Microsoft

Microsoft Official Course

AZ-400T03

Implementing Continuous Delivery

AZ-400T03

Implementing Continuous Delivery

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

MICROSOFT LICENSE TERMS

MICROSOFT INSTRUCTOR-LED COURSEWARE

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

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

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

1. DEFINITIONS.

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

o anything related to the Licensed Content, services, content (including code) on third party Internet sites or third-party programs; and

o 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 contre-façon sont exclues.

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

Cette limitation concerne:

· tout ce qui est relié au le contenu sous licence, aux services ou au contenu (y compris le code) figurant sur des sites Internet tiers ou dans des programmes tiers; et.

· les réclamations au titre de violation de contrat ou de garantie, ou au titre de responsabilité stricte, de négligence ou d'une autre faute dans la limite autorisée par la loi en vigueur.

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

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

Revised November 2014

Contents

Module 0 Welcome

Start Here

Azure DevOps Curriculum

Welcome to the **Implementing Continuous Delivery** 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 Continuous Delivery** area of the AZ-400 certification exam.

About this Course

Course Description

This course provides the knowledge and skills to implement continuous delivery. Students will learn how to design a release strategy, set up a release management workflow, and implement an appropriate deployment pattern.

Level: Intermediate

Audience

Students in this course are interested in DevOps continuous integration 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

After completing this course, students will be able to:

- Differentiate between a release and a deployment
- Define the components of a release pipeline
- Explain things to consider when designing your release strategy
- Classify a release versus a release process, and outline how to control the quality of both
- Describe the principle of release gates and how to deal with release notes and documentation
- Explain deployment patterns, both in the traditional sense and in the modern sense
- Choose a release management tool
- Explain the terminology used in Azure DevOps and other Release Management Tooling
- Describe what a Build and Release task is, what it can do, and some available deployment tasks
- Classify an Agent, Agent Queue and Agent Pool
- Explain why you sometimes need multiple release jobs in one release pipeline
- Differentiate between multi-agent and multi-configuration release job
- Use release variables and stage variables in your release pipeline
- Deploy to an environment securely, using a service connection
- Embed testing in the pipeline

- List the different ways to inspect the health of your pipeline and release by using, alerts, service hooks and reports
- Create a release gate
- Describe deployment patterns
- Implement Blue Green Deployment
- Implement Canary Release
- Implement Progressive Exposure Deployment

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 – Design a Release Strategy

In this module, you'll be introduced to

- Introduction to Continuous Delivery
- Release Strategy Recommendations
- Building a High Quality Release Pipeline
- Choosing a Deployment Pattern
- Choosing the Right Release Management Tool
- Lab: Building a Release Strategy

Module 2 – Set Up a Release Management Workflow

In this module, you'll be introduced to

- Create a Release Pipeline
- Provision and Configure Environments
- Manage and Modularize Tasks and Templates
- Integrate Secrets with the Release Pipeline
- Configure Automated Integration and Functional Test Automation
- Automate Inspection of Health
- Lab: Automating your infrastructure deployments in the Cloud with Terraform and Azure Pipelines
- Lab: Setting up secrets in the pipeline with Azure Key vault
- Lab: Setting up and Running Load Tests
- Lab: Setting up and Running Functional Tests
- Lab: Using Azure Monitor as release gate
- Lab: Creating a Release Dashboard

Module 3 – Implement an Appropriate Deployment Pattern

In this module, you'll be introduced to

- Introduction to Deployment Patterns

- Implement Blue Green Deployment
- Feature Toggles
- Canary Releases
- Dark Launching
- AB Testing
- Progressive Exposure Deployment
- Lab: Blue-Green Deployments
- Lab: Traffic Manager

✓ This course uses the **Microsoft DevOps Lab Environment**[2] to provide a hands-on learning environment.

Demonstration Environment Setup

This course includes the following demonstrations:

- Demonstration Selecting an Artifact Source
- Demonstration Setting Up Stages
- Demonstration Selecting Delivery and Deployment Cadence
- Demonstration Setting Up Manual Approvals
- Demonstration Setting Up Release Gates
- Demonstration Setting Up Service Connections
- Demonstration Creating and Managing Task Groups
- Demonstration Creating and Managing Variable Groups
- Demonstration Setting Up Pipeline Secrets
- Demonstration Setting Up Service Hooks to Monitor the Pipeline
- Demonstration Set Up a Blue-Green Deployment
- Demonstration Ring-based Deployment

To be able to follow along with the demonstrations, you will need to have already set up the following pre-requisites.

Pre-requisites

The focus of these deployments is on Continuous Deployment. They assume that you have already set up Continuous Integration, which you can do by following the steps:

1. Perform Task 1 (only) in the following lab:

Azure DevOps Lab Prerequisites[3]

2. Perform Tasks 1 through 3 in the following lab:

Enabling Continuous Integration with Azure Pipelines[4]

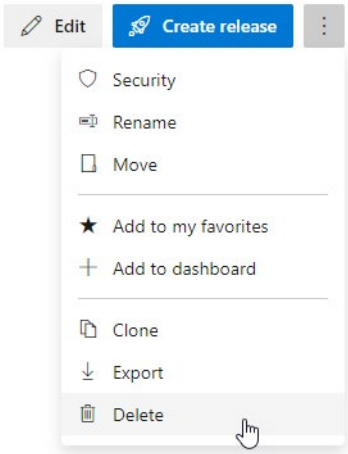
2 https://azuredevopslabs.com/
3 https://azuredevopslabs.com/labs/azuredevops/prereq/
4 https://azuredevopslabs.com/labs/azuredevops/continuousintegration/

Notes on these tasks:

- Task 1 Step 2: There is no Pipelines sub-menu under the Pipelines main menu item. Click Builds instead.
- Task 1 Step 3: The New Pipeline button does not exist as shown. Click "+ New", then "New build pipeline" instead.
- Task 2 Step 2: It is likely that you will already be shown the details for the Agent without having to select it as you are asked to.
- Task 2 Step 4: You might receive a build warning. This can be ignored.
- Task 3 Step 6: There is no Pipelines sub-menu under the Pipelines main menu item. Click Builds instead.

3. Remove any existing release pipelines by doing the following:

- In your Azure DevOps environment, from the main menu, select **Pipelines**, then **Releases**.
- If there are any existing release pipelines, select them and click the ellipsis button beside the Create release button, and click the **Delete** menu item. (NOTE: make sure you are in the correct section and do not delete any build pipelines)



If you are prompted to confirm the deletion, click **OK**.

Repeat this until there are no release pipelines.

4. Ensure that you have an Azure subscription that you can use, and create the following resources:

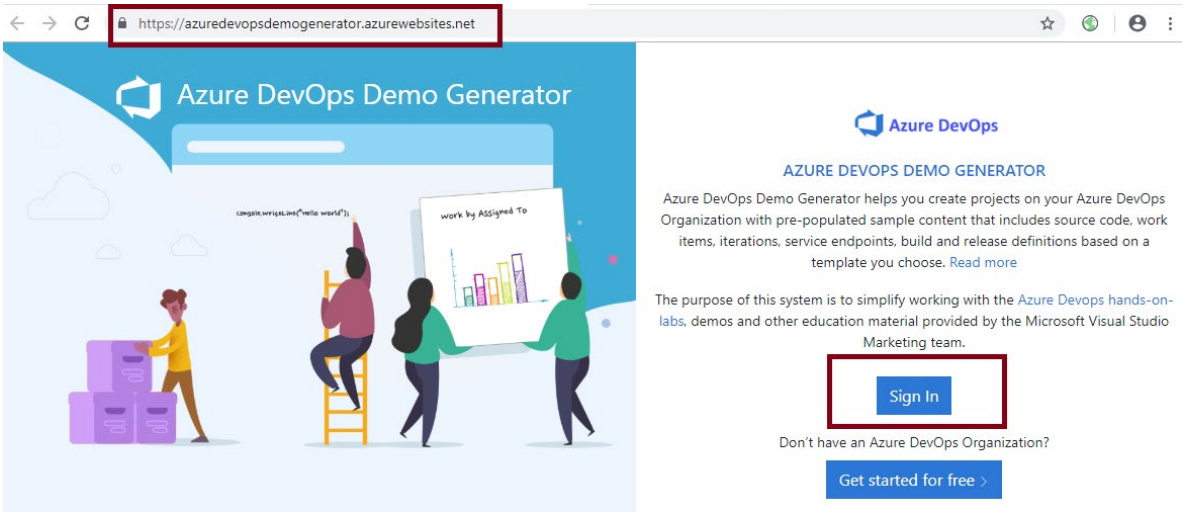
- a resource group
- a storage account in the resource group
- a BLOB container in the storage account
- an Azure Key Vault in the resource group

Note: Any names are suitable and no further configuration is required at this stage.

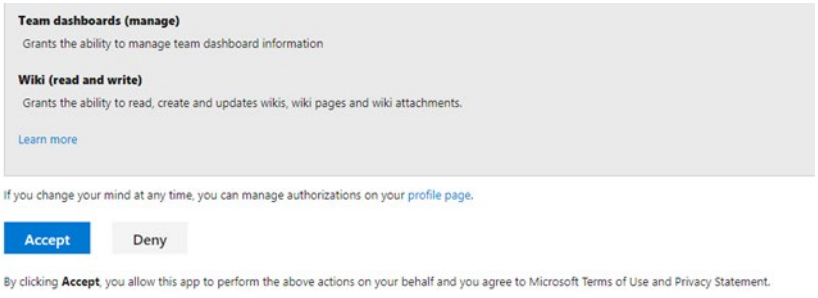
Lab Environment Setup

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**[5]. 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**[6]. 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**[7] program has been created. Click the **Sign In** button and sign in with your Azure DevOps account.

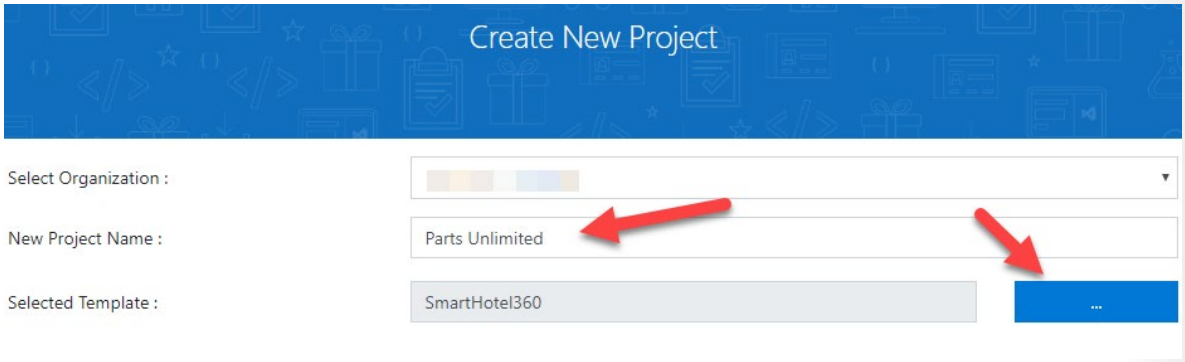


4. You will then be asked to confirm that the generator site can have permission to create objects in your Azure DevOps account.

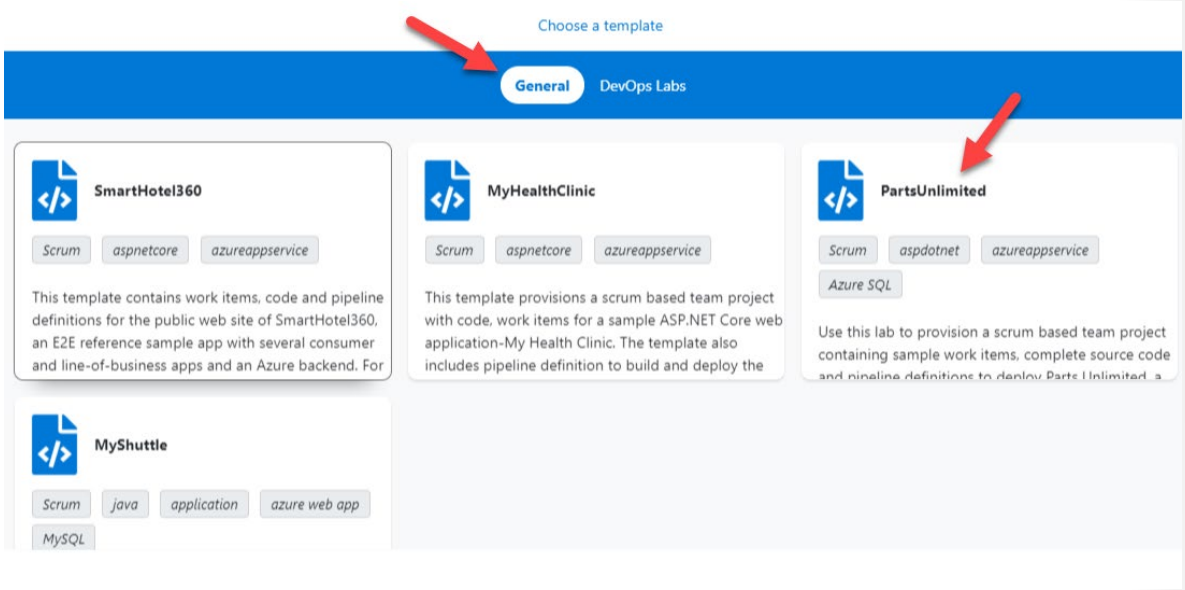


5. If you agree, click the **Accept** button and you should be greeted by the Create New Project screen:

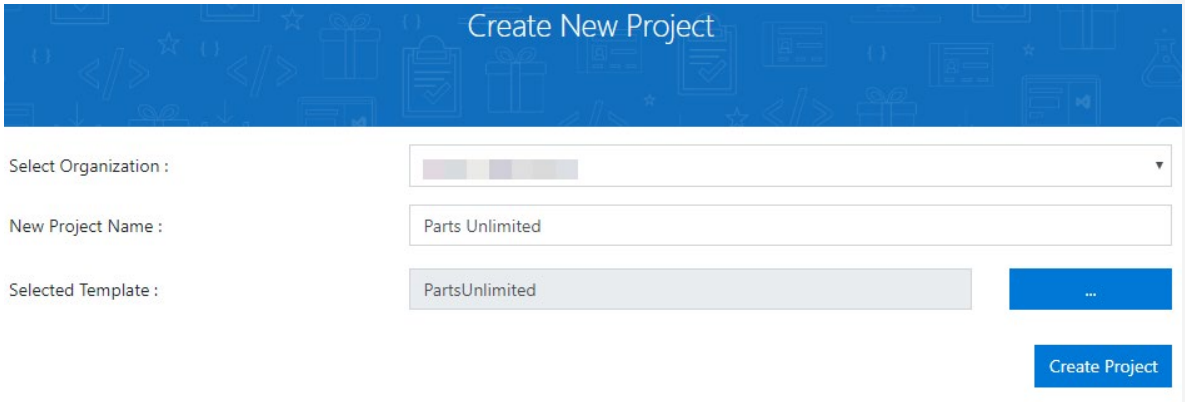
5 https://www.azuredevopslabs.com/
6 https://azure.microsoft.com/en-us/free/
7 https://azuredevopsdemogenerator.azurewebsites.net/



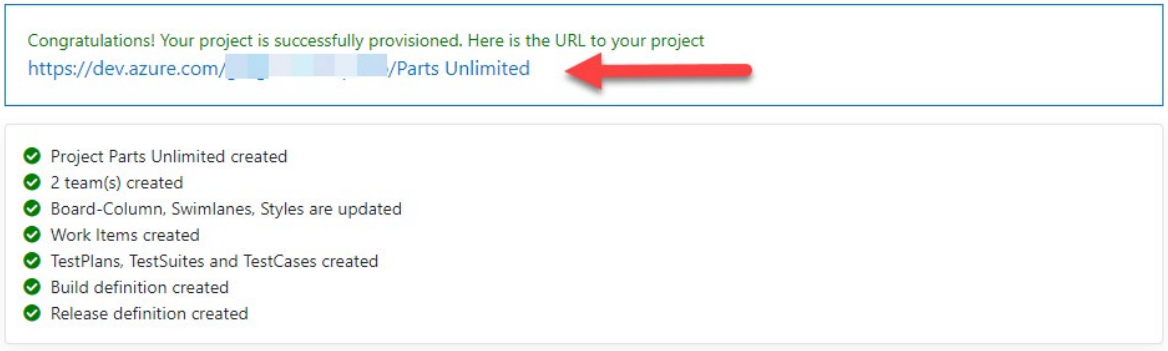
6. Select the appropriate organization (if you have more than one) and enter **Parts Unlimited** as the **New Project Name**, then click the ellipsis to view the available templates. These will change over time but you should see a screen similar to the following:



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 Design a Release Strategy

Introduction to Continuous Delivery

Introduction

Welcome to this module about designing a release strategy. In this module, we will talk about Continuous Delivery in general. In this introduction, we will cover the basics. I'll explain the concepts of Continuous Delivery, Continuous Integration and Continuous Deployment and also the relation to DevOps, and we will discuss why you would you need Continuous Delivery and Continuous Deployment. After that, we will talk about releases and deployments and the differences between those two.

Once we have covered these general topics, we will talk about release strategies and artifact sources, and walk through some considerations when choosing and defining those. We will also discuss the considerations for setting up deployment stages and your delivery and deployment cadence, and lastly about setting up your release approvals.

After that, we will cover some ground to create a high-quality release pipeline and talk about the quality of your release process and the quality of a release and difference between those two. We will take a look at how to visualize your release process quality and how to control your release using release gates as a mechanism. Finally, we will look at how to deal with release notes and documentation.

After these introductions, we will take a brief look at deployment patterns. In module 3 we will dive deeper into the details of the different deployment patterns, but in this module we will cover the basics. We will cover modern deployment patterns like canary releases, but we will also take a quick look at the traditional deployment patterns, like DTAP environments.

Finally, we take a look at choosing the right release management tool. There are a lot of tools out there. We will cover the components that you need to take a look at if you are going to choose the right release management tool product or company.

To finalize this module, we will introduce a hands-on exercise that you can complete in your own pace. This exercise will teach you how to design a release strategy using the components that we have covered in this module.

Learning objectives

At the end of this module, you will be able to:

- Differentiate between a release and a deployment
- Define the components of a release pipeline
- Explain things to consider when designing your release strategy
- Classify a release versus a release process, and outline how to control the quality of both
- Describe the principle of release gates and how to deal with release notes and documentation
- Explain deployment patterns, both in the traditional sense and in the modern sense
- Choose a release management tool

Why Continuous Delivery

In this chapter, we will talk about Continuous Delivery and why this is something you should consider.

We will first talk about Continuous Delivery in general. What is Continuous Delivery, what does it mean, what problems does it solve, and why should you care?

Traditional IT

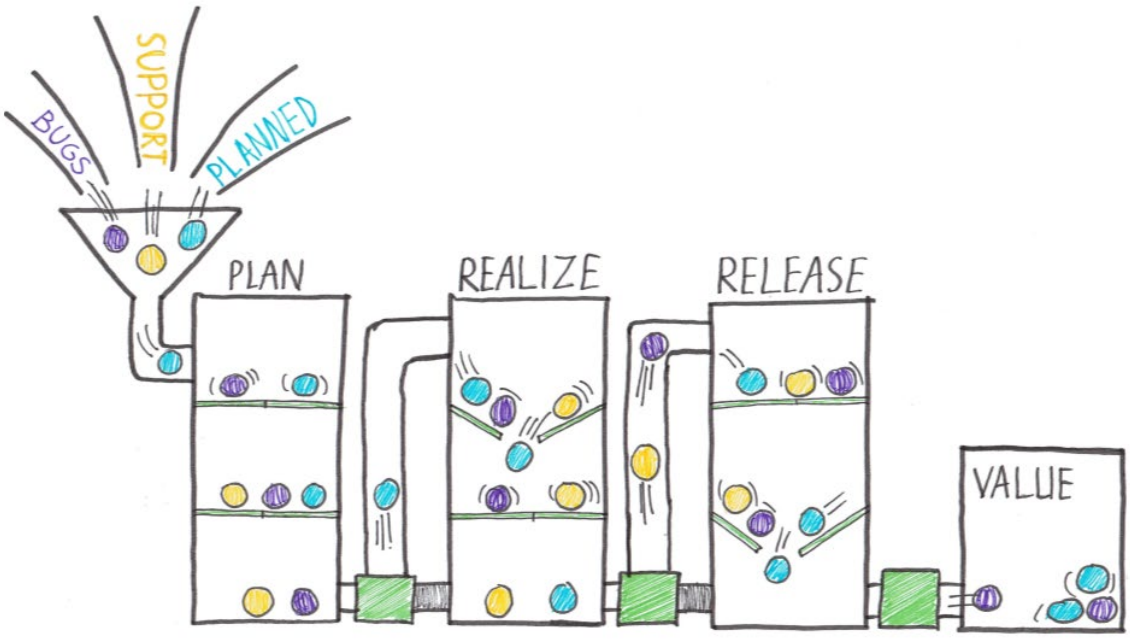
A few years ago, IT was a facilitating department. IT was there to support the business users, and because time had proven that developed software had bad quality by default, software changes were a risk. The resolution for this "quality problem" was to keep changes under strict control. The department that became responsible for controlling the changes became the IT(-Pro) department. In the past but also today, the IT(-Pro) department is responsible for the stability of the systems, while the development department is responsible for creating new value.

This split brings many companies in a difficult situation. Development departments are motivated to deliver value as soon as possible to keep their customers happy. On the other hand, IT is motivated to change nothing, because change is a risk and they are responsible for eliminating the risks and keeping everything stable. And what do we get out of this? Long release cycles.

Silo-Based Development vs Continuous Delivery

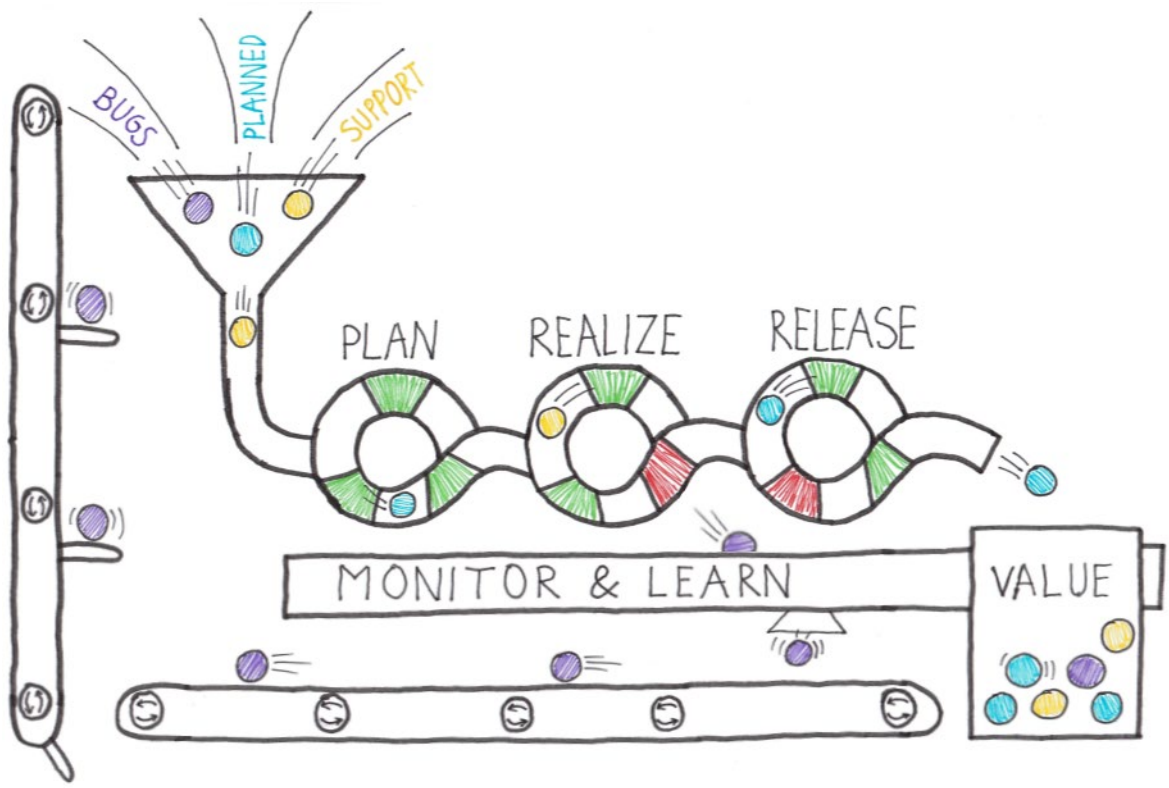
Long release cycles, a lot of testing, code freezes, night and weekend work and a lot of people involved, ensure that everything works. But the more we change, the more risk it entails, and we are back at the beginning. On many occasions resulting in yet another document or process that should be followed.

This is what I call silo-based development.



If we look at this picture of a traditional, silo-based value stream, we see Bugs and Unplanned work, necessary updates or support work and planned (value adding) work, all added to the backlog of the teams. When everything is planned and the first "gate" can be opened, everything drops to the next phase. All the work, and thus all the value moves in piles to the next phase. It moves from Plan phase to a Realize phase where all the work is developed, tested and documented, and from here, it moves to the release phase. All the value is released at the same time. As a result, the release takes a long time.

But times have changed, and we need to deal with a new normal. Our customers demand working software, and they wanted it yesterday. If we cannot deliver, they go to a competitor. And competition is fierce. With the internet, we always have global competition. With competitors on our whole stack but also with competitors that deliver a best of breed tool for one aspect of the software we built. We need to deliver fast, and the product we make must be good. And we should do this with our software production being cheap and quality being high. To achieve this, we need something like Continuous Delivery.



We need to move towards a situation where the value is not piled up and released all at once, but where value flows through a pipeline. Just like in the picture, a piece of work is a marble. And only one piece of work can flow through the pipeline at once. So work has to be prioritized in the right way. As you can see the pipeline has green and red outlets. These are the feedback loops or quality gates that we want to have in place.

A feedback loop can be different things:

- A unit test to validate the code
- An automated build to validate the sources
- An automated test on a Test environment
- Some monitor on a server
- Usage instrumentation in the code

If one of the feedback loops is red, the marble cannot pass the outlet and it will end up in the Monitor and Learn tray. This is where the learning happens. The problem is analyzed and solved so that the next time a marble passes the outlet, it is green.
Every single piece of work flows through the pipeline until it ends up in the tray of value. The more that is automated the faster value flows through the pipeline.

Moving to Continuous Delivery

Companies want to move toward Continuous Delivery. They see the value. They hear their customers. Companies want to deliver their products as fast as possible. Quality should be higher. The move to production should be faster. Technical Debt should be lower.

A great way to improve your software development practices was the introduction of Agile and Scrum. Last year around 80% of all companies claimed that they adopted Scrum as a software development practice. By using Scrum, many teams can produce a working piece of software after a sprint of maybe 2 or 3 weeks. But producing working software is not the same as delivering working software. The result is that all "done" increments are waiting to be delivered in the next release, which is coming in a few months.

What we see now, is that Agile teams within a non-agile company are stuck in a delivery funnel. The bottleneck is no longer the production of working software, but the problem has become the delivery of working software. The finished product is waiting to be delivered to the customers to get business value, but this does not happen. Continuous Delivery needs to solve this problem.

What is Continuous Delivery?

Now that we know a bit more about the necessity to move towards Continuous Delivery, it is good to spend some time on the definition of Continuous Delivery and how this relates to DevOps.

Continuous Delivery (CD) is a set of processes, tools and techniques for the rapid, reliable and continuous development and delivery of software.

This means that Continuous Delivery goes beyond the release of software through a pipeline. The pipeline is a crucial component and also the main focus of this course, but Continuous Delivery is more.

To explain this a bit more, look at the eight principles of Continuous Delivery:

1. The process for releasing/deploying software must be repeatable and reliable.
2. Automate everything!
3. If something is difficult or painful, do it more often.
4. Keep everything in source control.
5. Done means "released."
6. Build quality in!
7. Everybody has responsibility for the release process.
8. Improve continuously.

If we want to fulfill these 8 principles, we can see that an automated pipeline does not suffice. In order to deploy more often, we need to reconsider our software architecture (monoliths are hard to deploy), our testing strategy (manual tests do not scale very well), our organization (separated business and IT departments do not work smoothly), and so forth.

This course will focus on the release management part of Continuous Delivery, but be aware of the other changes you might encounter. Find the next bottleneck in your process, solve it and learn from it, and repeat this forever.

How does this all relate to DevOps? If we look at the definition of DevOps from Donovan Brown:

DevOps is the union of people, process, and products to enable Continuous Delivery of value to our end users

Looking at this definition, Continuous Delivery is an enabler for DevOps. DevOps focuses on organizations and bringing people together to Build and Run their software products.

Continuous Delivery is a practice. Being able to deliver software on-demand. Not necessarily a 1000 times a day. Deploying every code change to production is what we call Continuous Deployment.

To be able to do this we need automation, we need a strategy, and we need pipelines. And this is what we will cover in the rest of the course.

Releases and Deployments

One of the essential steps in moving software more quickly to production is by changing the way we deliver the software to production. In our industry, it is widespread that we have teams that need to do overtime in the weekend to install and release new software. This is mainly caused by the fact that we have two parts of the release process bolted together. The moment we deploy the new software, we also release new features to the end users.

The best way to move your software to production safely while maintaining stability is by separating these two concerns. So we separate deployments from our release. This can also be phrased as separating your functional release from your technical release (deployment).

Separating technical and functional release

When we want to get started with separating the technical and functional release, we need to start with our software itself. The software needs to be built in such a way that new functionality or features can be hidden from end-users while it is running.

A common way to do this, is the use of Feature Toggles. The simplest form of a Feature Toggle is an if statement that either executes or does not execute a certain piece of code. By making the if-statement configurable you can implement the Feature Toggle. We will talk about Feature Toggles in Module 3 in more detail.

More information about Feature Toggles[1]

Once we have prepared our software, we need to make sure that the installation will not expose any new or changed functionality to the end user.

When the software has been deployed, we need to watch how the system behaves. Does it act the same as it did in the past?

If it is clear that the system is stable and operates the same as it did before, we can decide to flip a switch. This might reveal one or more features to the end user, or change a set of routines that are part of the system.

The whole idea of separating deployment from release (exposing features with a switch) is compelling and something we want to incorporate in our Continuous Delivery practice. It helps us with more stable releases and better ways to roll back when we run into issues when we have a new feature that produces problems.

We switch it off again and then create a hotfix. By separating deployment from the release of a feature, you create the opportunity to deploy any time of the day, since the new software will not affect the system that already works.

What is a release and what is a deployment

When we talk about releases and deployments, we see that commonly used tools deal a bit differently with the terminology as we did in the previous chapter. To make sure you both understand the concepts and the technical implementation in many tools, you need to know how tool vendors define the difference between a release and a deployment.

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

A release is a package or container that holds a versioned set of artifacts specified in a release pipeline in your CI/CD process. It includes a snapshot of all the information required to carry out all the tasks and actions in the release pipeline, such as the stages (or environments), the tasks for each one, the values of task parameters and variables, and the release policies such as triggers, approvers, and release queuing options. There can be multiple releases from one release pipeline (or release process).

Deployment is the action of running the tasks for one stage, which results in a tested and deployed application, and other additional activities that are specified for that stage. Initiating a release starts each deployment based on the settings and policies defined in the original release pipeline. There can be multiple deployments of each release even for one stage. When a deployment of a release fails for a stage, you can redeploy the same release to that stage.

Read more about Release and Deployment in a tool perspective[2]

Discussion

What are your bottlenecks?

Have a discussion about the need for Continuous Delivery in your organization and what blocks the implementation.

Topics you might want to discuss are:

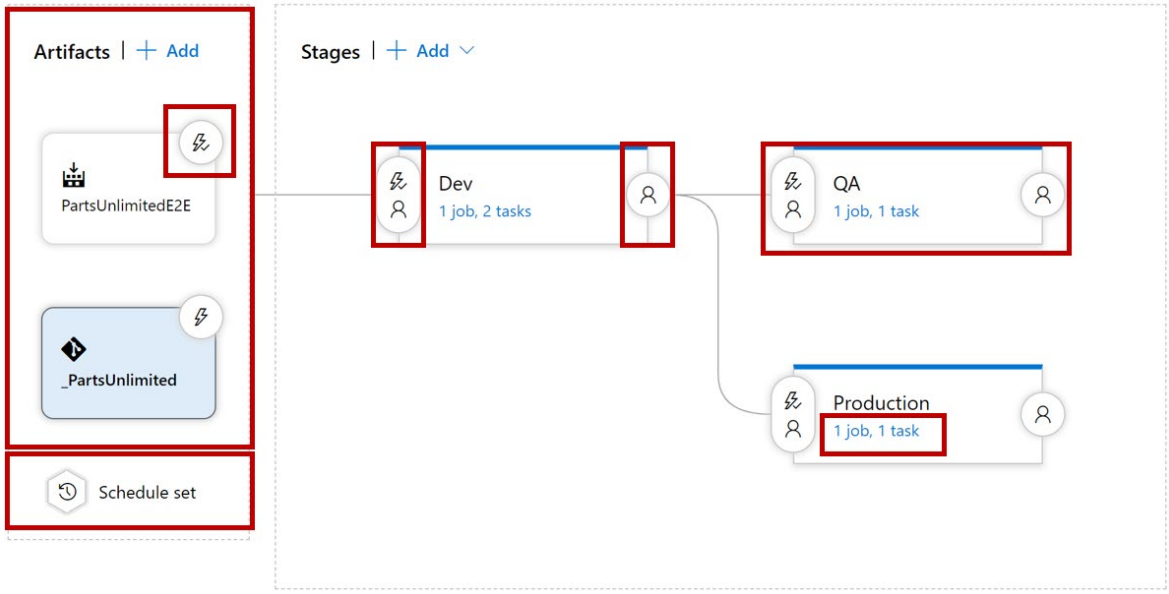
- Does your organization need Continuous Delivery?
- Do you use Agile/Scrum?
    - Is everybody involved or only the Dev departments?
- Can you deploy your application multiple times per day? Why or why not?
- What is the main bottleneck for Continuous Delivery in your organization?
    - The Organization
    - Application Architecture
    - Skills
    - Tooling
    - Tests
    - other things?

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

Release Strategy Recommendations

Release Strategy - Introduction and Overview

Different components make up a release pipeline. In short, a release pipeline takes an artifact and moves this artifact through different stages, so it can eventually be installed on a production environment.



Let us quickly walk through all the components step by step.

The first component in a release pipeline is an artifact. Artifacts can come from different sources. The most common source is a package from a build pipeline. Another commonly seen artifact source is for example source control. Furthermore, a release pipeline has a trigger: the mechanism that starts a new release.

A trigger can be:

- A manual trigger, where people start to release by hand
- A scheduled trigger, where a release is triggered based on a specific time, or
- A continuous deployment trigger, where another event triggers a release. For example, a completed build.

Another vital component of a release pipeline are stages or sometimes called environments. This is where the artifact will be eventually installed. For example, the artifact contains the compiled website, and this will be installed on the web server or somewhere in the cloud. You can have many stages (environments), and part of the release strategy is to find out what the appropriate combination of stages is.

Another component of a release pipeline is approval. In many cases, people want to sign off a release before it is installed on the environment. In more mature organizations, this manual approval process can be replaced by an automatic process that checks the quality before the components move on to the next stage.

Finally, we have the tasks within the various stages. The tasks are the steps that need to be executed to install, configure, and validate the installed artifact.

In this part of the module, we will walk through all the components of the release pipeline in detail and talk about what to consider for each component.

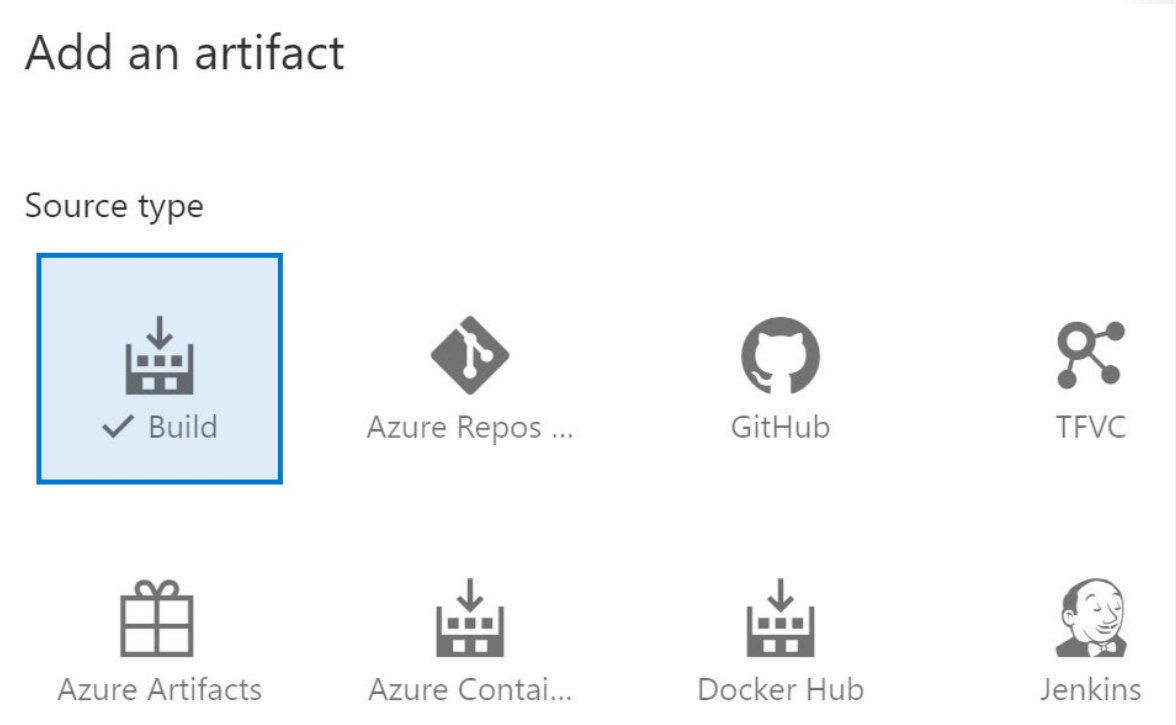
The components that make up the release pipeline or process are used to create a release. There is a difference between a release and the release pipeline or process.
The release pipeline is the blueprint through which releases are done. We will cover more of this when discussing the quality of releases and releases processes.

Understand the difference between a release and a release pipeline[3]

Artifacts and Artifact sources

What is an artifact? An artifact is a deployable component of your application. These components can then be deployed to one or more environments. In general, the idea about build and release pipelines and Continuous Delivery is to build once and deploy many times. This means that an artifact will be deployed to multiple environments. To achieve this, this implies that the artifact is a stable package. The only thing that you want to change when you deploy an artifact to a new environment is the configuration. The contents of the package should never change. This is what we call **immutability**[4]. We should be 100% sure that the package that what we build, the artifact, remains unchanged.

How do we get an artifact? There are different ways to create and retrieve artifacts, and not every method is appropriate for every situation.



The most common and most used way to get an artifact within the release pipeline is to use a build artifact. The build pipeline compiles, tests, and eventually produces an immutable package, which is stored in a secure place (storage account, database etc.).

The release pipeline then uses a secure connection to this secured place to get the build artifact and perform additional actions to deploy this to an environment. The big advantage of using a build artifact is

3 https://docs.microsoft.com/en-us/azure/devops/pipelines/release/?view=vsts
4 https://docs.microsoft.com/en-us/azure/devops/artifacts/artifacts-key-concepts?view=vsts

that the build produces a versioned artifact. The artifact is linked to the build and gives us automatic traceability. We can always find the sources that produced this artifact.

Another possible artifact source is version control. We can directly link our version control to our release pipeline. The release is then related to a specific commit in our version control system. With that, we can also see which version of a file or script is eventually installed. In this case, the version does not come from the build, but from version control. A consideration for choosing a version control artifact instead of a build artifact can be that you only want to deploy one specific file. If no additional actions are required before this file is used in the release pipeline, it does not make sense to create a versioned package containing one that file. Helper scripts that perform actions to support the release process (clean up, rename, string actions) are typically good candidates to get from version control.

Another possibility of an artifact source can be a network share containing a set of files. However, you should be aware of the possible risk. The risk is that you are not 100% sure that the package that you are going to deploy is the same package that was put on the network share. If other people can access the network share as well, the package might be compromised. For that reason, this option will not be sufficient to prove integrity in a regulated environment (banks, insurance companies).

Last but not least, container registries are a rising star when it comes to artifact sources. Container registries are versioned repositories where container artifacts are stored. By pushing a versioned container to the content repository, and consuming that same version within the release pipeline, it has more or less have the same advantages as using a build artifact stored in a safe location.

Considerations for choosing the right artifact source

When you use a release pipeline to deploy your packages to production you need to have some traceability. That means you want to know where the package that you are deploying originates from. It is essential to understand that the sources that you built and checked into your version control are precisely the same as the sources that you are going to deploy to the various environments that are going to be used by your customers. Primarily when you work in a regulated environment like a bank or an insurance company, auditors ask you to provide traceability to sources that you deployed to prove the integrity of the package.

Another crucial aspect of your artifacts is auditability. You need to know who changed that line of code and who triggered the build that produces the artifact that is being deployed.

A useful mechanism to make sure you can provide the right traceability and auditability is using immutable packages. That is not something that you can buy, but something that you need to implement yourself. By using a build pipeline that produces a package which is stored in a location that cannot be accessed by humans, you ensure the sources are unchanged throughout the whole release process. This is an essential concept of release pipelines.

You identify an immutable package by giving it a version so that you can refer to it in a later stage. Versioning strategy is a complex concept and is not in the scope of this module, but by having a unique identification number or label attached to the package, and making sure that this number or label cannot be changed or modified afterwards, you ensure traceability and auditability from source code to production.

Read more about Semantic Versioning[5]

5 https://semver.org/

Choosing the right artifact source is tightly related to the requirements you have regarding traceability and auditability. If you need an immutable package (containing multiple files) that can never be changed and be traced, a build artifact is the best choice. If it is one file, you can directly link to source control.

You can also point at a disk or network share, but this implies some risk concerning auditability and immutability. Can you ensure the package never changed?

Everything you need to know about Artifacts and Artifact Sources in Azure DevOps can be found on the

Microsoft Docs[6]

Demonstration Selecting an Artifact Source

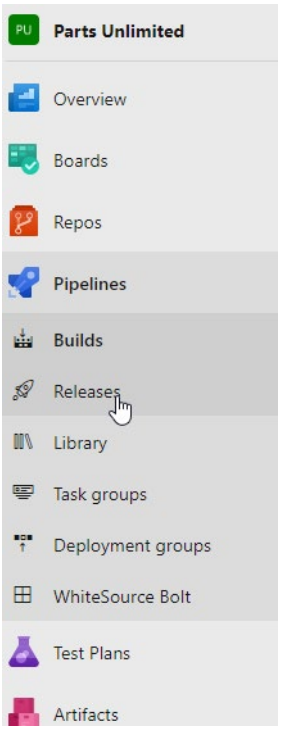
In this demonstration, you will investigate Artifact Sources.

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

Steps

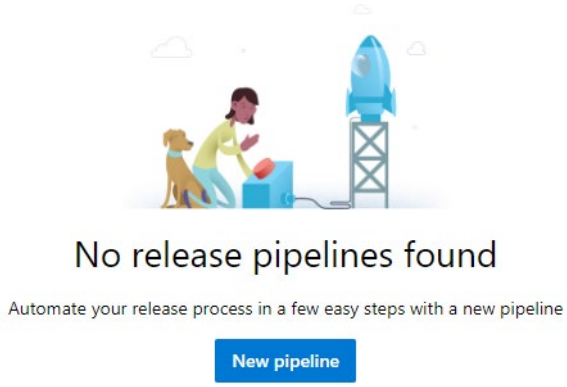
Let's take a look at how to work with one or more artifact sources in the release pipeline.

1. In the Azure DevOps environment, open the **Parts Unlimited** project, then from the main menu, click **Pipelines**, then click **Releases**.

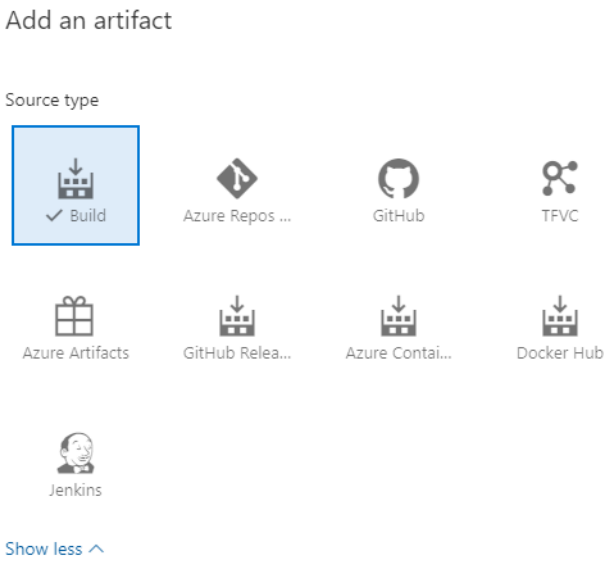


2. In the main screen area, click **New pipeline**.

6 https://docs.microsoft.com/en-us/azure/devops/pipelines/release/artifacts?view=vsts

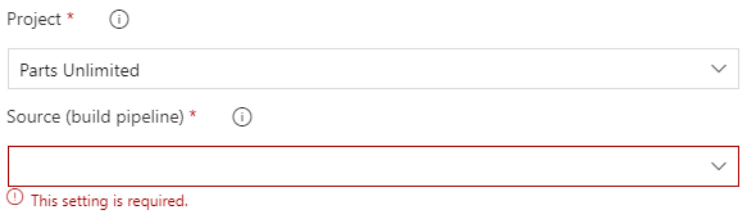


3. In the **Select a template** pane, note the available templates, but then click the **Empty job** option at the top. This is because we are going to focus on selecting an artifact source.
4. In the **Artifacts** section, click **+Add an artifact**.
5. Note the available options in the **Add an artifact** pane, and click the option to see **more artifact types**, so that you can see all the available artifact types:

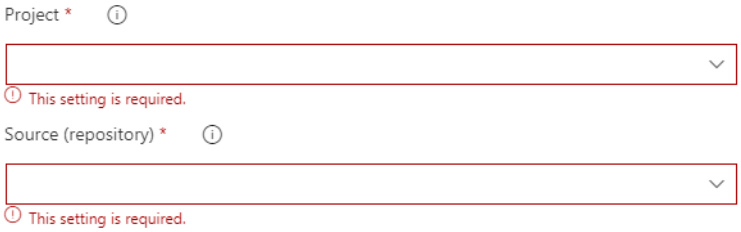


While we're in this section, let's briefly look at the available options.

6. Click **Build** and note the parameters required. This option is used to retrieve artifacts from an Azure DevOps Build pipeline. Using it requires a project name, and a build pipeline name. (Note that projects can have multiple build pipelines). This is the option that we will use shortly.

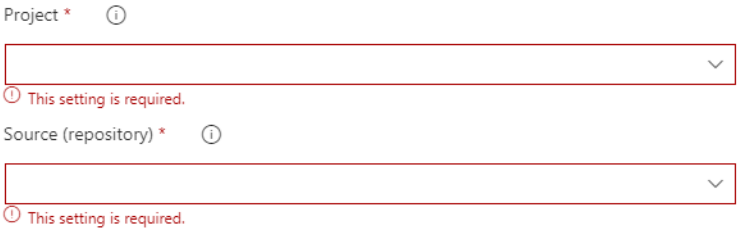


7. Click **Azure Repository** and note the parameters required. It requires a project name and also asks you to select the source repository.



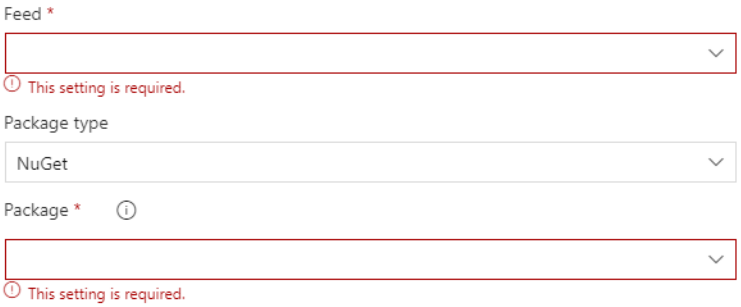
8. Click **GitHub** and note the parameters required. The **Service** is a connection to the GitHub repository. It can be authorized by either OAuth or by using a GitHub personal access token. You also need to select the source repository.

9. Click **TFVC** and note the parameters required. It also requires a project name and also asks you to select the source repository.

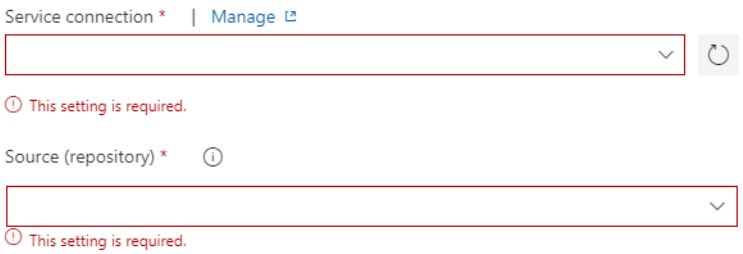


Note: A release pipeline can have more than one set of artifacts as input. A common example is a situation where as well as your project source, you also need to consume a package from a feed.

10. Click **Azure Artifacts** and note the parameters required. It requires you to identify the feed, package type, and package.

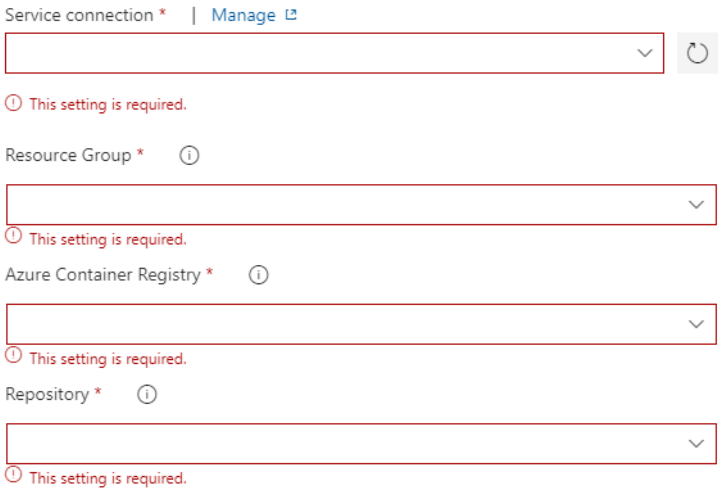


11. Click **GitHub Release** and note the parameters required. It requires a service connection and the source repository.

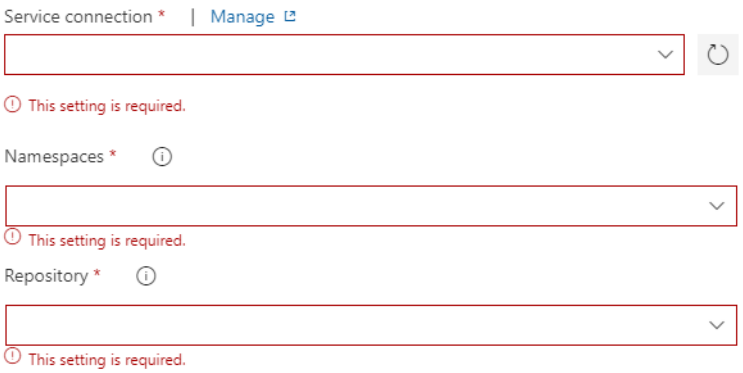


Note: we will discuss service connections later in the course.

12. Click **Azure Container Registry** and note the parameters required. Again, this requires a secure service connection, along with details of the Azure Resource Group that the container registry is located in. This allows you to provide all your Docker containers directly into your release pipeline.



13. Click **Docker Hub** and note the parameters required. This option would be useful if your containers are stored in Docker Hub rather than in an Azure Container Registry. After choosing a secure service connection, you need to select the namespace and the repository

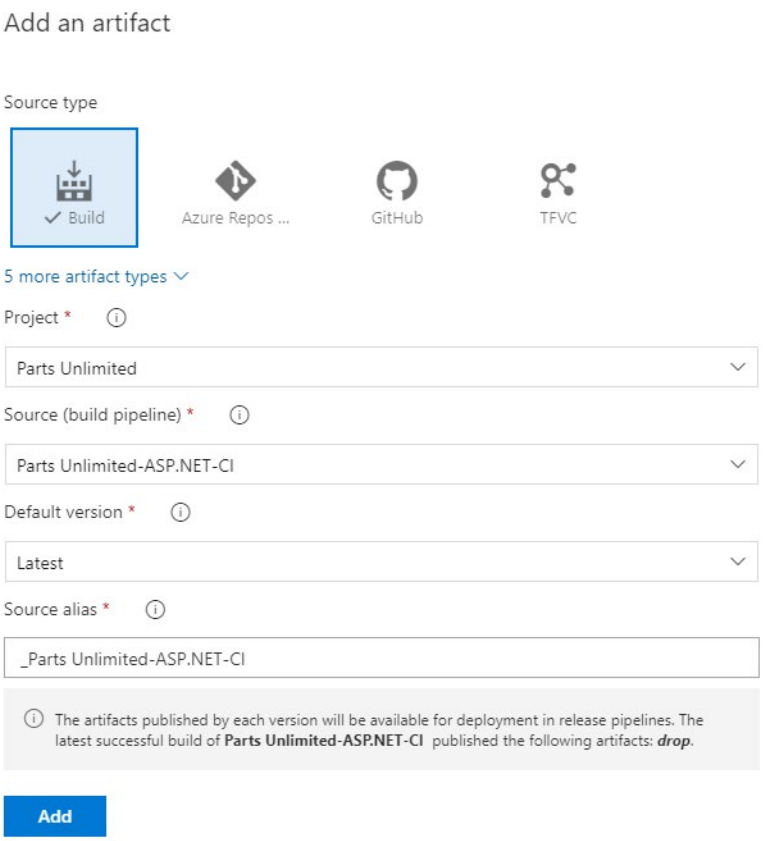


14. Last but not least, click **Jenkins** and note the parameters required. You do not need to get all your artifacts from Azure. You can retrieve them from a Jenkins build. So if you have a Jenkins Server in your infrastructure, you can use the build artifacts from there, directly in your Azure DevOps pipelines.

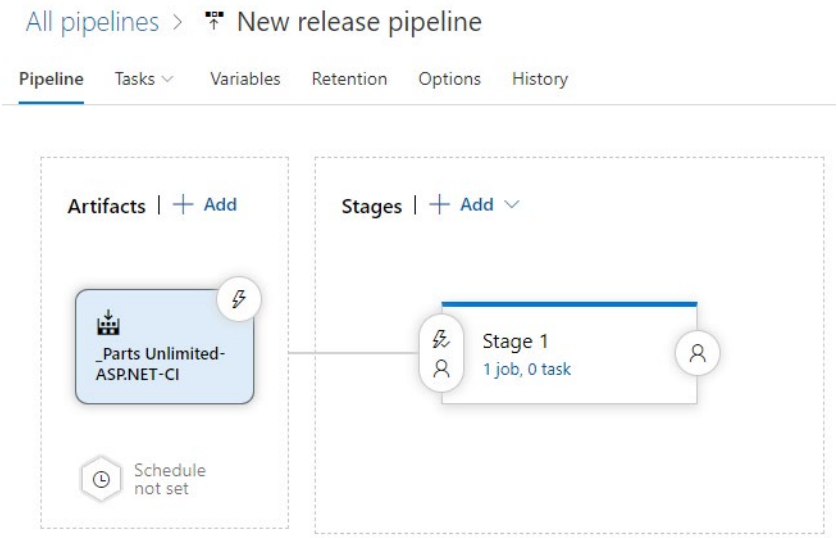
Configuring the Build Artifact

Let's return to adding our Build output as the artifact source.

15. Click the **Build** source type again. Note that the Project should show the current project. From the **Source (build pipeline)** drop down list, select **Parts Unlimited-ASP.NET-CI**. Take note of the default values for the other options, and then click **Add**.



We have now added the artifacts that we will need for later walkthroughs.



16. To save the work, click **Save**, then in the Save dialog box, click **OK**.

Deployment Stages

A stage or deployment stage is a logical and independent entity that represents where you want to deploy a release generated from a release pipeline. Sometimes a stage is called an environment. For

example Test or Production. But it does not neccesarily reflect the lifecycle of a product. It can represent any physical or real stage that you need. For example, the deployment in a stage may be to a collection of servers, a cloud, or multiple clouds. In fact, you can even use a stage to represent shipping the software to an app store, or the manufacturing process of a boxed product, or a way to group a cohort of users for a specific version of an application.

You must be able to deploy to a stage independently of other stages in the pipeline. There should be no dependency between stages in your pipeline. For example, your pipeline might consist of two stages A and B, and your pipeline could deploy Release 2 to A and Release 1 to B. If you make any assumptions in B about the existence of a certain release in A, the two stages are not independent.

Here are some suggestions and examples for stages:

- Dev, QA, Prod - As new builds are produced, they can be deployed to Dev. They can then be promoted to QA, and finally to Prod. At any time, each of these stages may have a different release (set of build artifacts) deployed to them. This is a good example of the use of stages in a release pipeline.
- Customer adoption rings (for example, early adopter ring, frequent adopter ring, late adopter ring) - You typically want to deploy new or beta releases to your early adopters more often than to other users. Therefore, you are likely to have different releases in each of these rings. This is a good example of the use of stages in a pipeline.
- Database and web tiers of an application - These should be modeled as a single stage because you want the two to be in sync. If you model these as separate stages, you risk deploying one build to the database stage and a different build to the web tier stage.
- Staging and production slots of a web site - There is clearly an interdependence between these two slots. You do not want the production slot to be deployed independently of the build version currently deployed to the staging slot. Therefore, you must model the deployment to both the staging and production slots as a single stage.
- Multiple geographic sites with the same application - In this example, you want to deploy your website to many geographically distributed sites around the globe and you want all of them to be the same version. You want to deploy the new version of your application to a staging slot in all the sites, test it, and - if all of them pass - swap all the staging slots to production slots. In this case, given the interdependence between the sites, you cannot model each site as a different stage. Instead, you must model this as a single stage with parallel deployment to multiple sites (typically by using jobs).
- Multiple test stages to test the same application - Having one or more release pipelines, each with multiple stages intended to run test automation for a build, is a common practice. This is fine if each of the stages deploys the build independently, and then runs tests. However, if you set up the first stage to deploy the build, and subsequent stages to test the same shared deployment, you risk overriding the shared stage with a newer build while testing of the previous builds is still in progress.

Considerations for setting up Deployment Stages

Now that we know what stage is and have some idea of what is commonly used as a stage, we need to review a few considerations around choosing the right stages. In most cases, organizations prefer the traditional approach when it comes to setting up the staging strategy. With the traditional method, you can think of setting up an environment for *Dev*, for *Test*, and *Production*. On many occasions, there is even an additional stage, *Staging* or *Acceptance*.

Even when companies use a cloud strategy, they still apply the general sense of environments. When applications are deployed to a cloud instead of a server in a data centre, you have the possibility to

rethink your strategy around stages. For example, a stage is not necessarily a long-lived entity. When we talk about Continuous Delivery, where we might deploy our application multiple times a day, we may assume that the application is also tested every time the application is deployed. The question that we need to ask ourselves is, do we want to test in an environment that is already in use, or do we want a testing environment that is clean from the start?

On many occasions both scenarios are valid. Sometimes you want to start from scratch, and sometimes you want to know what happens if you refresh the environment. In a DevOps world, we see infrastructure as just another piece of software (Infrastructure as Code). Using Cloud technology combined with Infrastructure as Code gives us new possibilities when it comes to environments. We are not limited to a fixed number of environments anymore. Instead, we can spin up environments on demand. When we want to test something, we spin up a new environment, deploy our code and run our tests. When we are done, we can clean up the environment. Traditional labels for environments, therefore, do not apply anymore. Let's take Test as an example. Maybe we have different test environments, one for load testing, one for integration testing, one for system testing and one for functional testing. The sky is the limit!

Depending on the needs of the organization and the DevOps teams, the number of stages and the purpose of stages vary. Some organizations stick to the DTAP (Dev, Test, Acceptance, Production) where others deploy directly to production with temporary stages in between.

Important things to consider are there for

- Is your stage long lived or short lived?
- What is the purpose of this specific stage?
- Who is going to use it?
- Is your target application overwriting an existing one would always be a fresh install?
- Do you need a new stage for bug fixes?
- Do you need an isolated environment with encrypted data or disconnected from a network?
- Can you afford downtime?
- Who is the owner of the stage? Who can apply changes?

These and maybe other considerations need to play a crucial role in defining the number of stages and the purpose of stages.

Everything you need to know about Deployment stages in combination with Azure DevOps you can find on the **Microsoft Docs**[7]

Discussion

What deployment stages would you define for your organization?

Have a discussion about what deployment stages you recognise in your organization?

Consider the following things:

- Is your stage long lived or short lived?
- What is the purpose of this specific stage?
- Who is going to use it?

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

- Is your target application overwriting an existing one would always be a fresh install?
- Do you need a new stage for bug fixes?
- Do you need an isolated environment with encrypted data or disconnected from a network?
- Can you afford downtime?
- Who is the owner of the stage? Who can apply changes?

Demonstration Setting Up Stages

In this demonstration, you will investigate Stages.

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

Steps

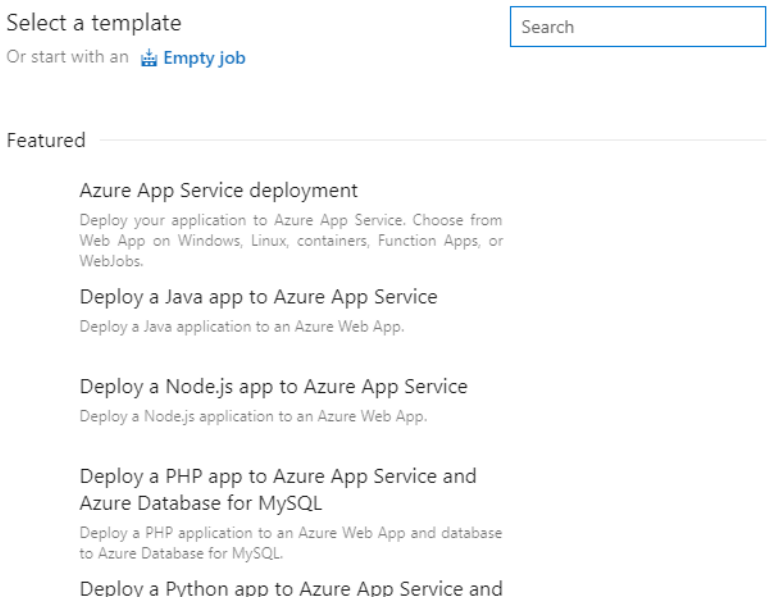
Let's now take a look at the other section in the release pipeline that we have created: Stages.

1. Click on **Stage 1** and in the Stage properties pane, set **Stage name** to **Development** and close the pane.

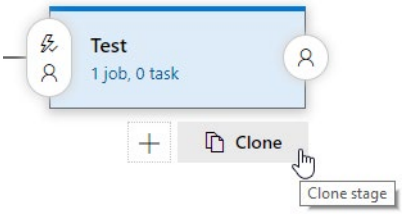
Note: stages can be based on templates. For example, you might be deploying a web application using node.js or Python. For this walkthrough, that won't matter because we are just focussing on defining a strategy.

2. To add a second stage, click **+Add** in the Stages section and note the available options. You have a choice to create a new stage, or to clone an existing stage. Cloning a stage can be very helpful in minimizing the number of parameters that need to be configured. But for now, just click **New stage**.

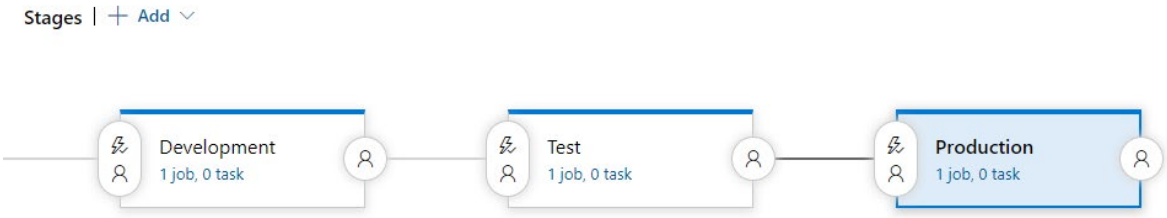
3. When the **Select a template** pane appears, scroll down to see the available templates. For now, we don't need any of these, so just click **Empty job** at the top, then in the Stage properties pane, set **Stage name** to **Test**, then close the pane.



4. Hover over the **Test** stage and notice that two icons appear below. These are the same options that were available in the menu drop down that we used before. Click the **Clone** icon to clone the stage to a new stage.



5. Click on the **Copy of Test** stage and in the stage properties pane, set **Stage name** to **Production** and close the pane.



We have now defined a very traditional deployment strategy. Each of the stages contains a set of tasks, and we will look at those tasks later in the course.

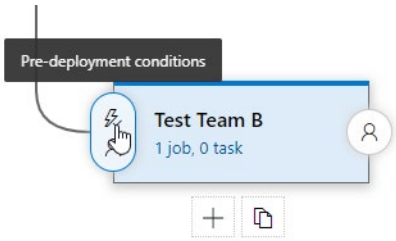
*Note: The same artifact sources move through each of the stages.

The lightning bolt icon on each stage shows that we can set a trigger as a predeployment condition. The person icon on both ends of a stage, show that we can have pre and post deployment approvers.

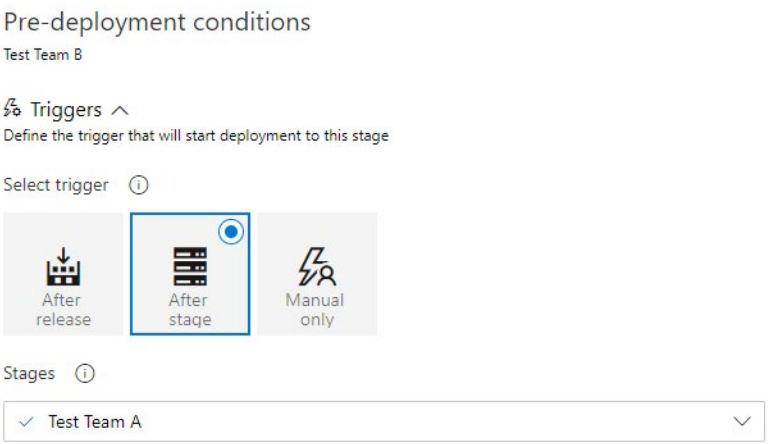
Concurrent stages

You'll notice that at present we have all the stages one after each other in a sequence. It is also possible to have concurrent stages. Let's see an example.

6. Click the **Test** stage, and on the stage properties pane, set **Stage name** to **Test Team A** and close the pane.
7. Hover over the **Test Team A** stage, and click the **Clone** icon that appears, to create a new cloned stage.
8. Click the **Copy of Test Team A** stage, and on the stage properties pane, set **Stage name** to **Test Team B** and close the pane.
9. Click the **Pre-deployment conditions** icon (i.e. the lightning bolt) on **Test Team B** to open the pre-deployment settings.



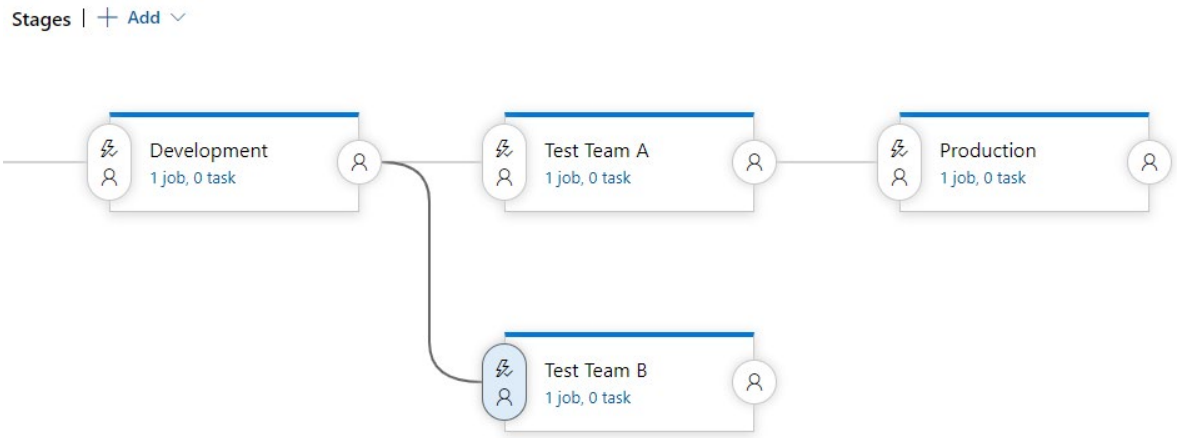
10. In the Pre-deployment conditions pane, note that the stage can be triggered in three different ways:



The stage can immediately follow Release. (That is how the Development stage is currently configured). It can require manual triggering. Or, more commonly, it can follow another stage. At present, it is following **Test Team A** but that's not what we want.

11. From the **Stages** drop down list, chose **Development** and uncheck **Test Team A**, then close the pane.

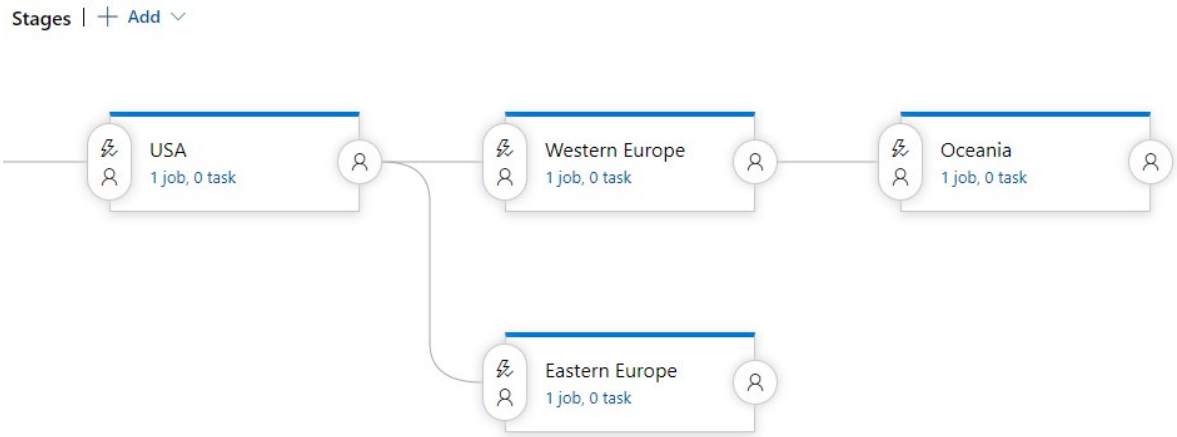
We now have two concurrent Test stages.



Stage vs Environment

You may have wondered why these items are called **Stages** and not **Environments**.

In the current configuration, we are in fact using them for different environments. But this is not always the case. Here is a deployment strategy based upon regions instead:



Azure DevOps pipelines are very configurable and support a wide variety of deployment strategies. The name **Stages** is a better fit than **Environment** even though the stages can be used for environments.

For now, let's give the pipeline a better name and save the work.

12. At the top of the screen, hover over the **New release pipeline** name and when a pencil appears, click it to edit the name. Type **Release to all environments** as the name and hit enter or click elsewhere on the screen.

12. For now, save the environment based release pipeline that you have created by clicking **Save**, then in the Save dialog box, click **OK**.

Delivery and Deployment Cadence, Schedules and Triggers

When you have a clear view of your different stages, you need to think about when you want to deploy to these stages. Then there is a difference between a release and a deployment, as we discussed earlier. The moment you create the release (the package that holds a versioned set of artifacts), can differ from the moment you deploy to an environment.

But both the release and stages make use of triggers. There are three types of triggers we recognize.

Continuous deployment Trigger
You can set up this trigger on your release pipeline. Once you do that, every time a build completes, your release pipeline will trigger, and a new release will be created.

Scheduled Triggers
This speaks for itself, but what it allows you to, is to set up time-based manner to start a new release. For example every night at 3:00 AM or at 12:00 PM. You can have one or multiple schedules per day, but it will always run on this specific time.

Manual trigger
With a manual trigger, a person or system triggers the release based on a specific event. When it is a person, it probably uses some UI to start a new release. When it is an automated process most likely, some event will occur, and by using the automation engine, which is usually part of the release management tool, you can trigger the release from another system.

As we mentioned in the introduction, Continuous Delivery is not only about deploying multiple times a day, it is about being able to deploy on demand. When we define our cadence, questions that we should ask ourselves are:

- Do we want to deploy our application?
- Do we want to deploy multiple times a day
- Can we deploy to a stage? Is it used?

    - For example, when a tester is testing an application during the day might not want to deploy a new version of the app during the test phase.
    - Another example, when your application incurs downtime, you do not want to deploy when users are using the application.

The frequency of deployment, or cadence, differs from stage to stage. A typical scenario that we often see is that continuous deployment happens to the development stage. Every new change ends up there once it is completed and builds. Deploying to the next phase does not always occur multiple times a day but only during the night.

When you are designing your release strategy, choose your triggers carefully and think about the required release cadence.

Some things we need to take into consideration are

- What is your target environment?
- Is it used by one team or is it used by multiple teams?
    - If a single team uses it, you can deploy frequently. Otherwise, you need to be a bit more careful.
- Who are the users? Do they want a new version multiple times a day?
- How long does it take to deploy?
- Is there downtime? What happens to performance? Are users impacted?

Some tools make a difference between a release and deployment. This is what we talked about in the introduction. You have to realise that a Trigger for the release pipeline only creates a new release. In most cases, you also need to set up triggers for the various stages as well to start deployments. For example, you can set up an automatic deployment to the first stage after the creation of a release. And, after that, when the deployment to the first stage is successful, start the deployment to the next stage(s).

More information about triggers[8]

More information about Stage Triggers[9]

Demonstration Selecting Delivery and Deployment Cadence

In this demonstration, you will investigate Delivery Cadence.

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

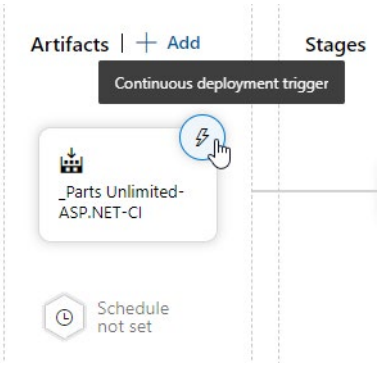
Steps

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

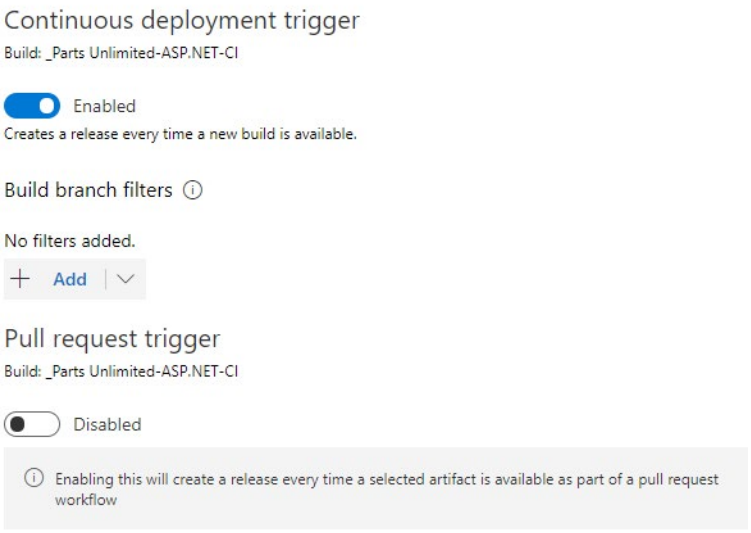
When we refer to a deployment, we are referring to each individual stage, and each stage can have its own set of triggers that determine when the deployment occurs.

1. Click the lightning bolt on the **_Parts Unlimited-ASP.NET-CI** artifact.

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



2. In the Continuous deployment trigger pane, click the **Disabled** option to enable continuous deployment. It will then say **Enabled**.



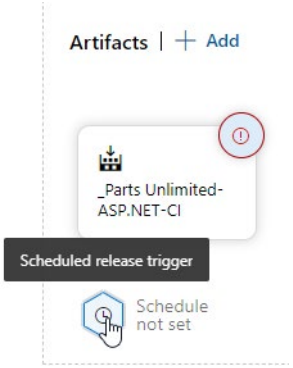
Once this is selected, every time that a build completes, a deployment of the release pipeline will start.

Note: You can filter which branches affect this, so for example you could choose the master branch or a particular feature branch.

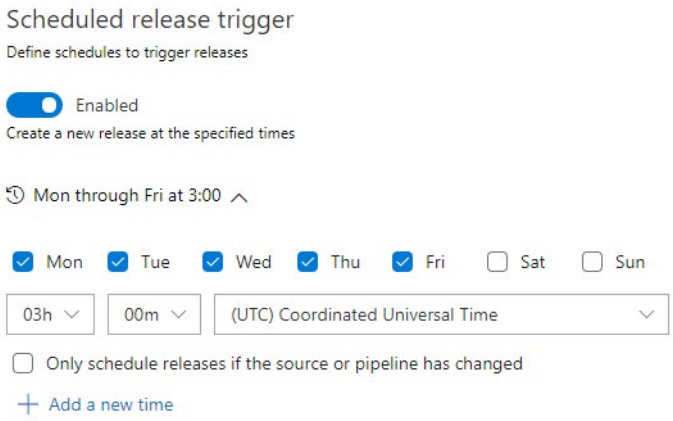
Scheduled Deployments

You might not want to have a deployment commence every time a build completes. That might be very disruptive to testers downstream if it was happening too often. Instead, it might make sense to set up a deployment schedule.

3. Click on the **Scheduled release trigger** icon to open its settings.



4. In the Scheduled release trigger pane, click the **Disabled** option to enable scheduled release. It will then say **Enabled** and additional options will appear.



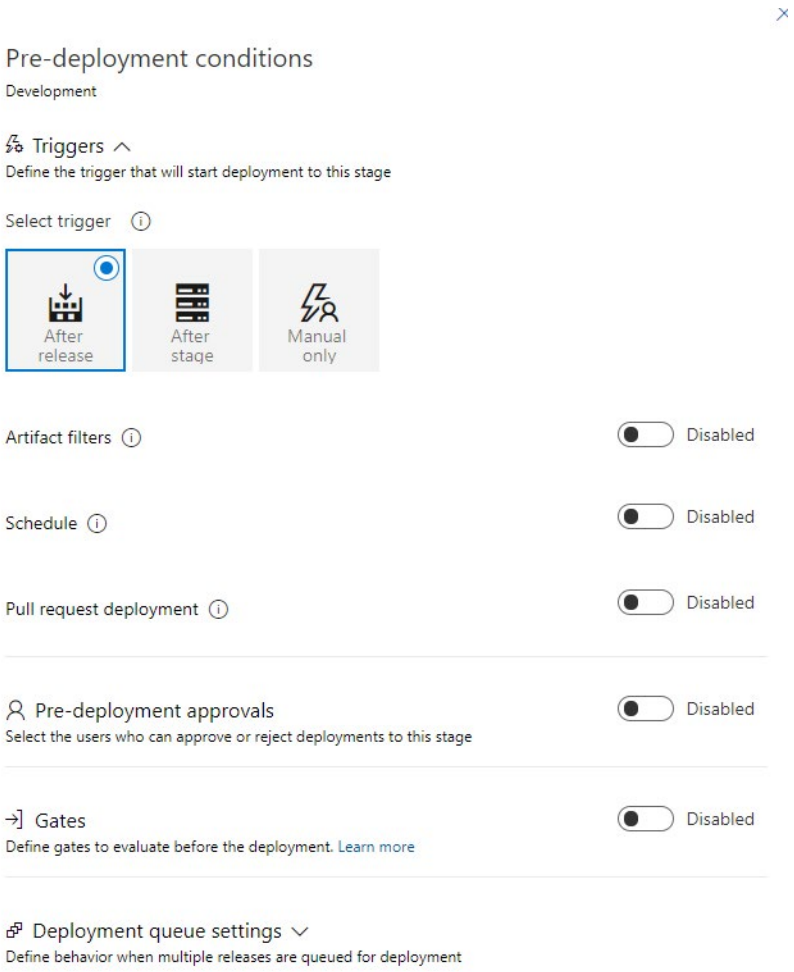
You can see in the screenshot above that a deployment using the release pipeline would now occur each weekday at 3AM. This might be convenient when you for example, share a stage with testers who work during the day. You don't want to constantly deploy new versions to that stage while they're working. This setting would create a clean fresh environment for them at 3AM each weekday.

Note: The default timezone is UTC. You can change this to suit your local timezone as this might be easier to work with when creating schedules.

5. For now, we don't need a scheduled deployment, so click the **Enabled** button again to disable the scheduled release trigger and close the pane.

Pre-deployment Triggers

6. Click the lightning bolt on the **Development** stage to open the pre-deployment conditions.



Note that both artifact filters and a schedule can be set at the pre-deployment for each stage rather than just at the artifact configuration level.

Deployment to any stage doesn't happen automatically unless you have chosen to allow that.

Considerations for Release Approvals

In this part, we will talk about release approvals and release gates.

First, we take a look at manual approvals. This does not always resonate well when we talk about Continuous Delivery, because Continuous Delivery implies that you deliver multiple times a day.

As we have described in the introduction, Continuous Delivery is all about delivering on demand. But, as we discussed in the differences between release and deployment, delivery, or deployment, is only the technical part of the Continuous Delivery process. It is all about how you are technically able to install the software on an environment, but it does not say anything about the process that needs to be in place for a release.

Release approvals are not to control *how*, but control *if* you want to deliver multiple times a day.

Manual approvals also suit a significant need. Organizations that start with Continous Delivery often lack a certain amount of trust. They do not dare to release without a manual approval. After a while, when they find that the approval does not add any value and the release always succeeds, the manual approval is often replaced by an automatic check.

Things to consider when you are setting up a release approval are:

- What do we want to achieve with the approval?
  Is it an approval that we need for compliance reasons? For example. We need to adhere to the four-eyes principal to get out SOX compliance. Or, Is it an approval that we need to manage our dependencies. Or, is it an approval that needs to be in place purely because we need a sign off from an authority like Security Officers or Product Owners.
- Who needs to approve?
  We need to know who needs to approve the release. Is it a product owner, Security officer, or just someone that is not the one that wrote the code. This is important because the approver is part of the process. He is the one that can delay the process if not available. So be aware that.
- When do you want to approve?
  Another essential thing to consider is when to approve. This is a direct relationship with what happens after approval. Can you continue without approval? Or, is everything on hold until approval is given. By using scheduled deployments, you can separate approval from deployment.

Although manual approval is a great mechanism to control the release, it is not always useful. On many occasions, the check can be done in an earlier stage. For example, approving a change that has been made in Source Control.

Scheduled deployments already solve the dependency issue. You do not have to wait for a person in the middle of the night. But there is still a manual action involved. If you want to eliminate manual activities altogether, but still want to have control you start talking about automatic approvals or release gates.

Example:

In many organizations, there are so-called dependency meetings. This is a planning session where the release schedule of dependent components is discussed. Think of downtime of a database server or an update of an API. This takes a lot of time an effort, and the only thing that is needed is a signal if the release can proceed. Instead of having this meeting you can also create a mechanism where people press a button on a form when the release cannot advance. When the release starts, it checks the state of the gate by calling an API. If the "gate" is open, we can continue. Otherwise, we stop the release.

By using scripts and API's, you can create your own release gates instead of a manual approval. Or at least extending your manual approval. Other scenarios for automatic approvals are for example.

- Incident and issues management. Ensure the required status for work items, incidents, and issues. For example, ensure that deployment only occurs if no bugs exist.
- Notify users such as legal approval departments, auditors, or IT managers about a deployment by integrating with approval collaboration systems such as Microsoft Teams or Slack, and waiting for the approval to complete.
- Quality validation. Query metrics from tests on the build artifacts such as pass rate or code coverage and only deploy if they are within required thresholds.
- Security scan on artifacts. Ensure security scans such as anti-virus checking, code signing, and policy checking for build artifacts have completed. A gate might initiate the scan and wait for it to complete, or check for completion.
- User experience relative to baseline. Using product telemetry, ensure the user experience hasn't regressed from the baseline state. The experience level before the deployment could be considered a baseline.
- Change management. Wait for change management procedures in a system such as ServiceNow complete before the deployment occurs.

- Infrastructure health. Execute monitoring and validate the infrastructure against compliance rules after deployment, or wait for proper resource utilisation and a positive security report.

In short, approvals and gates give you additional control over the start and completion of the deployment pipeline. They can usually be set up as a pre-deployment and post-deployment condition, that can include waiting for users to approve or reject deployments manually, and checking with other automated systems until specific requirements are verified. In addition, you can configure a manual intervention to pause the deployment pipeline and prompt users to carry out manual tasks, then resume or reject the deployment.

To find out more about Release Approvals and Gates, check these documents.

- **Release approvals and gates overview**[10]
- **Release Approvals**[11]
- **Release Gates**[12]

Demonstration Setting Up Manual Approvals

In this demonstration, you will investigate Manual Approval.

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

Steps

Let's now take a look at when our release pipeline needs manual approval before deployment of a stage starts, or manual approval that the deployment of a stage completed as expected.

While DevOps is all about automation, manual approvals are still very useful. There are many scenarios where they are needed. For example, a product owner might want to sign off a release before it moves to production. Or the scrum team wans to make sure that no new software is deployed to the test environment before someone signs off on it, because they might need to find an appropriate time if it's constantly in use.

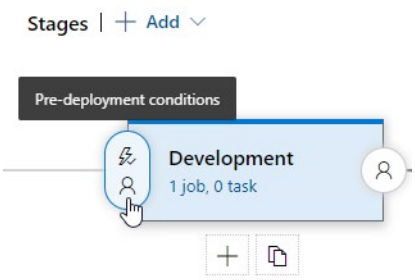
This can help to gain trust in the DevOps processes within the business.

Even if the process will later be automated, people might still want to have a level of manual control until they become comfortable with the processes. Explicit manual approvals can be a great way to achieve that.

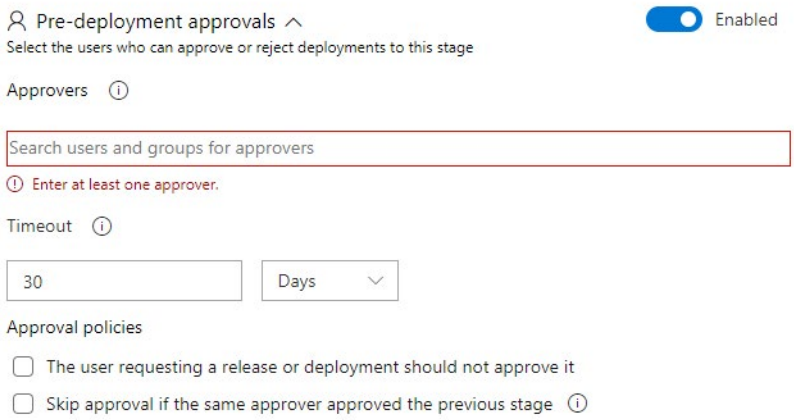
Let's try one.

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

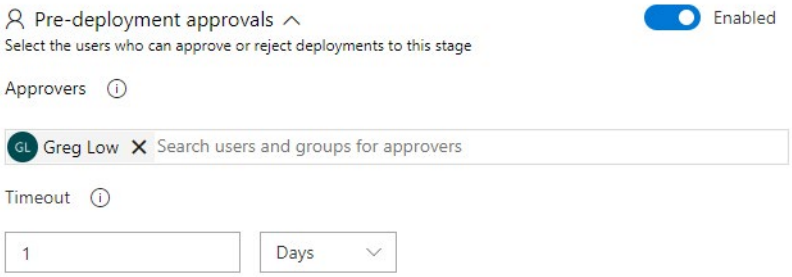
10 https://docs.microsoft.com/en-us/azure/devops/pipelines/release/approvals/approvals?view=vsts
11 https://docs.microsoft.com/en-us/azure/devops/pipelines/release/approvals/approvals?view=vsts
12 https://docs.microsoft.com/en-us/azure/devops/pipelines/release/approvals/gates?view=vsts



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

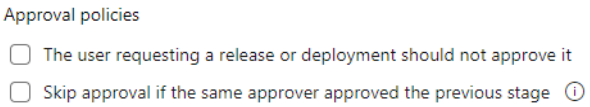


3. In the **Approvers** list, find your own name and select it. Then set the **Timeout** to **1 Days**.



Note: Approvers is a list, not just a single value. If you add more than one person in the list, you can also choose if they need to approve in sequence, or if either or both approvals are needed.

4. Take note of the approver policy options that are available:



It is very common to not allow a user who requests a release or deployment to also approve it. In this case, we are the only approver so we will leave that unchecked.

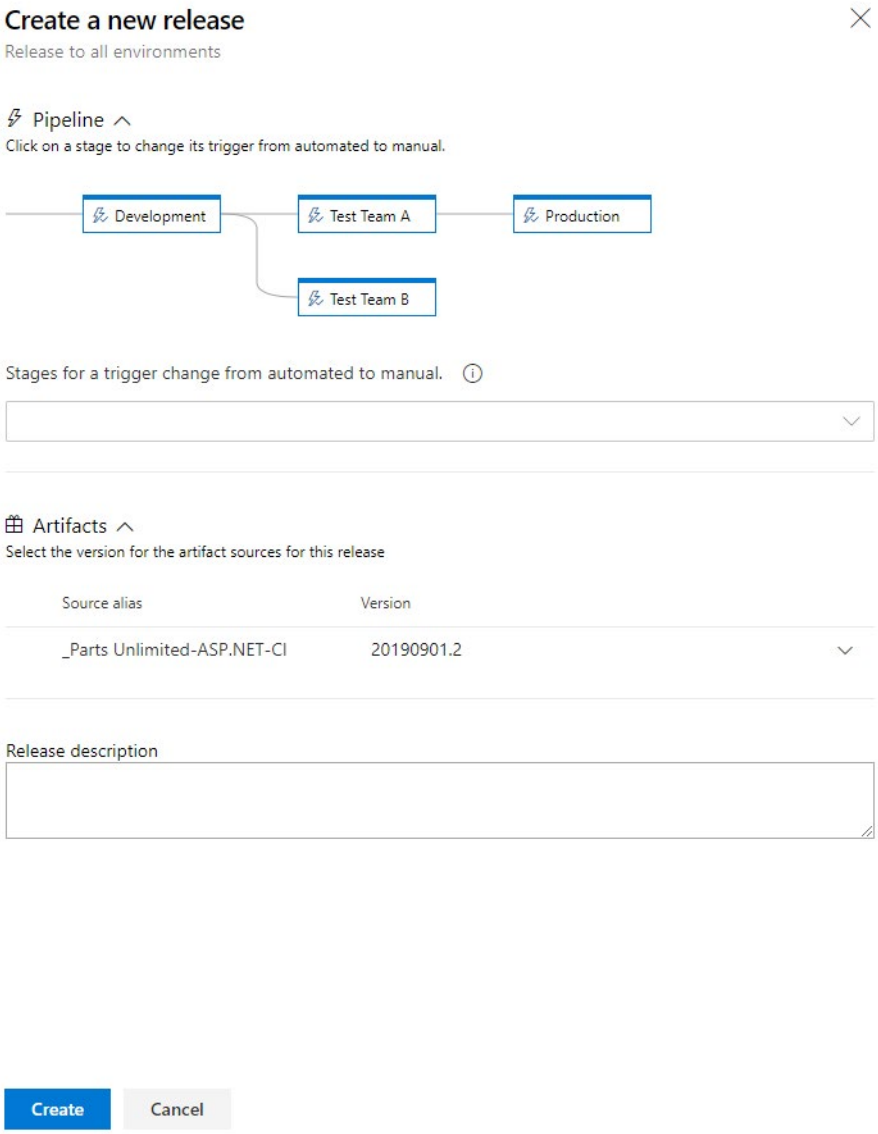
5. Close the Pre-deployment conditions pane and notice that a checkmark has appeared beside the person in the icon.

Test the Approval

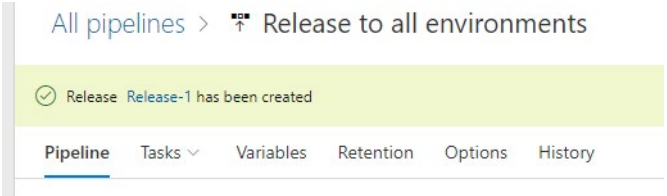
Now it's time to see what happens when an approval is required.

6. Click **Save** to save the work, and **OK** on the popup window.
7. Click **Create release** to start the release process.

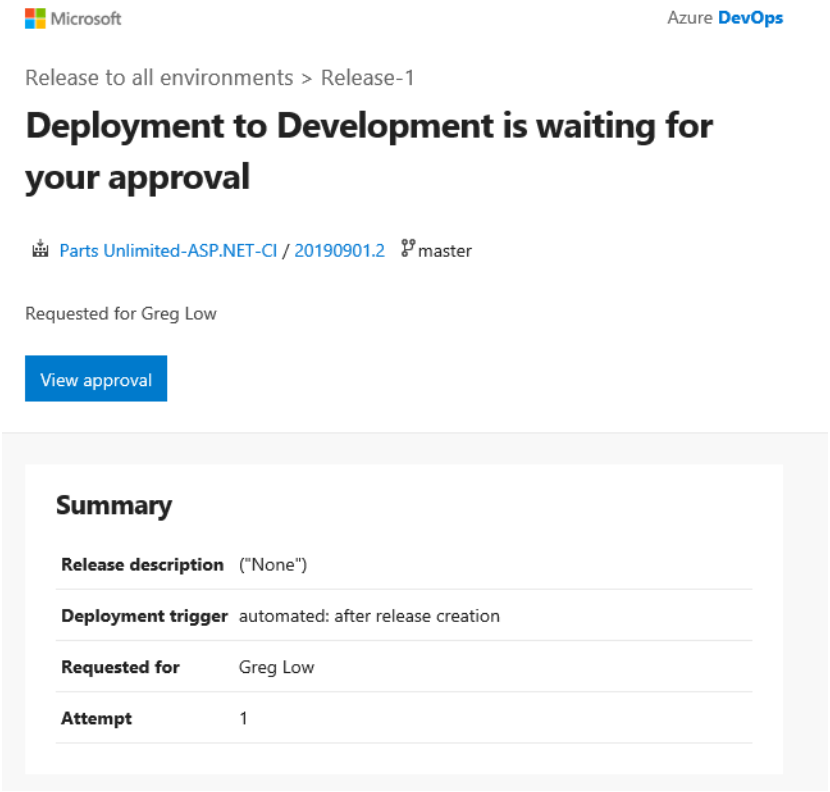
8. In the **Create a new release** pane, note the available options, then click **Create**.



9. In the upper left of the screen, you can see that a release has been created.

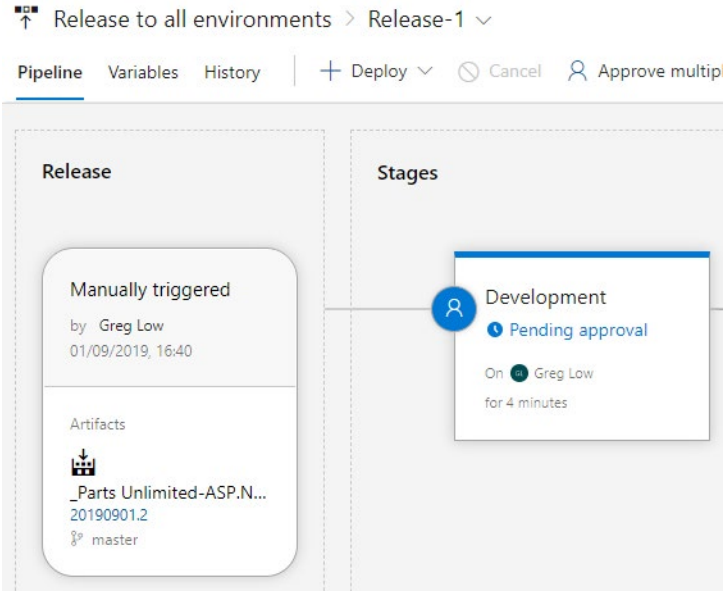


10. At this point, an email should have been received, indicating that an approval is required.



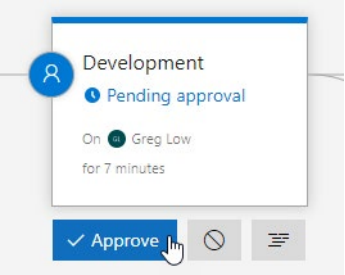
At this point, you could just click the link in the email, but instead, we'll navigate within Azure DevOps to see what's needed.

11. Click on the **Release 1 Created** link (or whatever number it is for you) in the area we looked at in Step 9 above. We are then taken to a screen that shows the status of the release.



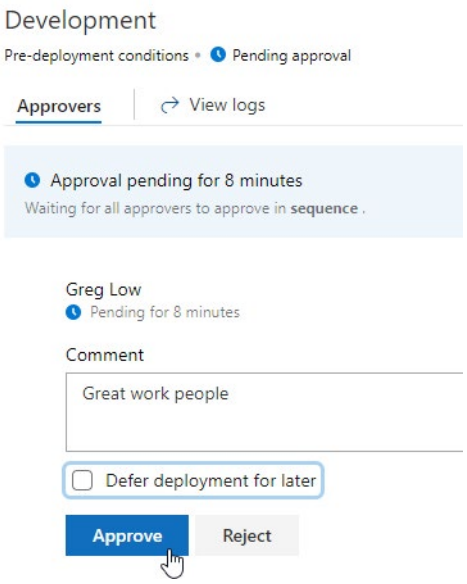
You can see that a release has been manually triggered and that the Development stage is waiting for an approval. As an approver, you can now perform that approval.

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

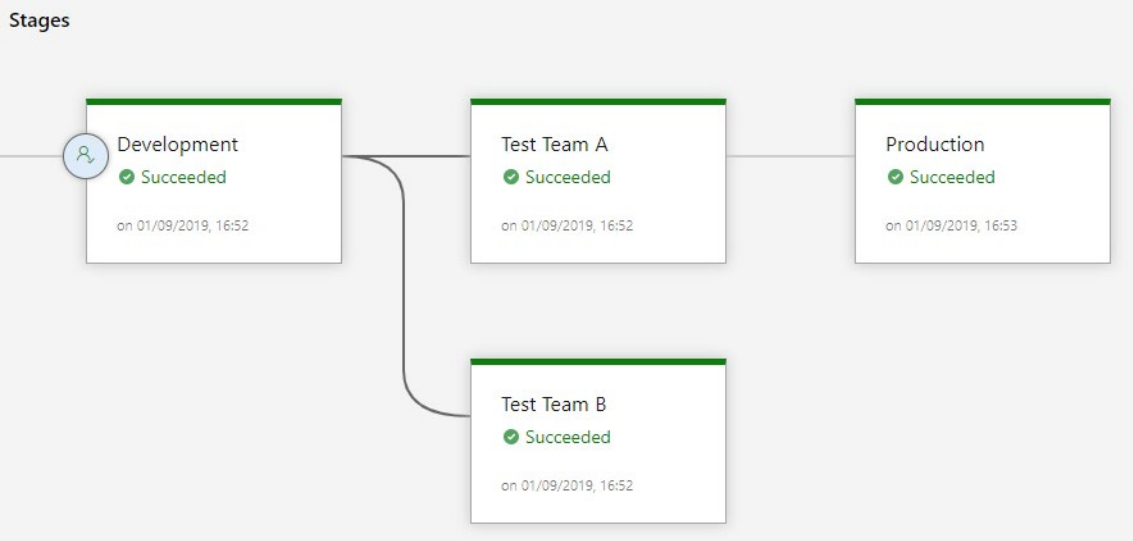


Note: Options to cancel the deployment or to view the logs are also provided at this point

13. In the Development approvals window, add a comment and click **Approve**.



The deployment stage will then continue. Watch as each stage proceeds and succeeds.



Demonstration Setting Up Release Gates

In this demonstration walkthrough, you will investigate Release Gates.

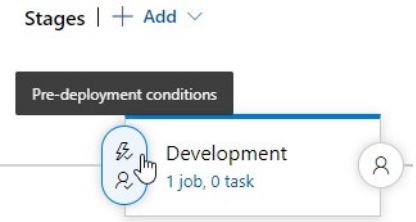
Note: Before starting this walkthrough, make sure you have performed the steps in the prerequisites section and the previous walkthroughs.

Steps

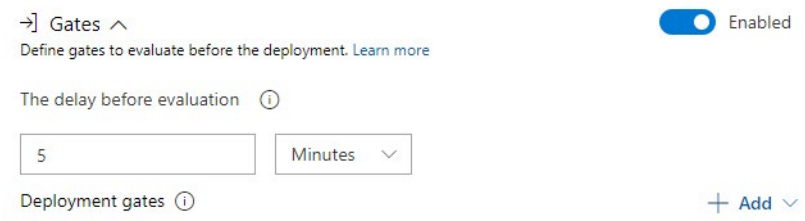
Let's now take a look at when our release pipeline needs to perform automated checks for issues like code quality, before continuing with the deployments. That automated approval phase is achieved by using Release Gates.

Let's take a look at configuring a release gate.

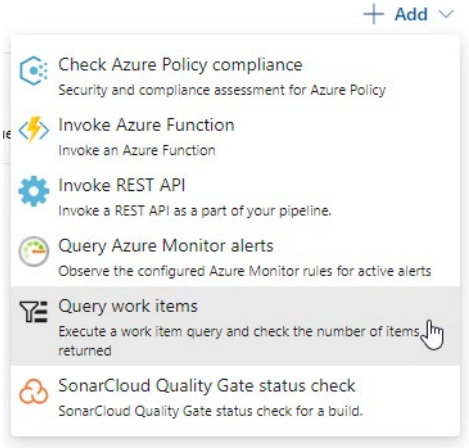
1. Click the lightning icon on the **Development** stage to open the pre-deployment conditions settings.



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

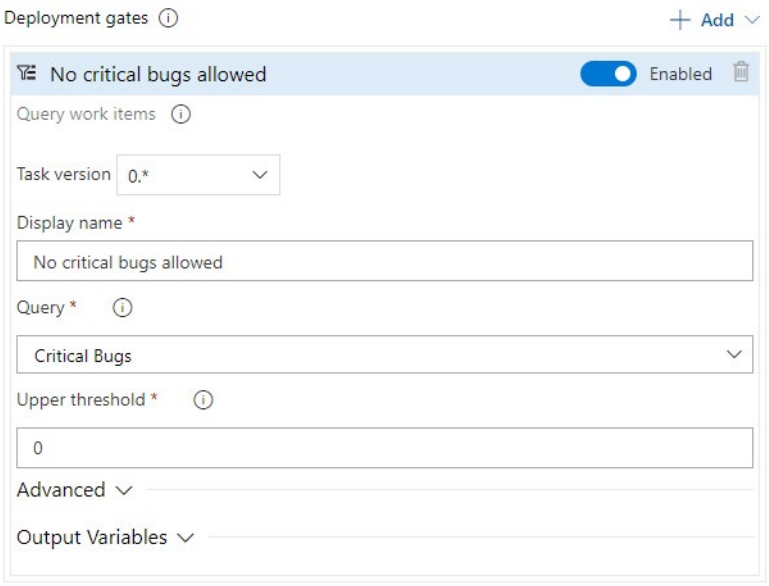


3. Click **+Add** to see the available types of gates, then click **Query work items**.

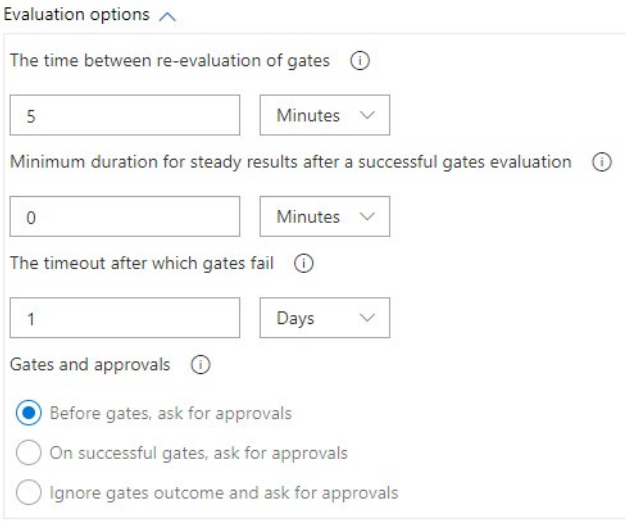


We will use the **Query work items** gate to check if there are any outstanding bugs that need to be dealt with. It does this by running a work item query. This is an example of what is commonly called a **Quality Gate**.

4. Set **Display name** to **No critical bugs allowed**, and from the **Query** drop down list, choose **Critical Bugs**. Leave the **Upper threshold** set to zero because we don't want to allow any bugs at all.



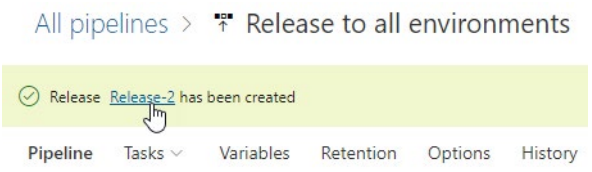
5. Click the drop down beside **Evaluation options** to see what can be configured. While 15 minutes is a reasonable value in production, for our testing, change **The time between re-evaluation of gates** to **5 Minutes**.



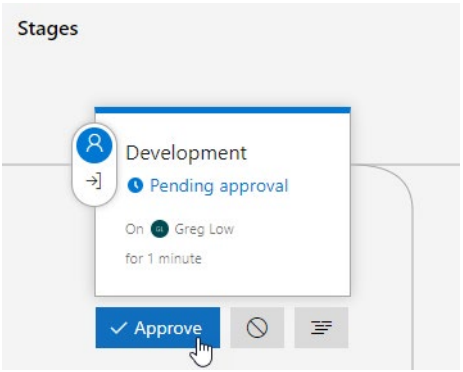
The release gate doesn't just fail or pass a single time. It can keep evaluating the status of the gate. It might fail the first time, but after re-evaluation, it might then pass if the underlying issue has been corrected.

6. Close the pane and click **Save** and **OK** to save the work.
7. Click **Create release** to start a new release, and in the **Create a new release** pane, click **Create**.

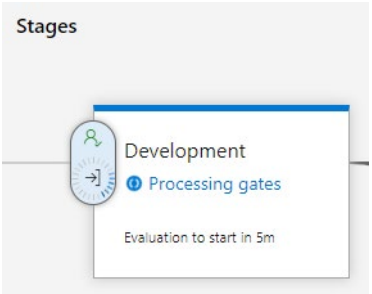
8. Click on the release number to see how the release is proceeding.



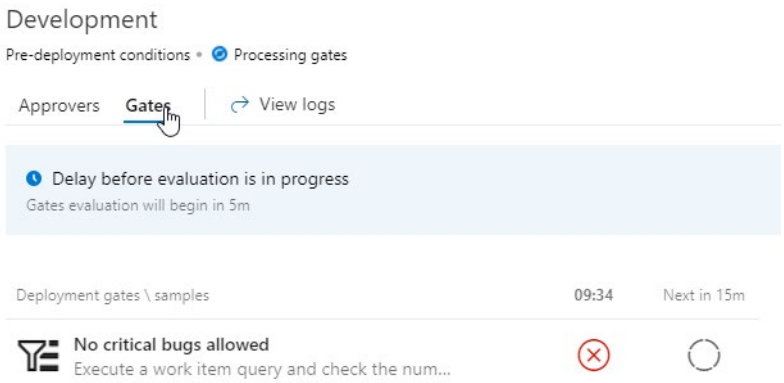
9. If it is waiting for approval, click **Approve** to allow it to continue, and in the **Development** pane, click **Approve**.



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



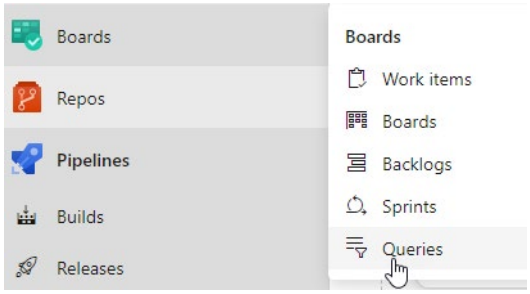
10. In the **Development** pane, click **Gates** to see the status of the release gates.



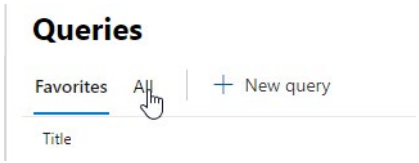
You will notice that the gate failed the first time it was checked. In fact, it will be stuck in the processing gates stage, as there is a critical bug. Let's look at that bug and resolve it.

11. Close the pane and click **Save** then **OK** to save the work.

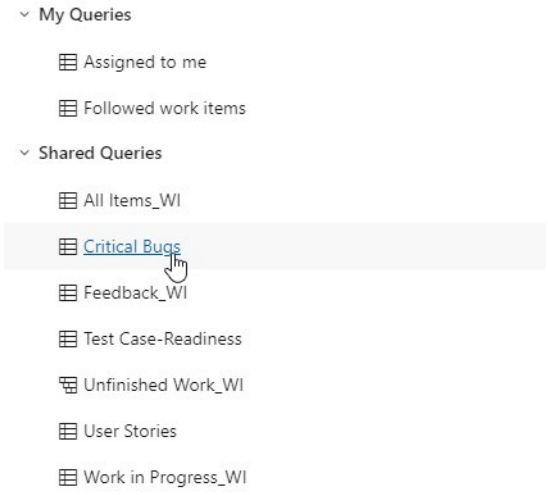
12. In the main menu, click **Boards** then click **Queries**.



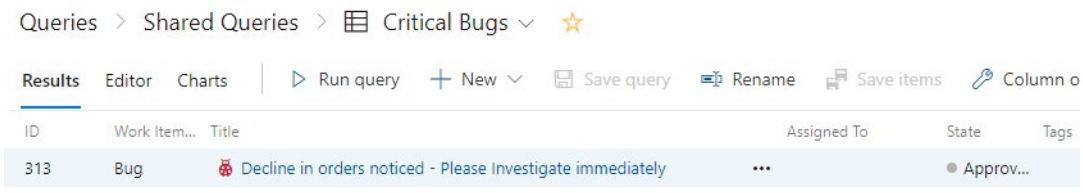
13. In the **Queries** window, click **All** to see all the available queries.



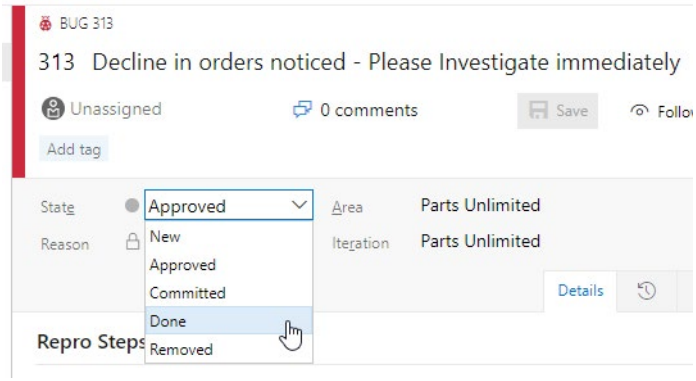
14. From the list of queries, click **Critical bugs**.



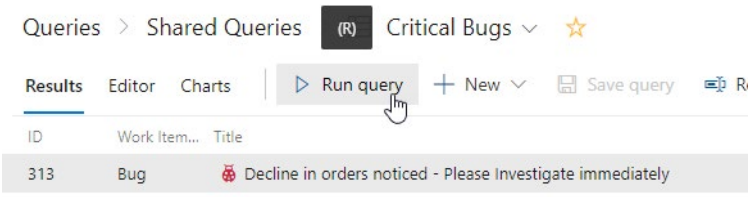
You will see that there is one critical bug that needs to be resolved.



15. In the properties pane for the bug, change the **State** to **Done**, then click **Save**.

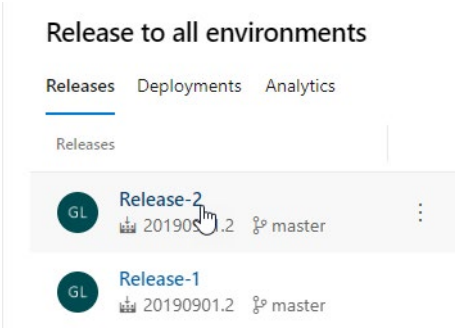


16. Click **Run query** to re-execute the work item query.

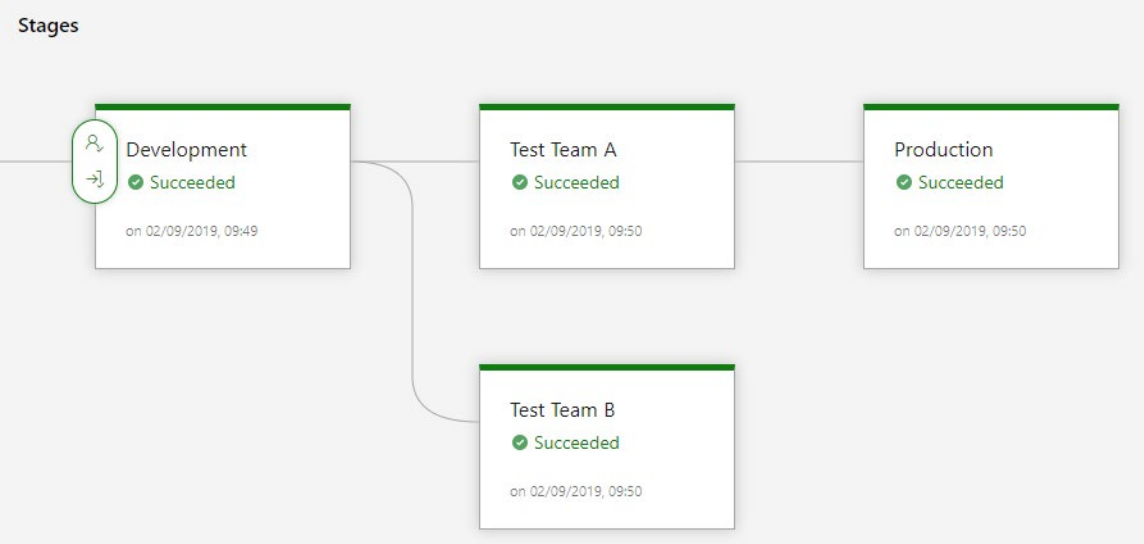


Note that there are now no critical bugs that will stop the release.

17. Return to the release by clicking **Pipelines** then **Releases** in the main menu, then clicking the name of the latest release.



18. When the release gate is checked next time, the release should continue and complete successfully.



Gates are a very powerful automated approval mechanism.

Clean up

To avoid excessive wait time in later walkthroughs, we'll disable the release gates.

19. In the main menu, click **Pipelines**, then click **Releases**, then click **Edit** to open the release pipeline editor.
20. Click the **Pre-deployment conditions** icon (i.e. the lightning bolt) on the **Development** task, and in the **Pre-deployment conditions** pane, click the switch beside **Gates** to disable release gates.
21. Click **Save**, then click **OK**.

Building a High Quality Release Pipeline

Building a High-Quality Release pipeline

Before we dive into high-quality release pipelines, we need to consider the difference between a release and a release process. Or, when you talk about tooling, a release pipeline.

We start with defining a release process or release pipeline. The release pipeline contains all the steps that you walk through when you move your artifact, that comes from one of the artifact sources that we discussed earlier, through the stages or environments.

The stage can be a development stage, a test stage or a production stage or just a stage where a specific user can access the application. We discussed stages earlier in this module. Also part of your pipeline are the people that approve the release or the deployment to a specific stage, the triggers or the schedule on which the releases executes and the release gates, the automatic approvals of the process.

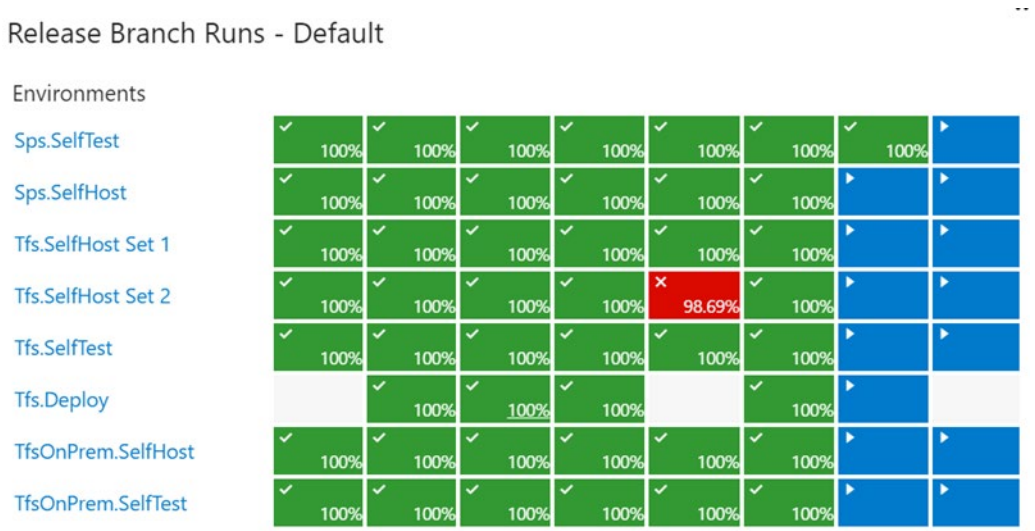
The release itself is something different. The release is an instance of the release pipeline. You can compare it with object inheritance. In Object Orientation, a class contains the blueprint or definition of an object. But the object itself is an instance of that blueprint.

Measuring quality

How do you measure the quality of your release process? The quality of your release process cannot be measured directly because it is a process. What you can measure is how good your process works. If your release process constantly changes, this might be an indication that there is something wrong in the process. If your releases constantly fail, and you constantly have to update your release process to make it work, might also be an indication that something is wrong with your release process.

Maybe something is wrong with the schedule on which your release runs, and you notice that your release always fails at a particular day or at a certain time. Or your release always fails after the deployment to another environment. This might be an indication that some things are maybe dependent or related.

What you can do to keep track of your release process quality, is creating visualisations about the quality of all the releases following that same release process or release pipeline. For example, adding a dashboard widget which shows you the status of every release.



The release also has a quality aspect, but this is tightly related to the quality of the actual deployment and the package that has been deployed.

When we want to measure the quality of a release itself, we can perform all kinds of checks within the pipeline. Of course, you can execute all different types of tests like integration tests, load tests or even you UI tests while running your pipeline and check the quality of the release that you are deploying.

Using a quality gate is also a perfect way to check the quality of your release. There are many different quality gates. For example, a gate that monitors to check if everything is healthy on your deployment targets, work item gates that verify the quality of your requirements process. You can add additional security and compliance checks. For example, do we comply with the 4-eyes principle, or do we have the proper traceability?

Using release gates as quality gate

A quality gate is the best way to enforce a quality policy in your organization. It is there to answer one question: can I deliver my application to production or not?

A quality gate is located before a stage that is dependent on the outcome of a previous stage. In the past, a quality gate was typically something that was monitored by a QA department. They had a number of documents or guidelines, and they verified if the software was of a good enough quality to move on to the next stage. When we think about Continuous Delivery, all manual processes are a potential bottle-neck.

We need to reconsider the notion of quality gates and see how we can automate these checks as part of our release pipeline. By using automatic approval using a release gate, you can automate the approval and validate your company's policy before moving on.

Many quality gates can be considered.

- No new blocker issues
- Code coverage on new code greater than 80%
- No license violations
- No vulnerabilities in dependencies
- No new technical debt introduced

- Compliance checks
    - Are there work items linked to the release?
    - Is the release started by someone else as the code committer?
- Is the performance not affected after a new release?

Defining quality gates make the release process better, and you should always consider adding them.

Release Notes and Documentation

When you deploy a new release to a customer or install new software on your server, and you want to communicate what has been released to your customer, the usual way to do this is the use of release notes.

But where do the release notes come from? There are different ways where you can store your release notes, and I will walk through the various ways in this section of the course.

Document store

An often used way of storing release notes is by creating text files, or documents in some document store. This way, the release notes are stored together with other documents. The downside of this approach is that there is no direct connection between the release in the release management tool and the release notes that belong to this release.

Wiki

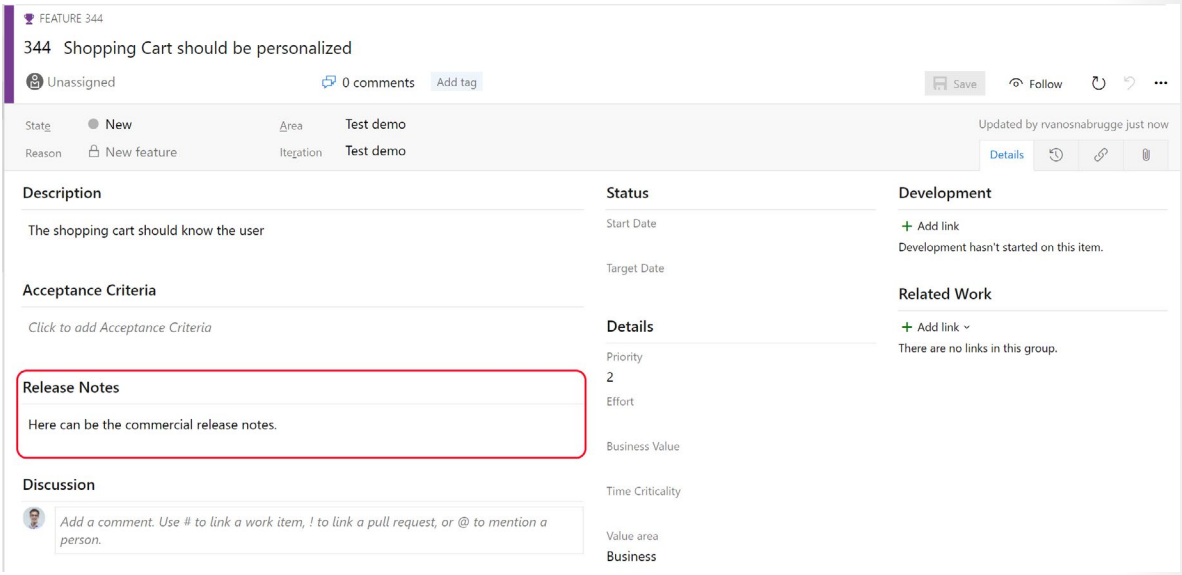
The most used way that is used at customers is to store the release notes in a Wiki. For example, Confluence from Atlassian, SharePoint Wiki, SlimWiki or the Wiki in Azure DevOps.
The release notes are created as a page in the wiki, and by using hyperlinks, relations can be associated with the build, the release and the artifacts.

In the code base

When you look at it, release notes belong strictly to the release. To the features you implemented and the code you wrote. In that case, the best option might be to store release notes as part of your code repository. The moment, the team completes a feature, they or the product owner also write the release notes and saves these alongside the code. This way it becomes living documentation because the notes change with the rest of the code.

In a work item

Another option is to store your release notes as part of your work items. Work items can be Bugs, Tasks, Product Backlog Items or User Stories. To save release notes in work items, you can create or use a separate field within the work item. In this field, you type the publicly available release notes that will be communicated to the customer. With a script or specific task in your build and release pipeline, you can then generate the release notes and store them as an artifact or publish them to an internal or external website.



- **Generate Release Notes Build Task**[13]
- **[WIKI Updater Tasks**[14]
- **Atlassian Confluence**[15]
- **Azure DevOps Wiki**[16]

There is a difference between functional and technical documentation. There is also a difference between documentation designing the product, mostly written up front and documentation describing the product afterwards, like manuals or help files. Storing technical documentation about your products, that is still in the design phase, is usually done on a document sharing portal, like SharePoint or Confluence.

A better and more modern way to store your documentation is to create a wiki. Wiki's do not contain Documents, Presentations or Spreadsheets but text files called Markdown Files. These markdowns can refer to pictures, can hold code samples and can be part of your code repository. Code repositories can deal very well with text files. Changes and history can be easily tracked by using the native code tools.

However, the most significant advantage of using a Wiki instead of documents is that a Wiki is accessible for everyone in your team. By giving the right permissions to all the team members, people can work together on the documentation instead of having to wait for each other when working on the same document.

Manuals or documentation that you release together with the product, should be treated as source code. When the product changes and new functionality is added, the documentation needs to change as well. You can store the documentation as part of your code repository, or you can create a new repository containing your documentation. In any case, the documentation should be treated the same way as your source code. Create a documentation artifact in your build pipeline, and deliver this artifact to the release pipeline. The release pipeline can then deploy the documentation to a site or just include in in the boxed product.

13 https://marketplace.visualstudio.com/items?itemName=richardfennellBM.BM-VSTS-XplatGenerateReleaseNotes
14 https://marketplace.visualstudio.com/items?itemName=richardfennellBM.BM-VSTS-WIKIUpdater-Tasks
15 https://www.atlassian.com/software/confluence
16 https://azure.microsoft.com/en-us/services/devops/wiki/

Choosing a Deployment Pattern

Choosing a deployment pattern

A deployment pattern is a way of how you choose to move your application to production. Traditionally the way to do it was to set up a *Development* environment, a *Test* environment and a *Production* environment and perform more or less the same steps on each environment. Nowadays we can have Infrastructure on-demand, and this opens up a new range of possibilities when it comes to deployment patterns. We are not limited to a fixed set of environments but have various options.

When we move towards Continuous Delivery, we also need to consider that deploying multiple times a day, can impact our users. They do not want to be surprised with a changed UI after a refresh or 5 minutes of downtime during a transaction. Modern deployment patterns built upon software architecture patterns. A good example is the use of Feature Toggles that enable you to hide specific functionality.

In this chapter, we will briefly discuss some deployment Patterns. In Module 3 we will cover them in more details, together with some demos and HandsOn exercises. The most important thing to realise is that not every application can follow these deployment patterns without changing the application and architecture.

Traditional Deployment Pattern

Dev, QA, Prod - As new builds are produced, they can be deployed to Dev. They can then be promoted to QA, and finally to Prod. At any time, each of these stages may have a different release (set of build artifacts) deployed to them. This is an excellent example of the use of stages in a release pipeline.

Modern Deployment Patterns

Blue-Green Deployment
The Blue-Green deployment is all about ensuring you have two production environments, as identical as possible. After deployment and tests, the environments are swapped.

Canary Release
A canary release is only pushed to a small number of users. Therefore its impact is relatively small should the new code prove to be buggy. Changes can be reversed quickly.

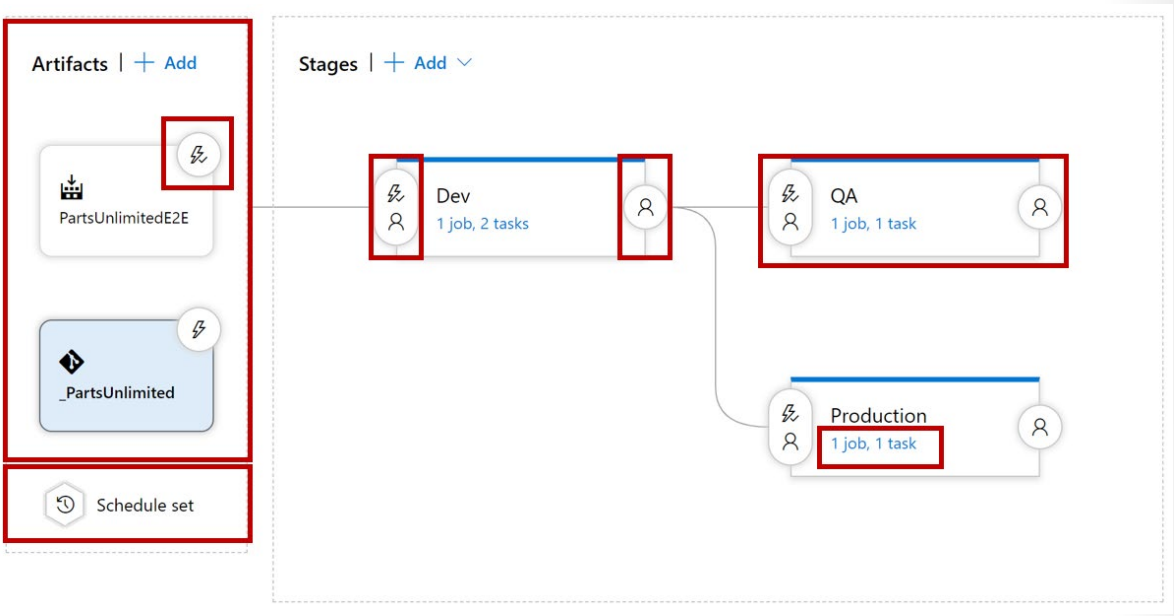
Dark Launching
Dark launching is the process of releasing production-ready features to a subset of your users before a full release. This enables you to decouple deployment from release, get real user feedback, test for bugs, and assess infrastructure performance.

Progressive Exposure Deployment
In a progressive exposure deployment, you expose the new software to a subset of users, that you extend gradually over time. Impact (also called blast radius), is evaluated through observation, testing, analysis of telemetry, and user feedback.

Choosing the Right Release Management Tool

Overview Release Management Tools

As discussed in an earlier chapter, a release pipeline consists of different components.



When choosing the right Release Management tool, you should look at the possibilities of all the different components and map them to the needs you have. There are many tools available in the marketplace from which we will discuss some in the next chapter. The most important thing to notice is that not every vendor or tool treats Release Management in the same manner.

The tools in the marketplace can be divided into 2 categories

- Tools that can do Build and Continuous Integration and Deployment
- Tools that can do Release Management

In many cases, companies only require the deployment part of Release Management. Deployment, or installing software can be done by all the build tools out there. Primarily because the technical part of the release is executing a script or running a program. Release Management that requires approvals, quality gates, and different stages, needs a different kind of tool that usually tightly integrate with the build and CI tools but are not the same thing.

In the previous chapters we have discussed all the components that are part of the release pipeline, and here we will briefly highlight the things you need to consider when choosing a Release Management Tool.

Artifacts and Artifact source

As we have seen in the previous chapter, artifacts can come from different sources. When you want to treat your artifact as a versioned package, it needs to be stored somewhere before your release pipeline consumes it. Considerations for choosing your tool can be:

- Which Source Control systems are supported?
- Can you have 1 or more artifact sources in you release pipeline? In other words, can you combine artifacts from different sources into one release?

- Does it integrate with your build server?
- Does it support other build servers?
- Does it support container registries?
- How do you secure the connection to the artifact source?
- Can you extend the artifact sources?

Triggers and Schedules

Triggers are an essential component in the pipeline. Triggers are required to start a release but, if you want to have multiple stages, also to start a deployment
Considerations for choosing your trigger can be:

- Does your system support Continuous Deployment triggers.
- Can you trigger releases from an API (for integration with other tools)
- Can you schedule releases
- Can you schedule and trigger each stage separately?

Approvals and gates

Starting a release and using scripts, executables or deployable artifacts does not make the difference between a Continuous Integration/Build tool and a Release Management tool. Adding a release approval workflow to the pipeline is the critical component that does make the difference.
Considerations When it comes to approvals:

- Do you need approvals for your release pipeline?
- Are the approvers part of your company? Do they need a tool license?
- Do you want to use manual or automatic approvals? Or both?
- Can you approve with an API (integration with other tools)
- Can you set up a workflow with approvers (optional and required)?
- Can you have different approvers per stage?
- Can you have more than one approver? Do you need them all to approve?
- What are the possibilities for automatic approval?
- Can you have a manual approval or step in your release pipeline?

Stages

Running a Continuous Integration pipeline that build and deploys your product is a very commonly used scenario.
But what if you want to deploy the same release to different environments? When choosing the right release management tool, you should consider the following things when it comes to stages (or environments)

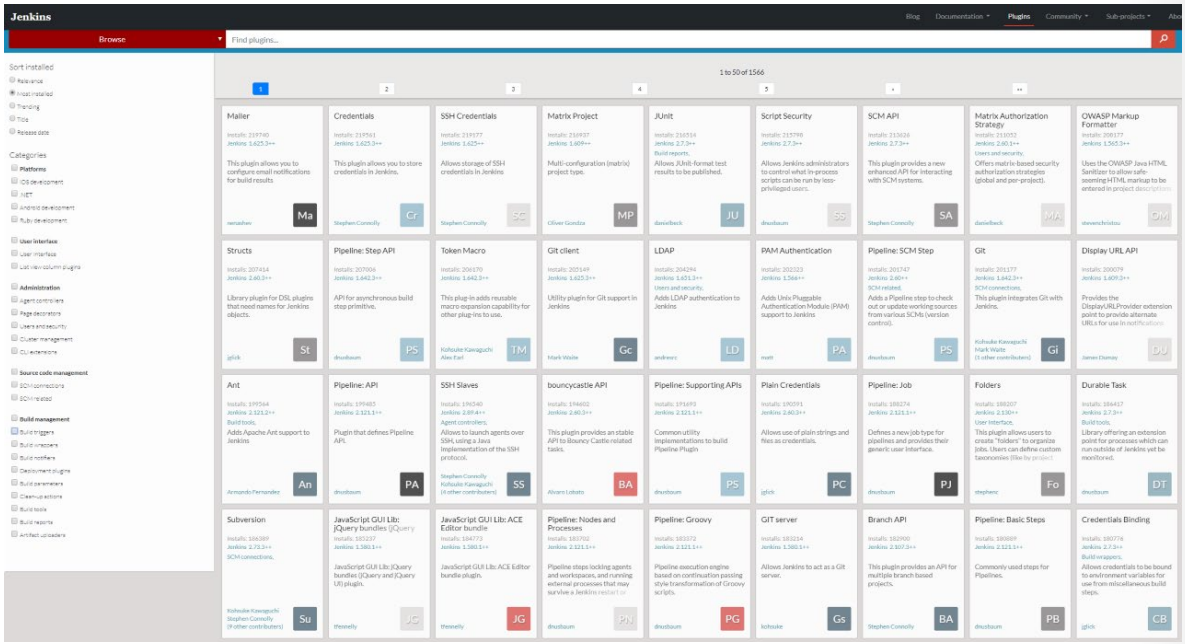
- Can you use the same artifact to deploy to different stages?
- Can you differ the configuration between the stages
- Can you have different steps for each stage?

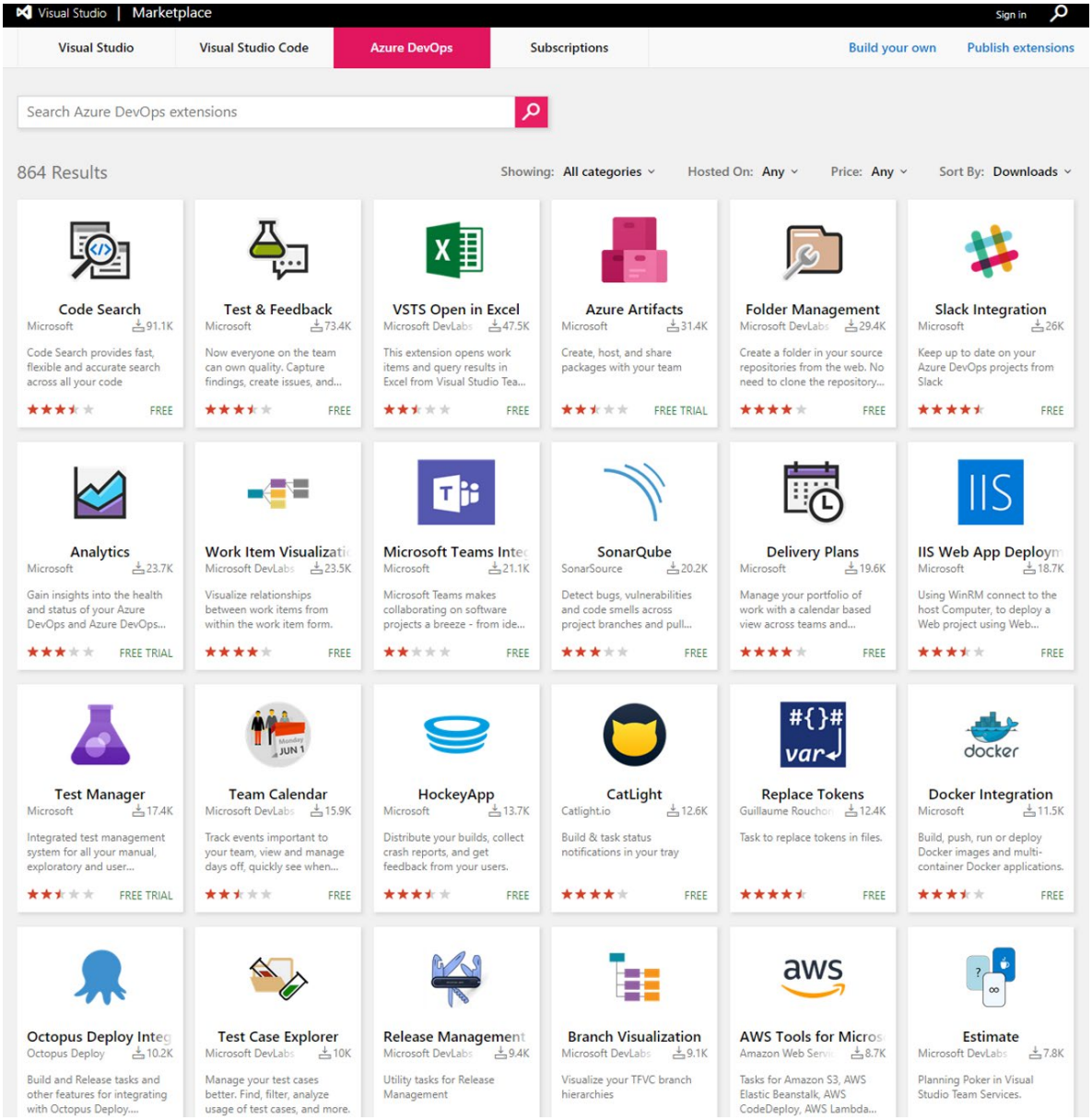
- Can you follow the release between the stages?
- Can you track the artifacts / work items and source code between the stages?

Build and Release Tasks

Last but not least the work needs to be done within the pipeline. It is not only about the workflow and orchestration, the code needs to be deployed or installed. Things to consider when it comes to the execution of tasks

- How do you create your steps? Is it running a script (bat, shell, PowerShell cli) or are there specialised tasks?
- Can you create your own tasks?
- How do tasks authenticate to secure sources?
- Can tasks run on multiple platforms?
- Can tasks be easily reused
- Can you integrate with multiple environments? (Linux, Windows, Container Clusters, PaaS, CLoud)
- Can you control the tasks that are used in the pipeline





Traceability, Auditability and Security

One of the most essential things in enterprises and companies that need to adhere to compliance frameworks is Traceability, Auditability and Security. Although this is not explicitly related to a release pipeline, it is an important part.

When it comes to compliance 3 principles are fundamental:

- 4-eyes principle
    - Is the deployed artifact reviewed by at least one other person.
    - Is the person that deploys another person as the one that writes the code

- Traceability
    - Can we see where the released software originates from (which code)
    - Can we see the requirements that led to this change
    - Can we follow the requirements through the code, build and release
- Auditablity
    - Can we see who, when and why the release process changed
    - Can we see who, when and why a new release has been deployed

Security is vital in this. When people can do everything, including deleting evidence, this is not ok. Setting up the right roles, permissions and authorisation are important to protect your system and your pipeline.

When looking at an appropriate Release Management tool, you can consider

- Does it integrate with your company's Active Directory?
- Can you set up roles and permissions?
- Is there change history of the release pipeline itself?
- Can you ensure the artifact did not change during the release?
- Can you link requirements to the release?
- Can you link source code changes to the release pipeline?
- Can you enforce approval or 4-eyes principle?
- Can you see release history and the people who triggered the release?

Release Management Tools Comparison

The following tools are mainstream in the current ecosystem. You will find links to the product websites where you can explore the product and see if it fits the needs as we described in the previous chapter.

- What Artifacts and Artifact source does the tool support
- What Triggers and Schedules?
- Does the tool support Approvals and gates
- Can you define multiple stages?
- How does the Build and Release Tasks work?
- How does the tool deal with Traceability, Auditability and Security
- What is the Extensibility model?

Per tool is indicated if it is part of a bigger suite. Integration with a bigger suite gives you a lot of advantages regarding traceability, security and auditability. A lot of integration is already there out of the box.

Jenkins

The leading open source automation server, Jenkins provides hundreds of plugins to support building, deploying and automating any project.

- On-prem system. Offered as SaaS by third-party

- No part of a bigger suite
- Industry standard, especially in the full stack space
- Integrates with almost every source control system
- Rich ecosystem of plugins
- CI/Build tool with deployment possibilities.
- No release management capabilities

Links

- **https://jenkins.io/**
- **https://cloudblogs.microsoft.com/opensource/2018/09/21/configure-jenkins-cicd-pipeline-deploy-asp-net-core-application/**
- **Jenkins in action with Azure Service Fabric**[17]

Circle CI

CircleCI's continuous integration and delivery platform help software teams rapidly release code with confidence by automating the build, test, and deploy process. CircleCI offers a modern software development platform that lets teams ramp quickly, scale easily, and build confidently every day.

- CircleCI is a cloud-based system or an on-prem system
- Rest API — you have access to projects, build and artifacts
- The result of the build is going to be an artifact.
- Integration with GitHub and BitBucket
- Integrates with various clouds
- Not part of a bigger suite
- Not fully customizable

Links

- **https://circleci.com/**
- **How to get started on CircleCI 2.0: CircleCI 2.0 Demo**[18]

Azure DevOps Pipelines

Azure Pipelines helps you implement a build, test, and deployment pipeline for any app. Tutorials, references, and other documentation show you how to configure and manage continuous integration and Continuous Delivery (CI/CD) for the app and platform of your choice.

- Hosted on Azure as a SaaS in multiple regions and available as an on-prem product
- Complete Rest API for everything around Build and Release Management

17 https://www.youtube.com/watch?v=5RYmooIZqS4
18 https://www.youtube.com/watch?v=KhjwnTD4oec

- Integration with many build and source control systems (Github, Jenkins, Azure Repos, Bitbucket, Team Foundation Version Control, etc.)
- Cross Platform support, all languages and platforms
- Rich marketplace with extra plugins, build tasks and release tasks and dashboard widgets
- Part of the Azure DevOps suite. Tightly integrated
- Fully customizable
- Manual approvals and Release Quality Gates supported
- Integrated with (Azure) Active Directory
- Extensive roles and permissions

Links

- **https://azure.microsoft.com/en-us/services/devops/pipelines/**
- **Building and Deploying your Code with Azure Pipelines**[19]

GitLab Pipelines

GitLab helps teams automate the release and delivery of their applications to enable them to shorten the delivery lifecycle, streamline manual processes and accelerate team velocity. With Continuous Delivery (CD), built into the pipeline, deployment can be automated to multiple environments like staging and production, and support advanced features such as canary deployments. Because the configuration and definition of the application are version controlled and managed, it is easy to configure and deploy your application on demand.

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

Atlassian Bamboo

Bamboo is a continuous integration (CI) server that can be used to automate the release management for a software application, creating a Continuous Delivery pipeline.

https://www.atlassian.com/software/bamboo/features

XL Deploy/XL Release

XL Release is an end-to-end pipeline orchestration tool for Continuous Delivery and DevOps teams. It handles automated tasks, manual tasks, and complex dependencies and release trains. And XL Release is designed to integrate with your change and release management tools.

https://xebialabs.com/products/xl-release/

19 https://www.youtube.com/watch?v=NuYDAs3kNV8

Review Questions

Module 1 Review Questions

Cyclomatic Complexlity

Would adding a feature flag increase or decrease the cyclomatic complexity of the code?

Suggested Answer
Increase

Technical Debt

Would adding a feature flag increase or decrease technical debt?

Suggested Answer
Increase

Deployment Pattern

You plan to slowly increase the traffic to a newer version of your site. What type of deployment pattern is this?

Suggested Answer
Blue-green

Immutable Objects

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

Suggested Answer
You make a new one and (possibly) remove the old one

Compliance

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

Suggested Answer
Release gate

Lab Building a Release Strategy

Lab: Building a release strategy

In this exercise, you need to design a release strategy for a PartsUnlimited. Based on the concepts you learned, your assignment is to draw a picture or design a release pipeline in Azure DevOps where you think about the following things.

- Choosing the rights artifact source
- Which deployment stages do you choose
- Release triggers and deployment triggers
- Manual Approvals and/or release gates
- Applicable deployment pattern(s)

Document your solution and arguments.

Scenario

PartsUnlimited wants to launch a new Greenfield website to their customers. They have a group of customers that is called “the inner circle.” They have been customers for a long time and had a trust relationship with the management of PartsUnlimited. In close collaboration, these customers have been involved as a stakeholder in the development of the new website.

There are two development teams involved in this new website. There is a backend system, that is exposed by a REST API. Development team “Formula 1” develops this backend and uses Azure DevOps to store the sources and work items. An Angular Website consumes the REST API. This is developed by the "The Wheelies." “The Wheelies” use Jenkins as their build server and store their sources in Github.

A dedicated group of “customers” is already using the software as a very early adopter. The CEO wants to expand the user base to "the inner circle gradually", and when this all goes well, to the broader public. Because of architectural constraints, there still is downtime, when a new version of the backend is deployed. This will be solved in a later iteration. Customers only use the system during the day and should not be impacted by a system change. The “testers” and "inner circle" do not mind about the downtime, only the end-users.

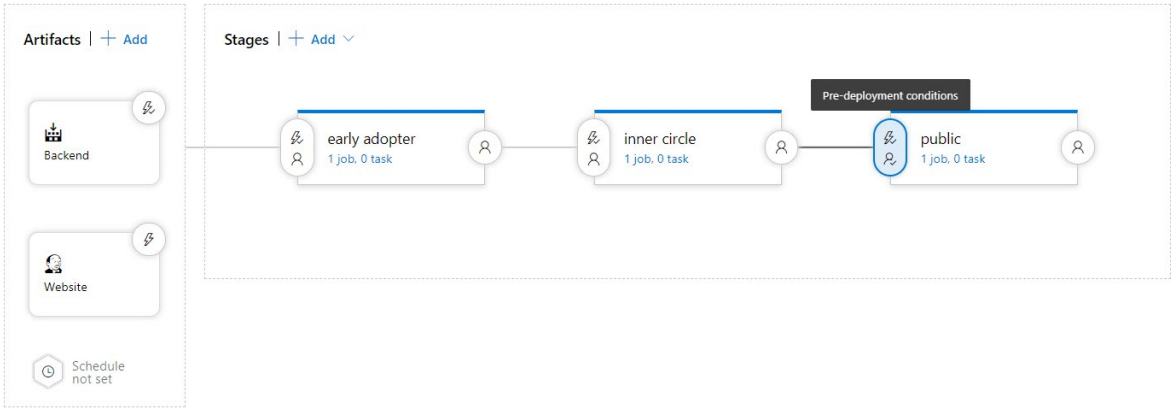
The Product Owner, John, wants to approve the release of a new version until he trusts that everything deploys without errors. Tim, the system administrator wants to have a solution to the “maintenance” problem. He wants to be sure that no new releases are deployed when he is working on the system. He proposes a weekly planning session where a schedule is made, but he is open for an alternative solution.

Lab Solution - Building a release strategy

Possible solution - Building a release strategy

As in every situation, there are different approaches and solutions, and everything is good, as long as the thoughts behind it are good.

This proposed solution is one of the possibilities



Artifact Sources

The artifacts come from an automated build system. We choose to have one release pipeline that combines the artifacts from Azure DevOps Build and Jenkins. This way the two development teams can stick to their own tool stack

Deployment Stages

We have set up 3 stages. Early adopters, Inner circle (for the trusted users), and public.

Cadence

We have set up a continuous deployment cadence for the release pipeline. Every new artifact from Jenkins and the Azure DevOps build will trigger a new release. The release is then automatically deployed to the early adopters. That night, the same release is deployed to the inner circle. This is done with a schedule. The next day, the deployment to public is initiated. This starts when the product owner John approves it.

Approvals and Gates

When a new deployment is triggered to the public stage, John needs to approve, and when he approves the release it is scheduled for the night. This is to ensure there is no downtime. There is also a release gate. This gate checks the status of the maintenance window. Tim, the system admin, has a UI where he can “stop” releases. The release gate monitors this switch.

Deployment patterns

The pattern we follow here is a Progressive Exposure Deployment. We expose new functionality to new groups to reduce impact.

Module 2 Set Up a Release Managment Workflow

Create a Release Pipeline

Set up a Release Management Workflow

Introduction to this module

As you might have discovered in the first module, Continuous Delivery is a concept containing much more than a release pipeline. For the sake of this course, we will use the release pipeline as a mechanism to implement a significant part of Continuous Delivery. In the first module, we covered strategies for designing a release pipeline, and we talked about many of the concepts that are shared amongst all the different CI/CD and release management tools that are available in the marketplace.

Continuous Delivery is much more about enabling teams within your organization. Enable them to deliver the software on demand. Making it possible that you can press a button at any time of the day, and still have a good product means a number of things. It says that the code needs to be high quality, the build needs to be fully automated and tested, and the deployment of the software needs to be fully automated and tested as well.

We covered the basics in module 1. Now we need to dive a little bit further into the release management tooling. We will include a lot of things coming from Azure pipelines. A part of the Azure DevOps suite. Azure DevOps is an integrated solution for implementing DevOps and Continuous Delivery in your organization. We will cover some specifics of Azure pipelines, but this does not mean they do not apply for other products available in the marketplace. Many of the other tools share the same concepts and only differ in naming.

Release pipelines

A release pipeline, in its simplest form, is nothing more than the execution of a number of steps. In this module, we will dive a little bit further into the details of one specific stage. The steps that need to be executed and the mechanism that you need to execute the steps within the pipeline.

In this module, we will talk about agent and agent pools that you might need to execute your release pipeline. We will look at variables for the release pipeline and the various stages.

After that, we dive into the tasks that you can use to execute your deployment. Do you want to use script files or do you want to use specific tasks that can perform one job outstanding? For example, the marketplaces of both Azure DevOps and Jenkins have a lot of tasks in the store that you can use to make your life a lot easier.

We will talk about secrets and secret management in your pipeline. A fundamental part to secure your not only your assets but also the process of releasing your software. At the end of the module, we will talk about alerting mechanisms. How to report on your software, how to report on your quality and how to get notified by using service hooks. Finally, we will dive a little bit further into automatic approvals using automated release gates.

Learning objectives

At the end of this module:

- You are familiar with the terminology used in Azure DevOps and other Release Management Tooling
- You know what a Build and Release task is, what it can do, and are familiar with some available deployment tasks
- You know what an Agent, Agent Queue and Agent Pool is and what an agent is all about
- You know what a release job is, and why you sometimes need multiple release jobs in one release pipeline
- You know the difference between multi-agent and multi-configuration release job
- You know what release variables are, what stage variables are and how to best use them in your release pipeline
- You know what a Service Connection is and how you can use them to deploy to an environment securely
- You know what the possibilities for testing are in the pipeline and how to embed testing in the pipeline
- You are familiar with the different ways to inspect the health of your pipeline and release by using, alerts, service hooks and reports
- You know what a release gate is and how to create one

Definitions and Glossary

In the previous module, we have covered the concepts of a release pipeline. Many terms and definitions are used. The hardest part is that these terms vary from tool to tool. In this part, you find a list of names and definitions and synonyms.

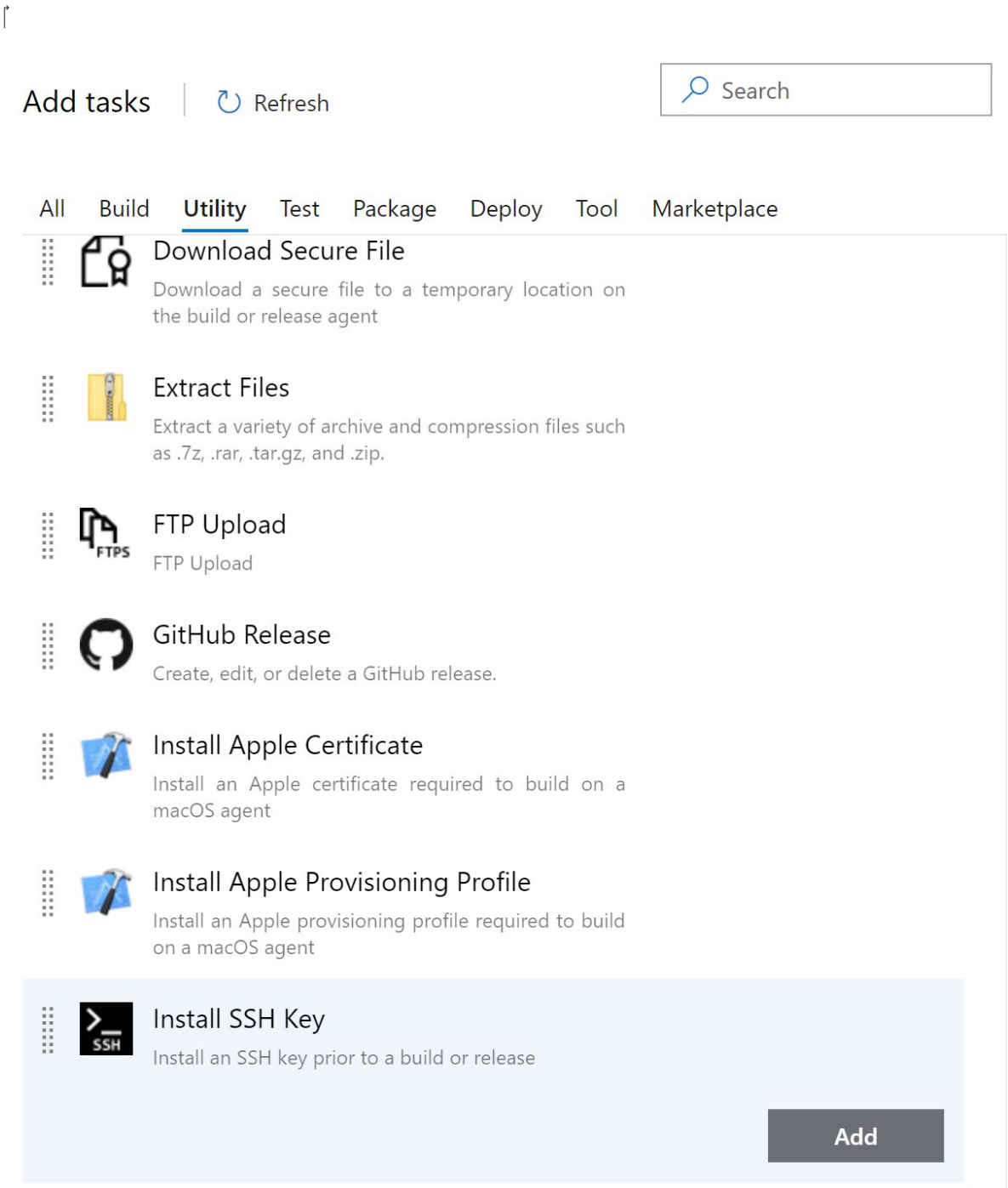
In this list, you find the term that is used in Azure DevOps.

Term	Description	Synonym
Stage	an isolated and independent target for deployment	Environment

Term	Description	Synonym
Job	A phase in the release pipeline that can run simultaneously with other phases on different Operating Systems	Phases
Agent	The program that runs the build or release	
Build & Release Task	Tasks are units of executable code used to perform designated actions in a specified order.	Action, Plugin, App
Release pipeline	The process that runs when deploying an artifact. Including triggers, approvals and gates	release process, pipeline, release definition
CI/CD	Continuous Integration / Continuous Deployment	
Release gate	An automated check that approves the continuation	Quality Gate, Automatic Approval
Service Connection	A secure connection to an environment or service	Service Endpoint

Build and Release Tasks

A build and release platform requires the ability to execute any number of repeatable actions during the build process. Tasks are units of executable code used to perform designated actions in a specified order.



Add steps to specify what you want to build, the tests that you want to run, and all of the other steps needed to complete the build process. There are steps for building, testing, running utilities, packaging, and deploying.

If a task is not available, you can find a lot of community tasks in the marketplace. Jenkins, Azure DevOps and Atlassian have an extensive marketplace where additional tasks can be found.

More information regarding Build and Release tasks in Azure DevOps can be **found here**[1]

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

Links

- **Out of the box tasks in Azure DevOps**[2]
- **Bamboo marketplace**[3]
- **Jenkins Marketplace**[4]
- **Azure DevOps Marketplace**[5]

Important Deployment Tasks

The choice of deployment tasks depends on the target environment to which you want to deploy your software.

If you want to deploy a mobile app, you can create a package and upload it with a script or use a task that directly deploys to the Apple Store or Google Play store.

If you want to deploy to a server running on premises, then you might want to use a PowerShell or command line script to install the software. Or, if you're going to use a specialised task, you can use the web deployment task, that deploys MS deploy packages directly on an IIS

You can also use an SSH Task and use, Shell tasks to execute the necessary actions on the target server.

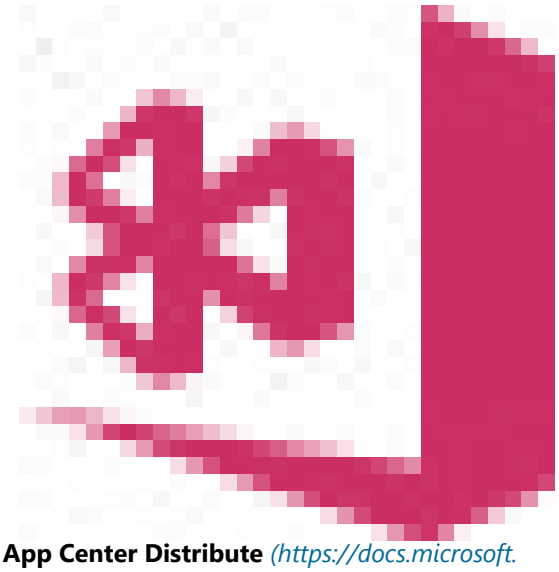
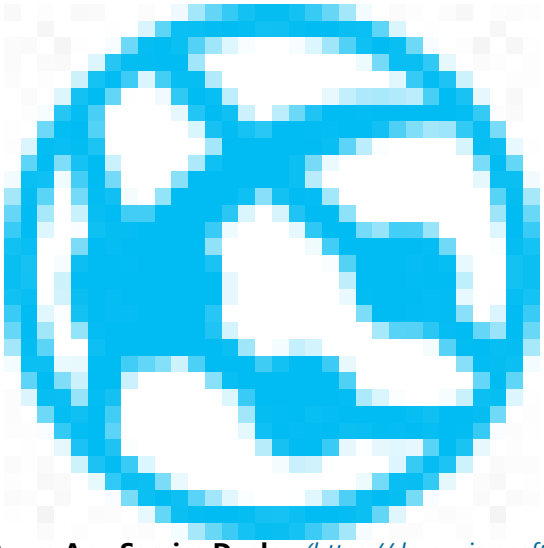
When you want to deploy to Azure or another cloud, you can use command line tools specialised for that cloud, or some specialised tasks that you can use to install your resources.

When you want to deploy to a container cluster, you can run a docker command in a command line task, or you can use specific docker tasks. They do a lot of the plumbing and secure connections to the container cluster and container registry.

In the following section, a list of out of the box tasks is listed so you can see what is available for you to deploy. Regardless of the environment that you want to target.

2 https://github.com/Microsoft/vsts-tasks
3 https://marketplace.atlassian.com/addons/app/bamboo/trending
4 https://plugins.jenkins.io/
5 https://marketplace.visualstudio.com/

Deploy

	Task
 **App Center Distribute** *(https://docs.microsoft.com/en-us/appcenter/distribution/)*	Distribute app builds to testers and users via App Center
 **Azure App Service Deploy** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-rm-web-app-deployment?view=vsts)*	Update Azure App Service using Web Deploy / Kudu REST APIs

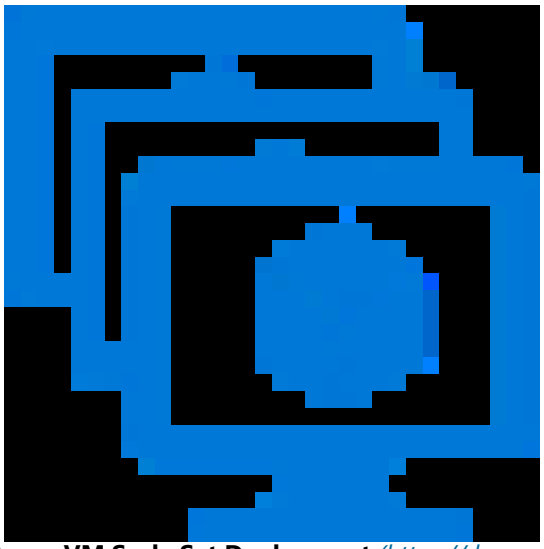
	Task
 **Azure App Service Manage** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-app-service-manage?view=vsts)*	Start, Stop, Restart or Slot swap for an Azure App Service
 **Azure CLI** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-cli?view=vsts)*	Run a shell or batch script containing Azure CLI commands against an Azure subscription

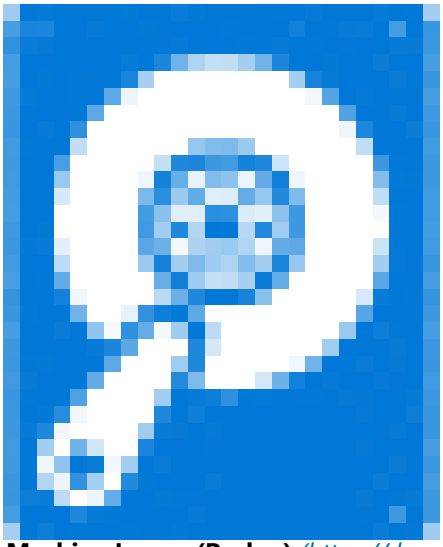
	Task
 **Azure Cloud PowerShell Deployment** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-cloud-powershell-deployment?view=vsts)*	Deploy an Azure Cloud Service
 **Azure File Copy** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-file-copy?view=vsts)*	Copy files to Azure blob or VM(s)

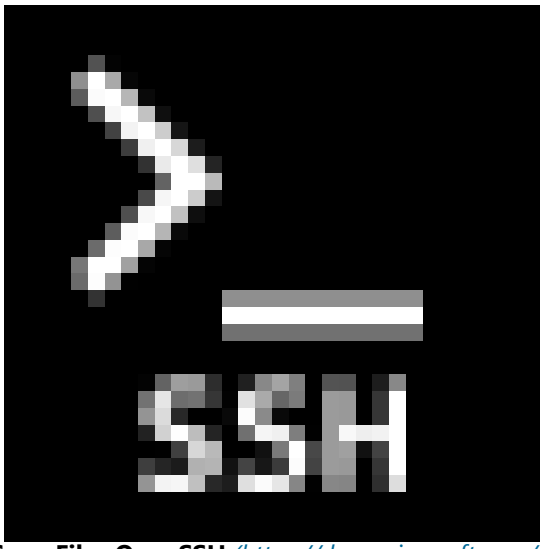
	Task
 **Azure Key Vault** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-key-vault?view=vsts)*	Incorporate secrets from an Azure Key Vault into a release pipeline
 **Azure Monitor Alerts** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-monitor-alerts?view=vsts)*	Configure alerts on available metrics for an Azure resource

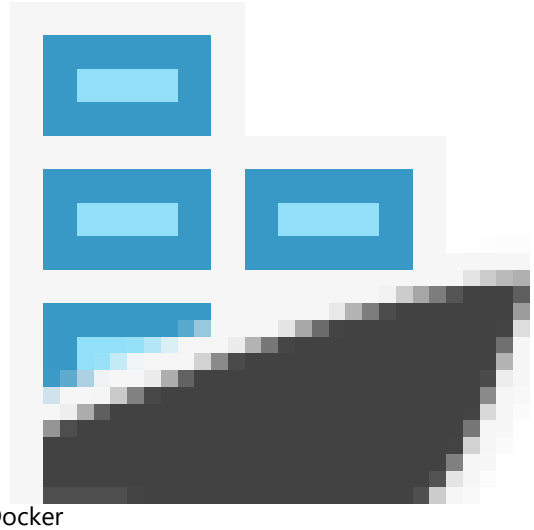
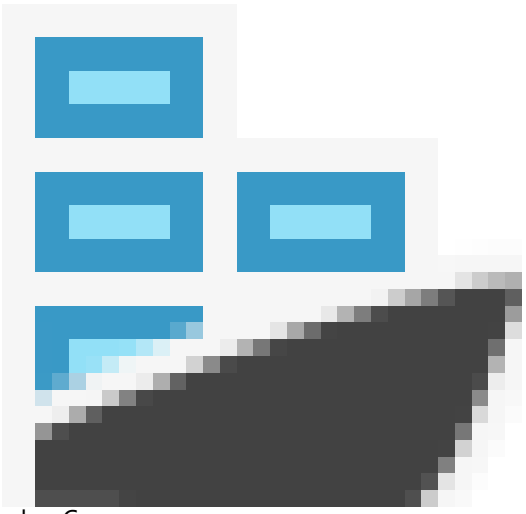
	Task
 **Azure MySQL Deployment** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-mysql-deployment?view=vsts)*	Run your scripts and make changes to your Azure DB for MySQL.
 **Azure Policy Check Gate** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-policy-check-gate?view=vsts)*	Security and compliance assessment with Azure policies on resources that belong to the resource group and Azure subscription.

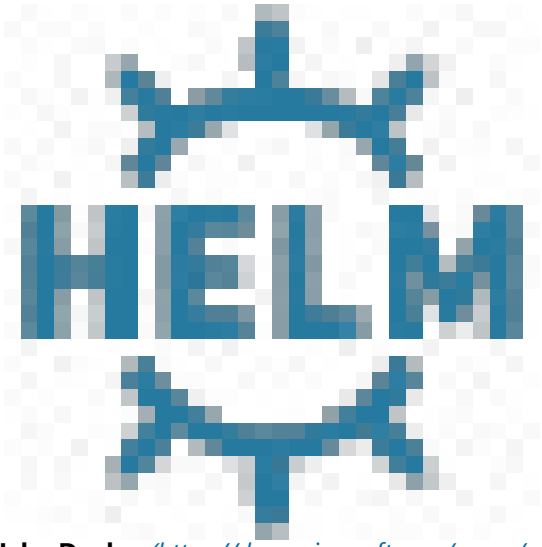
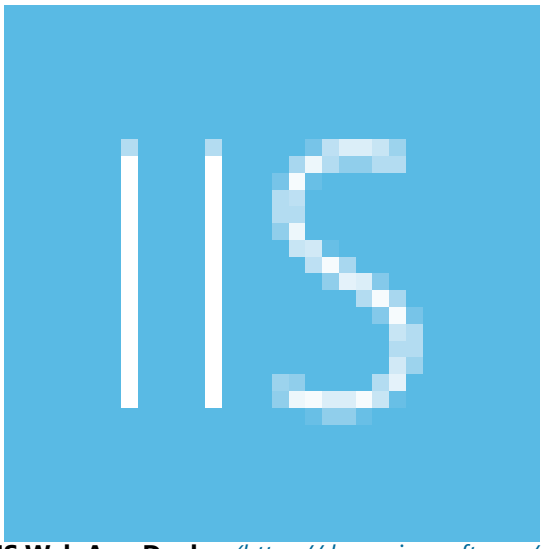
	Task
 **Azure PowerShell** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-powershell?view=vsts)*	Run a PowerShell script within an Azure environment
 **Azure Resource Group Deployment** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-resource-group-deployment?view=vsts)*	Deploy, start, stop, delete Azure Resource Groups

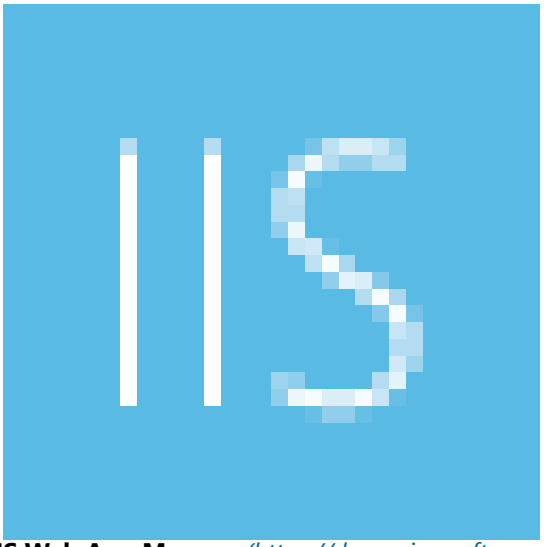
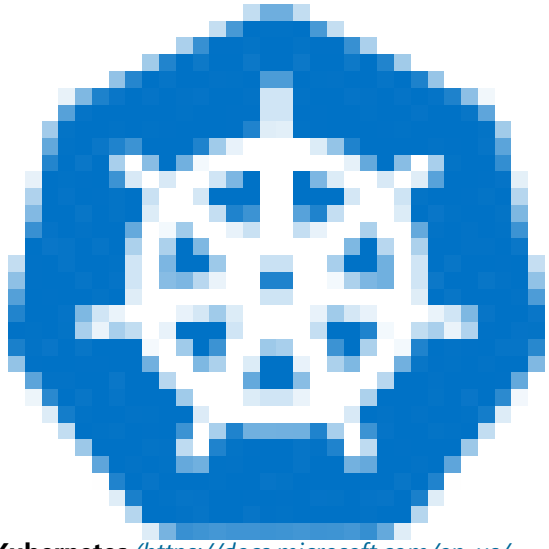
	Task
 **Azure SQL Database Deployment** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/sql-azure-dacpac-deployment?view=vsts)*	Deploy an Azure SQL database using DACPAC or run scripts using SQLCMD
 **Azure VM Scale Set Deployment** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-vmss-deployment?view=vsts)*	Deploy a virtual machine scale set image.

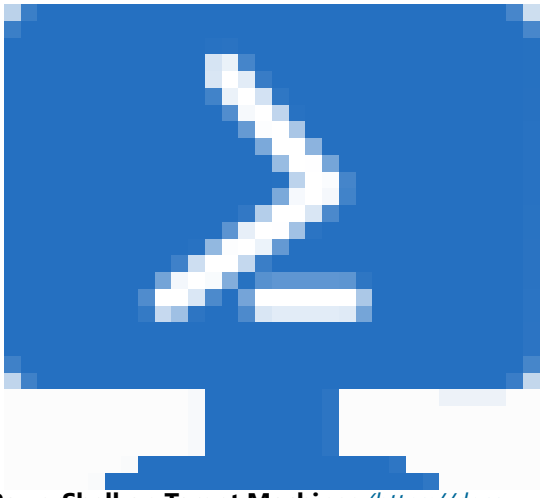
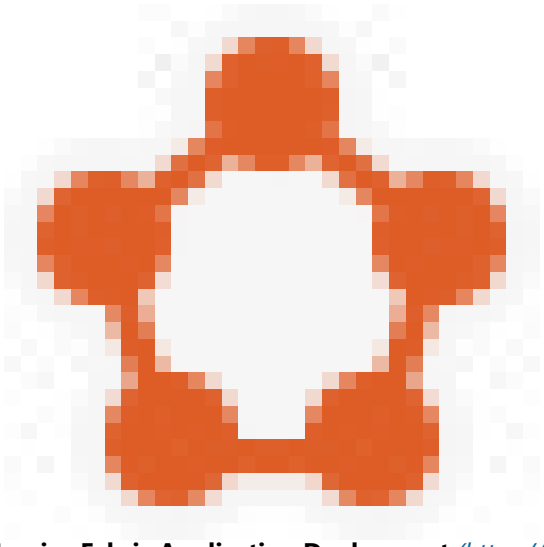
	Task
 **Build Machine Image (Packer)** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/packer-build?view=vsts)*	Build a machine image using Packer.
 **Chef** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/chef?view=vsts)*	Deploy to Chef environments by editing environment attributes

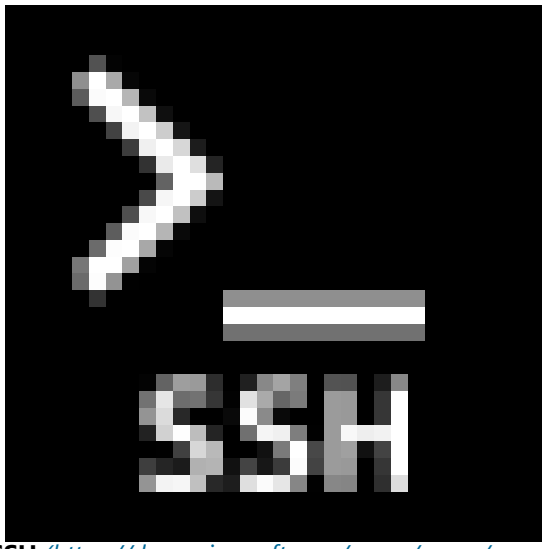
	Task
 **Chef Knife** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/chef-knife?view=vsts)*	Run Scripts with knife commands on your chef workstation
 **Copy Files Over SSH** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/copy-files-over-ssh?view=vsts)*	Copy files from source folder to target folder on a remote machine over SSH

	Task
 Docker	Build, tag, push, or run Docker images, or run a Docker command. Task can be used with Docker or Azure Container registry
 Docker Compose	Build, push or run multi-container Docker applications.

	Task
 **Helm Deploy** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/helm-deploy?view=vsts)*	Deploy, configure, update your Kubernetes cluster in Azure Container Service by running helm commands.
 **IIS Web App Deploy** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/iis-web-app-deployment-on-machine-group?view=vsts)*	Deploy a website or web app to a machine group using WebDeploy

	Task
 **IIS Web App Manage** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/iis-web-app-management-on-machine-group?view=vsts)*	Create or update a website, web app, virtual directory, or application pool on a machine group
 **Kubernetes** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/kubernetes?view=vsts)*	Deploy, configure, update your Kubernetes cluster in Azure Container Service by running kubectl commands.

	Task
 **PowerShell on Target Machines** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/powershell-on-target-machines?view=vsts)*	Execute PowerShell scripts on remote machine(s)
 **Service Fabric Application Deployment** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/service-fabric-deploy?view=vsts)*	Deploy a Service Fabric application to a cluster

	Task
 **Service Fabric Compose Deploy** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/service-fabric-compose-deploy?view=vsts)*	Deploy a Service Fabric application to a cluster using a compose file
 **SSH** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/ssh?view=vsts)*	Run shell commands or a script on a remote machine using SSH

	Task
 **Windows Machine File Copy** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/windows-machine-file-copy?view=vsts)*	Copy files to remote machine(s)
 **WinRM SQL Server DB Deployment** *(https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/sql-dacpac-deployment-on-machine-group?view=vsts)*	Deploy a SQL Server database using DACPAC or SQL scripts

This webpage[6] contains the full list of tasks.

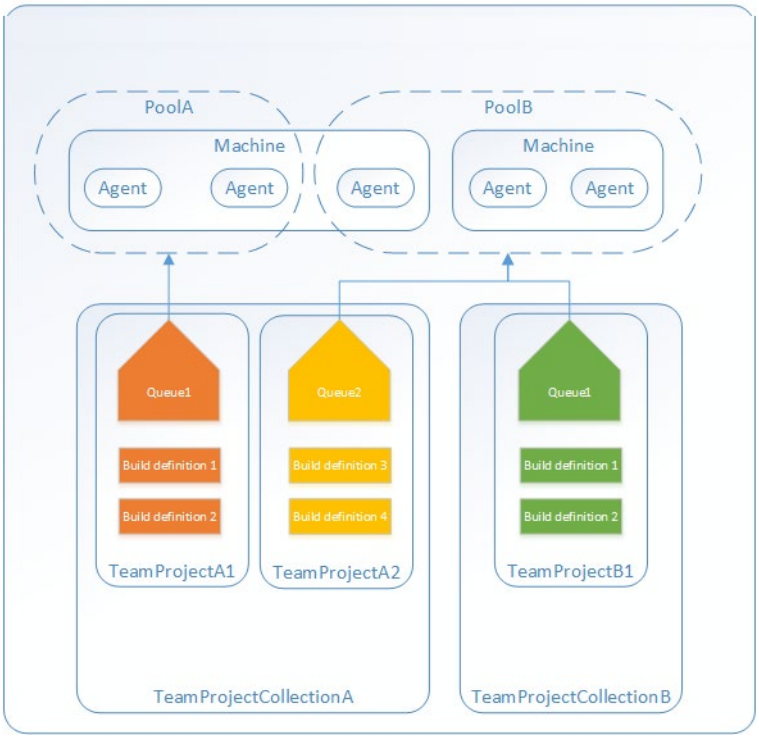
Agents, Agent Pools and Queues

A build agents is a running background service which listens for commands from the central server and starts and executes the actual build or release processes. It is installed and configured separately from the server (or saas solution).

6 https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/index?view=vsts

An agent can most of the times run on different Operating Systems and in some cases as a SaaS (hosted agent) solution.

Agents are organized into pools and provide access to the pools by using queues.



Agent pools

Agent pools are used to organize and define permission boundaries around your agents. In Azure DevOps, Pools are scoped to your organization. You can share your pool across multiple team project collections.

Agent Queues

An agent queue provides access to a pool of agents. When you create a build or release definition, you specify which queue it uses. Queues are scoped to your team project collection so that you can share them across build and release definitions in multiple team projects.

Build Agents and Server Options

Build agents are tasked with performing the builds. To build your code or deploy your software, you need at least one agent. As you add more code and add more people, you will eventually need more agents to form a pool.

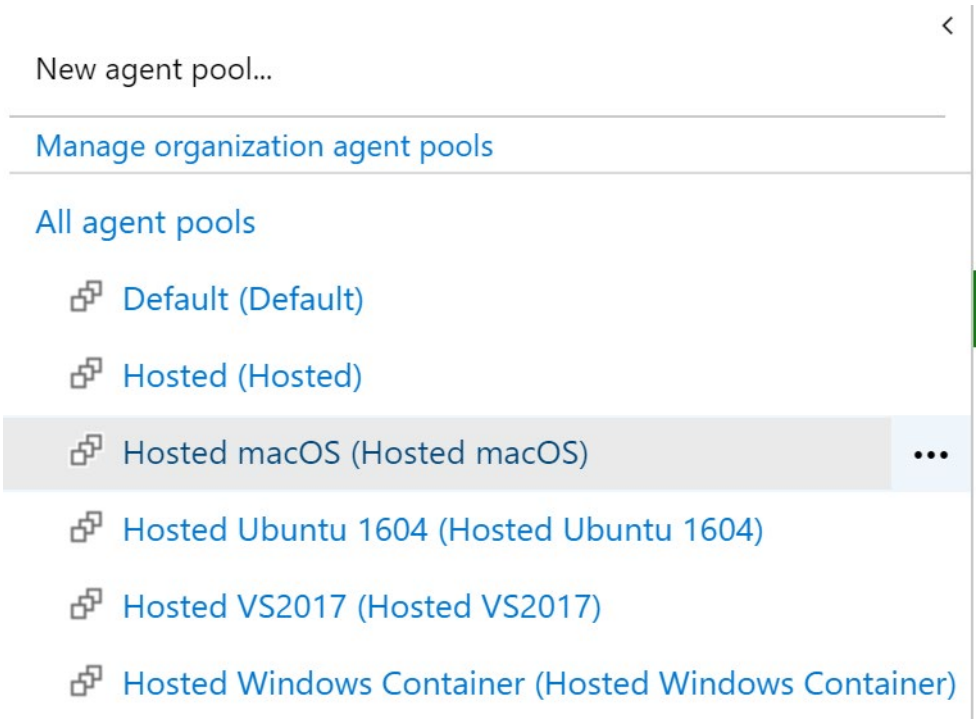
Pipelines are based on build agents, which are implemented as a background service installed on a virtual machine. Build agents are then grouped into *agent pools*, which are supplied from an *agent queue*. This means that build pipelines do not interact with build agents directly, but instead place a request in an agent queue, which is owned by a team project. Each agent queue feeds a single agent pool, and multiple queues may feed the same agent. The pool then provides build agents based on availability and the build definition requirements.

Build agents come in two types:

Hosted agents. These agents exist within their own hosted pool and are maintained and upgraded by the vendor. Hosted agents have specific limitations and advantages:

- Hosted agents have no cost and are immediately available, and have most common software and libraries installed.
- Do not have an interactive mode.
- Do not allow administrative privilege or allow logon.

Hosted agents come in many flavours. Based on the needs of your product and the platform that you target you can choose an appropriate agent. The agent software is cross-platform and can run on any platform



Private (or Custom) agents. Private agents are provisioned on private virtual machines (VMs) and are custom built to accommodate the project's needs.

System capabilities

System capabilities are name/value pairs that you can use to ensure that your build definition is run only by agents that meet the criteria that you specified. Environment variables automatically appear in the list. Some capabilities (such as frameworks) are also added automatically.

When a build is queued, the system sends the job only to agents that have the capabilities **demanded by the build definition**[7].

7 https://docs.microsoft.com/en-us/azure/devops/pipelines/build/options?view=vsts&tabs=yaml

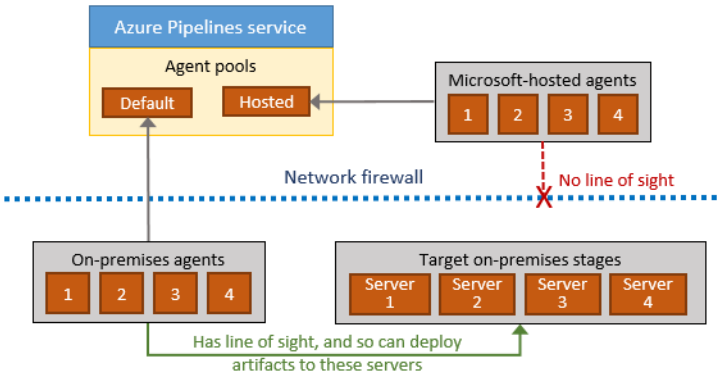
User capabilities

You can manually add capabilities (name/value pairs) that you know your agent has and that you want your build definition to be able to **demand**[8].

Agents and Deployment Groups

Agents run on a separate server and install and execute actions on a deployment target. By having an agent installed in this way, you need to have permissions to access the target environment. The target environment needs to accept incoming traffic. When you use Hosted agents, running in the cloud, you cannot use these agents to deploy software on your on-premises server.

To overcome this issue, you can run agents in your local network that communicate with the central server.



If this also does not work, you need to fall back on the traditional approach of installing agents on the target environment. In Azure DevOps, this is called Deployment Groups. The agent on the target server is registered and do everything themselves. This means that they only need to have outbound access and is sometimes the only allowed option

- **More about agents**[9]
- **More about Deployment Groups**[10]

Learn more

Read more about Hosted Agent[11]

Read more on agent pools and queues[12]

Information about Hosted agents running Linux[13]

More information on setting up private agents in Windows[14]

More information on setting up Private Agents in OSX[15]

8 https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/v2-linux?view=vsts
9 https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/agents?view=vsts
10 https://docs.microsoft.com/en-us/azure/devops/pipelines/release/deployment-groups/?view=vsts
11 https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/hosted?view=vsts&tabs=yaml
12 https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/pools-queues?view=vsts
13 https://docs.microsoft.com/en-us/vsts/build-release/actions/agents/v2-linux?view=vsts
14 https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/v2-windows?view=vsts
15 https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/v2-osx?view=vsts

More information on setting up Private Agents in Linux[16]

Release Agent Jobs Introduction

What is a Release Agent Job?

You can organize your build or release pipeline into jobs. Every build or deployment pipeline has at least one job.

A job is a series of tasks that run sequentially on the same target. This can be a Windows server, a Linux server, a container or a deployment group. A release job is executed by a build/release agent. This agent can only execute one job at the same time.

During the design of your job, you specify a series of tasks that you want to run on the same agent. At runtime (when either the build or release pipeline is triggered), each job is dispatched as one or more jobs to its target.

A scenario that speaks to the imagination, where Jobs play an essential role is the following.

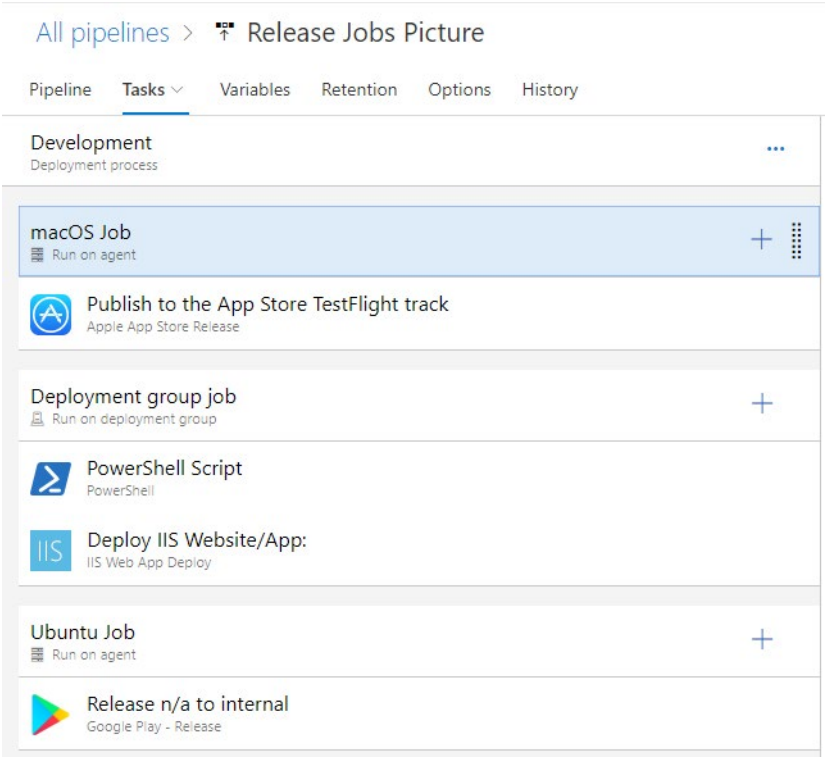
Assume that you built an application, with a backend in .Net, a front end in Angular and a native IOS mobile App. This might be developed in 3 different source control repositories triggering three different builds, delivering three different artifacts.

The release pipeline brings the artifacts together and wants to deploy the backend, frontend, and Mobile App all together as part of 1 release. The deployment needs to take place on different agents. An IOS app needs to be built and distributed from a Mac, and the angular app is hosted on Linux so best deployed from a Linux machine. The backend might be deployed from a Windows machine.

Because you want all three deployments to be part of one pipeline, you can define multiple Release Jobs, which target the different agents, server or deployment groups.

By default, jobs run on the host machine where the agent is installed. This is convenient and typically well-suited for projects that are just beginning to adopt continuous integration (CI). Over time, you may find that you want more control over the stage where your tasks run.

16 https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/v2-linux?view=vsts



More about Jobs[17]

Using Release Jobs

You can have different Release Jobs. In this chapter, they are briefly covered

Jobs or Agent Jobs

A job is a series of tasks that run sequentially on the same target. At design time in your job, you specify a set of tasks that you want to run on a common target. At runtime (when either the build or release pipeline is triggered), each job is dispatched as one or more jobs to its target.

When the target is an agent, the tasks are run on the computer that hosts the agent. This agent can only execute one job at the same time.

More about Agent Jobs[18]

Server or Agentless Jobs

Tasks in a server or agentless job are orchestrated by and executed on the server (Azure Pipelines or TFS). A server job does not require an agent or any target computers. Only a few tasks, such as the Manual Intervention and Invoke REST API tasks, are supported in a server job at present.

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

Container Jobs

Containers offer a lightweight abstraction over the host operating system. You can select the exact versions of operating systems, tools, and dependencies that your build requires. When you specify a container in your pipeline, the agent will first fetch and start the container. Then, each step of the job will run inside the container.

More about Container Jobs[19]

Deployment Group Jobs

Deployment groups make it easy to define groups of target servers for deployment. Tasks that you define in a deployment group job run on some or all of the target servers, depending on the arguments you specify for the tasks and the job itself.

You can select specific sets of servers from a deployment group to receive the deployment by specifying the machine tags that you have defined for each server in the deployment group. You can also specify the proportion of the target servers that the pipeline should deploy to at the same time. This ensures that the app running on these servers is capable of handling requests while the deployment is taking place.

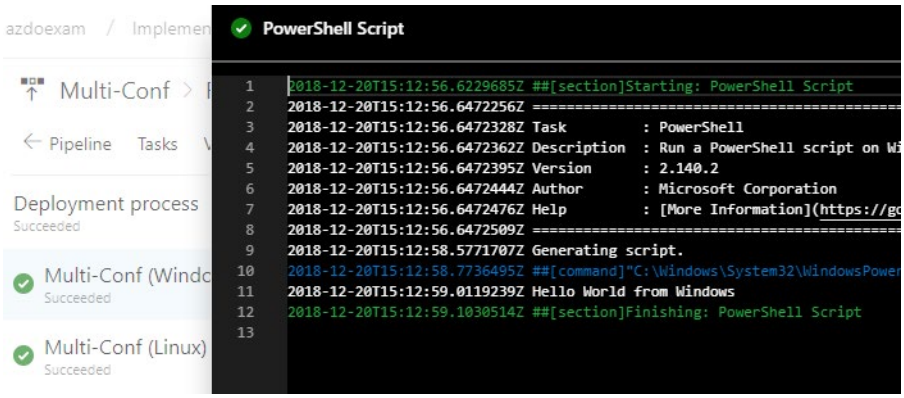
More about Deployment Group Jobs[20]

Multi-Configuration and Multi-Agent

We talked about Jobs and how to make it possible to divide the work between different agents or servers. Sometimes you want to run the tasks of a pipeline multiple times but with a slightly different configuration, or split a large batch of work amongst multiple agents.

There are three different types of job you can run.

- **None**: Tasks will run on a single agent.
- **Multi-configuration**: Run the same set of tasks on multiple configurations as specified in the multipliers. Configurations will run in parallel, and each configuration will use a single agent. For example
  - Run the release once with configuration Setting A on WebApp A and setting B for WebApp B
  - Deploy to different geographic regions.
  - Multi-configuration testing: run a set of tests in parallel - once for each test configuration.



19 https://docs.microsoft.com/en-us/azure/devops/pipelines/process/container-phases?view=vsts&tabs=yaml
20 https://docs.microsoft.com/en-us/azure/devops/pipelines/process/deployment-group-phases?view=vsts&tabs=yaml

Find more information here[21]

- **Multi-agent**: Run the same set of tasks on multiple agents using the specified number of agents. For example, you can run a broad suite of 1000 tests on a single agent. Or, you can use two agents and run 500 tests on each one in parallel.

https://docs.microsoft.com/en-us/azure/devops/pipelines/process/multiple-phases?view=vsts&tabs=designer

Discussion

How to use Release jobs

Do you see a purpose for Release Jobs in your pipeline and how would you set it up?

Topics you might want to consider are:

- Do you have artifacts from multiple sources?
- Do you want to run deployments on different servers simultaneously?
- Do you need multiple platforms?
- How long does your release take?
- Can you run your deployment in parallel or does it need to run in sequence?

Release Variables

Variables give you a convenient way to get critical bits of data into various parts of the pipeline. As the name suggests, the contents of a variable may change between releases, stages of jobs of your pipeline. The system predefines some variables, and you are free to add your own as well.

The most important thing you need to think about when using variables in the release pipeline is the scope of the variable. You can imagine that a variable containing the name of the target server may vary between a Development environment and a Test Environment.

Within the release pipeline, you can use variables in different scopes and different ways.

More information[22]

Predefined variables

When running your release pipeline, there are always variables that you need that come from the agent or context of the release pipeline. For example, the agent directory where the sources are downloaded, the build number or build id, the name of the agent or any other information. This information is usually accessible in pre-defined variables that you can use in your tasks.

More information about predefined variables[23]

Release pipeline variables

Choose a release pipeline variable when you need to use the same value across all the stages and tasks in the release pipeline, and you want to be able to change the value in a single place.

21 https://docs.microsoft.com/en-us/azure/devops/pipelines/process/phases?view=vsts&tabs=designer#multi-configuration
22 https://docs.microsoft.com/en-us/azure/devops/pipelines/release/variables?view=vsts&tabs=batch
23 https://docs.microsoft.com/en-us/azure/devops/pipelines/release/variables?view=vsts&tabs=batch#default-variables

Stage variables

Share values across all of the tasks within one specific stage by using stage variables. Use a stage-level variable for values that vary from stage to stage (and are the same for all the tasks in a stage).

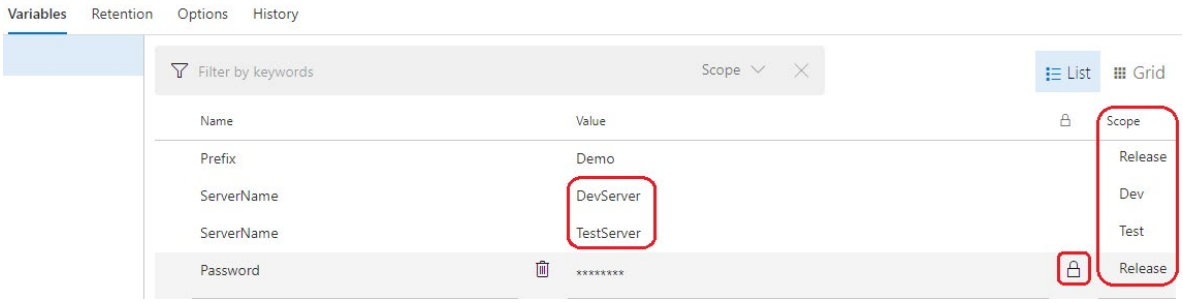
Variable groups

Share values across all of the definitions in a project by using variable groups. We will cover variable groups later in this module.

Normal and Secret variables

Because the tasks of the pipeline are executed on an agent, usually variable values are passed to the various tasks using environment variables. The task knows how to read this. You should be aware that a variable contains clear text and can be exposed on the target system. When you use the variable in the log output, you can also see the value of the variable. When the pipeline has finished, the values will be cleared.

You can mark a variable in the release pipeline as secret. This way the secret is hidden from the log output. This is especially useful when writing a password or other sensitive information



Provision and Configure Environments

Provision and Configure Different Target Environments

The release pipeline deploys software to a target environment. But, it is not only the software that will be deployed with the release pipeline. If you are truly focusing on Continuous Delivery, Infrastructure as Code and spinning up infrastructure as part of your release pipeline is very important.

When we focus on the deployment of the infrastructure, we should first consider the differences between the target environments that we can deploy to.

- On-Premises servers
- Cloud servers or Infrastructure as a Service (IaaS). For example Virtual machines or networks.
- Platform as a Service (PaaS) and Functions as a Service (FaaS). For example Web apps or storage accounts.
- Clusters.

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

On-Premises servers

In most cases, when you deploy to an on-premises server, the hardware and the operating system is already in place. The server is already there and ready Sometimes empty but most of the times not. In this case, the release pipeline can focus on deploying the application only.

In some cases, you might want to start or stop a virtual machine (for example Hyper-V or VMWare). The scripts that you use to start or stop the on-premises servers should be part of your source control and be delivered to your release pipeline as a build artifact. Using a task in the release pipeline, you can run the script that starts or stops the servers.

When you want to take it one step further, and you want to configure the server as well. You should take a look at technologies like PowerShell Desired State Configuration(DSC), or use tools like Puppet and Chef. All these products will maintain your server and keep it in a particular state. When the server changes its state, they (Puppet, Chef, DSC) recover the changed configuration to the original configuration.

Integrating a tool like Puppet, Chef or Powershell DSC in to the release pipeline is no different from any other task you add.

Infrastructure as a service.

When you use the cloud as your target environment things change a little bit. Some organizations did a lift and shift from their on-premises server to cloud servers. Then your deployment works the same as to an on-premises server. But when you use the cloud to provide you with Infrastructure as a Service (IaaS), you can leverage the power of the cloud, to start and create servers when you need them.

This is where Infrastructure as Code (IaC) starts playing a significant role. By creating a script or template, you can create a server or other infrastructural components like a SQL server, a network or an IP address. By defining a template or using a command line and save it in a script file, you can use that file in your release pipeline tasks to execute this on your target cloud. As part of your pipeline, the server (or another component) will be created. After that, you can execute the steps actually to deploy the software.

Technologies like Azure Resource Manager (ARM) or Terraform are great to create infrastructure on demand.

Platform as a Service

When you are moving from Infrastructure as a Service (IaaS) towards Platform as a Service (PaaS), you will get the infrastructure from the cloud that you are running on.

For example: In Azure, you can choose to create a Web application. The server, the hardware, the network, the public IP address, the storage account, and even the web server, is arranged by the cloud. The user only needs to take care of the web application that will run on this platform.

The only thing that you need to do is to provide the templates which instruct the cloud to create a WebApp. The same goes for Functions as a Service(FaaS or Serverless technologies. In Azure called Azure Functions and in AWS called AWS Lambda.

You only deploy your application, and the cloud takes care of the rest. However, you need to instruct the platform (the cloud) to create a placeholder where your application can be hosted. You can define this template in ARM or Terraform. You can use the Azure CLI or command line tools or in AWS use CloudFormation. In all cases, the infrastructure is defined in a script file and live alongside the application code in source control.

Clusters.

Last but not least you can deploy your software to a cluster. A cluster is a group of servers that work together to host high-scale applications.

When you run a cluster as Infrastructure as a Service, you need to create and maintain the cluster. This means that you need to provide the templates to create a cluster. You also need to make sure that you roll out updates, bug fixes and patches to your cluster. This is comparable with Infrastructure as a Service.

When you use a hosted cluster, you should consider this as Platform as a Service. You instruct the cloud to create the cluster, and you deploy your software to the cluster. When you run a container cluster, you can use the container cluster technologies like Kubernetes or Docker Swarm.

Summary

Regardless of the technology, you choose to host your application, the creation, or at least configuration of your infrastructure should be part of your release pipeline and part of your source control repository. Infrastructure as Code is a fundamental part of Continuous Delivery and gives you the freedom to create servers and environments on demand.

Links

- **AWS Cloudformation**[24]
- **Terraform**[25]
- **Powershell DSC**[26]
- **AWS Lambda**[27]

24 https://aws.amazon.com/cloudformation/
25 https://www.terraform.io/
26 https://docs.microsoft.com/en-us/powershell/dsc/overview/overview
27 https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

- **Azure Functions**[28]
- **Chef**[29]
- **Puppet**[30]
- **Azure Resource Manager /ARM**[31]

Lab: Automating your infrastructure deployments in the Cloud with Terraform and Azure Pipelines

Overview

Terraform[32] is a tool for building, changing, and versioning infrastructure safely and efficiently. Terraform can manage existing and popular cloud service providers as well as custom in-house solutions.

Configuration files describe to Terraform the components needed to run a single application or your entire datacenter. Terraform generates an execution plan describing what it will do to reach the desired state, and then executes it to build the described infrastructure. As the configuration changes, Terraform is able to determine what changed and create incremental execution plans which can be applied.

What's covered in this lab

In this lab, you will see

- How open source tools, such as Terraform can be leveraged to implement Infrastructure as Code (IaC)
- How to automate your infrastructure deployments in the Cloud with Terraform and Azure Pipelines

Follow the instruction here[33]

Setting up Endpoints to other environments

As discussed in the previous chapter, there are various target environments that you can set up and can use as a deployment target.

The course **AZ-400T05 Implementing Application Infrastructure** contains an extensive overview of setting up this Application Infrastructure and using them in your pipelines.

Demonstration Setting Up Service Connections

In this demonstration, you will investigate Service Connections.

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

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

28 https://azure.microsoft.com/en-us/services/functions
29 https://www.chef.io/chef/
30 https://puppet.com/
31 https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-overview
32 https://www.terraform.io/intro/index.html
33 https://azuredevopslabs.com/labs/vstsextend/terraform/

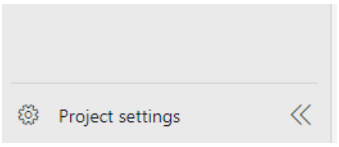
Steps

Let's now take a look at how a release pipeline can access resources that require a secure connection. In Azure DevOps, these are implemented by Service Connections.

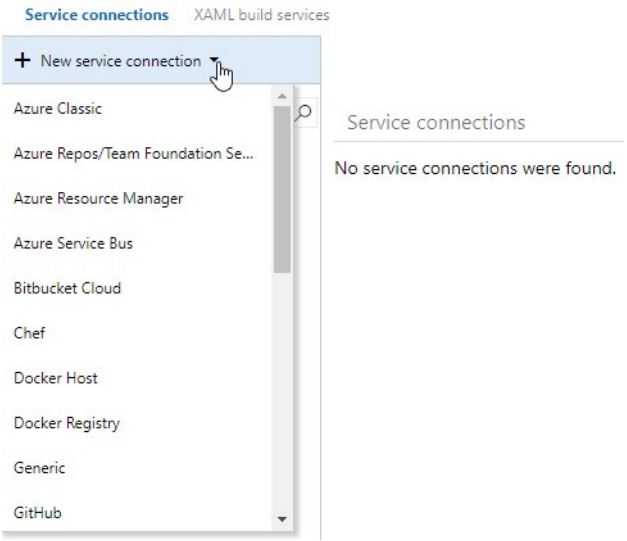
You can set up a service connection to environments to create a secure and safe connection to the environment that you want to deploy to. Service connections are also used to get resources from other places in a secure manner. For example, you might need to get your source code from GitHub.

In this case, let's take a look at configuring a service connection to Azure.

1. From the main menu in the **Parts Unlimited** project, click **Project settings** at the bottom of the screen.



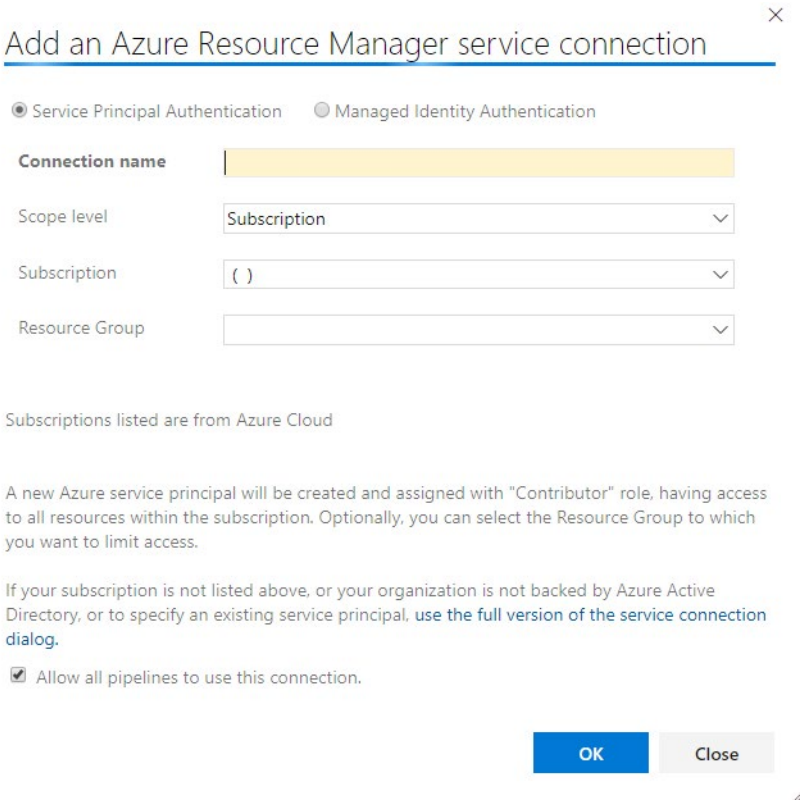
2. In the Project Settings pane, from the **Pipelines** section, click **Service connections**. Click the drop down beside **+New service connection**.



As you can see, there are many types of service connections. You can create a connection to the Apple App Store or to the Docker Registry, to Bitbucket, or to Azure Service bus.

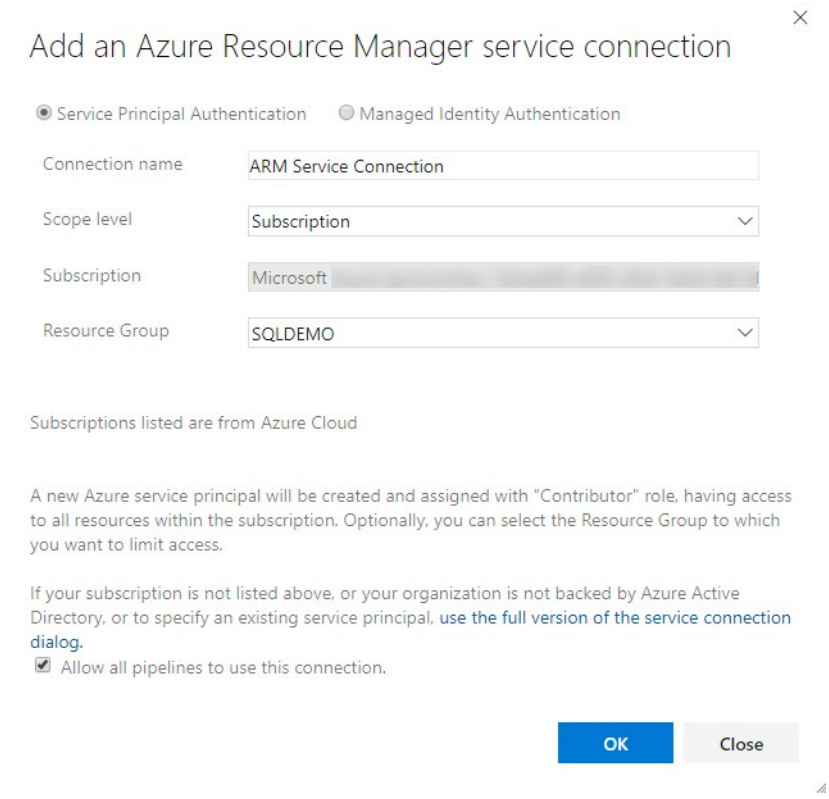
In this case, we want to deploy a new Azure resource, so we'll use the Azure Resource Manager option.

3. Click **Azure Resource Manager** to add a new service connection.



4. Set the **Connection name** to **ARM Service Connection**, click on an Azure **Subscription**, then select an existing **Resource Group**.

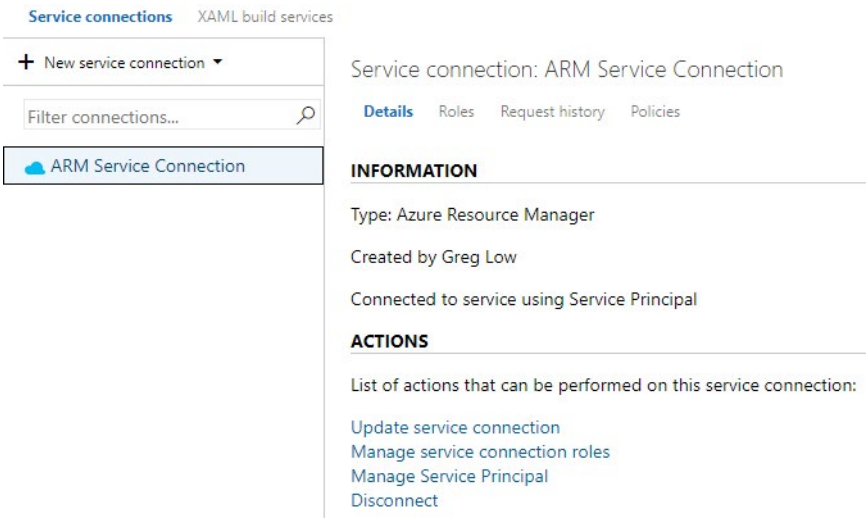
Note: You might be prompted to logon to Azure at this point. If so, logon first.



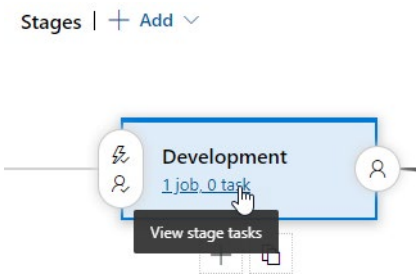
Notice that what we are actually creating is a **Service Principal**. We will be using the Service Principal as a means of authenticating to Azure. At the top of the window, there is also an option to set up Managed Identity Authentication instead.

The Service Principal is a type of service account that only has permissions in the specific subscription and resource group. This makies it a very safe way to connect from the pipeline.

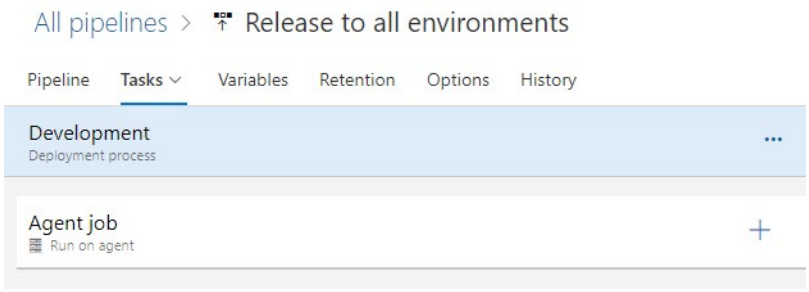
5. Click **OK** to create it. It will then be shown in the list.



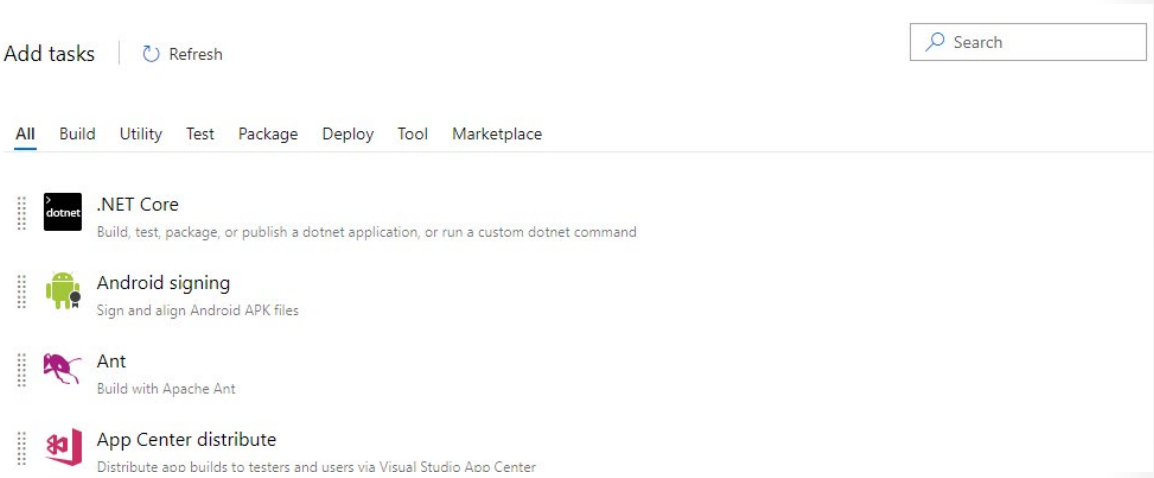
6. In the main Parts Unlimited menu, click **Pipelines** then **Releases*, then **Edit** to see the release pipeline. Click the link to **View stage tasks**.



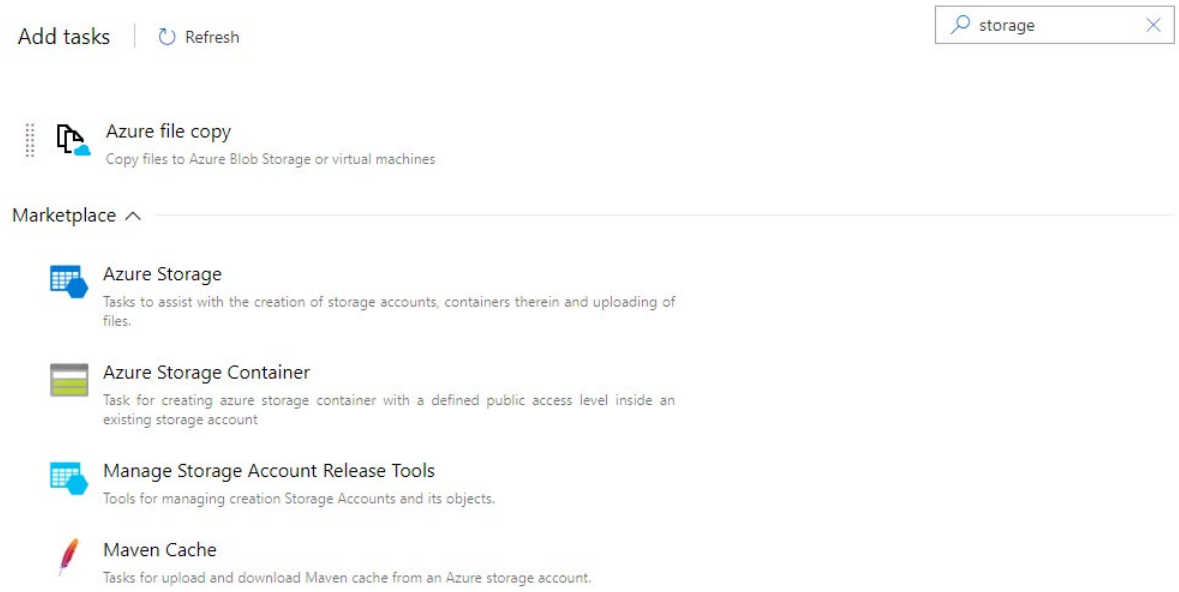
The current list of tasks is then shown. Because we started with an empty template, there are no tasks as yet. Each stage can execute many tasks.



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

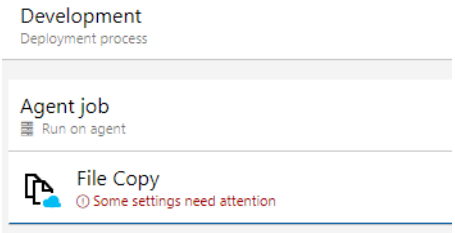


8. In the **Search** box, enter the word **storage** and note the list of storage-related tasks. These include standard tasks, and tasks available from the Marketplace.

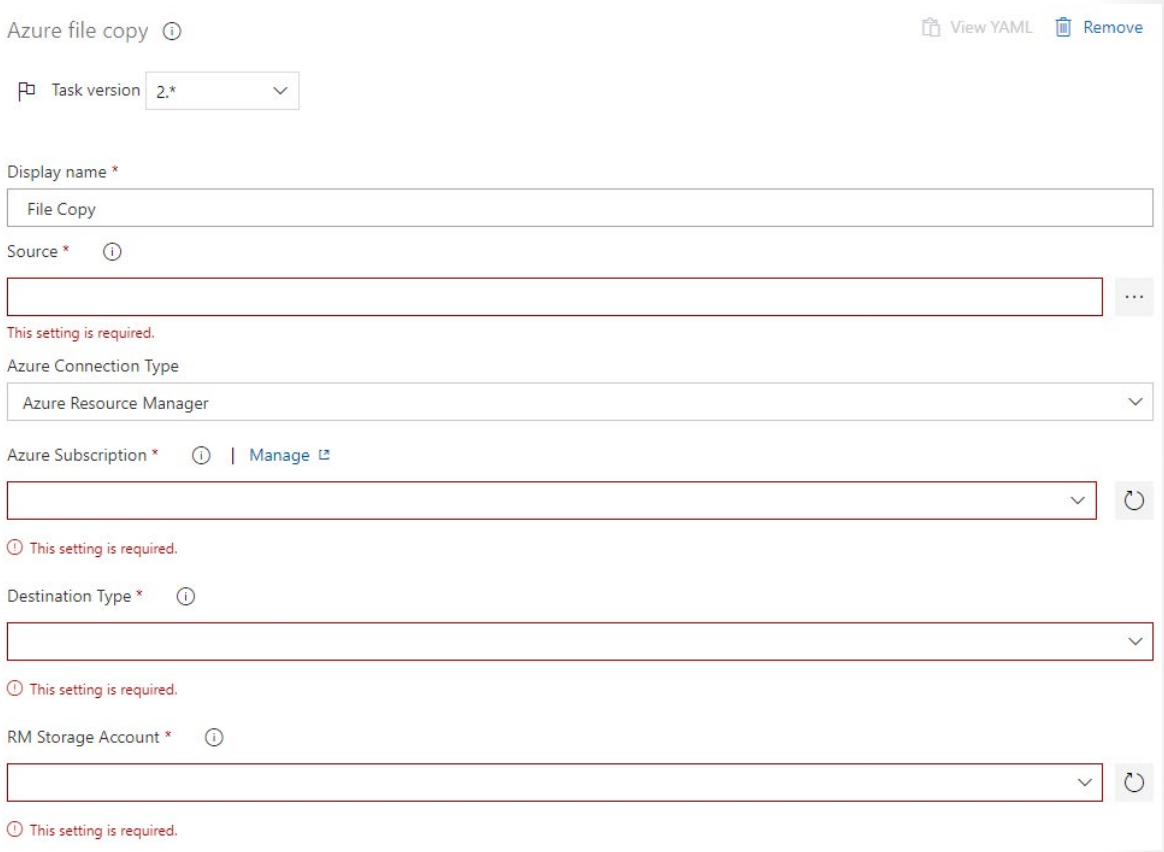


We will use the Azure file copy task to copy one of our source files to a storage account container.

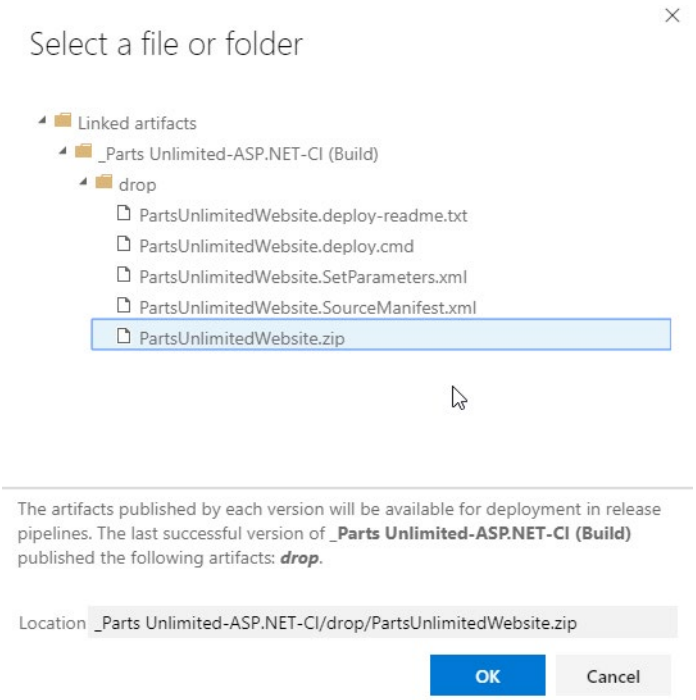
9. Hover over the **Azure file copy** task type, and click **Add** when it appears. The task will be added to the stage but requires further configuration.



10. Click the **File Copy** task to see the required settings.

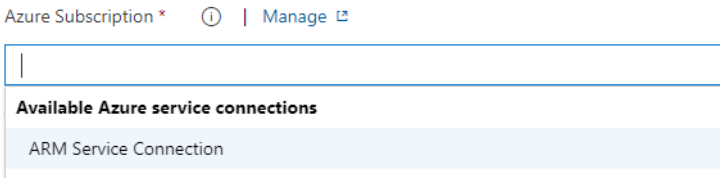


11. Set the **Display Name** to **Backup website zip file**, then click the ellipsis beside **Source** and locate the file as follows, then click **OK** to select it.



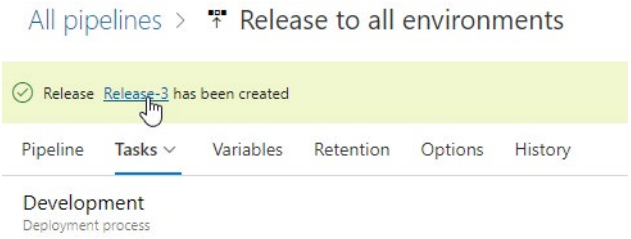
We then need to provide details of how to connect to the Azure subscription. The easiest and most secure way to do that is to use our new Service Connection.

12. From the **Azure Subscription** drop down list, find and select the **ARM Service Connection** that we created.

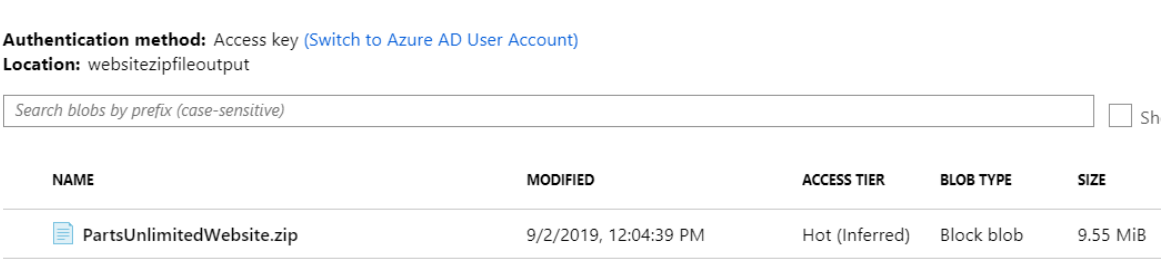


13. From the **Destination Type** drop down list, select **Azure Blob**, and from the **RM Storage Account** and **Container Name**, select the storage account, and enter the name of the container, then click **Save** at the top of the screen and **OK**.

14. To test the task, click **Create release**, and in the **Create a new release** pane, click **Create**.
15. Click the new release to view the details.



16. On the release page, approve the release so that it can continue.
17. Once the **Development** stage has completed, you should see the file in the Azure storage account.



A key advantage of using service connections is that this type of connection is is managed in a single place within the project settings, and doesn't involve connection details spread throughout the pipeline tasks.

Manage And Modularize Tasks and Templates

Manage And Modularize Tasks and Templates

Introduction

When you deploy multiple applications and have multiple builds and release pipelines, reusability comes into mind.

For example, you deploy to the same set of servers and use the same connection strings. You want to store the values in one place so you can easily update them.

The same goes for some actions you always want to perform together. For example, stop a website, running a script, start a website.

Sometimes you want to take this even one step further. You want to do some specialized actions, that are not available in the public marketplace. By writing a reusable task that you can use in your organization or even make public, is an excellent way to encapsulate logic and distribute this safely and securely.

Within the Azure DevOps suite, three important concepts enable reusability.

- Task Groups
- Variable Groups
- Custom Build and Release Tasks

Task Groups

A task group allows you to encapsulate a sequence of tasks, already defined in a build or a release pipeline, into a single reusable task that can be added to a build or release pipeline, just like any other task. You can choose to extract the parameters from the encapsulated tasks as configuration variables, and abstract the rest of the task information.

Task groups are a way to standardize and centrally manage deployment steps for all your applications. When you include a task group in your definitions, and then make a change centrally to the task group, the change is automatically reflected in all the definitions that use the task group. There is no need to change each one individually.

More about Task Groups[34]

Variable Groups

A variable group is used to store values that you want to make available across multiple builds and release pipelines.

Examples

- Store the username and password for a shared server
- Store a share connection string
- Store the geolocation of an application
- Store all settings for a specific application

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

More about Variable Groups[35]

Custom Tasks

Instead of using the out-of-the-box tasks, or using a command line or shell script, you can also use your custom build and release task. By creating your own tasks, the tasks are available for publicly or privately to everyone you share it with.

Creating your own task has significant advantages.

- You get access to variables that are otherwise not accessible,
- you can use and reuse secure endpoint to a target server
- you can safely and efficiently distribute across your whole organization
- users do not see implementation details.

More information about creating your own custom tasks[36]

Demonstration Creating and Managing Task Groups

In this demonstration, you will investigate Task Groups.

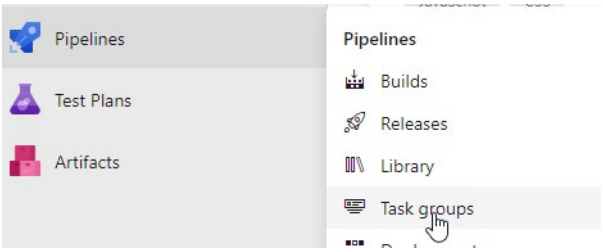
Note: Before starting this walkthrough, make sure you have performed the steps in the prerequisites section and the previous walkthroughs.

Steps

Let's now take a look at how a release pipeline can reuse groups of tasks.

It's common to want to reuse a group of tasks in more than one stage within a pipeline or in different pipelines.

1. In the main menu for the **Parts Unlimited** project, click **Pipelines** then click **Task groups**.

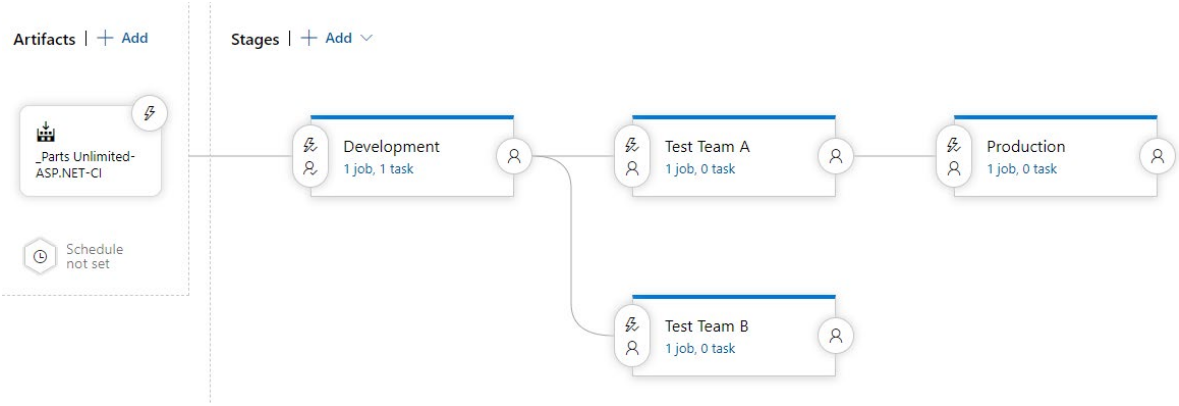


You will notice that you don't currently have any task groups defined.

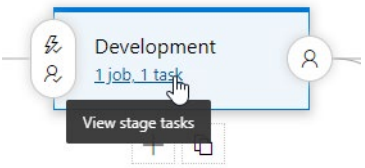
35 https://docs.microsoft.com/en-us/azure/devops/pipelines/library/variable-groups?view=vsts
36 https://docs.microsoft.com/en-us/azure/devops/extend/develop/add-build-task?view=vsts

There is an option to import task groups but the most common way to create a task group is directly within the release pipeline, so let's do that.

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

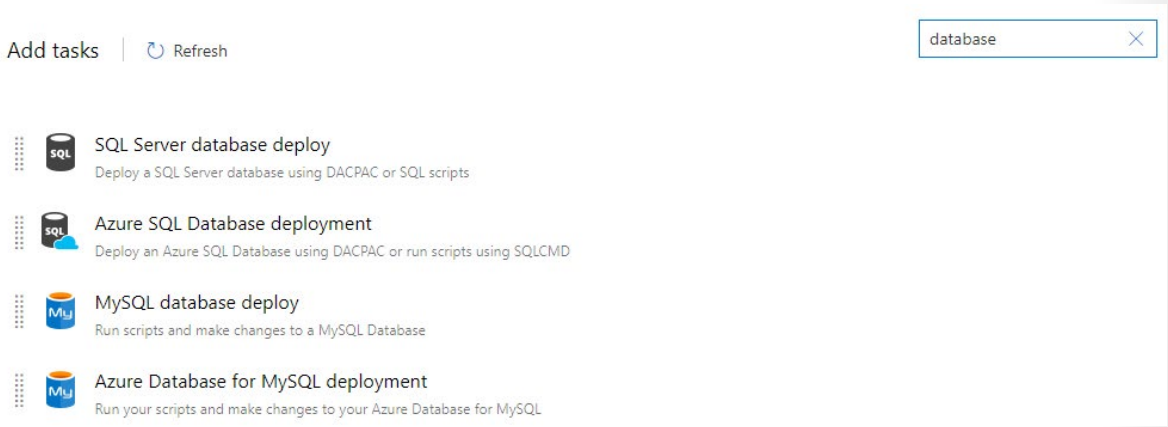


3. The **Development** stage currently has a single task. We will add another task to that stage. Click the **View stage tasks** link to open the stage editor.



You can see that there is currently one task.

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

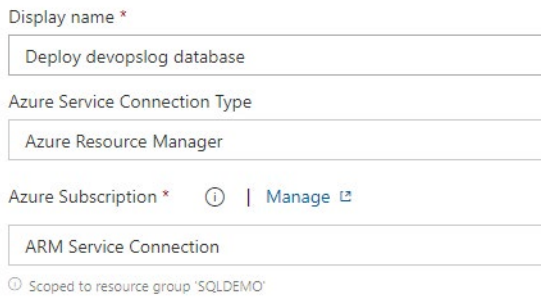


We will add a task to deploy an Azure SQL Database.

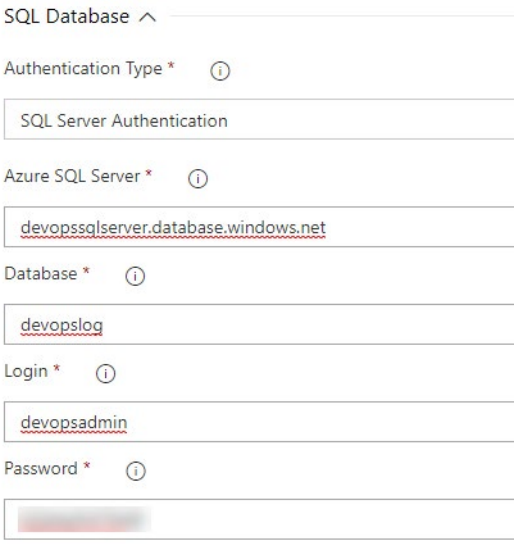
5. Hover over the **Azure SQL Database Deployment** option and click **Add**. Click the **Azure SQL DacpacTask** when it appears in the list, to open the settings pane.

6. Set the **Display name** to **Deploy devopslog database**, and from the **Azure Subscriptions" drop down list, click **ARM Service Connection**.

Note: we are able to reuse our service connection here

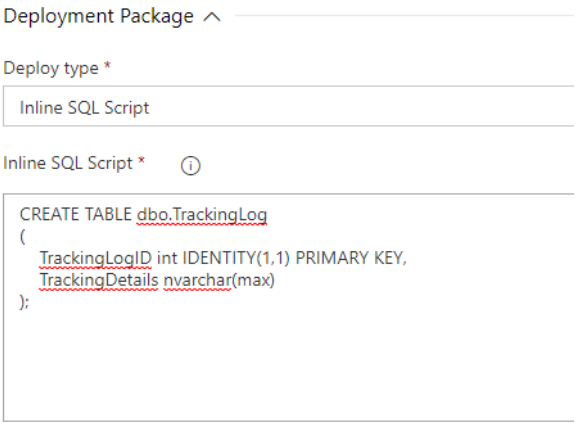


7. In the **SQL Database** section, set a unique name for the SQL Server, set the **Database** to **devopslog**, set the **Login** to **devopsadmin**, and set any suitable password.



8. In the **Deployment Package** section, set the **Deploy type** to **Inline SQL Script**, set the **Inline SQL Script** to:

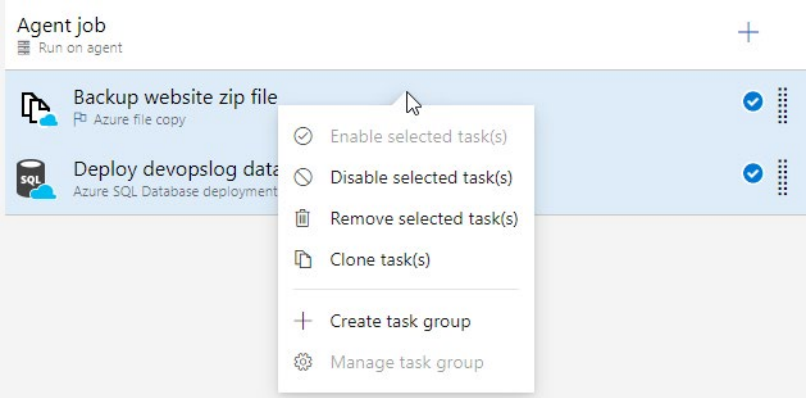
```
CREATE TABLE dbo.TrackingLog
(
    TrackingLogID int IDENTITY(1,1) PRIMARY KEY,
    TrackingDetails nvarchar(max)
);
```



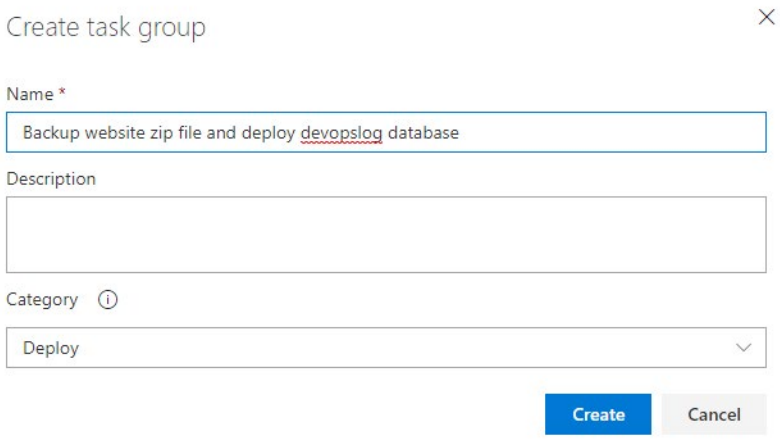
9. Click **Save** then **OK** to save the work.

Now that we have two tasks, let's use them to create a task group.

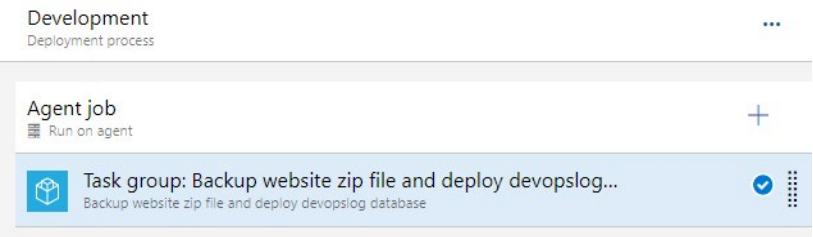
10. Click to select the **Backup website zip file** task and also select the **Deploy devopslog database** task, then right-click either task.



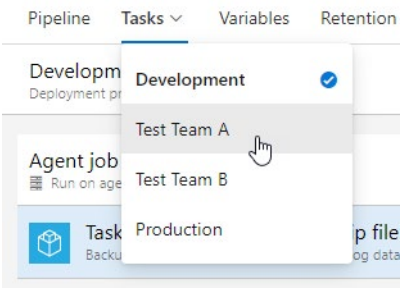
11. Click **Create task group**, then in the **Create task group** window, set **Name** to **Backup website zip file and deploy devopslog**. Click the **Category** drop down list to see the available options. Ensure that **Deploy** is selected, and click **Create**.



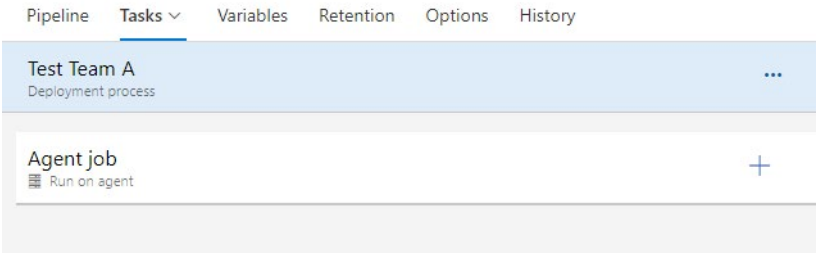
In the list of tasks, the individual tasks have now disappeared and the new task group appears instead.



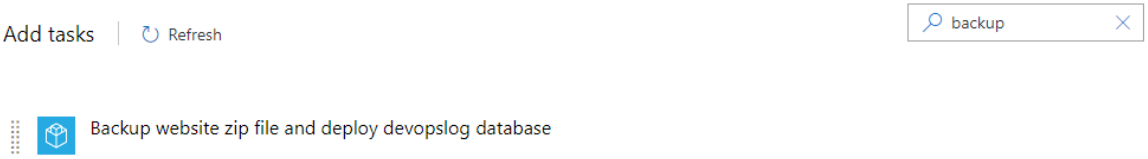
12. From the **Task** drop down list, select the **Test Team A** stage.



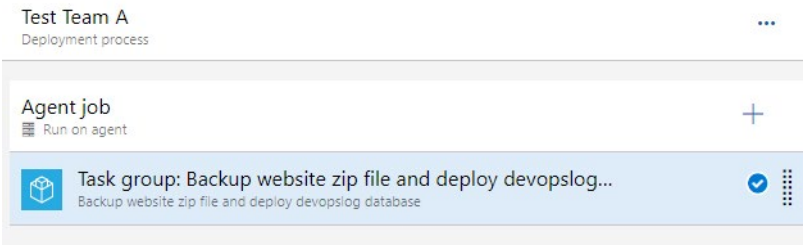
There are currently no tasks in the stage.



13. Click the **+** sign to the right of **Agent job** to add a new task. In the **Search** box, type **backup** and notice that the new task group appears like any other task.



14. Hover on the task group and click **Add** when it appears.



Task groups allow for each reuse of a set of tasks and limits the number of places where edits need to occur.

Walkthrough cleanup

15. Click **Remove** to remove the task group from the **Test Team A** stage.
16. From the **Tasks** drop down list, select the **Development** stage. Again click **Remove** to remove the task group from the **Development** stage.
17. Click **Save** then **OK**.

Demonstration Creating and Managing Variable Groups

In this demonstration, you will investigate Variable Groups.

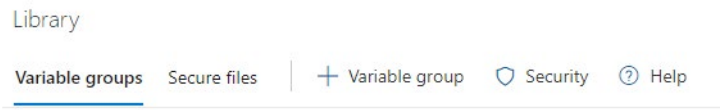
Note: Before starting this walkthrough, make sure you have performed the steps in the prerequisites section and the previous walkthroughs.

Steps

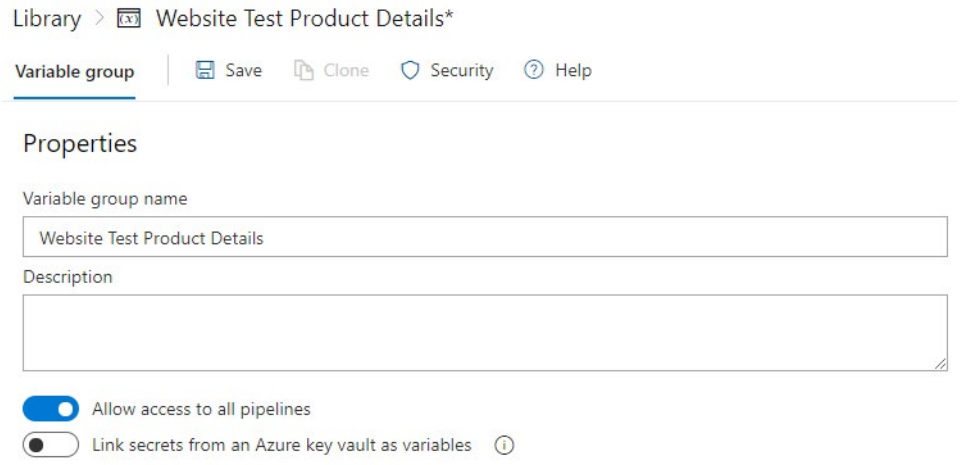
Let's now take a look at how a release pipeline can make use of predefined sets of variables, called Variable Groups.

Similar to the way we used task groups, variable groups provide a convenient way to avoid the need to redefine many variables when defining stages within pipelines, and even when working across multiple pipelines. Let's create a variable group and see how it can be used.

1. On the main menu for the **Parts Unlimited** project, click **Pipelines**, then click **Library**. There are currently no variable groups in the project.



2. Click **+ Variable group** to commence creating a variable group. Set **Variable group name** to **Website Test Product Details**.



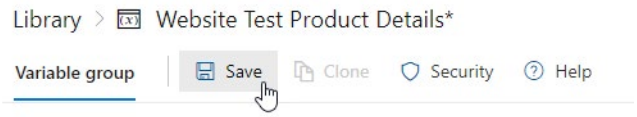
3. In the **Variables** section, click **+Add**, then in **Name**, enter **ProductCode**, and in **Value**, enter **RED-POLOXL**.

You can see an extra column that shows a lock. It allows you to have variable values that are locked and not displayed in the configuation screens. While this is often used for values like passwords, notice that there is an option to link secrets from an Azure key vault as variables. This would be a preferable option for variables that are providing credentials that need to be secured outside the project.

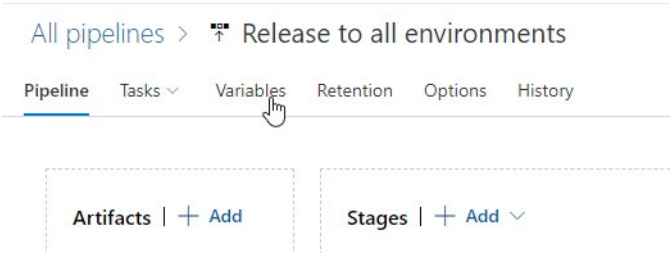
In this example, we are just providing details of a product that will be used in testing the website.

4. Add another variable called **Quantity** with a value of **12**.
5. Add another variable called **SalesUnit** with a value of **Each**.

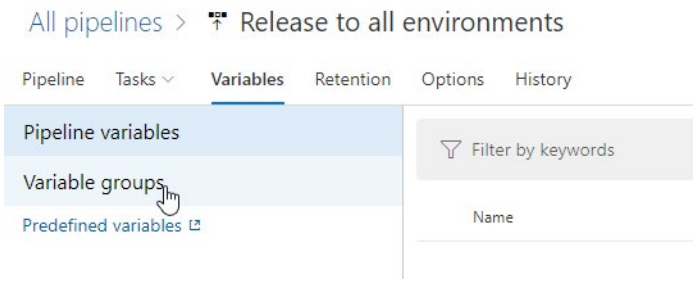
6. Click **Save** to save the new variable group.



7. On the main menu, click **Pipelines**, then click **Releases**, then click **Edit** to return to editing the release pipeline that we have been working on. From the top menu, click **Variables**.



8. In the left-hand pane, click **Variable Groups**.

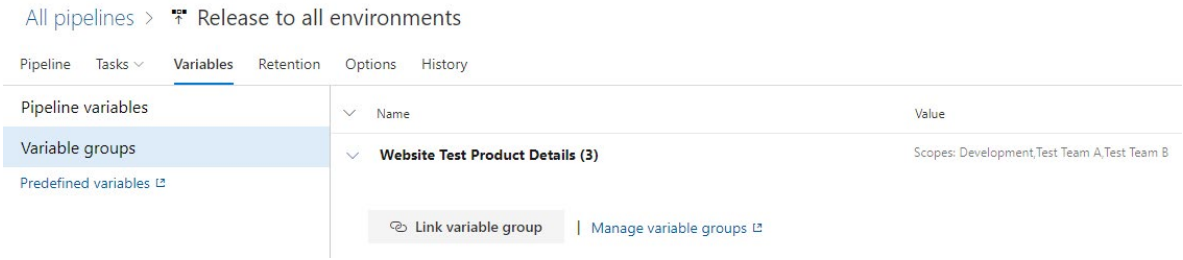


Variable groups are linked to pipelines, rather than being directly added to them.

9. Click **Link variable group** , then in the **Link variable group** pane, click to select the **Website Test Product Details** variable group (notice that it shows you how many variables are contained), then in the **Variable group scope**, select the **Development**, **Test Team A**, and **Test Team B** stages.

We need the test product for development and during testing but we do not need it in production. If it was needed in all stages, we would have chosen **Release** for the Variable group scope instead.

10. Click **Link** to complete the link.



The variables contained in the variable group are now available for use within all stages except Production, just the same way as any other variable.

Integrate Secrets with the Release Pipeline

Integrate Secrets within the release pipeline

When you deploy your applications to a target environment, there are almost always secrets involved.

- Secrets to access the target environment (servers, storage accounts)
- Secrets to access resources (connections strings, tokens, username/passwords)
- Secrets that your application uses (config files)

When your software is packaged as an artifact and moves through the different stages, secrets need to inserted during the execution of the pipeline. Mainly because secrets (should) differ between stages and environments but, just as important, secrets should NEVER be part of your source control repository.

There are different ways to deal with secrets in your pipelines. In this chapter, we walk through some options.

Using Service Connections

A Service Connection is a securely stored configuration of a connection to a server or service. By using this Service Connection, the pipeline tasks can execute securely against this endpoint. This way now credentials or secrets are needed as part of the release pipeline.

More about Service Connections[37]

Using secret variables

A straightforward and convenient way to add secrets to your pipeline is the use of secret variables. Earlier in this chapter we covered variables. By making the variable secret, it is hidden from view in all log files and unrecoverable.

Secret variables are often used to store secrets, connection strings and are used as replacement values in the pipeline. By creating placeholders in a target file, and replacing the placeholder with the real value in the pipeline, you create a secret free repository.

More about variables[38]

Storing secrets in a key vault

Another option of securing passwords and secrets is using a Keyvault. This is an excellent option because it allows you to keep the secrets outside of the pipeline and be able to retrieve them in another way (if you have the appropriate rights).

Retrieve with a variable group

To make use of this option within your build and release pipelines you need to create a variable group that refers to this KeyVault.

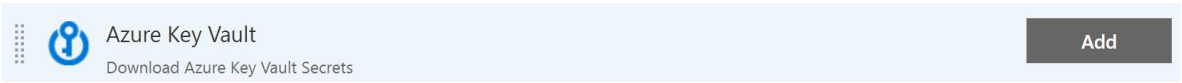
Read here how to link secrets from key vault in variable groups[39]

37 https://docs.microsoft.com/en-us/azure/devops/pipelines/library/service-endpoints?view=vsts
38 https://docs.microsoft.com/en-us/azure/devops/pipelines/process/variables?view=vsts
39 https://docs.microsoft.com/en-us/vsts/build-release/concepts/library/variable-groups?view=vsts#link-secrets-from-an-azure-key-vault-as-variables

and **here how to use them**[40]

Accessing Keyvault from within the pipeline

You can also access Keyvault variables without a variable group by using the dedicated build task.



- **Using Azure Key Vault Secrets for your VSTS Releases**[41]

Demonstration Setting Up Pipeline Secrets

In this demonstration, you will investigate Pipeline Secrets.

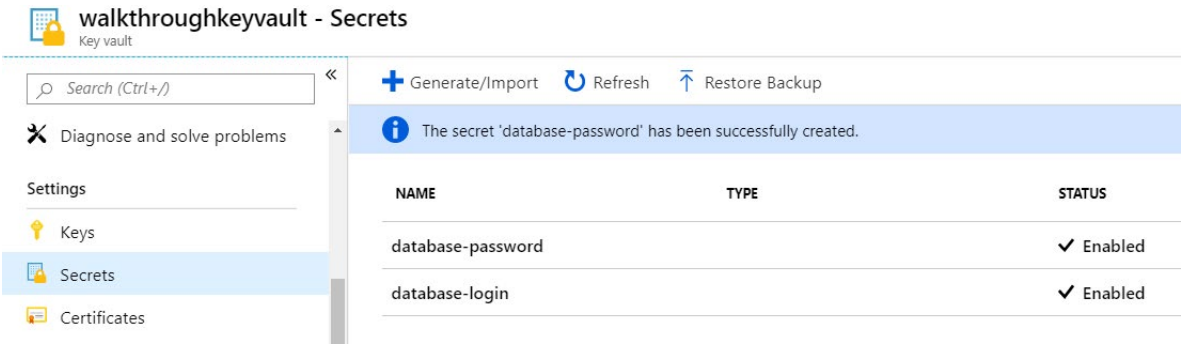
Note: Before starting this walkthrough, make sure you have performed the steps in the prerequisites section and the previous walkthroughs.

Steps

Let's now take a look at how a release pipeline can make use of the secrets that it needs during deployment.

In the walkthrough on variable groups, you saw that variable groups can work in conjunction with Azure Key Vault.

1. In the Azure Portal, in the key vault (that was set up in the pre-requisites section) properties, in the **Secrets** section, create the following secrets using the **+Generate/Import** option with an **Upload option** of **Manual**.

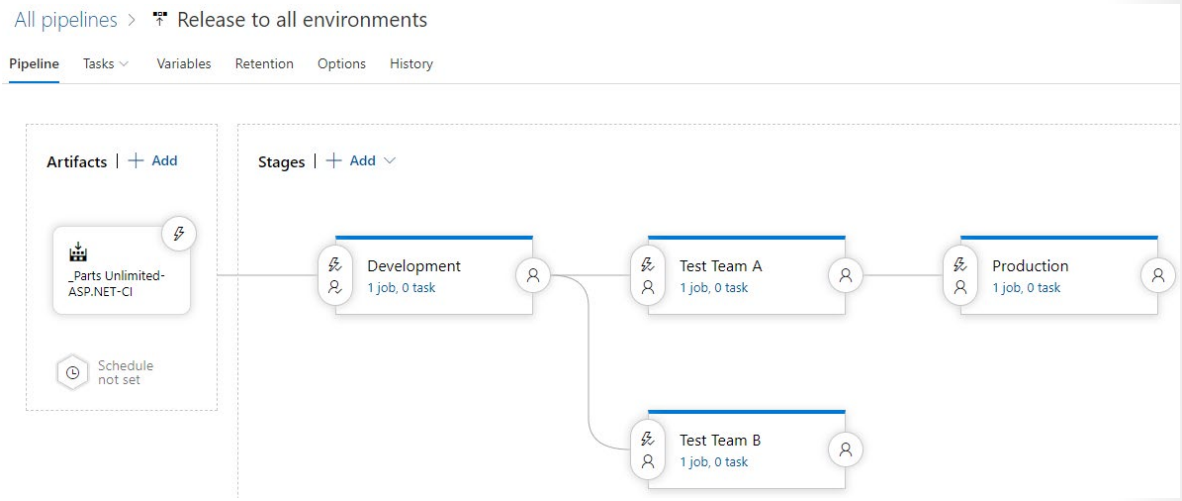


Adding secrets by using an Azure Key Vault task in a stage

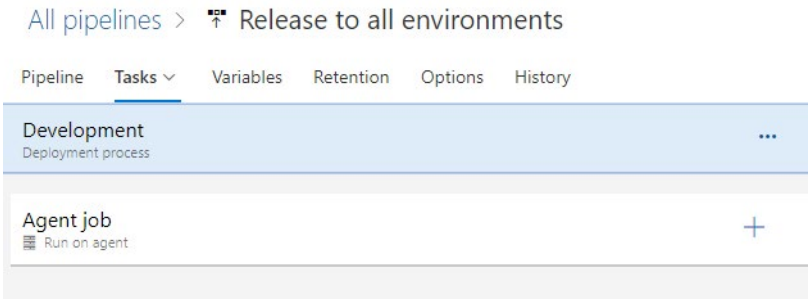
There are several ways that secrets from an Azure Key Vault can be used. The first option is to add an Azure Key Vault task to a stage.

2. In Azure DevOps, in the main menu for the **Parts Unlimited** project, click **Pipelines**, then click **Releases**, then click **Edit** to open the editing screen for the release pipeline that was created in earlier walkthroughs.

40 https://docs.microsoft.com/en-us/vsts/build-release/concepts/library/variable-groups?view=vsts#use-a-variable-group
41 http://www.lfraile.net/2018/02/using-azure-key-vault-secrets-for-your-vsts-releases/



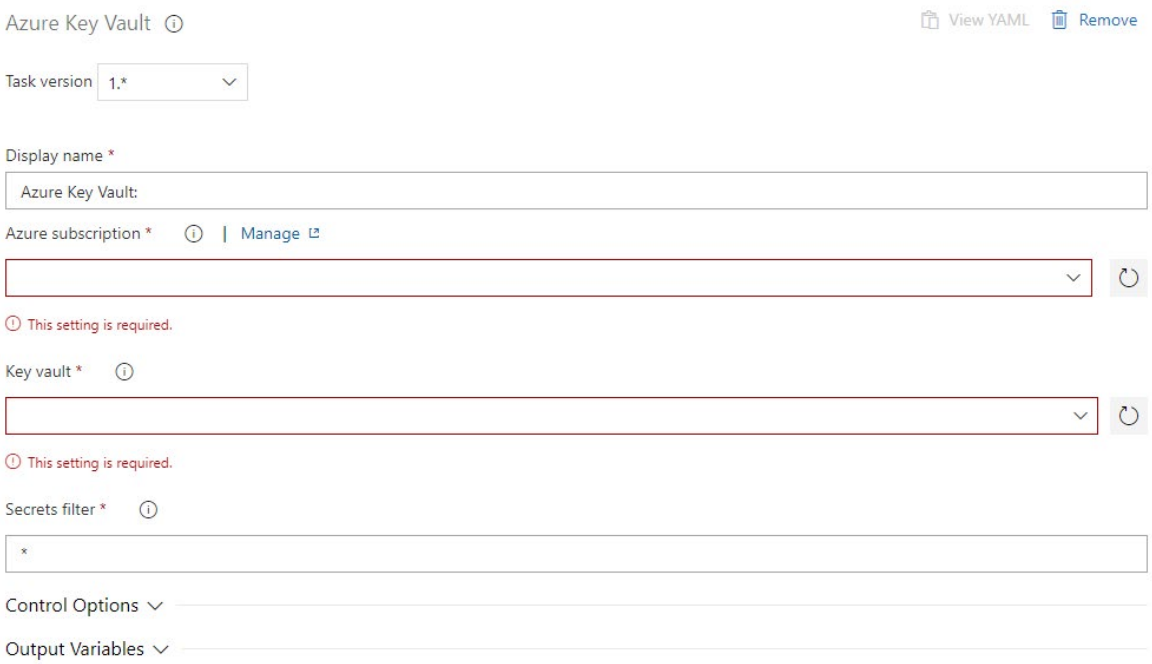
3. From the **Tasks** drop down list, click **Development** to open the tasks editor for the Development stage.



No tasks have been defined as yet.

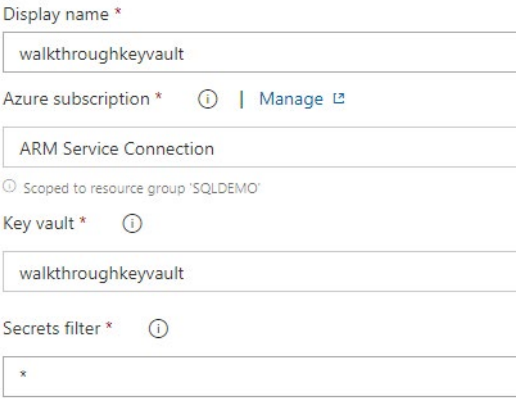
4. Click the **+** sign to the right hand side of **Agent job** to add a new task. In the **Search** box, type **vault**.

5. Hover over the **Azure Key Vault** task and when **Add** appears, click it, then in the task list, click the **Azure Key Vault** task to open its settings.



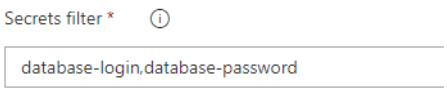
We can use the same service connection that was used in earlier walkthroughs.

6. Set **Display name** to the name of your key vault, and from the **Azure subscription** drop down list, select **ARM Service Connection**. In the **Key vault** drop down list, select your key vault.



The **Secrets filter** can be used to define which secrets are required for the project. The default is to bring in all secrets in the vault. It will be more secure to restrict the secrets to just those that are needed.

7. In the **Secrets filter**, enter **database-login,database-password**.



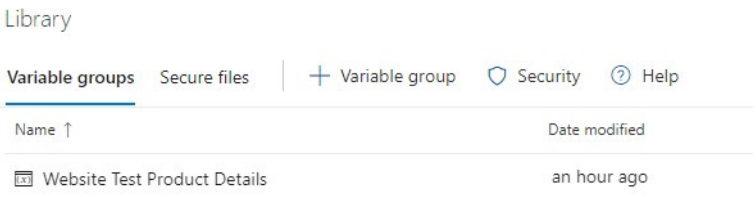
8. Click **Save** then **OK** to save your work.

The secrets can now be used like any other variables. You can reference them as $(database-login) and $(database-password) when configuring tasks.

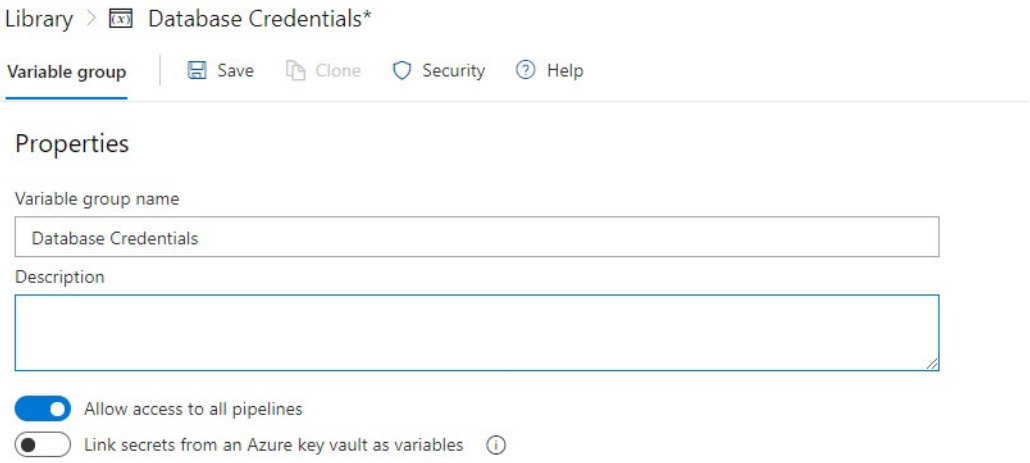
Adding secrets by using variable groups

Another way to configure Azure Key Vault secrets is by linking them to variable groups.

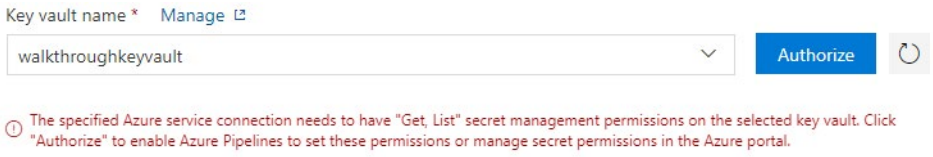
9. Click **Remove** to remove the Azure Key Vault task, then click **Save** and **OK**.
10. In the main menu, click **Pipelines**, then **Library**. The existing variable group is shown.



11. Click **+Variable group** to add a new variable group.



12. Click **Link secrets from an Azure key vault as variables**. In the **Azure subscription** drop down list, click **ARM Service Connection**, and from the **Key vault name** drop down list, select your key vault.

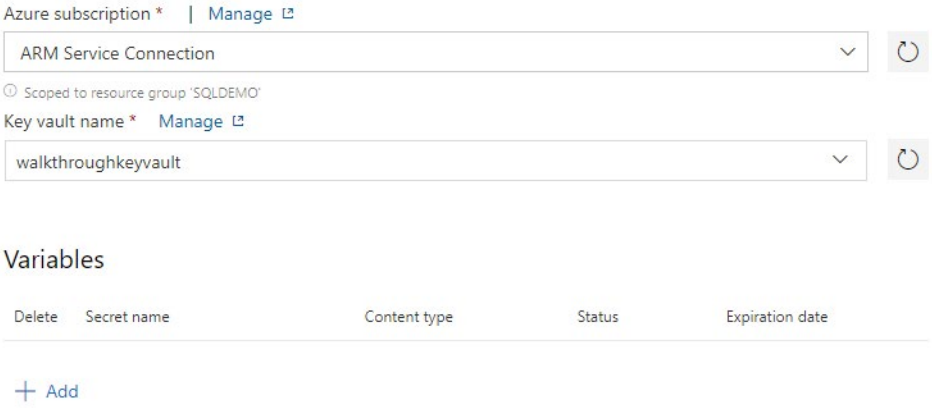


Note the warning that appears. Any service principal that needs to list secrets or get their values, needs to have been permitted to do so, by creating an access policy. Azure DevOps is offering to configure this for you.

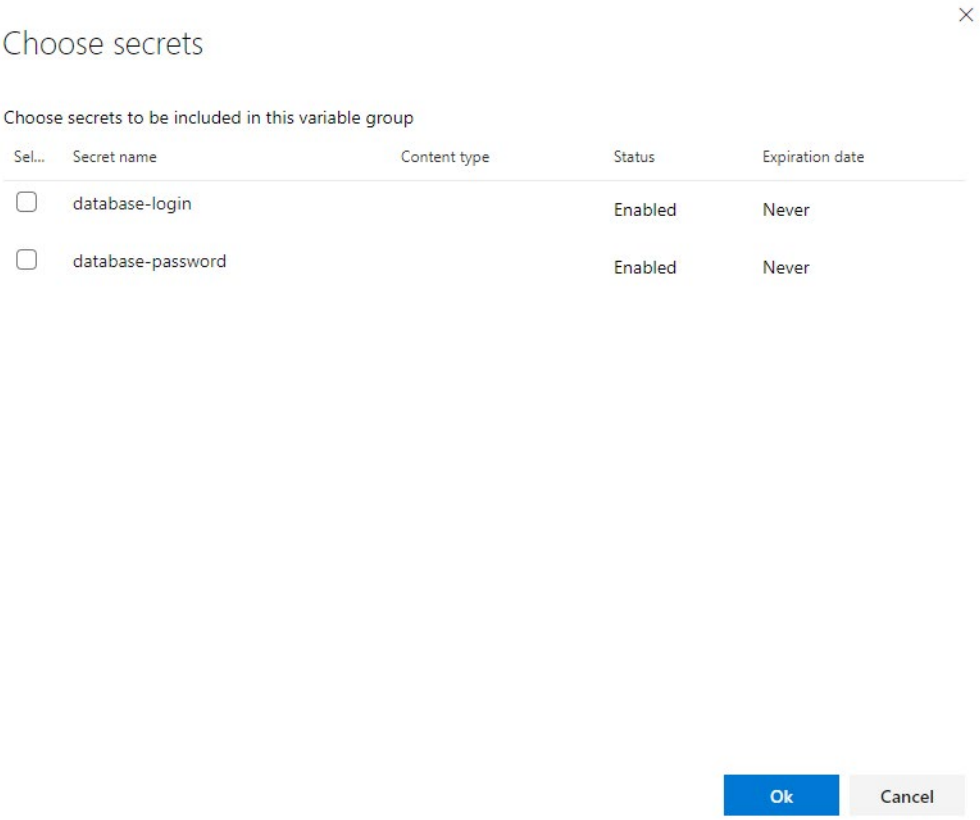
12. Click **Authorize** to create the required access policy.

Note: you will be required to log on to Azure to perform this action

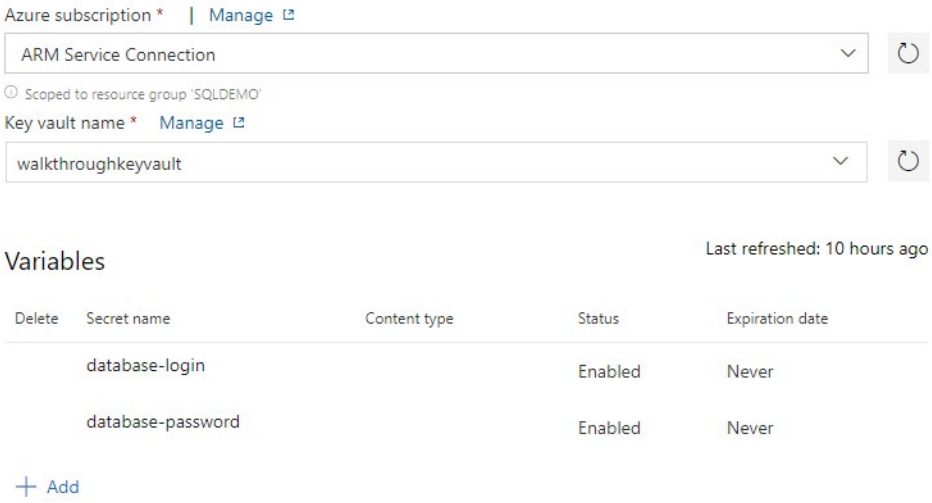
The warning should then disappear.



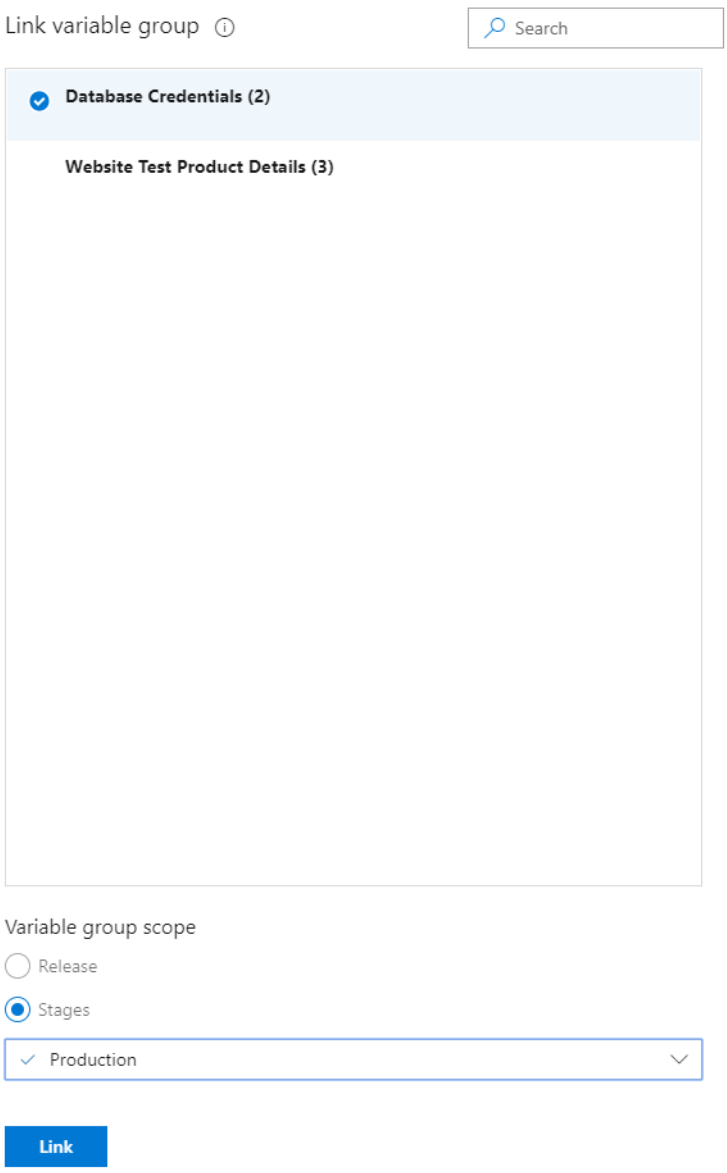
13. Click **+Add** to start to add the secrets.



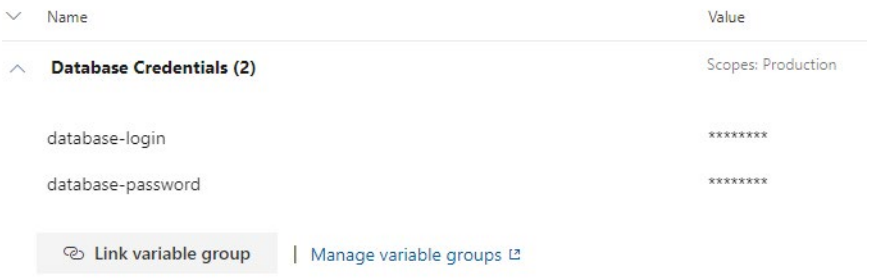
14. Click to select both database-login and database-password secrets, then click **Ok**.



15. Click **Save** to save the variable group.
16. In the main menu, click **Pipelines**, then **Releases**, then **Edit** to return to editing the release pipeline. Click **Variables**, then **Variable groups**, then **Link variable group**.
17. In the **Link variable group** pane, click to select **Database Credentials**, then from the **Stages** drop down list, select **Production**.



18. Click **Link** to link the variable group to the Production stage of the pipeline. Click the drop down beside **Database Credentials** so that you can see the variables that are now present.



19. Click **Save** then **OK**.

An important advantage of using variable groups to import Azure Key Vault secrets, over adding the Azure Key Vault task to a stage, is that they can be scoped to more than one stage in the pipeline, and linked to more than one pipeline.

Lab: Setting up secrets in the pipeline with Azure Key vault

Overview

You can use the Azure Key vault as a storage place for your secrets, keys, and certificates. You can benefit from this safe storage in your release pipelines by using the Azure Keyvault task or creating a variable group pointing to the Azure Keyvault.

What's covered in this lab?

This lab covers the configuration of the Azure Keyvault and the use of the secrets in the Azure Key vault in your release pipeline. We will show two different methods

- Using the Azure KeyVault Build/Release task
- Using the Variable Group linked to an Azure Key vault.

Before you begin

Refer the **Azure DevOps Labs Getting Started**[42] page to know the prerequisites for this lab.

Click the **Azure DevOps Demo Generator**[43] link and select the PartsUnlimited Demo project.

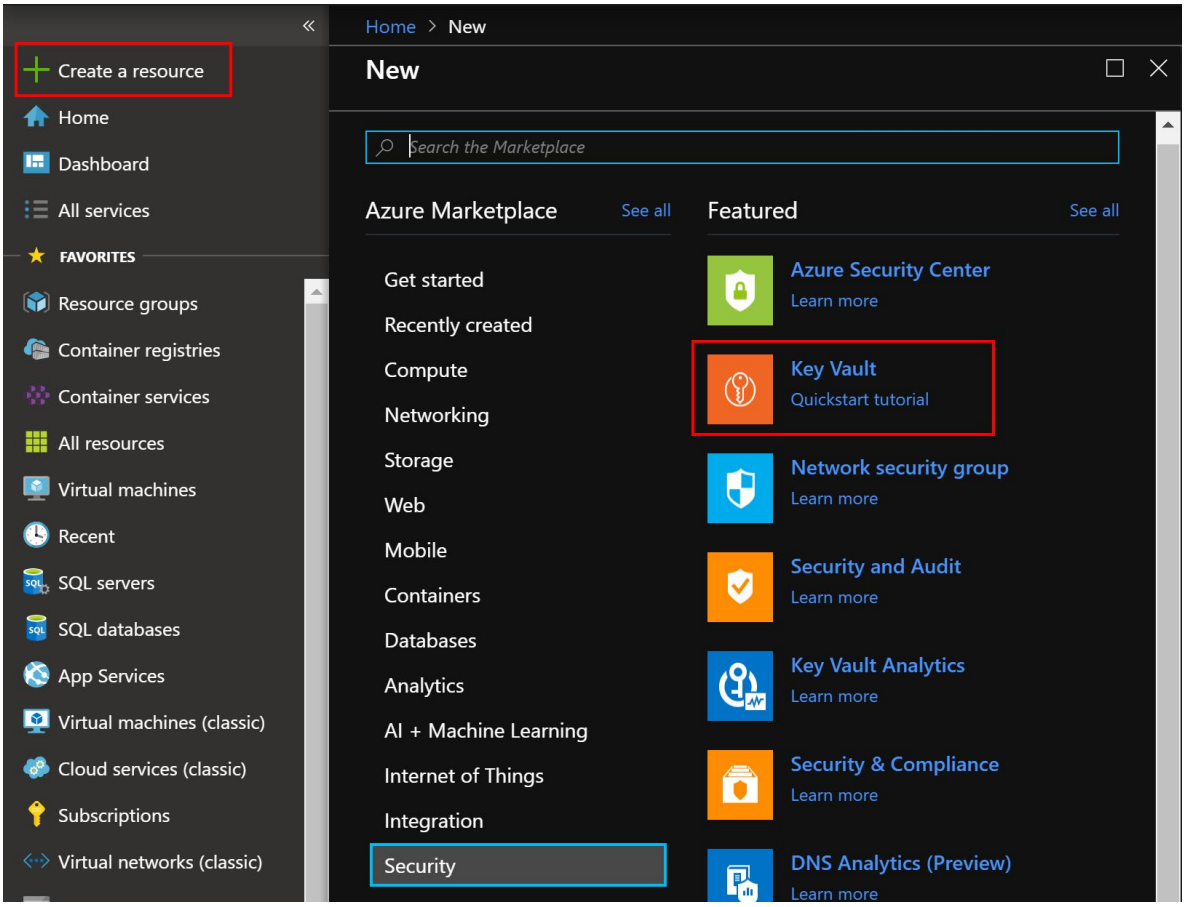
Setting up the Target Environment

You will create one resource group and an Azure Key vault. In the Azure Key vault, you will create two secrets.

You need to have an empty Azure DevOps project or the PartsUnlimited Demo project.

1. Go to Azure portal and click on +Create a resource and search and select Key Vault.

42 https://azuredevopslabs.com/labs/vstsextend/Setup/
43 https://azuredevopsdemogenerator.azurewebsites.net/Environment/CreateProject



2. Provide a name for the Key vault, create new Resource Group or select existing one from the drop-down and click Create.
3. Once the deployment succeeds, navigate to your Resource Group to see the resources created.
4. In the Azure Key vault, navigate to **Secrets**, select **+Generate/Import**, and create a secret. Add the name **password**. The value is random and up to you and select the **create** button.

Exercise 1: Configure Release pipeline with Azure Key vault build/release task

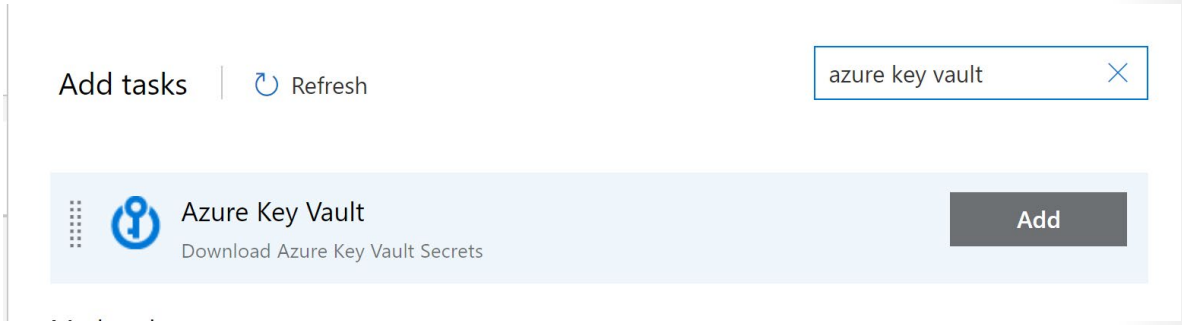
Setup Git Repo

1. Navigate to **Repos**, select the existing Repo and create a **New repository**. Call it secrets. **Initialize** the repository with a ReadMe.md file
2. Add a **New File** to the repository. Call it secrets.config.
3. Add the following lines of code to the file.

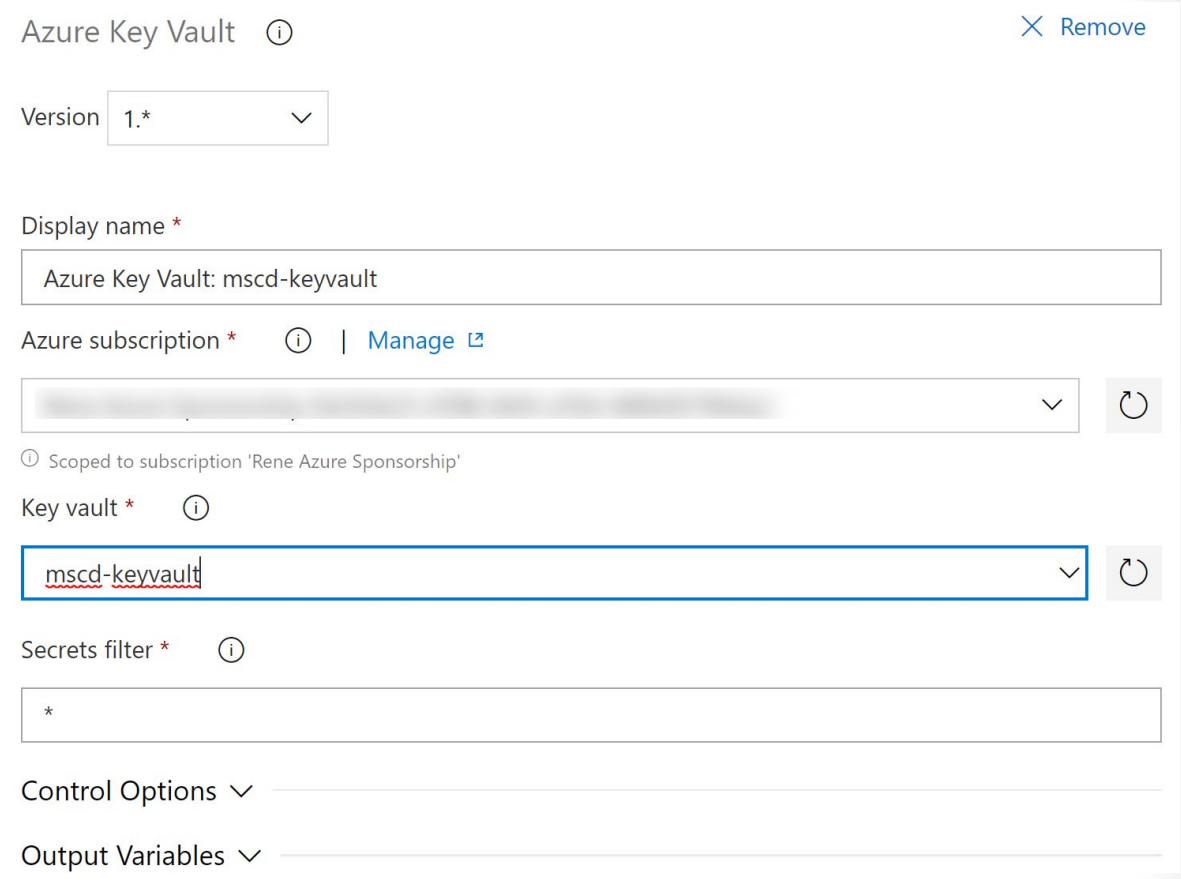
```
{
    "username": "user123",
    "password": "#{password}#"
}
```

Setup the pipeline

1. Navigate to Releases under Pipelines and Create a **New Release Pipeline**. Select **Empty job**.
2. **Add an Artifact** and select the **secrets Git Repo**
3. Add tasks to **Stage 1**. Add a Task and select the **Azure Key Vault** Task.



4. Click the task, select the **Azure Subscription** and select the **Key vault**.



5. Add a **Replace Tokens task**[44]. If it is not yet installed, install it first.
6. Run the release. The `#{password}#` token has been replaced with the secret from the Azure Key vault.

44 https://marketplace.visualstudio.com/items?itemName=qetza.replacetokens&targetId=4590486a-765d-48ea-a3f2-0772f9bcaefc

Exercise 2: Configure Release pipeline with variable group

1. Navigate to **Library** under Pipelines and Create a **+Variable Group**. Name it **secret** and **Link secrets from an Azure Key vault**
2. Select the **Azure Subscription** and select the **Key vault**
3. **Add** the secret as variable and **Save** the variable group

Setup the pipeline

1. Navigate to Releases under Pipelines and Create a **New Release Pipeline**. Select **Empty job**.
2. **Add an Artifact** and select the **secrets Git Repo**
3. Add a **Replace Tokens task**[45]. If it is not yet installed, install it first.
4. Go to **variables** and select **Variable Groups**. **Link variable group** to the *secret** variable group
5. Run the release. The `#{password}#` token has been replaced with the secret from the Azure Key vault

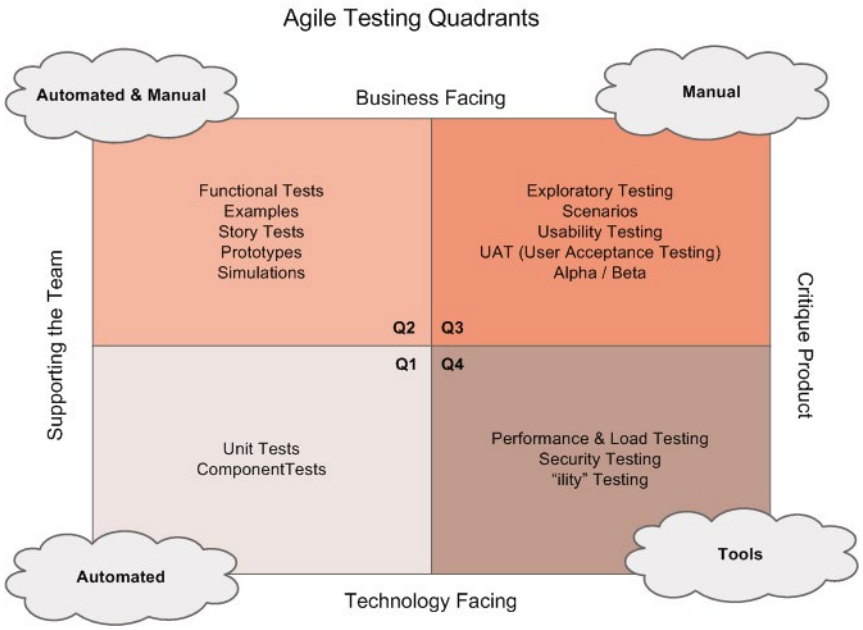
45 https://marketplace.visualstudio.com/items?itemName=qetza.replacetokens&targetId=4590486a-765d-48ea-a3f2-0772f9bcaefc

Configure Automated Integration and Functional Test Automation

Configure Automated Integration and Functional Test Automation

The first thing that comes to mind when talking about Continuous Delivery is that everything needs to be automated. Otherwise, you cannot deploy multiple times a day. But how to deal with testing then? Many companies still have a broad suite of manual tests that need to run before delivering to production. Somehow these tests need to run every time a new release is created.

Instead of automating all your manual tests into automated UI tests, you need to rethink your testing strategy. As Lisa Crispin describes in her book Agile Testing, you can divide your tests into multiple categories.



>source: http://lisacrispin.com/2011/11/08/using-the-agile-testing-quadrants/

We can make four quadrants where each side of the square defines what we are targeting with our tests.

- Business facing, meaning the tests are more functional and most of the time executed by end users of the system or by specialized testers that know the problem domain very well.
- Supporting the Team, means it helps a development team to get constant feedback on the product so they can find bugs fast and deliver a product with quality build in
- Technology facing, means the tests are rather technical and non-meaningful to business people. They are typical tests written and executed by the developers in a development team.
- Critique Product, are tests that are there to validate the workings of a product on it's functional and non-functional requirements.

Now we can place different test types we see in the different quadrants.
e.g., we can put functional tests, Story tests, prototypes and simulations in the first quadrant. These tests are there to support the team in delivering the right functionality and are business facing since they are more functional.

In quadrant two we can place tests like exploratory tests, Usability tests, acceptance tests, etc.

In quadrant three we place tests like Unit tests, Component tests, and System or integration tests.
In quadrant four we place Performance tests, load tests, security tests, and any other non-functional requirements test.

Now if you look at these quadrants, you can see that specific tests are easy to automate or are automated by nature. These tests are in quadrant 3 and 4
Tests that are automatable but most of the time not automated by nature are the tests in quadrant 1
Tests that are the hardest to automate are in quadrant 2

What we also see is that the tests that cannot be automated or are hard to automate are tests that can be executed in an earlier phase and not after release. This is what we call shift-left where we move the testing process more towards the development cycle.

We need to automate as many tests as possible. And we need to test as soon as possible. A few of the principles we can use are:

- Tests should be written at the lowest level possible
- Write once, run anywhere including production system
- Product is designed for testability
- Test code is product code, only reliable tests survive
- Test ownership follows product ownership

By testing at the lowest level possible, you will find that you have a large number of tests that do not require infrastructure or applications to be deployed. For the tests that need an app or infrastructure, we can use the pipeline to execute them.

To execute tests within the pipeline, we can run scripts or use tools that execute certain types of tests. On many occasions, these are external tools that you execute from the pipeline, like Owasp ZAP, SpecFlow, or Selenium. In other occasions, you can use test functionality from a platform like Azure. For example Availability or Load Tests that are executed from within the cloud platform.

When you want to write your own automated tests, choose the language that resembles the language from your code. In most cases, the developers that write the application should also write the test, so it makes sense to use the same language. For example, write tests for your .Net application in .Net, and write tests for your Angular application in Angular.

To execute Unit Tests or other low-level tests that do not need a deployed application or infrastructure, the build and release agent can handle this. When you need to execute tests with a UI or other specialized functionality, you need to have a Test agent that can run the test and report the results back. Installation of the test agent then needs to be done up front, or as part of the execution of your pipeline.

Setting up Test Infrastructure

Installing the test agent

To execute tests from within the pipeline, you need an agent that can run these tests. An agent can run on another machine and can be installed as part of the pipeline.

More information about setting up a Test Environment[46]

Running in the cloud

Hosted agents run the test agent, which enable you to run tests "in the cloud."

For Load Testing, you can also run from the cloud. This enables you to create a large set of users and test your application without having to set up your infrastructure.

Read more about Load Testing[47]

Lab: Setting up and Running Load Tests

How to set up a Load Test

In this video Donovan Brown will introduce you to Load Testing capability powered by Microsoft Azure and Azure DevOps. You'll see how to get started to test your website or API by taking advantage of various degrees of customization to your load test in Azure, Azure DevOps and even Visual Studio for the most flexibility.
Channel 9[48]

For more information:

- **Load Testing using Azure DevOps**[49]
- **Load Testing using Azure**[50]

Setting up a Load Test

There are many resources that you can use to start setting up your Load Test

- **Getting Started**[51]
- **Run URL-based load tests with Azure DevOps**[52]
- **Run Apache JMeter load tests with Azure DevOps**[53]
- **Performance test your Azure web app under load**[54]

Running a load test in the pipeline

In this lab, you will be introduced to the Web performance and Load testing capabilities provided in Visual Studio Enterprise 2017. You will walk through a scenario using a fictional online storefront where your goal is to model and analyze its performance with a number of simultaneous users. This will involve the definition of web performance tests that represent users browsing and ordering products, the definition of a load test based on the web performance tests, and finally the analysis of the load test results.

46 https://docs.microsoft.com/en-us/azure/devops/pipelines/test/set-up-continuous-test-environments-builds?view=vsts
47 https://docs.microsoft.com/en-us/azure/devops/test/load-test/overview?view=vsts
48 https://youtu.be/Qqh2OqB0wng
49 https://docs.microsoft.com/en-us/azure/devops/test/load-test/get-started-simple-cloud-load-test?view=vsts
50 https://docs.microsoft.com/en-us/vsts/load-test/app-service-web-app-performance-test?view=vsts
51 https://docs.microsoft.com/en-us/azure/devops/test/load-test/get-started-simple-cloud-load-test?view=vsts
52 https://docs.microsoft.com/en-us/azure/devops/test/load-test/get-started-jmeter-test?view=vsts
53 https://docs.microsoft.com/en-us/azure/devops/test/load-test/app-service-web-app-performance-test?view=vsts
54 https://visualstudio.microsoft.com/team-services/pricing/

Follow instructions here[55]

Lab: Setting up and Running Functional Tests

[HandsOn] Running a functional test in the pipeline

Selenium is a portable open source software-testing framework for web applications. It has the capability to operate on almost every operating system. It supports all modern browsers and multiple languages including .NET (C#), Java.

In this lab, you will learn how to execute selenium testcases on a C# web application, as part of the Azure DevOps Release pipeline.

Follow instructions here[56]

Linking Test Cases to your pipeline

When you already have a large amount of manual tests in test cases, you can also start using the benefits of running the test cases as part of your pipeline. When you automate a test case, the automation bit can be linked to the test case.

By attaching automation to the test case, the results are stored in the test case but also as part of your release pipeline.

- **Associate automated tests with test cases**[57]
- **Run automated tests from test plans**[58]

Additional Links

- **Behavior-Driven Design with SpecFlow**[59]
- **UI test with Selenium**[60]
- **Getting Started with Selenium Testing in a Continuous Integration Pipeline with Visual Studio**[61]

Setting up and Running Availability Tests

After you have deployed your web app or website to any server, you can set up tests to monitor its availability and responsiveness. It is useful to check if your application is still running and gives a healthy response.

Some applications have specific Health endpoints that can be checked by an automated process. The Health endpoint can be a simple HTTP status or a complex computation that uses and consumes crucial parts of your application.

For example, You can create a Health endpoint that queries the database. This way, you can check that your application is still accessible, but also the database connection is verified.

55 https://azuredevopslabs.com/labs/azuredevops/load/
56 https://www.azuredevopslabs.com/labs/vstsextend/Selenium/
57 https://docs.microsoft.com/en-us/azure/devops/test/associate-automated-test-with-test-case?view=vsts
58 https://docs.microsoft.com/en-us/azure/devops/test/run-automated-tests-from-test-hub?view=vsts
59 https://msdn.microsoft.com/en-us/magazine/dn296508.aspx
60 https://docs.microsoft.com/en-us/azure/devops/pipelines/test/continuous-test-selenium?view=vsts
61 https://blogs.msdn.microsoft.com/devops/2016/01/27/getting-started-with-selenium-testing-in-a-continuous-integration-pipeline-with-visual-studio/

You can create your own framework to create availability tests (ping test) or use a platform that can do this for you. Azure has the functionality to create Availability tests. You can use these tests in the pipeline and as release gates.

In Azure, you can set up availability tests for any HTTP or HTTPS endpoint that is accessible from the public internet. You don't have to add anything to the website you're testing. It doesn't even have to be your site: you could test a REST API service on which you depend.

There are two types of availability tests:

- URL ping test: a simple test that you can create in the Azure portal. You can check an URL and check the response and status code of the response
- Multi-step web test. These are a number of HTTP calls that are executed in sequence.

Read more about Availability Tests in Azure

- **Microsoft Docs**[62]
- **Monitor availability and responsiveness of any web site**[63]

62 https://azure.microsoft.com/nl-nl/blog/creating-a-web-test-alert-programmatically-with-application-insights/
63 https://docs.microsoft.com/en-us/azure/application-insights/app-insights-monitor-web-app-availability

Automate Inspection of Health

Automate Inspection of Health

Inspection of the release pipeline and release is something you should consider from the start.
When you run multiple deployments a day, you want to stay informed. You want to know whether a release passed or failed, you want to know the quality of the release, you want to know details about the release and how it performed, you want to stop releases when you detect something suspicious, and you want to visualize some of these things on a dashboard.

There are a few different things you can do to stay informed about your release pipeline in an automated fashion. In the following chapters, we will dive a bit deeper on these.

Release gates

Release gates allow automatic collection of health signals from external services and then promote the release when all the signals are successful at the same time or stop the deployment on timeout. Typically, gates are used in connection with incident management, problem management, change management, monitoring, and external approval systems.

Events, subscriptions, and notifications

Events are raised when certain actions occur, like when a release is started or a build completed.

A notification subscription is associated with a supported event type. The subscription ensures you get notified when a certain event occurs.

Notifications are usually emails that you receive when an event occurs to which you are subscribed.

Service Hooks

Service hooks enable you to perform tasks on other services when events happen in your Azure DevOps Services projects. For example, create a card in Trello when a work item is created or send a push notification to your team's Slack when a build fails. Service hooks can also be used in custom apps and services as a more efficient way to drive activities when events happen in your projects.

Reporting

Reporting is the most static approach when it comes to inspection, but in many cases also the most evident. Creating a dashboard which shows the status of your build and releases combined with team specific information is n many cases a valuable asset to get insights.

Read more about reporting in Azure DevOps[64]

Release gates

We talked about release gates earlier in the course. We can distinguish between 2 different types uses of release gates.

- As a quality gate. Check if the level of quality is still to decide whether or not the release can continue

64 https://docs.microsoft.com/en-us/azure/devops/report/dashboards/overview?view=vsts

- As an automatic approval. Check if certain things are done, before signing off to the next stage. This might in some cases be the same thing as a quality gate.

For example, In many organizations, there are so-called dependency meetings. This is a planning session where the release schedule of dependent components is discussed. Think of downtime of a database server or an update of an API. This takes a lot of time an effort, and the only thing that is needed is a signal if the release can proceed. Instead of having this meeting you can also create a mechanism where people press a button on a form when the release cannot advance. When the release starts, it checks the state of the gate by calling an API. If the "gate" is open, we can continue. Otherwise, we stop the release.

By using scripts and API's, you can create your own release gates instead of a manual approval. Or at least extending your manual approval. Other scenarios for automatic approvals are for example.

- Incident and issues management. Ensure the required status for work items, incidents, and issues. For example, ensure that deployment only occurs if no bugs exist.
- Notify users such as legal approval departments, auditors, or IT managers about a deployment by integrating with approval collaboration systems such as Microsoft Teams or Slack, and waiting for the approval to complete.
- Quality validation. Query metrics from tests on the build artifacts such as pass rate or code coverage and only deploy if they are within required thresholds.
- Security scan on artifacts. Ensure security scans such as anti-virus checking, code signing, and policy checking for build artifacts have completed. A gate might initiate the scan and wait for it to complete, or check for completion.
- User experience relative to baseline. Using product telemetry, ensure the user experience hasn't regressed from the baseline state. The experience level before the deployment could be considered a baseline.
- Change management. Wait for change management procedures in a system such as ServiceNow complete before the deployment occurs.
- Infrastructure health. Execute monitoring and validate the infrastructure against compliance rules after deployment, or wait for proper resource utilization and a positive security report.

In short, approvals and gates give you additional control over the start and completion of the deployment pipeline. They can usually be set up as a pre-deployment and post-deployment condition, that can include waiting for users to approve or reject deployments manually, and checking with other automated systems until specific requirements are verified. Besides, you can configure a manual intervention to pause the deployment pipeline and prompt users to carry out manual tasks, then resume or reject the deployment.

Release gates in Azure DevOps

A release gate is a concept that ships out of the Box in Azure DevOps. However, another tool can implement the same mechanism under a different name. When we take a look at the different types of release gate that ship out of the Box, you can already see that this can easily apply as well in any other tool.

You can configure a release gate before or after a stage and configure it in such a way that it evaluates the gate multiple times. For example, it can check the performance of your system, and when the gate fails, it can check again after a few minutes when the system warmed up.

For some examples around evaluation, **read this carefully**[65]

65 https://docs.microsoft.com/en-us/azure/devops/pipelines/release/approvals/gates?view=vsts#gate-evaluation-flow-examples

Please take a look at this video to see how release gates work in practice. The layout of Azure DevOps differs a bit since there is a new, but the functionality stays the same

- **Release Gates**[66]

To learn more about release gates

- **Release deployment control using gates**[67]
- **Invoke Azure Function task**[68]
- **Query Azure Monitor Alerts task**[69]
- **Query Work Items task**[70]
- **Publish To Azure Service Bus task**[71]
- **Invoke REST API task**[72]
- **Video: Taking control over your releases**[73]

Lab: Using Azure Monitor as release gate

As you may be aware, a release pipeline specifies the end-to-end release process for an application to be deployed across a range of environments. Deployments to each environment are fully automated by using phases and tasks. Ideally, you do not want new updates to the applications to be exposed to all the users at the same time. It is a best practice to expose updates in a phased manner i.e. expose to a subset of users, monitor their usage and expose to other users based on the experience the initial set of users had.

Approvals and gates enable you to take control over the start and completion of the deployments in a release. With approvals, you can wait for users to manually approve or reject deployments. Using release gates, you can specify application health criteria that must be met before release is promoted to the next environment. Prior to or after any environment deployment, all the specified gates are automatically evaluated until they all pass or until they reach your defined timeout period and fail.

Gates can be added to an environment in the release definition from the pre-deployment conditions or the post-deployment conditions panel. Multiple gates can be added to the environment conditions to ensure all the inputs are successful for the release.

This lab covers the configuration of the deployment gates and details how to add the control to Azure pipelines. You will configure a release definition with two environments for an Azure Web App. You will deploy to the Canary environment only when there are no blocking bugs for the app and mark the Canary environment complete only when there are no active alerts in Azure Monitor (Application Insights).

Follow instructions here[74]

66 https://channel9.msdn.com/Events/Connect/2017/T181
67 https://docs.microsoft.com/en-us/azure/devops/pipelines/release/approvals/gates?view=vsts
68 https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/utility/azure-function?view=vsts
69 https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/utility/azure-monitor?view=vsts
70 https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/utility/work-item-query?view=vsts
71 https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/utility/publish-to-azure-service-bus?view=vsts
72 https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/utility/http-rest-api?view=vsts
73 https://youtu.be/7WLcqwhTZ_4
74 https://azuredevopslabs.com/labs/vstsextend/releasegates/

Events, Subscriptions, and Notifications

One of the first requests many people have when working in a system that performs asynchronous actions is to have the ability to get notifications or alerts. Why? Because they do not want to open the application, log in and see if things changed over and over again.

The ability to receive Alerts and notifications is a powerful mechanism to get notified about certain events in your system at the moment they happen.

For example, when a build takes a while to complete, probably you do not want to stare to the screen until it has finished. But, you want to know when it does.

Getting an email or another kind of notification instead is very powerful and convenient. Another example is a system that needs to be monitored. You want to get notified by the system in real time. By implementing a successful alert mechanism, you can use alerts to react to situations before anybody is bothered by it proactively.

However, When you define alerts, you need to be careful. When you get alerts for every single event that happens in the system, your mailbox will quickly be flooded with a lot of alerts. The more alerts you get that are not relevant, the higher the change that people will never look at the alerts and notifications and will miss out on the important ones.

When defining alerts or notification, you need to think about the target audience. Who needs to react to the alerts? Do not send notifications to people for information only. They will stop looking at it very quickly.

Another thing to consider when defining alerts is the mechanism to deliver them. Do you want to send an email, or do you want to send a message in the Slack for your team? Or do you want to call another system to perform a particular action?

Within Azure DevOps, there are multiple ways to define your alerts. By using query and filter mechanisms, you can filter out specific alerts. For example, you only want to get notified for failed releases and not for successful ones.

Almost every action in the system raises an event to which you can subscribe to. A subscription is personal or for your whole team. When you have made a subscription, you can then select how you want the notification to be delivered.

Read more about notifications[75]

Events, subscriptions, and notifications[76]

Service Hooks

Service hooks enable you to perform tasks on other services when events happen in your Azure DevOps Services projects. For example, create a card in Trello when a work item is created or send a push notification to your team's mobile devices when a build fails. Service hooks can also be used in custom apps and services as a more efficient way to drive activities when events happen in your projects.

In Azure DevOps these Service Hooks are supported Out of the Box

Build and release	Collaborate	Customer support	Plan and track	Integrate
AppVeyor	Campfire	UserVoice	Trello	Azure Service Bus
Bamboo	Flowdock	Zendesk		Azure Storage

75 https://docs.microsoft.com/en-us/azure/devops/notifications/index?view=vsts
76 https://docs.microsoft.com/en-us/azure/devops/notifications/concepts-events-and-notifications?view=vsts

Build and release	Collaborate	Customer support	Plan and track	Integrate
Jenkins	HipChat			Web Hooks
MyGet	Hubot			Zapier
Slack				

To learn more about service hooks and how to use and create them read this webpage **Service Hooks in Azure DevOps**[77]

Demonstration Setting Up Service Hooks to Monitor the Pipeline

In this demonstration, you will investigate Service Hooks.

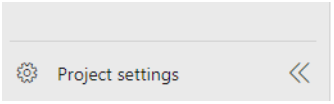
Note: Before starting this walkthrough, make sure you have performed the steps in the prerequisites section and the previous walkthroughs.

Steps

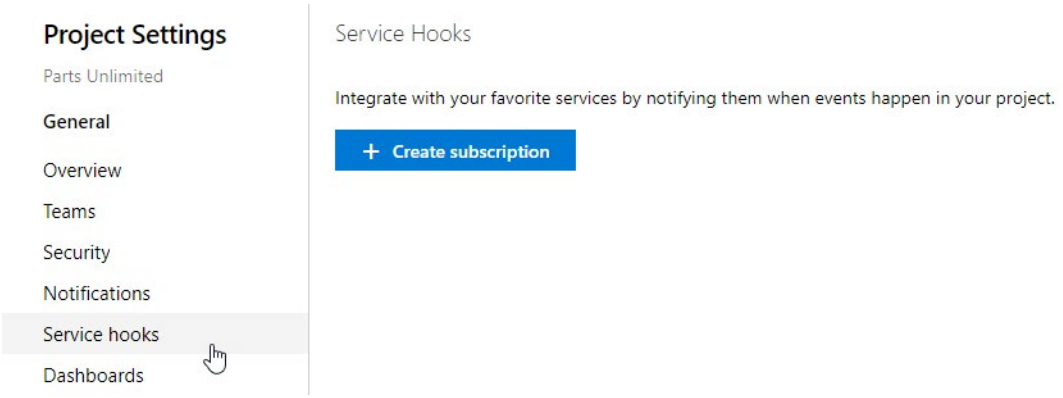
Let's now take a look at how a release pipeline can communicate with other services by using service hooks.

Azure DevOps can be integrated with a wide variety of other applications. It has built in support for many applications, and generic hooks for working with other applications. Let's take a look.

1. Below the main menu for the **Parts Unlimited** project, click **Project settings**.

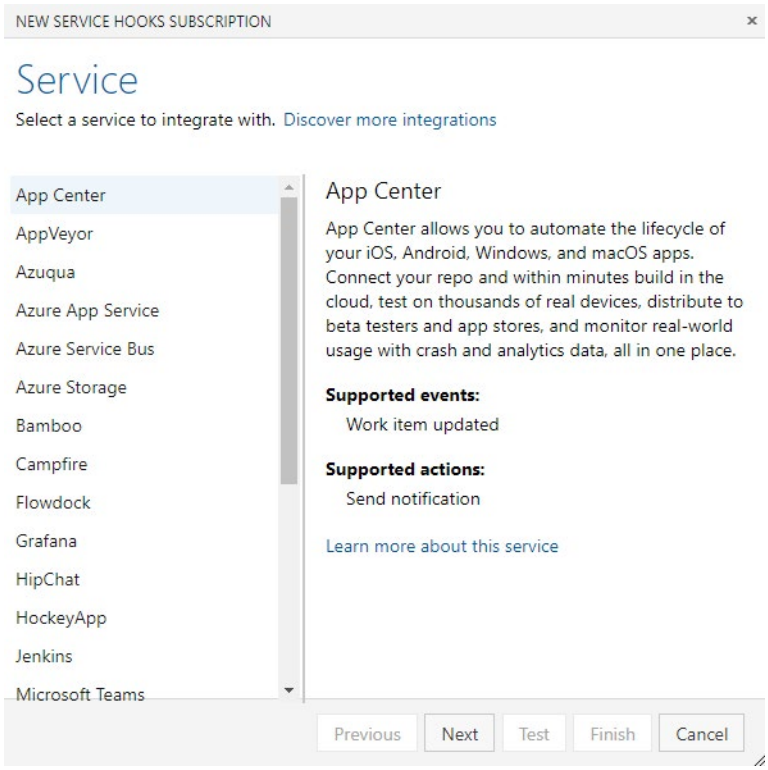


2. In the **Project settings** menu, click **Service hooks**.



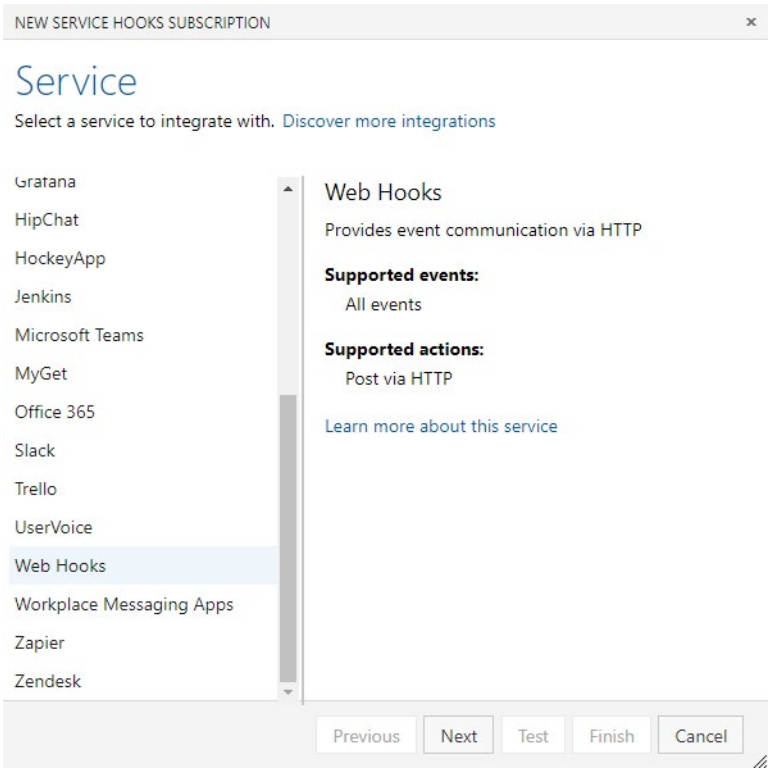
3. Click **+Create subscription**.

77 https://docs.microsoft.com/en-us/azure/devops/service-hooks/overview?view=vsts



By using service hooks, we can notify other applications that an event has occurred within Azure DevOps. We could also send a message to a team in **Microsoft Teams** or **Slack**. We could also trigger an action in **Bamboo** or **Jenkins**.

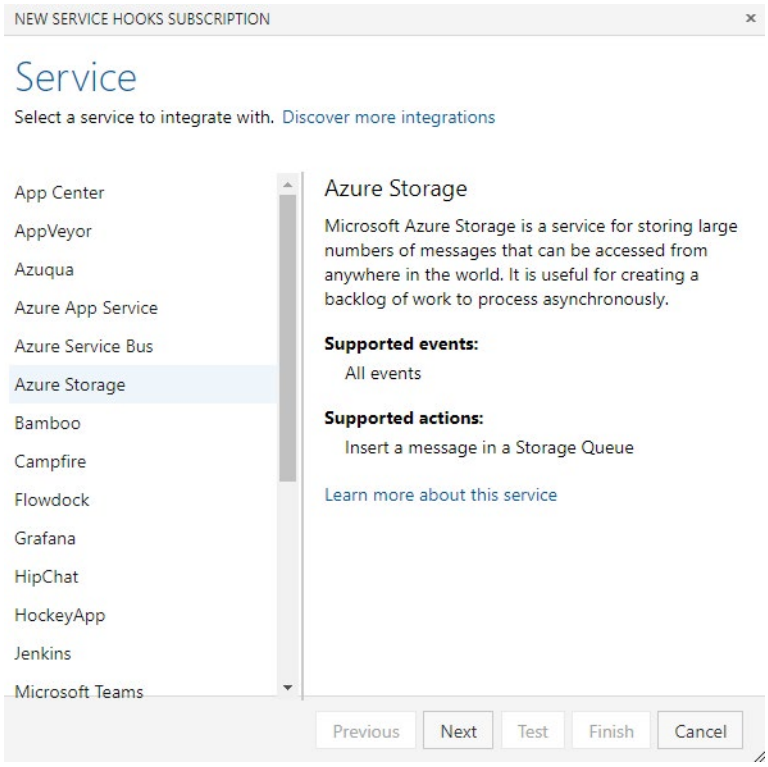
4. Scroll to the bottom of the list of applications and click on **Web Hooks**.



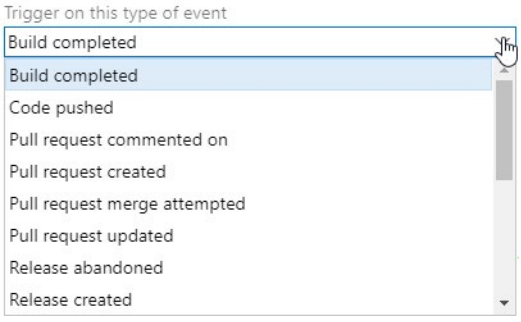
If the application that you want to communicate with isn't in the list of available application hooks, you can almost always use the **Web Hooks** option as a generic way to communicate. It allows you to make an HTTP POST when an event occurs. So, if for example, you wanted to call an Azure Function or an Azure Logic App, you could use this option.

To demonstrate the basic process for calling web hooks, we'll write a message into a queue in the Azure Storage account that we have been using.

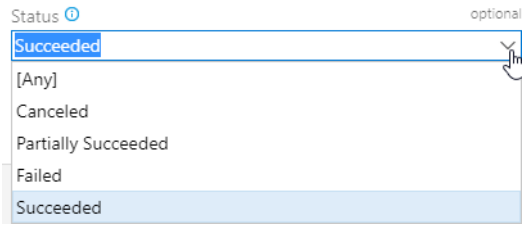
5. From the list of available applications, click **Azure Storage**.



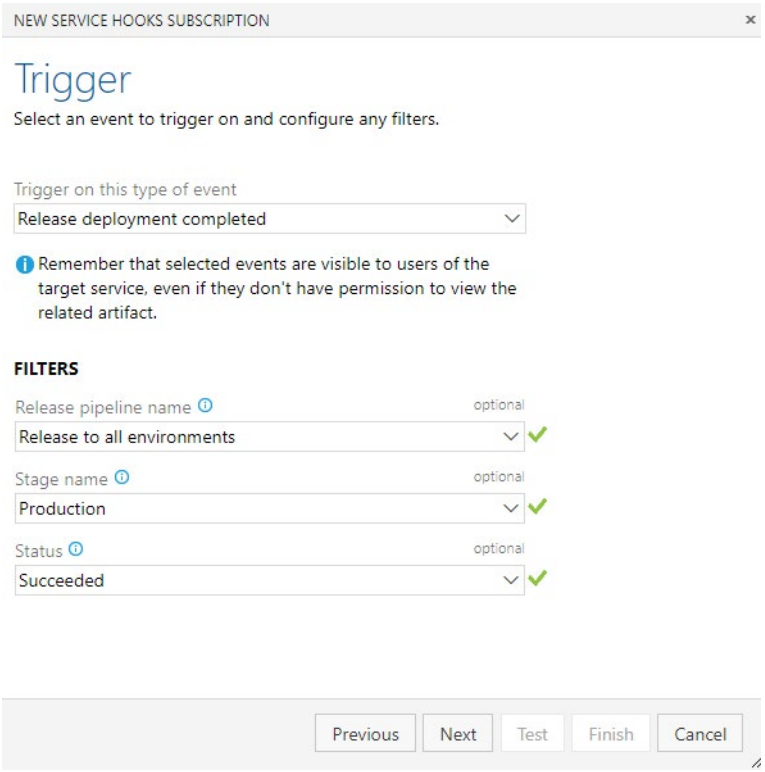
6. Click **Next**. In the **Trigger** page, we determine which event causes the service hook to be called. Click the drop down for **Trigger on this type of event** to see the available event types.



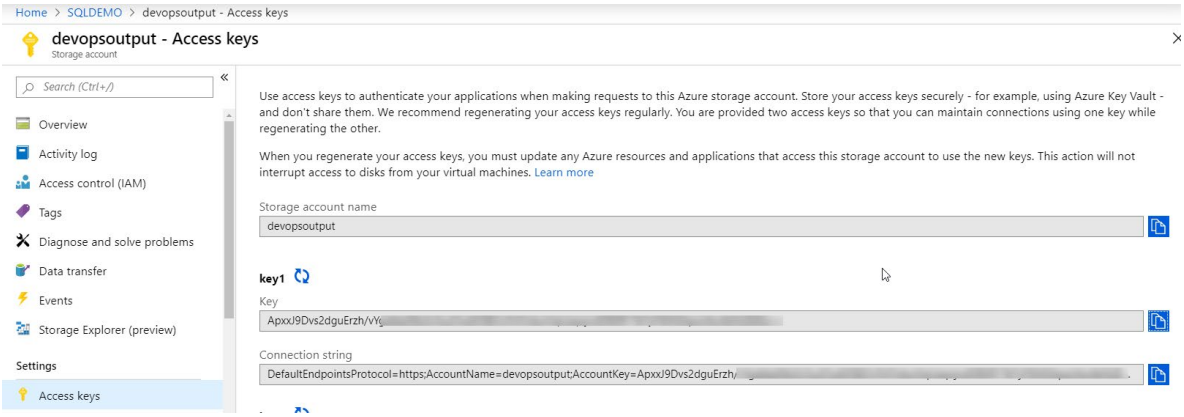
7. Ensure that **Release deployment completed** is selected, then in the **Release pipeline name** select **Release to all environments**. For **Stage**, select **Production**. Drop down the list for **Status** and note the available options.



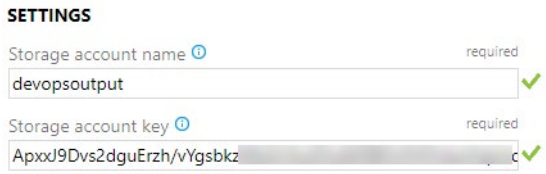
8. Ensure that **Succeeded** is selected, then click **Next**.



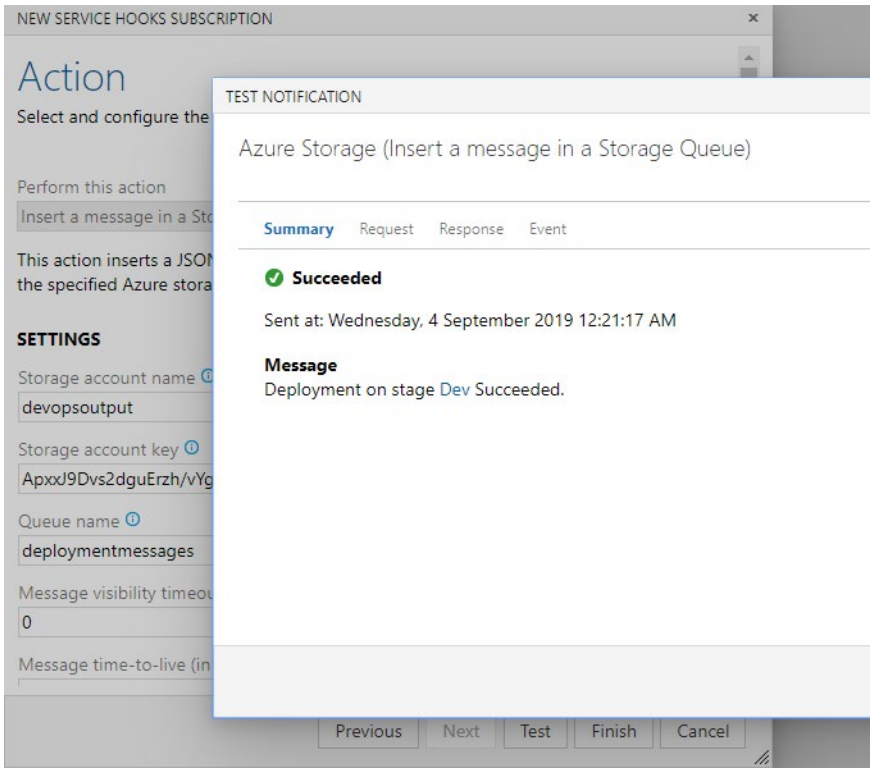
9. In the **Action** page, enter the name of your Azure storage account.
10. Open the Azure Portal, and from the settings for the storage account, in the **Access keys** section, copy the value for **Key**.



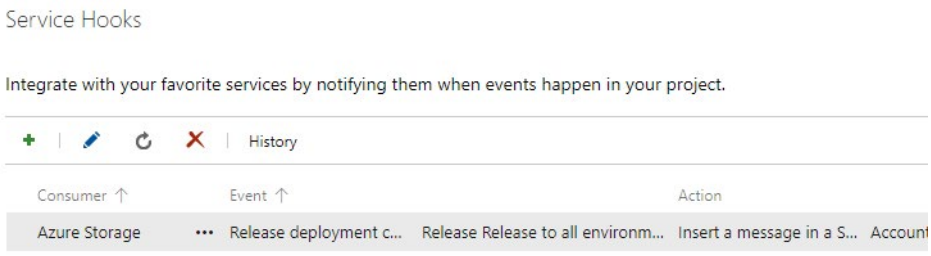
11. Back in the **Action** page in Azure DevOps, paste in the key.



12. For **Queue name** enter **deploymentmessages**, then click **Test**.



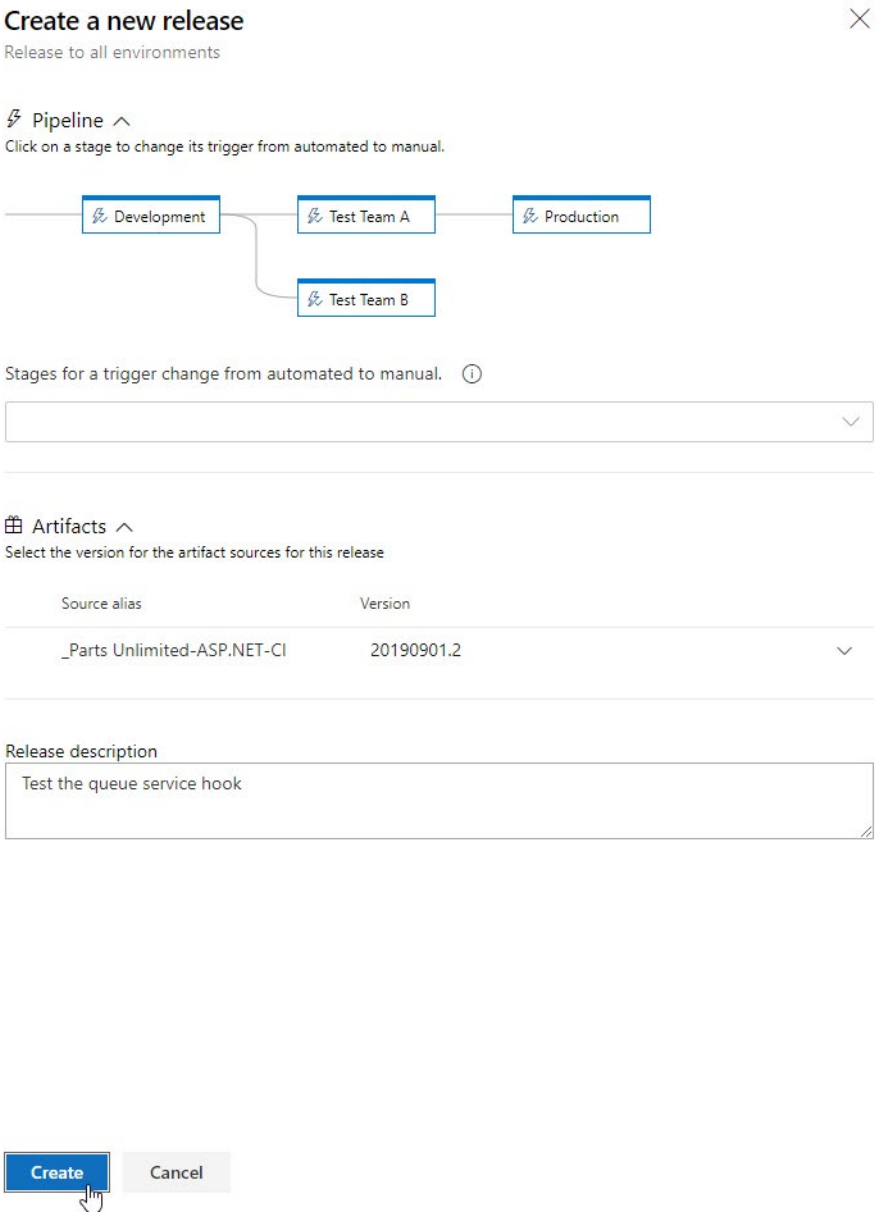
13. Make sure that the test succeeded, then click **Close**, and on the **Action** page, click **Finish**.



Create a release to test the service hook

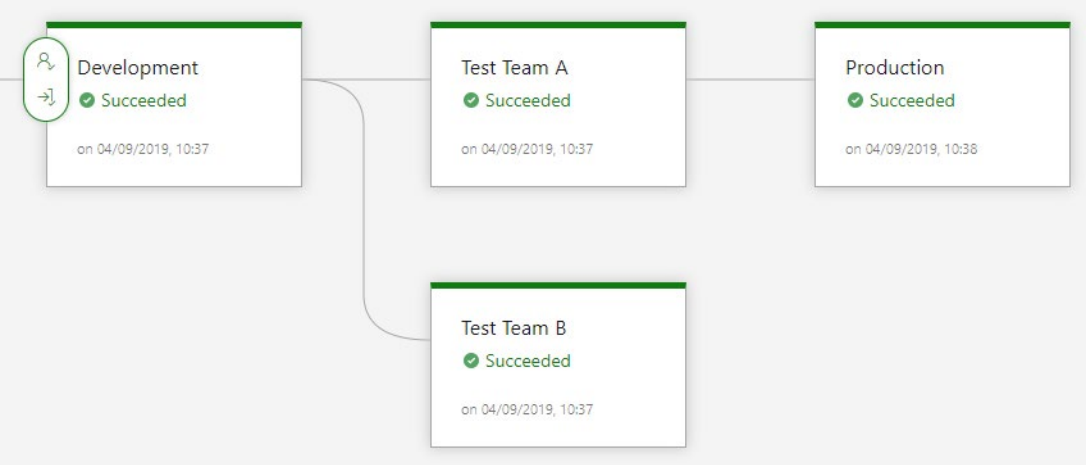
Now that you have successfully added the service hook, it's time to test it.

14. From the main menu of the **Parts Unlimited** project, click **Pipelines**, then click **Releases**, then click **Create release**, and in the **Create a new release** pane, enter **Test the queue service hook** for **Release description**, and click **Create**.



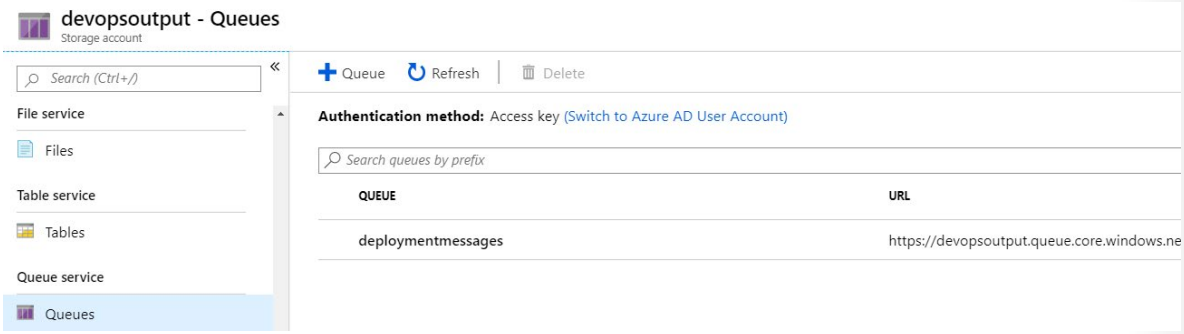
15. Click to view the release details.

16. If the release is waiting for approval, click to approve it and wait for the release to complete successfully.

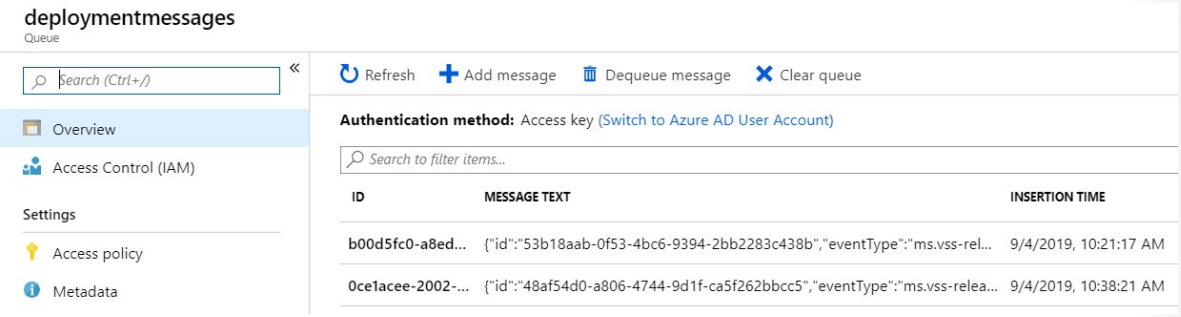


Check the queue contents

17. In the **Azure Portal**, in the blade for the storage account, click **Queues** from the **Queue service** section.



18. Click to open the **deploymentmessages** queue.



Note: if you have run multiple releases, you might have multiple messages

19. Click the latest message (usually the bottom of the list) to open it and review the message properties, then close the **Message properties** pane.

You have successfully integrated this message queue with your Azure DevOps release pipeline.

Lab: Creating a Release Dashboard

Overview

An essential part of setting up your release pipelines is able to have insights into them. By creating a Release dashboard that shows the relevant information, you can quickly see the status of your release and surrounding assets.

What's covered in this lab?

In this lab, we will cover the creation of a dashboard and some of the widgets you can use to gain insights. We will also take a look at the REST API to get information from Azure DevOps releases to use in your own applications or dashboards.

Before you begin

Refer the **Azure DevOps Labs Getting Started**[78] page to know the prerequisites for this lab.

78 https://azuredevopslabs.com/labs/vstsextend/Setup/

Complete **Task 1 from the Azure DevOps Lab Prerequisites**[79]

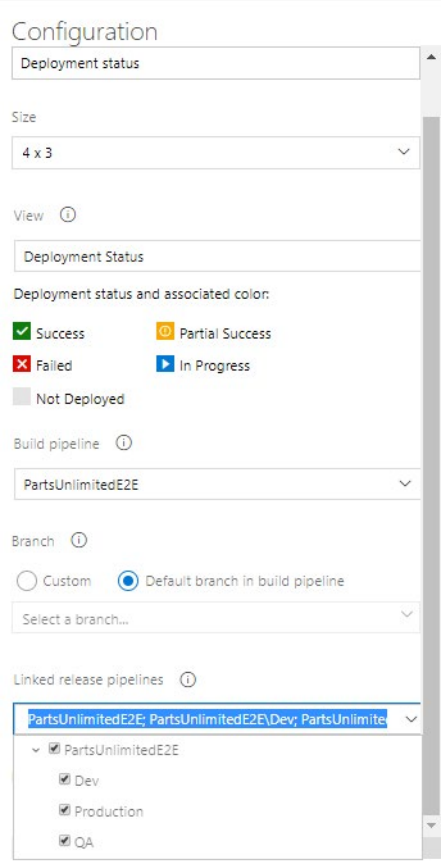
Setting up the Target Environment

You will run one build and one release to get some data in your project. Make sure you have an Azure account where you can deploy the application.

1. Navigate to **PipelinesReleases** and configure the Service Connections of all stages in the **PartsUnlimitedE2E**. Set the deployment slot name of the Dev Stage to *Dev* and the QA stage to *Staging*
2. In your Azure DevOps Project Go to **Pipelines** and click **Builds** and **Queue** a new **PartsUnlimitedE2E** build.
3. The application is built and deployed automatically

Exercise 1: Create a Release Dashboard

1. Navigate to **Dashboards** under Overview and Create a **+New Dashboard**. Name it **Releases**
2. Search the **Deployment Status** widget and add it to the dashboard
3. Click the configure icon, Set the size to *4x3 *, select *PartsUnlimitedE2E* build and all the stages of the Linked Release Pipeline and **Save** the configuration.



4. Search the **Release Pipeline Overview** widget and add it to the dashboard. Press the **Configure** icon and select the **PartsUnlimitedE2E** Release

79 https://azuredevopslabs.com/labs/azuredevops/prereq/#task-1-configuring-the-parts-unlimited-team-project

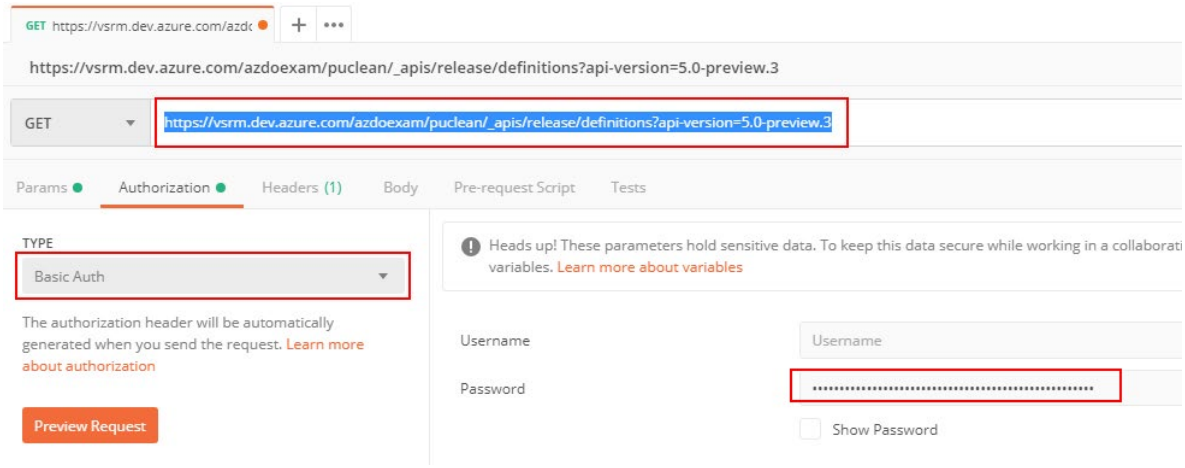
5. Search the **Build History** widget and add it to the dashboard. Press the **Configure** icon and select the **PartsUnlimitedE2E** Build
6. Install the **Team Project Health widget**[80] from the marketplace
7. Add the *Release Health Overview* and *Release Health Details* widget and configure them
8. Add the *Build Health Overview* and *Build Health Details* widget and configure them

Exercise 2: Use the REST API to get Release Information

In this exercise, we will use the tool Postman to show the use of the REST API. You can use any tool of your choice to call the REST API.

You can **download Postman here**[81]

1. Get a Personal Access Token by following the **steps described here**[82]
2. Use the URL `https://vsrm.dev.azure.com/{organization}/{project}/_apis/release/definitions?api-version=5.1` to get all Release Definitions.
3. In the authorization tab, select Basic Authentication and paste the Personal Access token in the password field



4. Press **Send** and look at the results

You can find documentation about the Full REST API **here**[83]

5. Try more calls

Links

- **https://docs.microsoft.com/en-us/azure/devops/report/dashboards/overview?view=vsts**
- **https://docs.microsoft.com/en-us/azure/devops/report/dashboards/widget-catalog?view=vsts**

80 https://marketplace.visualstudio.com/items?itemName=ms-devlabs.TeamProjectHealth
81 https://www.getpostman.com/downloads/
82 https://docs.microsoft.com/en-us/azure/devops/organizations/accounts/use-personal-access-tokens-to-authenticate?view=vsts
83 https://docs.microsoft.com/en-us/rest/api/azure/devops/?view=azure-devops-rest-5.0

Review Questions

Module 2 Review Questions

Question 1

How many deployment jobs can be run concurrently by a single agent?

Suggested Answer
One

Question 2

What should you create to store values that you want to make available across multiple build and release pipelines?

Suggested Answer
Variable group

Question 3

Which Azure-based tool can be used to safeguard cryptographic keys and other secrets used by cloud apps and services?

Suggested Answer
Azure Key Vault

Question 4

How can you provision the agents for deployment groups in each of your VMs?

Suggested Answer
One of the following:

Run the script that is generated automatically when a deployment group is created
Install the Azure Pipelines Agent Azure VM extension on each of the VMs
Use the Azure Resource Group Deployment task in your release pipeline

Question 5

How can you identify a default release variable?

Suggested Answer
It has Release. as its prefix (eg: Release.Reason)

Module 3 Implement an Appropriate Deployment Pattern

Introduction into Deployment Patterns

Implement an appropriate deployment pattern

Introduction to this module

This module is about implementing an appropriate deployment pattern. In the first module, we gave a general overview of some of the deployment patterns. In this module, we will talk about the different deployment patterns in more detail. We will start this module with a short introduction on architectural patterns and how they might affect your deployment pattern. The deployment patterns that we discuss cannot always be applied to every piece of software. To use a pattern, you might need to change parts of your architecture to enable this.

After this introduction, we will cover Blue-Green deployments. Deploying your new version of your application to a new environment where you can set it up, test it, validate it and then swap directly to production without any downtime.

Then we will talk about feature toggles (sometimes called feature flags, feature switch, etc.). Feature toggles are commonly used in modern deployment patterns. They are critical in the enablement of the separation of a functional and technical release. We covered this Module 1. We will discuss feature toggles in more detail in this module.

- What are feature toggles
- How can you implement them?
- What types of feature toggles are there?

We need to understand feature toggles correctly because they enable some of the other deployment patterns that we talk about.

Canary releases
Sometimes this is called Canary testing. Canary releases enable you to expose a feature to a small subset of users to make sure your application continues to behave correctly in a production scenario.

Dark Launching
This has some overlap with Canary releases. But there is a difference. A dark launch is about launching a feature to a group of users without announcing it or without visibility of the feature. This way you can measure how users react on a feature and how the system behaves.

A/B Testing
Then we will briefly talk about the concept of A/B Testing. Although A/B Testing is not something that is required to implement when doing Continuous Delivery, it is something that can be enabled by Continuous Delivery. A/B Testing is running experiments on a production server, with the primary target to increase the usage of a particular page or feature. By exposing the feature to only a subset of your population, statistical analysis can be done on what is successful and not.

Progressive exposure deployment
We will finalize this module by discussing Progressive exposure deployment, or, ring based deployment. Ring based deployment is gradually rolling out a feature to new users or environments to measure impact and reduce risk carefully.

Learning objectives

At the end of this module:

- You learned that architecture is a prerequisite for achieving Continuous Delivery
- You know different Deployment Patterns
- You know what Blue-Green Deployment is, and how to use deployment slots in Azure to implement this
- You know what a canary release is, and how to use Azure Traffic Manager to implement this
- You know what a feature toggle is, and how they are of great importance when implementing Continuous Delivery
- You know what a Dark Launch is and how this differs from a canary release
- You know the concept of A/B testing and how this is enabled by Continuous Delivery
- You know what Progressive Exposure Deployment and Ring based deployments are and how to implement this in Azure DevOps

Continuous Delivery and Architecture

As we have mentioned before in this course, Continuous Delivery is more than release management. Continuous Delivery is all about the process, the people, and the tools that you need to make sure that you can deliver your software on demand. You need to recognize that deployment is only 1 step within the whole Continuous Delivery process. To be able to deploy on demand or multiple times a day, all the prerequisites need to be in place.

For example:

Testing Strategy

Your testing strategy should be in place. If you need to run a lot of manual tests to validate your software, this is a bottleneck to deliver on demand.

Coding practices

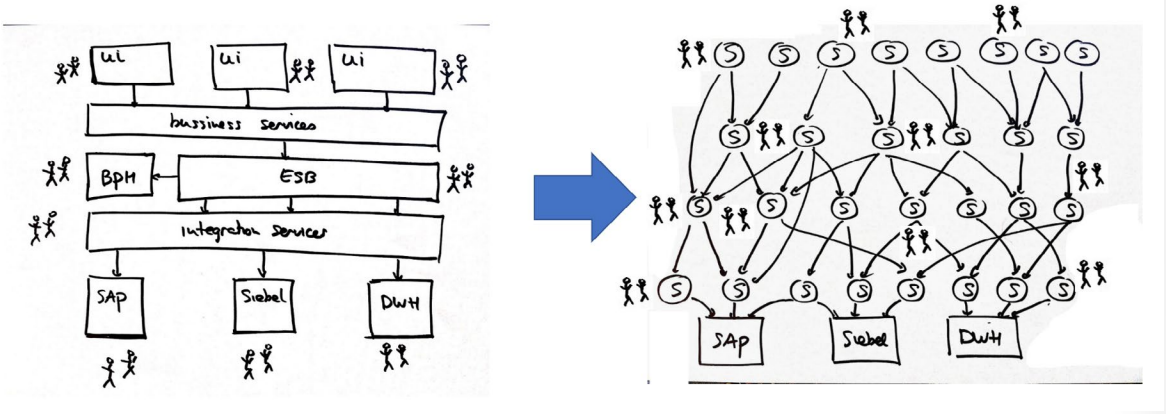
If your software is not written in a safe and maintainable manner, the chances are that you cannot maintain a high release cadence. When your software is complex because of a large amount of technical

Debt, it is hard to fast and reliably change the code. Writing high-quality software and high-quality tests are, therefore, an essential part of Continuous Delivery.

Architecture

The architecture of your application is always significant. But when implementing Continuous Delivery, it is maybe even more so. If your software is a monolith with a lot of tight coupling between the various components, it is difficult to deliver your software continuously. Every part that is changed might impact other parts that did not change. Automated tests can track a lot of these unexpected dependencies but it still hard. There is also the time aspect when working with different teams. When Team A relies on a service of Team B, Team A cannot deliver until Team A is done. This introduces another constraint on delivery.

Continuous Delivery for large software products is hard. For smaller parts, it is easier. Therefore, breaking up your software into smaller, independent pieces, is in many cases a good solution. Nowadays, you frequently hear the term microservices. A microservice is an autonomous, independent deployable, and scalable software component. They are small, and they are focused on doing one thing very well, and they can run autonomously. If one micro service changes, it should not impact any other microservices within your landscape. By choosing a microservices architecture, you will create a landscape of services that can be developed, tested and deployed separately from each other.
Of course, this implies other risks and complexity. You need to create to keep track of interfaces and how they interact with each other. And you need to maintain multiple application lifecycles instead of one.



In a traditional application, we can often see a multi-layer architecture. One layer with the UI, a layer with the business logic and services and a layer with the data services. Sometimes there are dedicated teams for the UI and the backend. When something needs to change, it needs to change in all the layers.

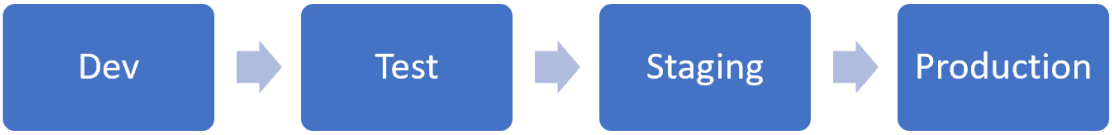
When moving towards a microservices architecture, all these layers are part of the same microservice. Only the microservice contains one specific function. The interaction between the microservices is done in an asynchronous matter. They do not call each other directly but make use of asynchronous mechanisms like queues or events

Each microservice has its own lifecycle and Continuous Delivery pipeline. If you built them correctly, you could deploy new versions of a microservice without impacting other parts of the system.

A microservice architecture is undoubtedly not a prerequisite for doing Continuous Delivery, but smaller software components certainly help in implementing a fully automated pipeline.

Deployment Patterns

When we have our prerequisites in place to be able to deliver our software continuously, we need to start thinking about a deployment pattern. Traditionally a deployment pattern was pretty straightforward.



The software was built, and when all features had been implemented, the software was deployed to an environment where a group of people could start using it.

The traditional or classical deployment pattern was moving your software to a development stage, a testing stage, maybe an acceptance or staging stage, and finally a production stage. The software moved as one piece through the stages. The production release was in most cases a Big Bang release, where users were confronted with a lot of changes at the same time. In spite of the different stages to test and validate, this approach still involves a lot of risks. By running all your tests and validation on non-production environments, it is hard to predict what happens when your production users start using it. Of course you can run load tests and availability tests, but in the end, there is no place like production.

End users always use your application differently. Unexpected events will happen in a data center, multiple events from multiple users will occur at the same time, triggering some code that has not been tested in that way. To overcome this, we need to embrace the fact that some features can only be tested in production.

Testing in production sounds a bit scary, but that should not be the case. When we talked about separating our functional and technical release, we already saw that it is possible to deploy features without exposing them to all the users. When we take this concept, of feature toggling, and use it together with our deployment patterns, we can test our software in production.

For example

- Blue-Green deployments
- Canary releasing
- Dark launching
- A/B testing
- Progressive Exposure Deployment

We will talk about the different patterns in the next chapters.

With all deployment patterns, the main thing to consider is the creation of feedback loops. The only way to implement a deployment pattern successfully is to have monitoring in place. You need to know how your system behaves. Before a deployment and after a deployment, so you can see the differences and the impact.

- Availability
  You need to know details about your availability. Is your app available? What are the response times?
- Performance
  How does your application perform? What is the average CPU usage? And the average memory usage? Are there spikes in performance?

- Usage
  You need to know about your usage. How do people interact with your system? In what time frames do they interact with your system? Do they finish the flows or do they drop off?
- Diagnostics
  You need to have a lot of diagnostic logging and telemetry available. To see what happens when something goes wrong, or to see what happened before a performance spike, or a user drops off.

In the next chapters, we will talk about the deployment patterns. Monitoring is out of scope but definitely something you should implement.

To get an overview of where to get started with DevOps Metrics read this blog post[1]

Discussion

A critical look at your architecture

Are your architecture and the current state of your software ready for Continuous Delivery?

Topics you might want to consider are:

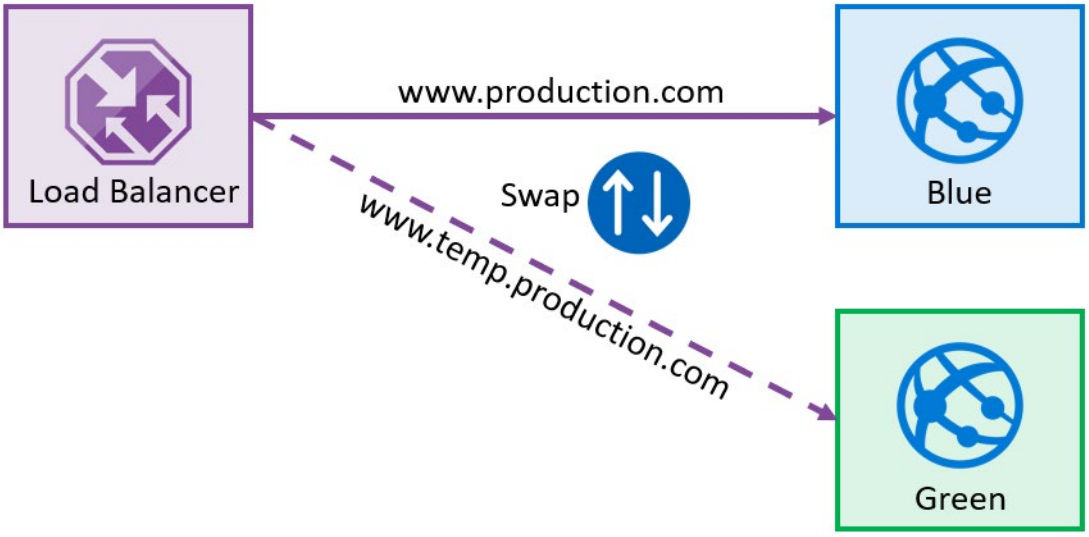
- Is your software built as one big monolith or is it divided into multiple components?
- Can you deliver parts of your application separately?
- Can you guarantee the quality of your software when deploying multiple times a week?
- How do you test your software?
- Do you run 1 or multiple versions of your software?
- Can you run multiple versions of your software side-by-side?
- What do need to improve to implement Continuous Delivery?

1 https://roadtoalm.com/2018/09/05/where-to-start-with-devops-metrics/

Implement Blue Green Deployment

Blue-Green deployment

Blue-Green deployment is a technique that reduces risk and downtime by running two identical environments. These environments are called Blue and Green. At any time, only one of the environments is live, with the live environment serving all production traffic.



For this example, Blue is currently live, and Green is idle.
As you prepare a new version of your software, the deployment and the final stage of testing takes place in the environment that is not live: in this example, Green. Once you have deployed and thoroughly tested the software in Green, you switch the router or load balancer, so all incoming requests now go to Green instead of Blue. Green is now live, and Blue is idle.

This technique can eliminate downtime due to app deployment. Besides, Blue-Green deployment reduces risk: if something unexpected happens with your new version on Green, you can immediately roll back to the last version by switching back to Blue.

When this involves database schema changes, this process is of course not straightforward. You can probably not swap your application. In that case, your application and architecture should be built in such a way, that it can handle both the old and the new database schema.

Deployment Slots

When using a cloud platform like Azure, doing Blue-Green deployments is relatively easy. You do not need to write your own code or set up infrastructure. When using web apps, you can use an out-of-the-box feature called Deployment Slots.

Deployment Slots are a feature of Azure App Service. They are live apps with their own hostnames. You can create different slots for your application (e.g., Dev, Test or Stage). The Production slot is the slot where your live app resides. With deployment slots, you can validate app changes in staging before swapping it with your production slot.

You can use a deployment slot to set up a new version of your application, and when ready, swap the production environment with the new, staging environment. This is done by an internal swapping of the IP addresses of both slots.

To learn more about Deployment Slots read more on these links

- **Set up Deployment Slots on Azure**[2]
- **Considerations on using Deployment Slots in your DevOps Pipeline**[3]
- **Enabling Deployment Slots To Deploy Applications To Azure App Service Safely**[4]

Demonstration Set Up a Blue-Green Deployment

In this demonstration, you will investigate Blue-Green Deployment.

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

Steps

Let's now take a look at how a release pipeline can be used to implement blue-green deployments.

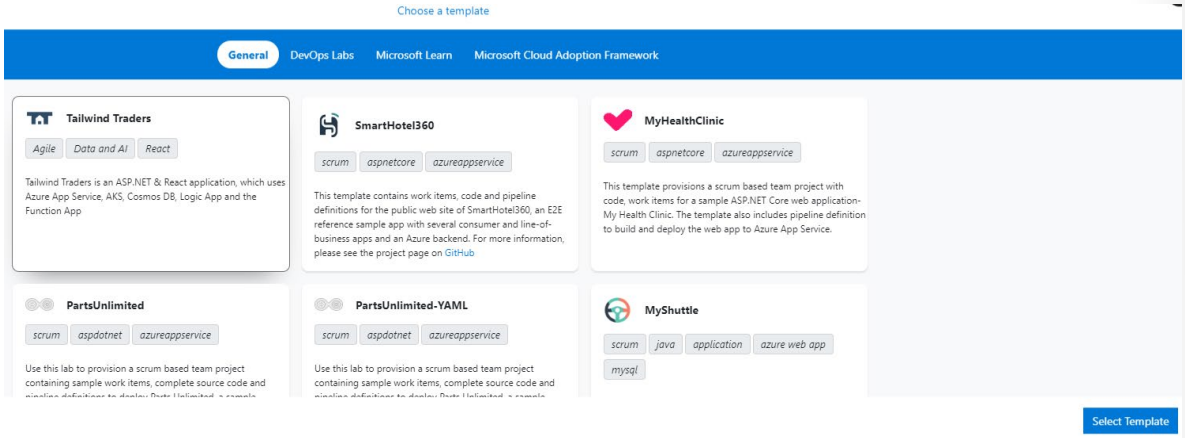
We'll start by creating a new project with a release pipeline that can perform deployments, by using the **Parts Unlimited** template again.

An initial app deployment

1. In a browser, navigate to http://https://azuredevopsdemogenerator.azurewebsites.net and click **Sign in**.

You will be prompted to sign in if necessary.

2. In the **Create New Project** window, select your existing Organization, set the **Project Name** to **PU Hosted** and click **Choose template**.



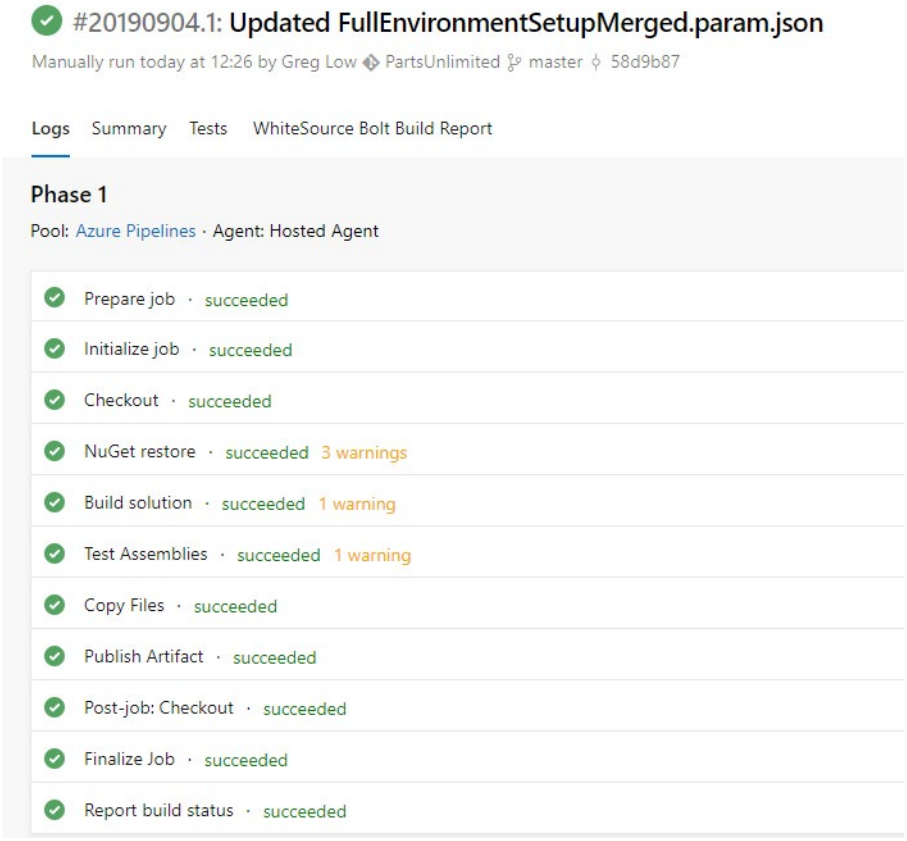
3. Click on the **PartsUnlimited** project (not the PartsUnlimited-YAML project), and click **Select Template**, then click **Create Project**. When the deployment completes, click **Navigate to project**.

2 https://docs.microsoft.com/en-us/azure/app-service/deploy-staging-slots
3 https://blogs.msdn.microsoft.com/devops/2017/04/10/considerations-on-using-deployment-slots-in-your-devops-pipeline/
4 https://blogs.msdn.microsoft.com/mvpawardprogram/2017/05/16/deploy-app-azure-app-service/

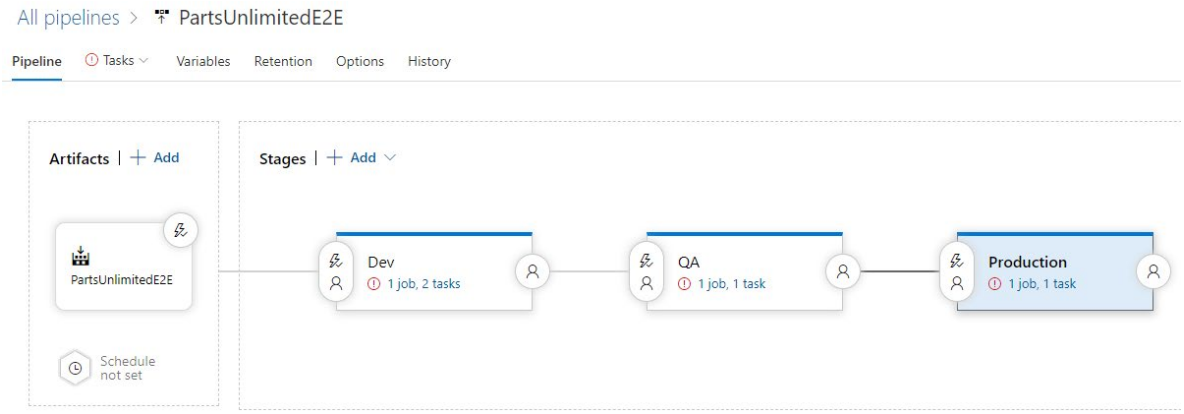
4. In the main menu for **PU Hosted**, click **Pipelines**, then click **Builds**, then **Queue** and finally **Run** to start a build.

The build should succeed.

Note: warnings might appear but can be ignored for this walkthrough



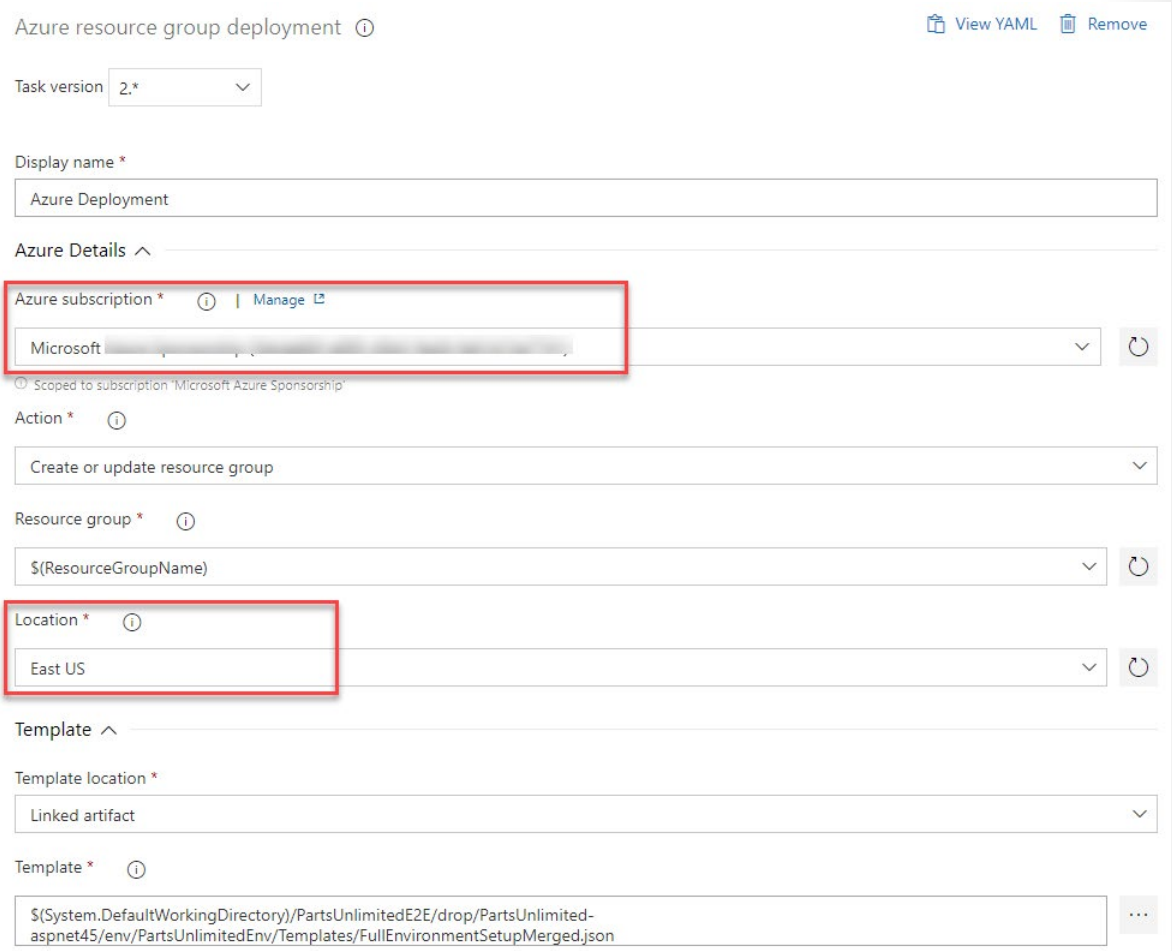
5. In the main menu, click **Releases**. Because a continuous integration trigger was in place, a release was attempted. However, we have not yet configured the release so it will have failed. Click **Edit** to enter edit mode for the release.



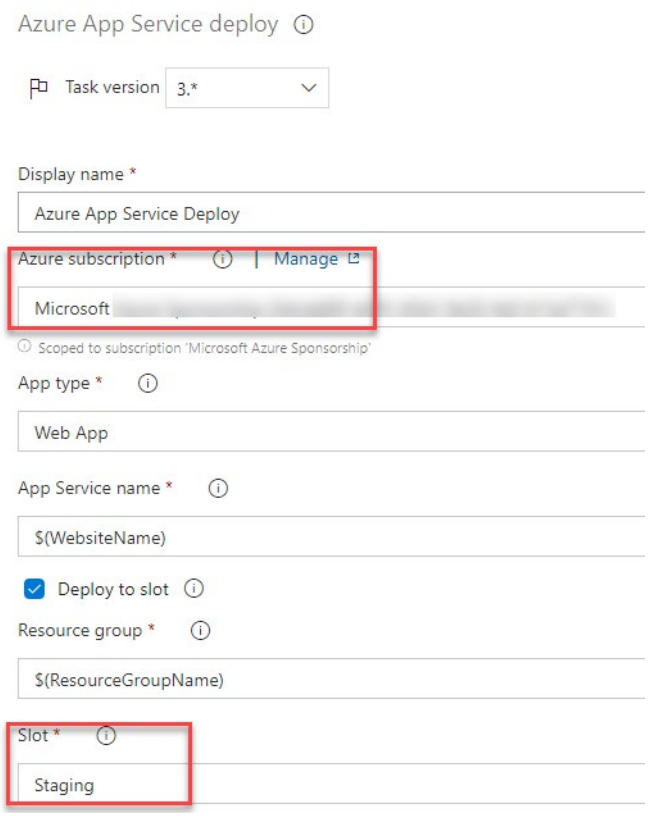
6. From the drop down list beside **Tasks**, select the **Dev** stage, then click to select the **Azure Deployment** task.

7. In the **Azure resource group deployment** pane, select your Azure subscription, then click **Authorize** when prompted. When authorization completes, select a **Location** for the web app.

Note: you might be prompted to log in to Azure at this point

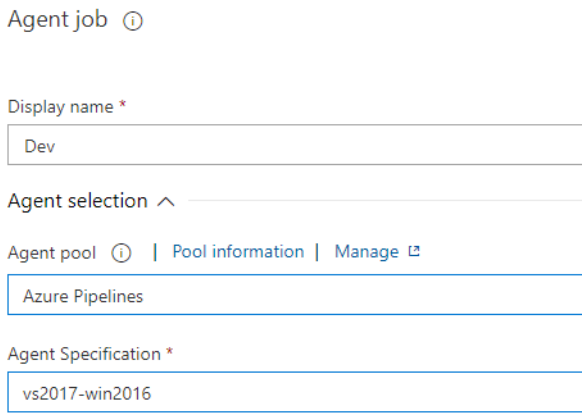


8. In the task list, click **Azure App Service Deploy** to open its settings. Again select your Azure subscription. Set the **Deployment slot** to **Staging**.

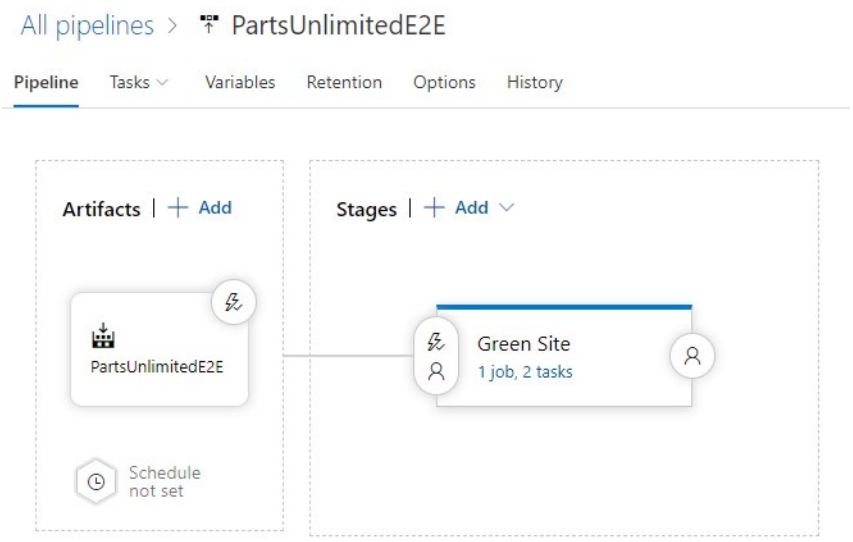


Note: the template creates a production site and two deployment slots: Dev and Staging. We will use Staging for our Green site.

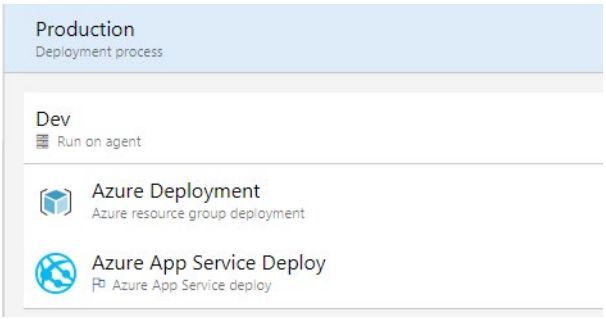
9. In the task list, click **Dev** and in the **Agent job** pane, select **Azure Pipelines** for the **Agent pool** and **vs2017-win2016** for the **Agent Specification**.



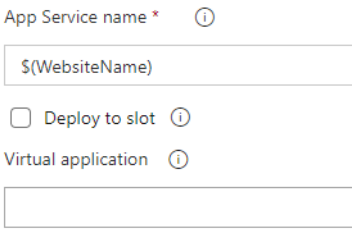
10. From the top menu, click **Pipelines**. Click the **Dev** stage, and in the properties window, rename it to **Green Site**. Click the **QA** stage, and click **Delete** and **Confirm**. Click the **Production** stage, and click **Delete** and **Confirm**. Click **Save** then **OK**.



11. Hover over the **Green Site** stage, and click the **Clone** icon when it appears. Change the **Stage name** to **Production**. From the **Tasks** drop down list, select **Production**.

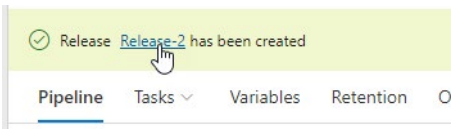


12. Click the **Azure App Service Deploy** task, and uncheck the **Deploy to slot** option. Click **Save** and **OK**.

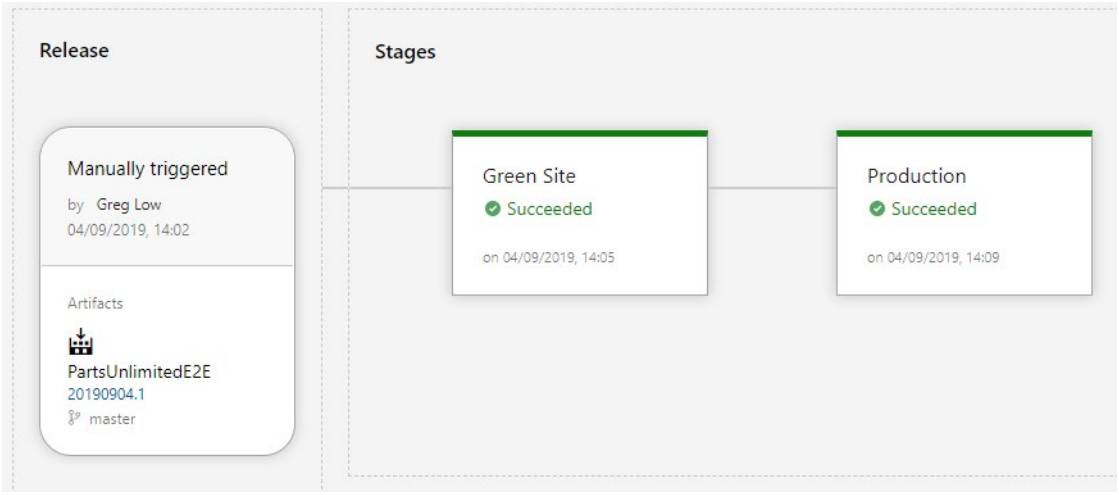


The production site isn't deployed to a deployment slot. It is deployed to the main site.

13. Click **Create release** then **Create** to create the new release. When it has been created, click the release link to view its status.



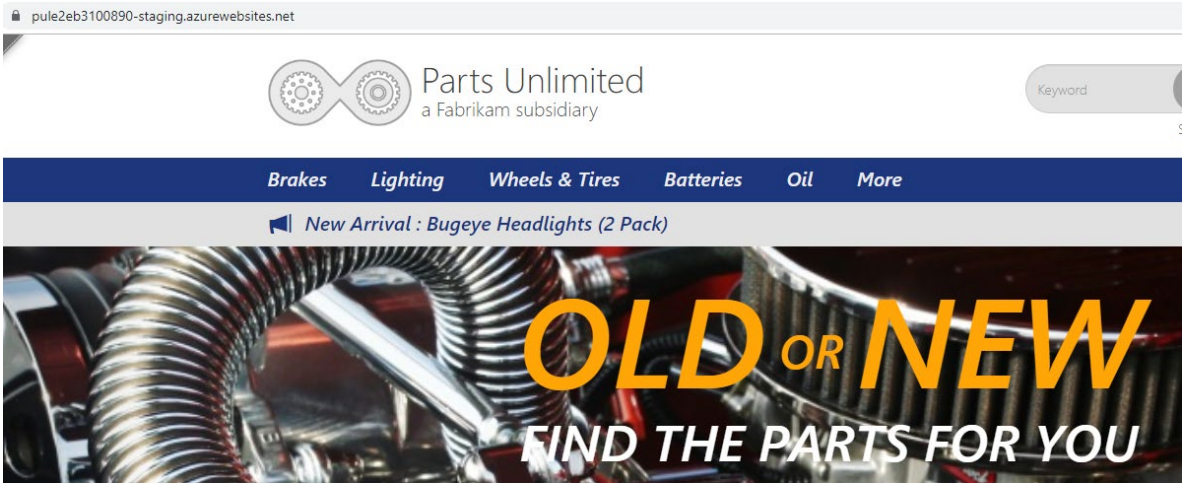
After a while, the deployment should succeed.



Test the Green Site and the Production Site

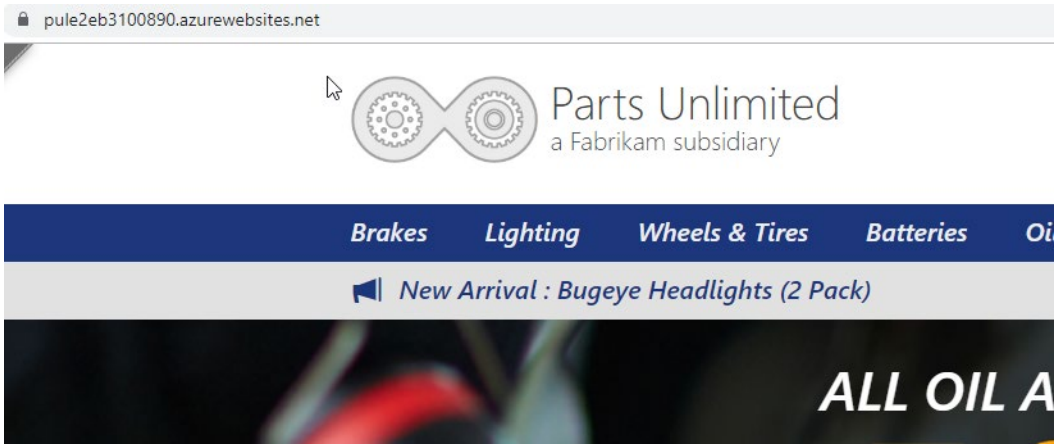
14. In the **Azure Portal**, open the blade for the **ASPDOTNET** resource group that was created by the project deployment. Notice the names of the web apps that have been deployed. Click to open the *Staging** web app's blade. Copy the URL from the top left hand side.

15. Open a new browser tab and navigate to the copied URL. It will take the application a short while to compile but then the Green website (on the Staging slot) should appear.



*Note: you can tell that the staging slot is being used because of the **-staging** suffix in the website URL*

16. Open another new browser tab, and navigate to the same URL but without the **-staging** slot. The production site should also be working.

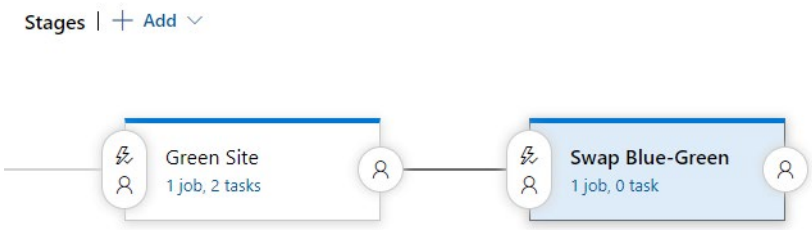


Note: Leave both browser windows open for later in the walkthrough

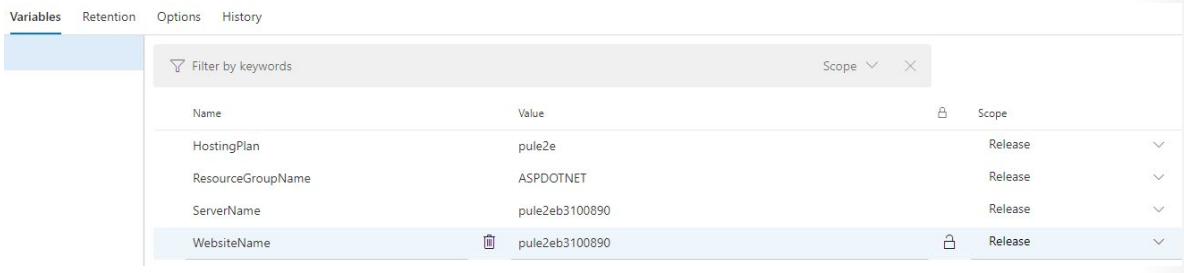
Configure Blue-Green swap and Approval

Now that both sites are working, let's configure the release pipeline for Blue-Green deployment.

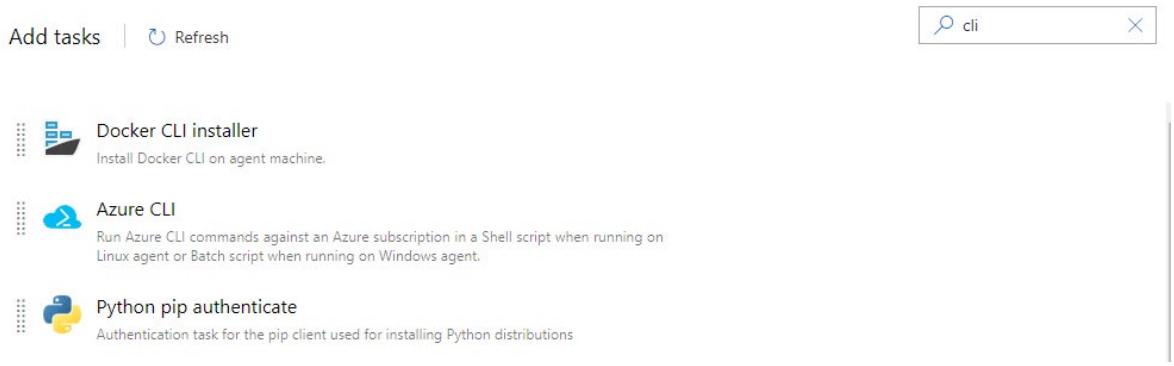
17. In **Azure DevOps**, in the main menu for the **PU Hosted** project, click **Pipelines**, then click **Releases**, then click **Edit** to return to edit mode.
18. Click the **Production** stage, click **Delete**, then **Confirm** to remove it. Click **+Add** to add an extra stage, and click **Empty job** for the template. Set **Swap Blue-Green** for the **Stage name**.



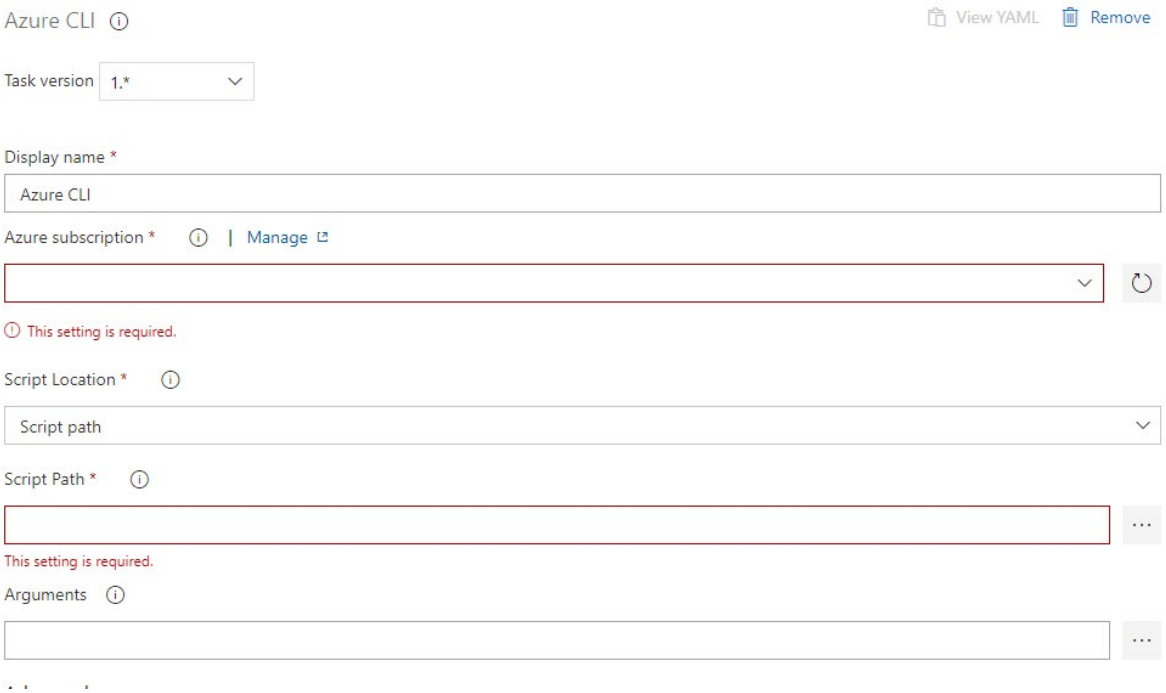
19. Click **Variables** and modify the **Scope** of **WebsiteName** to **Release**.



20. From the **Tasks** drop down list, click to select the **Swap Blue-Green** stage. Click the **+** to the right hand side of **Agent Job** to add a new task. In the **Search** box, type **cli**.

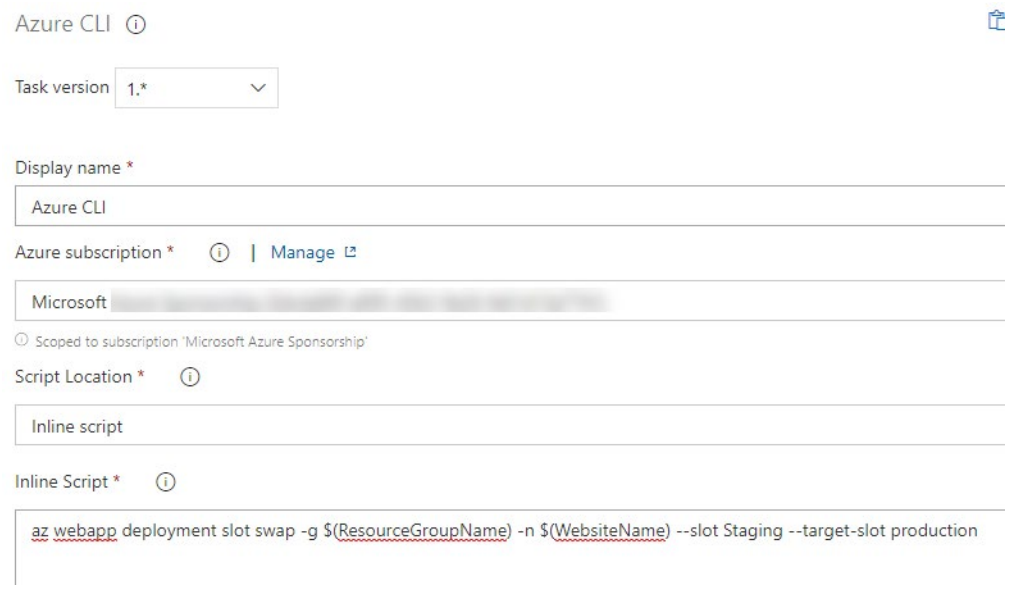


21. Hover over the **Azure CLI** template and when the **Add** button appears, click it, then click to select the **Azure CLI** task, to open its settings pane.

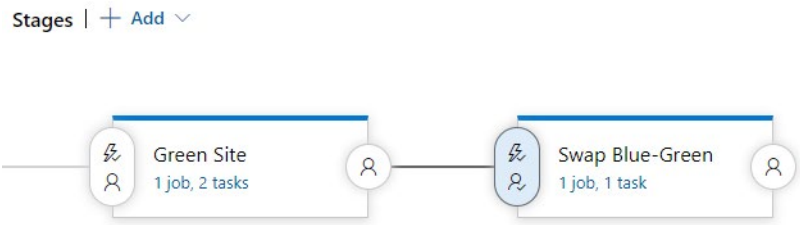


22. Configure the pane as follows, with your subscription, a **Script Location** of **Inline script**, and the **Inline Script** as follows:

```
az webapp deployment slot swap -g $(ResourceGroupName) -n $(WebsiteName) –slot Staging –target-slot production
```

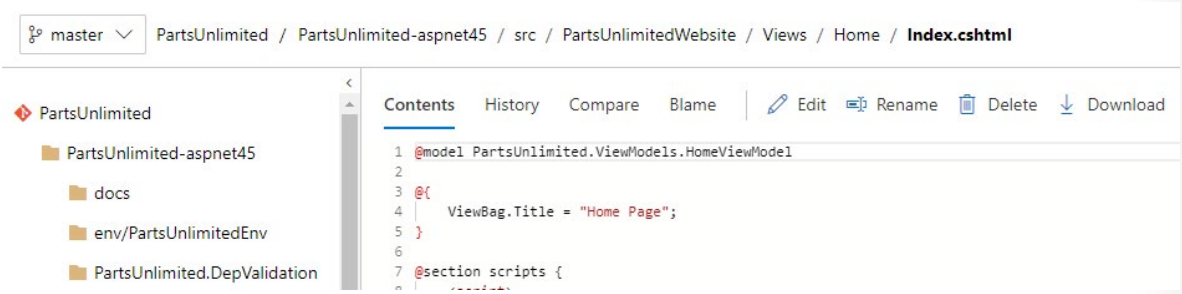


23. From the menu above the task list, click **Pineline**. Click the **Pre-deployment conditions** icon for the **Swap Blue-Green** task, then in the **Triggers** pane, enable **Pre-deployment approvals**.
24. Configure yourself as an approver, click **Save**, then **OK**.



Test the blue-green swap

25. In the **PU Hosted** main menu, click **Repos** then click **Files** to open the project files. Navigate to the following file.

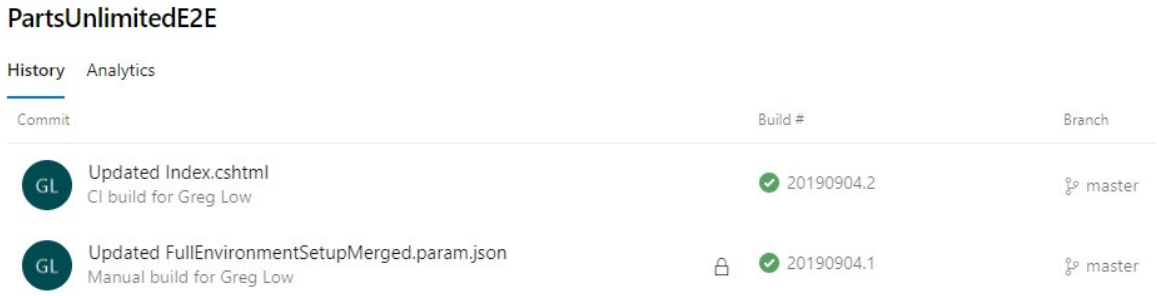


We will make a cosmetic change so that we can see that the website has been updated. We'll change the word **tires** in the main page rotation to **tyres** to target an international audience.

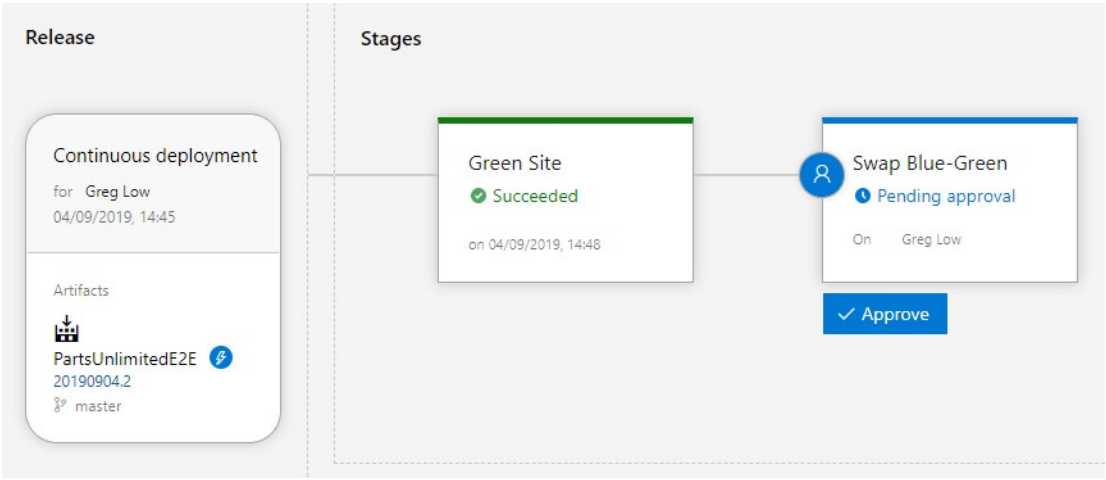
26. Click **Edit** to allow editing, then find the word **tires** and replace it with the word **tyres**. Click **Commit** and **Commit** to save the changes and trigger a build and release.

```
<div class="item" style="background-image: url('/Images/hero_imag
    <a href="@Url.Action("Browse", "Store", new { categoryId = 3
    <div class="container">
        <p>New tyres</p>
        <ul>
            <li>Improved fuel efficiency</li>
            <li>Superior wet weather breaking</li>
            <li>Added durability</li>
        </ul>
    </div>
```

27. From the main menu, click **Pipelines**, then **Builds**. Wait for the continuous integration build to complete successfully.



28. From the main menu, click **Releases**. Click to open the latest release (at the top of the list).

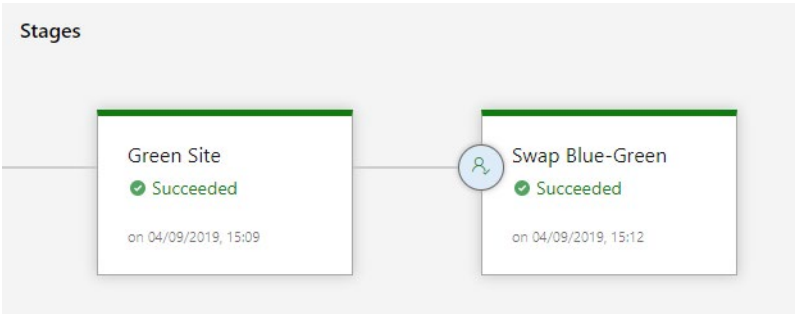


You are now being asked to approve the deployment swap across to production. We'll check the green deployment first.

29. Refresh the Green site (i.e. Staging slot) browser tab and see if your change has appeared. It now shows the altered word.

30. Refresh the Production site browser tab and notice that it still isn't updated.

31. As you are happy with the change, in release details click **Approve**, then **Approve** and wait for the stage to complete.



32. Refresh the Production site browser tab and check that it now has the updated code.

Final Notes

If you check the production site, you'll see it now has the previous version of the code.

This is the key difference with using Swap, rather than just having a typical deployment process from one staged site to the next. You have a very quick fall back option by being able to swap the sites back again if needed.

Lab: Blue-Green Deployment

Overview

In this lab, you will learn how to do a Blue-Green Deployment in the pipeline.

Before you begin

Refer the **Azure DevOps Labs Getting Started**[5] page to know the prerequisites for this lab.

Complete **Task 1 from the Azure DevOps Lab Prerequisites**[6]

Setting up the Target Environment

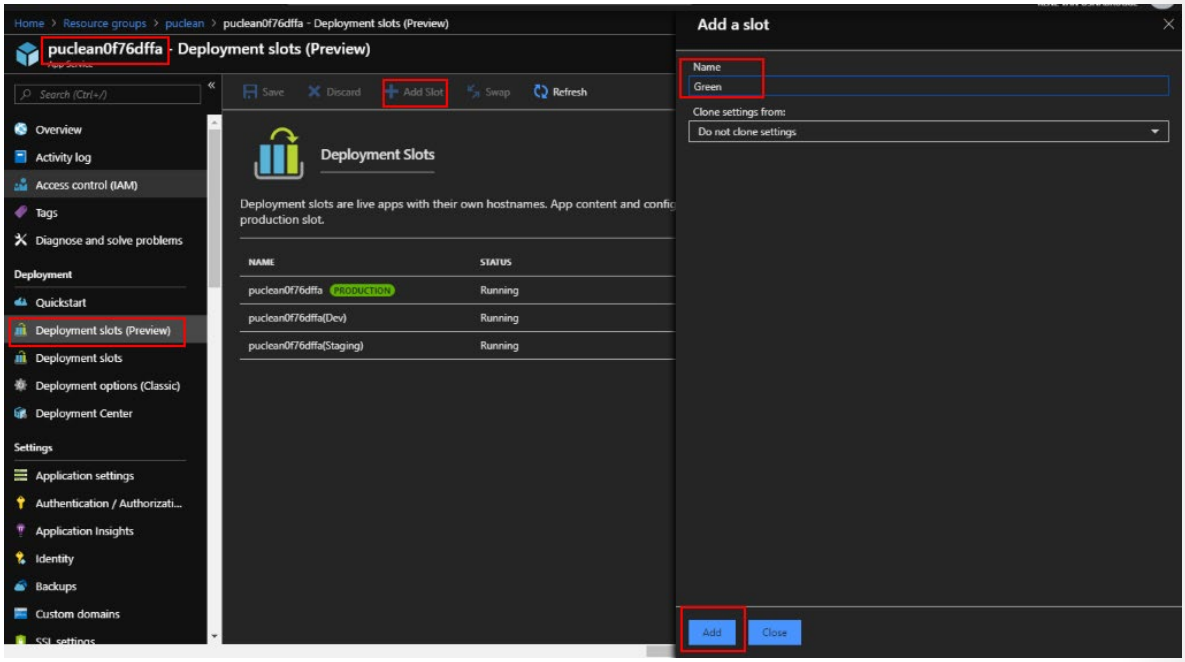
You will run one build and one release to get some data in your project. Make sure you have an Azure account where you can deploy the application.

1. Navigate to **PipelinesReleases** and configure the Service Connections of all stages in the **PartsUnlimitedE2E**. Set the deployment slot name of the Dev Stage to *Dev* and the QA stage to *Staging*
2. In you Azure DevOps Project Go to **Pipelines** and click **Builds** and **Queue** a new **PartsUnlimitedE2E** build.
3. The application is built and deployed automatically

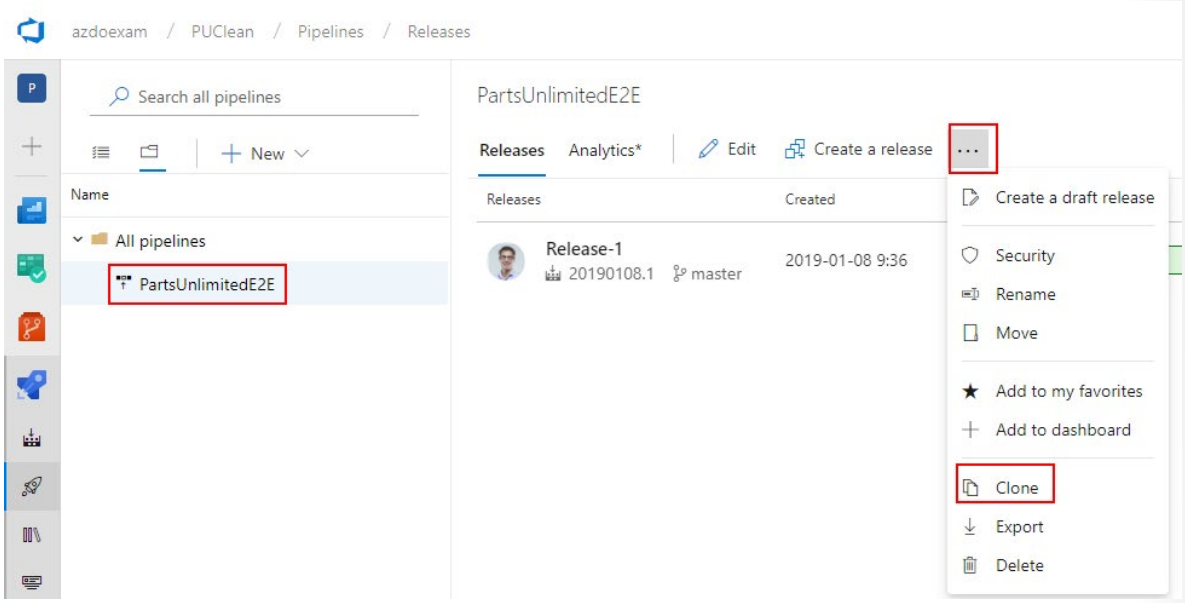
Exercise 1: Create a new Deployment Slot

1. In the Azure Portal, navigate to your deployed web application. Click **Deployment Slots** in the Blade, and **Add** a new slot "Green."

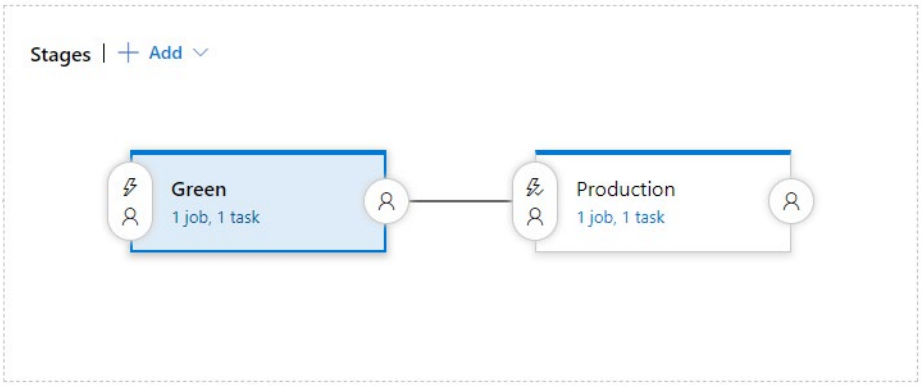
5 https://azuredevopslabs.com/labs/vstsextend/Setup/
6 https://azuredevopslabs.com/labs/azuredevops/prereq/#task-1-configuring-the-parts-unlimited-team-project



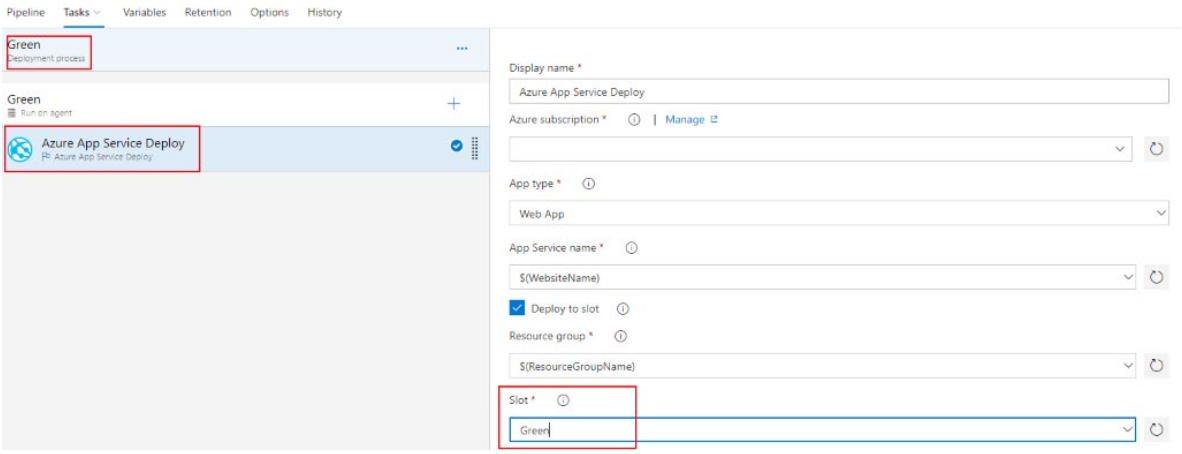
2. In Azure DevOps, navigate to **Releases** under Pipelines. **Clone** the **PartsUnlimitedE2E** release pipeline and call it Blue-Green.



3. In the Pipeline view, **remove** the Dev Stage, and **rename** the QA stage to Green.

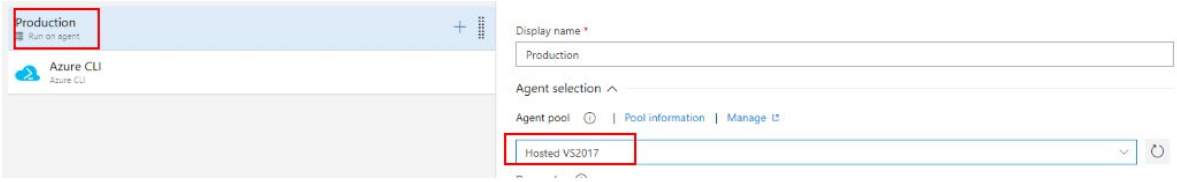


4. In the Green stage, select the **Azure App Service Deploy** task, and change the deployment **Slot** to Green



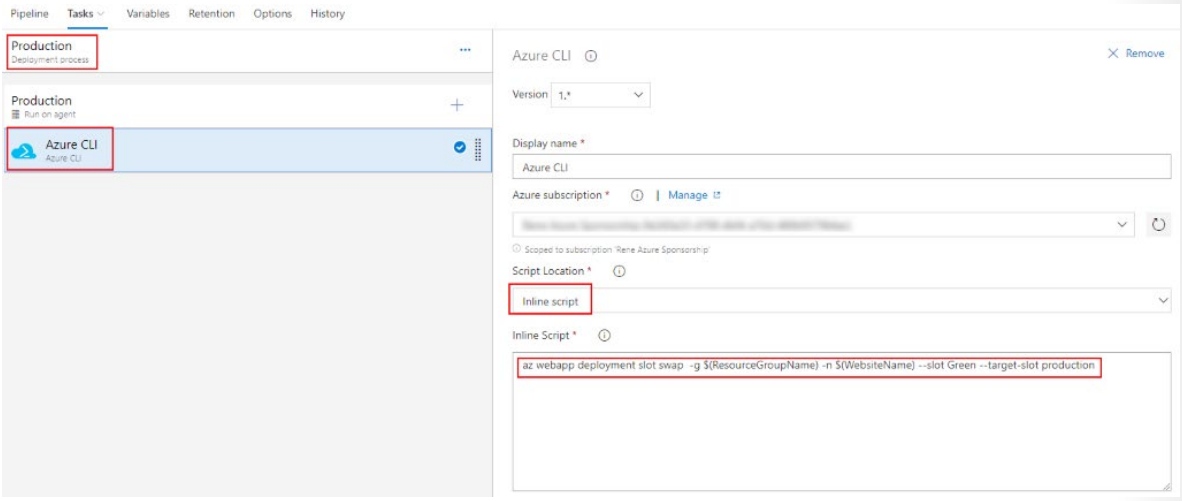
5. In the Production stage, **remove** the **Azure App Service Deploy** task. **Add** an **Azure CLI Task**, and set the Service Connection to your Azure Account.

6. **Select** the **Production** Agent Job and select the Hosted VS2017 agent.

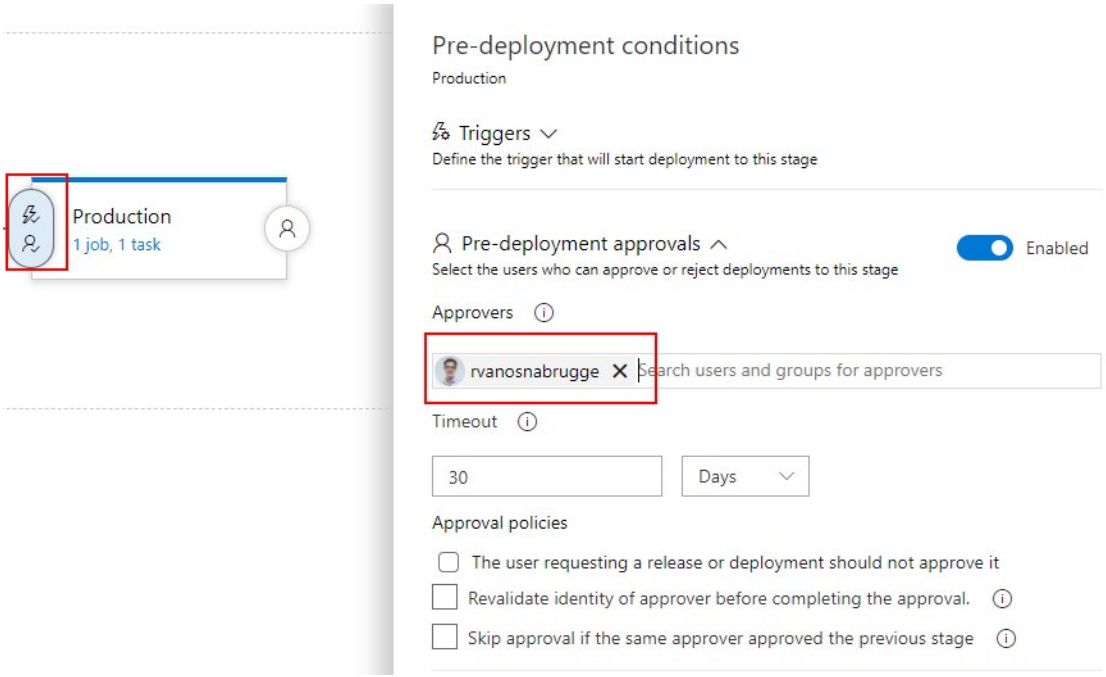


7. **Add** the following inline script to your Azure CLI task

```
az webapp deployment slot swap  -g $(ResourceGroupName) -n $(WebsiteName)
--slot Green --target-slot production
```



8. Add yourself as an **approver** for the production stage.



9. **Create new Release**. The web application is deployed to the Green slot. In your Azure Portal, select the Green Slot and navigate to the URL. Typically –> https://webappname-slotname.azurewebsites.net/ (in my example https://puclean0f76dffa-green.azurewebsites.net/)
10. When the application is available, **approve** the production stage. When the release completes, your application swapped from Green to Production.

You can deploy a new version of the application to make it visible. Change the background color of your application, run a new build and rerun this scenario.

Links

- To set up a pipeline from scratch, complete the lab **Embracing Continuous Delivery with Azure Pipelines**[7]
- To see how this works with Octopus Deploy, **read more here**[8]

7 https://azuredevopslabs.com/labs/azuredevops/continuousdeployment/
8 https://octopus.com/docs/deployment-examples/azure-deployments/deploying-a-package-to-an-azure-web-app/using-deployment-slots-with-azure-web-apps

Feature Toggles

Feature Toggles

In the previous module, we talked about the separation of technical and functional releases and how feature toggles play an essential role in that. By exposing new features by just "flipping a switch" at runtime, we can deploy new software without exposing any new or changed functionality to the end user.

The question is what strategy you want to use in releasing a feature to an end user.

- Reveal the feature to a segment of users, so you can see how the new feature is received and used.
- Reveal the feature to a randomly selected percentage of users.
- Reveal the feature to all users at the same time.

The business owner plays a vital role in the process, and you need to work closely together with him to choose the right strategy.

Just as in all the other deployment patterns and mentioned in the introduction, the most important part is that you always look at the way the system behaves.

The whole idea of separating feature deployment from Feature exposure is compelling and something we want to incorporate in our Continuous Delivery practice. It helps us with more stable releases and better ways to roll back when we run into issues when we have a new feature that produces problems. We switch it off again and then create a hotfix. By separating deployments from revealing a feature, you create the opportunity to release any time a day, since the new software will not affect the system that already works.

What are feature toggles

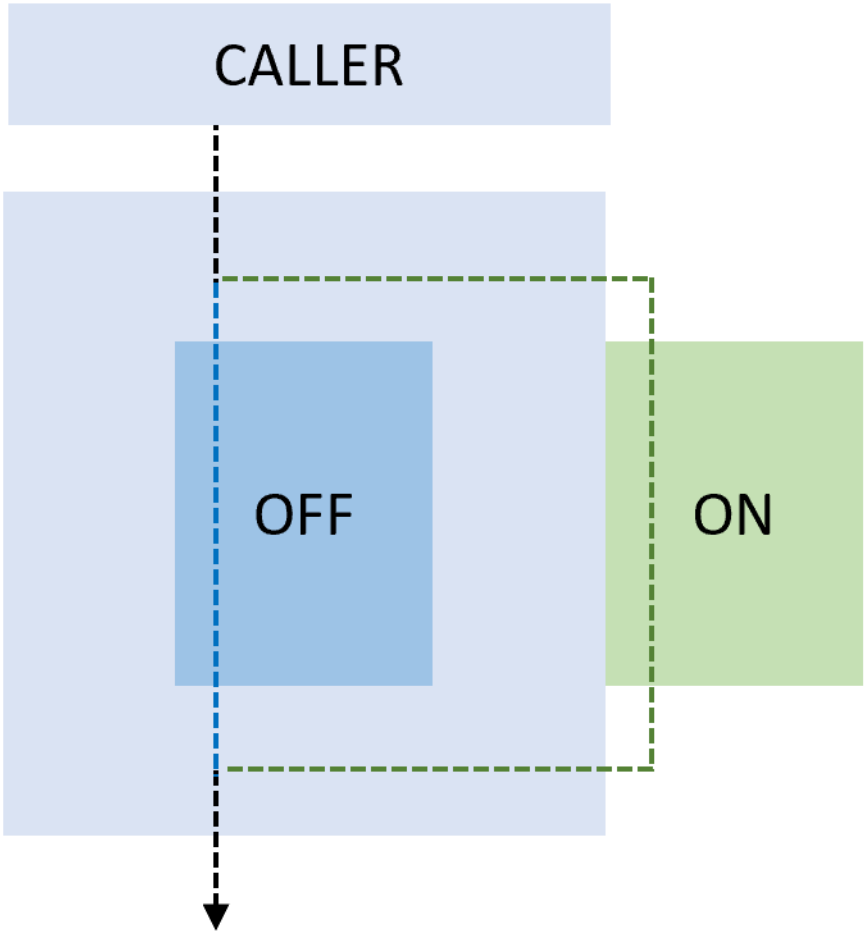
Feature toggles are also known as feature flippers, feature flags, feature switches, conditional features, etc.

Besides the power they give you on the business side, they provide an advantage on the development side as well. Feature toggles are a great alternative to branching as well. Branching is what we do in our version control system. To keep features isolated, we maintain a separate branch. The moment we want the software to be in production, we merge it with the release branch and deploy.

With feature toggles, you build new features behind a toggle. When a release occurs, your feature is "off" and should not be exposed to or impacting the production software.

How to implement a feature toggle

In the purest form, a feature toggle is an IF statement.



When the switch is off, it executes the code in the IF, otherwise the ELSE. Of course, you can make it much more intelligent, controlling the feature toggles from a dashboard, or building capabilities for roles, users, etc.

If you want to implement feature toggles, then there are many different frameworks available, both commercially as Open Source.

Links

- **Explore how to progressively expose your features in production for some or all users**[9]

Feature Toggles Maintenance

A feature toggle is just code. And to be more specific, conditional code. It adds complexity to the code and increases the technical debt. Be aware of that when you write them, and clean up when you do not need them anymore.

The idea of a toggle is that it is short lived and only stays in the software for the time it is necessary to release it to the customers.

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

Read more about a feature toggle[10]

You can classify the different types of toggles based on two dimensions as described by Martin Fowler. He states that you can look at the dimension of how long a toggle should be in your codebase and on the other side how dynamic the toggle needs to be.

You can see that only a few toggles stay in your software, but those are more toggles that you already used to authorize people in your system or some kill switches that you need as an administrator to enable you to turn features of when load or other issues make the system misbehave. So always evaluate the toggles you have and remove them when you can.

The most important thing is to remember that you need to remove the toggles from the software as soon as possible. If you do not do that, they will become a form of technical debt if you keep them around for too long.

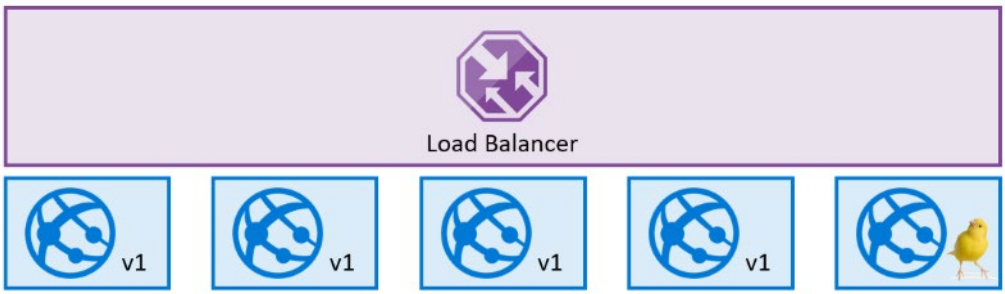
10 http://martinfowler.com/articles/feature-toggles.html

Canary Releases

Canary Releases

The term canary release comes from the days that miners took a canary with them into the coal mines. The purpose of the canary was to identify the existence of toxic gasses. The canary would die much sooner than the miner, giving them enough time to escape the potentially lethal environment.

A canary release is a way to identify potential problems as soon as possible without exposing all your end users to the issue at once. The idea is that you expose a new feature only to a minimal subset of users. By closely monitoring what happens the moment you enable the feature, you can get relevant information from this set of users and decide to either continue or rollback (disable the feature). If the canary release shows potential performance or scalability problems, you can build a fix for that and apply that in the canary environment. After the canary release has proven to be stable, you can move the canary release to the actual production environment.



Traffic Manager

In the previous module, we saw how Deployment slots in Azure Web Apps, enable you to swap between 2 different versions of your application quickly. If you want to have more control over the traffic that flows to your different versions, deployment slots alone is not enough. To control traffic in Azure, you can use a component called Traffic Manager.

Azure Traffic Manager is a DNS-based traffic load balancer that enables you to distribute traffic optimally to services across global Azure regions while providing high availability and responsiveness.

Traffic Manager uses DNS to direct client requests to the most appropriate service endpoint based on a traffic-routing method and the health of the endpoints.

An endpoint is an Internet-facing service hosted inside or outside of Azure. Traffic Manager provides a range of traffic-routing methods and endpoint monitoring options to suit different application needs and automatic failover models. Traffic Manager is resilient to failure, including the breakdown of an entire Azure region.

Traffic manager provides four options to distribute traffic:

Route traffic based on availability

This option is routing based on availability. This is predominantly a mechanism to provide failover if one app service instance appears to be down. Traffic manager will take care of routing the traffic to other instances that are available, so the end user will not experience any downtime. This is an excellent feature if you want to release new features without any downtime. You can create a new app service, then deploy the new software on the new app services and then bring the new app service online and let traffic manager move the traffic to the new instance. The moment all traffic is on the new instance you can then

remove the old instance. This way you move customers to new software, without causing downtime. This is more or less comparable to the deployment slots, but in this case, we can do this between different app services.

Route traffic round robin

This option is the round-robin routing of traffic. This is primarily used for load distribution.

Route traffic based on weight

This option is about routing traffic based on weights. The idea here is that you can set a weight, for example, 1:4, on the different app services it should route to. This causes 1/4 of the traffic to go to one instance and the other 3/4 of the traffic to go to the other instance. It selects the clients randomly. Therefore this is a useful mechanism to be used for A/B testing.

Route traffic based on latency

The final option is to route based on latency. This is primarily used to ensure you route traffic to the right geolocation ensuring the client has the lowest latency possible. But in case of a problem, this also means it will fail over to another data center since the one with shorter latency is not available at that moment.

When we look at the options the traffic manager offers, the most used options for Continuous Delivery are the options to route traffic based on availability and the option to route based on weights.

Read more about Azure Traffic Manager[11]

Read how Traffic Manager works[12]

Controlling your canary release

Using a combination of feature toggles, deployment slots and traffic manager you can achieve full control over the traffic flow, and enable your canary release.

You deploy the new feature to the new deployment slot or a new instance of an application, and you enable the feature after verifying the deployment was successful.
Next, you set the traffic to be distributed to a small percentage of the users, and you carefully watch the behavior of the application, e.g. by using application insights to monitor performance and stability of the application.

Demonstration Ring-based Deployment

In this demonstration, you will investigate Ring-based Deployment.

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

Steps

Let's now take a look at how a release pipeline can be used to stage features by using ring-based deployments.

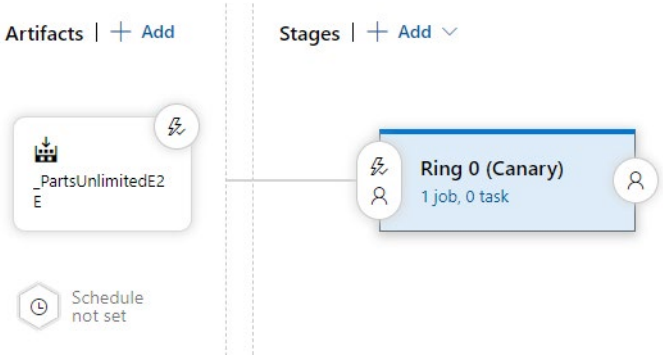
When I have a new features, I might want to just release it to a small number of users at first, just in case something goes wrong. In authenticated systems, I could do this by having those users as members of a security group, and letting members of that group use the new features.

11 https://docs.microsoft.com/en-us/azure/traffic-manager/traffic-manager-overview
12 https://docs.microsoft.com/en-us/azure/traffic-manager/traffic-manager-how-it-works

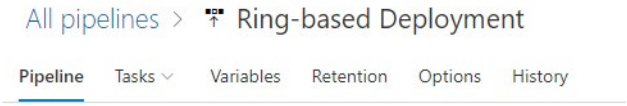
However, on a public web site, I might not have logged in users. Instead, I might want to just direct a small percentage of the traffic to use the new features. Let's see how that's configured. We'll create a new release pipeline that isn't triggered by code changes, but manually when we want to slowly release a new feature.

We start by assuming that a new feature has already been deployed to the Green site (i.e. the staging slot).

1. In the main menu for the **PU Hosted** project, click **Pipelines**, then click **Release**, click **+New**, then click **New release pipeline**.
2. When prompted to select a template, click **Empty job** from the top of the pane.
3. Click on the **Stage 1** stage and rename it to **Ring 0 (Canary)**.



4. Hover over the **New release pipeline** name at the top of the page, and when a pencil appears, click it, and change the pipeline name to **Ring-based Deployment**.



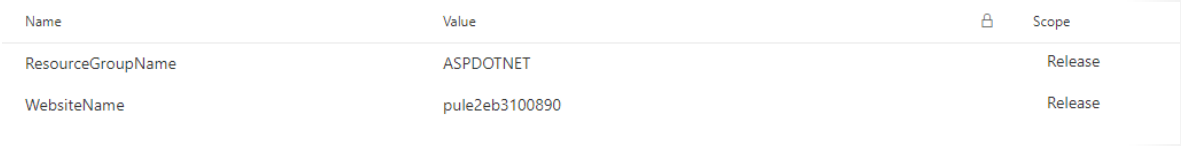
5. From the **Tasks** drop down list, select the **Ring 0 (Canary)** stage. Click the **+** to add a new task, and from the list of tasks, hover of **Azure CLI** and when the **Add** button appears, click it, then click to select the **Azure CLI** task in the task list for the stage.

6. In the **Azure CLI** settings pane, select your **Azure subscription**, set **Script Location** to **Inline script**, and set the **Inline Script** to the following, then click **Save** and **OK**.

```
az webapp traffic-routing set –resource-group $(ResourceGroupName) –name $(WebsiteName) –distribution staging=10
```

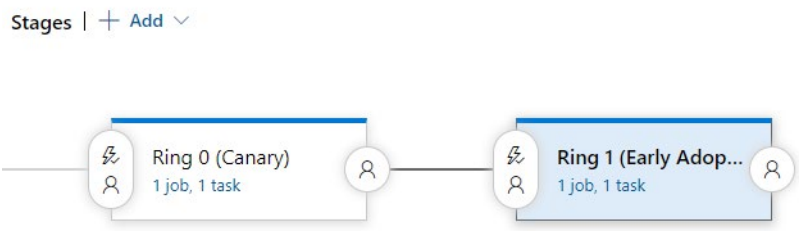
This distribution setting will cause 10% of the web traffic to be sent to the new feature Site (i.e. currently the staging slot).

7. From the menu above the task list, click **Variables**. Create two new variables as shown. (Make sure to use your correct website name).

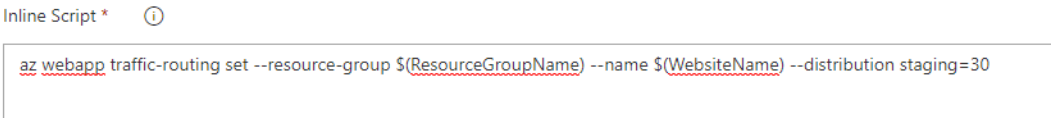


Name	Value	Scope
ResourceGroupName	ASPDOTNET	Release
WebsiteName	pule2eb3100890	Release

8. From the menu above the variables, click **Pipeline** to return to editing the pipeline. Hover over the **Ring 0 (Canary)** stage, and click the **Clone** icon when it appears. Select the new stage, and rename it to **Ring 1 (Early Adopters)**.

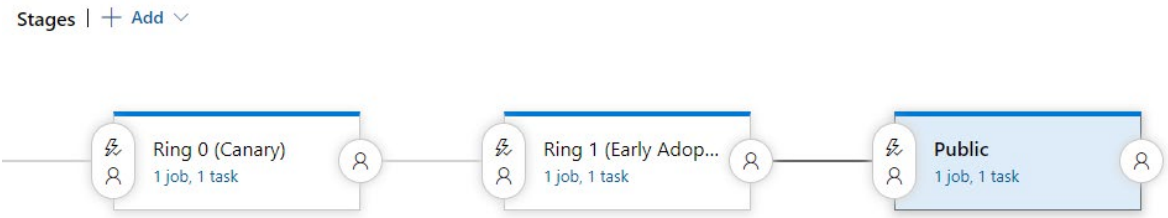


9. From the **Tasks** drop down list, select the **Ring 1 (Early Adopters)** stage, and select the **Azure CLI** task. Modify the script by changing the value of **10** to **30** to cause 30% of the traffic to go to the new feature site.

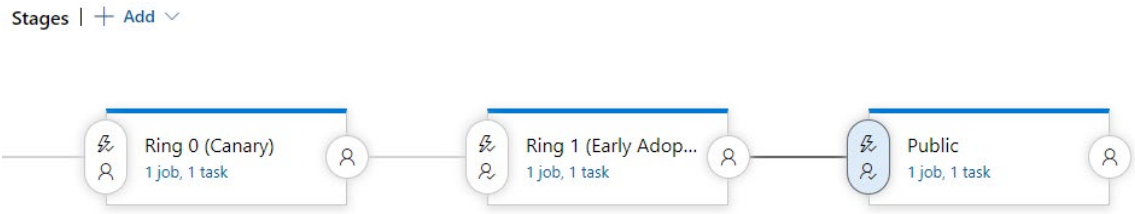


This allows us to move the new feature into wider distribution if it was working ok in the smaller set of users.

10. From the menu above the tasks, click **Pipeline** to return to editing the release pipeline. Hover over the **Ring 1 (Early Adopters)** stage and when the **Clone** icon appears, click it. Click to select the new stage and rename it to **Public**. Click **Save** and **OK**.



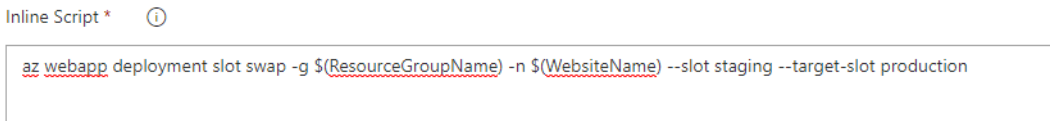
11. Click the **Pre-deployment conditions** icon for the **Ring 1 (Early Adopters)** stage, and add yourself as a pre-deployment approver. Do the same for the **Public** stage. Click **Save** and **OK**.



The first step in letting the new code be released to the public, is to swap the new feature site (i.e. the staging site) with the production, so that production is now running the new code.

12. From the **Tasks** drop down list, select the **Public** stage. Select the **Azure CLI** task, change the **Display name** to **Swap sites** and change the **Inline Script** to the following:

az webapp deployment slot swap -g $(ResourceGroupName) -n $(WebsiteName) –slot staging –target-slot production



Next we need to remove any traffic from the staging site.

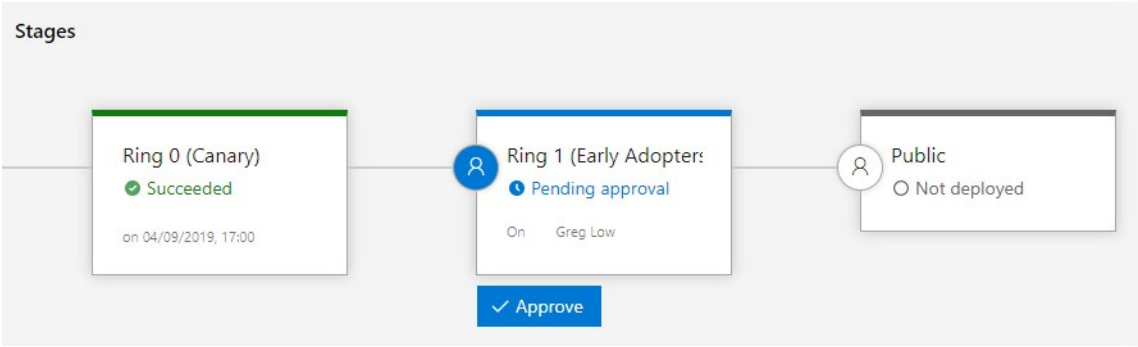
13. Right-click the **Swap sites** task, and click **Clone tasks(s)**. Select the **Swap sites copy** task, change its **Display name** to **Stop staging traffic**, and set the **Inline Script** to the following:

az webapp traffic-routing set –resource-group $(ResourceGroupName) –name $(WebsiteName) –distribution staging=0

14. Click **Save** then **OK** to save your work.

15. Click **Create release** and **Create** to start a release. Click the release link to see the progress.

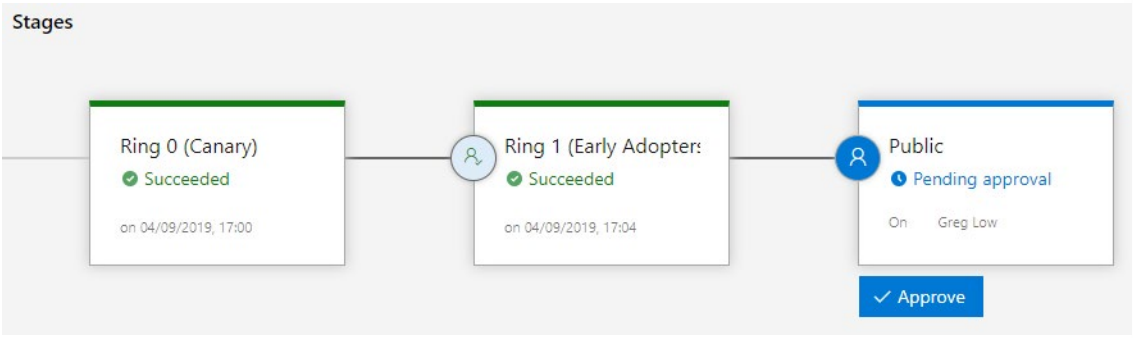
Wait until Ring 0 (Canary) has succeeded.



At this point, 10% of the traffic will be going to the new feature site.

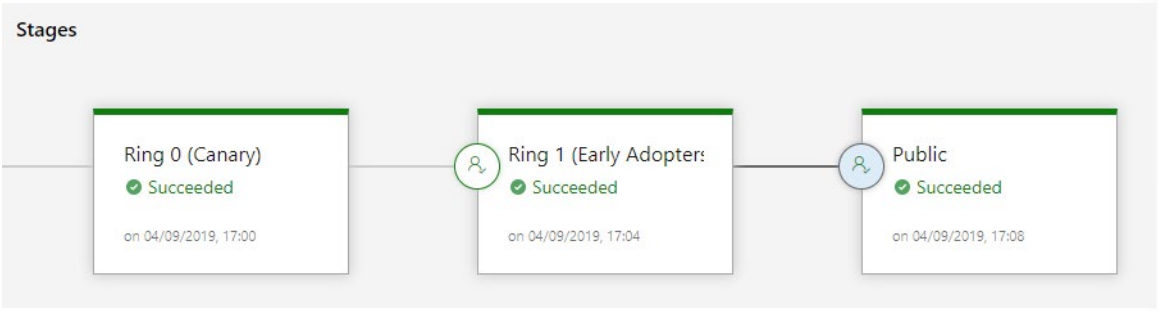
16. Click **Approve** on the **Ring 1 (Early Adopters)** stage, and then **Approve**.

When this stage completes, 30% of the traffic will now be going to the early adopters in ring 1.



17. Click **Approve** on the **Public** stage, and then **Approve**.

When this stage completes, all of the traffic will now be going to the swapped production site, running the new code.



The new feature has been fully deployed.

Lab: Traffic Manager

Overview

In this lab, you will learn how to set up Azure Traffic Manager to use in you Ring Based Deployments

Before you begin

Refer the **Azure DevOps Labs Getting Started**[13] page to know the prerequisites for this lab.

Complete **Task 1 from the Azure DevOps Lab Prerequisites**[14]

Setting up the Target Environment

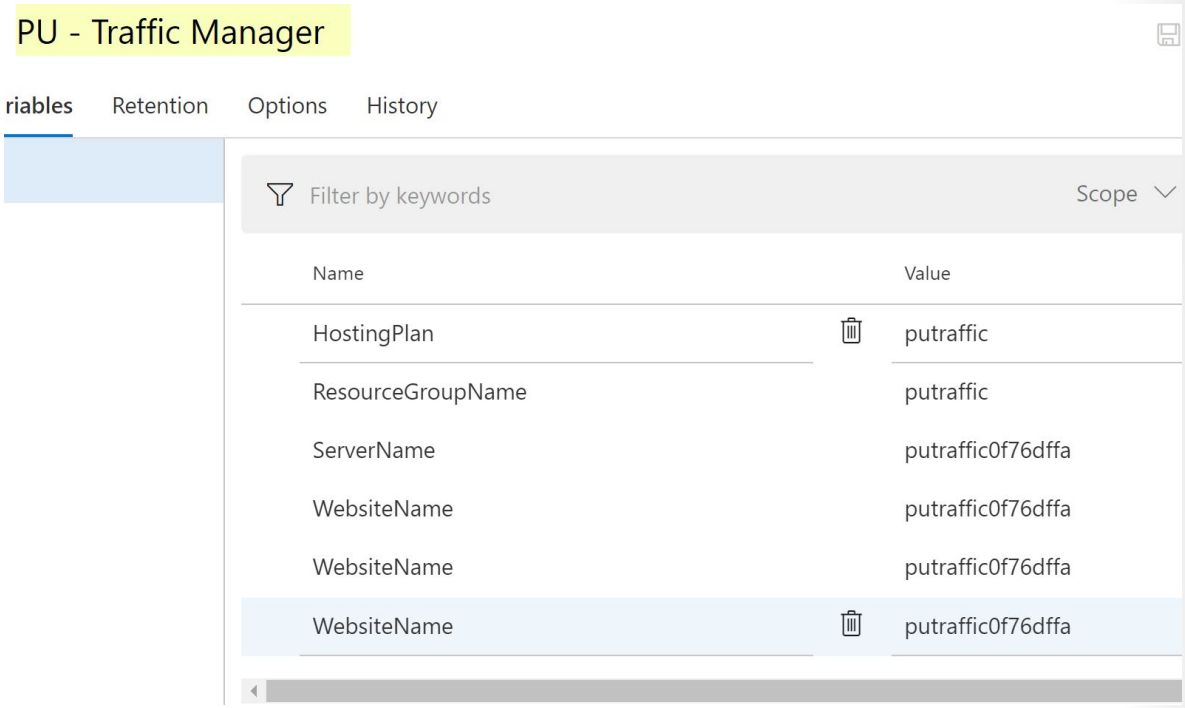
You will run one build and one release to get some data in your project. Make sure you have an Azure account where you can deploy the application.

1. Navigate to **PipelinesReleases** and configure the Service Connections of all stages in the **PartsUnlimitedE2E**. Set the deployment slot name of the Dev Stage to *Dev* and the QA stage to *Staging*
2. In you Azure DevOps Project Go to **Pipelines** and click **Builds** and **Queue** a new **PartsUnlimitedE2E** build.
3. The application is built and deployed automatically

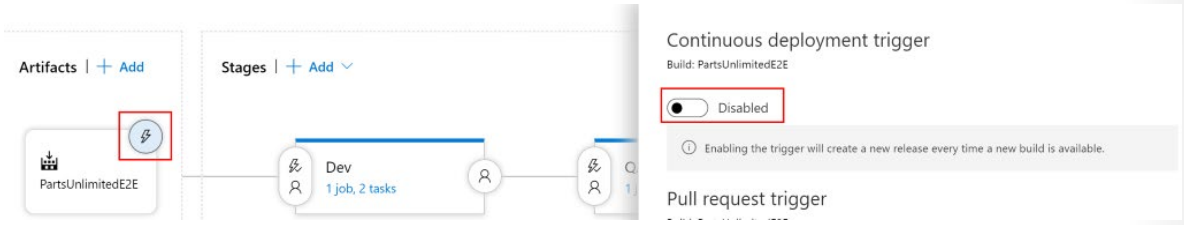
Exercise 1: Set Up Traffic Manager

1. In your Azure DevOps project make sure you deployed the PartsUnlimited web application successfully to Azure
2. **Clone** the Release Pipeline and name it PU-TrafficManager
3. **Update** the variables in the PU-TrafficManager pipeline to resemble this. e.g., putraffic

13 https://azuredevopslabs.com/labs/vstsextend/Setup/
14 https://azuredevopslabs.com/labs/azuredevops/prereq/#task-1-configuring-the-parts-unlimited-team-project



4. **Disable** the Continuous Deployment trigger in the PU-TrafficManager pipeline



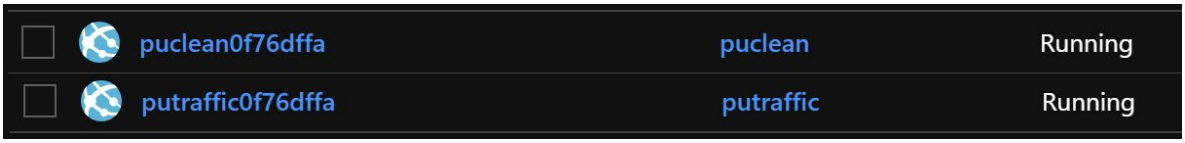
5. In your Azure DevOps Project, navigate to **Repos** and open your **PartsUnlimited** repository. Open the file **_Layout.cshtml** from the folder *PartsUnlimited/PartsUnlimited-aspnet45/src/PartsUnlimitedWebsite/Views/Shared* and add the following lines

```
@Scripts.Render("~/bundles/site")
<style>
    div.container.greendeployment {
        background-color: green;
    }
</style>

....

<body>
<header>
    <nav class="navbar navbar-default">
        <div class="container greendeployment">
            <div class="primary-menu">
```

6. A new PartUnlimitedE2E build will be queued automatically (Continuous Integration). After that, the new version will be automatically deployed to your "old" web application. You now have two websites running. One website with the green banner (we will call this PUclean), and one with the old version (PUTraffic).



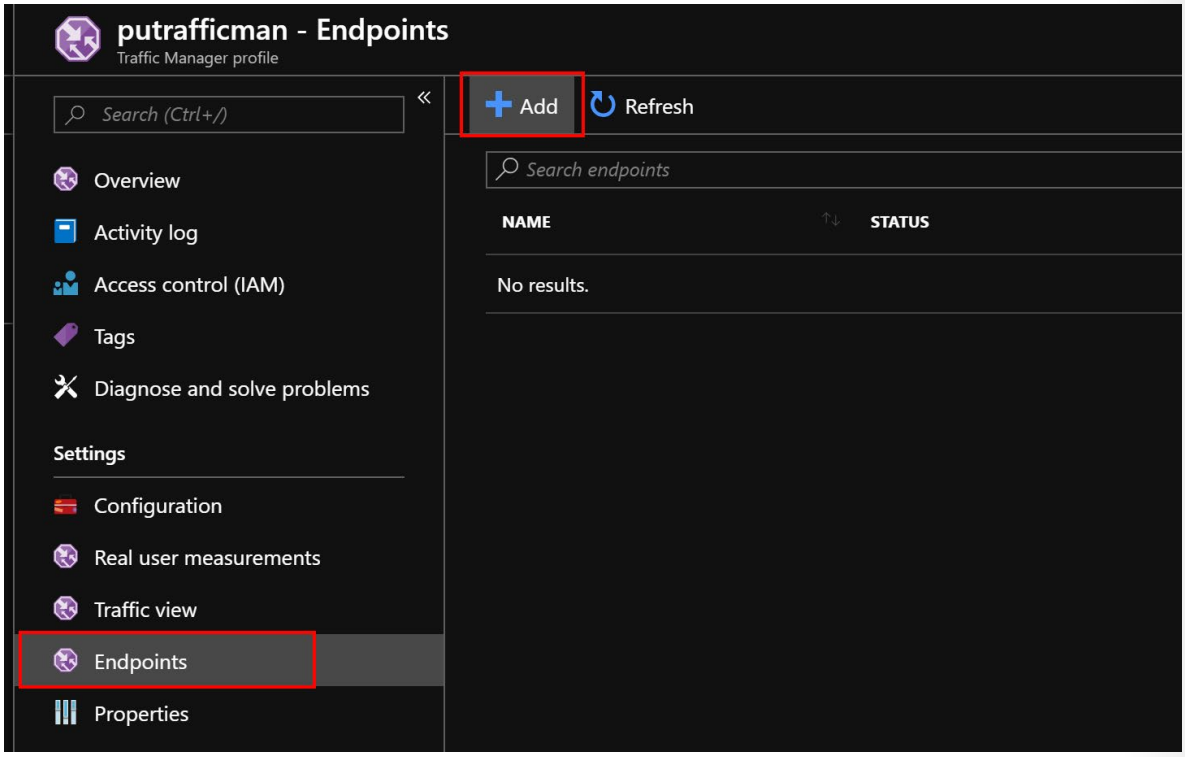
7. We will now create a Traffic Manager Resource in Azure. We can use the Azure Portal to do this, but we can also use the Azure CLI. You should first install the Azure CLI. This can be downloaded from **this website**[15].
8. Open a command prompt and use the command `az login` to login to your Azure Account
9. When you logged in, use the following command to create an Azure Traffic Manager in your "old" resource group

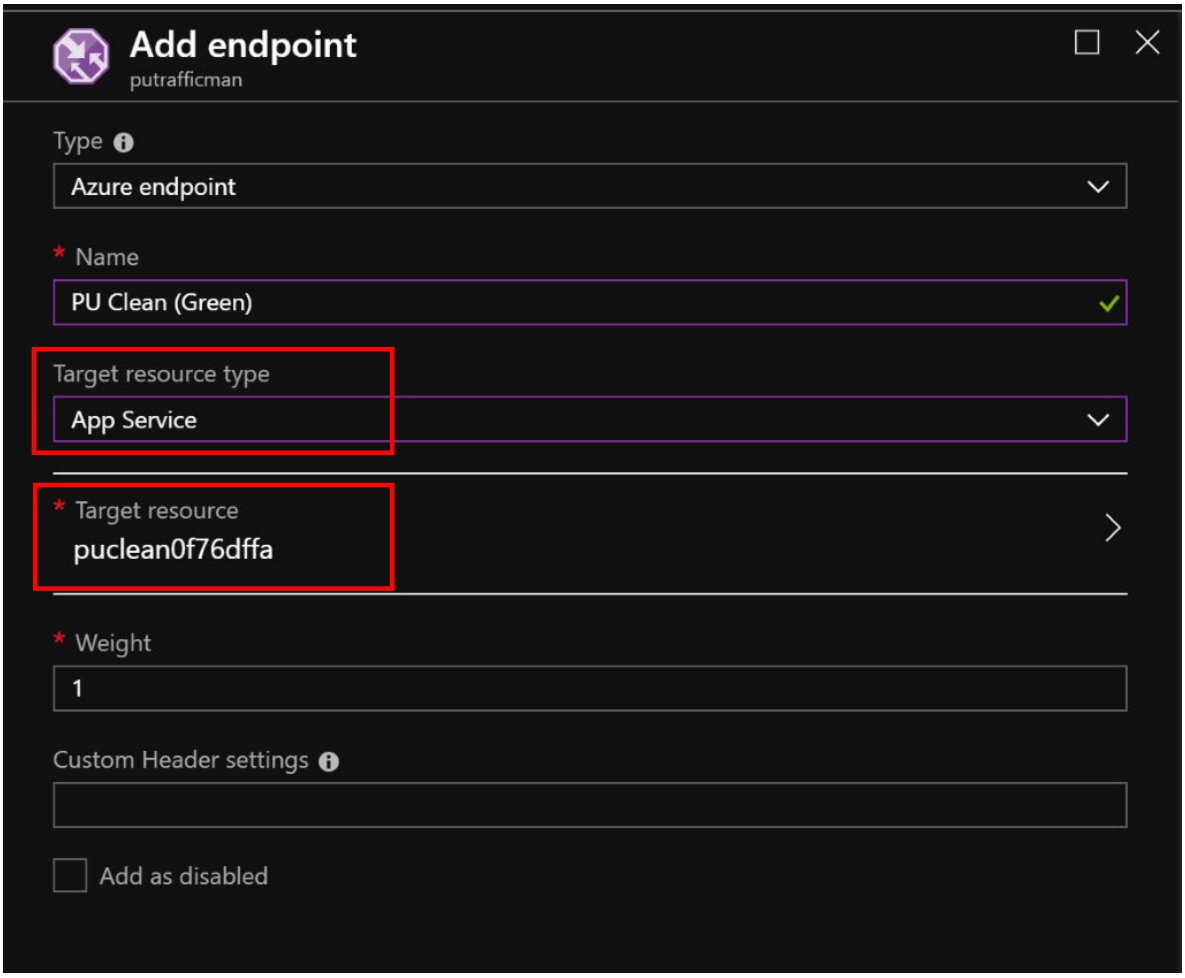
```
az network traffic-manager profile create --resource-group <yourresourcegroup-
name> --routing-method Weighted --name putrafficman --unique-dns-name
<uniquename>
```

10. Open the Azure portal and navigate to the newly created Traffic Manager. Create two endpoints.

- 1 endpoint points to App Service puclean
  - weight 1
- 1 endpoint points to App Service putraffic
  - weight 2

15 https://docs.microsoft.com/en-us/cli/azure/install-azure-cli-windows?view=azure-cli-latest





- To create this with the Azure CLI, use the following command

```
az network traffic-manager endpoint create --resource-group <yourresource-
groupname>  --profile-name putrafficman --name <puclean> --type azureEndpoints
--target-resource-id /subscriptions/subscription guid/resourceGroups/
resourcegroup/providers/Microsoft.Web/sites/websitename --weight 1
```

with this configuration, one request goes to the old website (putraffic, with the white banner)and 2 to the new website (with the green banner)

11. Open the properties of Traffic Manager, and get the URL. This looks like this http://uniquednsname.trafficmanager.net/ . Navigate to this URL and hit refresh a few times. You will see the site changing sometimes. The weight arranges this.
12. Use the Azure Portal or the Azure CLI to update the weight and reroute the traffic

```
az network traffic-manager endpoint update --resource-group <yourresource-
group> --profile-name putrafficman --name "PU Traffic" --type azureEndpoints
--weight 2

az network traffic-manager endpoint update --resource-group <yourresource-
group> --profile-name putrafficman --name "PU Clean" --type azureEndpoints
```

```
--weight 1
```

13. Refresh the browser multiple times or set up a load test to see the impact

You can use these commands in the pipeline as well as using the AZ CLI task.

Tip

If you have trouble accessing your website from a Load Test or location, make sure you configure it to allow TLS 1.0. More information is described in this blog.

Dark Launching

Dark launching

Dark Launching is in many ways similar to Canary Releases. However, the difference here is that you are looking to assess the response of users to new features in your frontend, rather than testing the performance of the backend.

The idea is that rather than launch a new feature for all users, you instead release it to a small set of users. Usually, these users are not aware they are being used as test users for the new feature and often you do not even highlight the new feature to them, hence the term "Dark" launching.

Another example of Dark launching is launching a new feature and use it on the backend to get metrics. Let me illustrate this with a real world "launch" example.

As Elon Musk describes in his biography, he applies all kinds of Agile development principles in his company SpaceX. SpaceX builds and launches rockets to launch satellites. SpaceX also uses Dark Launching. When they have a new version of a sensor, they install it alongside the old one. All data is measured and gathered both by the old and the new sensor. Afterward, they compare the outcomes of both sensors. Only when the new one has the same or improved results the old sensor is replaced.

The same concept can be applied in software. You run all data and calculation through your new feature, but it is not "exposed" yet.

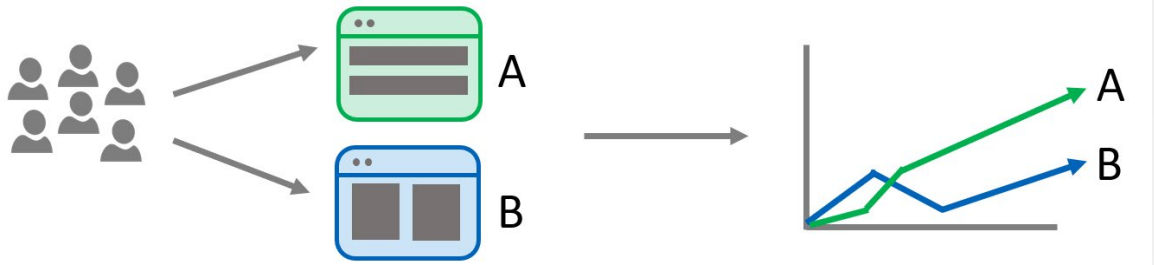
How to implement Dark Launching

In essence dark launching does not differ from a Canary Release or the implementation and switching of a feature toggle. The feature is released and only exposed at a particular time. Therefore, the techniques as described in the previous chapters, do also apply for Dark launching.

AB Testing

A/B Testing

A/B testing (also known as split testing or bucket testing) is a method of comparing two versions of a webpage or app against each other to determine which one performs better. A/B testing is mostly an experiment where two or more variants of a page are shown to users at random, and statistical analysis is used to determine which variation performs better for a given conversion goal.



A/B Testing is not part of Continuous Delivery or a pre-requisite for Continous Delivery. It is more the other way around. Continuous Delivery allows to quickly delivery MVP's to a production environment and your end-users.

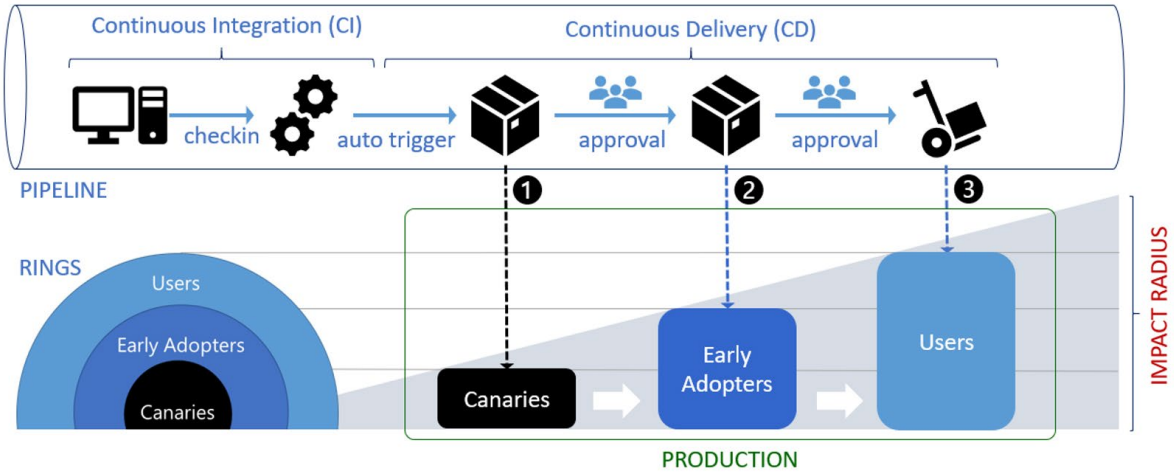
A/B testing is out of scope for this course. But because it is a powerful concept that is enabled by implementing Continous Delivery, it is mentioned here for you to dive into further.

Progressive Exposure Deployment

Progressive Exposure Deployment

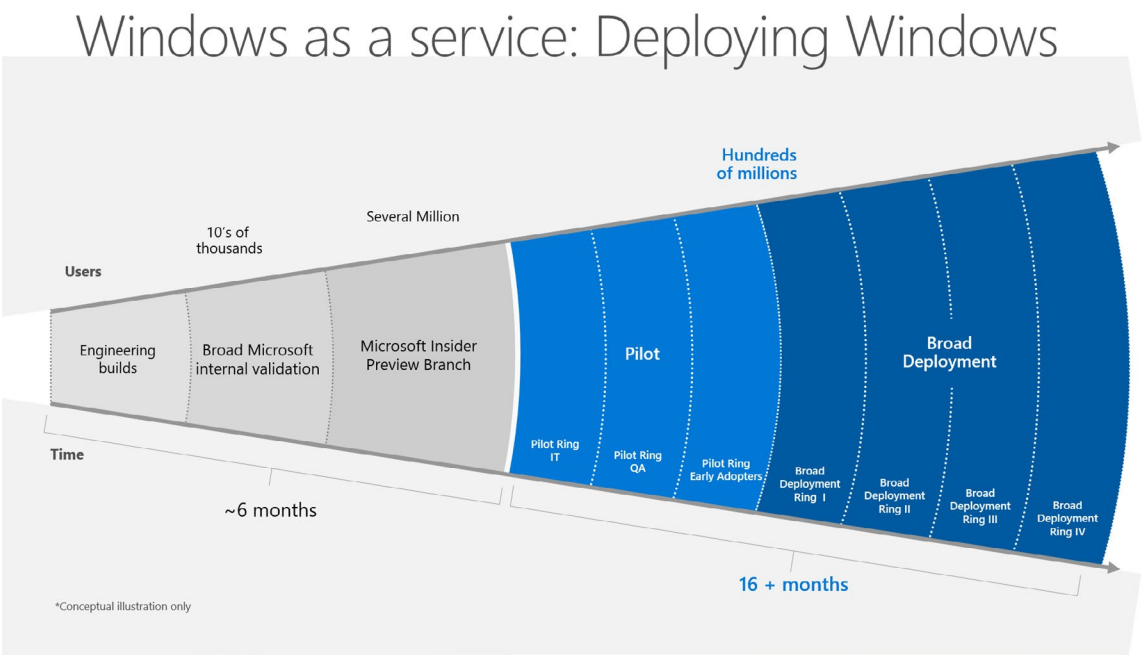
Progressive Exposure Deployment, or also called Ring based deployment, was first discussed in Jez Humble's Continuous Delivery book. They support the production-first DevOps mindset and limit the impact on end users, while gradually deploying and validating changes in production. Impact (also called blast radius), is evaluated through observation, testing, analysis of telemetry, and user feedback.

Rings are in essence an extension of the canary stage. The canary release releases to a stage to measure impact. Adding another ring is essentially the same thing.



With a ring based deployment, you deploy your changes to risk-tolerant customers first, and the progressively roll out to a larger set of customers.

The Microsoft Windows team, for example, uses these rings.



When you have identified multiple groups of users, and you see value in investing in a ring-based deployment, you need to define your setup.

Some organizations that use canary releasing have multiple deployment slots set up as rings. They first release the feature to ring 0 that is targeting a very well known set of users, most of the time only their internal organization. After things have been proven stable in ring 0, they then propagate the release to the next ring that also has a limited set of users, but outside their organization.
And finally, the feature is released to everyone. The way this is often done is just by flipping the switch on the feature toggles in the software.

Just as in the other deployment patterns, monitoring and health checks are essential. By using post-deployment release gates that check a ring for health, you can define an automatic propagation to the next ring after everything is stable. When a ring is not healthy, you can halt the deployment to the next rings to reduce the impact.

Links

Explore how to progressively expose your Azure DevOps extension releases in production to validate, before impacting all users[16]

16 https://docs.microsoft.com/en-us/azure/devops/articles/phase-rollout-with-rings?view=vsts

Review Questions

Module 3 Review Questions

Question 1

What is the easiest way to create a staging environment for an Azure webapp?

Suggested Answer
Create a webslot (deployment slot)

Question 2

What Azure-based tool can you use to divert a percentage of your web traffic to a newer version of an Azure website?

Suggested Answer
Azure Traffic Manager

Question 3

What characteristics make users suitable for working with Canary deployments?

Suggested Answer
High tolerance for issues; Like working with bleeding-edge code.

Question 4

What is a potential disadvantage of using Canary deployments?

Suggested Answer
Needing to look after multiple versions of code at the same time. Or, the users might not be the right ones to test changes in the particular deployment.

Question 5

Apart from the traffic routing method, what else does Azure Traffic Manager consider when making routing decisions?

Suggested Answer
Health of the end point. (It includes built-in endpoint monitoring and automatic endpoint failover)