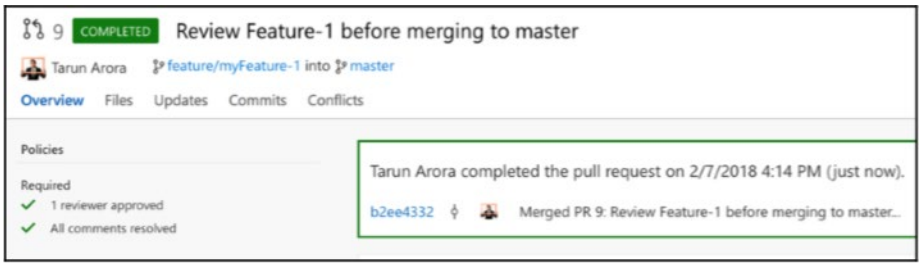


5. Create a new pull request (using the Azure DevOps CLI) to review the changes in the feature-1 branch:

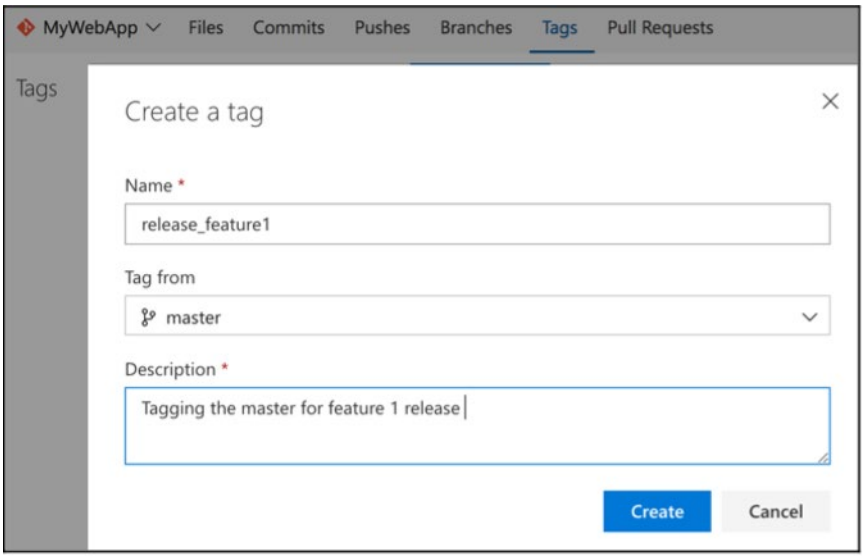
```
> az code pr create --title "Review Feature-1 before merging to master"
--work-items 38 39 `
          -d "#Merge feature-1 to master" `
          -s feature/myFeature-1 -t master -r myWebApp -p
   $prj -i $i
```

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

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



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



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

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

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


myWebApp> git checkout feature/myFeature-2

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

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

```
public class Program
{
    // Editing the same line (file from feature-2 branch)
    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 code pr create --title "Review Feature-2 before merging to master"
--work-items 40 42 `
            -d "#Merge feature-2 to master" `
            -s feature/myFeature-2 -t master -r myWebApp -p
   $prj -i $1
```

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

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

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

```
// Editing this file from [feature-fof branch]
public static void Main(string[] args)
{
    BuildWebHost(args).Run();
}

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

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

```
myWebApp> git add .

myWebApp> git commit -m "Adding FOF changes"

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

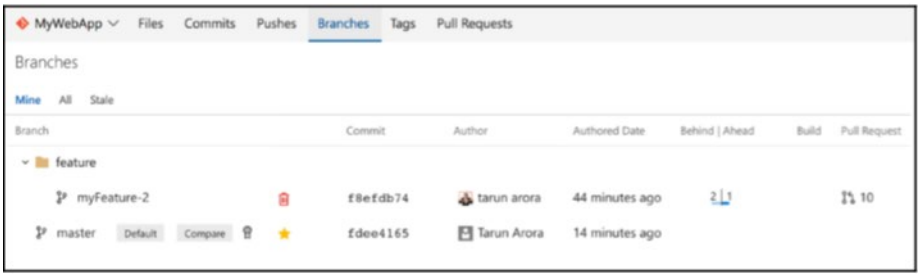
To https://dev.azure.com/Geeks/PartsUnlimited/_git/MyWebApp * [new branch]
fof/bug-1 -> fof/bug-1 Branch fof/bug-1 set up to track remote branch fof/
bug-1 from origin
```

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

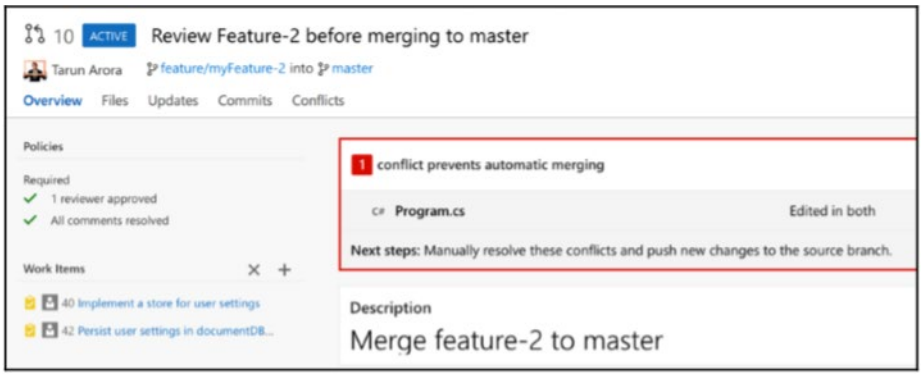
```
> az code pr create --title "Review Bug-1 before merging to master" --work-
items 100 `
            -d "#Merge Bug-1 to master" `
            -s fof/Bug-1 -t master -r myWebApp -p
   $prj -i $i
```

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

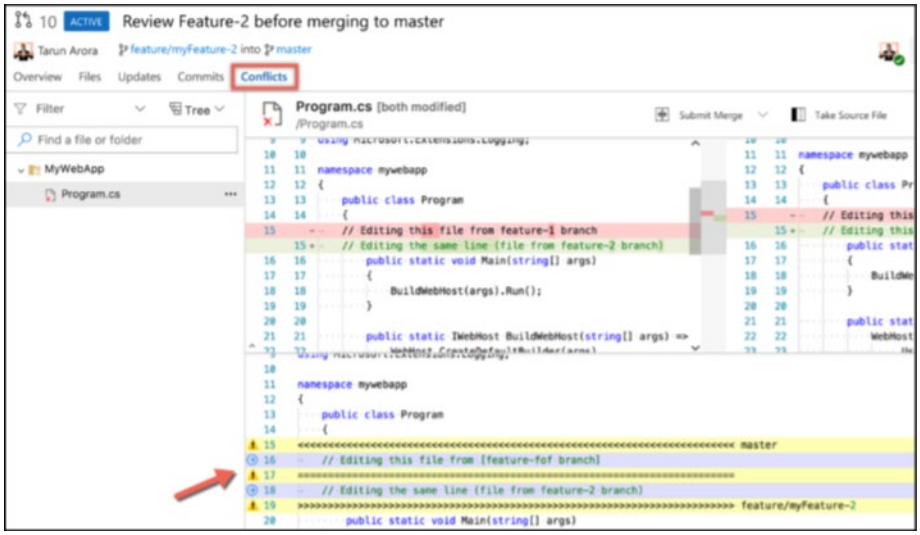
With the critical bug fix out of the way, let's go back to the review of the feature-2 pull request. The branches page makes it clear that the feature/myFeature-2 branch is one change ahead of the master and two changes behind the master:



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.

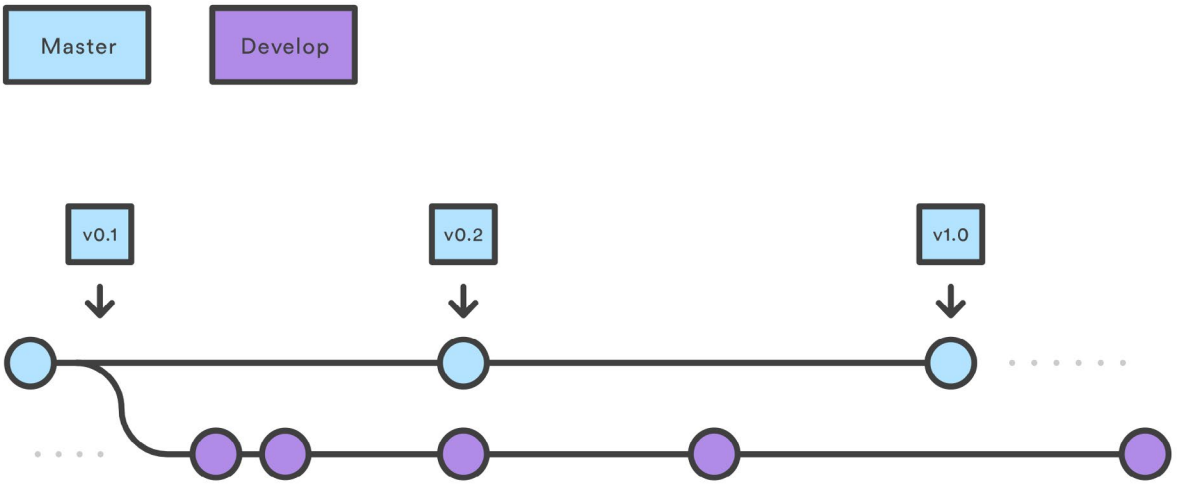
In addition to the abstract Gitflow Workflow idea, there is a more tangible git-flow toolset available which integrates with Git to provide specialized Gitflow Git command line tool extensions.

Getting Started

Gitflow is really just an abstract idea of a Git workflow. This means it dictates what kind of branches to set up and how to merge them together. We will touch on the purposes of the branches below. The git-flow toolset is an actual command line tool that has an installation process. The installation process for git-flow is straightforward. Packages for git-flow are available on multiple operating systems. On OSX

systems, you can execute brew install git-flow. On windows you will need to download and install git-flow. After installing git-flow you can use it in your project by executing git flow init. Git-flow is a wrapper around Git. The git flow init command is an extension of the default git init command and doesn't change anything in your repository other than creating branches for you.

How it works



Develop and Master Branches

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

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

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

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

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

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

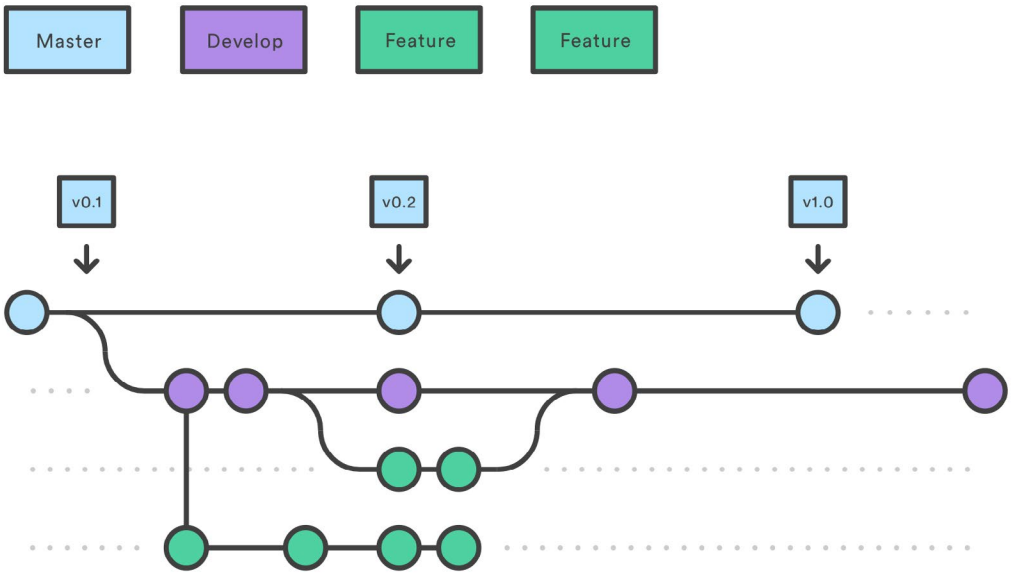
How to name your supporting branch prefixes?
Feature branches? [feature/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []

$ git branch
```

```
* develop
  master
```

Feature Branches

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



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

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

Creating a feature branch
Without the git-flow extensions:

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

When using the git-flow extension:

```
git flow feature start feature_branch
```

Continue your work and use Git like you normally would.

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

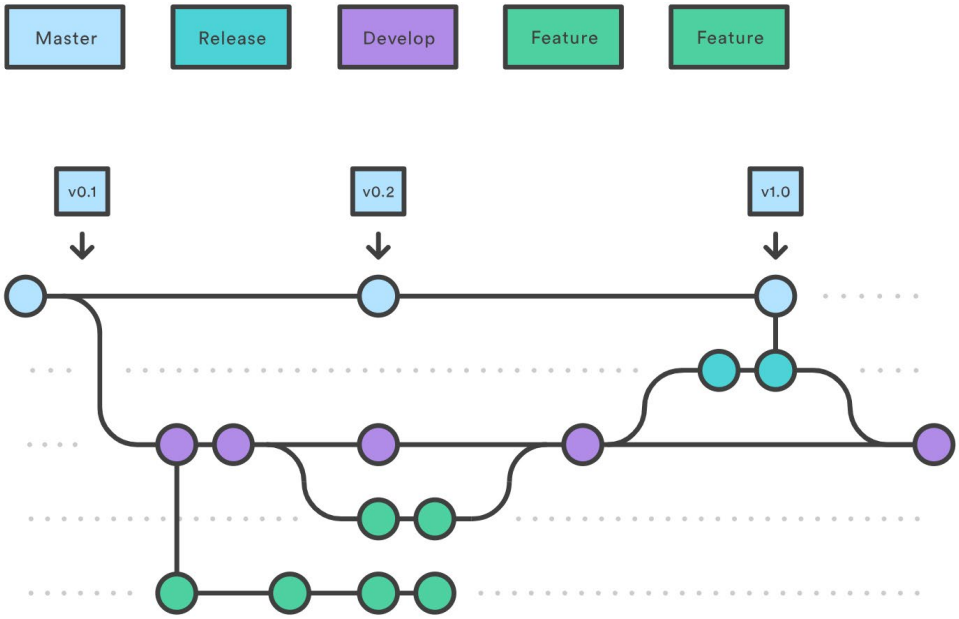
Without the git-flow extensions:

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

Using the git-flow extensions:

```
git flow feature finish feature_branch
```

Release Branches



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

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

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

Without the git-flow extensions:

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

When using the git-flow extensions:

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

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

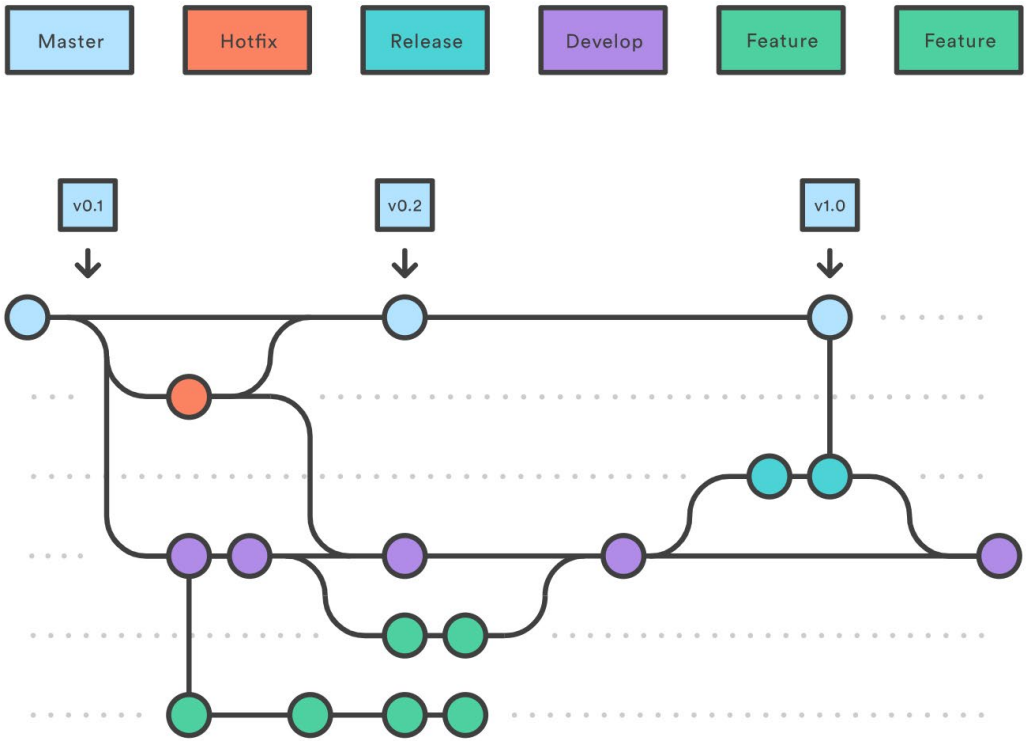
To finish a release branch, use the following methods:

Without the git-flow extensions:

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

Or with the git-flow extension:

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



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

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

Without the git-flow extensions:

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

When using the git-flow extensions:

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

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

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

Some key takeaways to know about Gitflow are:

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

The overall flow of Gitflow is:

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

Forking Branch Workflow

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

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

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

How it works

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

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

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

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

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

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

Forking vs cloning

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

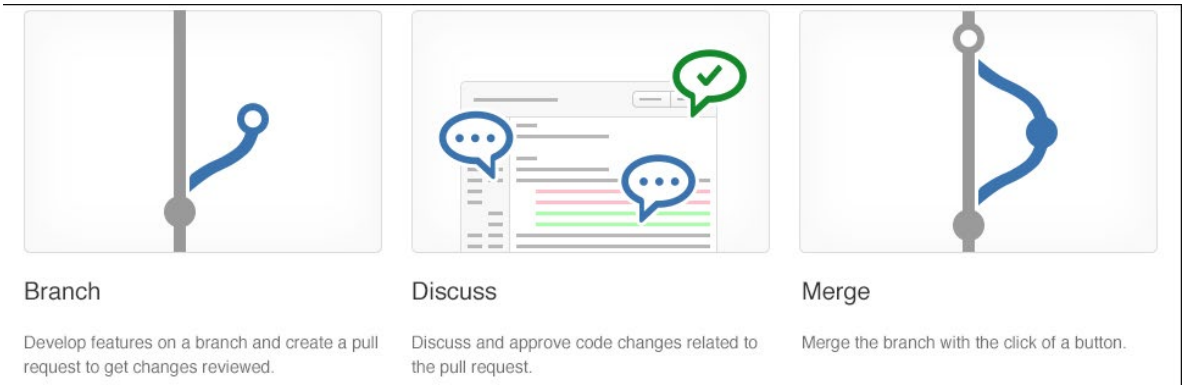
Collaborating with Pull Requests

Collaborating with Pull Requests

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

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

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



Get your code reviewed

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

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

Give great feedback

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

- Have the right people review the pull request
- Make sure that reviewers know what the code does
- Give actionable, constructive feedback
- Reply to comments in a timely manner

When assigning reviewers to your pull request, make sure you select the right set of reviewers. You want reviewers that will know how your code works, but try to also include developers working in other areas

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

Protect branches with policies

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

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

Azure Repos Collaborating with Pull Requests

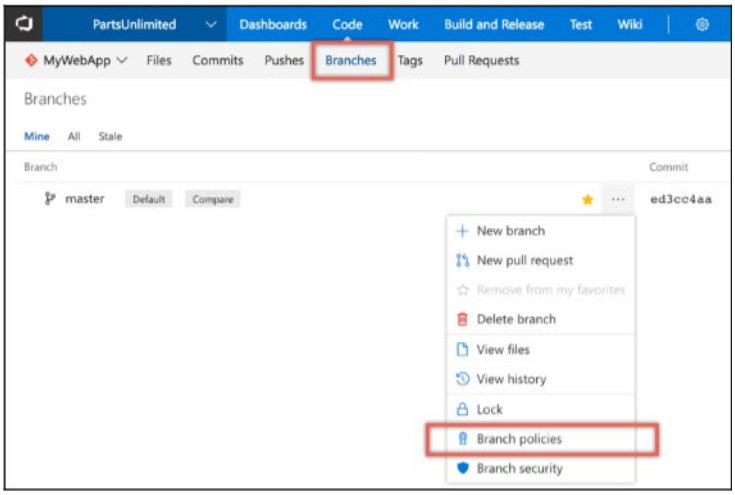
Code issues that are found sooner are both easier and cheaper to fix, therefore development teams strive to push code quality checks as far left into the development process as possible. As the name suggests, branch policies give you a set of out-of-the-box policies that can be applied to the branches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

Getting ready

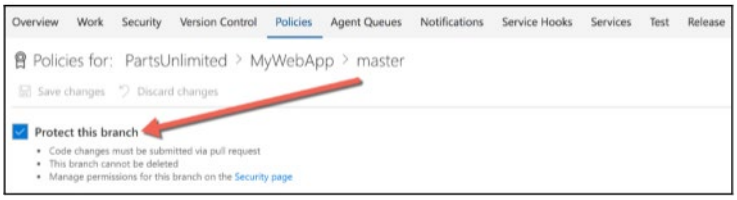
The out-of-the-box branch policies include several policies, such as build validation and enforcing a merge strategy. In this recipe, we'll only focus on the branch policies needed to set up a code-review workflow.

How to do it...

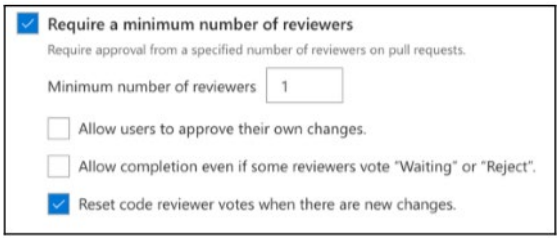
1. Open the branches view for the myWebApp Git repository in the parts unlimited team portal. Select the master branch, and from the pull-down context menu choose Branch policies:



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

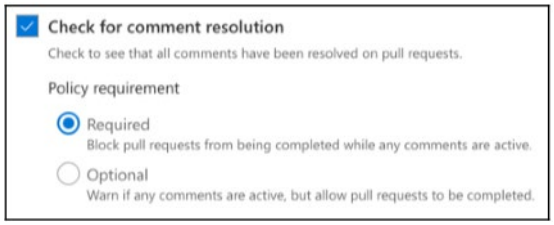


3. This presents the out-of-the-box policies. Check this option to select a minimum number of reviewers. Set the minimum number of reviewers to 1 and check the option to reset the code reviewer's votes when there are new changes:

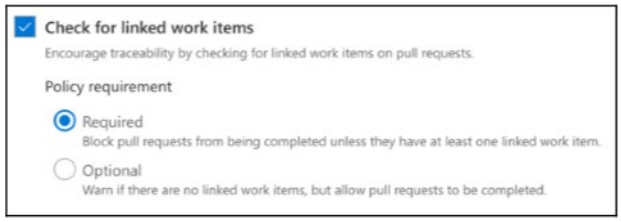


The Allow users to approve their own changes option allows the submitter to self-approve their changes. This is OK for mature teams, where branch policies are used as a reminder for the checks that need to be performed by the individual.

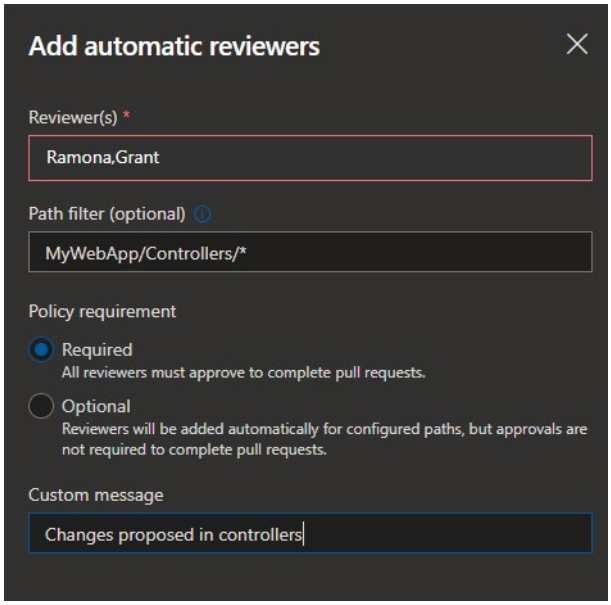
4. Use the review policy in conjunction with the comment-resolution policy. This allows you to enforce that the code review comments are resolved before the changes are accepted. The requester can take the feedback from the comment and create a new work item and resolve the changes, this at least guarantees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



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

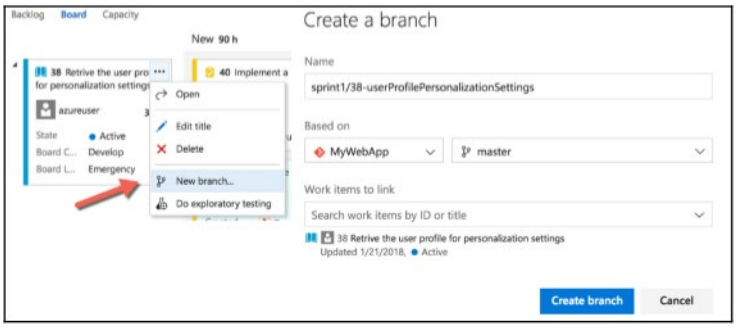


6. Select the option to automatically add code reviewers when a pull request is raised. You can map which reviewers are added based on the area of the code being changed:



How it works...

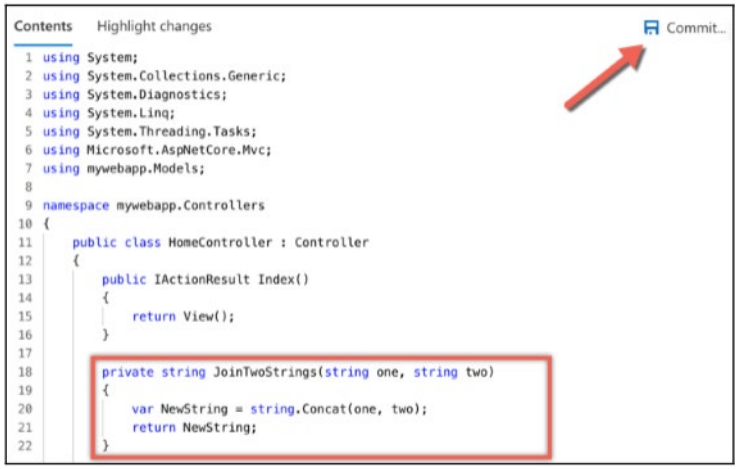
With the branch policies in place, the master branch is now fully protected. The only way to push changes to the master branch is by first making the changes in another branch and then raising a pull request to trigger the change-acceptance workflow. From one of the existing user stories in the work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



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

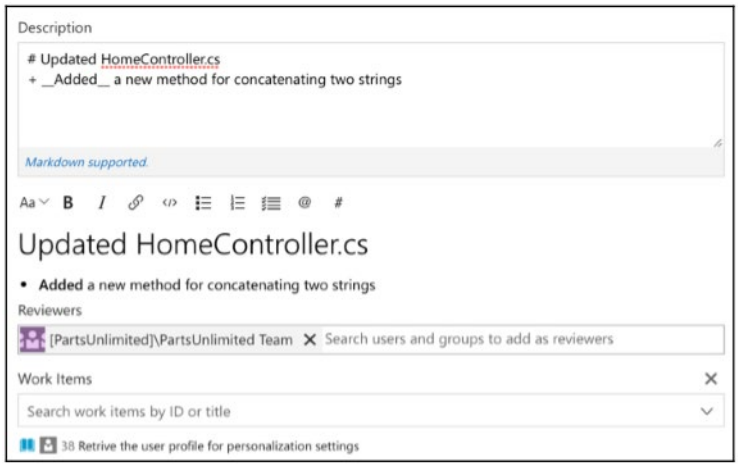
With the newly-created branch selected in the web portal, edit the HomeController.cs file to include the following code snippet and commit the changes to the branch. In the image below you'll see that after editing the file, you can directly commit the changes by clicking the commit button.

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

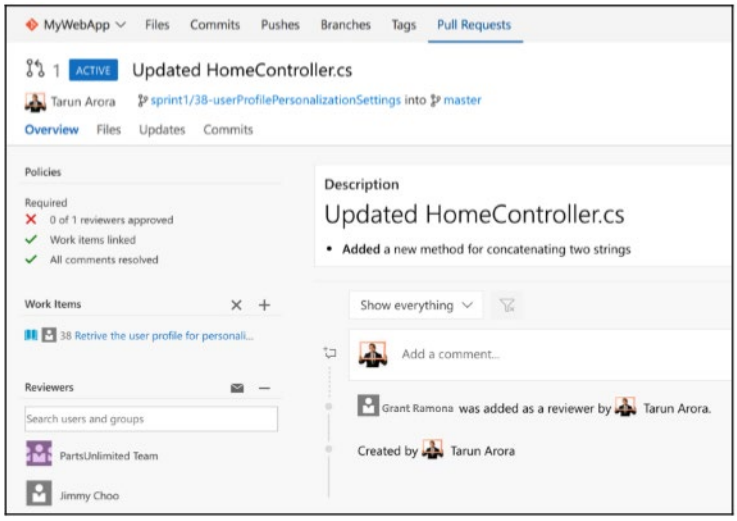


The code editor in web portal has several new features in Azure DevOps Server 2018, such as support for bracket matching and toggle white space.You can load the command palette by pressing . Among many other new options, you can now toggle the file using a file mini-map, collapse and expand, as well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

Lab - Code Review with Pull Requests

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.

- Exercise 6: Working with pull requests
- Exercise 7: Managing repositories

Version Controlling with Git in Azure Repos - **https://www.azuredevopslabs.com/labs/azuredevops/git/**

Why Care About GitHooks

Why Care about GitHooks

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

GitHooks

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

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

Practical use cases for using GitHooks

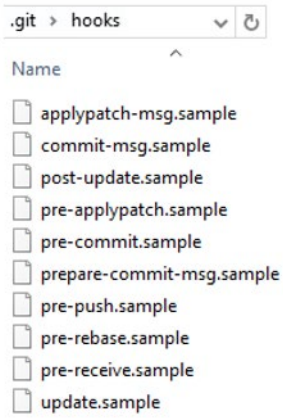
Since GitHooks simply execute the scripts on the specific event type they are called on, you can do pretty much anything with GitHooks. Some examples of where you can use hooks to enforce policies, ensure consistency, and control your environment…

- In Enforcing preconditions for merging
- Verifying work Item Id association in your commit message
- Preventing you & your team from committing faulty code
- Sending notifications to your team's chat room (Teams, Slack, HipChat, etc)

So where do I start?

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

executed. If your pre-commit script exits with a 0 (zero), you commit successfully, otherwise the commit fails.



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

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

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

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

PreCommit GitHook to scan commit for keywords

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

```
#!C:/Program\ Files/Git/usr/bin/sh.exe
matches=$(git diff-index --patch HEAD | grep '^+' | grep -Pi 'password|key-
word2|keyword3')
if [ ! -z "$matches" ]
then
    cat <<\EOT
Error: Words from the blacklist were present in the diff:
EOT
    echo $matches
    exit 1
fi
```

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

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

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

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

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

Server side GitHooks with Azure Repos

So far we have looked at the client side GitHooks on Windows, Azure Repos also exposes server side hooks. Azure DevOps uses the same mechanism itself to create Pull requests... You can read more about it **here**[2]

Interested in learning more about GitHooks, check out another useful usecase for applying GitHooks **here**[3]

GitHooks in Action

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

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

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

Getting ready

Let's start by exploring client-side Git hooks. Navigate to the repo.git\hooks directory – you'll find that there a bunch of samples, but they are disabled by default. For instance, if you open that folder, you'll find a file called precommit.sample. To enable it, just rename it to pre-commit by removing the .sample

2 https://docs.microsoft.com/en-gb/azure/devops/service-hooks/events?view=vsts#code-pushed
3 https://www.visualstudiogeeks.com/DevOps/UsingPowerShellForGitHooksWithVstsGitOnWindows

extension and make the script executable. When you attempt to commit using git commit, the script is found and executed. If your pre-commit script exits with a 0 (zero), you commit successfully, otherwise, the commit fails.

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

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

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

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

How to do it...

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

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

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

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

How it works...

In the script, Git diff-index is used to identify the code increment being committed. This increment is then compared against the list of specified keywords. If any matches are found, an error is raised to block the commit; the script returns an error message with the list of matches. In this case, the pre-commit script doesn't return 0 (zero), which means the commit fails.

There's more...

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

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

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

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

You can also use PowerShell scripts in your Git hooks – refer to the walkthrough on how to implement it here: https://www. visualstudiogeeks.com/DevOps/ UsingPowerShellForGitHooksWithVstsGitOnWindows.

See also

So far, we have looked at the client-side Git hooks on Windows, Azure DevOps Server also exposes server-side hooks. Azure DevOps Server uses the same mechanism to create pull requests. You can read more about the git.push server-side event here: https://docs. microsoft.com/en-us/vsts/service-hooks/events#git.push.

Fostering Internal Open Source

Fostering Internal Open Source

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

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

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

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

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

Implementing the Fork Workflow

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

What's in a fork

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

Sharing code between forks

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

Choosing between branches and forks

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

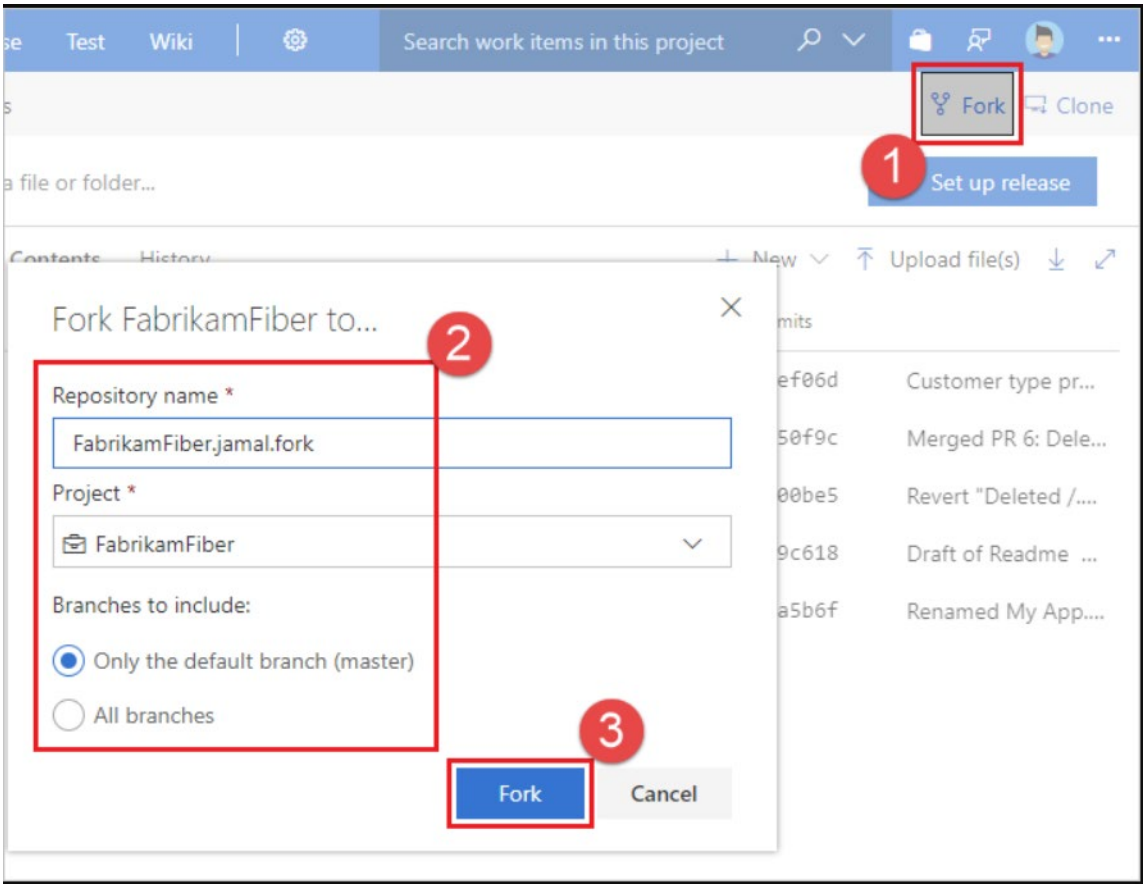
If your repository has a large number of casual or infrequent committers (similar to an open source project), we recommend the forking workflow. Typically only core contributors to your project have direct commit rights into your repository. You should ask collaborators from outside this core set of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

The forking workflow

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

Create the fork

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



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

Clone your fork locally

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

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

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

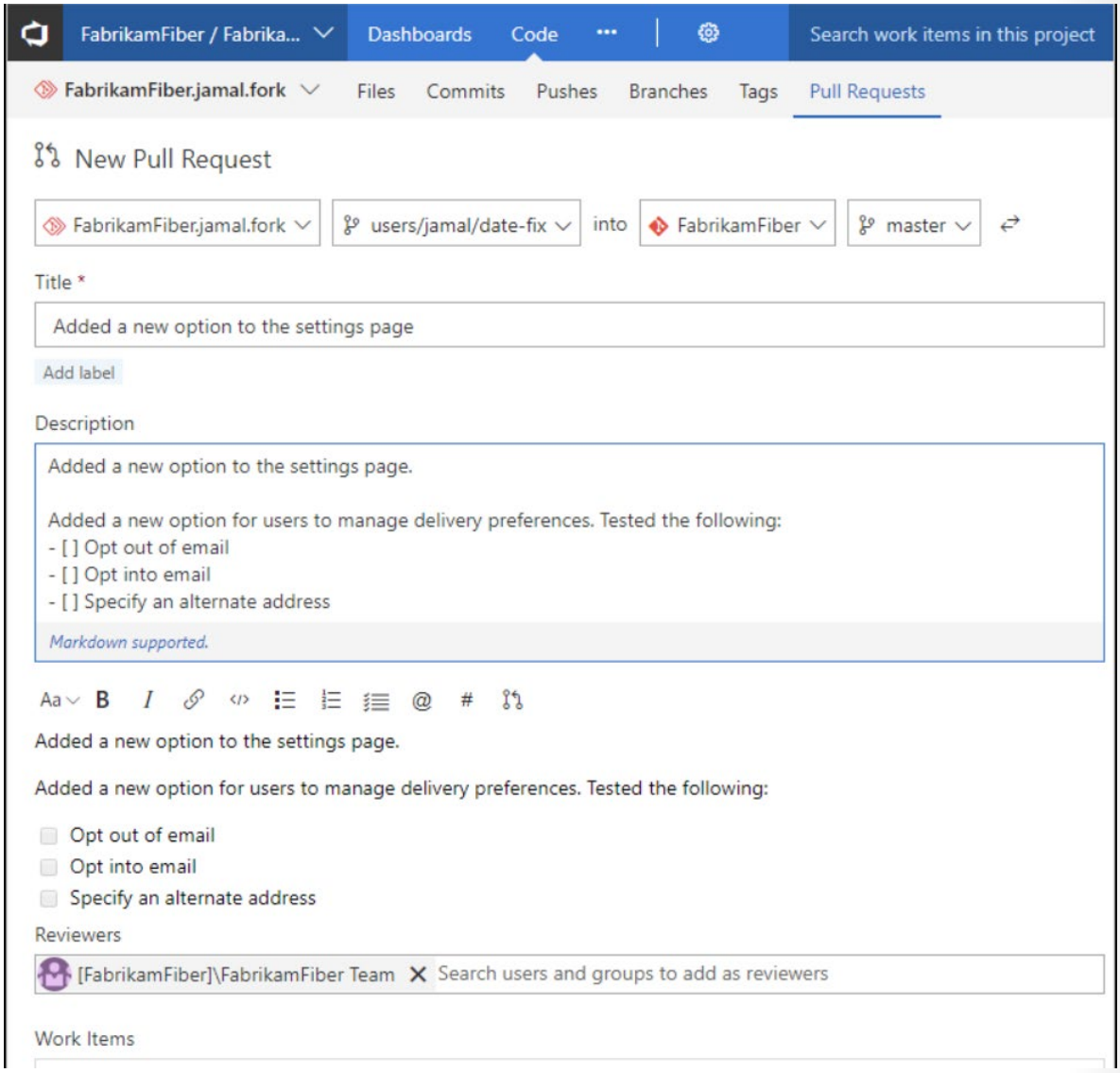
Make and push changes

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

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

Create and complete a PR

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



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

Sync your fork to latest

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

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

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

Inner Source with Forks

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

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

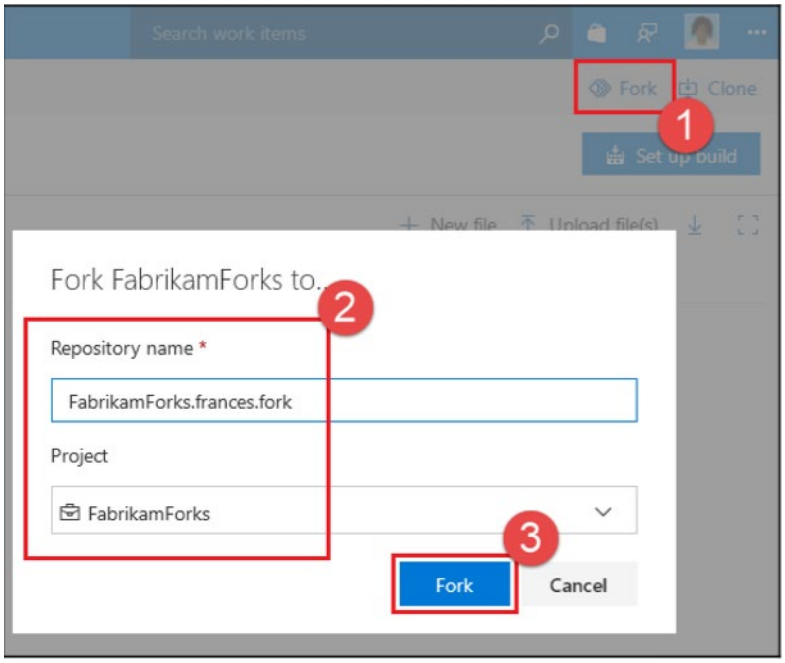
Software teams are encouraged to contribute to all projects internally, not just their own software projects. Forks are a great way to foster a culture of inner open source. Forks are a recent addition to the Azure DevOps Server-hosted Git repositories. In this recipe, we'll learn how to fork an existing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

Getting ready

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

How to do it...

1. Choose the Fork button (1), then choose the project where you want the fork to1. be created (2). Give your fork a name and choose the Fork button (3):

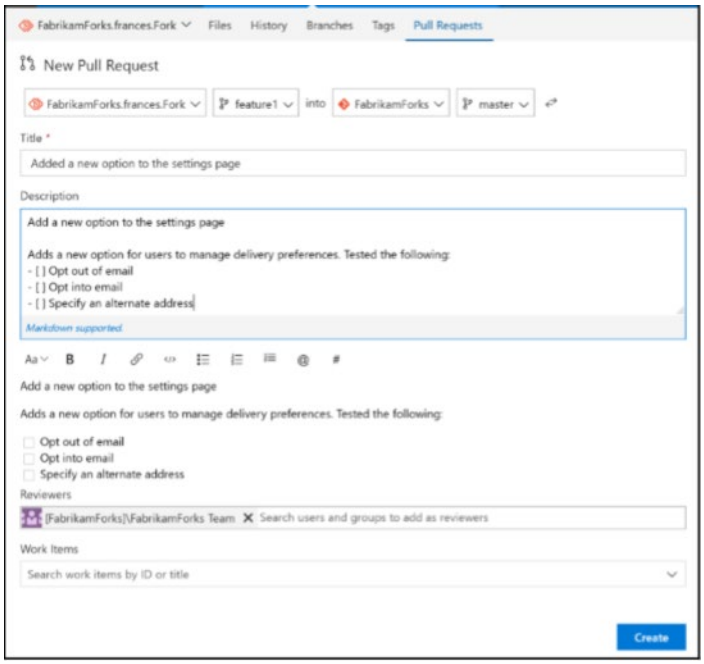


2. Once your fork is ready, clone it using the command line (https://docs.2. microsoft.com/en-us/vsts/git/tutorial/clone?tabs=command-line) or an IDE, such as Visual Studio (https://docs.microsoft.com/en-us/vsts/git/ tutorial/clone). The fork will be your origin remote. For convenience, you'll want to add the upstream repository (where you forked from) as a remote named upstream. On the command line, type the following

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

3. It's possible to work directly in the master – after all, this fork is your personal3. copy of the repo. We recommend you still work in a topic branch, though. This allows you to maintain multiple independent workstreams simultaneously. Also, it reduces confusion later when you want to sync changes into your fork. Make and commit your changes as you normally would. When you're done with the changes, push them to origin (your fork).
4. Open a pull request from your fork to the upstream. All the policies, required4. 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:



5. When your PR is accepted into upstream, you'll want to make sure your fork5. 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.

Git Version

Git Versioning

The goal of adding a new dependency to your code is to avoid reinventing the wheel. If there is code already available that does what you want, normally you would prefer to re-use it (there are exceptions, though) rather than investing your time in re-writing new code to solve the same task. On the other hand, once your software project has matured over time and is ready for production it may end up being a dependency for other software projects as well.

Initially, re-using software packages as much as possible looks like the best way forward to save time and effort in the long run; and it is indeed true. However, creating dependencies with other software packages also brings its own disadvantages, specially when the number of dependencies becomes larger and larger. Grouping off-the-shelf functions into software packages and defining dependencies between them has been traditionally at the core of software development.

If you're building libraries, products or any other software system, versioning is usually a pretty big deal. It's the only way to determine what version of that library, product or system you're looking at. Before an organization settles on a versioning strategy, many discussions have been held on what constitutes a major release versus a minor release, how to version the component from a marketing perspective, and how to deal with bug fixes. In addition to that, if that software system involves a library or framework, or just component, then you'd be pretty interested to know when an update to that component involves breaking changes.

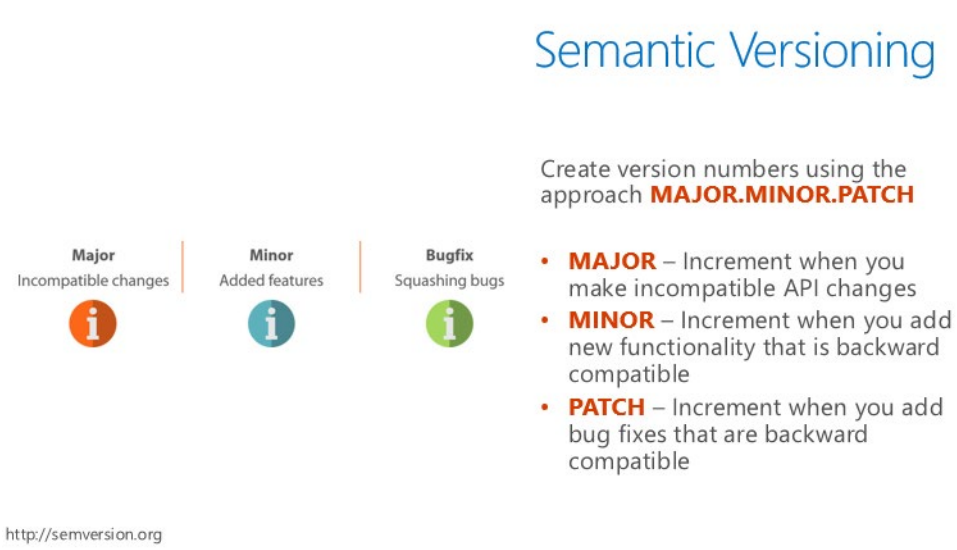
From the version of an app you should, for instance, be able to guess whether it’s a bugfix or it has introduced new features. Without properly set version codes you could totally lose your compass on what’s happening. So, when you set the version of your software always be careful and remember that, given a major.minor.patch version number, you should increment:

- patch when you make bug fixes
- minor when you introduce new features
- major when you make changes that break backwards-compatibility or completely change the user experience

That being said, in the daily practice, if you want to start with a development phase before going to production the simplest thing could be to start with a 0.1.0 version number and increment minors at each new development release. Also keep in mind that if you’re working on a production project it’s always a good idea to start with a 1.0.0 version code.

GitVersion Docs

GitVersion is a tool to help you achieve Semantic Versioning on your project. **Semantic Versioning is all about releases, not builds.** This means that the version only increases after you release, this directly conflicts with the concept of published CI builds. When you release the next version of your library/app/website/whatever you should only increment major/minor or patch then reset all lower parts to 0, for instance given 1.0.0, the next release should be either 2.0.0, 1.1.0 or 1.0.1. Bumping one of the version components by more than 1 in a single release means you will have gaps in your version number, which defeats the purpose of SemVer.



Fortunately, the open-source community has solved both of these problems for us. First, we have semantic versioning, which unambiguously defines how to update your version when you do hot fixes and patches, minor back-wards compatible improvements or breaking changes. They even define how you should post-fix your version numbers to denote pre-releases and build numbers. Assuming all decent software project are using Git these days, then the other problem is solved by following the Git branching workflows we've discussed earlier.

Wouldn't it be cool if some kind of tool existed that used the names of the branches and the tags on master (representing the production releases) to generate the proper version numbers for you? Well, once again, the open-source community comes to the rescue. Jake Ginnivan and the guys from ParticularLabs, the company behind NServiceBus, have build just that and named it GitVersion. So let's see how that works. GitVersion describes in detail on what branch you do your main development work, how you stabilize an upcoming release and how you track production releases.

GitVersion is able to calculate the version based on the state of your git repository. This requires some getting used to, but GitVersion is like a pure function that calculates the semantic version of any given commit based on three things:

- the nearest tag
- the commit messages between this commit and the nearest tag
- the name of the branch

It can be configured with a YAML file and it supports branching models like GitFlow and GitHubFlow (also called Feature branch workflow). Lets see how GitVersion calculates the semantic version.

GitVersion

In the world of software management there exists a dreaded place called “dependency hell.” The bigger your system grows and the more packages you integrate into your software, the more likely you are to find yourself, one day, in this pit of despair.

In systems with many dependencies, releasing new package versions can quickly become a nightmare. If the dependency specifications are too tight, you are in danger of version lock (the inability to upgrade a package without having to release new versions of every dependent package). If dependencies are specified too loosely, you will inevitably be bitten by version promiscuity (assuming compatibility with

more future versions than is reasonable). Dependency hell is where you are when version lock and/or version promiscuity prevent you from easily and safely moving your project forward.

As a solution to this problem, the open source world has invented Semantic Versioning (SemVer). SemVer introduces conventions about breaking changes into our version numbers so we can safely upgrade dependencies without fear of unexpected, breaking changes while still allowing us to upgrade downstream libraries to get new features and bug fixes. The convention is quite simple:

- {major}.{minor}.{patch}-{tag}+{buildmetadata}
- {major} is only incremented if the release has breaking changes (includes bug fixes which have breaking behavioural changes
- {minor} is incremented if the release has new non-breaking features
- {patch} is incremented if the release only contains non-breaking bug fixes
- {tag} is optional and denotes a pre-release of the version preceeding
- {buildmetadata} is optional and contains additional information about the version, but does not affect the semantic version preceding it.

Only one number should be incremented per release, and all lower parts should be reset to 0 (if {major} is incrememented, then {minor} and {patch} should become 0). For a more complete explaination check out **semver.org**[4] which is the official spec.

Taking this concept, the open source has created gitversion to automate the versioning of git repositories. GitVersion makes it easy to follow semantic versioning in your library by automatically calculating the next semantic version which your library/application is likely to use. In GitFlow the develop branch will bump the minor when master is tagged, while GitHubFlow will bump the patch.

Because one size does not always fit all, GitVersion provides many Variables for you to use which contain different variations of the version. For example SemVer will be in the format {major}.{minor}.{patch}-{tag}, but FullSemVer will also include build metadata: {major}.{minor}.{patch}-{tag}+{buildmetadata}

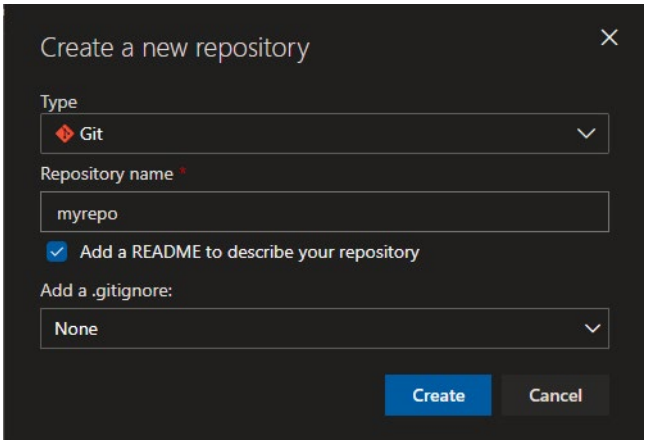
Getting Started

- Install the **GitVersion extension**[5] from the visual studio marketplace in your instance of Azure DevOps
- Install the GitVersion command line tool in your development environment from **chocolatey**[6]

How to do it

1. Create a new git repository in the partsunlimited Azure DevOps project, call the repository myrepo and add a readme markdown file.

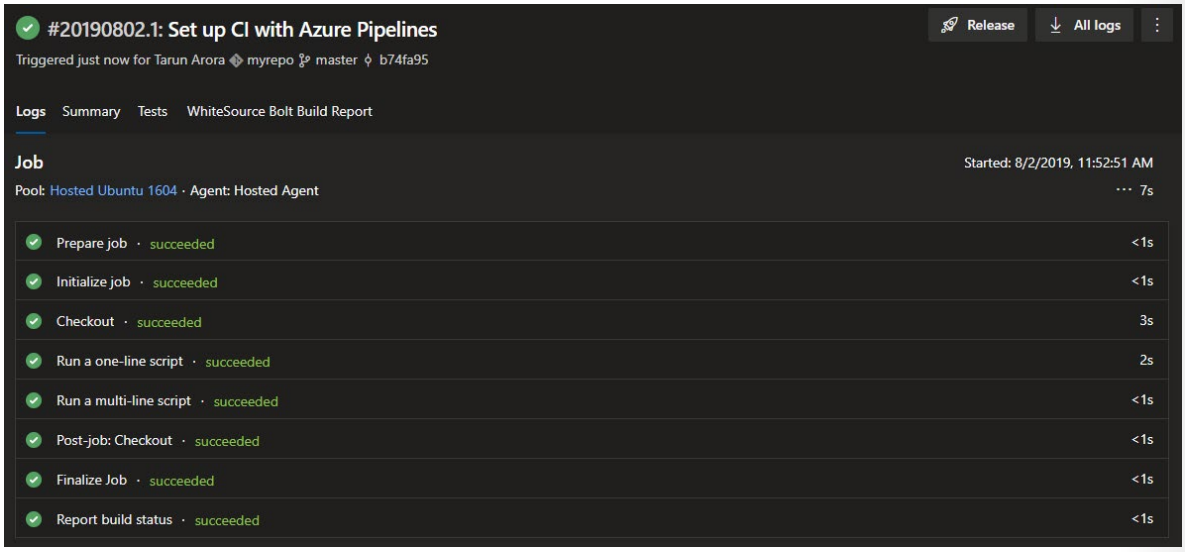
4 https://semver.org/
5 https://marketplace.visualstudio.com/items?itemName=gittools.gitversion&ssr=false#overview
6 https://chocolatey.org/packages/GitVersion.Portable/5.0.0



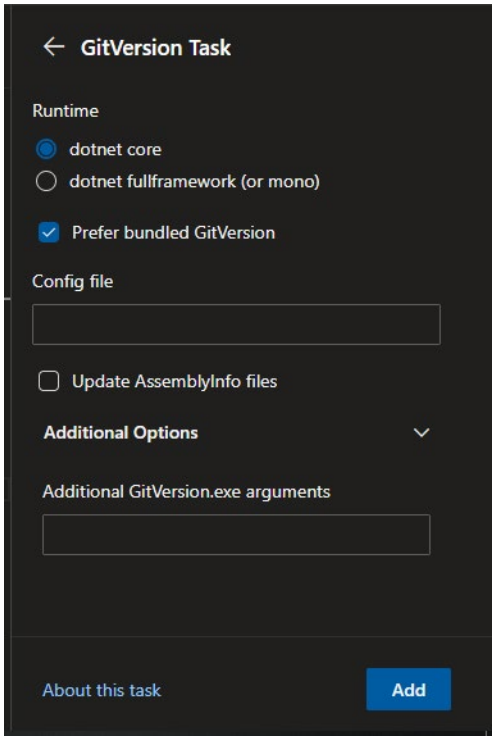
2. Next setup a build pipeline for the git repository myRepo (azure-pipeline.yaml)

```
trigger:
- master
- feature/*

pool:
  vmImage: 'ubuntu-latest'

steps:
- script: echo Hello, world!
  displayName: 'Run a one-line script'

- script: |
    echo Add other tasks to build, test, and deploy your project.
    echo See https://aka.ms/yaml
  displayName: 'Run a multi-line script'
```

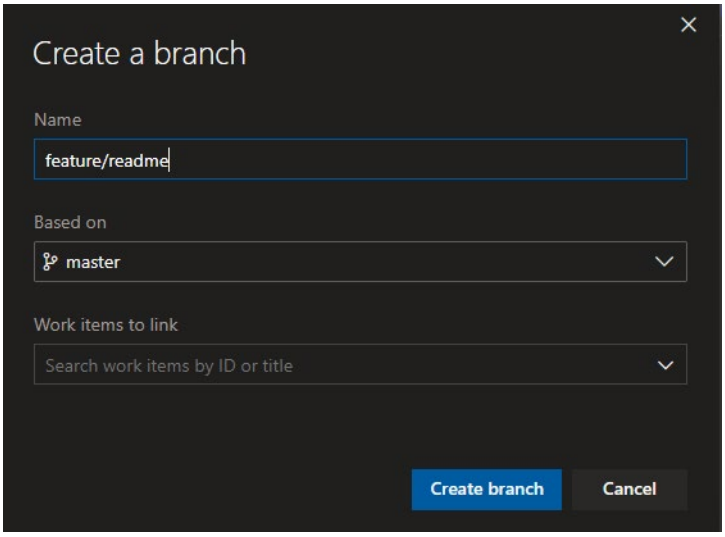
Commit the pipeline and trigger a run of the pipeline



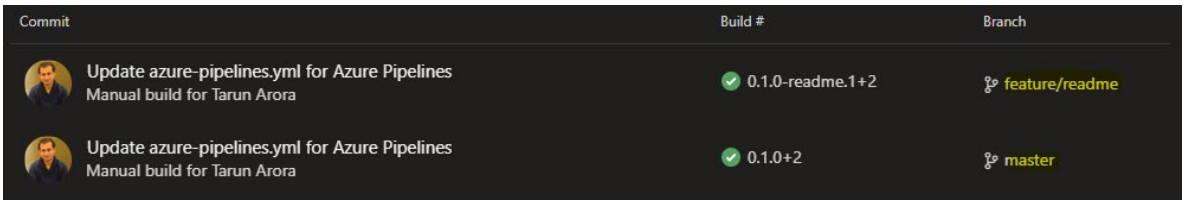
3. Edit the pipeline, search and include the GitVersion task as a new step in the pipeline



4. Based on master create a new branch and call it feature/readme

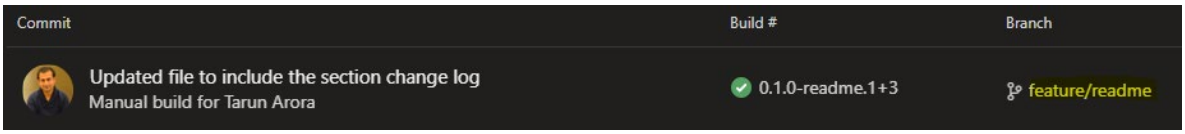


5. Now run a build using the master branch and then the readme branch. You'll see when the build completes, the name of the build is updated to reflect the version of the git repository based on semVer.

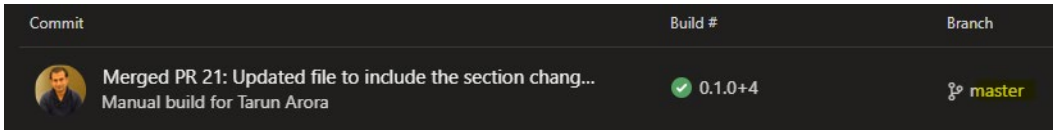


- The version number generated for the master branch reflects that the there are 2 commits in the master branch that are yet to be released.
- The version number generated for the readme branch reflects the name of the readme branch along with the number 2, reflecting that it also has 2 changes that are yet to be released.

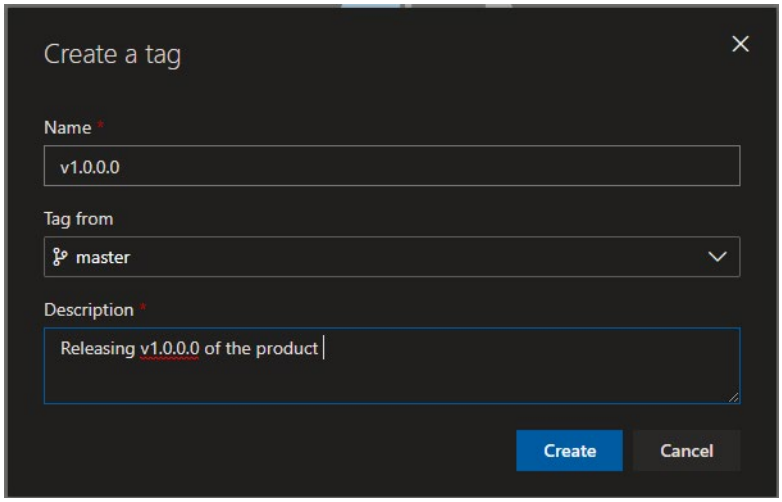
6. Next open up the readme file in the feature/readme branch and make some changes. Once changes have been committed trigger the build against the feature/readme branch again. The version number of the build generated from the feature/readme branch reflects that there are 3 changes that are yet to be released.



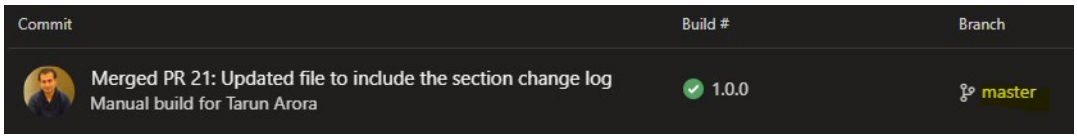
7. Next raise a pull request from feature/readme for master and merge the changes. Trigger a build for the master branch, the version number will reflect that there are now 4 changes in the master branch that are yet to be released.



8. Navigate to the Tags page in Azure Repos in the context of the myRepo git repository. Create a Tag v1.0.0.0 for the master branch.



9. Now trigger the build using the master branch, you'll see that the version number generated reflects 1.0.0 highlighting that all existing commits in the master branch have been tagged as released.



How it works...

GitVersion does a compare between the branches to work out the version number to be used. You can see the details of this in the GitVersion task logs. By automating that process it takes away the manual overhead.

```
INFO [08/02/19 12:03:47:14] Begin: Calculating base versions
  INFO [08/02/19 12:03:47:16] Fallback base version: 0.1.0 with commit count source 8abb7f66bda2c03c0cd467efd3648cab1a2fc390
  INFO [08/02/19 12:03:47:19] Git tag 'v1.0.0.0': 1.0.0+0 with commit count source 049305aaae070d2ae87ee598adca80fdc4485076
  INFO [08/02/19 12:03:47:25] Found multiple base versions which will produce the same SemVer (1.0.0), taking oldest source
  INFO [08/02/19 12:03:47:25] Base version used: Git tag 'v1.0.0.0': 1.0.0+0 with commit count source 049305aaae070d2ae87ee5
INFO [08/02/19 12:03:47:25] End: Calculating base versions (Took: 104.90ms)
INFO [08/02/19 12:03:47:25] Skipping version increment
INFO [08/02/19 12:03:47:25] 0 commits found between 049305aaae070d2ae87ee598adca80fdc4485076 and 049305aaae070d2ae87ee598adc
```

There's more...

GitVersion supports a wide spectrum of configurations, you can choose which process should be used when calculating the git version. This can be done by generating a GitVersion config file and referring

that as part of your pipeline execution. You can read more about it GitVersion and how to accomplish that **here**[7]

7 https://gitversion.readthedocs.io/en/latest/

Public Projects

Public 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 Account. This however blocks a lot of interesting scenarios where you would want to share your code and artifacts publically or simply provide a wiki library or build status page for unauthenticated users….

With public projects, users will be able to mark a Azure DevOps Team Project as public. This will enable anonymous users to be able to view the contents of that project in a read-only state enabling collaboration with anonymous (un-authenticated) users that wasn't possible before. Anonymous users will largely see the same views as authenticated users, with non-public functionality such as settings, or actions (such as queue build) hidden or disabled.

Public versus private projects

Projects in Azure DevOps provide a repository for source code and a place for a group of developers and teams to plan, track progress, and collaborate on building software solutions. One or more projects can be defined within an organization in Azure DevOps.

Users that aren't signed into the service have read-only access to public projects on Azure DevOps. Private projects, on the other hand, require users to be granted access to the project and signed in to access the services.

Supported services

Non-members of a public project will have read-only access to a limited set of services, specifically:

- Browse the code base, download code, view commits, branches, and pull requests
- View and filter work items
- View a project page or dashboard
- View the project Wiki
- Perform semantic search of the code or work items

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.

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.

Files in Git

Storing Files in Git

Git is great at keeping the footprint of your source code small because the differences between versions are easily picked out and code is easily compressed. Large files that don't compress well and change entirely between versions (such as binaries) present problems when stored in your Git repos. Git's fast performance comes from its ability to address and switch to all versions of a file from its local storage.

If you have large, undiffable files in your repo such as binaries, you will keep a full copy of that file in your repo every time you commit a change to the file. If many versions of these files exist in your repo, they will dramatically increase the time to checkout, branch, fetch, and clone your code.

What kind of files should you store in Git?

Source code-not dependencies

As your team works with editors and tools to create and update files, you should put these files into Git so your team can enjoy the benefits of Git's workflow. Don't commit other types of files, such as DLLs, library files, and other dependencies that aren't created by your team but your code depends on into your repo. Deliver these files through package management to your systems.

Package management bundles your dependencies and installs the files on your system when you deploy the package. Packages are versioned to ensure that code tested in one environment runs the same in another environment as long as they have the same installed packages.

Don't commit outputs

Don't commit the binaries, logs, tracing output or diagnostic data from your builds and tests. These are outputs from your code, not the source code itself. Share logs and trace information with your team through work item tracking tools or through team file sharing.

Store small, infrequently updated binary sources in Git

Binary source files that are infrequently updated will have relatively few versions committed, and will not take up very much space provided that their file size is small. Images for the web, icons, and other smaller art assets can fall into this category. It's better to store these files in Git with the rest of your source so your team can use consistent workflow.

Important: Even small binaries can cause problems if updated often. One hundred changes to a 100KB binary file uses up as much storage as 10 changes to a 1MB binary, and due to the frequency of updates to the smaller binary will take slow down branching performance more often than the large binary.

Don't commit large, frequently updated binary assets

Git will manage one main version of a file and then store only the differences from that version in a process known as deltification. Deltification and file compression allow Git to store your entire code history in your local repo. Large binaries usually change entirely between versions and are often already compressed, making these files difficult for Git to manage since the difference between versions is very large. Git must store the entire contents of each version of the file and has difficulty saving space through deltification and compression. Storing the full file versions of these files causes repo size to increase over time, reducing branching performance, increasing the clone times, and expanding storage requirements.

Git Large File Storage

Use Git Large File Storage (LFS)

When you have source files with large differences between versions and frequent updates, you can use Git LFS to manage these file types. Git LFS is an extension to Git which commits data describing the large files in a commit to your repo, and stores the binary file contents into separate remote storage. When you clone and switch branches in your repo, Git LFS downloads the correct version from that remote storage. Your local development tools will transparently work with the files as if they were committed directly to your repo.

Benefits

The benefit of Git LFS is that your team can use the familiar end to end Git workflow no matter what files your team creates. LFS files can be as big as you need them to be. Additionally, as of version 2.0, Git LFS supports file locking to help your team work on large, undiffable assets like videos, sounds, and game maps.

Git LFS is is fully supported and free in Azure DevOps Services. To use LFS with Visual Studio, you need at least Visual Studio 2015 Update 2. Just follow the instructions to install the client, set up LFS tracking for files on your local repo, and then push your changes to Azure Repos.

Limitations

Git LFS has some drawbacks that you should consider before adopting:

- Every Git client used by your team must install the Git LFS client and understand its tracking configuration.
- If the Git LFS client is not installed and configured correctly, you will not see the binary files committed through Git LFS when you clone your repo. Git will download the data describing the large file (which is what Git LFS commits to the repo) and not the actual binary file itself. Committing large binaries without the Git LFS client installed will push the binary to your repo.
- Git cannot merge the changes from two different versions of a binary file even if both versions have a common parent. If two people are working on the same file at the same time, they must work together to reconcile their changes to avoid overwriting the other's work. Git LFS provides file locking to help. Users must still take care to always pull the latest copy of a binary asset before beginning work.
- Azure Repos currently does not support using SSH in repos with Git LFS tracked files.
- If a user drags and drops a binary file via the web interface into a repo which is configured for Git LFS, the binary will be committed to the repo and not the pointers that would be committed via the Git LFS client.

Module 2 Review Questions

Module 2 Review Questions

Repo Types

What are the two main types of repositories and what are advantages of each?

Suggested Answer

Mono-repo - source code is kept in a single repository. Mono-repo advantages - Clear ownership, better scale, and narrow clones. Multiple-repo – each project has its own repository. Multiple repo advantages - Better developer testing, reduced code complexity, effective code reviews, sharing of common components, and easy refactoring.

Branching Types

Can you name and describe the three types of branching?

Suggested Answer

Feature branching - all feature development should take place in a dedicated branch instead of the master branch. Gitflow branching - a strict branching model designed around the project release. Forking Workflow - every developer uses a server-side repository.

GitHooks

What are GitHooks?

Suggested Answer

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

Best Practices

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

Suggested Answer

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 3 Implement & Manage Build Infrastructure

The Concept of Pipelines in DevOps

The Concept of Pipelines in DevOps

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

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

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

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

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

Build automation and Continuous Integration

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

Test Automation

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

Deployment Automation

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

Your Pipeline Needs Platform Provisioning and Configuration Management

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

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

Orchestrating it all: Release and Pipeline Orchestration

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

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

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

Azure Pipelines

Azure Pipelines

Azure Pipelines is a cloud service that you can use to automatically build and test your code project and make it available to other users. It works with just about any language or project type. Azure Pipelines combines continuous integration (CI) and continuous delivery (CD) to constantly and consistently test and build your code and ship it to any target.

Does Azure Pipelines work with my language and tools?

Azure Pipelines is a fully featured cross platform CI and CD service. It works with your preferred Git provider and can deploy to most major cloud services, which include Azure services.

Languages

You can use many languages with Azure Pipelines, such as Python, Java, PHP, Ruby, C#, and Go.

Version control systems

Before you use continuous integration and continuous delivery practices for your applications, you must have your source code in a version control system. Azure Pipelines integrates with GitHub, GitLab, Azure Repos, Bitbucket, and Subversion.

Application types

You can use Azure Pipelines with most application types, such as Java, JavaScript, Python, .NET, PHP, Go, XCode, and C++.

Deployment targets

Use Azure Pipelines to deploy your code to multiple targets. Targets include container registries, virtual machines, Azure services, or any on-premises or cloud target such as Microsoft Azure, Google Cloud, or Amazon cloud services.

Package formats

To produce packages that can be consumed by others, you can publish NuGet, npm, or Maven packages to the built-in package management repository in Azure Pipelines. You also can use any other package management repository of your choice.

Why should I use CI and CD and Azure Pipelines?

Implementing CI and CD pipelines helps to ensure consistent and quality code that's readily available to users.

Azure Pipelines is a quick, easy, and safe way to automate building your projects and making them available to users.

Use CI and CD for your project

Continuous integration is used to automate tests and builds for your project. CI helps to catch bugs or issues early in the development cycle, when they're easier and faster to fix. Items known as artifacts are produced from CI systems. They're used by the continuous delivery release pipelines to drive automatic deployments.

Continuous delivery is used to automatically deploy and test code in multiple stages to help drive quality. Continuous integration systems produce deployable artifacts, which includes infrastructure and apps. Automated release pipelines consume these artifacts to release new versions and fixes to the target of your choice.

Continuous integration (CI)	Continuous delivery (CD)
Increase code coverage.	Automatically deploy code to production.
Build faster by splitting test and build runs.	Ensure deployment targets have latest code.
Automatically ensure you don't ship broken code.	Use tested code from CI process.
Run tests continually.	

Use Azure Pipelines for CI and CD

There are several reasons to use Azure Pipelines for your CI and CD solution. You can use it to:

- Work with any language or platform.
- Deploy to different types of targets at the same time.
- Integrate with Azure deployments.
- Build on Windows, Linux, or Mac machines.
- Integrate with GitHub.
- Work with open-source projects.

Azure Key Terms

Understanding the basic terms and parts of Azure Pipelines helps you further explore how it can help you deliver better code more efficiently and reliably.

Agent

When your build or deployment runs, the system begins one or more jobs. An agent is installable software that runs one build or deployment job at a time.

Artifact

An artifact is a collection of files or packages published by a build. Artifacts are made available to subsequent tasks, such as distribution or deployment.

Build

A build represents one execution of a pipeline. It collects the logs associated with running the steps and the results of running tests.

Continuous delivery

Continuous delivery (CD) is a process by which code is built, tested, and deployed to one or more test and production stages. Deploying and testing in multiple stages helps drive quality. Continuous integration systems produce deployable artifacts, which includes infrastructure and apps. Automated release pipelines consume these artifacts to release new versions and fixes to existing systems. Monitoring and alerting systems run constantly to drive visibility into the entire CD process. This process ensures that errors are caught often and early.

Continuous integration

Continuous integration (CI) is the practice used by development teams to simplify the testing and building of code. CI helps to catch bugs or problems early in the development cycle, which makes them easier and faster to fix. Automated tests and builds are run as part of the CI process. The process can run on a set schedule, whenever code is pushed, or both. Items known as artifacts are produced from CI systems. They're used by the continuous delivery release pipelines to drive automatic deployments.

Deployment target

A deployment target is a virtual machine, container, web app, or any service that's used to host the application being developed. A pipeline might deploy the app to one or more deployment targets after build is completed and tests are run.

Job

A build contains one or more jobs. Each job runs on an agent. A job represents an execution boundary of a set of steps. All of the steps run together on the same agent. For example, you might build two configurations - x86 and x64. In this case, you have one build and two jobs.

Pipeline

A pipeline defines the continuous integration and deployment process for your app. It's made up of steps called tasks. It can be thought of as a script that defines how your test, build, and deployment steps are run.

Release

When you use the visual designer, you create a release pipeline in addition to a build pipeline. A release is the term used to describe one execution of a release pipeline. It's made up of deployments to multiple stages.

Task

A task is the building block of a pipeline. For example, a build pipeline might consist of build tasks and test tasks. A release pipeline consists of deployment tasks. Each task runs a specific job in the pipeline.

Trigger

A trigger is something that's set up to tell the pipeline when to run. You can configure a pipeline to run upon a push to a repository, at scheduled times, or upon the completion of another build. All of these actions are known as triggers.

Evaluate Use of Hosted vs Private Agents

Hosted vs Private Agents

To build your code or deploy your software you need at least one agent. As you add more code and people, you'll eventually need more. When your build or deployment runs, the system begins one or more jobs. An agent is installable software that runs one build or deployment job at a time.

Microsoft-hosted agents

If your pipelines are in Azure Pipelines, then you've got a convenient option to build and deploy using a Microsoft-hosted agent. With Microsoft-hosted agents, maintenance and upgrades are taken care of for you. Each time you run a pipeline, you get a fresh virtual machine. The virtual machine is discarded after one use.

For many teams this is the simplest way to build and deploy. You can try it first and see if it works for your build or deployment. If not, you can use a self-hosted agent.

Self-hosted agents

An agent that you set up and manage on your own to run build and deployment jobs is a self-hosted agent. You can use self-hosted agents in Azure Pipelines. Self-hosted agents give you more control to install dependent software needed for your builds and deployments.

You can install the agent on Linux, macOS, or Windows machines. You can also install an agent on a Linux Docker container. After you've installed the agent on a machine, you can install any other software on that machine as required by your build or deployment jobs.

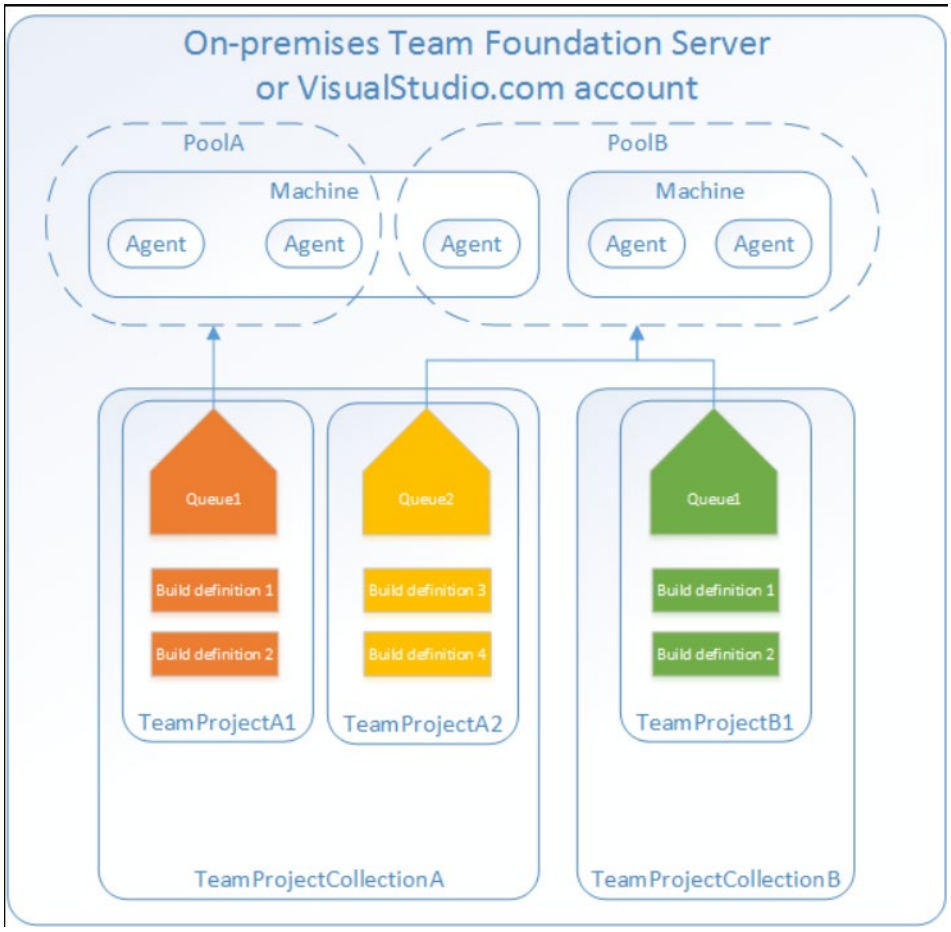
Agent Pools

Agent Pools

Instead of managing each agent individually, you organize agents into agent pools. An agent pool defines the sharing boundary for all agents in that pool. In Azure Pipelines, agent pools are scoped to the Azure DevOps organization; so you can share an agent pool across projects.

A project agent pool provides access to an organization agent pool. When you create a build or release pipeline, you specify which pool it uses. Pools are scoped to your project so you can only use them across build and release pipelines within a project.

To share an agent pool with multiple projects, in each of those projects, you create a project agent pool pointing to an organization agent pool. While multiple pools across projects can use the same organization agent pool, multiple pools within a project cannot use the same organization agent pool. Also, each project agent pool can use only one organization agent pool.



Default Agent Pools

The following organization agent pools are provided by default:

- Default pool: Use it to register **self-hosted agents**[1] that you've set up.

1 https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/agents?view=vsts

- Hosted Ubuntu 1604 pool (Azure Pipelines only): Enables you to build and release on Linux machines without having to configure a self-hosted Linux agent. Agents in this pool do not run in a container, but the Docker tools are available for you to use if you want to run **container jobs**[2].
- Hosted Linux pool (Azure Pipelines only): Enables you to build and release on Linux machines without having to configure a self-hosted Linux agent. The agents in this pool run on an Ubuntu Linux host inside the **vsts-agent-docker container**[3]. Note: this pool has been superceded by the Hosted Ubuntu 1604 pool. It was removed from the service on December 1, 2018.
- Hosted macOS pool (Azure Pipelines only): Enables you to build and release on macOS without having to configure a self-hosted macOS agent. This option affects where your data is stored. **Learn more**[4]
- Hosted VS2017 pool (Azure Pipelines only): The Hosted VS2017 pool is another built-in pool in Azure Pipelines. Machines in this pool have Visual Studio 2017 installed on Windows Server 2016 operating system. For a complete list of software installed on these machines, see **Microsoft-hosted agents**[5].
- Hosted pool (Azure Pipelines only): The Hosted pool is the built-in pool that is a collection of Microsoft-hosted agents. For a complete list of software installed on Microsoft-hosted agents, see **Microsoft-hosted agents**[6].
- Hosted Windows Container pool (Azure Pipelines only): Enabled you to build and release inside **Windows containers**[7]. Unless you're building using containers, Windows builds should run in the Hosted VS2017 or Hosted pools.

Each of these Microsoft-hosted organization agent pools is exposed to each project through a corresponding project agent pool. By default, all contributors in a project are members of the User role on each hosted pool. This allows every contributor in a project to author and run build and release pipelines using Microsoft-hosted pools.

Pools are used to run jobs. Learn about **specifying pools for jobs**[8].

Typical Situations for Agent Pools

If you've got a lot of agents intended for different teams or purposes, you might want to create additional pools as explained below.

Creating agent pools

Here are some typical situations when you might want to create agent pools:

- You're a member of a project and you want to use a set of machines owned by your team for running build and deployment jobs. First, make sure you're a member of a group in All Pools with the Administrator role. Next create a New project agent pool in your project settings and select the option to Create a new organization agent pool. As a result, both an organization and project-level agent pool will be created. Finally install and configure agents to be part of that agent pool.
- You're a member of the infrastructure team and would like to set up a pool of agents for use in all projects. First make sure you're a member of a group in All Pools with the Administrator role. Next create a New organization agent pool in your admin settings and select the option to Auto-provision corresponding project agent pools in all projects while creating the pool. This setting ensures all projects have a pool pointing to the organization agent pool. The system creates a pool for existing

2 https://docs.microsoft.com/en-us/azure/devops/pipelines/process/container-phases?view=vsts&tabs=yaml
3 https://github.com/Microsoft/vsts-agent-docker
4 https://www.microsoft.com/en-us/trustcenter/cloudservices/azure-devops-services-data-location
5 https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/hosted?view=vsts&tabs=yaml
6 https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/hosted?view=vsts&tabs=yaml
7 https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/
8 https://docs.microsoft.com/en-us/azure/devops/pipelines/process/phases?view=vsts&tabs=yaml

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

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.

Role on a project agent pool	Purpose
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 & Concurrency

Microsoft-Hosted vs Self-Hosted

One Parallel job in Azure Pipeline will let you run a single Build or Release job at any given time. This rule is true whether you run the job on Microsoft hosted or self hosted agents.

Microsoft-hosted CI/CD

If you want to run your builds and releases on machines that Microsoft manages, use Microsoft-hosted parallel jobs. Your jobs run on the pool of hosted agents.

Microsoft provides a free tier of service by default for every organization:

- Public project: 10 free Microsoft-hosted parallel jobs that can run for up to 360 minutes (6 hours) each time, with no overall time limit per month.
- Private project: One free parallel job that can run for up to 30 minutes each time, until you've used 1,800 minutes (30 hours) per month.

When the free tier is no longer sufficient:

- Public project: Contact Microsoft to get your free tier limits increased.
- Private project: You can pay for additional capacity per parallel job. Paid parallel jobs remove the monthly time limit and allow you to run each job for up to 360 minutes (6 hours). Buy Microsoft-hosted parallel jobs.

Self-hosted CI/CD

If you want Azure Pipelines to orchestrate your builds and releases, but use your own machines to run them, use self-hosted parallel jobs. You start by deploying agents on your machines. You can register any number of these self-hosted agents in your organization. Microsoft charges based on the number of jobs you want to run at a time, not the number of agents registered.

Microsoft provide a free tier of service by default in every organization:

- Public project: 10 free self-hosted parallel jobs.
- Private project: One self-hosted parallel job. Additionally, for each active Visual Studio + Enterprise subscriber who is a member of your organization, you get one additional self-hosted parallel job.

When the free tier is no longer sufficient:

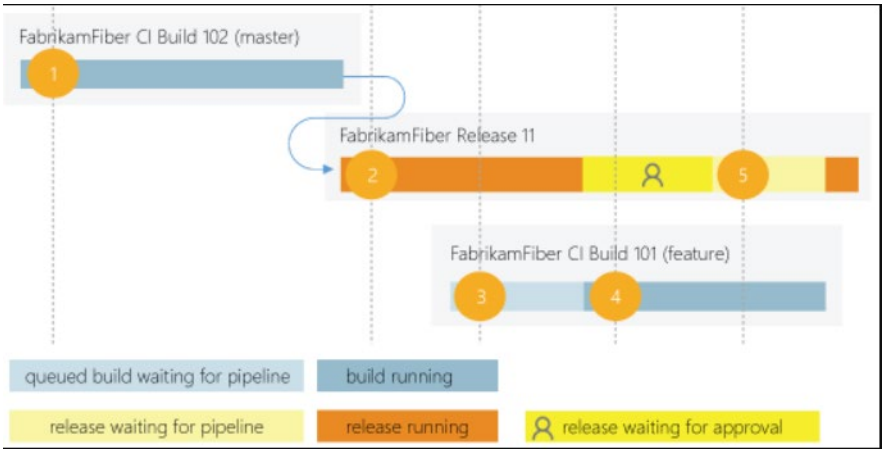
- Public project: Contact us to get your free tier limits increased.
- Private project: You can pay for additional capacity per parallel job. Buy self-hosted parallel jobs.

There are no time limits on self-hosted jobs.

Parallel Jobs

How a parallel job is consumed by a build or release

Consider an organization that has only one Microsoft-hosted parallel job. This job allows users in that organization to collectively run only one build or release job at a time. When additional jobs are triggered, they are queued and will wait for the previous job to finish.



A release consumes a parallel job only when it's being actively deployed to a stage. While the release is waiting for an approval or a manual intervention, it does not consume a parallel job.

Simple example of parallel jobs

- FabrikamFiber CI Build 102 (master branch) starts first.
- Deployment of FabrikamFiber Release 11 is triggered by completion of FabrikamFiber CI Build 102.
- FabrikamFiber CI Build 101 (feature branch) is triggered. The build can't start yet because Release 11's deployment is active. So the build stays queued.
- Release 11 waits for approvals. Fabrikam CI Build 101 starts because a release that's waiting for approvals does not consume a parallel job.
- Release 11 is approved. It resumes only after Fabrikam CI Build 101 is completed.

Relationship between jobs and parallel jobs

The term job can refer to multiple concepts, and its meaning depends on the context:

- When you define a build or release, you can define it as a collection of jobs. When a build or release runs, you can run multiple jobs as part of that build or release.
- Each job consumes a parallel job that runs on an agent. When there aren't enough parallel jobs available for your organization, then the jobs are queued up and run one after the other.

When you run a server job or deploy to a deployment group, you don't consume any parallel jobs.

Estimating Parallel Jobs

Determine how many parallel jobs you need

You can begin by seeing if the free tier offered in your organization is enough for your teams. When you've reached the 1,800-minute per month limit for the free tier of Microsoft-hosted parallel jobs, you can start by buying one parallel job to remove this monthly time limit before deciding to purchase more.

As the number of queued builds and releases exceeds the number of parallel jobs you have, your build and release queues will grow longer. When you find the queue delays are too long, you can purchase additional parallel jobs as needed.

Simple estimate

A simple rule of thumb: Estimate that you'll need one parallel job for every four to five users in your organization.

Detailed estimate

In the following scenarios, you might need multiple parallel jobs:

- If you have multiple teams, and if each of them require a CI build, you'll likely need a parallel job for each team.
- If your CI build trigger applies to multiple branches, you'll likely need a parallel job for each active branch.
- If you develop multiple applications by using one organization or server, you'll likely need additional parallel jobs: one to deploy each application at the same time.

View available parallel jobs

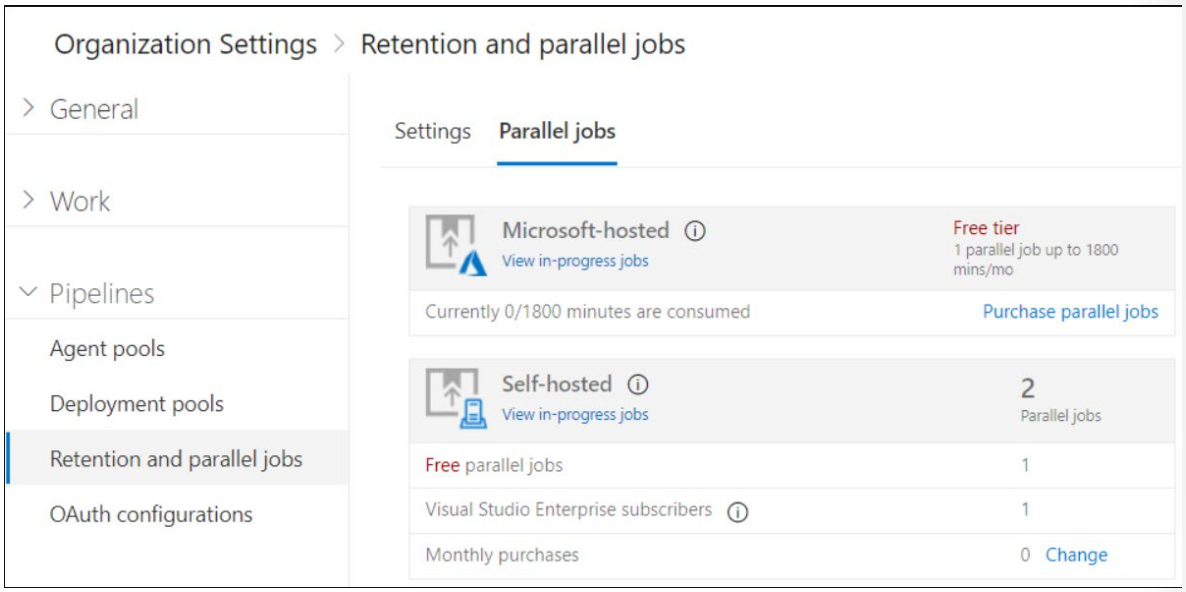
Browse to `Organization settings > Pipelines > Retention and parallel jobs > Parallel jobs`

Location of parallel jobs in organization settings

URL example: `https://{your_organization}/_admin/_buildQueue?_a=resourceLimits`

View the maximum number of parallel jobs that are available in your organization.

Select View in-progress jobs to display all the builds and releases that are actively consuming an available parallel job or that are queued waiting for a parallel job to be available.



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

How do I qualify for the free tier of public projects?

Microsoft will automatically apply the free tier limits for public projects if you meet both of these conditions:

- Your pipeline is part of an Azure Pipelines public project.
- Your pipeline builds a public repository from GitHub or from the same public project in your Azure DevOps organization.

Are there limits on who can use Azure Pipelines?

You can have as many users as you want when you're using Azure Pipelines. There is no per-user charge for using Azure Pipelines. Users with both basic and stakeholder access can author as many builds and releases as they want.

Are there any limits on the number of builds and release pipelines that I can create?

No. You can create hundreds or even thousands of definitions for no charge. You can register any number of self-hosted agents for no charge.

As a Visual Studio Enterprise subscriber, do I get additional parallel jobs for TFS and Azure Pipelines?

Yes. Visual Studio Enterprise subscribers get one parallel job in Team Foundation Server 2017 or later and one self-hosted parallel job in each Azure DevOps Services organization where they are a member.

What about the option to pay for hosted agents by the minute?

Some of our earlier customers are still on a per-minute plan for the hosted agents. In this plan, you pay $0.05/minute for the first 20 hours after the free tier, and $0.01/minute after 20 hours. Because of the following limitations in this plan, you might want to consider moving to the parallel jobs model:

When you're using the per-minute plan, you can run only one job at a time.

If you run builds for more than 14 paid hours in a month, the per-minute plan might be less cost-effective than the parallel jobs model.

Azure Pipelines YAML vs Visual Designer

Azure Pipelines and YAML

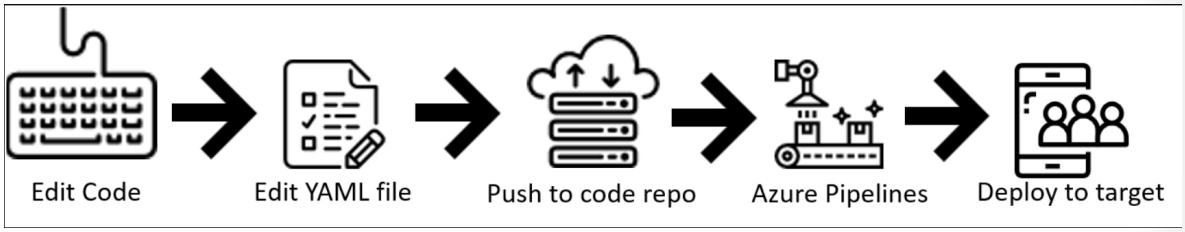
You can either use YAML to define your pipelines or use the visual designer to do the same.

When you use YAML, you define your pipeline mostly in code (a YAML file) alongside the rest of the code for your app. When you use the visual designer, you define a build pipeline to build and test your code, and then to publish artifacts. You also define a release pipeline to consume and deploy those artifacts to deployment targets.

Use Azure Pipelines with YAML

You can configure your pipelines in a YAML file that exists alongside your code.

- Configure Azure Pipelines to use your Git repo.
- Edit your azure-pipelines.yml file to define your build.
- Push your code to your version control repository. This action kicks off the default trigger to build and deploy and then monitor the results.
- Your code is now updated, built, tested, and packaged. It can be deployed to any target.



Benefits of using YAML

- The pipeline is versioned with your code and follows the same branching structure. You get validation of your changes through code reviews in pull requests and branch build policies.
- Every branch you use can modify the build policy by modifying the azure-pipelines.yml file.
- A change to the build process might cause a break or result in an unexpected outcome. Because the change is in version control with the rest of your codebase, you can more easily identify the issue.

If you think the YAML workflow is best for you, create your first pipeline by using **YAML**[9].

Azure Pipelines and Visual Designer

Use Azure Pipelines in the visual designer

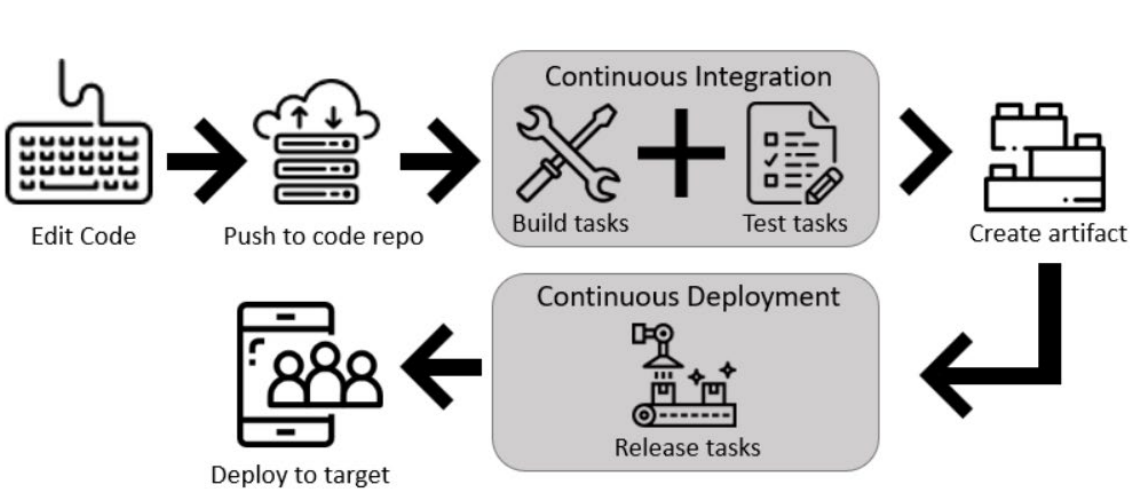
You can create and configure your build and release pipelines in the Azure DevOps Services web portal with the visual designer.

Configure Azure Pipelines to use your Git repo.

1. Use the Azure Pipelines visual designer to create and configure your build and release pipelines.

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

2. Push your code to your version control repository. This action triggers your pipeline and runs tasks such as building or testing code.
3. The build creates an artifact that's used by the rest of your pipeline to run tasks such as deploying to staging or production.
4. Your code is now updated, built, tested, and packaged. It can be deployed to any target.



Benefits of using the visual designer

The visual designer is great for users who are new to the world of continuous integration (CI) and continuous delivery (CD).

- The visual representation of the pipelines makes it easier to get started.
- The visual designer is located in the same hub as the build results. This location makes it easier to switch back and forth and make changes.

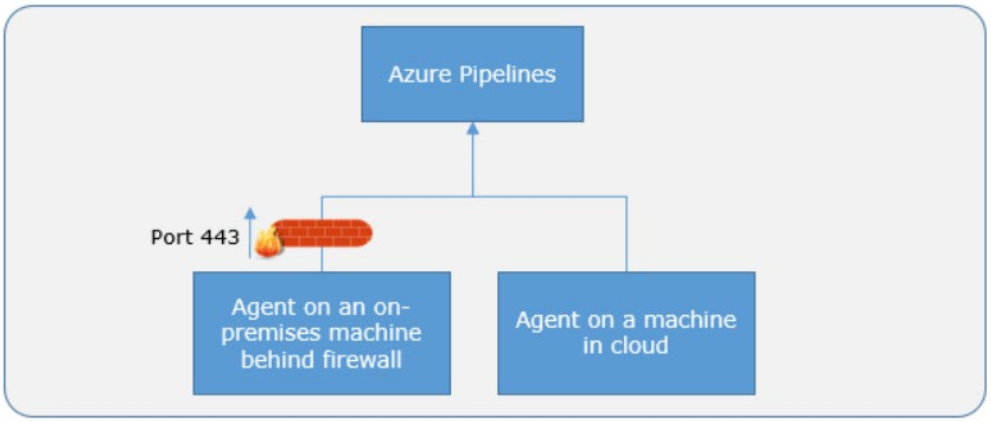
If you think the designer workflow is best for you, create your first pipeline by using the **visual designer**[10].

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

Setup Private Agents

Communication with Azure Pipelines

The agent communicates with Azure Pipelines to determine which job it needs to run, and to report the logs and job status. This communication is always initiated by the agent. All the messages from the agent to Azure Pipelines over HTTPS, depending on how you configure the agent. This pull model allows the agent to be configured in different topologies as shown below.



Here is a common communication pattern between the agent and Azure Pipelines.

The user registers an agent with Azure Pipelines or TFS 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.

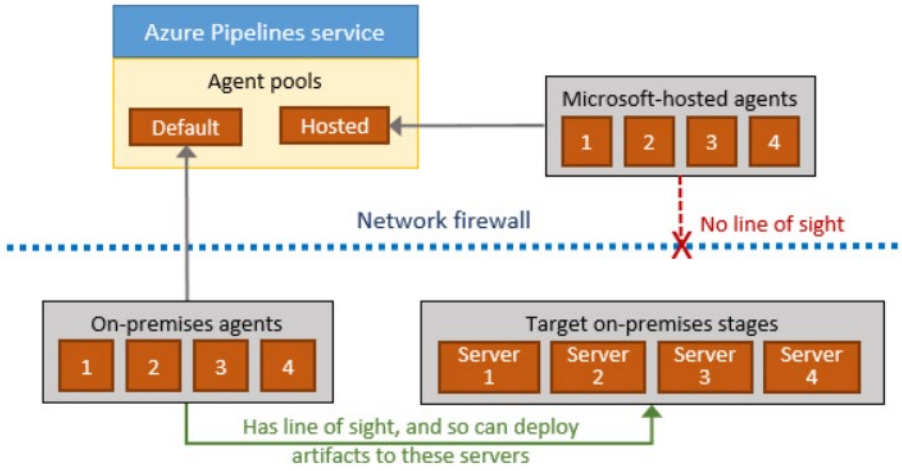
The payload of the messages exchanged between the agent and Azure Pipelines are secured using asymmetric encryption. Each agent has a public-private key pair, and the public key is exchanged with the server during registration. The server uses the public key to encrypt the payload of the job before sending 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

Communication to deploy to target servers

When you use the agent to deploy artifacts to a set of servers, it must have "line of sight" connectivity to those servers. The Microsoft-hosted agent pools, by default, have connectivity to Azure websites and servers running in Azure.

If your on-premises environments do not have connectivity to a Microsoft-hosted agent pool (which is typically the case due to intermediate firewalls), you'll need to manually configure a self-hosted agent on on-premises computer(s). The agents must have connectivity to the target on-premises environments, and access to the Internet to connect to Azure Pipelines or Team Foundation Server, as shown in the following schematic.



Other Considerations

Authentication

To register an agent, you need to be a member of the administrator role in the agent pool. The identity of agent pool administrator is needed only at the time of registration and is not persisted on the agent, and is not used in any subsequent communication between the agent and Azure Pipelines. In addition, you must be a local administrator on the server in order to configure the agent. Your agent can authenticate to Azure Pipelines or TFS using one of the following methods:

Personal Access Token (PAT)

Generate and use a PAT to connect an agent with Azure Pipelines. PAT is the only scheme that works with Azure Pipelines. Also, as explained above, this PAT is used only at the time of registering the agent, and not for subsequent communication.

Interactive vs. service

You can run your agent as either a service or an interactive process. Whether you run an agent as a service or interactively, you can choose which account you use to run the agent. Note that this is different from the credentials that you use when you register the agent with Azure Pipelines. The choice of agent account depends solely on the needs of the tasks running in your build and deployment jobs.

For example, to run tasks that use Windows authentication to access an external service, you must run the agent using an account that has access to that service. However, if you are running UI tests such as Selenium or Coded UI tests that require a browser, the browser is launched in the context of the agent account.

After you've configured the agent, we recommend you first try it in interactive mode to make sure it works. Then, for production use, we recommend you run the agent in one of the following modes so that it reliably remains in a running state. These modes also ensure that the agent starts automatically if the machine is restarted.

As a service. You can leverage the service manager of the operating system to manage the lifecycle of the agent. In addition, the experience for auto-upgrading the agent is better when it is run as a service.

As an interactive process with auto-logon enabled. In some cases, you might need to run the agent interactively for production use - such as to run UI tests. When the agent is configured to run in this mode, the screen saver is also disabled. Some domain policies may prevent you from enabling auto-log-on or disabling the screen saver. In such cases, you may need to seek an exemption from the domain policy, or run the agent on a workgroup computer where the domain policies do not apply.

Note: There are security risks when you enable automatic logon or disable the screen saver because you enable other users to walk up to the computer and use the account that automatically logs on. If you configure the agent to run in this way, you must ensure the computer is physically protected; for example, located in a secure facility. If you use Remote Desktop to access the computer on which an agent is running with auto-logon, simply closing the Remote Desktop causes the computer to be locked and any UI tests that run on this agent may fail. To avoid this, use the tscon command to disconnect from Remote Desktop. For example:

```
%windir%\System32\tscon.exe 1 /dest:console
```

Agent version and upgrades

Microsoft updates the agent software every few weeks in Azure Pipelines. The agent version is indicated in the format {major}.{minor}. For instance, if the agent version is 2.1, then the major version is 2 and the minor version is 1. When a newer version of the agent is only different in minor version, it is automatically upgraded by Azure Pipelines. This upgrade happens when one of the tasks requires a newer version of the agent.

If you run the agent interactively, or if there is a newer major version of the agent available, then you have to manually upgrade the agents. You can do this easily from the agent pools tab under your project collection or organization.

You can view the version of an agent by navigating to Agent pools and selecting the Capabilities tab for the desired agent.

```
Azure Pipelines: <a href="https://dev.azure.com/{your_organization}/_ad-
min/_AgentPool" title="" target="_blank" data-generated=''>https://dev.
azure.com/{your_organization}/_admin/_AgentPool</a>
```

Question and Answer

Do self-hosted agents have any performance advantages over Microsoft-hosted agents?

In many cases, yes. Specifically:

If you use a self-hosted agent you can run incremental builds. For example, you define a CI build pipeline that does not clean the repo and does not perform a clean build, your builds will typically run faster. When you use a Microsoft-hosted agent, you don't get these benefits because the agent is destroyed after the build or release pipeline is completed.

A Microsoft-hosted agent can take longer to start your build. While it often takes just a few seconds for your job to be assigned to a Microsoft-hosted agent, it can sometimes take several minutes for an agent to be allocated depending on the load on our system.

Can I install multiple self-hosted agents on the same machine?

Yes. This approach can work well for agents that run jobs that don't consume a lot of shared resources. For example, you could try it for agents that run releases that mostly orchestrate deployments and don't do a lot of work on the agent itself.

You might find that in other cases you don't gain much efficiency by running multiple agents on the same machine. For example, it might not be worthwhile for agents that run builds that consume a lot of disk and I/O resources.

You might also run into problems if parallel build jobs are using the same singleton tool deployment, such as npm packages. For example, one build might update a dependency while another build is in the middle of using it, which could cause unreliable results and errors.

Further instructions on how to set up private agents can be found here,

- **https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/v2-windows?view=vsts**
- **https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/proxy?view=vsts&tabs=windows**

Integrate Jenkins with Azure Pipelines

Lab - Integrate Jenkins with Azure Pipelines

In this lab, you will examine two methods for integrating Jenkins: run CI jobs in Jenkins separately, and wrap a Jenkins CI job inside an Azure pipeline. You will learn how to:

- Provision Jenkins on Azure VM using the Jenkins template available on the Azure Marketplace
- Configure Jenkins to work with Maven and Azure DevOps
- Create a build job in Jenkins
- Configure Azure Pipeline to integrate with Jenkins
- Configure a CD pipeline in Azure Pipelines to deploy the build artifacts

Configuring a CD pipeline for your Jenkins CI - **https://www.azuredevopslabs.com/labs/vstsextend/jenkins/**

Integration External Source Control with Azure Pipelines

Lab - Integration External Source Control with Azure Pipelines

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.

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

Integrate Your GitHub Projects With Azure Pipelines - **https://www.azuredevopslabs.com/labs/azuredevops/github-integration/**

Analyze & Integrate Docker Multi-stage Builds

Builder Patterns

With compiled runtimes like Go, Java and .NET, you'll want to first compile your code before having a binary that can be run. The components required to compile your code are not required to run your code. And the SDKs can be quite big, not to mention any potential attack surface area. A workaround which is informally called the builder pattern involves using two Docker images - one to perform a build and another to ship the results of the first build without the penalty of the build-chain and tooling in the first image.

An example of the builder pattern:

- Derive from a dotnet base image with the whole runtime/SDK (Dockerfile.build)
- Add source code
- Produce a statically-linked binary
- Copy the static binary from the image to the host (docker create, docker cp)
- Derive from SCRATCH or some other light-weight image (Dockerfile)
- Add the binary back in
- Push a tiny image to the Docker Hub

This normally meant having two separate Dockerfiles and a shell script to orchestrate all of the 7 steps above. Additionally the challenge with building on the host, including hosted build agents is we must first have a build agent with everything we need, including the specific versions. If your dev shop has any history of .NET Apps, you'll likely have multiple versions to maintain. Which means you have complex agents to deal with the complexities.

Multi-stage Builds

What are multi-stage builds?
Multi-stage builds give the benefits of the builder pattern without the hassle of maintaining three separate files... Lets take a look at a multi-stage dockerfile...

```
FROM microsoft/aspnetcore:2.0 AS base
WORKDIR /app
EXPOSE 80

FROM microsoft/aspnetcore-build:2.0 AS builder
WORKDIR /src
COPY *.sln ./
COPY Web/Web.csproj Web/
RUN dotnet restore
COPY . .
WORKDIR /src/Web
RUN dotnet build -c Release -o /app

FROM builder AS publish
RUN dotnet publish -c Release -o /app

FROM base AS production
```

```
WORKDIR /app
COPY --from=publish /app .
ENTRYPOINT ["dotnet", "Web.dll"]
```

At first, it simply looks like several dockerfiles stitched together. Multi-stage Dockerfiles can be layered or inherited. When you look closer, there are a couple of key things to realize.
Notice the 3rd stage

```
FROM builder AS publish
```

builder isn't an image pulled from a registry. It's the image we defined in stage 2, where we named the result of our our -build (sdk) image "builder". `Docker build` will create a named image we can later reference.

We can also copy the output from one image to another. This is the real power to compile our code with one base sdk image (`microsoft/aspnetcore-build`) , while creating a production image, based on an optimized runtime image. (`microsoft/aspnetcore`). Notice the line

```
COPY --from=publish /app .
```

This takes the /app directory from the publish image, and copies it to the working directory of the production image.

Breakdown Of Stages

The first stage provides the base of our optimized runtime image. Notice it derives from `microsoft/aspnetcore`. This is where we'd specify additional production configurations, such as registry configurations, MSIexec of additional components,... Any of those environment configurations you would hand off to your ops folks to prepare the VM.
The second stage is our build environment. `microsoft/aspnetcore-build` This includes everything we need to compile our code. From here, we have compiled binaries we can publish, or test. More on testing in a moment.

The 3rd stage derives from our builder. It takes the compiled output and "publishes" them, in .NET terms. Publishing simply means take all the output required to deploy your "app/service/component" and place it in a single directory. This would include your compiled binaries, graphics (images), javascript, etc.

The 4th stage is taking the published output, and placing it in the optimized image we defined in the first stage.

Why Is Publish Separate From Build?

You'll likely want to run unit tests to verify your compiled code, or the aggregate of the compiled code from multiple developers being merged together, continues to function as expected. To run unit tests, you could place the following stage between builder and publish.

```
FROM builder AS test
WORKDIR /src/Web.test
RUN dotnet test
```

If your tests fail, the build will cease to continue.

Why Is Base First?

You could argue this is simply the logical flow. We first define the base runtime image. Get the compiled output ready, and place it in the base image. However, it's more practical. While debugging your applications under Visual Studio Container Tools, VS will debug your code directly in the base image. When you hit F5, Visual Studio will compile the code on your dev machine. It will then volume mount the output to

the built runtime image; the first stage. This way you can test any configurations you've made to your production image, such as registry configurations or otherwise.

When `docker build --target base` is executed, docker starts processing the dockerfile from the beginning, through the stage (target) defined. Since base is the first stage, we take the shortest path, making the F5 experience as fast as possible. If base was after compilation (builder), you'd have to wait for all the subsequent steps to complete. One of the perf optimizations we make with VS Container Tools is to take advantage of the Visual Studio compilations on your dev machine.

Multiple Projects and Solutions

The multi-stage dockerfile on the previous page is based on a Visual Studio solution. The full example can be found in this **github repo**[11] representing a Visual Studio solution with a Web and API project. The additional unit tests are under the AddingUnitTests branch.

The challenge with solutions is they represent a collection of projects. We often think of dockerfiles specific to a single image. While true, that single image may be the result of multiple "projects".

Consider the common pattern to develop shared dlls that may represent your data access layer, your logging component, your business logic, an authentication library, or a shipping calculation. The Web or API project may each reference these project(s). They each need to take the compiled output from those project and place them in the optimized image. This isn't to say we're building yet another monolithic application. There will certainly be additional services, such as checkout, authentication, profile management, communicating with the telco switch. But there's a balance. Microservices doesn't mean every shared piece of code is it's own service.

If we look at the solution, we'll notice a few key aspects:

- Each project, which will represent a final docker image, has its own multi-stage dockerfile
- Shared component projects that are referenced by other resulting docker images do not have dockerfiles
- Each dockerfile assumes its context is the solution directory. This gives us the ability to copy in other projects
- There's a docker-compose.yml in the root of the solution. This gives us a single file to build multiple images, as well as specify build parameters, such as the image name:tag

```
Multi.sln
docker-compose.yml
[Api]
  Dockerfile
[Web]
  Dockerfile
```

We can now build the solution with a single docker command. We'll use docker-compose as our compose file has our image names as well as the individual build defintions

```
version: '3'

services:
  web:
    image: stevelas.azurecr.io/samples/multiproject/web
```

11 https://github.com/SteveLasker/AspNetCoreMultiProject

```
    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\AspNetCoreMultiPro-
ject
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.

Lab - Deploying a Multi-Container Application

This lab uses a Docker-based ASP.NET Core web application - MyHealthClinic and is deployed to a Kubernetes cluster running on Azure Container Service (AKS) using Azure DevOps.There is a mhc-aks.yaml manifest file which consists of definitions to spin up Deployments and Services such as Load Balancer in

the front and Redis Cache in the backend. The MHC application will be running in the mhc-front pod along with the Load Balancer.

The following tasks will be performed:

- Create an Azure Container Registry (ACR), AKS and Azure SQL server
- Provision the Azure DevOps Team Project with a .NET Core application using the Azure DevOps Demo Generator tool.
- Configure application and database deployment, using Continuous Deployment (CD) in the Azure DevOps
- Initiate the build to automatically deploy the application

Deploying a multi-container application to Azure Kubernetes Services - **https://www.azuredevopslabs.com/labs/vstsextend/kubernetes/**

Module 3 Review Questions

Module 3 Review Questions

Pipelines

What is a pipeline and why is it used?

Suggested Answer
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.

Pipeline Advantages

What are some advantages of Azure pipelines?

Suggested Answer
Work with any language or platform - Python, Java, PHP, Ruby, C#, and Go; 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; work with open-source projects.

Agents

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

Suggested Answer
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.

Agent Pools

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

Suggested Answer
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.

Creating Pipelines

What are two ways to configure your Azure pipelines?

Suggested Answer
YAML file and Visual Designer

Module 4 Managing Application Config and Secrets

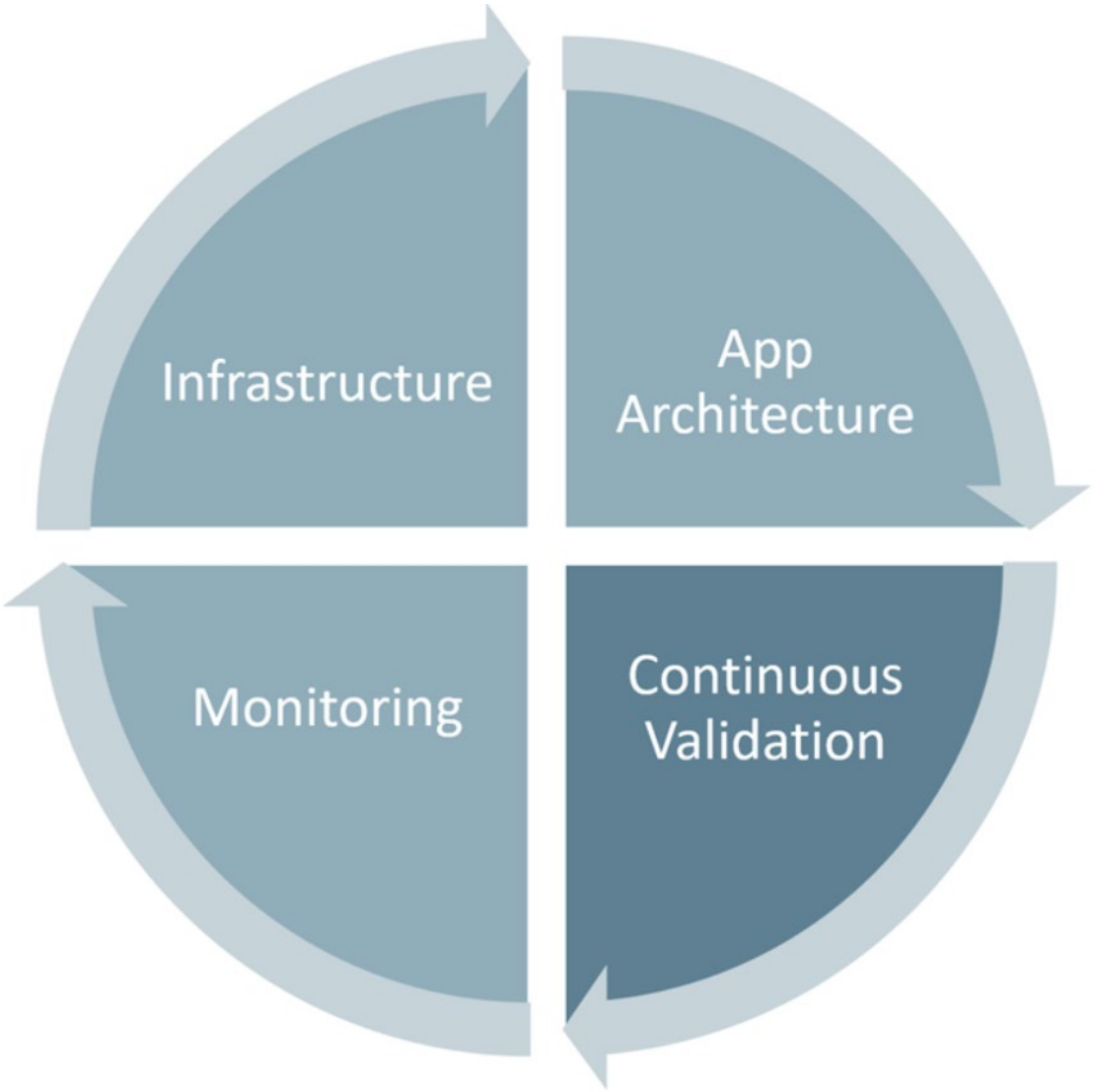
Introduction to Security

Introduction to Security

While DevOps way of working enables development teams to deploy applications faster, going faster over a cliff doesn't really help! Thanks to the cloud, DevOps teams have access to unprecedented infrastructure and scale. But that also means they can be approached by some of the most nefarious actors on the internet, as they risk the security of their business with every application deployment. Perimeter-class security is no longer viable in such a distributed environment, so now companies need to adapt a more micro-level security across application and infrastructure and have multiple lines of defence.

With continuous delivery, how do you ensure your applications are secure and stay secure? How can you find and fix security issues early in the process? This begins with practices commonly referred to as DevSecOps. DevSecOps incorporates the security team and their capabilities into your DevOps practices making security a responsibility of everyone on the team.

Security needs to shift from an afterthought to being evaluated at every step of the process. Securing applications is a continuous process that encompasses secure infrastructure, designing an architecture with layered security, continuous security validation, and monitoring for attacks.



Security is everyone's responsibility and needs to be looked at holistically across the application life cycle. In this module we'll discuss practical examples using real code for automating security tasks. We'll also see how continuous integration and deployment pipelines can accelerate the speed of security teams and improve collaboration with software development teams.

SQL Injection Attack

SQL Injection (SQLi) is a type of an injection attack that makes it possible to execute malicious SQL statements. These statements control a database server behind a web application. Attackers can use SQL Injection vulnerabilities to bypass application security measures. They can go around authentication and authorization of a web page or web application and retrieve the content of the entire SQL database. They can also use SQL Injection to add, modify, and delete records in the database.

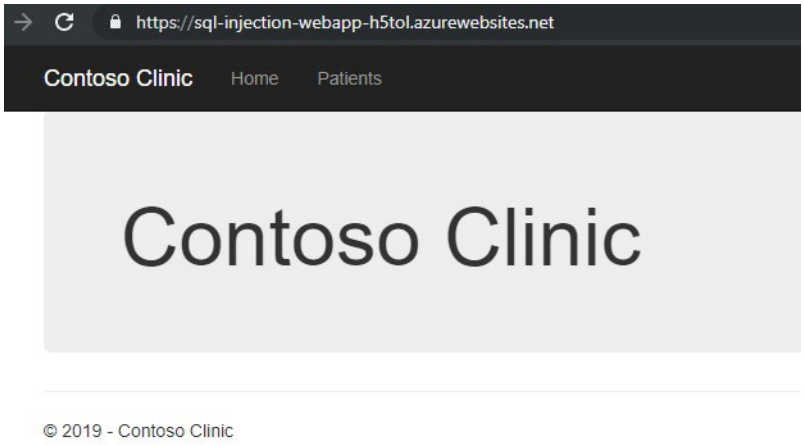
An SQL Injection vulnerability may affect any website or web application that uses an SQL database such as MySQL, Oracle, SQL Server, or others. Criminals may use it to gain unauthorized access to your sensitive data: customer information, personal data, trade secrets, intellectual property, and more.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

ties. The OWASP organization (Open Web Application Security Project) lists injections in their OWASP Top 10 2017 document as the number one threat to web application security.

In this tutorial i'll walk you through an example of SQL injection simulation...

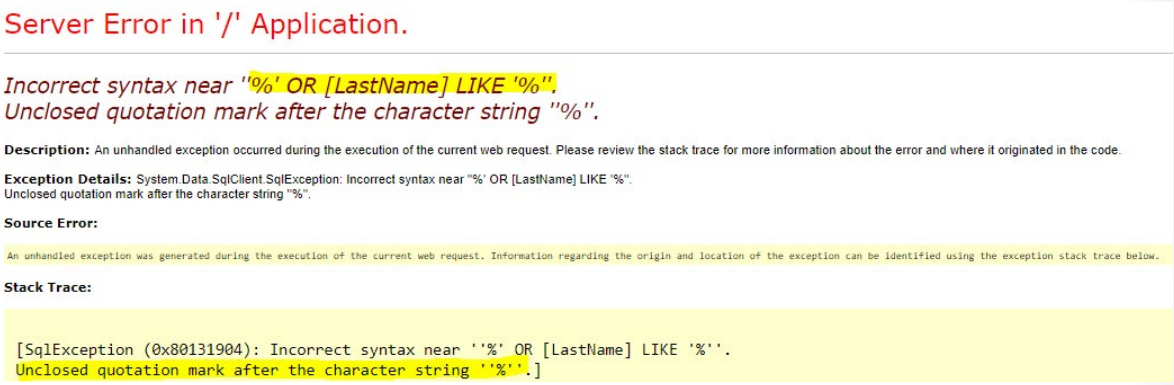
Getting Started

- Use the **SQL Injection ARM template here**[1] to provision a web app and a sql database with known sql injection vulnerability
- Ensure you can browse to the 'Contoso Clinic' web app provisioned in your sql injection resource group



How it works...

1. Navigate to the Patients view and in the search box type `"'"` and hit enter. You'll see an error page with SQL exception indicating that the search box is feeding the text into a SQL statement



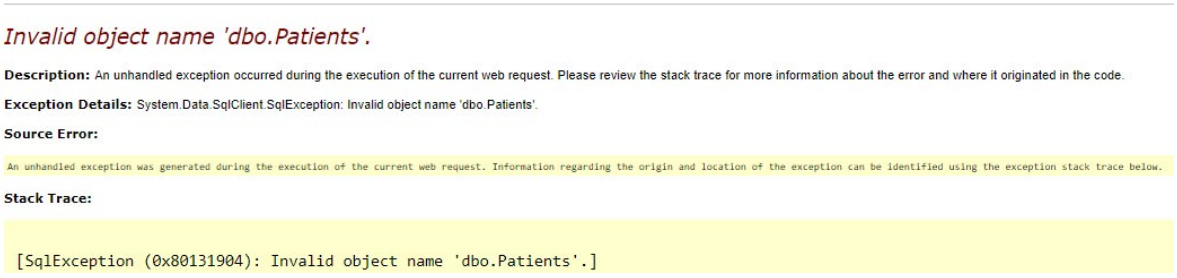
The helpful error message is enough to guess that the text in the search box is being appended into the sql statement.

2. Next try passing SQL statement `'AND FirstName = 'Kim'--` in the search box. You'll see that the results in the list below are filtered down to only show the entry with firstname Kim

1 https://azure.microsoft.com/en-us/resources/templates/101-sql-injection-attack-prevention/

3. You can try to order the list by SSN by using this statement in the search box `'order by SSN--`

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



There's more

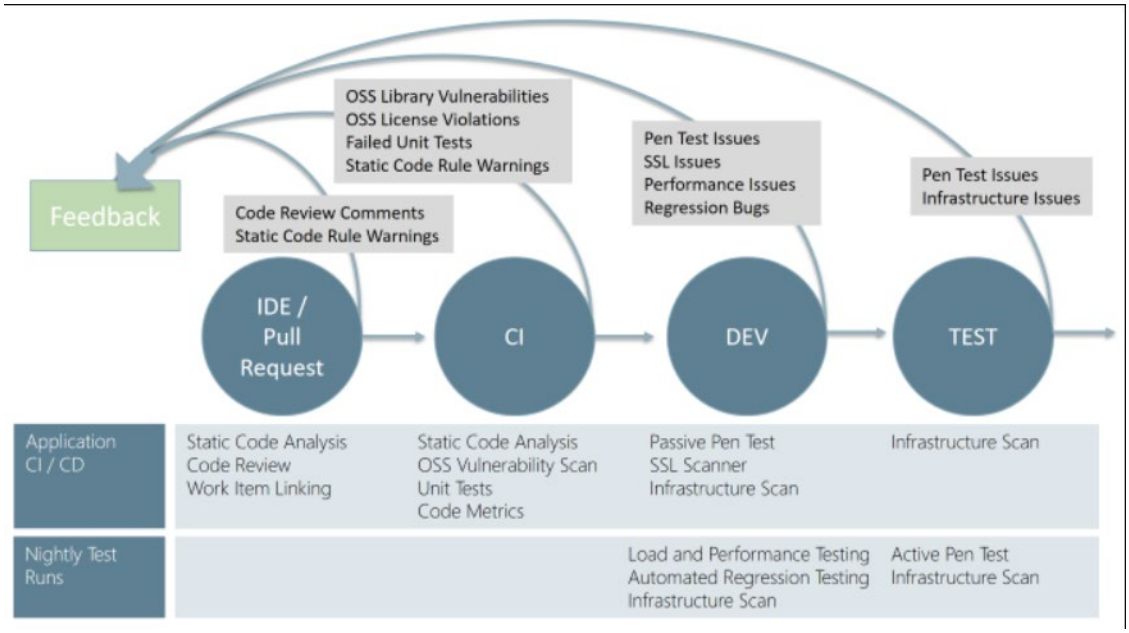
The Azure security centre team has other **playbooks**[2] you can look at to learn how vulnerabilities are exploited to trigger a virus attack and a DDoS attack.

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

Implement Secure & Compliant Development Processes

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.



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

CI (Continuous Integration)

The CI build should be executed as part of the pull request (PR-CI) process and once the merge is complete. Typically, the primary difference between the two runs is that the PR-CI process doesn't need to do any of the packaging/staging that is done in the CI build. These CI builds should run static code analysis tests to ensure that the code is following all rules for both maintenance and security. Several tools can be used for this.

- 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

Many of the tools seamlessly integrate into the Azure Pipelines build process. Visit the VSTS Marketplace for more information on the integration capabilities of these tools.

In addition to code quality being verified with the CI build, two other tedious or ignored validations are scanning 3rd party packages for vulnerabilities and OSS license usage. Often when we ask about 3rd party package vulnerabilities and the licenses, the response is fear or uncertainty. Those organizations that are trying to manage 3rd party packages vulnerabilities and/or OSS licenses, explain that their process for doing so is tedious and manual. Fortunately, there are a couple of tools by WhiteSource Software that can make this identification process almost instantaneous. The tool runs through each build and reports all of the vulnerabilities and the licenses of the 3rd party packages. WhiteSource Bolt is a new option, which includes a 6-month license with your Visual Studio Subscription. Bolt provides a report of these items but doesn't include the advanced management and alerting capabilities that the full product offers. With new vulnerabilities being regularly discovered, your build reports could change even though your code doesn't. Checkmarx includes a similar WhiteSource Bolt integration so there could be some overlap between the two tools. **See, Manage your open source usage and security as reported by your CI/CD pipeline for more information about WhiteSource and the Azure Pipelines integration.**[3]

Application Deployment to Dev and Test

Application Deployment to DEV and TEST

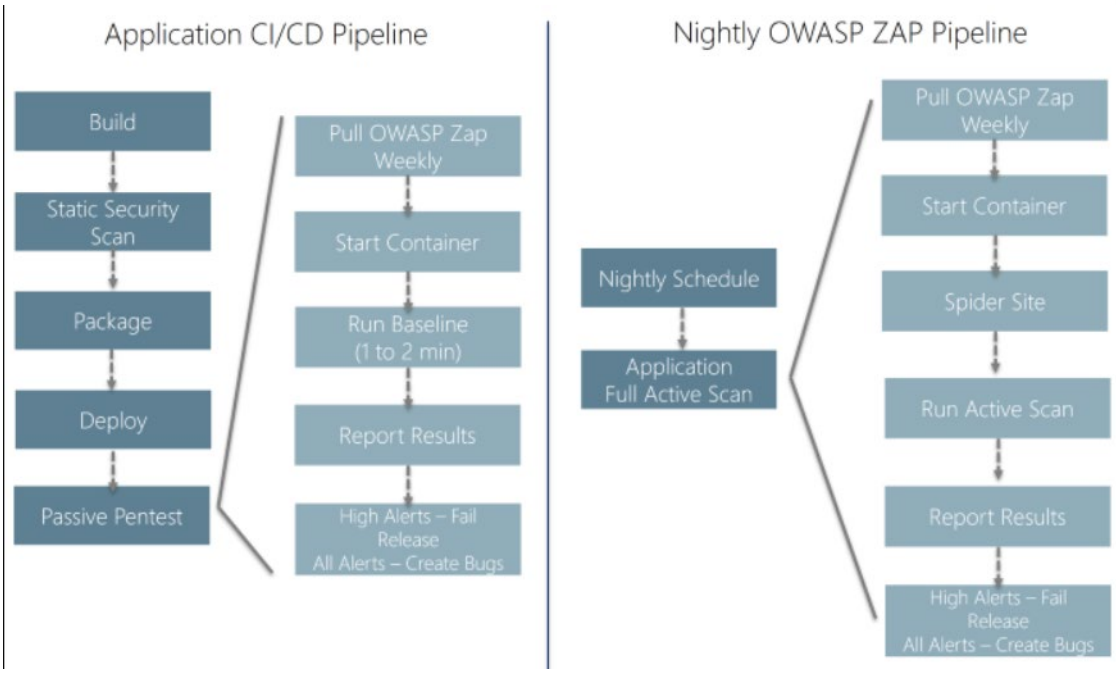
Once your code quality is verified, and the application is deployed to a lower environment like development or QA, the process should verify that there are not any security vulnerabilities in the running application. This can be accomplished by executing automated penetration test against the running application to scan it for vulnerabilities. There are different levels of tests that are categorized into passive tests and active tests. Passive tests scan the target site as is but don't try to manipulate the requests to expose additional vulnerabilities. These can run fast and are usually a good candidate for a CI process that you want to complete in a few minutes. Whereas the Active Scan can be used to simulate many techniques that hackers commonly use to attack websites. These tests can also be referred to dynamic or fuzz tests because the tests are often trying a large number of different combinations to see how the site reacts to verify that it doesn't reveal any information. These tests can run for much longer, and typically you don't want to cut these off at any particular time. These are better executed nightly as part of a separate Azure DevOps release.

One tool to consider for penetration testing is **OWASP ZAP**. OWASP is a worldwide not-for-profit organization dedicated to helping improve the quality of software. ZAP is a free penetration testing tool for beginners to professionals. ZAP includes an API and a weekly docker container image that can be integrated into your deployment process. **Refer to the oswa zap vsts extension**[4] repo for details on how to set up the integration. Here we're going to explain the benefits of including this into your process.

3 https://blogs.msdn.microsoft.com/visualstudioalmrangers/2017/06/08/manage-your-open-source-usage-and-security-as-reported-by-your-cicd-pipeline/

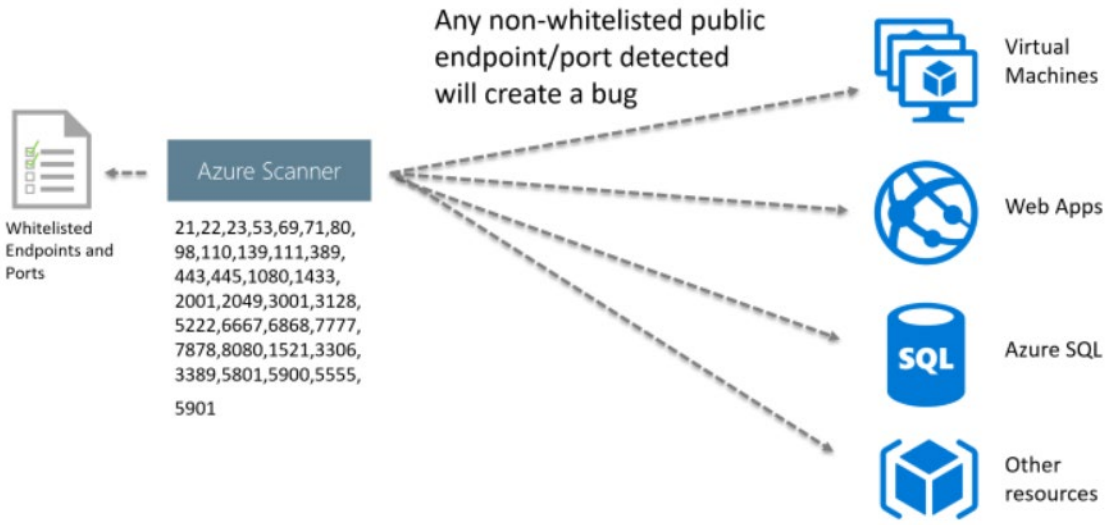
4 https://github.com/deliveron/owasp-zap-vsts-extension

The application CI/CD pipeline should run within a few minutes, so you don't want to include any long-running processes. The baseline scan is designed to identify vulnerabilities within a couple of minutes making it a good option for the application CI/CD pipeline. The Nightly OWASP ZAP can spider the website and run the full Active Scan to evaluate the most combinations of possible vulnerabilities. OWASP ZAP can be installed on any machine in your network, but we like to use the OWASP Zap/Weekly docker container within Azure Container Services. This allows for the latest updates to the image and also allows being able to spin up multiple instances of the image so several applications within an enterprise can be scanned at the same time. The following figure outlines the steps for both the Application CI/CD pipeline and the longer running Nightly OWASP ZAP pipeline.



Infrastructure Vulnerabilities

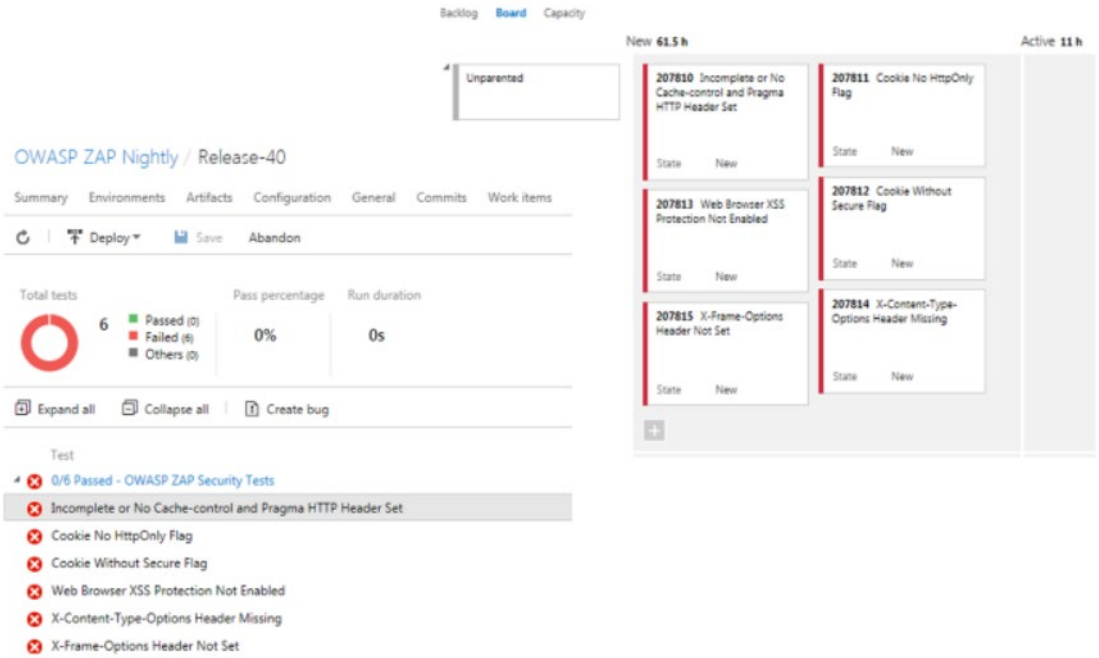
In addition to validating the application, the infrastructure should also be validated to check for any vulnerabilities. When using the public cloud such as Azure, deploying the application and shared infrastructure is easy, so it is important to validate that everything has been done securely. Azure includes many tools to help report and prevent these vulnerabilities including Security Center and Azure Policies. Also, we have set up a scanner that can ensure any public endpoints and ports have been whitelisted or else it will raise an infrastructure issue. This is run as part of the Network pipeline to provide immediate verification, but it also needs to be executed each night to ensure that there aren't any resources publicly exposed that should not be.



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.



Even with continuous security validation running against every change to help ensure new vulnerabilities are not introduced, hackers are continuously changing their approaches, and new vulnerabilities are being discovered. Good monitoring tools allow you to help detect, prevent, and remediate issues discovered while your application is running in production. Azure provides a number of tools that provide

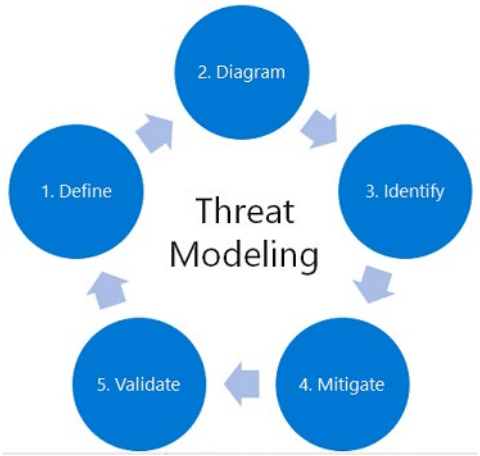
detection, prevention, and alerting using rules such as **OWASP Top 10**[5] / modSecurity and now even using machine learning to detect anomalies and unusual behavior to help identify attackers.

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.

Threat Modeling

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.

5 https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

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 overview - **https://docs.microsoft.com/en-us/azure/security/azure-security-threat-modeling-tool-feature-overview** [6]

Microsoft Threat Modeling Tool - **https://blogs.msdn.microsoft.com/secdevblog/2018/09/12/microsoft-threat-modeling-tool-ga-release/**

Threat-Modeling

Security cannot be a separate department in a silo. Security has to be part of DevOps, together they are called DevSecOps. The biggest weakness is not knowing the weakness in your solution. To re-mediate this, Microsoft has created a threat modelling tool, that helps you understand potential security vulnerabilities in your solution.

The Threat Modelling Tool is a core element of the Microsoft Security Development Life cycle (SDL). It allows software architects to identify and mitigate potential security issues early, when they are relatively easy and cost-effective to resolve. As a result, it greatly reduces the total cost of development. The tool has been designed with non-security experts in mind, making threat modelling easier for all developers by providing clear guidance on creating and analysing threat models.

The tool enables anyone to:

- Communicate about the security design of their systems
- Analyse those designs for potential security issues using a proven methodology
- Suggest and manage mitigation's for security issues

In this tutorial, we'll see how easy is it to use Threat Modelling tool to see potential vulnerabilities in your infrastructure solution that one should be thinking about when provisioning and deploying the Azure resources and the application solution into the solution...

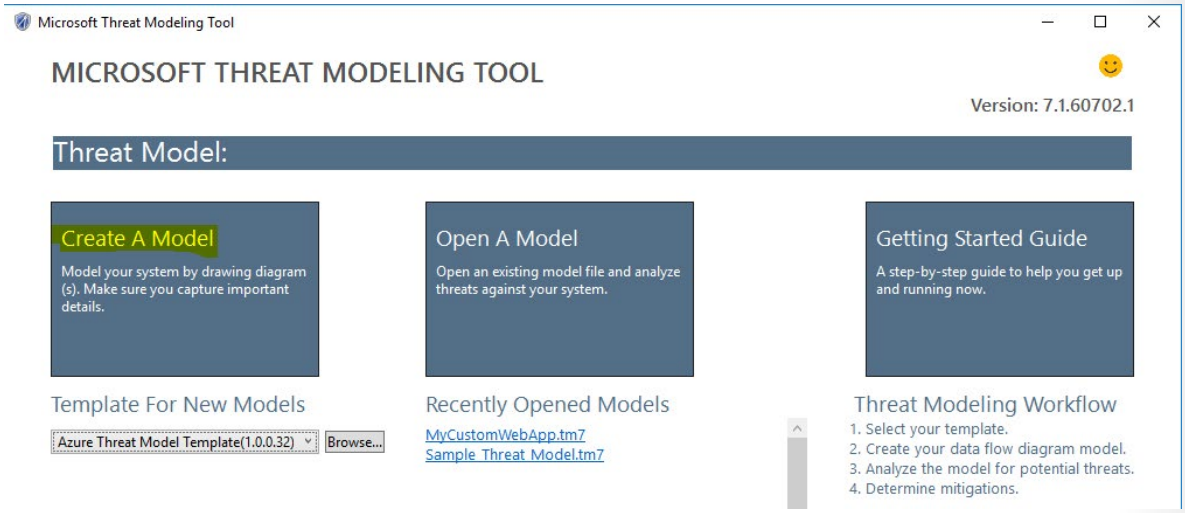
Getting Started

- Download and install the **Threat Modelling tool**[7]

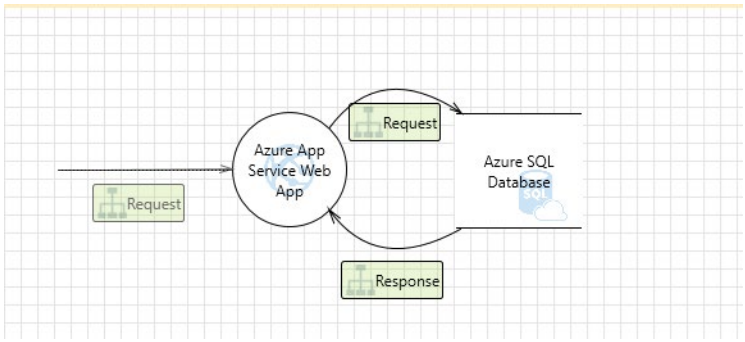
How to do it...

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

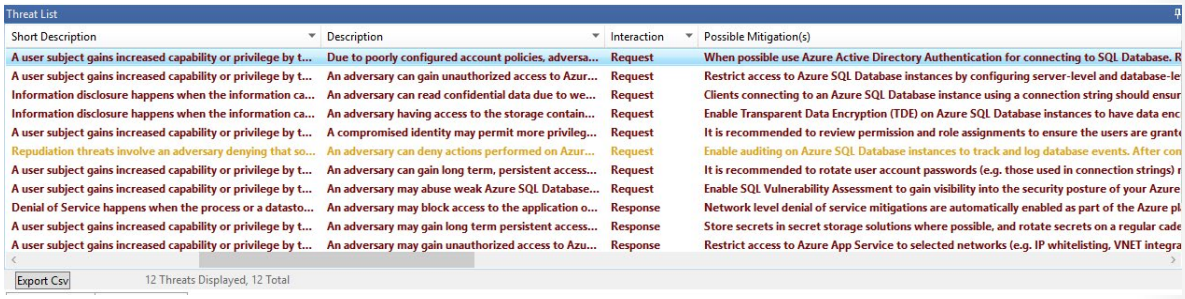
6 https://docs.microsoft.com/en-us/azure/security/azure-security-threat-modeling-tool-feature-overview
7 https://aka.ms/threatmodelingtool



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…

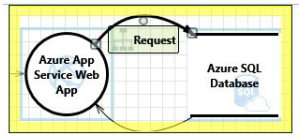


3. From the toolbar menu select View -> Analysis view, the analysis view will show you a full list of threats categorised by severity.



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 Mitigation(s):	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 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 Mitigation(s):	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 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 **here**[8]

8 https://docs.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats

Rethinking Application Config Data

Rethinking Application Config Data

Configuration information in files

The majority of application runtime environments include configuration information that's held in files deployed with the application. In some cases, it's possible to edit these files to change the application behavior after it's been deployed. However, changes to the configuration require the application be redeployed, often resulting in unacceptable downtime and other administrative overhead.

Local configuration files also limit the configuration to a single application, but sometimes it would be useful to share configuration settings across multiple applications. Examples include database connection strings, UI theme information, or the URLs of queues and storage used by a related set of applications. It's challenging to manage changes to local configurations across multiple running instances of the application, especially in a cloud-hosted scenario. It can result in instances using different configuration settings while the update is being deployed. In addition, updates to applications and components might require changes to configuration schemas. Many configuration systems don't support different versions of configuration information.

Example

It’s 2:00 AM, Adam is done making all changes to his super awesome code piece, the tests are all running fine, he hit commit -> push -> all commits pushed successfully to git. Happily, he drives back home. Ten mins later he gets a call from the SecurityOps engineer, – Adam did you push the Secret Key to our public repo?

YIKES!! that blah.config file Adam thinks, how could I have forgotten to include that in .gitignore, the nightmare has already begun ….

We can surely try to blame Adam here for committing the sin of checking in sensitive secrets and not following the recommended practices of managing configuration files, but the bigger question is that if the underlying toolchain had abstracted out the configuration management from the developer, this fiasco would have never happened!

History

The virus was injected a long time ago…
Since the early days of .NET, there has been the notion of app.config and web.config files which provide a playground for developers to make their code flexible by moving common configuration into these files. When used effectively, these files are proven to be worthy of dynamic configuration changes. However a lot of time we see the misuse of what goes into these files. A common culprit is how samples and documentation have been written, most samples out in the web would usually leverage these config files for storing key elements such as ConnectionStrings, and even password. The values might be obfuscated but what we are telling developers is that “hey, this is a great place to push your secrets!”. So, in a world where we are preaching using configuration files, we can’t blame the developer for not managing the governance of it. Don’t get me wrong; I am not challenging the use of Configuration here, it is an absolute need of any good implementation, I am instead debating the use of multiple json, XML, yaml files in maintaining configuration settings. Configs are great for ensuring the flexibility of the application, config files, however, in my opinion, are a pain to manage especially across environments.

A ray of hope: The DevOps movement

In recent years, we have seen a shift around following some great practices around effective DevOps and some great tools (Chef, Puppet) for managing Configuration for different languages. While these have helped to inject values during CI/CD pipeline and greatly simplified the management of configuration,

the blah.config concept has not completely moved away. Frameworks like ASP.NET Core support the notion of appSettings.json across environments, the framework has made it very effective to use these across environments through interfaces like IHostingEnvironment and IConfiguration but we can do better.

Separation of Concerns

Clean code: 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

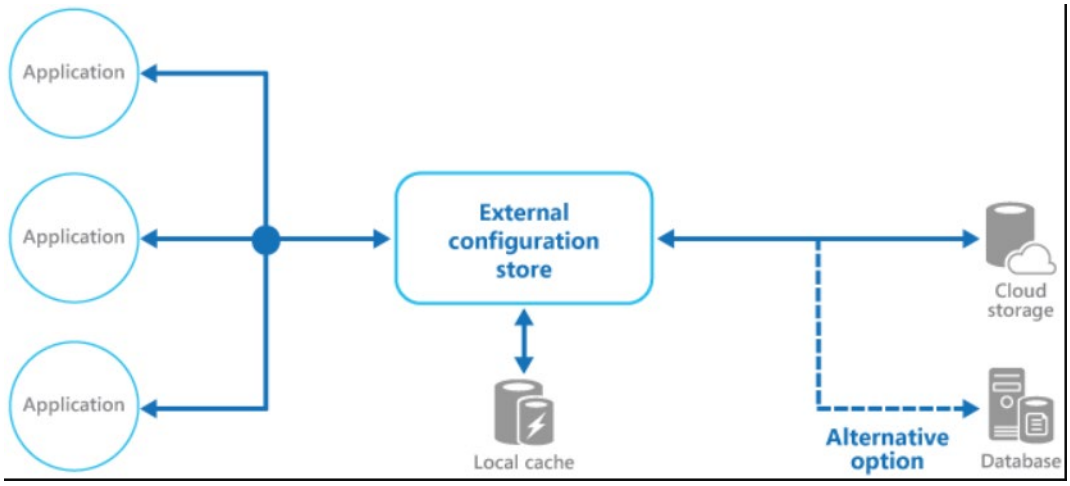
External Configuration Store pattern

Store the configuration information in external storage, and provide an interface that can be used to quickly and efficiently read and update configuration settings. The type of external store depends on the hosting and runtime environment of the application. In a cloud-hosted scenario it's typically a cloud-based storage service, but could be a hosted database or other system.

The backing store you choose for configuration information should have an interface that provides consistent and easy-to-use access. It should expose the information in a correctly typed and structured format. The implementation might also need to authorize users' access in order to protect configuration data, and be flexible enough to allow storage of multiple versions of the configuration (such as development, staging, or production, including multiple release versions of each one).

Many built-in configuration systems read the data when the application starts up, and cache the data in memory to provide fast access and minimize the impact on application performance. Depending on the type of backing store used, and the latency of this store, it might be helpful to implement a caching

mechanism within the external configuration store. For more information, see the Caching Guidance. The figure illustrates an overview of the External Configuration Store pattern with optional local cache.



This pattern is useful for:

- Configuration settings that are shared between multiple applications and application instances, or where a standard configuration must be enforced across multiple applications and application instances.
- A standard configuration system that doesn't support all of the required configuration settings, such as storing images or complex data types.
- As a complementary store for some of the settings for applications, perhaps allowing applications to override some or all of the centrally-stored settings.
- As a way to simplify administration of multiple applications, and optionally for monitoring use of configuration settings by logging some or all types of access to the configuration store.

Integrating Azure Key Vault with Azure Pipeline

Applications contain many secrets, such as connection strings, passwords, certificates, and tokens, which if leaked to unauthorized users can lead to a severe security breach. This can result in serious damage to the reputation of the organization and in compliance issues with different governing bodies. Azure Key Vault allows you to manage your organization's secrets and certificates in a centralized repository. The secrets and keys are further protected by Hardware Security Modules (HSMs). It also provides versioning of secrets, full traceability, and efficient permission management with access policies.

For more information on Azure Key Vault, visit **What is Azure Key Vault**[9].

In this tutorial we'll cover...

- We will create a key vault, from the Azure portal, to store a MySQL server password
- We will configure permissions to let a service principal to read the value
- We will retrieve the password in an Azure pipeline and passed on to subsequent tasks

9 https://docs.microsoft.com/en-us/azure/key-vault/key-vault-overview

How to do it...

Follow the steps in this lab for **integrating Azure KeyVault with Azure DevOps**[10].

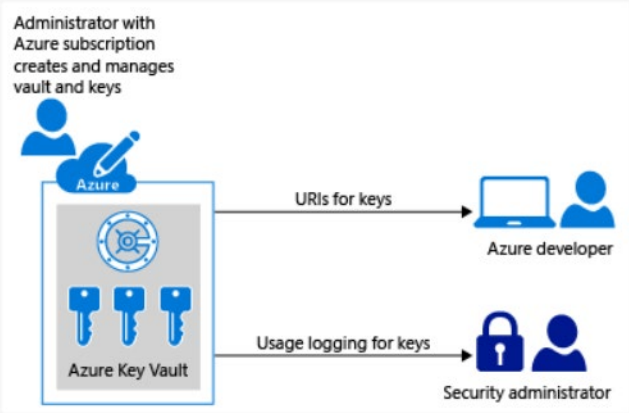
10 https://www.azuredevopslabs.com/labs/vstsextend/azurekeyvault/

Manage Secrets, Tokens & Certificates

Manage Secrets, Tokens & Certificates

Azure Key Vault helps solve the following problems:

- **Secrets Management** - Azure Key Vault can be used to Securely store and tightly control access to tokens, passwords, certificates, API keys, and other secrets.
- **Key Management** - Azure Key Vault can also be used as a Key Management solution. Azure Key Vault makes it easy to create and control the encryption keys used to encrypt your data.
- **Certificate Management** - Azure Key Vault is also a service that lets you easily provision, manage, and deploy public and private Secure Sockets Layer/Transport Layer Security (SSL/TLS) certificates for use with Azure and your internal connected resources.
- **Store secrets backed by Hardware Security Modules** - The secrets and keys can be protected either by software or FIPS 140-2 Level 2 validates HSMs.



Why use Azure Key Vault?

Centralize application secrets

Centralizing storage of application secrets in Azure Key Vault allows you to control their distribution. Key Vault greatly reduces the chances that secrets may be accidentally leaked. When using Key Vault, application developers no longer need to store security information in their application. This eliminates the need to make this information part of the code. For example, an application may need to connect to a database. Instead of storing the connection string in the app codes, store it securely in Key Vault.

Your applications can securely access the information they need by using URIs that allow them to retrieve specific versions of a secret after the application's key or secret is stored in Azure Key Vault. This happens without having to write custom code to protect any of the secret information.

Securely store secrets and keys

Secrets and keys are safeguarded by Azure, using industry-standard algorithms, key lengths, and hardware security modules (HSMs). The HSMs used are Federal Information Processing Standards (FIPS) 140-2 Level 2 validated.

Access to a key vault requires proper authentication and authorization before a caller (user or application) can get access. Authentication establishes the identity of the caller, while authorization determines the operations that they are allowed to perform.

Authentication is done via Azure Active Directory. Authorization may be done via role-based access control (RBAC) or Key Vault access policy. RBAC is used when dealing with the management of the vaults and key vault access policy is used when attempting to access data stored in a vault.

Azure Key Vaults may be either software- or hardware-HSM protected. For situations where you require added assurance you can import or generate keys in hardware security modules (HSMs) that never leave the HSM boundary. Microsoft uses Thales hardware security modules. You can use Thales tools to move a key from your HSM to Azure Key Vault.

Finally, Azure Key Vault is designed so that Microsoft does not see or extract your data.

Monitor access and use

Once you have created a couple of Key Vaults, you will want to monitor how and when your keys and secrets are being accessed. You can do this by enabling logging for Key Vault. You can configure Azure Key Vault to:

- Archive to a storage account.
- Stream to an event hub.
- Send the logs to Log Analytics.

You have control over your logs and you may secure them by restricting access and you may also delete logs that you no longer need.

Simplified administration of application secret

When storing valuable data, you must take several steps. Security information must be secured, it must follow a lifecycle, it must be highly available. Azure Key Vault simplifies a lot of this by:

- Removing the need for in-house knowledge of Hardware Security Modules
- Scaling up on short notice to meet your organization's usage spikes.
- Replicating the contents of your Key Vault within a region and to a secondary region. This ensures high availability and takes away the need of any action from the administrator to trigger the fail over.
- Providing standard Azure administration options via the portal, Azure CLI and PowerShell.
- Automating certain tasks on certificates that you purchase from Public CAs, such as enrollment and renewal.

In addition, Azure Key Vaults allow you to segregate application secrets. Applications may access only the vault that they are allowed to access, and they be limited to only perform specific operations. You can create an Azure Key Vault per application and restrict the secrets stored in a Key Vault to a specific application and team of developers.

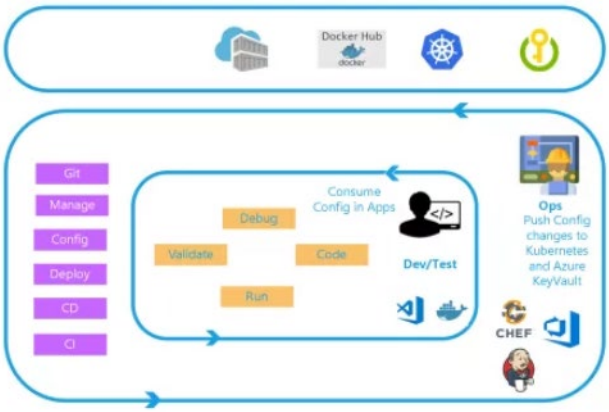
Integrate with other Azure services

As a secure store in Azure, Key Vault has been used to simplify scenarios like Azure Disk Encryption, the always encrypted functionality in SQL server and Azure SQL Database, Azure web apps. Key Vault itself can integrate with storage accounts, event hubs and log analytics.

DevOps Inner and Outer Loop

While you can store configuration and secrets together, it violates our separation of concern principle, so the recommendation is to leverage a separate store for persisting secrets. This allows a secure channel for sensitive configuration data such as ConnectionStrings, enables the operations team to have Credentials, Certificate, Token in one repository and minimizes the security risk in case the Configuration Store gets compromised.

The below diagram shows how these roles play together in a DevOps Inner loop and Outer loop. The inner loop is focused on the developer teams iterating over their solution development; they consume the configuration published by the outer loop. The Ops Engineer govern the Configuration management and push changes into Azure KeyVault and Kubernetes that are further isolated per environment.



Kubernetes and Azure Key Vault

Kubernetes and Azure Key Vault to the rescue

So we have talked a lot about governance elements and the need to move out of configuration files, lets now use some of the magic available in Kubernetes and Azure Key Vault to implement this. In particular, we would be using the following capabilities of these technologies:

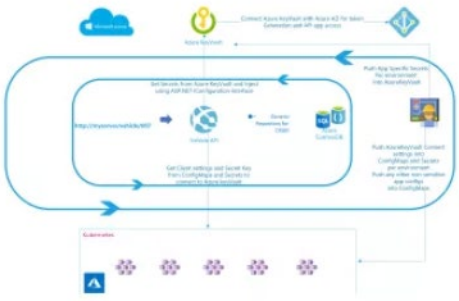
- Azure Key Vault Secret store
- Kubernetes ConfigMaps
- Kubernetes Secrets

Why not just use Kubernetes you may ask?

That's a valid question since Kubernetes supports both a ConfigMap store and Secret store. Remember our principle around the separation of concerns and how we can ensure enhanced security by separating out configuration from secrets. Having secrets in the same cluster as the configuration store can make them prone to higher risks. An additional benefit is Maintainability. Azure Key Vault gives the ability to provide a distributed "Secure Store as a Service" option that provides not just secret management but also Certificates and Key management as part of the service. The SecOps engineers can lock down access to the store without need of cluster admins permissions which allows for clear delineation of responsibilities and access.

The Scenario

We will be building a Vehicle microservice which provides CRUD operations for sending vehicle data to a CosmosDB document store. The sample micro-service needs to interact with the Configuration stores to get values such as connectionstring, database name, collection name, etc. We interact with Azure Key Vault for this purpose. Additionally, the application needs the Authentication token for Azure Key Vault itself, these details along with other Configuration will be stored in Kubernetes.



Mapping the above diagram to our roles and responsibilities earlier:

The Ops engineer/scripts are the Configuration Custodian and they are the only ones who work in the outer loop to manage all the configuration. They would have CI/CD scripts that would inject these configurations or use popular framework tools to enable the insertion during the build process.

The Vehicle API is the ASP.NET Core 2.0 application and is the Configuration Consumer here; the consumer is interested in getting the values without really worrying about what the value is and which environment it belongs to. The ASP.NET Core framework provides excellent support for this through its Configuration extensibility support. You can add as many providers as you like and they can be bound an IConfiguration object which provides access to all the configuration. In the below code snippet, we provide the configuration to be picked up from environment variables instead of a configuration file. The ASP.NET Core 2.0 framework also supports extensions to include Azure Key Vault as a configuration provider, and under the hood, the Azure Key Vault client allows for secure access to the values required by the application.

```
// add the environment variables to config
  config.AddEnvironmentVariables();

 // add azure key vault configuration using environment variables
 var buildConfig = config.Build();

 // get the key vault  uri
 var vaultUri = buildConfig["kvuri"].Replace("{vault-name}",   buildCon-
fig["vault"]);
 // setup KeyVault store for getting configuration values
 config.AddAzureKeyVault(vaultUri, buildConfig["clientId"], buildConfig["cli-
entSecret"]);
```

AzureKeyVault is the Secret Store for all the secrets that are application specific. It allows for the creation of these secrets and also managing the lifecycle of them. It is recommended that you have a separate Azure KeyVault per environment to ensure isolation. The following command can be used to add a new configuration into KeyVault:

```
#Get a list of existing secrets
az keyvault secret list --vault-name  -o table

#add a new secret to keyvault
az keyvault secret set -n MySecret --value MyValue --description "my custom
value" --vault-name
```

Kubernetes is the Configuration Store and serves multiple purposes here:

1. Creation of the ConfigMaps and Secret: Since we are injecting the Azure KeyVault authentication information using Kubernetes, the Ops engineer will provide these values using two constructs provided in the Kubernetes infrastructure. ConfigMaps and Secrets. The following shows how to add config maps and secrets from the Kubernetes command line:

```
kubectl create configmap vault --from-literal=vault=
kubectl create configmap kvuri --from-literal=kvuri=https://{vault-name}.
vault.azure.net/
kubectl create configmap clientid --from-literal=clientId=
kubectl create secret generic clientsecret --from-literal=clientSecret=
```

The clientsecret is the only piece of secure information we store in Kubernetes, all the application specific secrets are stored in Azure KeyVault. This is comparatively safer since the above scripts do not need to go in the same git repo. (so we don't check them in by mistake) and can be managed separately. We still control the expiry of this secret using Azure KeyVault, so the Security engineer still has full control over access and permissions.

1. Injecting Values into the Container: During runtime, Kubernetes will automatically push the above values as environment variables for the deployed containers, so the system does not need to worry about loading them from a configuration file. The Kubernetes configuration for the deployment looks like below. As you would notice, we only provide a reference to the ConfigMaps and Secret that have been created instead of punching in the actual values.

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: vehicle-api-deploy #name for the deployment
  labels:
    app: vehicle-api #label that will be used to map the service, this tag
is very important
spec:
  replicas: 1
  selector:
    matchLabels:
      app: vehicle-api #label that will be used to map the service, this
tag is very important
  template:
    metadata:
      labels:
        app: vehicle-api #label that will be used to map the service, this
tag is very important
    spec:
      containers:
      - name: vehicleapi #name for the container configuration
        image: <yourdockerhub>/<youdockerimage>:<youdockertagversion> #
**CHANGE THIS: the tag for the container to be deployed
        imagePullPolicy: Always #getting latest image on each deployment
        ports:
```

```
        - containerPort: 80 #map to port 80 in the docker container
        env: #set environment variables for the docker container using
configMaps and Secret Keys
        - name: clientId
          valueFrom:
            configMapKeyRef:
              name: clientid
              key: clientId
        - name: kvuri
          valueFrom:
            configMapKeyRef:
              name: kvuri
              key: kvuri
        - name: vault
          valueFrom:
            configMapKeyRef:
              name: vault
              key: vault
        - name: clientsecret
          valueFrom:
            secretKeyRef:
              name: clientsecret
              key: clientSecret
      imagePullSecrets: #secret to get details of private repo, disable
this if using public docker repo
      - name: regsecret
```

Implement Tools for Managing Security and Compliance

SonarCloud

Technical debt can be classified as the measure between the codebase's current state and an optimal state. Technical debt saps productivity by making code hard to understand, easy to break, and difficult to validate, in turn creating unplanned work, ultimately blocking progress. Technical debt is inevitable! It starts small and grows over time through rushed changes, lack of context, and lack of discipline. Organizations often find that more than 50% of their capacity is sapped by technical debt. The hardest part of fixing technical debt is knowing where to start. SonarQube is an open source platform that is the de facto solution for understanding and managing technical debt. In this recipe, we'll learn how to leverage SonarQube in a build pipeline to identify technical debt.

Getting ready

SonarQube is an open platform to manage code quality. Originally famous in the Java community, SonarQube now supports over 20 programming languages. The joint investments made by Microsoft and SonarSource make SonarQube easier to integrate in Pipelines and better at analyzing .NET-based applications. You can read more about the capabilities offered by SonarQube here: https://www.sonarqube.org/. SonarSource, the company behind SonarQube offers a hosted SonarQube environment called as SonarCloud.

In this tutorial, we'll be analyzing the technical debt in one of the dotnet core sample repositories in the partsunlimited team project. If you don't already have an instance of SonarCloud, then set one up by following the instructions here: https://sonarcloud.io/about/sq

To get started with SonarCloud, you'll also need to install the SonarCloud pipeline tasks to your Azure DevOps. You can do so from the marketplace: https://marketplace.visualstudio.com/items?itemName=SonarSource.sonarcloud

How to do

Follow along this lab on using **SonarCloud with AzureDevOps**[11].

Lab - SonarCloud

In this lab, Driving continuous quality of your code with SonarCloud, you will you will learn how to integrate Visual Studio Team Services with SonarCloud. You will learn how to:

- Setup a VSTS project and CI build to integrate with SonarCloud
- Analyze SonarCloud reports
- Integrate static analysis into the VSTS pull request process

✓ Note that you must have already completed the prerequisite labs in the Welcome section.

Driving continuous quality of your code with SonarCloud - **https://www.azuredevopslabs.com/labs/vstsextend/sonarcloud/**

11 https://www.azuredevopslabs.com/labs/vstsextend/sonarcloud/

Implement Continuous Security Validation

Today developers don't hesitate to use components that are available in public package sources (such as npm or NuGet). With the aim of faster delivery and better productivity, using open source software (OSS) components is encouraged across many organisations. However, as the dependency on these third-party OSS components increases, the risk of security vulnerabilities or hidden license requirements also increases compliance issues.

For a business, this is critical, as issues related to compliance, liabilities and customer personally identifiable information (PII) can cause a great deal of privacy and security concerns. Identifying such issues early on in the release cycle gives you an advanced warning and allows you enough time to fix the issues. There are many tools such as WhiteSource, Veracode, and Checkmarx that are available, can scan for these vulnerabilities within the build and release pipelines.

In this tutorial, we will explore how you can make use of these tools to scan vulnerabilities in your dependent NuGet packages. We'll be using the free WhiteSource Bolt extension to scan for dependencies during the CI build.

How to do it

Follow along in the Azure DevOps Lab **Manage Open source security and and license with WhiteSource**[12]

Lab - WhiteSource

In this lab, Managing Open-source security and license with WhiteSource, you will use WhiteSource Bolt with Azure DevOps to automatically detect alerts on vulnerable open source components, outdated libraries, and license compliance issues in your code. You will be using WebGoat, a deliberately insecure web application, maintained by OWASP designed to teach web application security lessons. You will learn how to:

- Detect and remedy vulnerable open source components.
- Generate comprehensive open source inventory reports per project or build.
- Enforce open source license compliance, including dependencies' licenses.
- Identify outdated open source libraries with recommendations to update.

✓ Note that you must have already completed the prerequisite labs in the Welcome section.

Managing Open-source security and license with WhiteSource - **https://www.azuredevopslabs.com/labs/vstsextend/WhiteSource/**

Securing Infrastructure with AzSk

With the adoption of infrastructure as code and development teams taking on more of the Ops activities, organisations using Azure for enterprise hosting are practically at risk of breach with every new release they roll out to these environments… In this tutorial we'll walk through AzSDK Security Verification Tests that can be setup in the CI/CD pipeline using Azure DevOps to automate the inspection of your infrastructure in Azure and block releases that can potentially compromise your infrastructure.

The DevOps way of working coupled with the use of cloud technologies has practically removed the biggest barriers from the software delivery life-cycle… Development teams don't care about security as

12 https://www.azuredevopslabs.com/labs/vstsextend/WhiteSource/

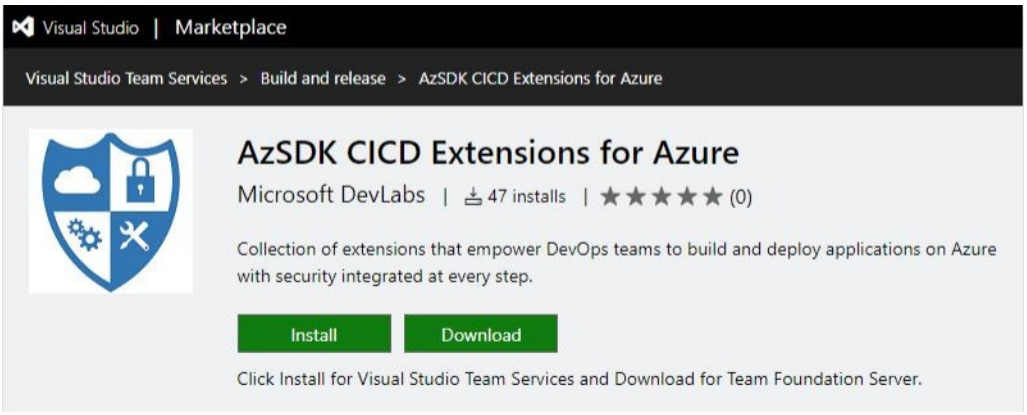
much as they should. It's hard to blame the development teams though, the tools used by security are not easy to understand… The reports are long and hard to interpret… The approach of compliance driven security doesn't practically fit into the fast pace of software delivery we are used to today.

DevSecOps helps bring a fresh perspective by introducing a culture of making everyone accountable for security, using automation to move the process of inspection left and overall looking at security with a 360 lens.

Anyone doing cloud at scale is likely to have multiple Azure subscriptions with hundreds of resource groups in Azure dedicated to the application in question. Resource Groups comprising of resources such as Web App's, Azure Functions, Blob Storage, Redis Cache, App Insights, Service Bus, SQL warehouse among some other Azure resource types. The ratio of security consultants to the number of releases makes it practically impossible for security to inspect the changes to these resource types to guarantee compliance of best practices.

Getting started

- Start by installing the AzSDK extension from the Azure DevOps **marketplace**[13]



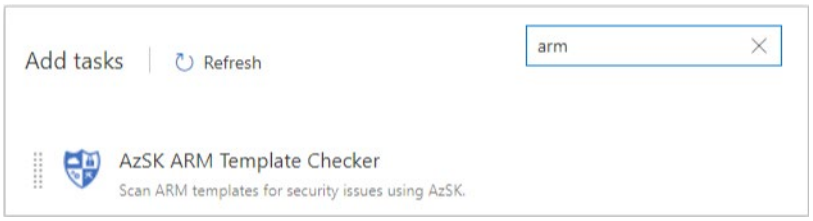
How to do it

The AzSk extension gives you the ability to both scan ARM templates to identify shortcoming in your ARM templates and also gives you the ability to scan actual Azure resources provisioned in an Azure subscription for compliance to best practices. We'll cover both in this tutorial.

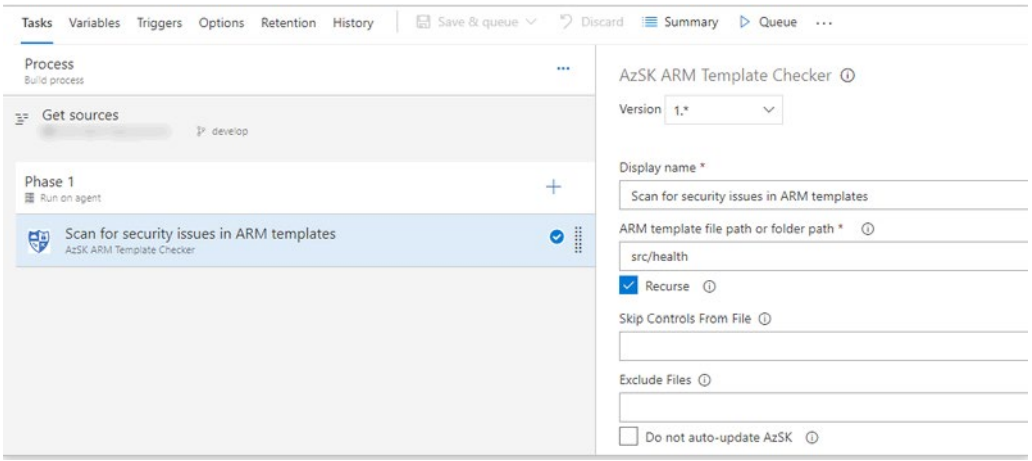
Scanning Azure Resource Manager (ARM) Templates for best practices

- Create a build definition that is mapped to the git repository where you store your ARM templates
- Next add the AzSK ARM Template Checker task to the build pipeline

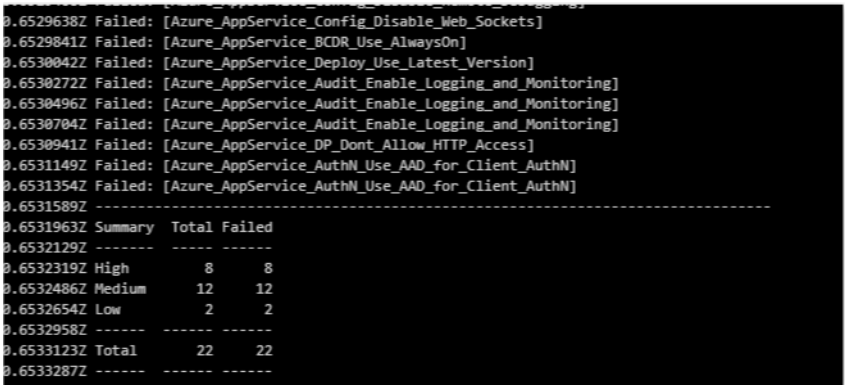
13 https://marketplace.visualstudio.com/items?itemName=azsdktm.AzSDK-task



- Next, configure the task - To start scanning, you just need to provide the root folder where you have ARM templates (and optionally mark it to scan recursively).



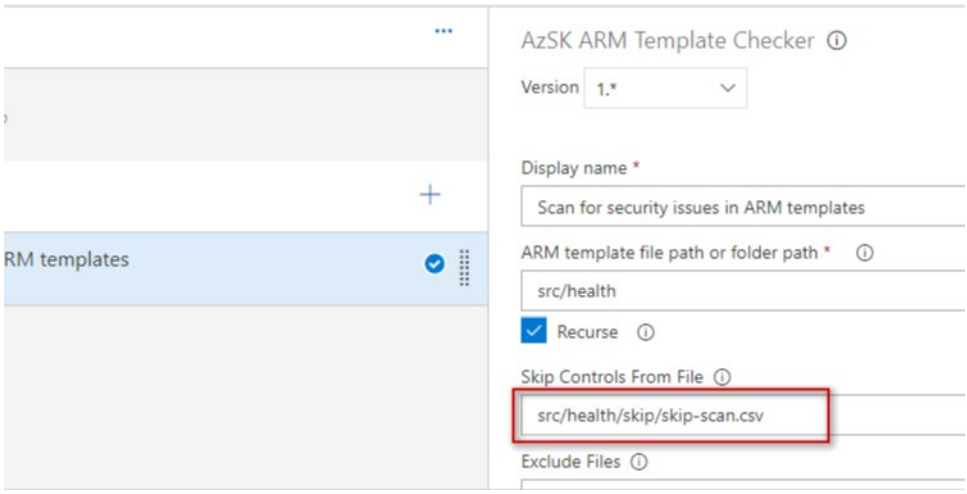
- Run the build and let the build complete. You will see glimpse of the scan in the build output itself.



- Once the build completes, you will find that task has attached all the results to the build log. You will be surprised to see the issues you find in your ARM templates.

FeatureNa	Status	Supporte	Severity	PropertyP	LineNumb	CurrentVa	ExpectedV	ExpectedV	ResourceF	ResourceL	Description
AppServic	Failed	Microsoft.	Medium	Not found	-1		$.sku.capa	GreaterTh	resources	30	App Service must be deplsoyed on a minimum of two instances to ensure availability
AppServic	Failed	Microsoft.	High	Not found	-1		$.properti	'False'	resources	30	Remote debugging must be turned off for App Service
AppServic	Failed	Microsoft.	Medium	Not found	-1		$.properti	'False'	resources	30	Web Sockets should be disabled for App Service
AppServic	Failed	Microsoft.	Medium	Not found	-1		$.properti	'True'	resources	30	'Always On' should be configured for App Service
AppServic	Failed	Microsoft.	Low	Not found	-1		$.properti	Allow 'v4.	resources	30	The latest version of .NET framework version should be used for App Service
AppServic	Failed	Microsoft.	Medium	Not found	-1		$.properti	'True'	resources	30	Auditing and Monitoring must be enabled for App Service
AppServic	Failed	Microsoft.	Medium	Not found	-1		$.properti	'True'	resources	30	Auditing and Monitoring must be enabled for App Service
AppServic	Failed	Microsoft.	Medium	Not found	-1		$.properti	'True'	resources	30	Auditing and Monitoring must be enabled for App Service
AppServic	Failed	Microsoft.	High	Not found	-1		$.properti	'True'	resources	30	App Service must only be accessible over HTTPS
AppServic	Failed	Microsoft.	High	Not found	-1		$.properti	'True'	resources	30	App Service must authenticate users using Azure Active Directory backed credentials
AppServic	Failed	Microsoft.	High	Not found	-1		$.properti	Non-null s	resources	30	App Service must authenticate users using Azure Active Directory backed credentials
Storage	Failed	Microsoft.	Medium	Not found	-1		$.properti	'True'	resources	95	HTTPS protocol must be used for accessing Storage Account resources
AppServic	Failed	Microsoft.	Medium	resources	62	"[paramet	$.sku.capa	GreaterTh	resources	49	App Service must be deplsoyed on a minimum of two instances to ensure availability
AppServic	Failed	Microsoft.	High	Not found	-1		$.properti	'False'	resources	49	Remote debugging must be turned off for App Service
AppServic	Failed	Microsoft.	Medium	Not found	-1		$.properti	'False'	resources	49	Web Sockets should be disabled for App Service
AppServic	Failed	Microsoft.	Medium	Not found	-1		$.properti	'True'	resources	49	'Always On' should be configured for App Service

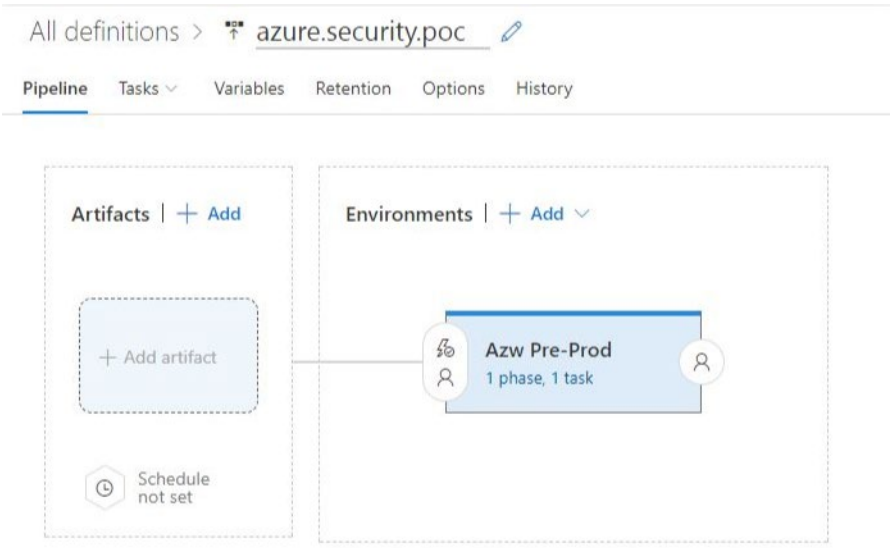
- Occasionally you will find issues which you have decided as safe to ignore. The task allows you to skip such failures from the scan so that you will not get alerted as a security issue or cause in build failures. The way you configure this is simple - Use the generated csv file and keep only entries you need to ignore from the scan and commit to your repository as a csv file. Then specify this file in the task as below.



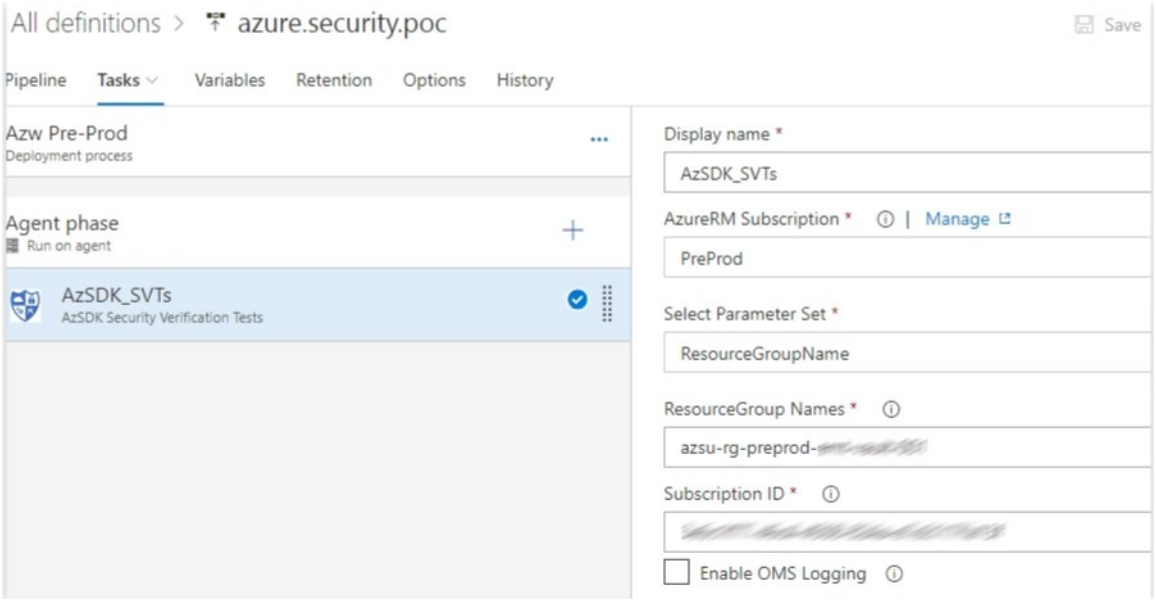
The task currently scans App Service, Storage, SQL, CDN, Traffic Manager, Document DB, Redis Cache, and Data Lake services only.

Scanning Azure Resources for best practices

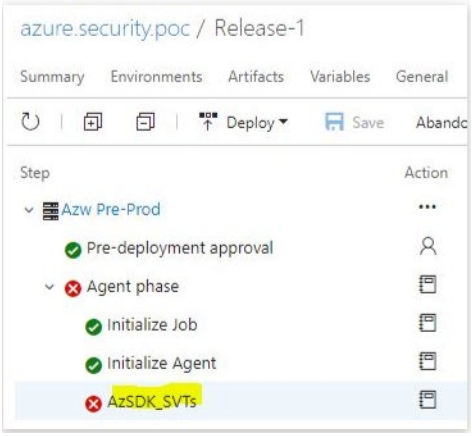
- Create a new release Azure pipeline, add an environment



- In the newly added environment add the azSdk task and configure the Azure DevOps SPN for the Azure subscription, specify the name of the Azure resource group you wish to inspect as well as the Azure subscription id the resource group resides in.



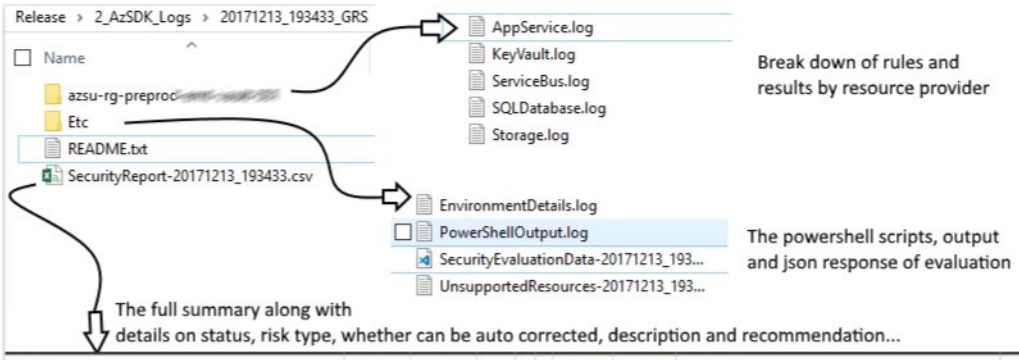
- Run the release pipeline and wait for the release to complete… The default security policies are evaluated, you have the option of customizing or creating your subset… If the setup isn't compliant the release pipeline will fail by default…



Now review the detailed logs from the inspection…

ControlID	Status	FeatureNa	Resou	Re	Ch	ControlSe	Supports	Description	Recomme
Azure_Storage_AuthN_Dont_Allow_Anonymous	Passed	Storage	azsu-i	azsut		High	Yes	The Access Type for containers must not be set to '	Run comn
Azure_Storage_Audit_Issue_Alert_AuthN_Req	Failed	Storage	azsu-i	azsut		High	Yes	Alert rules must be configured for tracking anonyn	Run comn
Azure_Storage_Deploy_Use_Geo_Redundant	Passed	Storage	azsu-i	azsut		High	Yes	Use geo-redundant Storage Accounts	Note: Ena
Azure_Storage_DP_Encrypt_At_Rest_Blob	Passed	Storage	azsu-i	azsut		High	Yes	Sensitive data in Storage Blob must be encrypted a	Run comn
Azure_Storage_Audit_AuthN_Requests	Failed	Storage	azsu-i	azsut		Medium	Yes	Storage Account must be configured to log and mo	Run comn
Azure_Storage_DP_Encrypt_In_Transit	Failed	Storage	azsu-i	azsut		High	Yes	HTTPS protocol must be used for accessing Storage	Run comn
Azure_Storage_AuthZ_Use_IP_ACL	Manual	Storage	azsu-i	azsut		Medium	No	Use IP-restrictions in SAS tokens to only permit ac	Restrict st
Azure_Storage_AuthZ_Clients_Use_SAS	Manual	Storage	azsu-i	azsut		High	No	End user/client apps should access Storage Accour	Do not use
Azure_Storage_DP_Rotate_Keys	Manual	Storage	azsu-i	azsut		Medium	No	Storage Account keys must be rotated periodically	Rotate Stc
Azure_Storage_AuthZ_Allow_Limited_Access_to_Ser	Manual	Storage	azsu-i	azsut		High	No	Use Stored Access Policies with least privileges ne	Create a S
Azure_KeyVault_AuthN_Use_Cert_Auth_for_Apps	Error	KeyVault	azsu-i	azsu-		High	No	Azure Active Directory applications, which have ac	Remove a

The logs give you the full picture! The extension gives you a criticality rating and also tells you if the issue can be auto fixed, a description and recommendation for the specific line item as well…



ControlID	Status	FeatureNa	Resou	Re	Ch	ControlSe	Supports	Description	Recomme
Azure_Storage_AuthN_Dont_Allow_Anonymous	Passed	Storage	azsu-i	azsut		High	Yes	The Access Type for containers must not be set to '	Run comn
Azure_Storage_Audit_Issue_Alert_AuthN_Req	Failed	Storage	azsu-i	azsut		High	Yes	Alert rules must be configured for tracking anonyn	Run comn
Azure_Storage_Deploy_Use_Geo_Redundant	Passed	Storage	azsu-i	azsut		High	Yes	Use geo-redundant Storage Accounts	Note: Ena
Azure_Storage_DP_Encrypt_At_Rest_Blob	Passed	Storage	azsu-i	azsut		High	Yes	Sensitive data in Storage Blob must be encrypted a	Run comn
Azure_Storage_Audit_AuthN_Requests	Failed	Storage	azsu-i	azsut		Medium	Yes	Storage Account must be configured to log and mo	Run comn
Azure_Storage_DP_Encrypt_In_Transit	Failed	Storage	azsu-i	azsut		High	Yes	HTTPS protocol must be used for accessing Storage	Run comn
Azure_Storage_AuthZ_Use_IP_ACL	Manual	Storage	azsu-i	azsut		Medium	No	Use IP-restrictions in SAS tokens to only permit ac	Restrict st
Azure_Storage_AuthZ_Clients_Use_SAS	Manual	Storage	azsu-i	azsut		High	No	End user/client apps should access Storage Accour	Do not use
Azure_Storage_DP_Rotate_Keys	Manual	Storage	azsu-i	azsut		Medium	No	Storage Account keys must be rotated periodically	Rotate Stc
Azure_Storage_AuthZ_Allow_Limited_Access_to_Ser	Manual	Storage	azsu-i	azsut		High	No	Use Stored Access Policies with least privileges ne	Create a S
Azure_KeyVault_AuthN_Use_Cert_Auth_for_Apps	Error	KeyVault	azsu-i	azsu-		High	No	Azure Active Directory applications, which have ac	Remove a

Summary

Pushing Security left does not mean using the old ways of security compliance checks earlier in the life-cycle, instead it is an opportunity to leverage the DevOps mindset to innovate and automate the security inspection to allow development teams to release features with confidence.

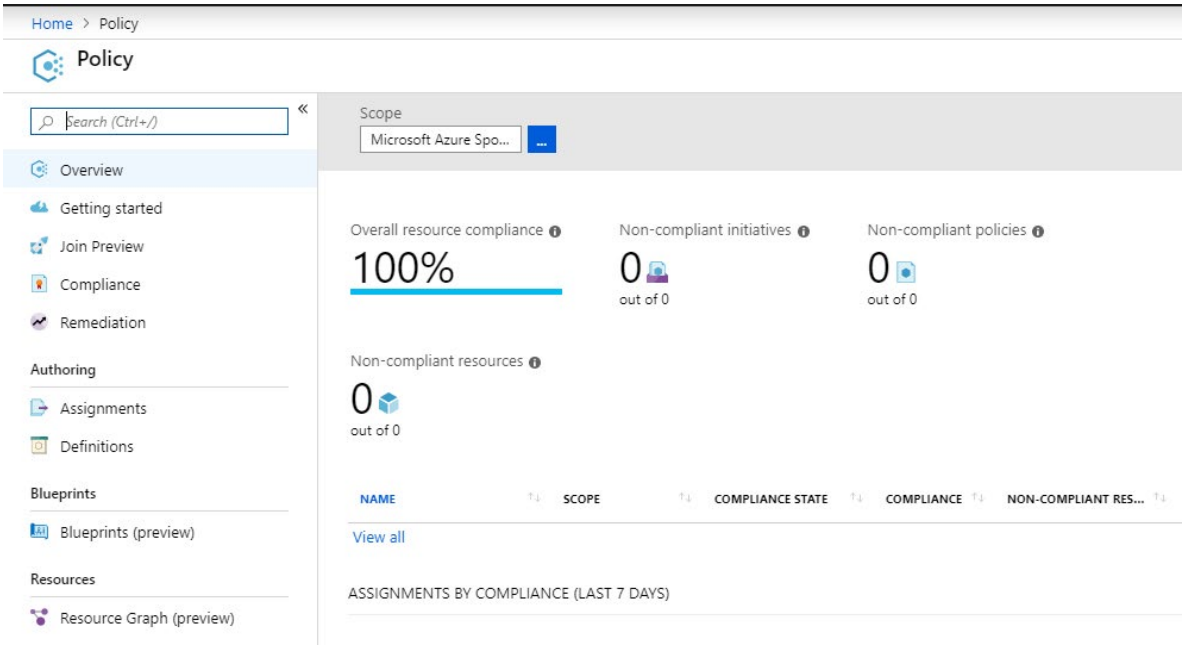
Azure Policy and Governance

If you don't have standards in IT, things can get out of control and there's no change in cloud. In fact, there's more of a need for governance than ever, if you don't have it in cloud it could cause allot of issues, excessive costs, issues supporting and a bad cloud experience to name a few. Azure Policy is essentially designed to help set those standards on Azure use (using policies), making sure it's used in the right way highlighting issues (using compliance) and help fix those issues (using remediation).

You can manually create policies (to check Azure resources are configured in a certain way) or you can use the growing number of pre-built policies. In this tutorial we'll look at a simple Azure Policy to define allowed Azure regions for resource provisioning.

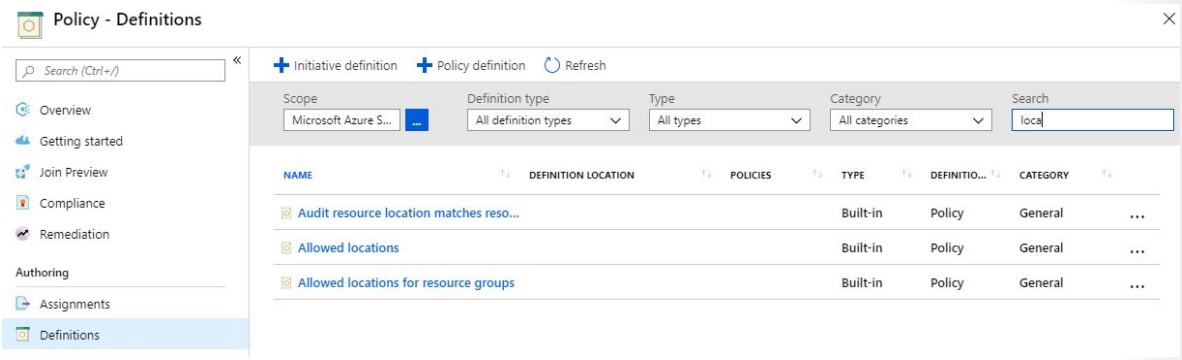
How to do it...

1. Launch the Azure Portal and create a new resource group called `az-rg-labs-azurepolicy-001`
2. From within Azure Portal, open Policy, from the left pane click on Definitions.

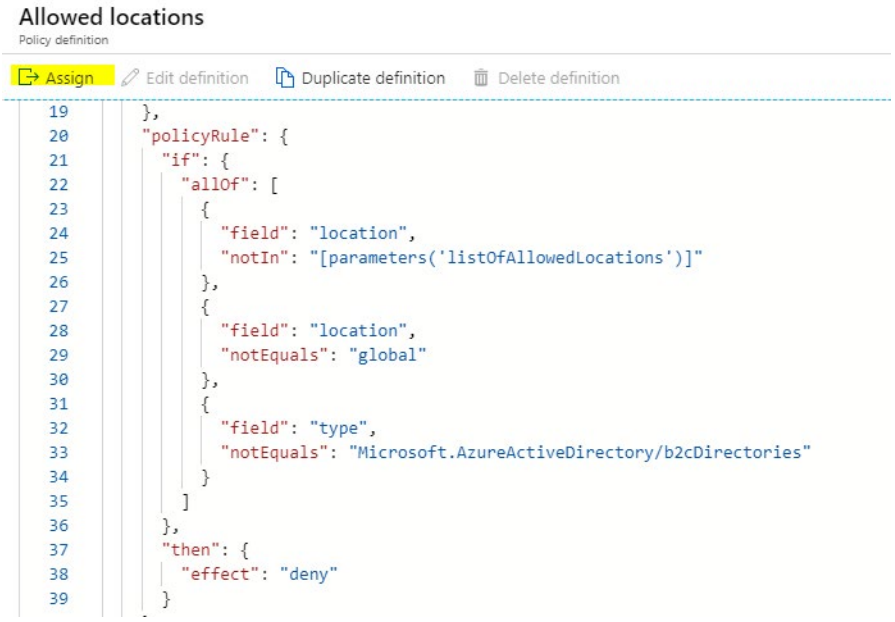


Definitions are effectively the rules you want to impose. You can use the built in policies, duplicate and edit them, or create your own from various templates like those on **GitHub**[14]

14 https://github.com/Azure/azure-policy



3. You can see the definition is a JSON file that needs a list of allowed locations and will cause a deny. You could duplicate this definition and add more checks if needed but we'll just assign it. Click Assign.



4. When assigning the policy, a scope needs to be selected… There are 3 options, as listed below, choose Resource Group.

- Management Group
- Subscription
- Resource Group

5. When assigning the policy, in the basics section change the assignment name and add a description.
6. To test this policy, in the resource group `az-rg-labs-azurepolicy-001` try to create a resource with location other than the allowed location.

There's more

Policies in Azure are a great way to scale enterprise policies for provisioning infrastructure across your estate. You can now learn more about policies **here**[15]. Policies are part of Azure governance which now also supports **blue prints**[16] and **Resource Graphs**[17].

15 https://docs.microsoft.com/en-us/azure/governance/policy/overview
16 https://azure.microsoft.com/en-gb/services/blueprints/
17 https://azure.microsoft.com/en-gb/features/resource-graph/

Module 4 Review Questions

Module 4 Review Questions

OWASP ZAP

What is OWASP ZAP and how can it be used?

Suggested Answer
OWASP ZAP can be used for penetration testing. Testing can be active or passive. Conduct a quick baseline scan to identify vulnerabilities. Conduct nightly more intensive scans.

Threat Modeling

What are the five stages of threat modeling?

Suggested Answer
Define security requirements. Create an application diagram. Identify threats. Mitigate threats. Validate that threats have been mitigated.

WhiteSource Bolt

Why would you use WhiteSource Bolt?

Suggested Answer
Use WhiteSource Bolt to automatically detect alerts on vulnerable open source components, outdated libraries, and license compliance issues in your code.

Azure Key Vault

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

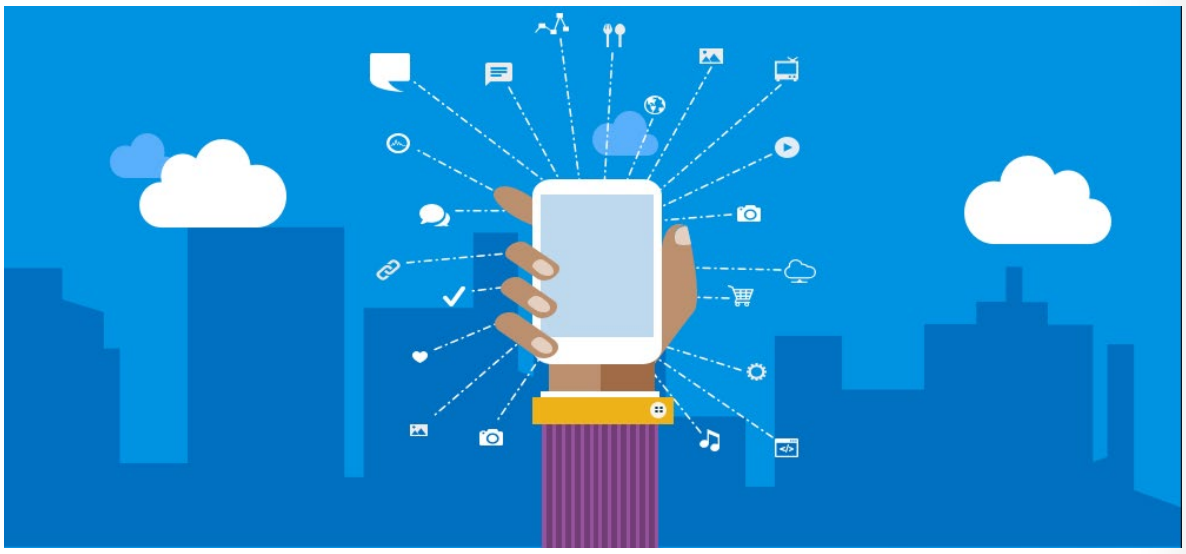
Suggested Answer
Azure Key Vault is a cloud key management service which allows you to create, import, store & maintain keys and secrets used by your cloud applications. The applications have no direct access to the keys, which helps improving the security & control over the stored keys & secrets. Use the Key Vault to centralize application and configuration secrets, securely store secrets and keys, and monitor access and use.

Module 5 Implement a Mobile DevOps Strategy

Introduction to Mobile DevOps

Mobile DevOps? DevOps? What's the Difference?

With all things DevOps, the answer is both simple and complex. The practices and the culture should be the same between DevOps and Mobile DevOps (simple), but the tooling required to successfully integrate a Mobile DevOps practice is going to be different than tooling for your DevOps practice (complex). Before we talk too much about the tooling for Mobile DevOps, let's take a look at the core practices that make up a complete DevOps solution, and understand how those apply to Mobile DevOps.



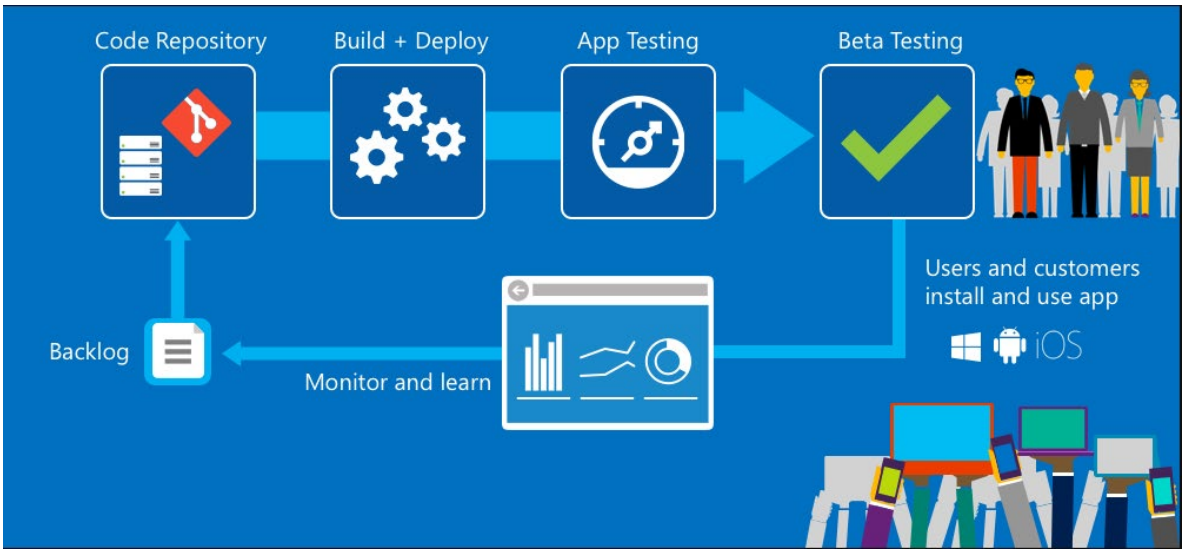
A DevOps practice should start with a production first mindset, meaning that everything that is done is to enable the app to be successful once it is live for end users. You should not wait until your app is being

deployed to start thinking about how to guarantee it is successful in production, that should be built into the app with the first few lines of code. For web apps, this might be integrating logging and application performance management SDK's. For mobile apps you should you be integrating SDK's that allow you track and respond to app crashes as their user impact dictates.

The DevOps team should always have clear goals, and the best way to do that is through a prioritised backlog. But how do you prioritise the backlog? With data driven decisions. To get that data, you need to instrument your app so you can understand how users are adopting and using your app. Ideally, you will find a way to enable conversations with your apps users to really understand the changes they want to see and what a great app looks like in their mind.

Now that you know where you should invest in your app, you need to quickly move those new features and enhancements out to your users. You can never sacrifice speed for quality or you simply trade problems, you never bring more value. Ensuring that you can move through all phases of testing to quickly ship a stable app is critical.

The final focal points of a DevOps practice should be around making sure your teams are cross functional, flexible, and enabled to respond based on their learning's from the above practices. They should focus on minimizing technical debt at every opportunity to maintain their flexibility and responsiveness.



All of this is the same between a DevOps and a Mobile DevOps practice, and it should lead you to the same goals:

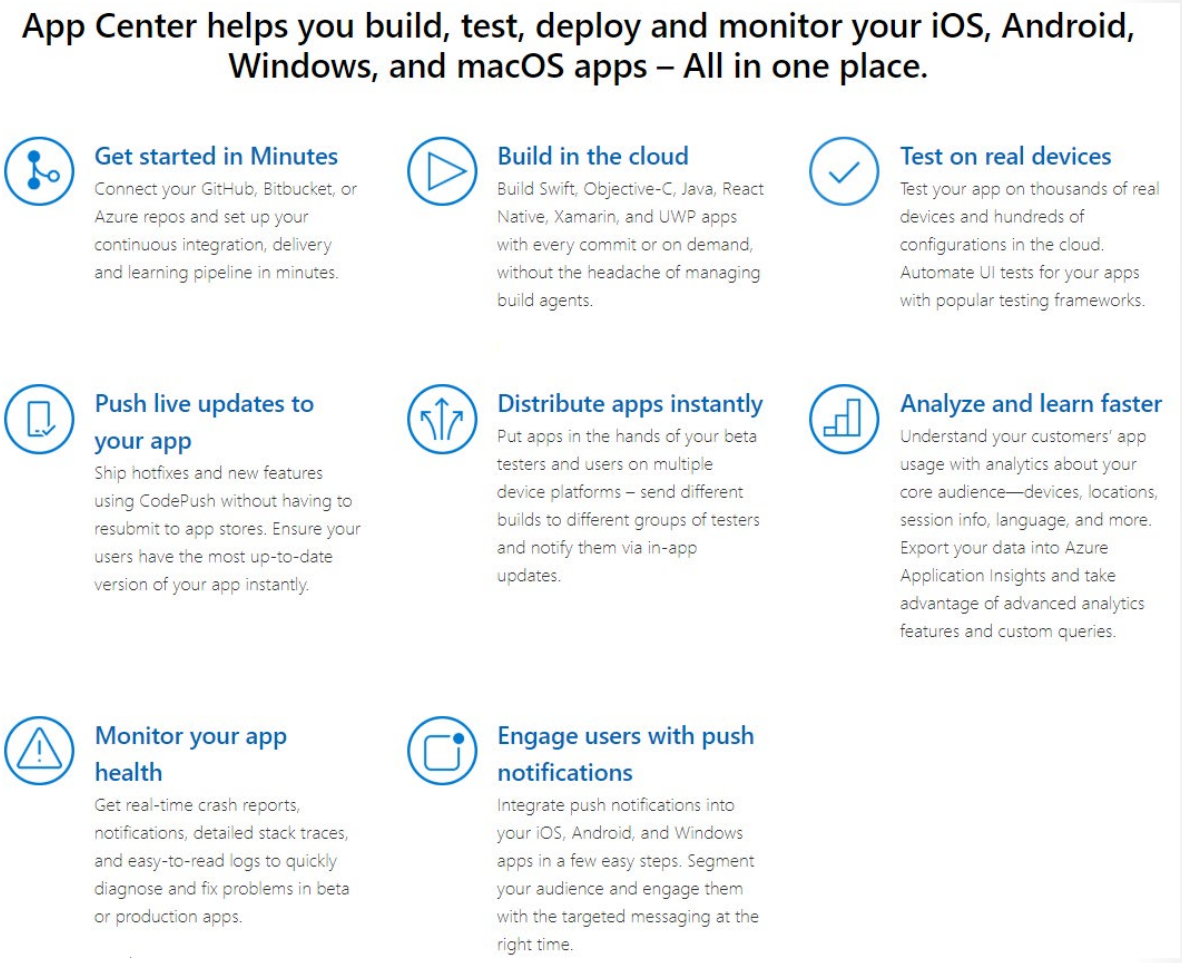
- Minimize work in progress
- Ensure highest quality user experience for new apps and updates
- Scale your delivery with automated process

To achieve these goals and maintain a strong Mobile DevOps practices requires a committed team, a cultural change and the right toolset. Microsoft offers this toolset through Visual Studio Mobile Center. Also called app center in short, it brings together multiple services commonly used by mobile developers into an integrated cloud solution.

Introduction to Visual Studio App Center

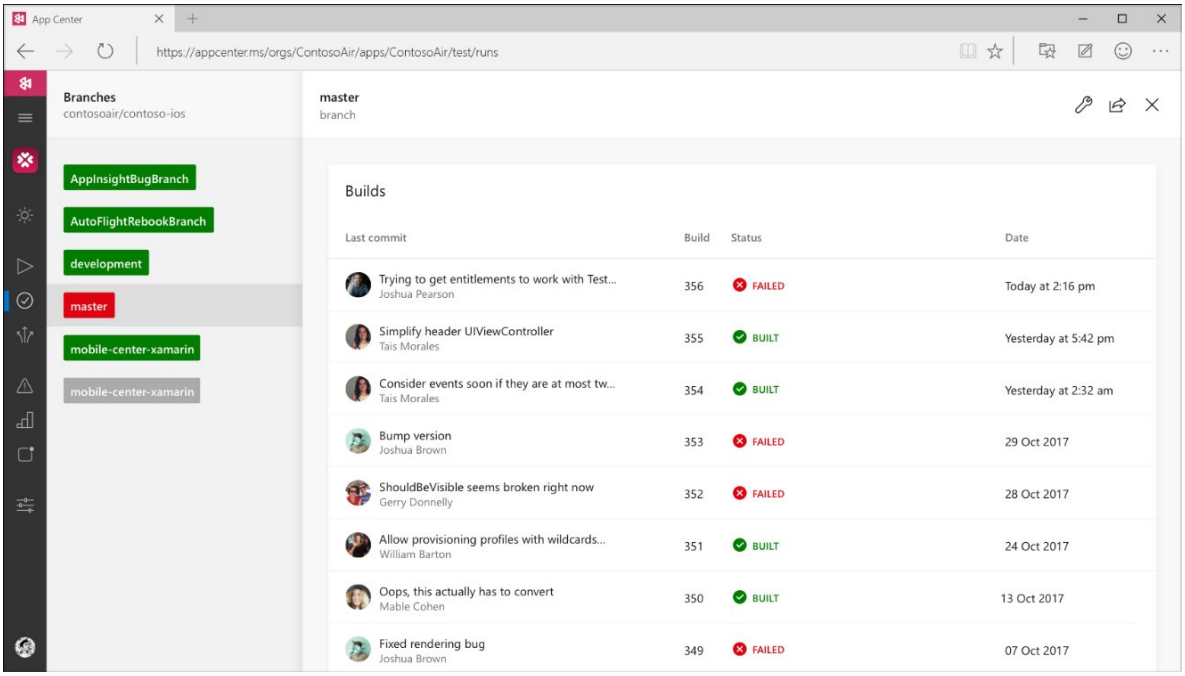
Introduction to Visual Studio App Center

Visual Studio App Center is mission control for apps. App Center brings together multiple services commonly used by mobile developers into an integrated cloud solution. Developers use App Center to Build, Test, and Distribute applications. Once the app's deployed, developers monitor the status and usage of the app using the Analytics and Diagnostics services, and engage with users using the Push service.



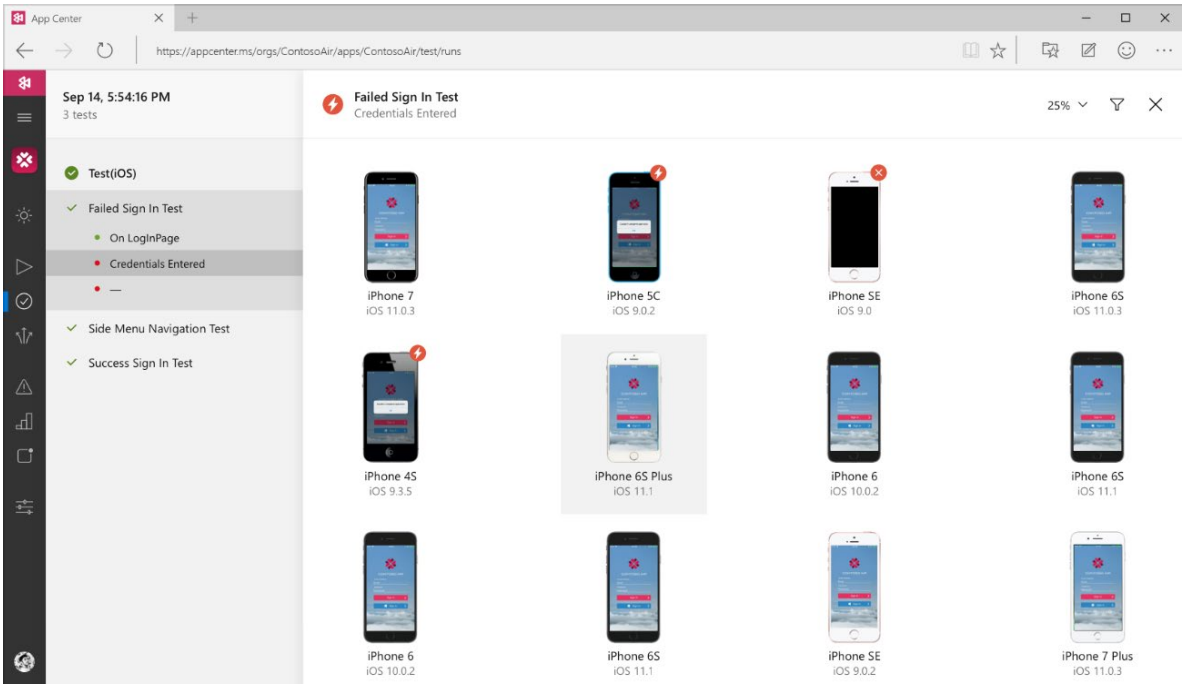
Continuous Integration in Minutes

Build apps more frequently, faster. Take the pain out of building your iOS, Android, Windows, and macOS apps locally. Connect to your GitHub, Bitbucket or Azure repos and build your apps automatically with every pull request or on demand, without the headache of managing build agents. Create an installable app package automatically with every push to your repository. Supports GitHub, or Git repos on Bitbucket and Azure DevOps. No additional build hardware required.



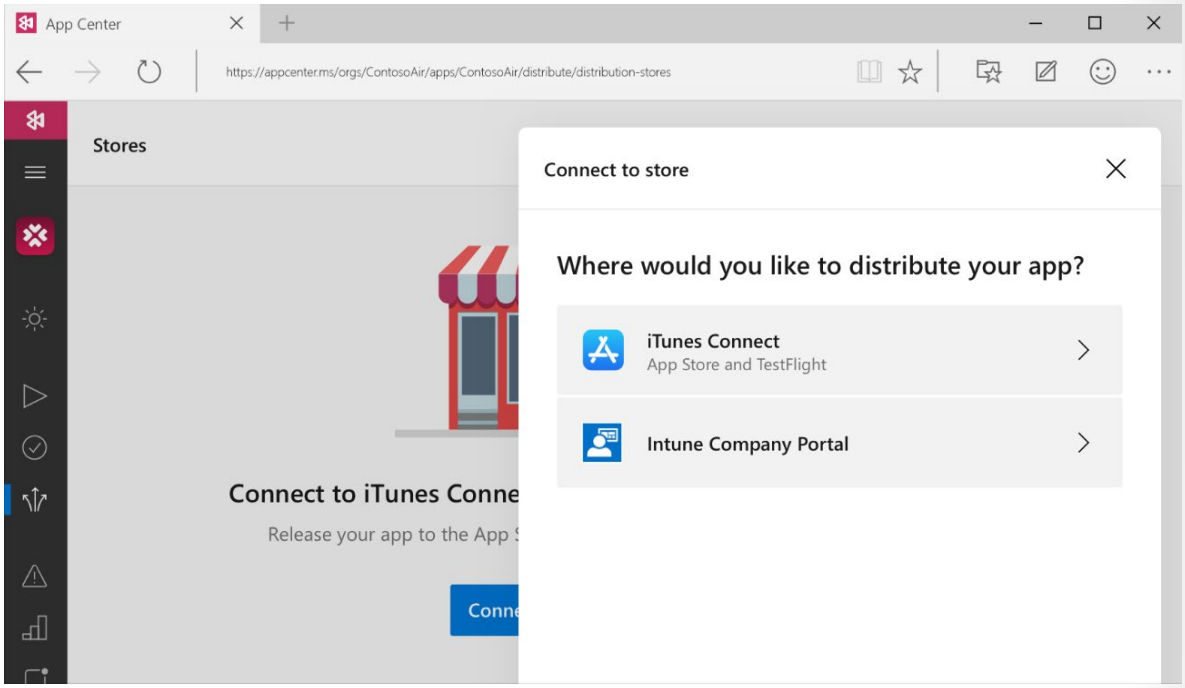
Continuous Quality on Real Devices

Ship higher-quality apps with confidence. Automate UI tests on thousands of real-world devices and hundreds of configurations in the cloud, using popular frameworks such as Appium, Espresso, and XCUITest. Test every UI interaction your users can do, and diagnose bugs and performance problems every time you build, with detailed step-by-step tracking reports, screenshots, and logs. Run your tests on more than 400 unique device configurations. Tests can be written for iOS and Android apps with Xamarin. UITest, Appium, Espresso (Android), and XCUITest (iOS). The next generation of Xamarin Test Cloud.



Continuous Delivery that Works

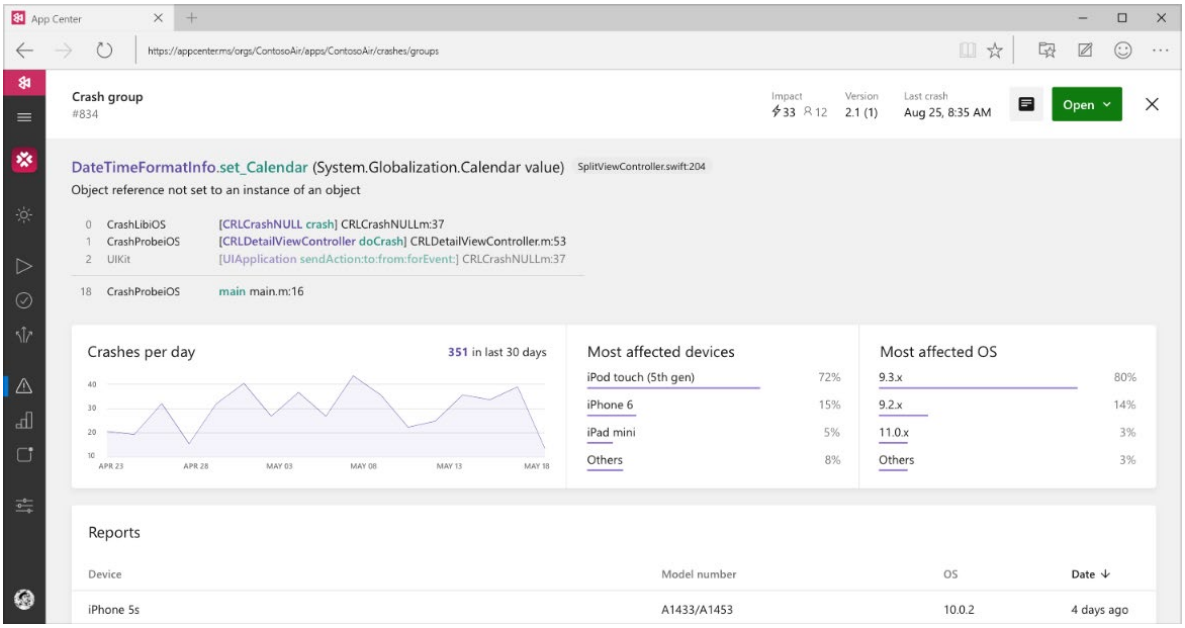
Deploy everywhere with ease… Distribute your app to beta testers and users on Android, iOS, Windows, and macOS with every commit or on demand. Send different builds to different groups of testers and notify them via in-app updates. When ready, release to Apple's App Store, Google Play, and to Microsoft Intune. Users can install the app via email distribution lists for testing, much as they'd download an app from the app store.



Continuous Learning for Growth

Insightful crash reports

Monitor the health of your app with advanced capabilities such as intelligent crash grouping and management, faster debugging with symbolication, and detailed crash reports. Get notified and fix issues as they come up.

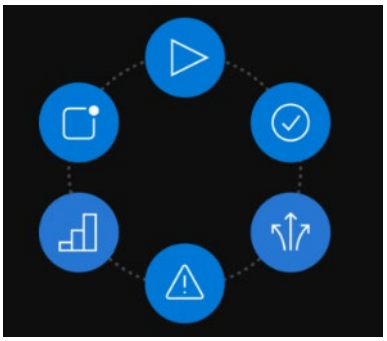


Real-time analytics

Grow your audience by focusing on what's important with deep insights about user sessions, top devices, OS versions, behavioral analytics and event trackers for your iOS, Android, Windows, and macOS apps. Easily create custom events to track anything.

Flexibility & choice

Choose only the App Center services you need
Have your own analytics solution? Using a different beta distribution platform? No problem. Our open-sourced SDKs and APIs let you integrate just the services of App Center you need.



Manage Mobile Target Device Sets and Distribution Groups

Continous Delivery Benefits and Challenges

Think about one of your mobile apps… How often do you publish updates to it? A couple of times per year? Quarterly? Monthly, bi-weekly? Every day? Multiple times per day? No! That's absurd, right?

Or is it? Some world-class app engineering teams are able to consistently release their apps weekly or every two weeks at high quality. How do they manage to do that? Part of the explanation is commitment to a continuous delivery process in some form. Let's look at some of the benefits and challenges of continuous delivery, and, more importantly, why this is accessible to everyone. You don't need to be a venture-backed startup or a huge company to do this stuff!

Benefits of Continuous Delivery for Mobile Apps

A continuous delivery process entails being able to release quickly, with high quality, at any given time. Let's contrast this with what we've seen a couple of times. The business wants a critical hotfix pushed out ASAP?

- NO! Fred is on vacation and he's the only guy on the team who can build, sign for distribution, and upload to iTunes.
- NO! We've not gone through the full QA cycle for the current master branch, and it will take three days to go through the manual regression tests, and, by the way, who's got the iPhone 4S running iOS 9.2?
- NO! We're not entirely sure how long the release process takes; reviews could take anything from 5-12 days.

Sound familiar?

With a continuous delivery process, things become easier. The business wants a critical hotfix pushed out ASAP? Not a problem! A developer makes a branch/pull-request with the change and gets peer review. An automated process builds and regression-tests the app using a CI server, test lab, and battery of automated regression tests. The developer merges the change to the master branch and pushes the release button! Continuous Delivery reduces your lead time, getting changes from idea to end-users faster. It reduces risk and variance in the release process because of automation and high repetition.

"Sure," you might say, "but we don't have the resources to maintain such an infrastructure." With Visual Studio App Center, this is accessible to everyone via the cloud at low cost.

Challenges for Continuous Delivery

App Store review times, a major barrier to continuous delivery in the past, have been declining steadily since 2015, and are **now about one day on average**[1]. Still, there remain several challenges with implementing a continuous delivery process. For example, to build iOS apps, you need a Mac-based CI server, which is uncommon in most companies, and configuring this for CI can be **time-consuming**[2]. With App Center, building in the cloud (even for iOS) takes less than **five minutes to set up**[3].

But what about quality? Thorough testing of apps is time-consuming and difficult. There are literally thousands of device models out there, with different and often limited hardware capabilities, screen sizes

1 http://appreviewtimes.com/
2 http://www.andrewbirch.com/tech/blog/2015/8/15/setting-up-a-continuous-build-environment-for-xamarin-part-1-jenkins
3 https://channel9.msdn.com/events/Build/2017/B8072

and resolutions. There are numerous platforms, including Android, iOS, macOS, and Windows, and your app likely supports multiple versions of each.

One way to address the challenge is to rely on outsourced manual testing with providers that offer specialist testers with access to large libraries of different mobile devices. This has benefits, but also drawbacks:

- You need to clearly communicate your requirements.
- You need to interpret the test results.
- The tests are performed by humans, who make mistakes.
- The turnaround for test passes can be slow, ranging from days to weeks.
- Integrating this testing process into an automated Continuous Integration/Continuous Delivery pipeline can be hard.

Instead, what if you had access to a large set of cloud-hosted devices on which you could run automated regression tests of your app with a single line script in your CI system? What if the tests produced visual test reports, so that not only could you perform automated tests, but also be able to review layout and compare differences across device models and OS versions? That's what Visual Studio App Center Test offers.

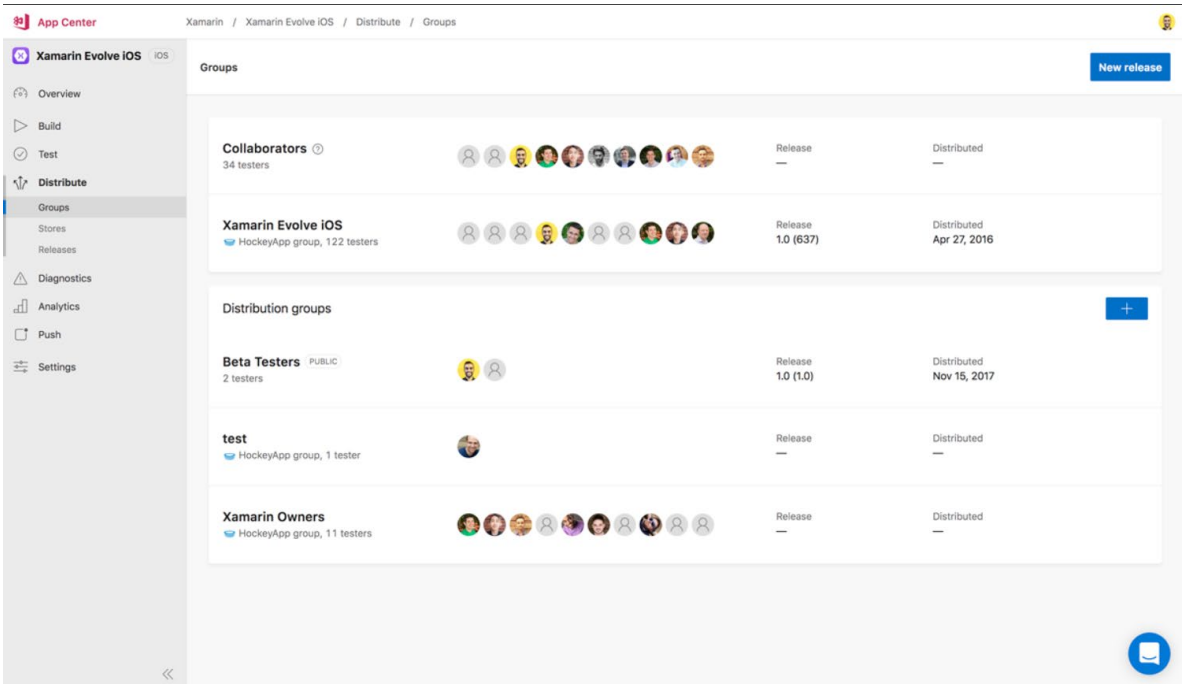
Distribution Groups

Getting Started: How to Extend Continuous Builds to Continuous Tests

Let's just quickly understand the terminology and how to enable it...

Distribution Groups

A distribution group could be based on types of users such as beta testers or internal stakeholders, or based on different stages of the development cycle such as nightly build or staging. Invite testers to your groups with just an email address. When distributing a new release in App Center, you'll select one of your distribution groups as the destination.



Manage Distribution Groups

Distribution Groups are used to control access to releases. A Distribution Group represents a set of users that can be managed jointly and can have common access to releases. Example of Distribution Groups can be teams of users, like the QA Team or External Beta Testers or can represent stages or rings of releases, such as Staging.

Creating a Distribution Group

To create a new Distribution Group log into the App Center portal, select your desired app, click Distribute in the navigation pane, and lastly select the New Group group button from the top of the screen. Provide a name for the Distribution Group. You can then add users to this group by email. You can also add additional users after the group has been created.

Managing Users in a Distribution Group

Clicking on a Distribution Group will allow for management of the group. You can use the invitation box to add additional users. Or select users from the table to remove them from the group. From this page you can also see the full release history for this Distribution Group by clicking on the releases tab.

It's incredibly easy to add automated UI tests into your App Center builds. App Center has thousands of physical Apple and Android devices on which to run your newly-built app, and App Center supports the most popular UI testing frameworks: Espresso, XCUITest, Appium, and Xamarin.UITest.

Adding Azure Active Directory (AAD) groups to a Distribution Group

In addition to adding testers via email, we now support the addition of AAD groups to a distribution group. Any member of an AAD group can link their organization's subscription to their AAD tenant in App Center. Doing so will enable members of an organization to start adding managed groups to their app's distribution groups. To get started, link your AAD tenant to your organization using the following steps:

- Login to the App Center portal, then select the organization to which you like to link to your AAD tenant
- On the navigator pane that opens, click Manage
- On the azure active directory panel on the Manage page, click the connect button
- You will be redirected to a Microsoft login page, login with your Microsoft/AAD credentials
- You will be redirected to the App Center portal, click the desired tenant you would like to attach.
- At the bottom right of the presented window, click the connect

Once your tenant is connected to the organization, you can add an AAD group as you would an individual tester.

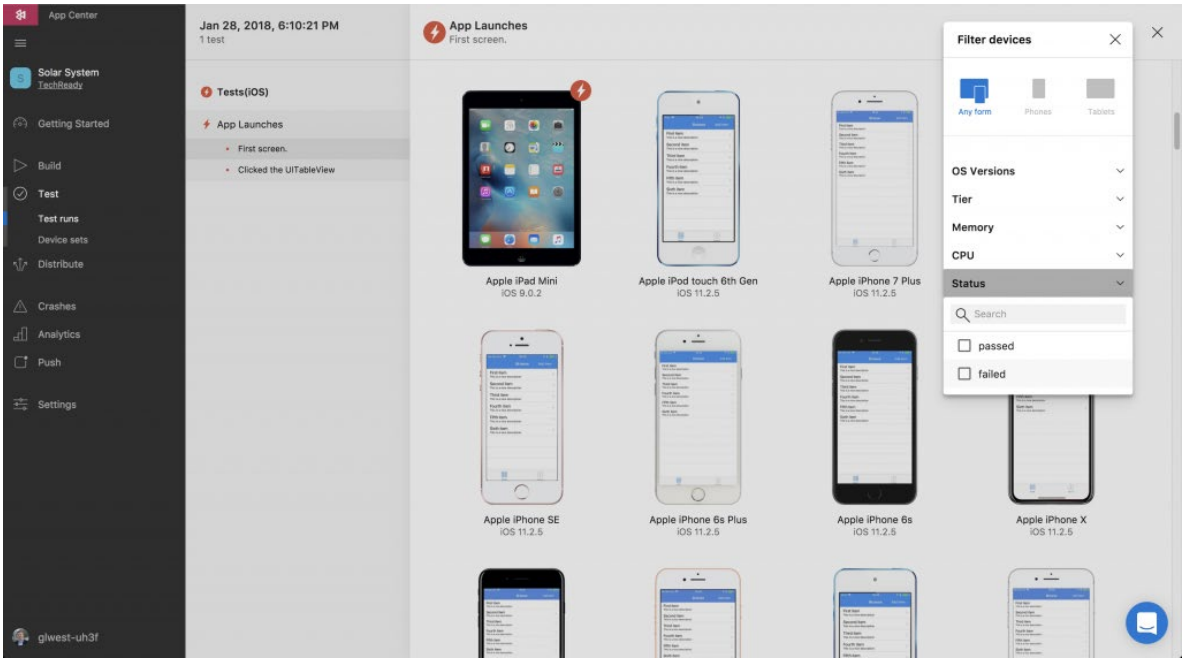
Run a UI Test

To run a UI test:

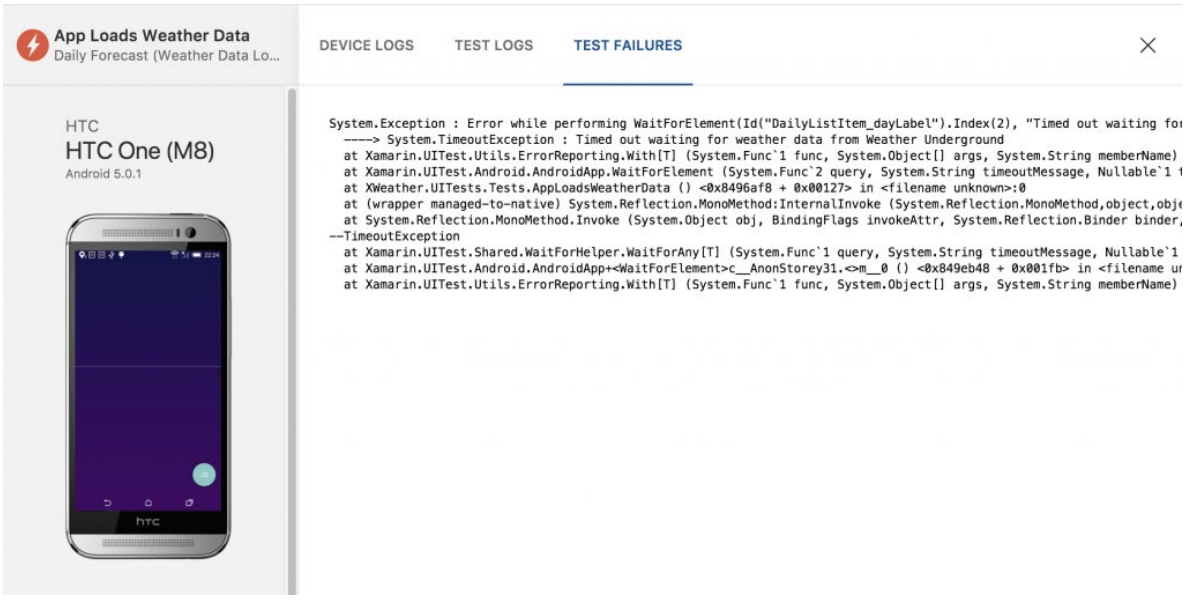
- Using the App Center Test CLI, upload your test suite automatically after completing a build.
- Once you generate a test upload shell command, add it to your app's repo at the solution or project level.

When your test code finishes executing, a detailed report will appear in the App Center Test and the CLI will return a URL linking directly to it. Please see this example command as a reference. Instrumenting your continuous builds with continuous testing is a sure way to shorten the feedback loop, greatly improving app quality.

When the test report is complete, you can easily identify the failed devices by the red "lightning" indicator. Additionally, you can filter just the failed devices or by other parameters by selecting the icon in the upper right corner of the test report.



For every test device, App Center provides device logs, test framework logs, and the line in which the failure occurred to help you easily identify the regression. Start by examining the test failures for clues about where in the test execution the failed assertion occurred. If this is not revealing, dive into the test and device logs for more thorough investigation to determine the root cause of the failure.



App Distribution Center

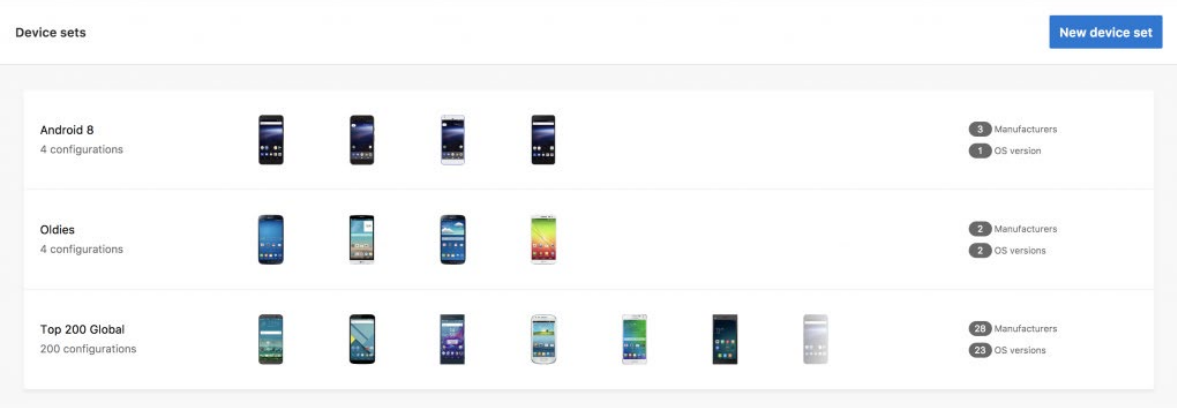
App Center Distribution is a tool for developers to release application binaries to their end user devices. The speed and integration capabilities make this tool perform admirably for release of pre-production or

beta software to testing devices. App Center Distribution supports the distribution of application packages for iOS, Android, UWP, and macOS.

Here are the top three features to try in App Center Test:

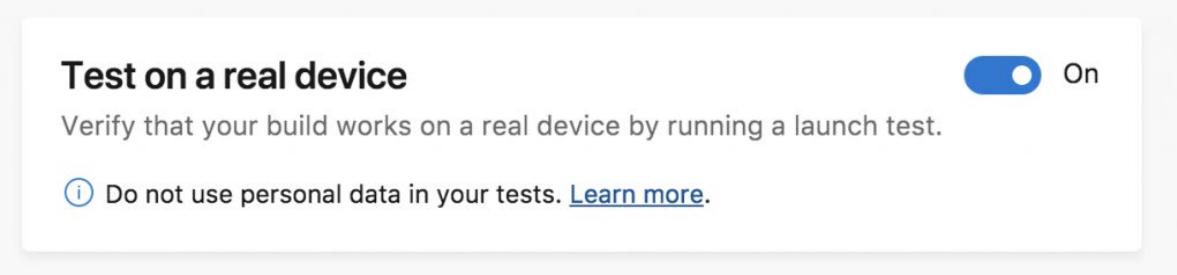
1. Named Device Sets

It's best to run your app on as many devices as possible before shipping to your users. App Center offers named device sets to help you organize and reuse collections of mobile hardware in our labs. Try this out today by adding a "Top 20" set that you can update as you see trends in your users' devices changing.

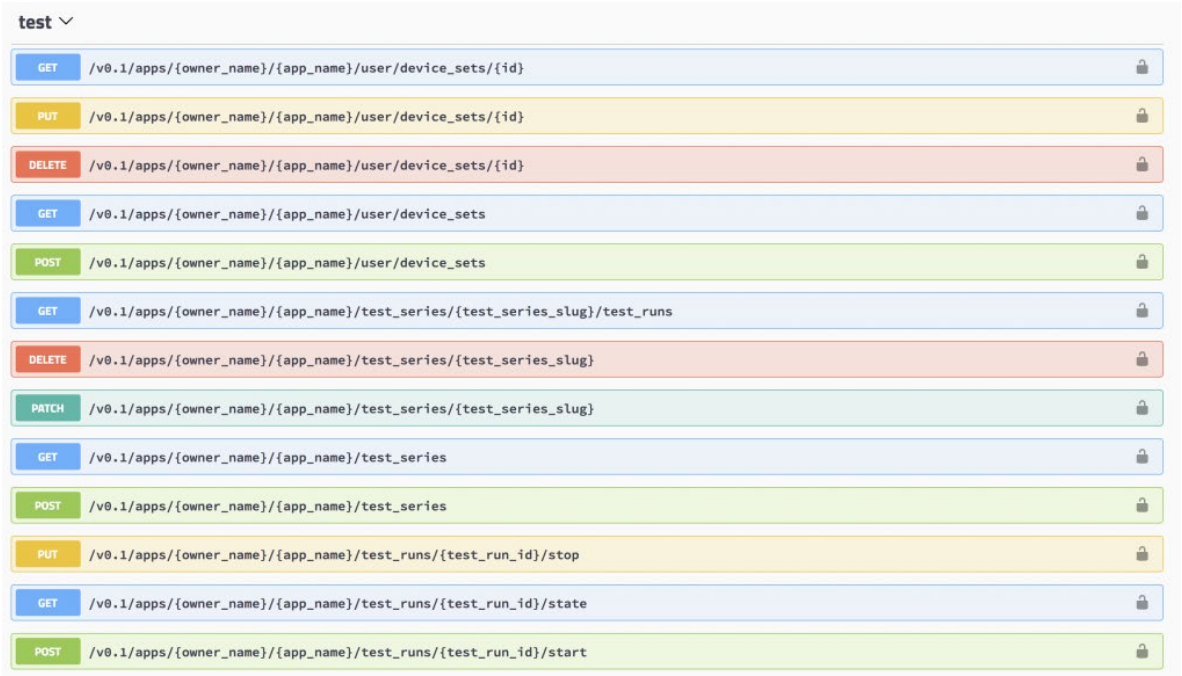


2. Launch Test

To ensure your latest app build runs and opens on our devices before you have any test code written, you can enable launch test in your build configuration. Validate your app's ability to start on our wide variety of hardware today!



3. Improved API App Center Test provides a robust REST API giving full access to all data generated during your test run. The data in your test report is accessible via our API, including screenshots, device logs, test runner logs, and failed device specifics. To get started with App Center API today, please see the following documentation.



Manage Target UI Test Device Sets

Manage Target UI Test Device Sets

You can use App Center Build's Launch Test feature to run your latest successful build on a real device using App Center Test. Launching your app after each build validates that your latest build is working.

Configuring your branch

First, the branch needs to be setup before it's ready to run the launch test. You can read more about configuring your branch in the Configure a build respectively for **Android**[4] and **iOS**[5].

Secondly, you need a subscription to App Center Test before you can enable launch test. If you're a first-time user of App Center Test, there's a free trial so you can see how it works.

When you have an active subscription it's time to enable testing, all you'll need to do is to toggle Run a launch test on a device option to on and click Finish setup button. During the new build, there will be two parts, a build and a test part, so it's normal to have prolonged build time. The benefit is that you know whether your app starts on a physical device.

Finding your launch test result can be done in two ways:

- When the test is completed the app collaborators will receive an e-mail with test results
- Go to Test in the left-hand menu and follow along in the progress

Unsupported configurations

When a build is running, the build configuration is composed of several parts: the build definitions you made in the code or in the IDE, and the settings you have chosen in App Center. The way configurations work is platform and language specific. Below are some known configurations where launch test is not supported.

- Some Android configurations can create several APK files. Which makes it unclear to the build service which binary to test. If you disable Generate one package(.apk) per selected ABI in your Android Build options, only one APK is created.
- Simulator builds for Xamarin apps don't output a binary executable file, and therefore can't be tested. This is the case for both Android and iOS.
- Android apps without Internet permissions cannot be tested using App Center Test. Internet permissions are needed for the test infrastructure to communicate with your app. In order to use launch test for your Android project, make sure to declare internet permissions in your app manifest
- iOS unsigned builds don't generate an .ipa file and they can't be tested on a real device.

4 https://docs.microsoft.com/en-us/appcenter/build/android/first-build
5 https://docs.microsoft.com/en-us/appcenter/build/ios/first-build

Provision Tester Devices for Deployment

Provision Tester Devices for Deployment

Getting your app onto your own device, or in the hands of potential testers, can be one of the most time-consuming challenges when developing an iOS app. To install an app on a single device, you must register the device with Apple and sign the app with the relevant provisioning profile. While this isn't much of an issue for a single device, you need to register and re-sign the app each time you install the app on a new set of devices. For a developer, this repetitive action is a huge loss of time; for a tester, it's a disruptive experience to receive an app and not be able to install it because your device hasn't been provisioned, yet.

App Center supports the release of auto-provisioning capabilities, this enables iOS developers to spend more time creating and shipping great apps, rather than managing device provisioning. With the auto-provisioning capability, you no longer need to know the device IDs of testers, coworkers, or stakeholders when building a new beta version of your app for testing. App Center integrates this capability directly into the install portal, so you can automate the distribution process and enable testers and team members to install your app with one click.

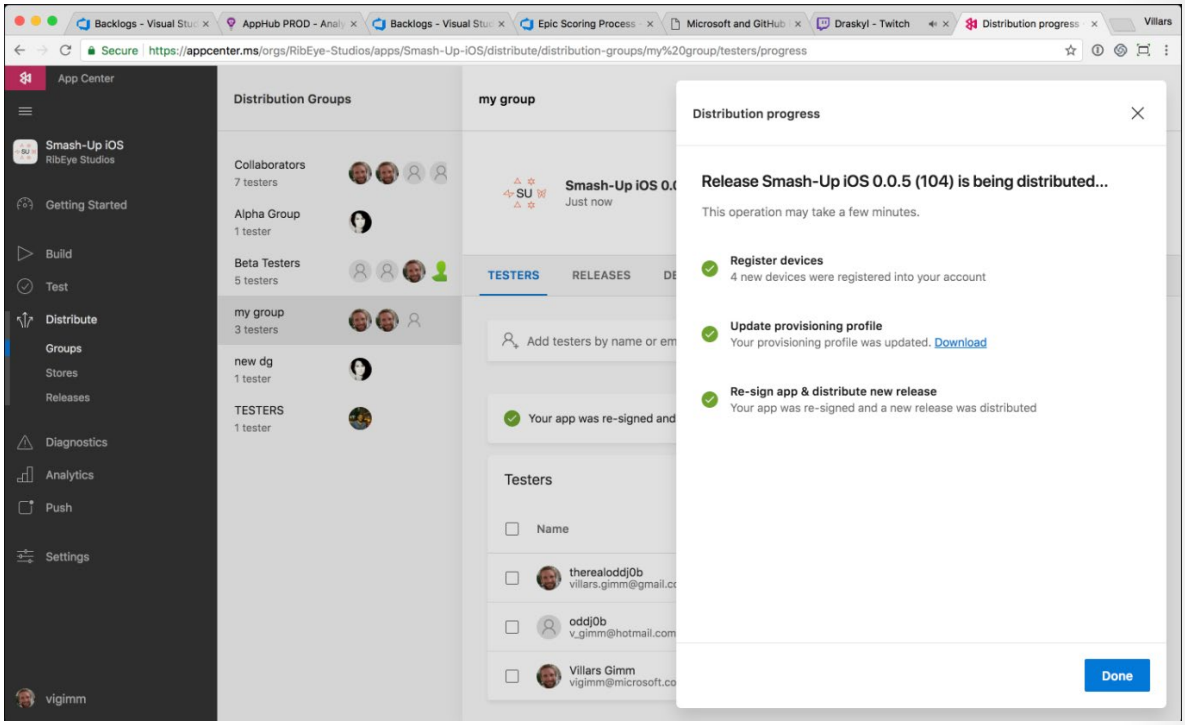
Setting Up Auto-Provisioning

All you need to get started with auto-provisioning is an ad-hoc provisioning profile and a production certificate, both of which are created in the Apple Developer portal, to produce a signed app ready for our distribution service.

Once you have an .ipa, select your app in App Center and navigate to the distribution service in the App Center portal to choose the distribution group you want to set up. Click on your distribution group settings and turn on Automatically manage devices. You'll need your Apple ID credentials and your distribution certificate to complete this step.

Now that you've turned on the Automatically manage devices setting, next time you release a new version of your app using the App Center's Build service or by manually uploading the app binary, App Center will automate the process for you and run the following steps without any action from you or your testers:

- Log into your developer portal.
- Register new device IDs.
- Add them to the app's provisioning profiles.
- Download all updated profiles.
- Re-sign the binary using the selected certificate and the updated profiles.
- Enable all testers in the distribution group to install your app.



Every time you release a new version of your build, testers will get an email notification to install the app and start using it.

Create Public and Private Distribution Groups

Types of Distribution Groups

Private Distribution Groups

In App Center, distribution groups are private by default. This means only testers invited via email can access the releases available to this group. Testers added to this group will receive a notification that they have been added to the app to test. After a release, testers that were previously added to this group will receive the new release notification email and will be required to login to their App Center account in order to access and download releases.

Public Distribution Groups

Distribution groups must be public to enable unauthenticated installs from public links. When creating a new distribution group, the options is available during the setup process. After giving your group a name, you can enable Allow public access.

To make an existing distribution group public, open the distribution group and click on the settings icon in the upper right-hand corner of the screen. From the settings modal, you can enable Allow public access.

As with private distribution groups, testers will receive an email notifying them that they've been invited to test the app and when a release is available to them. In order to access the app from here, testers will be required to login with their App Center account.

In addition to this, a public download link is displayed underneath the distribution group name at the top of the distribution group page. Anyone, including testers who aren't explicitly added to the distribution group, can access the release without signing in using the public download link.

Shared Distribution Group

Shared distribution groups are private or public distribution groups that are shared across multiple apps in a single organization. Shared distribution groups eliminate the need to replicate distribution groups across multiple apps. With a few clicks, you can give a shared distribution group access to any combination of apps belonging to a particular organization.

Unlike traditional public and private distribution groups, shared distribution groups are at the organization level rather than the app level. Due to this, the steps to create one are slightly different:

1. Login to the App Center portal, then select the organization to which you would like to add a shared group
2. On the navigator pane that opens, click People
3. On the People page, click the distribution group link to open the shared distribution groups page
4. On the top right corner of the shared distribution groups page, select the blue create new group button.
5. Once you have a distribution group, click the group entry in the table to add testers and apps to the group.

To add testers to your new shared distribution group, select the Testers tab and enter the emails of the desired testers.

To add apps to your new shared distribution group, select the Apps tab and enter the name of the desired apps that this group should have access to.

Other Considerations

Automatically manage devices

When releasing an iOS app signed with an ad-hoc or development provisioning profile, you must obtain tester's device IDs (UDIDs), and add them to the provisioning profile before compiling a release. When you enable the distribution group's Automatically manage devices setting, App Center automates the before mentioned operations and removes the constraint for you to perform any manual tasks. As part of automating the workflow, you must provide the user name and password for your Apple ID and your production certificate in a .p12 format.

App Center starts the automated tasks when you distribute a new release or one of your testers registers a new device. First, all devices from the target distribution group will be registered, using your Apple ID, in your developer portal and all provisioning profiles used in the app will be generated with both new and existing device ID. Afterward, the newly generated provisioning profiles are downloaded to App Center servers. Second, App Center uses your saved certificate and the downloaded provisioning profiles to re-sign your app so it can be installed on any device in the releases distribution group. Your testers will not have to wait for you to create a new release as the existing release is being updated and a download link will be available through the App Center Install portal.

Releasing a Build to a Distribution Group

To release a new build to a Distribution Group. Make use of the Distribute new Release button at the top of the screen and choose the Distribution Group from the list during the steps.

User Download Analytics

User download analytics allow you to monitor how testers within a distribution group are interacting with a release. Total and unique download counts for a release are available both at the top of each distribution group page as well as on the Release Details page. The total download count represents the total number of times the Install button has been hit by a tester. The unique download count represents the number of users that have installed the release. As an example, if one tester has downloaded a release to two different test devices, this would equal two total downloads and one unique download. Note that public distribution groups will only include the total download count and not the unique download count.

Module 5 Review Questions

Module 5 Review Questions

App Center

What is the Visual Studio App Center and why would you use it?

Suggested Answer
App Center brings together multiple services into an integrated cloud solution. Developers use App Center to Build, Test, and Distribute applications. Once the app's deployed, developers can monitor the status and usage of the app.

App Center

What are the main capabilities of the App Center?

Suggested Answer
Get started in minutes, push live updates to your app, monitor app health, build in the cloud, distribute apps instantly, engage users with push notification, test on real devices, and analyze and learn faster.

Distribution Groups

What is a Distribution Group and how is it used?

Suggested Answer
Distribution Groups are used to control access to releases. Distribution Group represents a set of users that can be managed jointly and can have common access to releases. You create the groups and can now add AAD groups to the distribution group.

Distribution Groups

What are the three types of Distribution Groups and how are they different?

Suggested Answer
Private Distribution Groups - This is the default. Only testers invited via email can access the releases available to this group. Public Distribution Groups - Used when you want enable unauthenticated installs from public links. Testers will receive an email notifying them that they've been invited to test the app and when a release is available to them. Shared Distribution Groups - Private or public distribution groups that are shared across multiple apps in a single organization. Shared distribution groups eliminate the need to replicate distribution groups across multiple apps.

Module 6 Course Completion

Final Exam

Final Exam Questions

DevOps_IDP.101_MC

Select which of the below is a centralised version control system?

- ☐ Git
- ☐ Bazaar
- ☐ Tfvc
- ☐ Mercurial

DevOps_IDP.102_MC

Is this statement correct?

Azure Repos supports both TFVC and Git repository in a single team project.

- ☐ true
- ☐ false

DevOps_IDP.103_MC

Which file can you configure to ensure that certain file types are never committed to the local Git repository?

- ☐ ignore.git
- ☐ gitignore
- ☐ gitignore.txt
- ☐ git.ignore

DevOps_IDP.104_CB

Select all operations that you can perform as an anonymous user accessing a git repo in Azure repos in a public project in Azure DevOps. (Select 2.)

- ☐ browse the code
- ☐ download the code
- ☐ commit code
- ☐ create new pull requests

DevOps_IDP.105_CB

Which type of merge strategies can you enforce for your pull requests when configuring branch policies in Azure Repos. (Select 2.)

- ☐ No-fast-forward merge
- ☐ Squash merge
- ☐ Git Flow merge
- ☐ No-fast-upstream merge

DevOps_IDP.106_CB

What security roles are available for user access management in Agent Pools in Azure Pipelines? (Select three.)

- ☐ Reader
- ☐ User
- ☐ Contributor
- ☐ Administrator
- ☐ All of the above

DevOps_IDP.107_CB

Which of the following types of Agents are available in Azure Pipelines? (Select two.)

- ☐ Self hosted
- ☐ Microsoft hosted
- ☐ Datacenter hosted

DevOps_IDP.108_MC

Is this statement correct?

Azure pipelines are cross platform and can run on Windows, Mac and Linux machines.

- ☐ true
- ☐ false

DevOps_IDP.109_CB

Pull requests in Azure Repos support which of the following capabilities? (Select four.)

- ☐ Multiple code reviewers
- ☐ Checking for comment resolution
- ☐ Voting and Emojis
- ☐ Build validation policies

DevOps_IDP.110_CB

Azure Pipelines support integration with Azure Key Vault to securely store secrets, which of these Azure DevOps components allows this integration? (Select two.)

- ☐ Azure Key Vault Build and Release Task
- ☐ Variable Group
- ☐ Azure Repos
- ☐ Service Connections

DevOps_IDP.111_MC

Is this statement correct?

Azure Pipelines offers out of box integration with the following source control systems Azure Repos, GitHub, GitHub Enterprise, Subversion, Bitbucket Cloud and External Git.

- ☐ true
- ☐ false

DevOps_IDP.112_CB

What DevOps tooling does Visual Studio App Center provide for your mobile applications? (Select four.)

- ☐ Build your apps
- ☐ Crash reporting
- ☐ App Distribution
- ☐ Device Testing

DevOps_IDP.113_MC

Is this statement correct?

Visual Studio App Center only supports private distribution groups.

- ☐ true
- ☐ false

DevOps_IDP.114_MC

Open Source Projects hosted in GitHub using Azure Pipelines get which of the following benefits?

- ☐ 10 free parallel jobs and unlimited minutes for public repos
- ☐ 5 free parallel jobs and 1800 minutes for public repos
- ☐ unlimited free parallel jobs and unlimited minutes for public repos
- ☐ 1 free parallel job and 1800 minutes for public repos

DevOps_IDP.115_CB

Secure DevOps kit extension can be used in Azure Pipelines to _____. (Select two.)

- ☐ Deploy Virtual Machines in Azure
- ☐ Scan ARM Template for vulnerabilities
- ☐ Scan Resources in Resource Groups in Azure for vulnerabilities
- ☐ Securly login to Azure DevOps

DevOps_IDP.116_MC

After you install Git and prior to issuing the first commit, which two configuration properties does the tool expect to be configured?

- ☐ username and email address
- ☐ username and password
- ☐ email address and password
- ☐ username and IP address

DevOps_IDP.117_CB

Azure DevOps allows connecting to Azure for infrastructure provisioning by giving you the ability to create a service endpoint. The service endpoint creation dialogue in Azure DevOps allows you to create the service endpoint at the followings scope. (Select two.)

- ☐ subscription
- ☐ resource group
- ☐ resource
- ☐ web app

DevOps_IDP.118_CB

Which of the following githooks are supported by git locally? (Select three.)

- ☐ commit-msg
- ☐ pre-push
- ☐ pre-commit
- ☐ pre-delete

DevOps_IDP.119_MC

Git supports storing large files in version control by using text pointers inside it's history rather than storing the actual object. What is this technology in git called?

- ☐ git pointers
- ☐ git large file storage
- ☐ git blobs
- ☐ virtual git

DevOps_IDP.120_CB

Azure Pipelines integrate with Jenkins. What capabilties of Jenkins can directly be used from Azure Pipelines? (Select two.)

- ☐ Jenkins Queue Job
- ☐ Jenkins Download Artifacts
- ☐ Jenkins create server
- ☐ Jenkins create new job

Answers

DevOps_IDP.101_MC

Select which of the below is a centralised version control system?

- ☐ Git
- ☐ Bazaar
- ■ Tfvc
- ☐ Mercurial

DevOps_IDP.102_MC

Is this statement correct?

Azure Repos supports both TFVC and Git repository in a single team project.

- ■ true
- ☐ false

DevOps_IDP.103_MC

Which file can you configure to ensure that certain file types are never committed to the local Git repository?

- ☐ ignore.git
- ■ gitignore
- ☐ gitignore.txt
- ☐ git.ignore

DevOps_IDP.104_CB

Select all operations that you can perform as an anonymous user accessing a git repo in Azure repos in a public project in Azure DevOps. (Select 2.)

- ■ browse the code
- ■ download the code
- ☐ commit code
- ☐ create new pull requests

DevOps_IDP.105_CB

Which type of merge strategies can you enforce for your pull requests when configuring branch policies in Azure Repos. (Select 2.)

- ■ No-fast-forward merge
- ■ Squash merge
- ☐ Git Flow merge
- ☐ No-fast-upstream merge

DevOps_IDP.106_CB

What security roles are available for user access management in Agent Pools in Azure Pipelines? (Select three.)

- ■ Reader
- ■ User
- ☐ Contributor
- ■ Administrator
- ☐ All of the above

DevOps_IDP.107_CB

Which of the following types of Agents are available in Azure Pipelines? (Select two.)

- ■ Self hosted
- ■ Microsoft hosted
- ☐ Datacenter hosted

DevOps_IDP.108_MC

Is this statement correct?

Azure pipelines are cross platform and can run on Windows, Mac and Linux machines.

- ■ true
- ☐ false

DevOps_IDP.109_CB

Pull requests in Azure Repos support which of the following capabilities? (Select four.)

- ■ Multiple code reviewers
- ■ Checking for comment resolution
- ■ Voting and Emojis
- ■ Build validation policies

DevOps_IDP.110_CB

Azure Pipelines support integration with Azure Key Vault to securely store secrets, which of these Azure DevOps components allows this integration? (Select two.)

- ■ Azure Key Vault Build and Release Task
- ■ Variable Group
- ☐ Azure Repos
- ☐ Service Connections

DevOps_IDP.111_MC

Is this statement correct?

Azure Pipelines offers out of box integration with the following source control systems Azure Repos, GitHub, GitHub Enterprise, Subversion, Bitbucket Cloud and External Git.

- ■ true
- ☐ false

DevOps_IDP.112_CB

What DevOps tooling does Visual Studio App Center provide for your mobile applications? (Select four.)

- ■ Build your apps
- ■ Crash reporting
- ■ App Distribution
- ■ Device Testing

DevOps_IDP.113_MC

Is this statement correct?

Visual Studio App Center only supports private distribution groups.

- ☐ true
- ■ false

DevOps_IDP.114_MC

Open Source Projects hosted in GitHub using Azure Pipelines get which of the following benefits?

- ■ 10 free parallel jobs and unlimited minutes for public repos
- ☐ 5 free parallel jobs and 1800 minutes for public repos
- ☐ unlimited free parallel jobs and unlimited minutes for public repos
- ☐ 1 free parallel job and 1800 minutes for public repos

DevOps_IDP.115_CB

Secure DevOps kit extension can be used in Azure Pipelines to _____. (Select two.)

- ☐ Deploy Virtual Machines in Azure
- ■ Scan ARM Template for vulnerabilities
- ■ Scan Resources in Resource Groups in Azure for vulnerabilities
- ☐ Securly login to Azure DevOps

DevOps_IDP.116_MC

After you install Git and prior to issuing the first commit, which two configuration properties does the tool expect to be configured?

- ■ username and email address
- ☐ username and password
- ☐ email address and password
- ☐ username and IP address

DevOps_IDP.117_CB

Azure DevOps allows connecting to Azure for infrastructure provisioning by giving you the ability to create a service endpoint. The service endpoint creation dialogue in Azure DevOps allows you to create the service endpoint at the followings scope. (Select two.)

- ■ subscription
- ■ resource group
- ☐ resource
- ☐ web app

DevOps_IDP.118_CB

Which of the following githooks are supported by git locally? (Select three.)

- ■ commit-msg
- ■ pre-push
- ■ pre-commit
- ☐ pre-delete

DevOps_IDP.119_MC

Git supports storing large files in version control by using text pointers inside it's history rather than storing the actual object. What is this technology in git called?

- ☐ git pointers
- ■ git large file storage
- ☐ git blobs
- ☐ virtual git

DevOps_IDP.120_CB

Azure Pipelines integrate with Jenkins. What capabilties of Jenkins can directly be used from Azure Pipelines? (Select two.)

- ■ Jenkins Queue Job
- ■ Jenkins Download Artifacts
- ☐ Jenkins create server
- ☐ Jenkins create new job