

Contents

Azure Pipelines

Overview

[What is Azure Pipelines?](#)

[How to use Azure Pipelines](#)

[Sign up for Azure Pipelines](#)

[Key concepts](#)

Quickstarts

[Create your first pipeline](#)

[Use the visual designer](#)

[Use the Azure portal](#)

Languages

[ASP.NET](#)

[.NET Core](#)

[Android](#)

[C/C++](#)

[Docker](#)

[Go](#)

[Helm](#)

[Java](#)

[JavaScript](#)

[PHP](#)

[Python](#)

[Ruby](#)

[Xamarin](#)

[Xcode](#)

Deployments

[Azure Stack](#)

[Azure SQL database](#)

[Azure Web App on Windows](#)

- [Linux VM](#)
- [npm](#)
- [NuGet](#)
- [PyPI](#)
- [SCVMM](#)
- [VMware](#)
- [Azure Web App for Containers](#)
- [Windows VM](#)
- [Artifacts](#)
 - [Overview](#)
 - [Pipeline artifacts \(Preview\)](#)
 - [Build artifacts](#)
 - [Maven](#)
 - [npm](#)
 - [NuGet](#)
 - [Python](#)
 - [Symbols](#)
 - [Universal](#)
- [Concepts](#)
 - [Agents](#)
 - [Build and release agents](#)
 - [Agent pools](#)
 - [Microsoft-hosted agents](#)
 - [Self-hosted Linux agents](#)
 - [Self-hosted macOS agents](#)
 - [Self-hosted Windows agents](#)
 - [Windows agents \(TFS 2015\)](#)
 - [Running behind a proxy](#)
 - [Using a self-signed certificate](#)
 - [Builds](#)
 - [History](#)
 - [Options](#)

[Triggers](#)

[Predefined variables](#)

[Releases](#)

[What is Release Management?](#)

[Release pipelines](#)

[Stages](#)

[Deployment groups](#)

[Provision deployment groups](#)

[Artifacts](#)

[Triggers](#)

[Variables](#)

[Approvals and gates](#)

[Approvals](#)

[Gates](#)

[Templates](#)

[Azure Policy Compliance](#)

[Releases and deployments](#)

[Repositories](#)

[Azure Repos Git](#)

[GitHub](#)

[TFVC](#)

[Pipeline options for Git](#)

[Pipelines](#)

[Tasks](#)

[Jobs](#)

[Jobs](#)

[Container jobs](#)

[Server jobs](#)

[Deployment group jobs](#)

[Multiple jobs](#)

[Conditions](#)

[Expressions](#)

- [Variables](#)
- [Resources](#)
- [Templates](#)
- [Scripts](#)
 - [Cross-platform scripts](#)
 - [Run a PowerShell script](#)
 - [Run Git commands](#)
- [Library](#)
 - [Variable groups](#)
 - [Task groups](#)
 - [Service connections](#)
 - [Secure files](#)
- [Licensing](#)
- [Parallel jobs](#)
- [Pipelines](#)
- [Retention policies](#)
- [Permissions](#)
 - [Set build and release permissions](#)
 - [Permissions & security roles](#)
- [How-to Guides](#)
 - [Build](#)
 - [Build multiple branches](#)
 - [Build on multiple platforms](#)
 - [Build a repo using the visual designer](#)
 - [Release](#)
 - [Connect to Azure](#)
 - [CD to Azure Web App](#)
 - [Azure Kubernetes Service](#)
 - [Set up a multi-stage release](#)
 - [Use approvals and gates](#)
 - [Jenkins CI for Azure Pipelines](#)
 - [Azure CD](#)
 - [Jenkins CI for Azure Pipelines](#)
 - [Azure AKS](#)

- Extend IIS Server deployments
- IIS servers (WinRM)
- Azure Government Cloud
- Azure Cosmos DB CI/CD

Test

- Test glossary
- Set up environments
- Unit test with builds
- UI test with Selenium
- Review test results
- Analyze test results
- Review code coverage
- Trace test requirements
- Use Test Impact Analysis
- Run tests in parallel
- Parallel test with VSTest
- UI testing considerations
- FAQs for testing

Package

- NuGet
 - Restore from Package Management
 - Set up Jenkins + NuGet

Migrate

- Migrate from XAML builds
- Migrate from Lab Management
- Use Build & Release for automated testing

Troubleshooting

- Troubleshoot build and release
- Debug deployment issues
- Troubleshoot Azure connections

Reference

- YAML schema

Tasks

Build tasks

[.NET Core](#)

[.NET Core CLI](#)

[Android build](#)

[Android signing](#)

[Ant](#)

[CMake](#)

[Docker](#)

[Docker Compose](#)

[Go](#)

[Gradle](#)

[Grunt](#)

[gulp](#)

[Index Sources & Publish Symbols](#)

[Jenkins Queue Job](#)

[Maven](#)

[MSBuild](#)

[Visual Studio Build](#)

[Xamarin.Android](#)

[Xamarin.iOS](#)

[Xcode](#)

[Xcode Package iOS](#)

Utility tasks

[Archive files](#)

[Azure Network Load Balancer](#)

[Bash](#)

[Batch script](#)

[Command line](#)

[Copy and Publish Build Artifacts](#)

[Copy Files](#)

[cURL Upload Files](#)

- [Decrypt File](#)
- [Delay](#)
- [Delete Files](#)
- [Download Build Artifacts](#)
- [Download Fileshare Artifacts](#)
- [Download Package](#)
- [Download Pipeline Artifact](#)
- [Download Secure File](#)
- [Extract Files](#)
- [FTP Upload](#)
- [GitHub Release](#)
- [Install Apple Certificate](#)
- [Install Apple Provisioning Profile](#)
- [Install SSH Key](#)
- [Invoke Azure Function](#)
- [Invoke REST API](#)
- [Jenkins Download Artifacts](#)
- [Manual Intervention](#)
- [PowerShell](#)
- [Publish Build Artifacts](#)
- [Publish Pipeline Artifact](#)
- [Publish to Azure Service Bus](#)
- [Python Script](#)
- [Query Azure Monitor Alerts](#)
- [Query Work Items](#)
- [Service Fabric PowerShell](#)
- [Shell script](#)
- [Update Service Fabric App Versions](#)

Test tasks

- [App Center Test](#)
- [Cloud-based Apache JMeter Load Test](#)
- [Cloud-based Load Test](#)

- [Cloud-based Web Performance Test](#)
- [Publish Code Coverage Results](#)
- [Publish Test Results](#)
- [Run Functional Tests](#)
- [Visual Studio Test](#)
- [Visual Studio Test Agent Deployment](#)
- [Package tasks](#)
 - [CocoaPods](#)
 - [Conda Environment](#)
 - [npm](#)
 - [npm Authenticate](#)
 - [NuGet](#)
 - [PyPI Publisher](#)
 - [Python Pip Authenticate](#)
 - [Python Twine Upload Authenticate](#)
 - [Xamarin Component Restore](#)
 - [Previous versions](#)
 - [NuGet Installer 0.*](#)
 - [NuGet Restore 1.*](#)
 - [NuGet Packager 0.*](#)
 - [NuGet Publisher 0.*](#)
 - [NuGet Command 0.*](#)
- [Deploy tasks](#)
 - [App Center Distribute](#)
 - [Azure App Service Deploy](#)
 - [Azure App Service Manage](#)
 - [Azure CLI](#)
 - [Azure Cloud PowerShell Deployment](#)
 - [Azure File Copy](#)
 - [Azure Key Vault](#)
 - [Azure Monitor Alerts](#)
 - [Azure MySQL Deployment](#)

- [Azure Policy Check Gate](#)
- [Azure PowerShell](#)
- [Azure Resource Group Deployment](#)
- [Azure SQL Database Deployment](#)
- [Azure VM Scale Set Deployment](#)
- [Build Machine Image \(Packer\)](#)
- [Chef](#)
- [Chef Knife](#)
- [Copy Files Over SSH](#)
- [Docker](#)
- [Docker Compose](#)
- [Helm Deploy](#)
- [IIS Web App Deploy \(Machine Group\)](#)
- [IIS Web App Manage \(Machine Group\)](#)
- [Kubernetes \(kubectl\)](#)
- [PowerShell on Target Machines](#)
- [Service Fabric App Deployment](#)
- [Service Fabric Compose Deploy](#)
- [SSH](#)
- [Windows Machine File Copy](#)
- [WinRM SQL Server DB Deployment](#)
- [Tool tasks](#)
 - [.NET Core Tool Installer](#)
 - [Go Tool Installer](#)
 - [Helm Installer](#)
 - [Java Tool Installer](#)
 - [Node.js Tool Installer](#)
 - [NuGet Tool Installer](#)
 - [Use Python Version](#)
 - [Use Ruby Version](#)
 - [Visual Studio Test Platform Installer](#)
- [File matching patterns](#)

File and variable transform

Azure Pipelines

Build and release

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.

Team Foundation Server (TFS) 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.

Build your app .NET Core

[Android](#)

[ASP.NET](#)

[C/C++ with GCC](#)

[C/C++ with VC++](#)

[Docker](#)

[Go](#)

[Java](#)

[JavaScript and Node.js](#)

[PHP](#)

[Python](#)

[Ruby](#)

[UWP](#)

[Xamarin](#)

[Xcode](#)

Deploy your app Azure Kubernetes Service

[Azure Stack](#)

[Azure SQL database](#)

[Azure Web Apps](#)

[Linux VM](#)

[npm](#)

[NuGet](#)

[Virtual Machine Manager](#)

[VMware](#)

[Web App for Containers](#)

[Windows VM](#)

[Test your app](#) [Test Reports](#)

[Test Analytics](#)

[Parallel testing for efficient pipelines](#)

[Run Selenium tests](#)

Videos

<https://channel9.msdn.com/Events/Microsoft-Azure/Azure-DevOps-Launch-2018/A101/player>

Build and deploy your code with Azure Pipelines

<https://channel9.msdn.com/Events/Microsoft-Azure/Azure-DevOps-Launch-2018/A102/player>

Continuously build GitHub projects with Azure Pipelines

More information

Step-by-step tutorials

[Build GitHub repositories](#)

[Build multiple branches](#)

[Set up a multi-stage release](#)

Concepts

[Build and release agents](#)

[Parallel jobs](#)

[Release pipelines](#)

Reference

[YAML schema](#)
[Build and release tasks](#)
[Permissions & security roles](#)

Build your app .NET Core

[Android](#)

[ASP.NET](#)

[C/C++ with GCC](#)

[C/C++ with VC++](#)

[Docker](#)

[Go](#)

[Java](#)

[JavaScript and Node.js](#)

[PHP](#)

[Python](#)

[Ruby](#)

[UWP](#)

[Xamarin](#)

[Xcode](#)

Deploy your app Azure SQL database

[Azure Web Apps](#)

[Linux VM](#)

[npm](#)

[NuGet](#)

[Virtual Machine Manager](#)

[VMware](#)

[Web App for Containers](#)

[Windows VM](#)

Test your app [Test Reports](#)

[Test Analytics](#)

[Parallel testing for efficient pipelines](#)

[Run Selenium tests](#)

More information

Step-by-step tutorials

[Build GitHub repositories](#)

[Build multiple branches](#)

[Set up a multi-stage release](#)

Concepts

[Build and release agents](#)

[Parallel jobs](#)

[Release pipelines](#)

Reference

[YAML schema](#)

[Build and release tasks](#)

[Permissions & security roles](#)

TFS 2013: We recommend that you [Migrate from XAML builds to new builds](#). If you're not yet ready to do that, then see [XAML builds](#).

Get started with Azure Pipelines

10/23/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Azure Pipelines is a fully featured continuous integration (CI) and continuous delivery (CD) service. It works with your preferred Git provider and can deploy to most major cloud services, which include Azure services. Start with your code on GitHub, GitLab, Bitbucket, or [Azure Repos](#). Then you can automate the build, test, and deployment of your code to Microsoft Azure, Google Cloud, or Amazon cloud services.

Use Azure Pipelines to configure and automate your build and delivery tools and environments in YAML (as infrastructure as code). Or you can use the visual designer in your Azure DevOps web portal at <https://dev.azure.com>. Azure Pipelines documentation shows you both approaches.

Follow the links in this short guide to sign up and use Azure Pipelines.

What is Azure Pipelines?

For more information on Azure Pipelines, see [What is Azure Pipelines?](#). See how you can use it to automate and streamline the build, test, and deployment of your code projects.

Use Azure Pipelines

Quickly learn the different ways you can [use Azure Pipelines](#) to automate your builds and releases.

Key concepts for Azure Pipelines

The [Key concepts for Azure Pipelines guide](#) explains how Azure Pipelines works. It also explains key terms and concepts.

What is Azure Pipelines?

11/19/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Azure Pipelines is a cloud service that you can use to automatically build and test your code project and make it available to other users. It works with just about any language or project type.

Azure Pipelines combines continuous integration (CI) and continuous delivery (CD) to constantly and consistently test and build your code and ship it to any target.

Does Azure Pipelines work with my language and tools?

Languages

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

Version control systems

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

Application types

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

Deployment targets

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

Package formats

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

What do I need to use Azure Pipelines?

To use Azure Pipelines, you need:

- An Azure DevOps organization.
- To have your source code stored in a version control system.

Pricing

If you use public projects, Azure Pipelines is free. If you use private projects, you can run up to 1,800 minutes (30 hours) of pipeline jobs for free every month. Learn more about how the pricing works based on [parallel jobs](#).

Why should I use CI and CD and Azure Pipelines?

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

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

Use CI and CD for your project

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

Continuous delivery is used to automatically deploy and test code in multiple stages to help drive quality.

Continuous integration systems produce deployable artifacts, which includes infrastructure and apps. Automated release pipelines consume these artifacts to release new versions and fixes to the target of your choice.

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

Use Azure Pipelines for CI and CD

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

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

How do I get started with Azure Pipelines?

To find out the best and easiest way to get started with Azure Pipelines, see the [Get started with Azure Pipelines guide](#).

Use Azure Pipelines

11/9/2018 • 2 minutes to read • [Edit Online](#)

You can use either [YAML](#) or the [visual designer](#) to define your pipelines.

Azure Pipelines

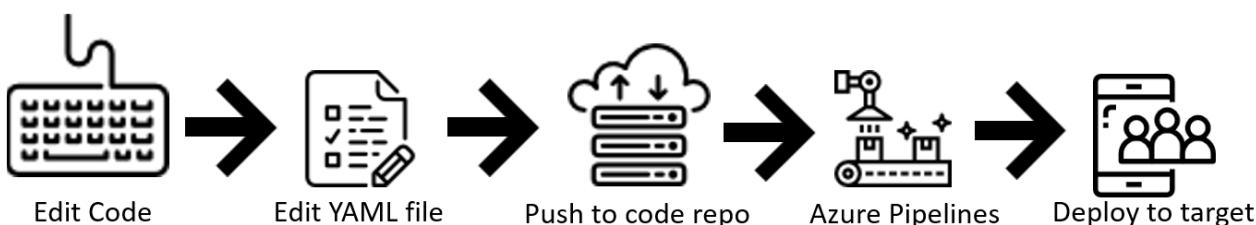
You can either use [YAML](#) to define your pipelines or use the [visual designer](#) to do the same.

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

Use Azure Pipelines with YAML

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

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



Benefits of using YAML

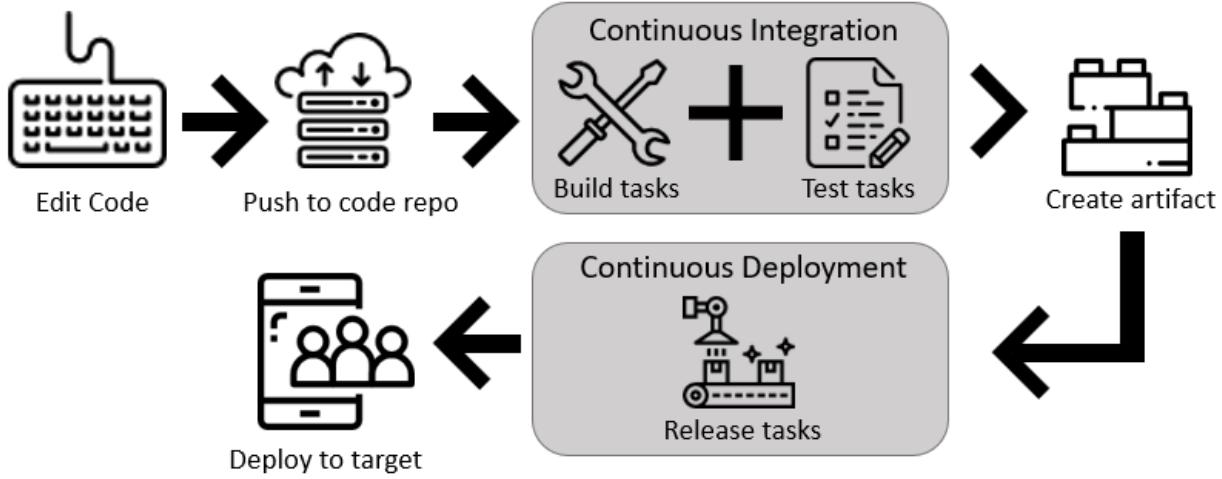
- The pipeline is versioned with your code and follows the same branching structure. You get validation of your changes through code reviews in pull requests and branch build policies.
- Every branch you use can modify the build policy by modifying the `azure-pipelines.yml` file.
- A change to the build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

If you think the YAML workflow is best for you, [create your first pipeline by using YAML](#).

Use Azure Pipelines in the visual designer

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

1. Configure Azure Pipelines to use your Git repo.
2. Use the Azure Pipelines visual designer to create and configure your build and release pipelines.
3. Push your code to your version control repository. This action triggers your pipeline and runs tasks such as building or testing code.
4. The build creates an artifact that's used by the rest of your pipeline to run tasks such as deploying to staging or production.
5. Your code is now updated, built, tested, and packaged. It can be deployed to any target.



Benefits of using the visual designer

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

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

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

Sign up for Azure Pipelines

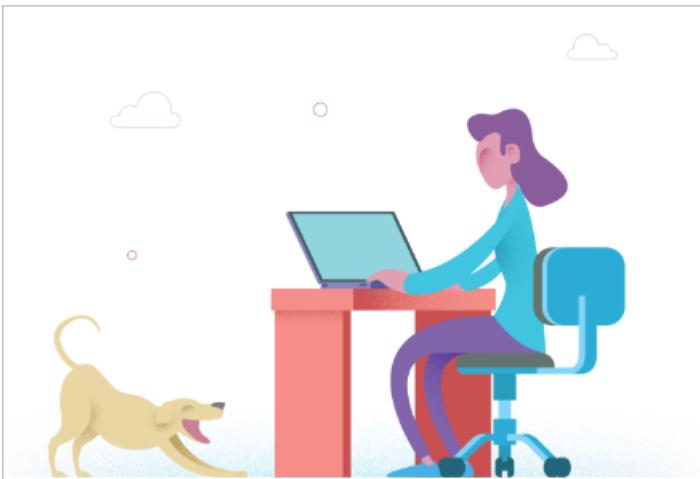
10/23/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

- From the [Azure website](#), select **Start free with Azure Pipelines**.

The screenshot shows the Microsoft Azure website with the URL <https://azure.microsoft.com/en-us/services/devops/pipelines/>. The page title is "Azure Pipelines". The main heading is "Azure Pipelines" followed by the subtext "Continuously build, test, and deploy to any platform and cloud". Below this is a large blue button with the text "Start free with Pipelines". A red box highlights this button. To the right of the text is a cartoon illustration of three people working with a large blue pipe system that ends in a rocket ship launching into the sky. At the bottom left, there is a link "Already have an account? Sign in to Azure DevOps >".

If you already [signed up for Azure DevOps Services](#), on your new project page, select **Azure Pipelines**.



Welcome to the project!

What service would you like to start with?

Boards Repos **Pipelines**

Test Plans

or manage your services

2. Select the location of your code, either **GitHub** or **Azure Repos**.

New build

Where is your code hosted?

Build your code in a few easy steps

- 1 Connection
- 2 Repository
- 3 Template
- 4 Save and run

Azure Repos
Unlimited free private repos

GitHub connected
Home to the world's largest community

3. Grant Azure Pipelines permission to access your GitHub repositories. Select **Authorize** to authorize with OAuth, or select **Install our app from the GitHub Marketplace**.

Select a repository

Grant Azure Pipelines permission to access your GitHub repositories to continue.

Authorize with OAuth

[Authorize](#)

Or, install our app from the GitHub Marketplace

By installing the GitHub App in your GitHub organization, this pipeline can run without using your personal GitHub identity.

[Install our app from the GitHub Marketplace](#)

Select a repository

Fetching 3,442 repositories from GitHub



4. Select a repository.

Select a repository



Filter by keywords

Affiliation: **Owner (+1)**



user/repository



contoso/node-express-realworld-example-app fork private



contoso/tin-octopus private

- ⓘ Showing repositories where you are the owner, or a collaborator. You can also include repositories for which you are an [organization member](#).

Choose a template

Analyzing your repository to recommend a compatible template...

5. Select a template.

Choose a template



Node.js with Gulp

Build a Node.js application using the Gulp task runner.

Recommended

Suggested templates [All templates](#)



Node.js with Grunt

Node.js with Grunt



Empty process

Start with a skeletal process you can flesh out to build your code.

6. Select **Save and run**.

/azure-pipelines.yml

[Save and run](#)

```
1 # Node.js with gulp
2 # Build a Node.js application using the gulp task
3 # https://aka.ms/yaml
4
5 queue: 'Hosted VS2017'
```

7. Select to commit directly to the master branch. Or select to create a new branch for this commit and start a pull request.

Save and run

X

Saving will commit .azure-devops.yml to the repository.

Setup CI with Azure DevOps

Add an optional extended description...

- Commit directly to the master branch
- Create a new branch for this commit and start a pull request.

Cancel

Save and run

Next steps

[Get started with Azure Pipelines](#)

Key concepts for new Azure Pipelines users

10/23/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Learn about the key concepts and components that are used in Azure Pipelines. Understanding the basic terms and parts of Azure Pipelines helps you further explore how it can help you deliver better code more efficiently and reliably.

Agent

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

For more in-depth information about the different types of agents and how to use them, see [Build and release agents](#).

Artifact

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

Build

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

Continuous delivery

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

Continuous integration

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

Deployment target

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

Job

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

Pipeline

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

Release

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

Task

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

Trigger

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

Create your first pipeline

11/26/2018 • 3 minutes to read • [Edit Online](#)

Azure Pipelines

This is a step-by-step guide to using Azure Pipelines to build a GitHub repository.

Prerequisites

- You need an Azure DevOps organization. If you don't have one, you can [create one for free](#). If your team already has one, then make sure you're an administrator of the Azure DevOps project that you want to use. (An Azure DevOps organization is different from your GitHub organization. Give them the same name if you want alignment between them.)
- You need a GitHub account, where you can create a repository.

Get the sample code

You can use Azure Pipelines to build an app written in any language. Select a sample repository of your choice from the following languages and fork it into your own GitHub user account:

PROGRAMMING LANGUAGE	REPOSITORY WITH A SAMPLE APP
.NET Core	https://github.com/MicrosoftDocs/pipelines-dotnet-core
Go	https://github.com/MicrosoftDocs/pipelines-go
Java	https://github.com/MicrosoftDocs/pipelines-java
Node.js	https://github.com/MicrosoftDocs/pipelines-javascript
Python	https://github.com/MicrosoftDocs/pipelines-python-django

You should now have a sample app in your GitHub account.

Get your first build

1. Sign in to your Azure DevOps organization and navigate to your project.
2. In your project, navigate to the **Pipelines** page. Then choose **New pipeline**.
3. Walk through the steps of the wizard by first selecting **GitHub** as the location of your source code.

The screenshot shows the 'New pipeline' wizard in Azure Pipelines. On the left, there's a vertical sidebar with icons for creating a new pipeline, cloning, and more. The main area has a title 'New pipeline' and a subtitle 'Build your code in a few easy steps'. A numbered list on the left indicates four steps: 1. Location (selected), 2. Repository, 3. Template, and 4. Save and run. To the right, there's a section titled 'Where is your code?' with two options: 'GitHub' (selected) and 'Azure Repos'.

4. For the second step, choose to **Authorize with OAuth** by selecting **Authorize**. You might be redirected to GitHub to sign in. Enter your GitHub credentials.
5. When you're redirected back to Azure Pipelines, select the **sample app** repository.
6. For the third step, Azure Pipelines analyzes the code in your repository. If your repository already contains an `azure-pipelines.yml` file, which is the case for all sample repositories, then this step is skipped. Or else, Azure Pipelines recommends a starter template based on the code in your repository.
7. In the final step, you're shown the YAML file that will be used.
8. If you see the **Run** button, then choose it and skip to the next step. If you see the **Save and run** button, then first select **Commit directly to the master branch**, and then choose **Save and run**.
9. Wait for the build to finish.

Add a CI status badge to your repository

Many developers like to show that they're keeping their code quality high by displaying a CI build status badge in their repo.



To copy the status badge to your clipboard:

1. In Azure Pipelines, go to the Build page to view the list of pipelines.
2. Select the pipeline that was created for you.
3. In the context menu for the pipeline, select **Status badge**.

The screenshot shows the Azure Pipelines web interface. On the left, there's a sidebar with various icons for different services like GitHub, Docker, Jenkins, etc. The main area is titled 'ASP.NET Core' and shows a list of build definitions. One definition, 'UnitTest project', has a context menu open. The menu items include 'Commit', 'Security', 'Rename/move', 'Pause builds', 'Add to my favorites', 'Add to team favorites', 'Clone', 'Save as a template', 'Export', and 'Status badge'. The 'Status badge' option is highlighted with a red box. To the right of the build definitions, there's a table with columns for 'Build #' and 'Branch', listing several completed builds.

4. Copy the sample Markdown from the status badge panel.

Now with the status badge in your clipboard, take the following steps in GitHub:

1. Inspect the `azure-pipelines.yml` file at the root of your repository. The YAML file contains the instructions for the pipeline. The next time you change any file in this repository, Azure Pipelines will automatically build your code.
2. Go back to the list of files and select the `Readme.md` file. Then choose **Edit**.
3. Paste the status badge Markdown that you copied in the previous section at the beginning of the `Readme.md` file.
4. Commit the change to the master branch.
5. Notice that the status badge appears in the description of your repository.

Back in Azure Pipelines, observe that a new build is queued. Its status might be either not started or running.

Next steps

You've just learned the basics of using Azure Pipelines. Now you're ready to further configure your pipeline to run tests, publish test results, create container images, or even deploy the app to a cloud service. Follow a track for the language of your choice:

- .NET Core
- Docker
- Go
- Java
- Node.js
- Python

To adjust the timeout of your job, see [Timeouts](#).

To run your pipeline in a container, see [Container jobs](#).

For details about building GitHub repositories, see [Build GitHub repositories](#).

To learn what else you can do in YAML pipelines, see [YAML schema reference](#).

Use the visual designer

11/19/2018 • 17 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

NOTE

This guidance applies to TFS version 2017.3 and newer.

TIP

For build pipelines, we recommend that you use YAML instead of the visual designer that is explained below. YAML allows you to use the same branching and code review practices for your pipeline as you would for your application code. See [Create your first pipeline](#).

We'll show you how to use the visual designer in Azure Pipelines to create a build and release that prints "Hello world". If you plan to use a YAML file instead of the visual designer, then see [Create your first pipeline](#) instead.

We'll show you how to use TFS to create a build and a release that prints "Hello world".

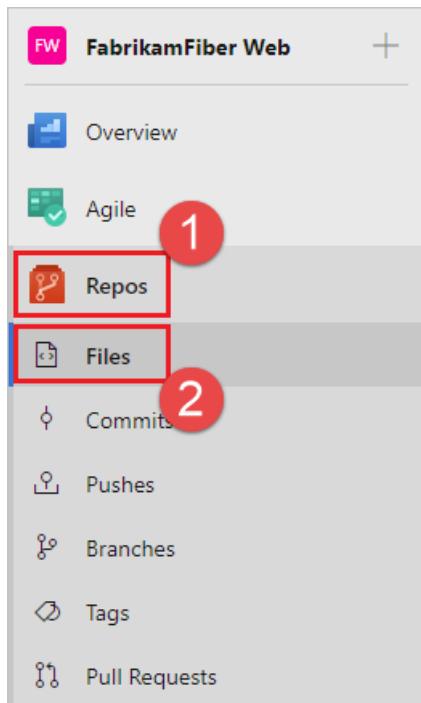
Prerequisites

- You need an Azure DevOps organization. If you don't have one, you can [create one for free](#). If your team already has one, then make sure you're an administrator of the Azure DevOps project that you want to use. (An Azure DevOps organization is different from your GitHub organization. Give them the same name if you want alignment between them.)
- A [self-hosted Windows agent](#).

Initialize your repository

If you already have a repository in your project, you can skip to the next step: [Add a script to your repository](#)

1. Go to **Azure Repos**. (The **Code** hub in the previous navigation)



2. If your project is empty, you will be greeted with a screen to help you add code to your repository.
Choose the bottom choice to **initialize** your repo with a `readme` file:

FabrikamFiber Pipelines is empty. Add some code!

Clone to your computer

HTTPS SSH https://regius.visualstudio.com/FabrikamFiber%20Pipelines/_git/Fabrika... OR [Clone in VS Code](#)

Generate Git credentials

Having problems authenticating in Git? Be sure to get the latest version of [Git for Windows](#) or our plugins for [IntelliJ](#), [Eclipse](#), [Android Studio](#) or [Windows command line](#).

or push an existing repository from command line

HTTPS SSH

```
git remote add origin https://regius.visualstudio.com/FabrikamFiber%20Pipelines/_git/FabrikamFiber%20Pipelines
git push -u origin --all
```

or import a repository

Import

or initialize with a README or gitignore

Add a README

1. Navigate to your repository by clicking **Code** in the top navigation.
2. If your project is empty, you will be greeted with a screen to help you add code to your repository.
Choose the bottom choice to **initialize** your repo with a `readme` file:

FabrikamFiber Pipelines is empty. Add some code!

Clone to your computer

HTTPS SSH https://regius.visualstudio.com/FabrikamFiber%20Pipelines/_git/Fabrika... OR [Clone in VS Code](#)

Generate Git credentials

Having problems authenticating in Git? Be sure to get the latest version of [Git for Windows](#) or our plugins for [IntelliJ](#), [Eclipse](#), [Android Studio](#) or [Windows command line](#).

or push an existing repository from command line

HTTPS SSH

```
git remote add origin https://regius.visualstudio.com/FabrikamFiber%20Pipelines/_git/FabrikamFiber%20Pipelines  
git push -u origin --all
```

or import a repository

Import

or initialize with a README or gitignore

Add a README

Add a .gitignore: None

Initialize

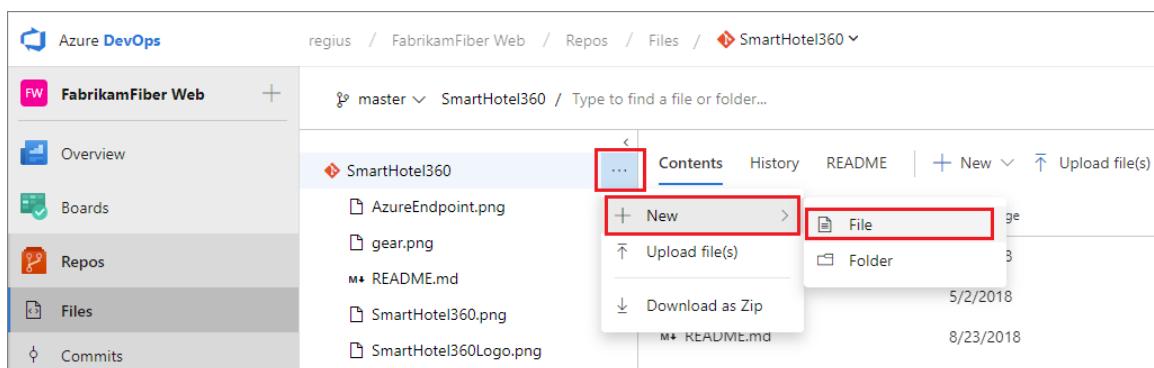
Add a script to your repository

Create a PowerShell script that prints `Hello world`.

1. Go to **Azure Repos**.

2. Add a file.

- [New navigation](#)
- [Previous navigation](#)



3. In the dialog box, name your new file and create it.

HelloWorld.ps1

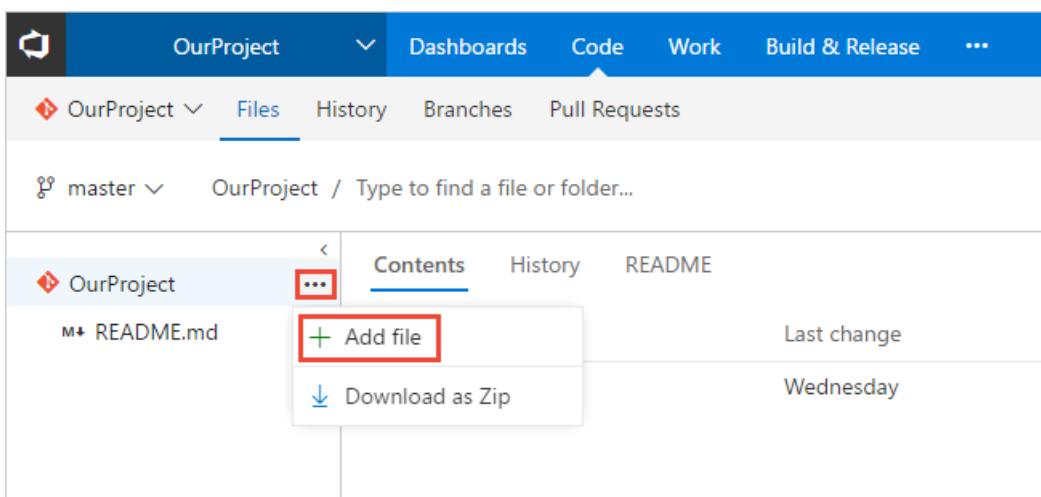
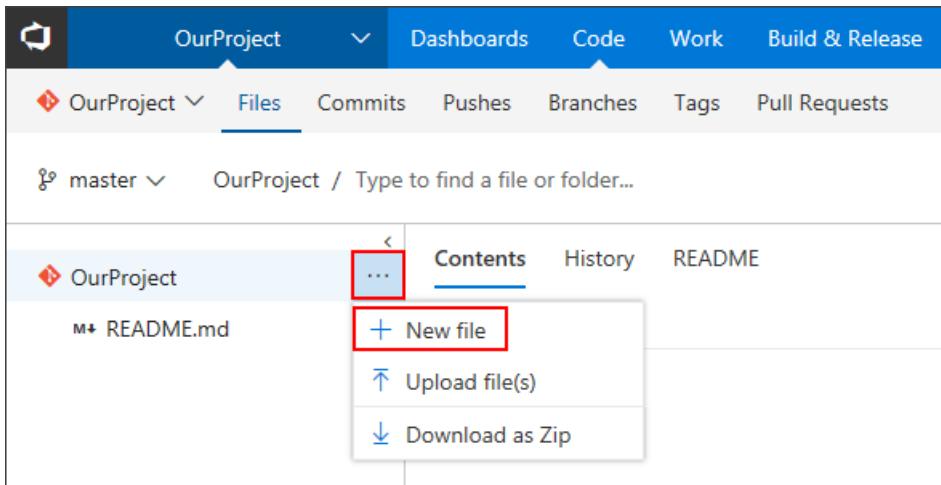
4. Copy and paste this script.

```
Write-Host "Hello world"
```

5. **Commit** (save) the file.

1. Go to the **Code** hub.

2. Add a file.



1. In the dialog box, name your new file and create it.

```
HelloWorld.ps1
```

2. Copy and paste this script.

```
Write-Host "Hello world"
```

3. **Commit** (save) the file.

In this tutorial, our focus is on CI/CD, so we're keeping the code part simple. We're working in an Azure Repos Git repository directly in your web browser.

When you're ready to begin building and deploying a real app, you can use a wide range of version control clients and services with Azure Pipelines CI builds. [Learn more](#).

Create a build pipeline

Create a build pipeline that prints "Hello world."

1. Select **Azure Pipelines**, it should automatically take you to the **Builds** page.

- [New navigation](#)
- [Previous navigation](#)

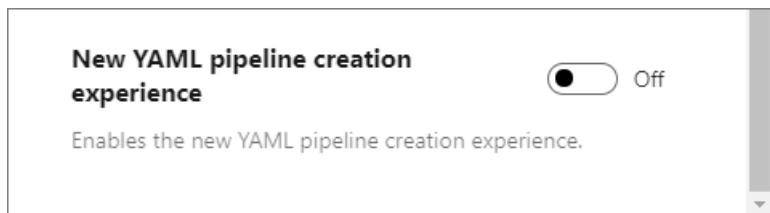
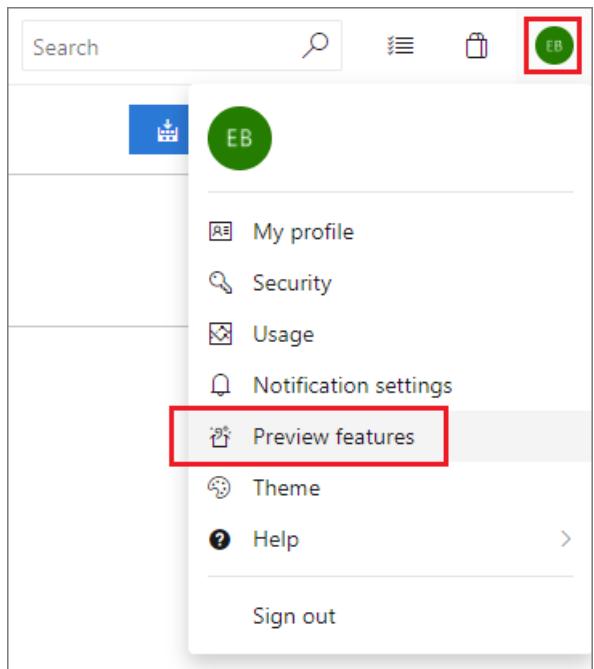
The screenshot shows the Azure DevOps interface for the 'FabrikamFiber Web' project. The left sidebar has a red box around the 'Pipelines' and 'Builds' items. The main area shows a search bar and a list of pipelines, with 'PublicWebCI' listed as the latest run on master 8 hours ago.

2. Create a new pipeline.

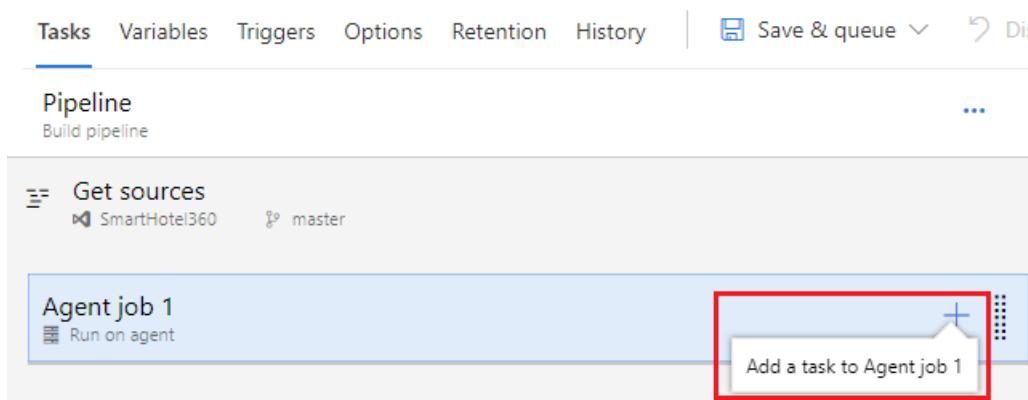
- [New navigation](#)
- [Previous navigation](#)

The screenshot shows the Azure DevOps interface for the 'FabrikamFiber Web' project. The 'New' button in the top right is highlighted with a red box. A dropdown menu appears, showing 'New build pipeline' selected.

For new Azure DevOps accounts, this will automatically take you to the *YAML pipeline creation experience*. To get to the visual designer and complete this guide, you must turn off the **preview feature** for the *New YAML pipeline creation experience*:



3. Make sure that the **source**, **project**, **repository**, and default **branch** match the location in which you created the script.
4. Start with an **Empty job**.
5. On the left side, select **Pipeline** and specify whatever **Name** you want to use. For the **Agent pool**, select **Hosted VS2017**.
6. On the left side, select the plus sign (+) to add a task to **Job 1**. On the right side, select the **Utility** category, select the **PowerShell** task from the list, and then choose **Add**.



7. On the left side, select your new **PowerShell** script task.
8. For the **Script Path** argument, select the **...** button to browse your repository and select the script you created.

The screenshot shows the Azure DevOps Pipeline editor. On the left, a pipeline named "Pipeline" is displayed with two tasks: "Get sources" and "Agent job 1". "Agent job 1" contains a "PowerShell Script" task. On the right, the "PowerShell Script" task is being configured. It has a "Version" dropdown set to "2.*", a "Display name" field containing "PowerShell Script", a "Type" section with "File Path" selected, and a "Script Path" field containing "HelloWorld.ps1" with a browse button (...).

9. Select **Save & queue**, and then select **Save**.

1. Select **Build and Release**, and then choose **Builds**.

The screenshot shows the "Builds" section under "Build and Release". It displays a list of build definitions. One definition, "HelloWorld.ps1", is shown in detail, listing its contents: "HelloWorld.ps1" (last change 9 minutes ago) and "README.md" (last change 17 minutes ago). The "Contents" tab is selected.

2. Create a new pipeline.

The screenshot shows the "Build Definitions" section under "Builds". It includes tabs for "Builds", "Releases", "Library", "Task Groups", and "Deployment Groups". A search bar and a "New" button are visible at the top right.

3. Start with an **empty pipeline**

4. Select **Pipeline** and specify whatever **Name** you want to use. For the **Agent pool**, select **Default**.

5. On the left side, select **+ Add Task** to add a task to the job, and then on the right side select the **Utility** category, select the **PowerShell** task, and then choose **Add**.

The screenshot shows the "Pipeline" editor. The left sidebar lists "Tasks", "Variables", "Triggers", "Options", "Retention", and "History". The main area shows a "Process" step and a "Get sources" task. At the bottom, a red box highlights the "+ Add Task" button.

6. On the left side, select your new **PowerShell** script task.

7. For the **Script Path** argument, select the **...** button to browse your repository and select the script you created.

The screenshot shows the 'Tasks' tab in the Azure Pipelines Task Editor. On the left, there's a list of tasks: 'Process' (Build process), 'Get sources' (OurProject, master), and a selected 'PowerShell Script' task. On the right, the configuration for the 'PowerShell Script' task is shown. It includes fields for 'Display name' (PowerShell Script), 'Type' (File Path), and 'Script Path' (HelloWorld.ps1). A red box highlights the '...' button next to the 'Script Path' field.

8. Select **Save & queue**, and then select **Save**.

1. Select **Azure Pipelines**, and then the **Builds** tab.

The screenshot shows the Azure Pipelines navigation bar. The 'Builds' tab under the 'Build & Release' section is highlighted with a red box. Other options like 'Releases', 'Test', and 'Library' are also visible in the dropdown menu.

2. Create a new pipeline.

The screenshot shows the 'Build & Release' screen. The 'Builds' tab is selected. A red box highlights the '+ New' button, which is used to start a new pipeline definition.

3. Start with an **empty pipeline**.

4. Select **Pipeline** and specify whatever **Name** you want to use.

5. On the **Options** tab, select **Default** for the **Agent pool**, or select whichever pool you want to use that has Windows build agents.

6. On the **Tasks** tab, make sure that **Get sources** is set with the **Repository** and **Branch** in which you created the script.

7. On the left side select **Add Task**, and then on the right side select the **Utility** category, select the **PowerShell** task, and then select **Add**.

8. On the left side, select your new **PowerShell** script task.

9. For the **Script Path** argument, select the **...** button to browse your repository and select the script you created.

The screenshot shows the 'Tasks' tab selected in the build pipeline editor. A 'PowerShell Script' task is highlighted. The configuration pane on the right shows the following fields:

- Display name:** PowerShell Script
- Type:** File Path
- Script Path:** HelloWorld.ps1 (with a red box around the three-dot ellipsis button)

10. Select **Save & queue**, and then select **Save**.

A build pipeline is the entity through which you define your automated build pipeline. In the build pipeline, you compose a set of tasks, each of which perform a step in your build. The task catalog provides a rich set of tasks for you to get started. You can also add PowerShell or shell scripts to your build pipeline.

Publish an artifact from your build

A typical build produces an artifact that can then be deployed to various stages in a release. Here to demonstrate the capability in a simple way, we'll simply publish the script as the artifact.

1. On the **Tasks** tab, select the plus sign (+) to add a task to **Job 1**.
2. Select the **Utility** category, select the **Publish Build Artifacts** task, and then select **Add**.

The screenshot shows the 'Tasks' tab selected in the build pipeline editor. A 'Publish Build Artifacts' task is highlighted. The configuration pane on the right shows the following fields:

- Display name ***: Publish Artifact: drop
- Path to publish ***: HelloWorld.ps1
- Artifact name ***: drop
- Artifact publish location ***: Visual Studio Team Services/TFS

Path to publish: Select the ... button to browse and select the script you created.

Artifact name: Enter drop .

Artifact publish location: Select **Azure Artifacts/TFS**.

1. On the **Tasks** tab, select **Add Task**.
2. Select the **Utility** category, select the **Publish Build Artifacts** task, and then select **Add**.

The screenshot shows the Azure Pipelines interface for a build pipeline named 'Process Build process'. On the left, under the 'Tasks' tab, there is a list of tasks: 'Get sources' (with 'OurProject' and 'master' selected), 'PowerShell Script' (using PowerShell), and 'Publish Artifact: drop' (selected and highlighted with a blue border). Below these is an 'Add Task' button. On the right, the 'Publish Build Artifacts' task is expanded. It has a 'Display name' of 'Publish Artifact: drop'. The 'Path to Publish' field contains 'HelloWorld.ps1' and has a red box around it. The 'Artifact Name' is 'drop' and the 'Artifact Type' is 'Server'. A 'Control Options' dropdown is also present.

Path to Publish: Select the button to browse and select the script you created.

Artifact Name: Enter .

Artifact Type: Select **Server**.

Artifacts are the files that you want your build to produce. Artifacts can be nearly anything your team needs to test or deploy your app. For example, you've got a .DLL and .EXE executable files and .PDB symbols file of a C# or C++ .NET Windows app.

To enable you to produce artifacts, we provide tools such as copying with pattern matching, and a staging directory in which you can gather your artifacts before publishing them. See [Artifacts in Azure Pipelines](#).

Enable continuous integration (CI)

1. Select the **Triggers** tab.
2. Enable **Continuous integration**.

A continuous integration trigger on a build pipeline indicates that the system should automatically queue a new build whenever a code change is committed. You can make the trigger more general or more specific, and also schedule your build (for example, on a nightly basis). See [Build triggers](#).

Save and queue the build

Save and queue a build manually and test your build pipeline.

1. Select **Save & queue**, and then select **Save & queue**.
2. On the dialog box, select **Save & queue** once more.

This queues a new build on the Microsoft-hosted agent.

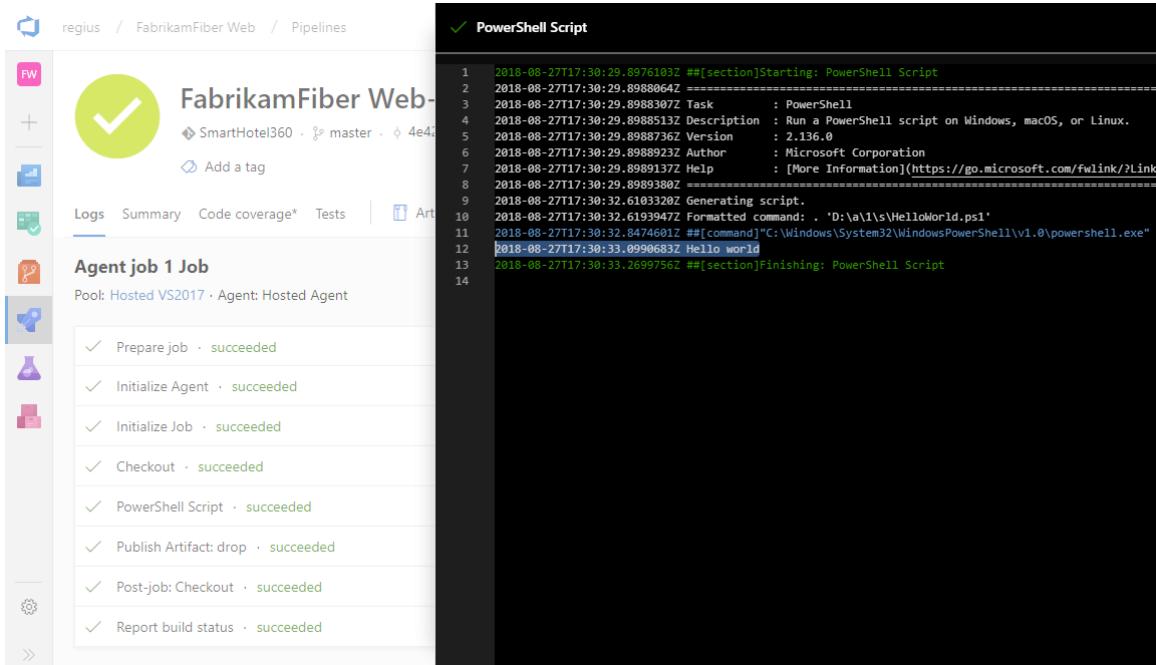
3. You see a link to the new build on the top of the page.
 - [New navigation](#)
 - [Previous navigation](#)



The screenshot shows the Azure Pipelines interface. On the left, there's a sidebar with icons for FW, a plus sign, and a minus sign. The main area has a header "FabrikamFiber Web-CI". Below it, a green notification bar says "Build #116 has been queued.".

Choose the link to watch the new build as it happens. Once the agent is allocated, you'll start seeing the live logs of the build. Notice that the PowerShell script is run as part of the build, and that "Hello world" is printed to the console.

- [New navigation](#)
- [Previous navigation](#)



The screenshot shows the Azure Pipelines build summary for "Agent job 1 Job". It includes tabs for Logs, Summary, Code coverage*, Tests, and Artifacts. The Logs tab is selected, showing the PowerShell script output:

```

1 2018-08-27T17:30:29.897610Z ##[section]Starting: PowerShell Script
2 2018-08-27T17:30:29.898806Z =====
3 2018-08-27T17:30:29.898830Z Task : PowerShell
4 2018-08-27T17:30:29.898851Z Description : Run a PowerShell script on Windows, macOS, or Linux.
5 2018-08-27T17:30:29.898873Z Version : 2.136.0
6 2018-08-27T17:30:29.898892Z Author : Microsoft Corporation
7 2018-08-27T17:30:29.898913Z Help : [More Information](https://go.microsoft.com/fwlink/?LinkId=80000)
8 2018-08-27T17:30:29.898930Z =====
9 2018-08-27T17:30:32.610332Z Generating script.
10 2018-08-27T17:30:32.619394Z Formatted command: . 'D:\a\1\s\HelloWorld.ps1'
11 2018-08-27T17:30:33.847460Z ##[command]"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -ExecutionPolicy Bypass -NoProfile -NonInteractive -File D:\a\1\s\HelloWorld.ps1
12 2018-08-27T17:30:33.899868Z Hello world
13 2018-08-27T17:30:33.269975Z ##[section]Finishing: PowerShell Script
14

```

The "Logs" tab also lists the steps of the job, all of which have succeeded:

- ✓ Prepare job · succeeded
- ✓ Initialize Agent · succeeded
- ✓ Initialize Job · succeeded
- ✓ Checkout · succeeded
- ✓ PowerShell Script · succeeded
- ✓ Publish Artifact: drop · succeeded
- ✓ Post-job: Checkout · succeeded
- ✓ Report build status · succeeded

4. Go to the build summary. On the **Artifacts** tab of the build, notice that the script is published as an artifact.

- [New navigation](#)
- [Previous navigation](#)

regius / FabrikamFiber Web / Pipelines

FabrikamFiber Web-CI 162

SmartHotel360 · master · 68131ea : Updated HelloWorld.ps1 · Manual build · R

Add a tag

Logs **Summary** Code coverage* Tests | Artifacts ▾ Release Edit Queue ▾

Progression

Deployments 0 No deployments were found for this build.

Build artifacts published 1

drop File container

1. Select **Save & queue**, and then select **Save & queue**.

2. On the dialog box, select **Save & queue** once more.

This queues a new build on the Microsoft-hosted agent.

3. You see a link to the new build on the top of the page.

Builds Releases Library Task Groups Deployment Groups

... > OurProject-CI

Build #7 has been queued.

Choose the link to watch the new build as it happens. Once the agent is allocated, you'll start seeing the live logs of the build. Notice that the PowerShell script is run as part of the build, and that "Hello world" is printed to the console.

Screenshot of the Azure Pipelines UI showing the build history and logs.

The left sidebar shows a tree view of builds, with "Build 1722" expanded. Under "Build", the following steps are listed:

- ✓ Initialize Agent
- ✓ Initialize Job
- ✓ Get Sources
- ✓ PowerShell Script
- ✓ Publish Artifact: drop
- ✓ Post Job Cleanup

Below these, under "Build", there are two more items:

- ✓ Finalize build
- ✓ Report build status

The main pane displays the build summary:

Hello world / Build 1722 / Build (Build not retained)

Build succeeded

Build
Ran for 4 seconds (Hosted Agent), completed 1 seconds ago

Console Logs Code coverage* Tests

```
git checkout -b <new-branch-name>
HEAD is now at 0ab86c0... Updated HelloWorld.ps1
=====
Finishing: Get Sources
=====
Starting: PowerShell Script
=====
Task      : PowerShell
Description : Run a PowerShell script
Version   : 1.2.3
Author    : Microsoft Corporation
Help      : [More Information](https://go.microsoft.com/fwlink/?LinkID=613736)
=====
'd:\a\1\s\HelloWorld.ps1'
Hello world
```

4. Go to the build summary.

Screenshot of the Azure Pipelines UI showing the build summary.

The left sidebar shows a tree view of builds, with "Build 1722" expanded. Under "Build", the following steps are listed:

- ✓ Initialize Agent

The main pane displays the build summary:

Hello world / Build 1722 / Build (Build not retained)

Build succeeded

5. On the **Artifacts** tab of the build, notice that the script is published as an artifact.

Screenshot of the Azure Pipelines UI showing the artifacts tab.

The left sidebar shows a tree view of builds, with "Build 1722" expanded. Under "Build", the following steps are listed:

- ✓ Initialize Agent

The main pane displays the build summary:

Hello world / Build 1722 / Build (Build not retained)

Build succeeded

Build 1722
Ran for 31 seconds (Hosted), completed 15.4 minutes ago

Summary Timeline **Artifacts** Code coverage* Tests

Name ↑

drop	Download	Explore
------	----------	---------

Artifacts Explorer

- drop
 - HelloWorld.ps1

You can view a summary of all the builds or drill into the logs for each build at any time by navigating to the **Builds** tab in **Azure Pipelines**. For each build, you can also view a list of commits that were built and the work items associated with each commit. You can also run tests in each build and analyze the test failures.

1. Select **Save & queue**, and then select **Save & queue**.

2. On the dialog box, select the **Queue** button.

This queues a new build on the agent. Once the agent is allocated, you'll start seeing the live logs of the build. Notice that the PowerShell script is run as part of the build, and that "Hello world" is printed to the console.

The screenshot shows the Azure DevOps interface for a build named 'Build 1722'. The 'Build' task is selected in the left sidebar. The main area displays a green banner at the top stating 'Build succeeded'. Below it, there's a summary: 'Build' ran for 4 seconds on a Hosted Agent and completed 1 second ago. Underneath, there are tabs for 'Console', 'Logs', 'Code coverage*', and 'Tests'. The 'Console' tab shows the output of a PowerShell script:

```
git checkout -b <new-branch-name>
HEAD is now at 0ab86c0... Updated HelloWorld.ps1
=====
Finishing: Get Sources
=====
Starting: PowerShell Script
=====
Task      : PowerShell
Description : Run a PowerShell script
Version   : 1.2.3
Author    : Microsoft Corporation
Help      : [More Information](https://go.microsoft.com/fwlink/?LinkID=613736)
=====
.'d:\a\1\HelloWorld.ps1'
Hello world
```

3. Go to the build summary.

The screenshot shows the Azure DevOps interface for a build named 'Build 1722'. The 'Build' task is selected in the left sidebar. The main area displays a green banner at the top stating 'Build succeeded'. Below it, there's a summary: 'Build' ran for 4 seconds on a Hosted Agent and completed 1 second ago. Underneath, there are tabs for 'Console', 'Logs', 'Code coverage*', and 'Tests'. The 'Console' tab shows the output of a PowerShell script. A red box highlights the build ID 'Build 1722' in the URL bar.

4. On the **Artifacts** tab of the build, notice that the script is published as an artifact.

The screenshot shows the Azure DevOps interface after a build has succeeded. At the top, it says "Hello world / Build 1722" and "Build not retained". Below that are buttons for "Edit build definition", "Queue new build...", "Download all logs as zip", and "Release". A green bar at the top indicates "Build succeeded". Below the bar, it says "Build 1722" and "Ran for 31 seconds (Hosted), completed 15.4 minutes ago". There are tabs for "Summary", "Timeline", "Artifacts" (which is selected and highlighted with a red box), "Code coverage*", and "Tests". Under the "Artifacts" tab, there's a "drop" folder listed. Below the folder are two buttons: "Download" and "Explore", with "Explore" also highlighted with a red box. A modal window titled "Artifacts Explorer" is open, showing the contents of the "drop" folder: "drop" and "HelloWorld.ps1".

You can view a summary of all the builds or drill into the logs for each build at any time by navigating to the **Builds** tab in **Build and Release**. For each build, you can also view a list of commits that were built and the work items associated with each commit. You can also run tests in each build and analyze the test failures.

Add some variables and commit a change to your script

We'll pass some build variables to the script to make our pipeline a bit more interesting. Then we'll commit a change to a script and watch the CI pipeline run automatically to validate the change.

1. Edit your build pipeline.
2. On the **Tasks** tab, select the PowerShell script task.
3. Add these arguments.
 - [New navigation](#)
 - [Previous navigation](#)

The screenshot shows the Azure DevOps Pipeline editor for the "FabrikamFiber Web-Cl" pipeline. The left sidebar shows the pipeline structure with tasks: "Get sources", "Agent job 1" (containing a "PowerShell Script" task), and "Publish Artifact: drop". The "PowerShell Script" task is currently selected. The right pane shows the configuration for this task. It includes fields for "PowerShell" (version 2.*), "Display name" (PowerShell Script), "Type" (File Path selected), "Script Path" (HelloWorld.ps1), and "Arguments" (-greeter "\$(Build.RequestedFor)" -trigger "\$(Build.Reason)". The "Arguments" field is highlighted with a red box.

The screenshot shows the Azure DevOps build pipeline editor. On the left, there's a list of tasks: 'Get sources' (OurProject, master), 'PowerShell Script' (PowerShell), and 'Publish Artifact: drop' (Publish Build Artifacts). Below these are 'Add Task' and 'Remove Task'. On the right, the 'PowerShell Script' task is selected. It has a 'Display name' of 'PowerShell Script', a 'Type' of 'File Path' (selected from a dropdown), and a 'Script Path' of 'HelloWorld.ps1'. The 'Arguments' field contains the command: '-greeter "\$(Build.RequestedFor)" -trigger "\$(Build.Reason)"'. This argument section is highlighted with a red box.

Arguments

```
-greeter "$(Build.RequestedFor)" -trigger "$(Build.Reason)"
```

4. Save the build pipeline.
5. Go to your **Files** in **Azure Repos** (the **Code** hub in the previous navigation and TFS).
6. Select the **HelloWorld.ps1** file, and then **Edit** the file.
7. Change the script as follows:

```
Param(
[string]$greeter,
[string]$trigger
)
Write-Host "Hello world" from $greeter
Write-Host Trigger: $trigger
```

8. **Commit** (save) the script.
 1. Go to **Azure Pipelines** and select **Queued**. Notice under the **Queued or running** section that a build is automatically triggered by the change that you committed.
 1. Go to the **Build and Release** page and select **Queued**. Notice under the **Queued or running** section that a build is automatically triggered by the change that you committed.
 1. Select the new build that was created and view its log.
 2. Notice that the person who changed the code has their name printed in the greeting message. You also see printed that this was a CI build.
- [New navigation](#)
 - [Previous navigation](#)

```

    1 2018-08-30T17:33:29.1723775Z ##[section]Starting: PowerShell Script
    2 2018-08-30T17:33:29.1729508Z =====
    3 2018-08-30T17:33:29.1729715Z Task : PowerShell
    4 2018-08-30T17:33:29.1729878Z Description : Run a PowerShell script on Windows, macOS, or Linux.
    5 2018-08-30T17:33:29.1730038Z Version : 2.136.0
    6 2018-08-30T17:33:29.1730205Z Author : Microsoft Corporation
    7 2018-08-30T17:33:29.1730373Z Help : [More Information](https://go.microsoft.com/fwlink/?LinkId=85760)
    8 2018-08-30T17:33:29.1730567Z =====
    9 2018-08-30T17:33:30.9668773Z Generating script.
   10 2018-08-30T17:33:30.9756003Z Formatted command: . 'D:\a\1\s\HelloWorld.ps1' -greeter "Elijah Batkoski"
   11 2018-08-30T17:33:31.0882007Z ##[command]"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -NoProfile -Command
   12 2018-08-30T17:33:31.3101400Z Hello world from Elijah Batkoski
   13 2018-08-30T17:33:31.3107687Z Trigger: Manual
   14 2018-08-30T17:33:31.4515161Z ##[section]Finishing: PowerShell Script
   15

```

Build succeeded

PowerShell Script
Ran for 1 seconds (Hosted Agent), completed 74 seconds

Logs

```

    1 2017-04-10T20:55:12.0502205Z ##[section]Starting: PowerShell Script
    2 2017-04-10T20:55:12.0592196Z =====
    3 2017-04-10T20:55:12.0602014Z Task : PowerShell
    4 2017-04-10T20:55:12.0602014Z Description : Run a PowerShell script
    5 2017-04-10T20:55:12.0602014Z Version : 1.2.3
    6 2017-04-10T20:55:12.0602014Z Author : Microsoft Corporation
    7 2017-04-10T20:55:12.0602014Z Help : [More Information](https://go.microsoft.com/fwlink/?LinkId=85760)
    8 2017-04-10T20:55:12.0602014Z =====
    9 2017-04-10T20:55:12.1292010Z ##[command]. 'd:\a\1\s\HelloWorld.ps1' -g
   10 2017-04-10T20:55:12.8952061Z Hello world from Raisa Pokrovskaya
   11 2017-04-10T20:55:12.8952061Z Trigger: IndividualCI
   12 2017-04-10T20:55:12.9002073Z ##[section]Finishing: PowerShell Script

```

We just introduced the concept of build variables in these steps. We printed the value of a variable that is automatically predefined and initialized by the system. You can also define custom variables and use them either in arguments to your tasks, or as environment variables within your scripts. To learn more about variables, see [Build variables](#).

You've got a build pipeline. What's next?

You've just created a build pipeline that automatically builds and validates whatever code is checked in by your team. At this point you can continue to the next section to learn about release pipelines. Or, if you prefer, you can [skip ahead](#) to create a build pipeline for your app.

Create a release pipeline

Define the process for running the script in two stages.

1. Go to the **Pipelines** tab, and then select **Releases**.
2. Select the action to create a **New pipeline**. If a release pipeline is already created, select the plus sign (+) and then select **Create a release pipeline**.
3. Select the action to start with an **Empty job**.
4. Name the stage **QA**.
5. In the Artifacts panel, select + **Add** and specify a **Source (Build pipeline)**. Select **Add**.
6. Select the **Lightning bolt** to trigger continuous deployment and then enable the **Continuous deployment trigger** on the right.

- [New navigation](#)
- [Previous navigation](#)

All pipelines > [New release pipeline](#) Save

[Pipeline](#) [Tasks](#) [Variables](#) [Retention](#) [Options](#) [History](#)

Artifacts | + Add

Stages | + Add ▾

Continuous deployment trigger
Build: _FabrikamFiber Web-CI

Enabled
Creates a release every time a new build is available.

Build branch filters ⓘ
No filters added.
+ Add | ▾

Pull request trigger
Build: _FabrikamFiber Web-CI

Disabled

7. Select the **Tasks** tab and select your **QA** stage.
8. Select the plus sign (+) for the job to add a task to the job.
9. On the **Add tasks** dialog box, select **Utility**, locate the **PowerShell** task, and then select its **Add** button.
10. On the left side, select your new **PowerShell** script task.
11. For the **Script Path** argument, select the ... button to browse your artifacts and select the script you created.
12. Add these **Arguments**:

```
-greeter "$(Release.RequestedFor)" -trigger "$(Build.DefinitionName)"
```

13. On the **Pipeline** tab, select the **QA** stage and select **Clone**.

- [New navigation](#)
- [Previous navigation](#)

[Pipeline](#) [Tasks](#) [Variables](#) [Retention](#) [Options](#) [History](#)

Artifacts | + Add

Stages | + Add ▾

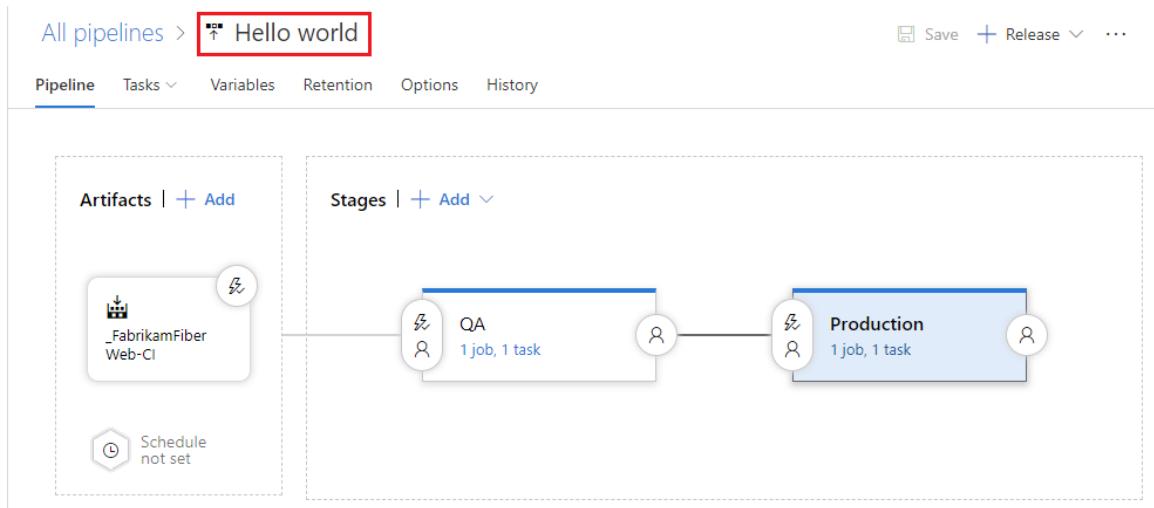
QA
1 job, 1 task

+ Clone

14. Rename the cloned stage **Production**.

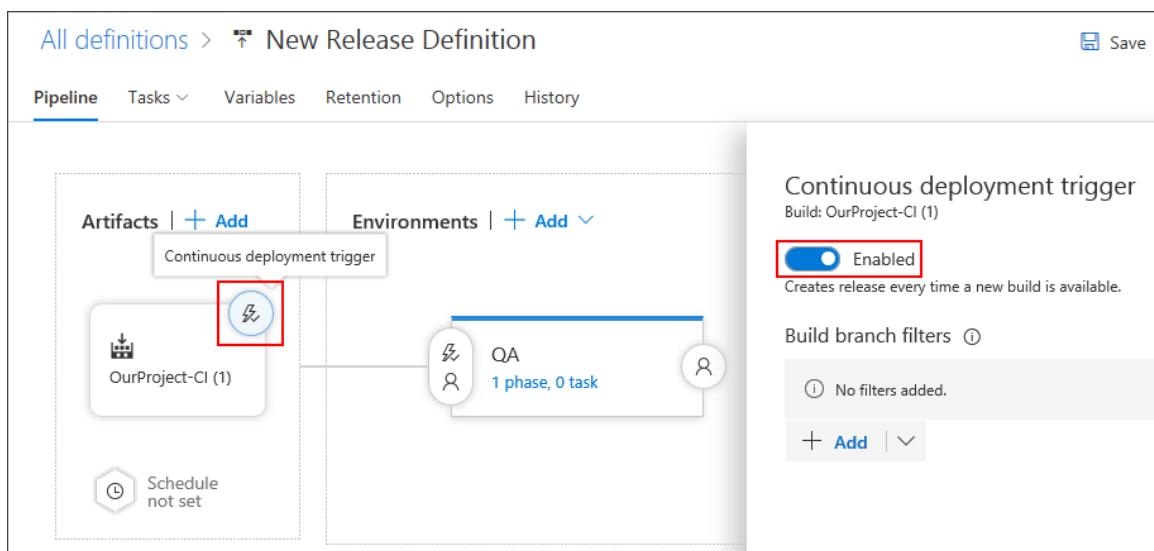
15. Rename the release pipeline **Hello world**.

- [New navigation](#)
- [Previous navigation](#)



16. Save the release pipeline.

1. Go to the **Build and Release** tab, and then select **Releases**.
2. Select the action to create a **New pipeline**. If a release pipeline is already created, select the plus sign (+) and then select **Create a release definition**.
3. Select the action to start with an **Empty definition**.
4. Name the stage **QA**.
5. In the Artifacts panel, select **+ Add** and specify a **Source (Build pipeline)**. Select **Add**.
6. Select the **Lightning bolt** to trigger continuous deployment and then enable the **Continuous deployment trigger** on the right.

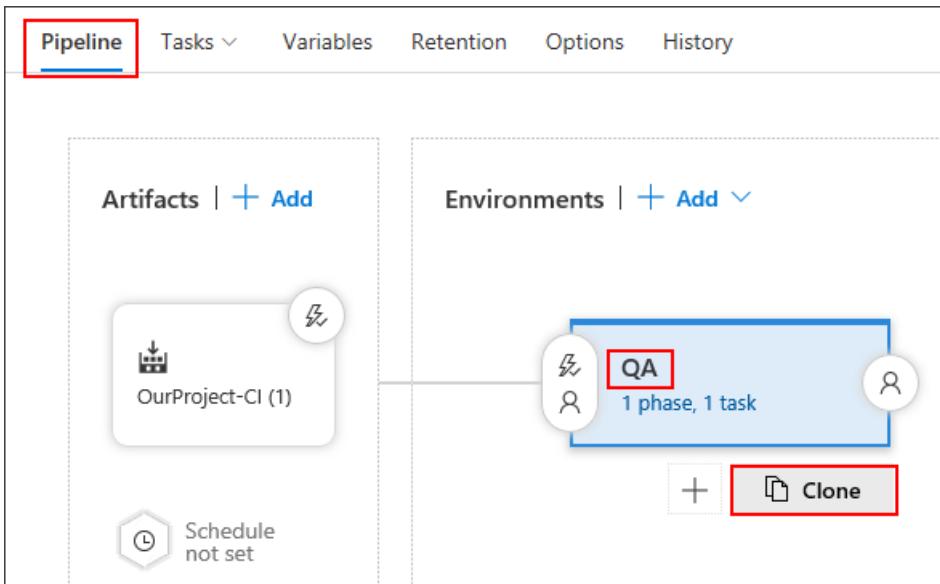


7. Select the **Tasks** tab and select your **QA** stage.
8. Select the plus sign (+) for the job to add a task to the job.
9. On the **Add tasks** dialog box, select **Utility**, locate the **PowerShell** task, and then select its **Add** button.

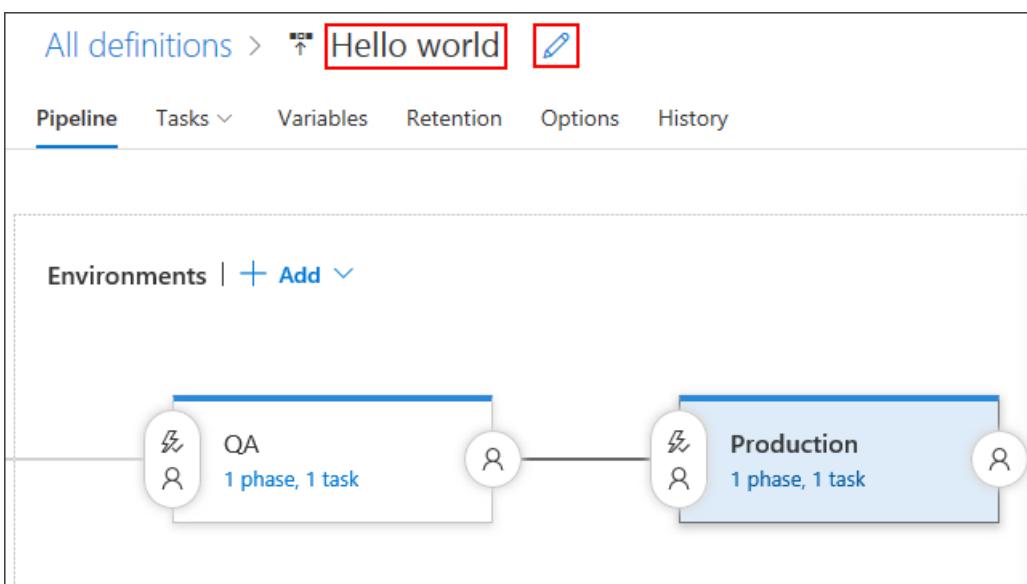
10. On the left side, select your new **PowerShell** script task.
11. For the **Script Path** argument, select the **...** button to browse your artifacts and select the script you created.
12. Add these **Arguments**:

```
-greeter "$(Release.RequestedFor)" -trigger "$(Build.DefinitionName)"
```

13. On the **Pipeline** tab, select the **QA** stage and select **Clone**.



14. Rename the cloned stage **Production**.
15. Rename the release pipeline **Hello world**.



16. Save the release pipeline.
1. Go to **Azure Pipelines**, and then to the **Releases** tab.
2. Select the action to create a **New pipeline**.
3. On the dialog box, select the **Empty** template and select **Next**.
4. Make sure that your **Hello world** build pipeline that you created above is selected. Select **Continuous deployment**, and then select **Create**.

5. Select **Add tasks** in the stage.
6. On the **Task catalog** dialog box, select **Utility**, locate the **PowerShell** task, and then select its **Add** button. Select the **Close** button.
7. For the **Script Path** argument, select the **...** button to browse your artifacts and select the script you created.
8. Add these **Arguments**:

```
-greeter "$(Release.RequestedFor)" -trigger "$(Build.DefinitionName)"
```

9. Rename the stage **QA**.

The screenshot shows the 'Environments' tab of a pipeline definition. The 'QA' environment is selected and highlighted with a red box. It contains one task, a PowerShell script task, which is also highlighted with a red box. The 'Run on agent' section shows the task type as 'PowerShell Script' and the command as 'PowerShell'. The 'Arguments' section contains the command provided in the previous step: `-greeter "$(Release.RequestedFor)" -trigger "$(Build.DefinitionName)"`. Other tabs like 'Artifacts', 'Variables', and 'Triggers' are visible at the top.

10. **Clone** the **QA** stage.

The screenshot shows the 'Environments' tab of a pipeline definition. The 'QA' environment is selected and highlighted with a blue box. A context menu is open over the environment, with the 'Clone environment' option highlighted with a red box. Other options in the menu include 'Assign approvers...', 'Configure variables...', 'Deployment conditions...', 'Delete', 'Save as template...', and 'Security...'. The 'Run on agent' section of the pipeline is visible in the background.

Leave **Automatically approve** and **Deploy automatically...** selected, and select **Create**.

11. Rename the new stage **Production**.
12. Rename the release pipeline **Hello world**.

The screenshot shows the Azure DevOps Releases interface. At the top, there are tabs for Builds, Releases, Packages, Library, Task Groups, Deployment Groups*, and Explorer. The Releases tab is selected. Below the tabs, there's a search bar labeled "Search release definitions..." and a button to "Save". A red box highlights the "Definition*" field, which contains the text "Hello world". To the right of the definition field is a pencil icon. Below the definition, there are tabs for Environments, Artifacts, Variables, Triggers, General, and Retention. Under the Environments tab, there are two stages: "QA" and "Production". Each stage has a status bar indicating "1 / 1 tasks enabled" and a "Run on agent" button. On the right side, there are buttons for "Add environment" and "Add tasks". Under "Add tasks", there is a section for "Run on agent" with a "PowerShell Script" task selected, showing its icon and name.

13. Save the release pipeline.

A release pipeline is a collection of stages to which the application build artifacts are deployed. It also defines the actual deployment pipeline for each stage, as well as how the artifacts are promoted from one stage to another.

Also, notice that we used some variables in our script arguments. In this case, we used [release variables](#) instead of the build variables we used for the build pipeline.

Deploy a release

Run the script in each stage.

1. Create a new release.

- [New navigation](#)
- [Previous navigation](#)

The screenshot shows the "All pipelines" view. At the top, there's a breadcrumb trail "All pipelines > Hello world" and a "Save" button. To the right, there are buttons for "Release" (highlighted with a red box) and "Create a draft release". Below the header, there are tabs for Pipeline, Tasks, Variables, Retention, Options, and History. The Pipeline tab is selected. On the left, there's an "Artifacts" section with a "FabrikamFiber Web-Cl" artifact and a "Schedule not set" button. On the right, there's a "Stages" section showing two stages: "QA" and "Production". Each stage has a status bar indicating "1 job, 1 task". The "QA" stage is highlighted with a blue bar at the top. Below the stages, there are "Add" and "Edit" buttons.

2. Define the trigger settings and artifact source for the release and then select **Create**.

3. Open the release that you just created.

- [New navigation](#)
- [Previous navigation](#)

All pipelines > Hello world

Release **Release1** has been created

Pipeline Tasks Variables Retention Options History

4. View the logs to get real-time data about the release.

- [New navigation](#)
- [Previous navigation](#)

 Hello world > Release-1

Pipeline Variables History | + Deploy | Cancel | Refresh | [Release \(old view\)](#)

Release

Manually triggered
by  Elijah Batkoski
8/27/2018 6:16 PM

Artifacts

 _FabrikamFiber Web-Cl
118
 master

Stages

QA
✓ Succeeded
on 8/27/2018 6:16 PM

 Redeploy  Logs

Production
⌚ Not deployed

1. Create a new release.

All definitions >  Hello World

Save + Release ...

+ Create release **+ Create draft release**

Pipeline Tasks Variables Retention Options History

Artifacts | + Add

 OurProject-CI (2) 

Schedule not set

Environments | + Add

 QA 1 phase, 1 task 

 Production 1 phase, 1 task 

2. Define the trigger settings and artifact source for the release and then select **Queue**.

3. Open the release that you just created.

All definitions > Hello World

Release **Release-2** has been created

Pipeline Tasks Variables Retention Options History

- View the logs to get real-time data about the release.

Hello World / Release-2

Summary Environments Artifacts Variables General Commits Work items Tests **Logs**

Deploy Save Abandon Download all logs as zip

Step	Action	Agent queue: Hosted VS2017
QA	...	Starting: Initialize Job ***** Prepare release directory. ReleaseId=1, TeamProjectId=eb7 Release folder: d:\a\r1\a Environment variables available
Pre-deployment approval	...	
Agent phase	...	
Production	...	

- Create a new release.

Definition: Hello world | Releases

Environments Artifacts Variables Triggers General Retention

Save + Release **Create Release**

+ Add environment QA Create Draft Release Agent PowerShell Script PowerShell

- Open the release that you just created.

Hello world | Edit

Overview **Releases** Deleted

Release **Release-2** has been created.

Lock Title Environments

Release-2 ...

- View the logs to get real-time data about the release.

The screenshot shows the Azure DevOps interface for a release named "Hello world / Release-1". The "Logs" tab is selected. The logs pane displays the following text:

```

Agent: Hosted Agent
Starting: Initialize job
=====
Prepare release directory.
ReleaseId=1, TeamProjectId=eb7...
Release folder: d:\a\r1\...
Environment variables available

```

You can track the progress of each release to see if it has been deployed to all the stages. You can track the commits that are part of each release, the associated work items, and the results of any test runs that you've added to the release pipeline.

Change your code and watch it automatically deploy to production

We'll make one more change to the script. This time it will automatically build and then get deployed all the way to the production stage.

1. Go to the **Code** hub, **Files** tab, edit the **HelloWorld.ps1** file, and change it as follows:

```

Param(
[string]$greeter,
[string]$trigger
)
Write-Host "Hello world" from $greeter
Write-Host Trigger: $trigger
Write-Host "Now that you've got CI/CD, you can automatically deploy your app every time your team checks in code."

```

2. **Commit** (save) the script.
3. Select the **Builds** tab to see the build queued and run.
4. After the build is completed, select the **Releases** tab, open the new release, and then go to the **Logs**.

Your new code automatically is deployed in the **QA** stage, and then in the **Production** stage.

- [New navigation](#)
- [Previous navigation](#)

The screenshot shows the PowerShell Script logs for a deployment. The logs include the following text:

```

1  2018-08-27T18:31:42.7222014Z ##[section]Starting: PowerShell Script
2  2018-08-27T18:31:42.7228660Z =====
3  2018-08-27T18:31:42.7228871Z Task      : PowerShell
4  2018-08-27T18:31:42.7229057Z Description : Run a PowerShell script on Windows, macOS, or Linux.
5  2018-08-27T18:31:42.7229253Z Version   : 2.136.0
6  2018-08-27T18:31:42.7229424Z Author    : Microsoft Corporation
7  2018-08-27T18:31:42.7229609Z Help      : [More Information](https://go.microsoft.com/fwlink/?LinkId=613736)
8  2018-08-27T18:31:42.7229827Z =====
9  2018-08-27T18:31:45.5150962Z Generating script.
10 2018-08-27T18:31:45.5196754Z Formatted command: . 'D:\a\r1\...\_FabrikamFiber Web-CI\drop\HelloWorld.ps1' -greeter "Elijah Batkoski"
11 2018-08-27T18:31:45.6901750Z ##[command]"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -NoLogo -NoProfile -NonInteractive
12 2018-08-27T18:31:45.9197804Z Hello world from Elijah Batkoski
13 2018-08-27T18:31:45.9203491Z Trigger: FabrikamFiber Web-CI
14 2018-08-27T18:31:45.9208413Z Now that you've got CI/CD, you can automatically deploy your app every time your team checks in code.
15 2018-08-27T18:31:46.0566737Z ##[section]Finishing: PowerShell Script
16

```

Hello World / Release-2

Summary Environments Artifacts Variables General Commits Work items Tests Logs History View All Details pane On

Abandon | Download all logs as zip | Send Email

Step	Action	Agent queue: Hosted VS2017 Agent: Hosted Agent
> QA	...	1 2018-04-25T14:53:54.9742188Z ##[section]Starting: PowerShell Script
v Production	...	2 2018-04-25T14:53:54.9746441Z =====
Pre-deployment approval	...	3 2018-04-25T14:53:54.9746591Z Task : PowerShell
v Agent phase	...	4 2018-04-25T14:53:54.9746697Z Description : Run a PowerShell scri
Initialize Agent	...	5 2018-04-25T14:53:54.9746796Z Version : 1.2.3
Initialize Job	...	6 2018-04-25T14:53:54.9746897Z Author : Microsoft Corporation
Download artifact - OurProject-CI (2)	...	7 2018-04-25T14:53:54.9747808Z Help : [More Information](ht
PowerShell Script	...	8 2018-04-25T14:53:54.9747921Z =====
Post-deployment approval	...	9 2018-04-25T14:53:55.0138848Z ##[command]. 'D:\a\1\a\OurProject-CI\HelloWorld\drop\HelloWorld.ps1'
		10 2018-04-25T14:53:57.2485315Z Hello world from Raisa Pokrovskaya
		11 2018-04-25T14:53:57.2486035Z Trigger: Hello world
		12 2018-04-25T14:53:57.2486230Z Now that you've got CI/CD, you can automatically d
		13 2018-04-25T14:53:57.3058801Z ##[section]Finishing: PowerShell Script
		14

Hello world / Release-7

Summary Environments Artifacts Variables General Commits Work items Tests Logs History

Deploy | Save | Abandon | Download all logs as zip | Send Email

Step	Action	Agent: Hosted Agent
> QA	...	1 2017-04-11T12:54:58.5891186Z ##[section]Starting: PowerShell Script
v Production	...	2 2017-04-11T12:54:58.6047463Z =====
Pre-deployment approval	...	3 2017-04-11T12:54:58.6047463Z Task : PowerShell
v Run on agent	...	4 2017-04-11T12:54:58.6047463Z Description : Run a PowerShell script
Initialize Agent	...	5 2017-04-11T12:54:58.6047463Z Version : 1.2.3
Initialize Job	...	6 2017-04-11T12:54:58.6047463Z Author : Microsoft Corporation
Download Artifacts	...	7 2017-04-11T12:54:58.6047463Z Help : [More Information](https://go.micro
PowerShell Script	...	8 2017-04-11T12:54:58.6047463Z =====
Post-deployment approval	...	9 2017-04-11T12:54:58.6672458Z ##[command]. 'd:\a\1\a\Hello_world\drop\HelloWorld.ps1'
		10 2017-04-11T12:54:59.3703946Z Hello world from Raisa Pokrovskaya
		11 2017-04-11T12:54:59.3703946Z Trigger: Hello world
		12 2017-04-11T12:54:59.3703946Z Now that you've got CI/CD, you can automatically d
		13 2017-04-11T12:54:59.4641245Z ##[section]Finishing: PowerShell Script
		14

In many cases, you probably would want to edit the release pipeline so that the production deployment happens only after some testing and approvals are in place. See [Approvals and gates overview](#).

Next steps

You've just learned the basics of using the visual designer to create and run a pipeline. Now you're ready to configure your build pipeline for the programming language you're using. Go ahead and create a new build pipeline, and this time, use one of the following templates.

LANGUAGE	TEMPLATE TO USE
.NET	ASP.NET
.NET Core	ASP.NET Core
C++	.NET Desktop
Go	Go

LANGUAGE	TEMPLATE TO USE
Java	Gradle
JavaScript	Node.js
Xcode	Xcode

Q & A

Where can I read articles about DevOps and CI/CD?

[What is Continuous Integration?](#)

[What is Continuous Delivery?](#)

[What is DevOps?](#)

What kinds of version control can I use

We've used a Git repository in Azure Repos to keep things focused on CI/CD for this tutorial.

When you're ready to get going with CI/CD for your app, you can use the version control system of your choice:

- Clients
 - [Visual Studio Code for Windows, macOS, and Linux](#)
 - [Visual Studio with Git for Windows](#) or [Visual Studio for Mac](#)
 - [Visual Studio with TFVC](#)
 - [Eclipse](#)
 - [Xcode](#)
 - [IntelliJ](#)
 - [Command line](#)
- Services
 - [Azure Pipelines](#)
 - Git service providers such as GitHub and Bitbucket
 - Subversion

How do I replicate a pipeline?

If your pipeline has a pattern that you want to replicate in other pipelines, clone it, export it, or save it as a template.

- [New navigation](#)
- [Previous navigation](#)

Builds Releases Packages Library Task Groups Deployment Groups*

Build Definitions

Search all definitions

Mine All Definitions Queued XAML

↑ Folder / Name Default branch summary Queued

<input checked="" type="checkbox"/>	>HelloWorld-CI	...
	OurProject-CI	Queue new build...
	OurProject-Nightly	Move definition
	OurProject-PreDeployment	View definition summary
		Edit...
		Add to my favorites
		Add to team favorites
		Clone...
		Export
		Rename...
		Save as a template...
		Delete definition
		Security...
		Add to dashboard >

The screenshot shows the 'Build Definitions' page in Azure DevOps. At the top, there are tabs for 'Builds', 'Releases', 'Packages', 'Library', 'Task Groups', and 'Deployment Groups*'. Below the tabs, there's a search bar and a '+ New' button. Underneath, there are filters for 'Mine', 'All Definitions', 'Queued', and 'XAML'. A folder structure is displayed with 'HelloWorld-CI' at the top, followed by 'OurProject-CI', 'OurProject-Nightly', and 'OurProject-PreDeployment'. A context menu is open over 'HelloWorld-CI', showing options like 'Queue new build...', 'Move definition', 'View definition summary', 'Edit...', 'Add to my favorites', 'Add to team favorites', 'Clone...', 'Export' (both of which are highlighted with red boxes), 'Save as a template...' (highlighted with a red box), 'Delete definition', 'Security...', and 'Add to dashboard'.

After you clone a pipeline, you can make changes and then save it.

After you export a pipeline, you can import it from the **All pipelines** tab.

After you create a template, your team members can use it to follow the pattern in new pipelines.

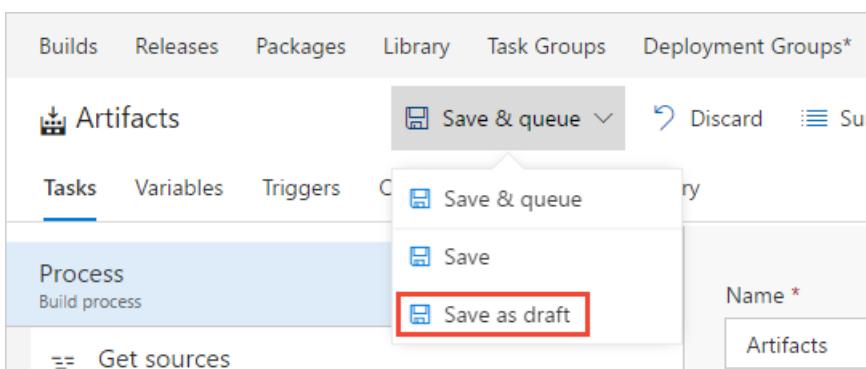
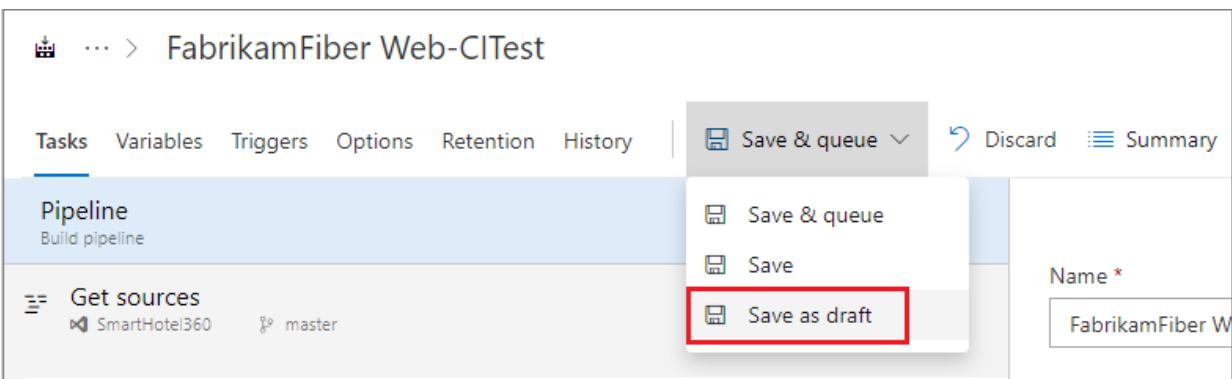
TIP

If you're using the **New Build Editor**, then your custom templates are shown at the bottom of the list.

How do I work with drafts?

If you're editing a build pipeline and you want to test some changes that are not yet ready for production, you can save it as a draft.

- [New navigation](#)
- [Previous navigation](#)



You can edit and test your draft as needed.

- [New navigation](#)
- [Previous navigation](#)

The screenshot shows the 'FabrikamFiber Web-CITest' pipeline in the list. The 'Edit' button in the top right corner is highlighted with a red box. On the left, the pipeline is listed under 'All build pipelines'.

The screenshot shows the 'Build Definitions' list. The 'Artifacts' definition is selected and has a checkmark next to it. A context menu is open over the definition, with the 'Edit...' option highlighted with a red box.

When you're ready you can publish the draft to merge the changes into your build pipeline.

- [New navigation](#)
- [Previous navigation](#)

The screenshot shows the Azure DevOps Pipeline editor interface. At the top, there's a breadcrumb trail: ... > FabrikamFiber Web-CITest. Below it is a toolbar with links: Tasks, Variables, Options, History, Save draft & queue, Discard, Summary, Queue, and Publish draft (which is highlighted with a red box). The main area shows a pipeline named "FabrikamFiber Web-CITest" with a single task: "Get sources" from "SmartHotel360" branch "master". To the right, there's a "Name" field set to "FabrikamFiber Web-CITest". Below this, there's a navigation bar with tabs: Builds, Releases, Packages, Library, Task Groups, Deployment Groups*, Explorer, Artifacts (which is selected and highlighted with a red box), Save draft & queue, Publish draft (highlighted with a red box again), Discard, Queue, and more options.

Or, if you decide to discard the draft, you can delete it from the **All Pipeline** tab shown above.

What else can I do when I queue a build?

You can queue builds [automatically](#) or manually.

When you manually queue a build, you can, for a single run of the build:

- Specify the [pool](#) into which the build goes.
- Add and modify some [variables](#).
- Add [demands](#).
- In a Git repository
 - Build a [branch](#) or a [tag](#).
 - Build a [commit](#).
- In a TFVC repository
 - Specify the source version as a [label](#) or [changeset](#).
 - Run a private build of a [shelveset](#). (You can use this option on either a [Microsoft-hosted agent](#) or a [self-hosted agent](#).)

Where can I learn more about build pipeline settings?

To learn more about build pipeline settings, see:

- [Getting sources](#)
- [Tasks](#)
- [Variables](#)
- [Triggers](#)
- [Options](#)
- [Retention](#)
- [History](#)

How do I programmatically create a build pipeline?

REST API Reference: [Create a build pipeline](#)

NOTE

You can also manage builds and build pipelines from the command line or scripts using the [Azure Pipelines CLI](#).

Use the Azure portal

10/16/2018 • 5 minutes to read • [Edit Online](#)

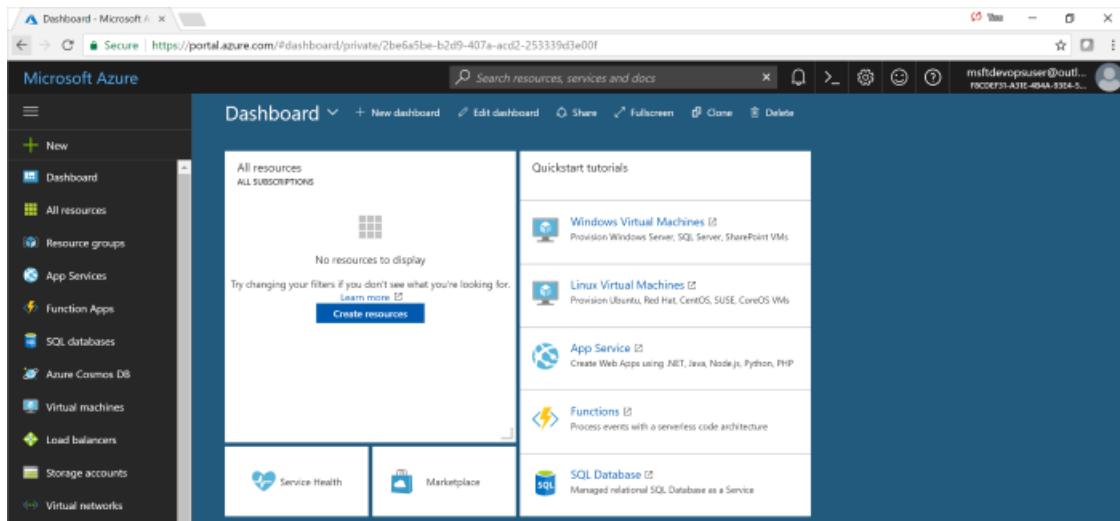
Azure Pipelines

The Azure portal presents a simplified experience to create Azure resources and to set up a continuous integration (CI) and continuous delivery (CD) pipeline for your application to those resources. It uses Azure Pipelines, which is the CI/CD solution for Azure, to configure the pipeline.

If you don't have an Azure subscription, you can get one free through [Visual Studio Dev Essentials](#).

Sign in to the Azure portal

1. Sign into the [Microsoft Azure portal](#).
2. Choose the **Create a resource** icon in the left navigation bar, then search for **DevOps project**. Choose **Create**.



Select a sample application and Azure service

1. Select the language of your choice. For this example, we will choose **Node.js**.
2. Select from a choice of several application frameworks. The default framework for Node.js is **Express.js**. Leave the default setting, and choose **Next**.
3. **Web App on Windows** is the default deployment target. The application framework, which you chose on the previous steps, dictates the type of Azure service deployment target available here. Leave the default service, and then choose **Next**.

Configure an Azure DevOps organization and an Azure subscription

1. Create a **new** Azure DevOps organization or choose an **existing** organization. Choose a **name** for your Azure DevOps project. Select your **Azure subscription**, **location**, and choose a **name** for your application. When you're done, choose **Done**.



Runtime

Framework

Service

4

Create

Almost there!

Ready to deploy an Express.js web app to a new Web App on Windows.

Visual Studio Team Services

A Continuous Delivery pipeline will be setup in Visual Studio Team Services (VSTS).

* Account

Create new Use existing

devopstutorial



* Name

nodesample



Azure

Change

We will create the following Azure resources and deploy an application.

* Subscription

Pay-As-You-Go



* App Service Location

South Central US



* Name

nodesamplesite



.azurewebsites.net

Pricing tier: S1 Standard

Previous

Done

2. In a few minutes, the **project dashboard** loads in the Azure portal. A sample application is set up in a repository in your Azure DevOps organization, a build executes, and your application deploys to Azure. This dashboard provides visibility into your **code repository**, **Azure Pipelines CI/CD pipeline**, and your **application in Azure**. On the right side of the dashboard, select **Browse** to view your running application.

The screenshot shows the Azure DevOps Project dashboard. On the left, the 'CI/CD Pipeline' section displays a flow from 'Code' (nodesampleproject master) through 'Build' (nodesampleprojectsite) to 'Production' (nodesampleprojectsite). A build step named 'Build 20171109.1' is shown as 'Succeeded' 1 min ago. The 'Repository' section shows 1 commit in the past 7 days. On the right, the 'Azure resources' section shows an 'Application endpoint' at <http://nodesampleprojectsites.azurewebsites.net>, which is listed as an 'App Service' named 'nodesampleprojectsites' and is currently 'Running'. The 'Application Insights' section shows a timeline from 3 PM to 8 PM with a single 'SERVER REQUEST' event. The 'Code' button in the Repository section is highlighted.

The Azure DevOps Project automatically configures a CI build and release trigger. You're now ready to collaborate with a team on a Node.js app with a CI/CD pipeline that automatically deploys your latest work to your web site.

Commit code changes and execute CI/CD

The Azure DevOps Project created a Git repository in your Azure Repos or GitHub organization. Follow the steps below to view the repository and make code changes to your application. Some of these steps will vary depending on the language that you started with.

1. On the left-hand side of the DevOps project dashboard, select the link for your **master** branch. This link opens a view to the newly created Git repository.
2. To view the repository clone URL, select **Clone** from the top right of the browser. You can clone your Git repository in your favorite IDE. In the next few steps, you can use the web browser to make and commit code changes directly to the master branch.
3. On the left-hand side of the browser, navigate to the **views/index.pug** file.
4. Select **Edit**, and make a change to the h2 heading. For example, type **Get started right away with the Azure DevOps Project** or make some other change.
5. Choose **Commit**, then save your changes.
6. In your browser, navigate to the **Azure DevOps project dashboard**. You should now see a build is in progress. The changes you just made are automatically built and deployed via an Azure Pipelines CI/CD pipeline.

Examine the Azure Pipelines CI/CD pipeline

The Azure DevOps Project automatically configured a full Azure Pipelines CI/CD pipeline in your Azure DevOps organization. Explore and customize the pipeline as needed. Follow the steps below to familiarize yourself with the Azure Pipelines build and release pipelines.

1. Select **Build Pipelines** from the **top** of the Azure DevOps Project dashboard. This link opens a browser tab and opens the Azure Pipelines build pipeline for your new project.
2. Move the mouse cursor to the right of the build pipeline next to the **Status** field. Select the **ellipsis** that

appears. This action opens a menu where you can perform several activities such as queue a new build, pause a build, and edit the build pipeline.

3. Select **Edit**.
4. From this view, **examine the various tasks** for your build pipeline. The build performs various tasks such as fetching sources from the Git repository, restoring dependencies, and publishing outputs used for deployments.
5. At the top of the build pipeline, select the **build pipeline name**.
6. Change the **name** of your build pipeline to something more descriptive. Select **Save & queue**, then select **Save**.
7. Under your build pipeline name, select **History**. You see an audit trail of your recent changes for the build. Azure Pipelines keeps track of any changes made to the build pipeline, and allows you to compare versions.
8. Select **Triggers**. The Azure DevOps Project automatically created a CI trigger, and every commit to the repository initiates a new build. You can optionally choose to include or exclude branches from the CI pipeline.
9. Select **Retention**. Based on your scenario, you can specify policies to keep or remove a certain number of builds.
10. Select **Pipelines**, then choose **Releases**. The Azure DevOps Project created an Azure Pipelines release pipeline to manage deployments to Azure.
11. On the left-hand side of the browser, select the **ellipsis** next to your release pipeline, then choose **Edit**.
12. The release pipeline contains a **pipeline**, which defines the release pipeline. Under **Artifacts**, select **Drop**. The build pipeline you examined in the previous steps produces the output used for the artifact.
13. To the right-hand side of the **Drop** icon, select the **Continuous deployment trigger**. This release pipeline has an enabled CD trigger, which executes a deployment every time there is a new build artifact available. Optionally, you can disable the trigger, so your deployments require manual execution.
14. On the left-hand side of the browser, select **Tasks**. The tasks are the activities your deployment pipeline performs. In this example, a task was created to deploy to **Azure App service**.
15. On the right-hand side of the browser, select **View releases**. This view shows a history of releases.
16. Select the **ellipsis** next to one of your releases, and choose **Open**. There are several menus to explore from this view such as a release summary, associated work items, and tests.
17. Select **Commits**. This view shows code commits associated with the specific deployment.
18. Select **Logs**. The logs contain useful information about the deployment pipeline. They can be viewed both during and after deployments.

Clean up resources

When no longer needed, you can delete the Azure App service and related resources created in this quickstart by using the **Delete** functionality on the Azure DevOps Project dashboard.

Next steps

When you configured your CI/CD pipeline in this quickstart, a build and release pipeline was automatically created in your Azure DevOps project. You can modify these build and release pipelines to meet the needs of your team. To learn more see this tutorial:

Videos

https://www.youtube.com/embed/_YGR9hOR_PI?rel=0	https://www.youtube.com/embed/3etwjubReJs?rel=0
https://www.youtube.com/embed/itwqMf9aR0w?rel=0	https://www.youtube.com/embed/P72xfZLkFJ0?rel=0

Build ASP.NET apps with Azure Pipelines or Team Foundation Server

12/3/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

This guidance explains how to build .NET Framework projects. For guidance on .NET Core projects, see [this topic](#).

NOTE

This guidance applies to TFS version 2017.3 and newer.

Example

This example shows how to build an ASP.NET project. To start, import (into Azure Repos or TFS) or fork (into GitHub) this repo:

```
https://github.com/Microsoft/devops-project-samples/tree/master/dotnet/aspnet/webapp/Application
```

The sample app is a Visual Studio solution that has two projects: An ASP.NET Web Application project that targets .NET Framework 4.5, and a Unit Test project.

NOTE

This scenario works on TFS, but some of the following instructions might not exactly match the version of TFS that you are using. Also, you'll need to set up a self-hosted agent, possibly also installing software. If you are a new user, you might have a better learning experience by trying this procedure out first using a free Azure DevOps organization. Then change the selector in the upper-left corner of this page from Team Foundation Server to **Azure DevOps**.

- After you have the sample code in your own repository, create a pipeline using the instructions in [Use the designer](#) and select the **ASP.NET Core** template. This automatically adds the tasks required to build the code in the sample repository.
- Save the pipeline and queue a build to see it in action.

Build environment

You can use Azure Pipelines to build your .NET Framework projects without needing to set up any infrastructure of your own. The [Microsoft-hosted agents](#) in Azure Pipelines have several released versions of Visual Studio pre-installed to help you build your projects. Use the **Hosted VS2017** agent pool to build on Visual Studio 2017 or Visual Studio 15.* versions. Use the **Hosted** agent pool to build using the tools in Visual Studio 2013 or Visual Studio 2015.

To change the agent pool on which to build, select **Tasks**, then select the **Process** node, and finally select the **Agent pool** that you want to use.

You can also use a [self-hosted agent](#) to run your builds. This is particularly helpful if you have a large repository and you want to avoid downloading the source code to a fresh machine for every build.

Your builds run on a [self-hosted agent](#). Make sure that you have the necessary version of the Visual Studio installed on the agent.

Build multiple configurations

It is often required to build your app in multiple configurations. The following steps extend the example above to build the app on four configurations: [Debug, x86], [Debug, x64], [Release, x86], [Release, x64].

1. Click the **Variables** tab and modify these variables:

- `BuildConfiguration` = `debug, release`
- `BuildPlatform` = `x86, x64`

2. Select **Tasks** and click on the **agent job** to change the options for the job:

- Select **Multi-configuration**.
- Specify **Multipliers**: `BuildConfiguration, BuildPlatform`

3. Select **Parallel** if you have multiple build agents and want to build your configuration/platform pairings in parallel.

Build .NET Core apps with Azure Pipelines or Team Foundation Server

12/3/2018 • 13 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

This guidance explains how to use Azure Pipelines or Team Foundation Server (TFS) to automatically build .NET Core projects and deploy or publish to targets with CI/CD pipelines.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions.

Service connections are called *service endpoints* in TFS 2018 and in older versions.

This guidance explains how to build .NET Core projects.

NOTE

This guidance applies to TFS version 2017.3 and newer.

Example

This example shows how to build a .NET Core project. To start, import this repo into Azure Repos or TFS, or fork it into GitHub:

<https://github.com/MicrosoftDocs/pipelines-dotnet-core>

- [YAML](#)
- [Designer](#)

The sample code includes an `azure-pipelines.yml` file at the root of the repository. You can use this file to build the app.

Follow all the instructions in [Create your first pipeline](#) to create a pipeline for the sample app.

To learn more about YAML, see [YAML schema reference](#).

YAML builds aren't yet available on TFS.

Read through the rest of this topic to learn some of the common ways to customize your .NET Core pipeline.

Build environment

You can use Azure Pipelines to build your .NET Core projects on Windows, Linux, or macOS without needing to set up any infrastructure of your own. The [Microsoft-hosted agents](#) in Azure Pipelines have several released versions of the .NET Core SDKs preinstalled.

- [YAML](#)
- [Designer](#)

Add the following snippet to your `azure-pipelines.yml` file to select the appropriate agent pool:

```
pool:  
  vmImage: 'ubuntu-16.04' # other options: 'macOS-10.13', 'vs2017-win2016'
```

The Microsoft-hosted agents don't include some of the older versions of the .NET Core SDK. They also don't typically include prerelease versions. If you need these kinds of SDKs on Microsoft-hosted agents, add the **.NET Core Tool Installer** task to the beginning of your process.

- [YAML](#)
- [Designer](#)

If you need a version of the .NET Core SDK that isn't already installed on the Microsoft-hosted agent, add the following snippet to your `azure-pipelines.yml` file:

```
- task: DotNetCoreInstaller@0  
  inputs:  
    version: '2.1.300' # replace this value with the version that you need for your project
```

TIP

As an alternative, you can set up a [self-hosted agent](#) and save the cost of running the tool installer. You can also use self-hosted agents to save additional time if you have a large repository or you run incremental builds.

You can build your .NET Core projects by using the .NET Core SDK and runtime on Windows, Linux, or macOS. Your builds run on a [self-hosted agent](#). Make sure that you have the necessary version of the .NET Core SDK and runtime installed on the agent.

Restore dependencies

NuGet is a popular way to depend on code that you don't build. You can download NuGet packages by running the `dotnet restore` command either through the [.NET Core](#) task or directly in a script in your pipeline.

You can download NuGet packages from Azure Artifacts, NuGet.org, or some other external or internal NuGet repository. The **.NET Core** task is especially useful to restore packages from authenticated NuGet feeds.

You can download NuGet packages from NuGet.org.

`dotnet restore` internally uses a version of `NuGet.exe` that is packaged with the .NET Core SDK. `dotnet restore` can only restore packages specified in the .NET Core project `.csproj` files. If you also have a Microsoft .NET Framework project in your solution or use `package.json` to specify your dependencies, you must also use the **NuGet** task to restore those dependencies.

In .NET Core SDK version 2.0 and newer, packages are restored automatically when running other commands such as `dotnet build`.

In .NET Core SDK version 2.0 and newer, packages are restored automatically when running other commands such as `dotnet build`. However, you might still need to use the **.NET Core** task to restore packages if you use an authenticated feed.

If your builds occasionally fail when restoring packages from NuGet.org due to connection issues, you can use Azure Artifacts in conjunction with [upstream sources](#) and cache the packages. The credentials of the pipeline are automatically used when connecting to Azure Artifacts. These credentials are typically derived from the **Project Collection Build Service** account.

If you want to specify a NuGet repository, put the URLs in a `NuGet.config` file in your repository. If your feed is authenticated, manage its credentials by creating a NuGet service connection in the **Services** tab under **Project Settings**.

If you use Microsoft-hosted agents, you get a new machine every time you run a build, which means restoring the packages every time. This restoration can take a significant amount of time. To mitigate this issue, you can either use Azure Artifacts or a self-hosted agent, in which case, you get the benefit of using the package cache.

- [YAML](#)
- [Designer](#)

To restore packages, use the `dotnet restore` command:

```
- script: dotnet restore
```

Or to restore packages from a custom feed, use the **.NET Core** task:

```
- task: DotNetCoreCLI@2
  inputs:
    command: restore
    projects: '**/*.csproj'
    feedsToUse: config
    nugetConfigPath: NuGet.config      # Relative to root of the repository
    externalFeedCredentials: <Name of the NuGet service connection>
```

For more information about NuGet service connections, see [publish to NuGet feeds](#).

YAML builds aren't yet available on TFS.

Build your project

You build your .NET Core project by running the `dotnet build` command in your pipeline.

- [YAML](#)
- [Designer](#)

To build your project by using the .NET Core task, add the following snippet to your `azure-pipelines.yml` file:

```
- script: dotnet build # Include additional options such as --configuration to meet your need
```

You can run any `dotnet` command in your pipeline. The following example shows how to install and use a .NET global tool, [dotnetsay](#):

```
- script: dotnet tool install -g dotnetsay
- script: dotnetsay
```

YAML builds aren't yet available on TFS.

Run your tests

Use the **.NET Core** task to run unit tests in your .NET Core solution by using testing frameworks like MSTest, xUnit, and NUnit. One benefit of using this built-in task instead of a script to run your tests is that the results of the tests are automatically published to the server. These results are then made available to you in the build summary and can be used for troubleshooting failed tests and test-timing analysis.

- [YAML](#)
- [Designer](#)

Add the following snippet to your `azure-pipelines.yml` file:

```
- task: DotNetCoreCLI@2
  inputs:
    command: test
    projects: '**/*Tests/*.csproj'
    arguments: '--configuration $(buildConfiguration)'
```

An alternative is to run the `dotnet test` command with a specific logger and then use the **Publish Test Results** task:

```
- script: dotnet test <test-project> --logger trx
- task: PublishTestResults@2
  inputs:
    testRunner: VSTest
    testResultsFiles: '**/*.trx'
```

YAML builds aren't yet available on TFS.

Collect code coverage

If you're building on the Windows platform, code coverage metrics can be collected by using the built-in coverage data collector. For this functionality, the test project must reference [Microsoft.NET.Test.SDK](#) version 15.8.0 or higher. If you use the **.NET Core** task to run tests, coverage data is automatically published to the server. The **.coverage** file can be downloaded from the build summary for viewing in Visual Studio.

- [YAML](#)
- [Designer](#)

Add the following snippet to your `azure-pipelines.yml` file:

```
- task: DotNetCoreCLI@2
  inputs:
    command: test
    projects: '**/*Tests/*.csproj'
    arguments: '--configuration $(buildConfiguration) --collect "Code coverage"'
```

If you choose to run the `dotnet test` command, specify the test results logger and coverage options. Then use the **Publish Test Results** task:

```
- script: dotnet test <test-project> --logger trx --collect "Code coverage"
- task: PublishTestResults@2
  inputs:
    testRunner: VSTest
    testResultsFiles: '**/*.trx'
```

YAML builds aren't yet available on TFS.

TIP

If you're building on Linux or macOS, you can use [Coverlet](#) or a similar tool to collect code coverage metrics. Code coverage results can be published to the server by using the [Publish Code Coverage Results](#) task. To leverage this functionality, the coverage tool must be configured to generate results in Cobertura or JaCoCo coverage format.

Package and deliver your code

After you've built and tested your app, you can upload the build output to Azure Pipelines or TFS, create and publish a NuGet package, or package the build output into a .zip file to be deployed to a web application.

- [YAML](#)
- [Designer](#)

Publish artifacts to Azure Pipelines

To simply publish the output of your build to Azure Pipelines, add the following code to your `azure-pipelines.yml` file:

```
- task: PublishBuildArtifacts@1
```

This code takes all the files in `$(Build.ArtifactStagingDirectory)` and upload them as an artifact of your build. For this task to work, you must have already published the output of your build to this directory by using the `dotnet publish --output $(Build.ArtifactStagingDirectory)` command. To copy additional files to this directory before publishing, see [Utility: copy files](#).

Publish to a NuGet feed

To create and publish a NuGet package, add the following snippet:

```
- script: dotnet pack /p:PackageVersion=$(version) # define version variable elsewhere in your pipeline

- task: NuGetCommand@2
  inputs:
    command: push
    nuGetFeedType: external
    publishFeedCredentials: '<Name of the NuGet service connection>'
    versioningScheme: byEnvVar
    versionEnvVar: version
```

For more information about versioning and publishing NuGet packages, see [publish to NuGet feeds](#).

Deploy a web app

To create a .zip file archive that's ready for publishing to a web app, add the following snippet:

```
- task: DotNetCoreCLI@2
  inputs:
    command: publish
    publishWebProjects: True
    arguments: '--configuration $(BuildConfiguration) --output $(Build.ArtifactStagingDirectory)'
    zipAfterPublish: True
```

To publish this archive to a web app, see [Azure Web Apps deployment](#).

YAML builds aren't yet available on TFS.

Build a container

You can build a Docker container image after you build your project. For more information, see [Docker](#).

Troubleshooting

If you're able to build your project on your development machine, but you're having trouble building it on Azure Pipelines or TFS, explore the following potential causes and corrective actions:

- We don't install prerelease versions of the .NET Core SDK on Microsoft-hosted agents. After a new version of the .NET Core SDK is released, it can take a few weeks for us to roll it out to all the datacenters that Azure Pipelines runs on. You don't have to wait for us to finish this rollout. You can use the **.NET Core Tool Installer**, as explained in this guidance, to install the desired version of the .NET Core SDK on Microsoft-hosted agents.
- Check that the versions of the .NET Core SDK and runtime on your development machine match those on the agent. You can include a command-line script `dotnet --version` in your pipeline to print the version of the .NET Core SDK. Either use the **.NET Core Tool Installer**, as explained in this guidance, to deploy the same version on the agent, or update your projects and development machine to the newer version of the .NET Core SDK.
- You might be using some logic in the Visual Studio IDE that isn't encoded in your pipeline. Azure Pipelines or TFS runs each of the commands you specify in the tasks one after the other in a new process. Look at the logs from the Azure Pipelines or TFS build to see the exact commands that ran as part of the build. Repeat the same commands in the same order on your development machine to locate the problem.
- If you have a mixed solution that includes some .NET Core projects and some .NET Framework projects, you should also use the **NuGet** task to restore packages specified in `package.json` files. Similarly, you should add **MSBuild** or **Visual Studio Build** tasks to build the .NET Framework projects.
- If your builds fail intermittently while restoring packages, either NuGet.org is having issues, or there are networking problems between the Azure datacenter and NuGet.org. These aren't under our control, and you might need to explore whether using Azure Artifacts with NuGet.org as an upstream source improves the reliability of your builds.
- Occasionally, when we roll out an update to the hosted images with a new version of the .NET Core SDK or Visual Studio, something might break your build. This can happen, for example, if a newer version or feature of the NuGet tool is shipped with the SDK. To isolate these problems, use the **.NET Core Tool Installer** task to specify the version of the .NET Core SDK that's used in your build.

Q&A

Where can I learn more about Azure Artifacts and the TFS Package Management service?

[Package Management in Azure Artifacts and TFS](#)

Where can I learn more about .NET Core commands?

[.NET Core CLI tools](#)

Where can I learn more about running tests in my solution?

[Unit testing in .NET Core projects](#)

Where can I learn more about tasks?

[Build and release tasks](#)

Build Android apps with Azure Pipelines or Team Foundation Server

12/3/2018 • 4 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

This guidance explains how to use Azure Pipelines or Team Foundation Server (TFS) to automatically build, test, and deploy Android apps with CI/CD pipelines.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions.

Service connections are called *service endpoints* in TFS 2018 and in older versions.

NOTE

This guidance uses YAML-based pipelines available in Azure Pipelines. For TFS, use tasks that correspond to those used in the YAML below.

This guide explains creating pipelines for Android projects. Before this guidance, read the [YAML quickstart](#).

Get started

You can build Android projects using [Microsoft-hosted agents](#) that include tools for Android. Or, you can use [self-hosted agents](#) with specific tools you need.

Sample code

To get started using a sample Android project, fork this repository in GitHub, or import it into Azure Repos or TFS:

```
https://github.com/MicrosoftDocs/pipelines-android
```

Your code

To get started using your own code, add the following YAML to a file named **azure-pipelines.yml** in the root of your repository. Change values to match your project configuration. See the [Gradle](#) task for more about these options.

```
# https://docs.microsoft.com/azure/devops/pipelines/languages/android
pool:
  vmImage: 'macOS-10.13'

steps:
- task: Gradle@2
  inputs:
    workingDirectory: ''
    gradleWrapperFile: 'gradlew'
    gradleOptions: '-Xmx3072m'
    publishJUnitResults: false
    testResultsFiles: '**/TEST-*.xml'
    tasks: 'assembleDebug'
```

Adjust the build path

Adjust the **workingDirectory** value if your `gradlew` file isn't in the root of the repository. The directory value should be relative to the root of the repository, such as `AndroidApps/MyApp` or `$(system.defaultWorkingDirectory)/AndroidApps/MyApp`.

Adjust the **gradleWrapperFile** value if your `gradlew` file isn't in the root of the repository. The file path value should be relative to the root of the repository, such as `AndroidApps/MyApp/gradlew` or `$(system.defaultWorkingDirectory)/AndroidApps/MyApp/gradlew`.

Adjust Gradle tasks

Adjust the **tasks** value for the build variant you prefer, such as `assembleDebug` or `assembleRelease`. For details, see Google's Android development documentation: [Build a debug APK](#) and [Configure build variants](#).

Sign and align an Android APK

If your build does not already [sign and zipalign](#) the APK, add the [Android Signing](#) task to the YAML. An APK must be signed to run on a device instead of an emulator. Zipaligning reduces the RAM consumed by the app.

Important: We recommend storing each of the following passwords in a [secret variable](#).

```
- task: AndroidSigning@2
  inputs:
    apkFiles: '**/*.apk'
    jarsign: true
    jarsignerKeystoreFile: 'pathToYourKeystoreFile'
    jarsignerKeystorePassword: '$(jarsignerKeystorePassword)'
    jarsignerKeystoreAlias: 'yourKeystoreAlias'
    jarsignerKeyPassword: '$(jarsignerKeyPassword)'
    zipalign: true
```

Test on the Android Emulator

Note: The Android Emulator is currently available only on the **Hosted macOS** agent.

Test on Azure-hosted devices

Add the [App Center Test](#) task to test the app in a hosted lab of iOS and Android devices. An [App Center](#) free trial is required which must later be converted to paid.

```

# App Center Test
# Test app packages with Visual Studio App Center.
- task: AppCenterTest@1
  inputs:
    appFile:
    #artifactsDirectory: '$(Build.ArtifactStagingDirectory)/AppCenterTest'
    #prepareTests: # Optional
    #frameworkOption: 'appium' # Required when prepareTests == True# Options: appium, espresso, calabash,
    uitest, xcuitest
    #appiumBuildDirectory: # Required when prepareTests == True && Framework == Appium
    #espressoBuildDirectory: # Optional
    #espressoTestApkFile: # Optional
    #calabashProjectDirectory: # Required when prepareTests == True && Framework == Calabash
    #calabashConfigFile: # Optional
    #calabashProfile: # Optional
    #calabashSkipConfigCheck: # Optional
    #uiTestBuildDirectory: # Required when prepareTests == True && Framework == Uitest
    #uitestStoreFile: # Optional
    #uiTestStorePassword: # Optional
    #uitestKeyAlias: # Optional
    #uiTestKeyPassword: # Optional
    #uiTestToolsDirectory: # Optional
    #signInfo: # Optional
    #xcUITestBuildDirectory: # Optional
    #xcUITestIpaFile: # Optional
    #prepareOptions: # Optional
    #runTests: # Optional
    #credentialsOption: 'serviceEndpoint' # Required when runTests == True# Options: serviceEndpoint, inputs
    #serverEndpoint: # Required when runTests == True && CredsType == ServiceEndpoint
    #username: # Required when runTests == True && CredsType == Inputs
    #password: # Required when runTests == True && CredsType == Inputs
    #appSlug: # Required when runTests == True
    #devices: # Required when runTests == True
    #series: 'master' # Optional
    #dsymDirectory: # Optional
    #localeOption: 'en_US' # Required when runTests == True# Options: da_DK, nl_NL, en_GB, en_US, fr_FR,
de_DE, ja_JP, ru_RU, es_MX, es_ES, user
    #userDefinedLocale: # Optional
    #loginOptions: # Optional
    #runOptions: # Optional
    #skipWaitingForResults: # Optional
    #cliFile: # Optional
    #showDebugOutput: # Optional

```

Retain artifacts with the build record

Add the [Copy Files](#) and [Publish Build Artifacts](#) tasks to store your APK with the build record or test and deploy it in subsequent pipelines. See [Artifacts](#).

```

- task: CopyFiles@2
  inputs:
    contents: '**/*.apk'
    targetFolder: '$(build.artifactStagingDirectory)'
- task: PublishBuildArtifacts@1

```

Deploy

App Center

Add the [App Center Distribute](#) task to distribute an app to a group of testers or beta users, or promote the app to Intune or Google Play. A free [App Center](#) account is required (no payment is necessary).

```

# App Center Distribute
# Distribute app builds to testers and users via App Center
- task: AppCenterDistribute@1
  inputs:
    serverEndpoint:
    appSlug:
    appFile:
    #symbolsOption: 'Apple' # Optional. Options: apple
    #symbolsPath: # Optional
    #symbolsPdbFiles: '**/*.pdb' # Optional
    #symbolsDsymFiles: # Optional
    #symbolsMappingTxtFile: # Optional
    #symbolsIncludeParentDirectory: # Optional
    #releaseNotesOption: 'input' # Options: input, file
    #releaseNotesInput: # Required when releaseNotesOption == Input
    #releaseNotesFile: # Required when releaseNotesOption == File
    #distributionGroupId: # Optional

```

Google Play

Install the [Google Play extension](#) and use the following tasks to automate interaction with Google Play. By default, these tasks authenticate to Google Play using a [service connection](#) that you configure.

Release

Add the [Google Play Release](#) task to release a new Android app version to the Google Play store.

```

- task: GooglePlayRelease@2
  inputs:
    apkFile: '**/*.apk'
    serviceEndpoint: 'yourGooglePlayServiceConnectionName'
    track: 'internal'

```

Promote

Add the [Google Play Promote](#) task to promote a previously-released Android app update from one track to another, such as `alpha` → `beta`.

```

- task: GooglePlayPromote@2
  inputs:
    packageName: 'com.yourCompany.appPackageName'
    serviceEndpoint: 'yourGooglePlayServiceConnectionName'
    sourceTrack: 'internal'
    destinationTrack: 'alpha'

```

Increase rollout

Add the [Google Play Increase Rollout](#) task to increase the rollout percentage of an app that was previously released to the `rollout` track.

```

- task: GooglePlayIncreaseRollout@1
  inputs:
    packageName: 'com.yourCompany.appPackageName'
    serviceEndpoint: 'yourGooglePlayServiceConnectionName'
    userFraction: '0.5' # 0.0 to 1.0 (0% to 100%)

```

Related extensions

- [Codified Security](#) (Codified Security)
- [Google Play](#) (Microsoft)
- [Mobile App Tasks for iOS and Android](#) (James Montemagno)

- [Mobile Testing Lab](#) (Perfecto Mobile)
- [React Native](#) (Microsoft)

C++

10/15/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

This guidance explains how to build C++ projects.

NOTE

This guidance applies to TFS version 2017.3 and newer.

Example

This example shows how to build a C++ project. To start, import (into Azure Repos or TFS) or fork (into GitHub) this repo:

```
https://github.com/adventureworks/cpp-sample
```

NOTE

This scenario works on TFS, but some of the following instructions might not exactly match the version of TFS that you are using. Also, you'll need to set up a self-hosted agent, possibly also installing software. If you are a new user, you might have a better learning experience by trying this procedure out first using a free Azure DevOps organization. Then change the selector in the upper-left corner of this page from Team Foundation Server to **Azure DevOps**.

- After you have the sample code in your own repository, create a pipeline using the instructions in [Use the designer](#) and select the **.NET Desktop** template. This automatically adds the tasks required to build the code in the sample repository.
- Save the pipeline and queue a build to see it in action.

Build multiple configurations

It is often required to build your app in multiple configurations. The following steps extend the example above to build the app on four configurations: [Debug, x86], [Debug, x64], [Release, x86], [Release, x64].

- Click the **Variables** tab and modify these variables:

- `BuildConfiguration = debug, release`
- `BuildPlatform = x86, x64`

- Select **Tasks** and click on the **agent job** to change the options for the job:

- Select **Multi-configuration**.

- Specify **Multipliers:** `BuildConfiguration, BuildPlatform`
3. Select **Parallel** if you have multiple build agents and want to build your configuration/platform pairings in parallel.

Copy output

To copy the results of the build to Azure Pipelines or TFS, perform these steps:

1. Click the **Copy Files** task. Specify the following arguments:

- **Contents:** `**\$(BuildConfiguration)**\?(*.exe|*.dll|*.pdb)`

Build Docker apps by using Azure Pipelines or Team Foundation Server

11/29/2018 • 14 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

This guidance explains how to use Azure Pipelines or Team Foundation Server (TFS) to build Docker images and push them to registries such as Docker Hub or Azure Container Registry with continuous integration and delivery (CI/CD) pipelines.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

NOTE

This guidance applies to TFS version 2017.3 and newer.

NOTE

This article helps you start using Azure Pipelines by using Docker commands. As an alternative, Azure Pipelines has a built-in [Docker task](#) that you can use to build and push the container images to a container registry. [Learn more about how the task helps with Docker best practices and standards.](#)

Example

This example shows how to build a Docker image and push it to a registry.

- [YAML](#)
- [Designer](#)

1. To start, complete the steps from the example section in one of the following languages:

- [.NET Core](#)
- [JavaScript](#)
- [Python](#)

The sample repos include a `Dockerfile` at the root of the repository. You must have a working build pipeline before you continue.

2. Define two variables in your build pipeline in the web UI:

- **dockerId:** Your Docker ID for Docker Hub or the admin user name for the Azure Container Registry.
- **dockerPassword:** Your password for Docker Hub or the admin password for Azure Container Registry.

If you use Azure Container Registry, make sure that you have [pre-created the registry in the Azure portal](#). You can get the admin user name and password from the **Access keys** section of the registry in the Azure portal.

3. If you have a Docker Hub account, and you want to push the image to your **Docker Hub registry**, use the web UI to change the YAML file in the build pipeline from `azure-pipelines.yml` to `azure-pipelines.docker.yml`. This file is present at the root of your sample repository.
4. If you set up an **Azure container registry** and you want to push the image to that registry, use the web UI to change the YAML file in the build pipeline from `azure-pipelines.yml` to `azure-pipelines.acr.yml`. This file is present at the root of your sample repository.
5. Queue a new build and watch it create and push a Docker image to the registry.

YAML builds are not yet available on TFS.

Now that you've run a Docker build pipeline, you're ready to learn some of the common changes that people make to customize their Docker build.

Build environment

You can use Azure Pipelines to build and push your Docker images without needing to set up any infrastructure of your own. You can build either Windows or Linux container images. The Microsoft-hosted agents in Azure Pipelines have Docker pre-installed on them. We frequently update the version of Docker on these agent machines. To know which version of Docker is installed, see [Microsoft-hosted agents](#).

- [YAML](#)
- [Designer](#)

Add the following snippet to your `azure-pipelines.yml` file to select the appropriate agent pool:

```
pool:
  vmImage: 'ubuntu-16.04' # other options: 'macOS-10.13', 'vs2017-win2016'
```

Microsoft-hosted Linux agents

When you use the Microsoft-hosted Linux agents, you get a fresh Linux virtual machine with each build. This virtual machine runs the [agent](#) and acts as a Docker host.

Microsoft-hosted VS2017 (Windows) agents

Use the Windows agents ([win2016-vs2017](#) image if you use YAML, **Hosted VS2017** if you use the designer) to build Windows container images. When you use this pool, you get a fresh Windows Server 2016 virtual machine with each build. The virtual machine runs the [agent](#) and acts as a Docker host. Some of the common images, such as `microsoft/dotnet-framework`, `microsoft/aspnet`, `microsoft/aspnetcore-build`, `microsoft/windowsservercore`, and `microsoft/nanoserver`, are pre-cached on this Docker host. Building new images from these images will therefore be faster.

NOTE

- By using Windows agents, you can build Docker images only with Windows Server 2016 as the container OS. You cannot build Docker images with Windows Server 1803 as the container OS because the host operating system on the virtual machines is Windows Server 2016.
- We don't yet have a pool of Microsoft-hosted agents running Windows Server 1803. Until this is available, you can build Windows Server 1803 images by using self-hosted agents.

Microsoft-hosted MacOS agents

You cannot use macOS agents to build container images because Docker is not installed on these agents.

Self-hosted agents

As an alternative to using Microsoft-hosted agents, you can set up [self-hosted agents](#) with Docker installed. This is useful if you want to cache additional images on the Docker host and further improve the performance of your builds.

Your builds run on a [self-hosted agent](#). Make sure that you have Docker installed on the agent.

Build an image

You can build a Docker image by running the `docker build` command in a script or by using the [Docker task](#).

- [YAML](#)
- [Designer](#)

To run the command in a script, add the following snippet to your `azure-pipelines.yml` file.

```
- script: docker build -t $(dockerId)/$(imageName) . # add options to this command to meet your needs
```

You can run any docker commands as part of the script block in your YAML file. If your Dockerfile depends on another image from a protected registry, you have to first run a `docker login` command in your script. If you want to avoid managing the username and password as a secret, you can use the [Docker task](#), which uses the service connection for `docker login`. After you have used the [Docker task](#) to log in, the session is maintained for the duration of the job. You can then use follow-up tasks to execute any scripts.

```
- script: docker login -u $(dockerId) -p $(pswd) <docker-registry-url>
```

Make sure that you define the Docker password as a [secret variable](#) in the build pipeline and not in the YAML file.

You don't need to specify `docker-registry-url` in the `login` command, if you are connecting to Docker Hub.

YAML builds are not yet available on TFS.

Integrate build and test tasks

Often you'll want to build and test your app before creating the Docker image. You can orchestrate this process either in your build pipeline or in your Dockerfile.

Build and test in your build pipeline

In this approach, you use the build pipeline to orchestrate building your code, running your tests, and creating an image. This approach is useful if you want to:

- Use tasks (either built-in tasks or those you get from the Azure DevOps Marketplace) to define the pipeline used to build and test your app.
- Run tasks that require authentication via service connections (for example: authenticated NuGet or npm feeds).
- Publish test results.

To create an image, you run a `docker build` command at the end of your build pipeline. Your Dockerfile contains the instructions to copy the results of your build into the container.

The instructions in the [earlier example](#) demonstrate this approach. The test results published in the example can be viewed under the [Tests tab](#) in the build.

Build and test in your Dockerfile

In this approach, you use your Dockerfile to build your code and run tests. The build pipeline has a single step to

run `docker build`. The rest of the steps are orchestrated by the Docker build process. It's common to use a [multi-stage Docker build](#) in this approach. The advantage of this approach is that your build process is entirely configured in your Dockerfile. This means your build process is portable from the development machine to any build system. One disadvantage is that you can't use Azure Pipelines and TFS features such as tasks, jobs, or test reporting.

For an example of using this approach, follow these steps:

1. The sample repos that you used in the [earlier example](#) also include a **Dockerfile.multistage** for this approach:

- [Dockerfile.multistage in .NET Core sample](#)
- [Dockerfile.multistage in JavaScript sample](#)

Replace the content in the `Dockerfile` at the root of your repository with the content from `Dockerfile.multistage`.

2. Then, define your build pipeline:

- [YAML](#)
- [Designer](#)

Replace the contents in the `azure-pipelines.yml` file at the root of your repo with the following content:

```
pool:  
  vmImage: 'ubuntu-16.04'  
  
steps:  
  - script: docker build -t $(dockerId)/$(imageName) . # include other options to meet your needs
```

YAML builds are not yet available on TFS.

Push an image

After you've built a Docker image, you can push it to a Docker registry or to Azure Container Registry. You can do this by using either the `docker push` command or by using the `Docker` task. The [Docker task](#) makes the process easier for you because it sets up an authenticated connection to your registry or Azure Container Registry.

- [YAML](#)
- [Designer](#)

To push the image to Docker Hub, add the following snippet to the `azure-pipelines.yml` file at the root of your repo:

```
- script: |  
  docker login -u $(dockerId) -p $(pswd)  
  docker push $(dockerId)/$(imageName)
```

To build and push the image to Azure Container Registry, use the following snippet:

```
- script: |  
  docker build -t $(dockerId).azurecr.io/$(imageName) .  
  docker login -u $(dockerId) -p $(pswd) $(dockerId).azurecr.io  
  docker push $(dockerId).azurecr.io/$(imageName)
```

YAML builds are not yet available on TFS.

Use Docker Compose

Docker Compose enables you to bring up multiple containers and run tests. For example, you can use a `docker-compose.yml` file to define two containers that need to work together to test your application: a web service that contains your application and a test driver. You can build new container images every time you push a change to your code. You can wait for the test driver to finish running tests before bringing down the two containers.

If you use Microsoft-hosted agents, you don't have to run any additional steps to install and use docker-compose.

To extend the [earlier example](#) to use docker-compose:

1. Your sample repo already includes a `docker-compose.yml` file in the `docs` folder.

2. Add a **Bash** step to your build pipeline:

- [YAML](#)
- [Designer](#)

Add the following snippet to your `azure-pipelines.yml` file.

```
- script: |
  docker-compose -f docs/docker-compose.yml --project-directory . -p docs up -d
  docker wait docs_sut_1
  docker-compose -f docs/docker-compose.yml --project-directory . down
```

YAML builds are not yet available on TFS.

NOTE

When you're using agents in the Hosted Linux Preview pool, the agent runs inside a container. The network of this container is not bridged to the network of the containers that you spin up through Docker Compose. As a result, you can't communicate from the agent to one of the containers in the composition, for example, to drive tests. The preferred workaround is to upgrade to the *Hosted Ubuntu 1604* pool, where the agents don't run inside a container.

If you can't upgrade, another way to solve this problem is to explicitly create another test driver as a container within the composition, as we did in the earlier example. Another solution is to use `docker-compose exec` and target a specific container in the composition from your script.

Build ARM containers

When you use Microsoft-hosted Linux agents, you create Linux container images for the x64 architecture. To create images for other architectures (for example, x86, ARM, and so on), you can use a machine emulator such as [QEMU](#). The following steps illustrate how to create an ARM container image:

1. Author your Dockerfile so that an Intel binary of QEMU exists in the base Docker image. For example, the Raspbian Docker image from [Resin](#) already has this.

```
FROM resin/rpi-raspbian
```

2. Run the following script in your build pipeline.

```
# register QEMU binary - this can be done by running the following Docker image
docker run --rm --privileged multiarch/qemu-user-static:register --reset
# build your image
docker build -t $(dockerId)/$(imageName) .
```

Troubleshooting

If you can build your image on your development machine, but you're having trouble building it on Azure Pipelines or TFS, the following solutions might help:

- Check that you are using the correct type of agents - Microsoft-hosted Linux or Microsoft-hosted Windows - to mimic the type of container images you build on your development machine.
- If you use Microsoft-hosted agents to run your builds, the Docker images are not cached from build to build because you get a new machine for every build. This will make your builds on Microsoft-hosted agents run longer than those on your development machine.
- If you use agents from the Hosted Linux Preview pool, the agent itself runs in a container. This has some implications when you use docker-compose to spin up additional containers. As an example, there is no network connectivity from the agent container to the composition containers. Use `docker-compose exec` as a way of executing commands from the agent container in one of the composition containers. Also, you should upgrade those builds to use the *Hosted Ubuntu 1604* pool, where the agents do not run in containers.

If you can build your image on your development machine, but you're having trouble building it on Azure Pipelines or TFS, check the version of Docker on the agent. Ensure that it matches what you have on your development machine. You can include a command-line script `docker --version` in your build pipeline to print the version of Docker.

Build Go projects with Azure Pipelines

10/13/2018 • 3 minutes to read • [Edit Online](#)

Azure Pipelines

This guidance explains how to use Azure Pipelines to automatically build and test Go projects with CI/CD pipelines.

Example

To get started using a sample Go project, fork this repository in GitHub, or import it into Azure Repos or TFS:

```
https://github.com/MicrosoftDocs/pipelines-go
```

The sample code includes an `azure-pipelines.yml` file at the root of the repository. You can use this file to build the app.

Follow instructions in [Create your first pipeline](#) to create a build pipeline for the sample project.

The rest of this topic describes ways to customize your Go build pipeline.

Build environment

You can use Azure Pipelines to build your Go projects without needing to set up any infrastructure of your own. Modern versions of Go are preinstalled on [Microsoft-hosted agents](#) in Azure Pipelines. You can use Linux, macOS, or Windows agents to run your builds.

For the exact versions of Go that are preinstalled, refer to [Microsoft-hosted agents](#).

Create a file named **azure-pipelines.yml** in the root of your repository. Then, add the following snippet to your `azure-pipelines.yml` file to select the appropriate agent pool:

```
# https://docs.microsoft.com/azure/devops/pipelines/languages/go
pool:
  vmImage: 'ubuntu-16.04' # Other options: 'macOS-10.13', 'vs2017-win2016'
```

Set up a Go workspace

As the Go documentation [describes](#), a Go workspace consists of a root directory to which the `$GOPATH` environment variable points. Within that directory are standard subdirectories:

- `bin` to contain executable commands
- `pkg` to contain compiled packages (`.a` files)
- `src` to contain Go source files (`.go`, `.c`, `.g`, `.s`)

When an Azure Pipelines build fetches code from a remote repository, it places the code in the default working directory of the build. This doesn't match the expected structure of a Go workspace. To address this, add the following snippet to your `azure-pipelines.yml` file. Note: this script runs in bash on Linux and macOS agents, but must be modified for Windows.

```

variables:
  GOBIN:  '$(GOPATH)/bin' # Go binaries path
  GOROOT: '/usr/local/go1.11' # Go installation path
  GOPATH:  '$(system.defaultWorkingDirectory)/gopath' # Go workspace path
  modulePath:  '$(GOPATH)/src/github.com/${build.repository.name}' # Path to the module's code

steps:
- script: |
    mkdir -p '$(GOBIN)'
    mkdir -p '$(GOPATH)/pkg'
    mkdir -p '$(modulePath)'
    shopt -s extglob
    mv !(gopath) '$(modulePath)'
    echo '##vso[task.prependpath]$(GOBIN)'
    echo '##vso[task.prependpath]$(GOROOT)/bin'
  displayName: 'Set up the Go workspace'

```

If your code is not in GitHub, change the `modulePath` variable's use of `github.com` to an appropriate value for your module.

This snippet does the following:

1. Sets `$GOROOT` to the version of Go that should be used.
2. Sets other well-known Go environment variables to their proper values.
3. Creates a Go workspace in a subdirectory named `gopath` with child directories `bin`, `pkg`, and `src`.
4. Moves code that was fetched from the remote repository into the workspace's `src` directory
5. Adds the version of Go and the workspace's `bin` directory to the path.

Install dependencies

go get

Use `go get` to download the source code for a Go project or to install a tool into the Go workspace. Add the following snippet to your `azure-pipelines.yml` file:

```

- script: go get -v -t -d ./...
  workingDirectory: '$(modulePath)'
  displayName: 'go get dependencies'

```

dep ensure

Use `dep ensure` if your project uses dep to download dependencies imported in your code. Running `dep ensure` clones imported repositories into your project's vendor directory. Its `Gopkg.lock` and `Gopkg.toml` files guarantee that everyone working on the project uses the same version of dependencies as your build. Add the following snippet to your `azure-pipelines.yml` file. Note: this script runs in bash on Linux and macOS agents, but must be modified for Windows.

```

- script: |
    if [ -f Gopkg.toml ]; then
      curl https://raw.githubusercontent.com/golang/dep/master/install.sh | sh
      dep ensure
    fi
  workingDirectory: '$(modulePath)'
  displayName: 'Download dep and run `dep ensure`'

```

Test

Use `go test` to test your go module and its subdirectories (`./...`). Add the following snippet to your `azure-pipelines.yml` file:

```
- script: go test -v ./...
  workingDirectory: '$(modulePath)'
  displayName: 'Run tests'
```

Build

Use `go build` to bulid your Go project. Add the following snippet to your `azure-pipelines.yml` file:

```
- script: go build -v .
  workingDirectory: '$(modulePath)'
  displayName: 'Build'
```

Build a container image

You can also build and publish a Docker container image for your app. For more information, see [Docker](#).

Related extensions

[Go extension for Visual Studio Code \(Microsoft\)](#)

Use Helm to package Docker-based applications

11/19/2018 • 4 minutes to read • [Edit Online](#)

Azure Pipelines

This guidance explains how to use [Helm](#) to package a Docker based application in to a Helm chart.

Helm is a tool that streamlines deploying and managing Kubernetes applications using a packaging format called [charts](#). You can define, version, share, install, and upgrade even the most complex Kubernetes application using Helm.

1. Helm helps you combine multiple Kubernetes manifests (yaml) like service, deployments, configmaps etc. in to a single unit called Helm Charts. You don't need to either invent or use a tokenization or templating tool.
2. Helm Charts also help you manage application dependencies and deploy as well as rollback as a unit. They are also easy to create, version, publish and share with other partner teams

A Helm chart consists of metadata, definitions, config and documentation. This can be either stored in the same code repository as your application code or in a separate repository. Helm can package these files into a chart archive (*.tgz file), which gets deployed to a Kubernetes cluster.

The steps required to build a container image and pushing it to a container registry remains the same. Once that has been done, we start creating a Helm Chart archive package.

Azure Pipelines has built-in support for Helm charts:

1. [Helm Tool installer task](#) can be used to get the right version of Helm on the agents.
2. [Helm package and deploy task](#) can be used to package the application and deploy it to a Kubernetes cluster.
3. You can use the task to install/update Tiller to a Kubernetes namespace, securely connect to the Tiller over TLS for deploying charts, use the task to run any Helm command like lint
4. The Helm task also supports connecting to an Azure Kubernetes Service by using Azure Service Connection. You can connect to any Kubernetes cluster by using kubeconfig or Service Account as well.
5. The Helm deployments can be supplemented by using Kubectl task. For example create/update imagepullsecret

Define your CI build process using Helm

We'll show you how to set up a continuous integration workflow for your containerized application using Azure Pipelines and Helm.

Prerequisites

You'll need an Azure subscription. You can get one free through [Visual Studio Dev Essentials](#).

You'll need a Docker container image published to a Kubernetes Cluster (for example: Azure Kubernetes Service). To set up a continuous integration (CI) build process, see:

- [Build and publish a Docker image](#).

Package and publish a Helm chart

1. In the **Build & Release** hub, open the [build pipeline](#) created to build and publish a Docker image.
2. Select Tasks tab and click on + icon to add **Helm tool installer** task to ensure that the agent which runs the subsequent tasks has Helm and Kubernetes installed on it.

3. Click on + icon again to add new **Package and deploy Helm charts** task. Configure the properties as follows:

- **Connection Type:** Select 'Azure Resource Manager' to connect to an AKS cluster by using Azure Service Connection. Select 'Container registry' to connect to any Kubernetes cluster by using kubeconfig or Service Account.
- **Azure Subscription:** Select a connection from the list under **Available Azure Service Connections** or create a more restricted permissions connection to your Azure subscription. If you are using Azure DevOps and if you see an **Authorize** button next to the input, click on it to authorize Azure DevOps to connect to your Azure subscription. If you do not see the desired Azure subscription in the list of subscriptions, see [Azure Resource Manager service connection](#) to manually set up the connection.
- **Resource Group:** Enter or select the resource group of your **AKS cluster**.
- **Kubernetes cluster:** Enter or select the **AKS cluster** you have created.
- **Command:** Select **init** as Helm command.

-**Arguments:** Provide additional arguments for the Helm command. In this case set this field as "--client-only" to ensure the installation of only the Helm client.

4. Again click on + icon to add another **Package and deploy Helm charts** task Configure the properties as follows:

- **Command:** Select **package** as Helm command. You can run any Helm command by using the task and passing command options as arguments. When you select **package** as the helm command, the task recognizes it and shows only the relevant fields.
 - **Chart Path:** Enter the path to your Helm chart. Chart path can be a path to a packaged chart or a path to an unpacked chart directory. For example if './redis' is specified the task will run 'helm package ./redis'.
- Version:** Specify a semver version to be set on the chart.
- Destination:** Choose the destination to publish the Helm chart. If it is the working directory, just set "\$(Build.ArtifactStagingDirectory)"
- Update dependency:** Tick this checkbox to run helm dependency update before installing the chart. Task runs 'helm package --dependency-update' and updates dependencies from 'requirements.yaml' to dir 'charts/' before packaging.

5. Again click on + icon to add a **Publish Artifacts** task

6. Save the build pipeline.

Build Java projects with Azure Pipelines or Team Foundation Server

11/19/2018 • 4 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#)

NOTE

This guidance uses YAML-based pipelines available in Azure Pipelines. For TFS, use tasks that correspond to those used in the YAML below.

This guidance explains how to use Azure Pipelines or Team Foundation Server (TFS) to automatically build Java projects with CI/CD pipelines. See [Android](#) for Android-specific projects.

Example

To get started using a sample Java project, fork this repository in GitHub, or import it into Azure Repos or TFS:

```
https://github.com/MicrosoftDocs/pipelines-java
```

The sample code includes an `azure-pipelines.yml` file at the root of the repository. You can use this file to build the project.

Follow instructions in [Create your first pipeline](#) to create a build pipeline for the sample project.

The rest of this topic describes ways to customize your Java build pipeline.

Choose an agent

You can use Azure Pipelines to build your Java projects on [Microsoft-hosted agents](#) that include modern JDKs and other tools for Java. Or, you can use [self-hosted agents](#) with specific tools that you need.

Create a file named **azure-pipelines.yml** in the root of your repository. Then, add the following snippet to your `azure-pipelines.yml` file to select the appropriate agent pool:

```
# https://docs.microsoft.com/azure/devops/pipelines/languages/java
pool:
  vmImage: 'ubuntu-16.04' # Other options: 'macOS-10.13', 'vs2017-win2016'
```

Build your code with Maven

To build with Maven, add the following snippet to your `azure-pipelines.yml` file. Change values, such as the path to your `pom.xml` file, to match your project configuration. See the [Maven](#) task for more about these options.

```
steps:
- task: Maven@3
  inputs:
    mavenPomFile: 'pom.xml'
    mavenOptions: '-Xmx3072m'
    javaHomeOption: 'JDKVersion'
    jdkVersionOption: '1.11'
    jdkArchitectureOption: 'x64'
    publishJUnitResults: false
    testResultsFiles: '**/TEST-*.xml'
    goals: 'package'
```

Customize Maven

Customize the build path

Adjust the `mavenPomFile` value if your `pom.xml` file isn't in the root of the repository. The file path value should be relative to the root of the repository, such as `IdentityService/pom.xml` or `$(system.defaultWorkingDirectory)/IdentityService/pom.xml`.

Customize Maven goals

Set the `goals` value to a space-separated list of goals for Maven to execute, such as `clean package`.

For details about common Java phases and goals, see [Apache's Maven documentation](#).

Build your code with Gradle

To build with Gradle, add the following snippet to your `azure-pipelines.yml` file. See the [Gradle](#) task for more about these options.

```
steps:
- task: Gradle@2
  inputs:
    workingDirectory: ''
    gradleWrapperFile: 'gradlew'
    gradleOptions: '-Xmx3072m'
    javaHomeOption: 'JDKVersion'
    jdkVersionOption: '1.11'
    jdkArchitectureOption: 'x64'
    publishJUnitResults: false
    testResultsFiles: '**/TEST-*.xml'
    tasks: 'build'
```

Customize Gradle

Choose the version of Gradle

The version of Gradle installed on the agent machine will be used unless your repository's `gradle/wrapper/gradle-wrapper.properties` file has a `distributionUrl` property that specifies a different Gradle version to download and use during the build.

Adjust the build path

Adjust the `workingDirectory` value if your `gradlew` file isn't in the root of the repository. The directory value should be relative to the root of the repository, such as `IdentityService` or `$(system.defaultWorkingDirectory)/IdentityService`.

Adjust the `gradleWrapperFile` value if your `gradlew` file isn't in the root of the repository. The file path value should be relative to the root of the repository, such as `IdentityService/gradlew` or `$(system.defaultWorkingDirectory)/IdentityService/gradlew`.

Adjust Gradle tasks

Adjust the `tasks` value for the tasks that Gradle should execute, such as `build` or `check`.

For details about common Java Plugin tasks for Gradle, see [Gradle's documentation](#).

Build your code with Ant

To build with Ant, add the following snippet to your `azure-pipelines.yml` file. Change values, such as the path to your `build.xml` file, to match your project configuration. See the [Ant](#) task for more about these options.

```
steps:
- task: Ant@1
  inputs:
    workingDirectory: ''
    buildFile: 'build.xml'
    javaHomeOption: 'JDKVersion'
    jdkVersionOption: '1.11'
    jdkArchitectureOption: 'x64'
    publishJUnitResults: false
    testResultsFiles: '**/TEST-*.xml'
```

Build your code using a command line or script

To build with a command line or script, add one of the following snippets to your `azure-pipelines.yml` file.

Inline script

The `script:` step runs an inline script using Bash on Linux and macOS and Command Prompt on Windows. For details, see the [Bash](#) or [Command line](#) task.

```
steps:
- script: |
  echo Starting the build
  mvn package
  displayName: 'Build with Maven'
```

Script file

This snippet runs a script file that is in your repository. For details, see the [Shell Script](#), [Batch script](#), or [PowerShell](#) task.

```
steps:
- task: ShellScript@2
  inputs:
    scriptPath: 'build.sh'
```

Build JavaScript and Node.js apps with Azure Pipelines or Team Foundation Server

12/3/2018 • 18 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

This guidance explains how to use Azure Pipelines or Team Foundation Server (TFS) to automatically build JavaScript and Node.js apps with CI/CD pipelines.

NOTE

This guidance applies to Team Foundation Server (TFS) version 2017.3 and newer.

Example

For a working example of how to build a Node.js app, import (into Azure Repos or TFS) or fork (into GitHub) this repo:

```
https://github.com/MicrosoftDocs/pipelines-javascript
```

The sample code in this repo is a Node server implemented with the Express.js framework that echoes "Hello world." Tests for the app are written through the Mocha framework.

- [YAML](#)
- [Designer](#)

The sample code includes an `azure-pipelines.yml` file at the root of the repository. You can use this file to build the app.

Follow all the instructions in [Create your first pipeline](#) to create a build pipeline for the sample app.

YAML builds are not yet available on TFS.

Read through the rest of this topic to learn some of the common ways to customize your JavaScript build process.

Build environment

You can use Azure Pipelines to build your JavaScript apps without needing to set up any infrastructure of your own. Tools that you commonly use to build, test, and run JavaScript apps - like npm, Node, Yarn, and Gulp - are preinstalled on [Microsoft-hosted agents](#) in Azure Pipelines. You can use either Windows or Linux agents to run your builds.

For the exact version of Node.js and npm that is preinstalled, refer to [Microsoft-hosted agents](#). To install a

specific version of these tools on Microsoft-hosted agents, add the **Node Tool Installer** task to the beginning of your process.

Use a specific version of Node.js

- [YAML](#)
- [Designer](#)

If you need a version of Node.js and npm that is not already installed on the Microsoft-hosted agent, add the following snippet to your `azure-pipelines.yml` file.

NOTE

The hosted agents are regularly updated, and setting up this task will result in spending significant time updating to a newer minor version every time the pipeline is run. Use this task only when you need a specific Node version in your pipeline.

```
- task: NodeTool@0
  inputs:
    versionSpec: '8.x' # replace this value with the version that you need for your project
```

YAML builds are not yet available on TFS.

To update just the npm tool, run the `npm i -g npm@version-number` command in your build process.

Use multiple node versions

You can build and test your app on multiple versions of Node.

- [YAML](#)
- [Designer](#)

```
pool:
  vmImage: 'ubuntu-16.04'
strategy:
  matrix:
    node_8_x:
      node_version: 8.x
    node_9_x:
      node_version: 9.x

steps:
- task: NodeTool@0
  inputs:
    versionSpec: $(node_version)

- script: npm install
```

YAML builds are not yet available on TFS.

Install tools on your build agent

- [YAML](#)
- [Designer](#)

If you have defined tools needed for your build as development dependencies in your project's `package.json` or `package-lock.json` file, install these tools along with the rest of your project dependencies through npm. This will install the exact version of the tools defined in the project, isolated from other versions that exist on the

build agent.

```
- script: npm install --only=dev
```

Run tools installed this way by using npm's `npx` package runner, which will first look for tools installed this way in its path resolution. The following example calls the `mocha` test runner but will look for the version installed as a dev dependency before using a globally installed (through `npm install -g`) version.

```
- script: npx mocha
```

To install tools that your project needs but that are not set as dev dependencies in `package.json`, call `npm install -g` from a script stage in your pipeline.

The following example installs the latest version of the [Angular CLI](#) by using `npm`. The rest of the pipeline can then use the `ng` tool from other `script` stages.

```
- script: npm install -g @angular/cli
```

These tasks will run every time your pipeline runs, so be mindful of the impact that installing tools has on build times. Consider configuring [self-hosted agents](#) with the version of the tools you need if overhead becomes a serious impact to your build performance.

YAML builds are not yet available on TFS.

Dependency management

In your build, use [Yarn](#) or Azure Artifacts/TFS to download packages from the public npm registry, which is a type of private npm registry that you specify in the `.npmrc` file.

npm

You can use NPM in a few ways to download packages for your build:

- Directly run `npm install` in your build pipeline. This is the simplest way to download packages from a registry that does not need any authentication. If your build doesn't need development dependencies on the agent to run, you can speed up build times with the `--only=prod` option to `npm install`.
- Use an [npm task](#). This is useful when you're using an authenticated registry.
- Use an [npm Authenticate task](#). This is useful when you run `npm install` from inside your task runners - Gulp, Grunt, or Maven.

If you want to specify an npm registry, put the URLs in an `.npmrc` file in your repository. If your feed is authenticated, manage its credentials by creating an npm service connection on the **Services** tab under **Project Settings**.

- [YAML](#)
- [Designer](#)

To install npm packages by using a script in your build pipeline, add the following snippet to `azure-pipelines.yml`.

```
- script: npm install
```

To use a private registry specified in your `.npmrc` file, add the following snippet to `azure-pipelines.yml`.

```
- task: Npm@1
  inputs:
    customEndpoint: <Name of npm service connection>
```

To pass registry credentials to npm commands via task runners such as Gulp, add the following task to `azure-pipelines.yml` before you call the task runner.

```
- task: npmAuthenticate@0
  inputs:
    customEndpoint: <Name of npm service connection>
```

YAML builds are not yet available on TFS.

If your builds occasionally fail because of connection issues when you're restoring packages from the npm registry, you can use Azure Artifacts in conjunction with [upstream sources](#), and cache the packages. The credentials of the build pipeline are automatically used when you're connecting to Azure Artifacts. These credentials are typically derived from the **Project Collection Build Service** account.

If you're using Microsoft-hosted agents, you get a new machine every time you run a build - which means restoring the dependencies every time. This can take a significant amount of time. To mitigate this, you can use Azure Artifacts or a self-hosted agent. You'll then get the benefit of using the package cache.

Yarn

- [YAML](#)
- [Designer](#)

Use a simple script stage to invoke [Yarn](#) to restore dependencies. Yarn is available preinstalled on some [Microsoft-hosted agents](#). You can install and configure it on self-hosted agents like any other tool.

```
- script: yarn install
```

YAML builds are not yet available on TFS.

Run JavaScript compilers

- [YAML](#)
- [Designer](#)

Use compilers such as [Babel](#) and the [TypeScript](#) `tsc` compiler to convert your source code into versions that are usable by the Node.js runtime or in web browsers.

If you have a [script object](#) set up in your project's `package.json` file that runs your compiler, invoke it in your pipeline by using a script task.

```
- script: npm run compile
```

You can call compilers directly from the pipeline by using the script task. These commands will run from the root of the cloned source-code repository.

```
- script: tsc --target ES6 --strict true --project tsconfigs/production.json
```

YAML builds are not yet available on TFS.

Run unit tests

- [YAML](#)
- [Designer](#)

Configure your pipelines to run your JavaScript tests so that they produce results formatted in the JUnit XML format. You can then publish the results to VSTS easily by using the built-in [Publish Test Results](#) task.

If your test framework doesn't support JUnit output out of the box, you'll need to add support through a partner reporting module, such as [mocha-junit-reporter](#). You can either update your test script to use the JUnit reporter, or if the reporter supports command-line options, pass those into the task definition.

The following table lists the most commonly used test runners and the reporters that can be used to produce XML results:

TEST RUNNER	REPORTERS TO PRODUCE XML REPORTS
mocha	mocha-junit-reporter mocha-multi-reporters
jasmine	jasmine-reporters
jest	jest-junit jest-junit-reporter
karma	karma-junit-reporter
Ava	tap-xunit

This example uses the [mocha-junit-reporter](#) and invokes `mocha test` directly by using a script task. This produces the JUnit XML output at the default location of `./test-results.xml`.

```
- script: mocha test --reporter mocha-junit-reporter
```

If you have defined a `test` script in your project's `package.json` file, you can invoke it by using `npm test` just as you would from the command line.

```
- script: npm test
```

Publish test results

To publish the results, use the [Publish Test Results](#) task.

```
- task: PublishTestResults@2
  inputs:
    testRunner: JUnit
    testResultsFiles: ./test-results.xml
```

Publish code coverage results

If your test scripts run a code coverage tool such as [Istanbul](#), add the [Publish Code Coverage Results](#) task to publish code coverage results along with your test results. When you do this, you can find coverage metrics in the build summary and download HTML reports for further analysis. The task expects Cobertura reporting output, so ensure that your code coverage tool runs with the necessary options to generate the right output. (For example, Istanbul needs `--report cobertura`.)

```
- task: PublishCodeCoverageResults@1
  inputs:
    codeCoverageTool: Cobertura
    summaryFileLocation: '$(System.DefaultWorkingDirectory)/**/*coverage.xml'
    reportDirectory: '$(System.DefaultWorkingDirectory)/**/coverage'
```

YAML builds are not yet available on TFS.

End-to-end browser testing

Run tests in headless browsers as part of your pipeline with tools like [Protractor](#) or [Karma](#). Then publish the results for the build to VSTS with these steps:

1. Install a headless browser testing driver such as headless Chrome or Firefox, or a browser mocking tool such as PhantomJS, on the build agent.
2. Configure your test framework to use the headless browser/driver option of your choice according to the tool's documentation.
3. Configure your test framework (usually with a reporter plug-in or configuration) to output JUnit-formatted test results.
4. Set up a script task to run any CLI commands needed to start the headless browser instances.
5. Run the end-to-end tests in the pipeline stages along with your unit tests.
6. Publish the results by using the same [Publish Test Results](#) task alongside your unit tests.

Package web apps

- [YAML](#)
- [Designer](#)

Package applications to bundle all your application modules with intermediate outputs and dependencies into static assets ready for deployment. Add a pipeline stage after your compilation and tests to run a tool like [Webpack](#) or [ng build](#) by using the Angular CLI.

The first example calls `webpack`. To have this work, make sure that `webpack` is configured as a development dependency in your `package.json` project file. This will run `webpack` with the default configuration unless you have a `webpack.config.js` file in the root folder of your project.

```
- script: webpack
```

The next example uses the `npm` task to call `npm run build` to call the `build` script object defined in the project `package.json`. Using script objects in your project moves the logic for the build into the source code and out of the of the pipeline.

```
- script: npm run build
```

YAML builds are not yet available on TFS.

JavaScript frameworks

AngularJS

For AngularJS apps, you can include Angular-specific commands such as `ng test`, `ng build`, and `ng e2e`. To use AngularJS CLI commands in your build pipeline, you need to install the [angular/cli npm package](#) on the build agent.

- [YAML](#)
- [Designer](#)

```
- script: |
  npm install -g @angular/cli
  npm install
  ng build --prod
```

YAML builds are not yet available on TFS.

For tests in your build pipeline that require a browser to run (such as the **ng test** command in the starter app, which runs Karma), you need to use a headless browser instead of a standard browser. In the AngularJS starter app, an easy way to do this is to:

1. Change the `browsers` entry in your `karma.conf.js` project file from `browsers: ['Chrome']` to `browsers: ['ChromeHeadless']`.
2. Change the `singleRun` entry in your `karma.conf.js` project file from a value of `false` to `true`. This helps make sure that the Karma process stops after it runs.

Webpack

You can use a webpack configuration file to specify a compiler (such as Babel or TypeScript) to transpile JSX or TypeScript to plain JavaScript, and to bundle your app.

- [YAML](#)
- [Designer](#)

```
- script: |
  npm install webpack webpack-cli --save-dev
  npx webpack --config webpack.config.js
```

YAML builds are not yet available on TFS.

React

All the dependencies for your React app are captured in your `package.json` file. The steps outlined in the earlier example section should work for building and testing a React app.

Vue

All the dependencies for your Vue app are captured in your `package.json` file. The steps outlined in the earlier example section should work for building and testing a Vue app.

Build task runners

It's common to use [Gulp](#) or [Grunt](#) as a task runner to build and test a JavaScript app.

Gulp

- [YAML](#)
- [Designer](#)

Gulp is preinstalled on Microsoft-hosted agents. To run the `gulp` command in the YAML file:

```
- script: gulp # include any additional options that are needed
```

If the steps in your `gulpfile.js` file require authentication with an npm registry:

```
- task: npmAuthenticate@0
  inputs:
    customEndpoint: <Name of npm service connection>

- script: gulp          # include any additional options that are needed
```

Add the [Publish Test Results](#) task to publish JUnit or xUnit test results to the server.

```
- task: PublishTestResults@2
  inputs:
    testResultsFiles: '**/TEST-RESULTS.xml'
    testRunTitle: 'Test results for JavaScript using gulp'
```

Add the [Publish Code Coverage Results](#) task to publish code coverage results to the server. You can find coverage metrics in the build summary, and you can download HTML reports for further analysis.

```
- task: PublishCodeCoverageResults@1
  inputs:
    codeCoverageTool: Cobertura
    summaryFileLocation: '$(System.DefaultWorkingDirectory)/**/*coverage.xml'
    reportDirectory: '$(System.DefaultWorkingDirectory)/**/coverage'
```

YAML builds are not yet available on TFS.

Grunt

- [YAML](#)
- [Designer](#)

Grunt is preinstalled on Microsoft-hosted agents. To run the grunt command in the YAML file:

```
- script: grunt          # include any additional options that are needed
```

If the steps in your `Gruntfile.js` file require authentication with a npm registry:

```
- task: npmAuthenticate@0
  inputs:
    customEndpoint: <Name of npm service connection>

- script: grunt          # include any additional options that are needed
```

YAML builds are not yet available on TFS.

Package and deliver your code

After you have built and tested your app, you can upload the build output to Azure Pipelines or TFS, create and publish an npm or Maven package, or package the build output into a .zip file to be deployed to a web application.

- [YAML](#)
- [Designer](#)

Publish files to Azure Pipelines

To simply upload the entire working directory of files, add the following to your `azure-pipelines.yml` file.

```
- task: PublishBuildArtifacts@1
  inputs:
    PathToPublish: '$(System.DefaultWorkingDirectory)'
```

To upload a subset of files, first copy the necessary files from the working directory to a staging directory, and then use the **PublishBuildArtifacts** task.

```
- task: CopyFiles@2
  inputs:
    SourceFolder: '$(System.DefaultWorkingDirectory)'
    Contents: |
      **\*.js
      package.json
    TargetFolder: '$(Build.ArtifactStagingDirectory)'

- task: PublishBuildArtifacts@1
```

Publish a module to a npm registry

If your project's output is an `npm` module for use by other projects and not a web application, use the `npm` task to publish the module to a local registry or to the public npm registry. You must provide a unique name/version combination each time you publish, so keep this in mind when configuring publishing steps as part of a release or development pipeline.

The first example assumes that you manage version information (such as through an `npm version`) through changes to your `package.json` file in version control. This example uses the script task to publish to the public registry.

```
- script: npm publish
```

The next example publishes to a custom registry defined in your repo's `.npmrc` file. You'll need to set up an [npm service connection](#) to inject authentication credentials into the connection as the build runs.

```
- task: Npm@1
  inputs:
    command: publish
    publishRegistry: useExternalRegistry
    publishEndpoint: https://my.npmregistry.com
```

The final example publishes the module to an Azure DevOps Services package management feed.

```
- task: Npm@1
  inputs:
    command: publish
    publishRegistry: useFeed
    publishFeed: https://my.npmregistry.com
```

For more information about versioning and publishing npm packages, see [Publish npm packages](#).

Deploy a web app

To create a .zip file archive that is ready for publishing to a web app, add the following snippet:

```
- task: ArchiveFiles@2
  inputs:
    rootFolderOrFile: '$(System.DefaultWorkingDirectory)'
    includeRootFolder: false
```

To publish this archive to a web app, see [Azure web app deployment](#).

YAML builds are not yet available on TFS.

Build a container

You can build a Docker image and push it to a container registry after you build your project.

If your application doesn't require orchestration with other containers, use the [Docker](#) task to build a container with your packaged application code and push it to your Docker registry.

This example builds a Docker image. The `Dockerfile` for the image is located in the root of the source code repo, but you can configure it by using the `dockerFile` input. The image is not pushed to a Docker registry after it's built.

```
- script: docker build -f Dockerfile -t contoso/myjavascriptapp:$(Build.BuildId)
```

After the Docker image is built, push it to a container registry by using the [Docker](#) task but with `command` set to `push An Image`. This example pushes to any container registry, including Docker Hub.

```
- task: Docker@1
  inputs:
    command: push An Image
    containerregistrytype: container Registry
    dockerRegistryEndpoint: registry.contoso.org
    imageName: contoso/myjavascriptcontainer:v1.0.0
```

For more information, see the [Docker pipeline guide](#).

Troubleshooting

If you can build your project on your development machine but are having trouble building it on Azure Pipelines or TFS, explore the following potential causes and corrective actions:

- Check that the versions of **Node.js** and the task runner on your development machine match those on the agent. You can include command-line scripts such as `node --version` in your build pipeline to check what is installed on the agent. Either use the **Node Tool Installer** (as explained in this guidance) to deploy the same version on the agent, or run `npm install` commands to update the tools to desired versions.
- If your builds fail intermittently while you're restoring packages, either the npm registry is having issues or there are networking problems between the Azure datacenter and the registry. These factors are not under our control, and you might need to explore whether using Azure Artifacts with an npm registry as an upstream source improves the reliability of your builds.

Q&A

Where can I learn more about Azure Artifacts and the TFS Package Management service?

[Package Management in Azure Artifacts and TFS](#)

Where can I learn more about tasks?

[Build, release, and test tasks](#)

Build PHP projects with Azure Pipelines

11/19/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

This guidance explains how to use Azure Pipelines to automatically build and test PHP projects with CI/CD pipelines.

Example

For a working example of how to build a PHP project, import (into Azure Repos or TFS) or fork (into GitHub) this repo:

```
https://github.com/MicrosoftDocs/pipelines-php
```

The sample code includes an `azure-pipelines.yml` file at the root of the repository. You can use this file to build the project.

Follow all the instructions in [Create your first pipeline](#) to create a build pipeline for the sample project.

Build environment

You can use Azure Pipelines to build your PHP projects without needing to set up any infrastructure of your own. PHP is preinstalled on [Microsoft-hosted agents](#) in Azure Pipelines, along with many common libraries per PHP version. You can use Linux, macOS, or Windows agents to run your builds.

For the exact versions of PHP that are preinstalled, refer to [Microsoft-hosted agents](#).

Use a specific PHP version

On the Microsoft-hosted Ubuntu agent, multiple versions of PHP are installed. A symlink at `/usr/bin/php` points to the currently set PHP version, so that when you run `php`, the set version executes. To use a PHP version other than the default, the symlink can be pointed to that version using the `update-alternatives` tool. Set the PHP version that you prefer by adding the following snippet to your `azure-pipelines.yml` file and changing the value of the **phpVersion** variable accordingly.

```
# https://docs.microsoft.com/azure/devops/pipelines/languages/php
pool:
  vmImage: 'ubuntu-16.04'

variables:
  phpVersion: 7.2

steps:
- script: |
    sudo update-alternatives --set php /usr/bin/php$(phpVersion)
    sudo update-alternatives --set phar /usr/bin/phar$(phpVersion)
    sudo update-alternatives --set phpdbg /usr/bin/phpdbg$(phpVersion)
    sudo update-alternatives --set php-cgi /usr/bin/php-cgi$(phpVersion)
    sudo update-alternatives --set phar.phar /usr/bin/phar.phar$(phpVersion)
    php -version
  displayName: 'Use PHP version $(phpVersion)'
```

Install dependencies

To use Composer to install dependencies, add the following snippet to your `azure-pipelines.yml` file.

```
- script: composer install --no-interaction --prefer-dist
displayName: 'composer install'
```

Test with phpunit

To run tests with phpunit, add the following snippet to your `azure-pipelines.yml` file.

```
- script: phpunit
displayName: 'Run tests with phpunit'
```

Retain the PHP app with the build record

To save the artifacts of this build with the build record, add the following snippet to your `azure-pipelines.yml` file.

Optionally, customize the value of **rootFolderOrFile** to alter what is included in the archive.

```
- task: ArchiveFiles@2
inputs:
rootFolderOrFile: '$(system.defaultWorkingDirectory)'
includeRootFolder: false
- task: PublishBuildArtifacts@1
```

Build a container image

You can also build and publish a Docker container image for your PHP app. For more information, see [Docker](#).

Build Python apps with Azure Pipelines

11/26/2018 • 5 minutes to read • [Edit Online](#)

Azure Pipelines

This guidance explains how to use Azure Pipelines to automatically build, test, and deploy Python apps or scripts with CI/CD pipelines.

Example

For a working example of how to build a Python app with Django, import (into Azure Repos or TFS) or fork (into GitHub) this repo:

```
https://github.com/MicrosoftDocs/pipelines-python-django
```

The sample code includes an `azure-pipelines.yml` file at the root of the repository. You can use this file to build the project.

Follow all the instructions in [Create your first pipeline](#) to create a build pipeline for the sample project.

Build environment

You can use Azure Pipelines to build your Python projects without needing to set up any infrastructure of your own. Python is preinstalled on [Microsoft-hosted agents](#) in Azure Pipelines. You can use Linux, macOS, or Windows agents to run your builds.

For the exact versions of Python that are preinstalled, refer to [Microsoft-hosted agents](#). To install a specific version of Python on Microsoft hosted agents, add the [Use Python Version](#) task to the beginning of your pipeline.

Use a specific Python version

Add the [Use Python Version](#) task to set the version of Python used in your pipeline. This snippet sets subsequent pipeline tasks to use Python 3.6.

```
# https://docs.microsoft.com/azure/devops/pipelines/languages/python
pool:
  vmImage: 'ubuntu-16.04' # other options: 'macOS-10.13', 'vs2017-win2016'

steps:
- task: UsePythonVersion@0
  inputs:
    versionSpec: '3.6'
    architecture: 'x64'
```

Use multiple Python versions

To run a pipeline with multiple Python versions, such as to test your project using different versions, define a job with a matrix of Python version values. Then set the [Use Python Version](#) task to reference the matrix variable for its Python version. Increase the **parallel** value to simultaneously run the job for all versions in the matrix, depending on how many parallel jobs are available.

```

# https://aka.ms/yaml
jobs:
- job: 'Test'
  pool:
    vmImage: 'ubuntu-16.04' # other options: 'macOS-10.13', 'vs2017-win2016'
  strategy:
    matrix:
      Python27:
        python.version: '2.7'
      Python35:
        python.version: '3.5'
      Python36:
        python.version: '3.6'
    maxParallel: 3

  steps:
- task: UsePythonVersion@0
  inputs:
    versionSpec: '$(python.version)'
    architecture: 'x64'

# Add additional tasks to run using each Python version in the matrix above

```

Create and activate an Anaconda environment

As an alternative to the **Use Python Version** task, create and activate an Anaconda environment with a specified version of Python. The following YAML uses the same `python.version` variable defined in the job matrix above.

- [Hosted Ubuntu 16.04](#)
- [Hosted macOS](#)
- [Hosted VS2017](#)

```

- bash: echo "##vso[task.prependpath]/usr/share/miniconda/bin"
  displayName: Add conda to PATH

- bash: conda create --yes --quiet --name myEnvironment python=$PYTHON_VERSION # [other packages ...]
  displayName: Create Anaconda environment

```

If you have an [environment.yml](#) defined for your Anaconda environment, you can also use that to create your environment:

```

- script: conda env create --quiet --file environment.yml
  displayName: Create Anaconda environment

```

NOTE

When you activate an Anaconda environment, it changes the current environment by doing things like editing `$PATH`. However, because each build step runs in its own process, these changes won't be persisted in subsequent steps.

When you run a script in a later step after creating an Anaconda environment, you need to activate the environment for each step:

- [Hosted Ubuntu 16.04](#)
- [Hosted macOS](#)
- [Hosted VS2017](#)

```
- bash: |
  source activate myEnvironment
  pytest -m unit --junitxml=junit/unit-test.xml
  displayName: 'Unit tests'

- bash: |
  source activate myEnvironment
  pytest -m integration --junitxml=junit/integration-test.xml
  displayName: 'Integration tests'
```

Run a Python script

If you have a Python script checked into the repo, you can run it using **script**. Add the following YAML to run a Python file named `example.py`.

```
- script: python src/example.py
```

If you want to write a Python script inline in the YAML file, use the [Python Script](#) task. Set the **targetType** to `inline` and put your code in the **script** section.

```
- task: PythonScript@0
  inputs:
    targetType: 'inline'
    script: |
      print('Hello world 1')
      print('Hello world 2')
```

Install dependencies

Install specific PyPI packages with pip

Add the following YAML to install or upgrade `pip` and two specific packages: `setuptools` and `wheel`.

```
- script: python -m pip install --upgrade pip setuptools wheel
  displayName: 'Install tools'
```

Install requirements with pip

After updating `pip` and friends, a typical next step is to install from `requirements.txt`.

```
- script: pip install -r requirements.txt
  displayName: 'Install requirements'
```

Install Anaconda packages with conda

Add the following YAML to install the `scipy` package in the conda environment named `myEnvironment`. See [Activate an Anaconda environment](#) above.

```
- script: conda install -n myEnvironment scipy
  displayName: 'Install conda libraries'
```

Test

Run lint tests with Flake8

Add the following YAML to install or upgrade `flake8` and use it to run lint tests.

```
- script: |
  python -m pip install flake8
  flake8 .
displayName: 'Run lint tests'
```

Test with pytest and collect coverage metrics with pytest-cov

Add the following YAML to install `pytest` and `pytest-cov`, run tests, output test results in JUnit format, and output code coverage results in Cobertura XML format.

```
- script: |
  pip install pytest
  pip install pytest-cov
  pytest tests --doctest-modules --junitxml=junit/test-results.xml --cov=com --cov-report=xml --cov-report=html
displayName: 'Test with pytest'
```

Publish test results

Add the [Publish Test Results](#) task to publish JUnit or xUnit test results to the server.

```
- task: PublishTestResults@2
inputs:
  testResultsFiles: '**/test-*.xml'
  testRunTitle: 'Publish test results for Python $(python.version)'
```

Publish code coverage results

Add the [Publish Code Coverage Results](#) task to publish code coverage results to the server. When you do this, coverage metrics can be seen in the build summary and HTML reports can be downloaded for further analysis.

```
- task: PublishCodeCoverageResults@1
inputs:
  codeCoverageTool: Cobertura
  summaryFileLocation: '$(System.DefaultWorkingDirectory)/**/coverage.xml'
  reportDirectory: '$(System.DefaultWorkingDirectory)/**/htmlcov'
```

Package and deliver your code

Authenticate with twine

The [Twine Authenticate task](#) stores authentication credentials for twine in the `PYPIRC_PATH` environment variable.

```
- task: TwineAuthenticate@0
inputs:
  artifactFeeds: 'feed_name1, feed_name2'
  externalFeeds: 'feed_name1, feed_name2'
```

Publish with twine

Then, add a [custom script task](#) to use `twine` to publish your packages.

```
- script: 'twine -r {feedName/EndpointName} --config-file $(PYPIRC_PATH) {package path to publish}'
```

Build a container image

You can also build and publish a Docker container image for your app. For more information, see [Docker](#).

Related extensions

- [Python Build Tools \(for Windows\)](#) (Steve Dower)
- [PyLint Checker](#) (Darren Fuller)
- [Python Test](#) (Darren Fuller)
- [Azure Pipelines Plugin for PyCharm \(IntelliJ\)](#) (Microsoft)
- [Python extension for Visual Studio Code](#) (Microsoft)

Build Ruby projects with Azure Pipelines

10/13/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

This guidance explains how to use Azure Pipelines to automatically build Ruby projects with CI/CD pipelines.

Example

For a working example of how to build a Ruby project, import (into Azure Repos or TFS) or fork (into GitHub) this repo:

```
https://github.com/MicrosoftDocs/pipelines-ruby
```

The sample code includes an `azure-pipelines.yml` file at the root of the repository. You can use this file to build the project.

Follow all the instructions in [Create your first pipeline](#) to create a build pipeline for the sample project.

Build environment

You can use Azure Pipelines to build your Ruby projects without needing to set up any infrastructure of your own. Ruby is preinstalled on [Microsoft-hosted agents](#) in Azure Pipelines. You can use Linux, macOS, or Windows agents to run your builds.

For the exact versions of Ruby that are preinstalled, refer to [Microsoft-hosted agents](#). To install a specific version of Ruby on Microsoft-hosted agents, add the [Use Ruby Version](#) task to the beginning of your pipeline.

Use a specific Ruby version

Add the [Use Ruby Version](#) task to set the version of Ruby used in your pipeline. This snippet adds Ruby 2.4 or later to the path and sets subsequent pipeline tasks to use it.

```
# https://docs.microsoft.com/azure/devops/pipelines/languages/ruby
pool:
  vmImage: 'ubuntu-16.04' # other options: 'macOS-10.13', 'vs2017-win2016'

steps:
- task: UseRubyVersion@0
  inputs:
    versionSpec: '>= 2.4'
    addToPath: true
```

Install Rails

To install Rails, add the following snippet to your `azure-pipelines.yml` file.

```
- script: gem install rails && rails -v
  displayName: 'gem install rails'
```

Install dependencies

To use Bundler to install dependencies, add the following snippet to your `azure-pipelines.yml` file.

```
- script: |
  gem install bundler
  bundle install --retry=3 --jobs=4
  displayName: 'bundle install'
```

Run Rake

To execute Rake in the context of the current bundle (as defined in your Gemfile), add the following snippet to your `azure-pipelines.yml` file.

```
- script: bundle exec rake
  displayName: 'bundle exec rake'
```

Publish test results

The sample code includes unit tests written using [RSpec](#). When Rake is run by the previous step, it runs the RSpec tests. The RSpec RakeTask in the Rakefile has been configured to produce JUnit style results using the `RspecJUnitFormatter`.

Add the [Publish Test Results](#) task to publish JUnit style test results to the server. When you do this, you get a rich test reporting experience that can be used for easily troubleshooting any failed tests and for test timing analysis.

```
- task: PublishTestResults@2
  inputs:
    testResultsFiles: '**/test-*.xml'
    testRunTitle: 'Ruby tests'
```

Publish code coverage results

The sample code uses [SimpleCov](#) to collect code coverage data when unit tests are run. SimpleCov is configured to use Cobertura and HTML report formatters.

Add the [Publish Code Coverage Results](#) task to publish code coverage results to the server. When you do this, coverage metrics can be seen in the build summary and HTML reports can be downloaded for further analysis.

```
- task: PublishCodeCoverageResults@1
  inputs:
    codeCoverageTool: Cobertura
    summaryFileLocation: '$(System.DefaultWorkingDirectory)/**/coverage.xml'
    reportDirectory: '$(System.DefaultWorkingDirectory)/**/coverage'
```

Build a container image

You can also build and publish a Docker container image for your Ruby app. For more information, see [Docker](#).

Build Xamarin apps with Azure Pipelines

11/19/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

This guidance explains how to use Azure Pipelines to automatically build Xamarin apps for Android and iOS.

Example

For a working example of how to build a Xamarin app, import (into Azure Repos or TFS) or fork (into GitHub) this repo:

```
https://github.com/MicrosoftDocs/pipelines-xamarin
```

The sample code includes an `azure-pipelines.yml` file at the root of the repository. You can use this file to build the app.

Follow all the instructions in [Create your first pipeline](#) to create a build pipeline for the sample app.

Build environment

You can use Azure Pipelines to build your Xamarin apps without needing to set up any infrastructure of your own. Xamarin tools are preinstalled on [Microsoft-hosted agents](#) in Azure Pipelines. You can use macOS or Windows agents to run Xamarin.Android builds, and macOS agents to run Xamarin.iOS builds. If you are using a self-hosted agent, you must install [Visual Studio Tools for Xamarin](#) for Windows agents or [Visual Studio for Mac](#) for macOS agents.

For the exact versions of Xamarin that are preinstalled, refer to [Microsoft-hosted agents](#).

Create a file named **azure-pipelines.yml** in the root of your repository. Then, add the following snippet to your `azure-pipelines.yml` file to select the appropriate agent pool:

```
# https://docs.microsoft.com/azure/devops/pipelines/languages/xamarin
pool:
  vmImage: 'macOS-10.13' # For Windows, use 'vs2017-win2016'
```

Build a Xamarin.Android app

To build a Xamarin.Android app, add the following snippet to your `azure-pipelines.yml` file. Change values to match your project configuration. See the [Xamarin.Android](#) task for more about these options.

```

variables:
  buildConfiguration: 'Release'
  outputDirectory: '$(build.binariesDirectory)/$(buildConfiguration)'

steps:
- task: NuGetToolInstaller@0

- task: NuGetCommand@2
  inputs:
    restoreSolution: '**/*.sln'

- task: XamarinAndroid@1
  inputs:
    configFile: '**/*Droid*.csproj'
    outputDirectory: '$(outputDirectory)'
    configuration: '$(buildConfiguration)'

```

Next steps

See [Android](#) guidance for information about:

- Signing and aligning an Android APK
- Testing on the Android Emulator
- Testing on Azure-hosted devices
- Retaining build artifacts with the build record
- Distributing through App Center
- Distributing through Google Play

Build a Xamarin.iOS app

To build a Xamarin.iOS app, add the following snippet to your `azure-pipelines.yml` file. Change values to match your project configuration. See the [Xamarin.iOS](#) task for more about these options.

```

variables:
  buildConfiguration: 'Release'

steps:
- task: XamariniOS@2
  inputs:
    solutionFile: '**/*.sln'
    configuration: '$(buildConfiguration)'
    packageApp: false
    buildForSimulator: true

```

Set the Xamarin SDK version on macOS

To set a specific Xamarin SDK version to use on the Microsoft-hosted macOS agent pool, add the following snippet before the `XamariniOS` task in your `azure-pipelines.yml` file. For details on properly formatting the version number (shown as **5_4_1** below), see [How can I manually select versions of tools on the Hosted macOS agent?](#)

```

- script: sudo $AGENT_HOMEDIRECTORY/scripts/select-xamarin-sdk.sh 5_4_1
  displayName: 'Select Xamarin SDK version'

```

Build Xamarin.Android and Xamarin.iOS apps with one pipeline

You can build and test your Xamarin.Android app, Xamarin.iOS app, and related apps in the same pipeline by defining [multiple jobs](#) in `azure-pipelines.yml`. These jobs can run in parallel to save time. The following complete

example builds a Xamarin.Android app on Windows, and a Xamarin.iOS app on macOS, using two jobs.

```
# https://docs.microsoft.com/vsts/pipelines/languages/xamarin
jobs:
- job: Android
  pool:
    vmImage: 'vs2017-win2016'
  variables:
    buildConfiguration: 'Release'
    outputDirectory: '$(build.binariesDirectory)/$(buildConfiguration)'
  steps:
    - task: NuGetToolInstaller@0
    - task: NuGetCommand@2
      inputs:
        restoreSolution: '**/*.sln'
    - task: XamarinAndroid@1
      inputs:
        configFile: '**/*droid*.csproj'
        outputDirectory: '$(outputDirectory)'
        configuration: '$(buildConfiguration)'

- job: iOS
  pool:
    vmImage: 'macOS-10.13'
  variables:
    buildConfiguration: 'Release'
  steps:
    - task: NuGetToolInstaller@0
    - task: NuGetCommand@2
      inputs:
        restoreSolution: '**/*.sln'
    - task: XamariniOS@2
      inputs:
        solutionFile: '**/*.sln'
        configuration: '$(buildConfiguration)'
        buildForSimulator: true
        packageApp: false
```

Next steps

See [Xcode](#) guidance for information about:

- Testing on Azure-hosted devices
- Retaining build artifacts with the build record
- Distributing through App Center
- Distributing through the Apple App Store

See [Sign your mobile app during CI](#) for information about signing and provisioning your app.

Build apps with Xcode using Azure Pipelines or Team Foundation Server

11/19/2018 • 4 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

This guidance explains how to use Azure Pipelines or Team Foundation Server (TFS) to automatically build Xcode projects with CI/CD pipelines.

Example

For a working example of how to build an app with Xcode, import (into Azure Repos or TFS) or fork (into GitHub) this repo:

```
https://github.com/MicrosoftDocs/pipelines-xcode
```

The sample code includes an `azure-pipelines.yml` file at the root of the repository. You can use this file to build the app.

Follow all the instructions in [Create your first pipeline](#) to create a build pipeline for the sample app.

Build environment

You can use Azure Pipelines to build your apps with Xcode without needing to set up any infrastructure of your own. Xcode is preinstalled on [Microsoft-hosted macOS agents](#) in Azure Pipelines. You can use the macOS agents to run your builds.

For the exact versions of Xcode that are preinstalled, refer to [Microsoft-hosted agents](#).

Create a file named **azure-pipelines.yml** in the root of your repository. Then, add the following snippet to your `azure-pipelines.yml` file to select the appropriate agent pool:

```
# https://docs.microsoft.com/azure/devops/pipelines/languages/xcode
pool:
  vmImage: 'macOS-10.13'
```

Build an app with Xcode

To build an app with Xcode, add the following snippet to your `azure-pipelines.yml` file. This is a minimal snippet for building an iOS project using its default scheme, for the Simulator, and without packaging. Change values to match your project configuration. See the [Xcode](#) task for more about these options.

```
variables:
  scheme: ''
  sdk: 'iphoneos'
  configuration: 'Release'

steps:
- task: Xcode@5
  inputs:
    sdk: '$(sdk)'
    scheme: '$(scheme)'
    configuration: '$(configuration)'
    xcodeVersion: 'default' # Options: default, 10, 9, 8, specifyPath
    exportPath: '$(agent.buildDirectory)/output/$(sdk)/$(configuration)'
    packageApp: false
```

Signing and provisioning

To sign and provision your app, see [Sign your mobile app during CI](#).

CocoaPods

If your project uses CocoaPods, you can run CocoaPods commands in your pipeline using a script, or with the [CocoaPods](#) task. The task optionally runs `pod repo update`, then runs `pod install`, and allows you to set a custom project directory. Following are common examples of using both.

```
- script: /usr/local/bin/pod install
  displayName: 'pod install using a script'

- task: CocoaPods@0
  displayName: 'pod install using the CocoaPods task with defaults'

- task: CocoaPods@0
  inputs:
    forceRepoUpdate: true
    projectDirectory: '$(system.defaultWorkingDirectory)'
  displayName: 'pod install using the CocoaPods task with a forced repo update and a custom project directory'
```

Testing on Azure-hosted devices

Add the [App Center Test](#) task to test the app in a hosted lab of iOS and Android devices. An [App Center](#) free trial is required which must later be converted to paid.

```

# App Center Test
# Test app packages with Visual Studio App Center.
- task: AppCenterTest@1
  inputs:
    appFile:
    #artifactsDirectory: '$(Build.ArtifactStagingDirectory)/AppCenterTest'
    #prepareTests: # Optional
    #frameworkOption: 'appium' # Required when prepareTests == True# Options: appium, espresso, calabash,
uitest, xcuitest
    #appiumBuildDirectory: # Required when prepareTests == True && Framework == Appium
    #espressoBuildDirectory: # Optional
    #espressoTestApkFile: # Optional
    #calabashProjectDirectory: # Required when prepareTests == True && Framework == Calabash
    #calabashConfigFile: # Optional
    #calabashProfile: # Optional
    #calabashSkipConfigCheck: # Optional
    #uiTestBuildDirectory: # Required when prepareTests == True && Framework == Uitest
    #uitestStoreFile: # Optional
    #uitestStorePassword: # Optional
    #uitestKeyAlias: # Optional
    #uitestKeyPassword: # Optional
    #uitestToolsDirectory: # Optional
    #signInfo: # Optional
    #xcUITestBuildDirectory: # Optional
    #xcUITestIpaFile: # Optional
    #prepareOptions: # Optional
    #runTests: # Optional
    #credentialsOption: 'serviceEndpoint' # Required when runTests == True# Options: serviceEndpoint, inputs
    #serverEndpoint: # Required when runTests == True && CredsType == ServiceEndpoint
    #username: # Required when runTests == True && CredsType == Inputs
    #password: # Required when runTests == True && CredsType == Inputs
    #appSlug: # Required when runTests == True
    #devices: # Required when runTests == True
    #series: 'master' # Optional
    #dsymDirectory: # Optional
    #localeOption: 'en_US' # Required when runTests == True# Options: da_DK, nl_NL, en_GB, en_US, fr_FR,
de_DE, ja_JP, ru_RU, es_MX, es_ES, user
    #userDefinedLocale: # Optional
    #loginOptions: # Optional
    #runOptions: # Optional
    #skipWaitingForResults: # Optional
    #cliFile: # Optional
    #showDebugOutput: # Optional

```

Retain artifacts with the build record

Add the [Copy Files](#) and [Publish Build Artifacts](#) tasks to store your IPA with the build record or test and deploy it in subsequent pipelines. See [Artifacts](#).

```

- task: CopyFiles@2
  inputs:
    contents: '**/*.ipa'
    targetFolder: '$(build.artifactStagingDirectory)'
- task: PublishBuildArtifacts@1

```

Deploy

App Center

Add the [App Center Distribute](#) task to distribute an app to a group of testers or beta users, or promote the app to Intune or the Apple App Store. A free [App Center](#) account is required (no payment is necessary).

```

# App Center Distribute
# Distribute app builds to testers and users via App Center
- task: AppCenterDistribute@1
  inputs:
    serverEndpoint:
    appSlug:
    appFile:
    #symbolsOption: 'Apple' # Optional. Options: apple
    #symbolsPath: # Optional
    #symbolsPdbFiles: '**/*.pdb' # Optional
    #symbolsDsymFiles: # Optional
    #symbolsMappingTxtFile: # Optional
    #symbolsIncludeParentDirectory: # Optional
    #releaseNotesOption: 'input' # Options: input, file
    #releaseNotesInput: # Required when releaseNotesOption == Input
    #releaseNotesFile: # Required when releaseNotesOption == File
    #distributionGroupId: # Optional

```

Apple App Store

Install the [Apple App Store extension](#) and use the following tasks to automate interaction with the App Store. By default, these tasks authenticate to Apple using a [service connection](#) that you configure.

Release

Add the [App Store Release](#) task to automate the release of updates to existing iOS TestFlight beta apps or production apps in the App Store.

```

- task: AppStoreRelease@1
  displayName: 'Publish to the App Store TestFlight track'
  inputs:
    serviceEndpoint: 'My Apple App Store service connection' # This service connection must be added by you
    appIdIdentifier: com.yourorganization.testapplication.etc
    ipaPath: '${build.artifactstagingdirectory}/**/*.ipa'
    shouldSkipWaitingForProcessing: true
    shouldSkipSubmission: true

```

Promote

Add the [App Store Promote](#) task to automate the promotion of a previously submitted app from iTunes Connect to the App Store.

```

- task: AppStorePromote@1
  displayName: 'Submit to the App Store for review'
  inputs:
    serviceEndpoint: 'My Apple App Store service connection' # This service connection must be added by you
    appIdIdentifier: com.yourorganization.testapplication.etc
    shouldAutoRelease: false

```

Related extensions

- [Apple App Store](#) (Microsoft)
- [Codified Security](#) (Codified Security)
- [MacinCloud](#) (Moboware Inc.)
- [Mobile App Tasks for iOS and Android](#) (James Montemagno)
- [Mobile Testing Lab](#) (Perfecto Mobile)
- [Raygun](#) (Raygun)
- [React Native](#) (Microsoft)
- [Version Setter](#) (Tom Gilder)

Deploy apps to Azure Stack

11/15/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Azure Stack is an extension of Azure that enables the agility and fast-paced innovation of cloud computing through a hybrid cloud and on-premises environment.

At present, Team Foundation Server can be used to deploy to Azure Stack with Azure AD and cannot be used to deploy to an Azure Stack with Azure Directory Federated Services (AD FS). Azure Stack with AD FS requires a [service principal with certificate](#), which is not currently supported in Azure Pipelines/TFS.

To enable connection to an Azure Stack, you specify it as the **Environment** parameter when you create an [Azure Resource Manager service connection](#). You must use the full version of the service connection dialog to manually define the connection. Before you configure a service connection, you should also ensure you meet all relevant compliance requirements for your application.

You can then use the service connection in your [build and release pipeline tasks](#).

Next

- [Deploy an Azure Web App](#)
- [Troubleshoot Azure Resource Manager service connections](#)
- [Azure Stack Operator Documentation](#)

Q&A

Are all the Azure tasks supported?

The following Azure tasks are validated with Azure Stack:

- [Azure PowerShell](#)
- [Azure File Copy](#)
- [Azure Resource Group Deployment](#)
- [Azure App Service Deploy](#)
- [Azure App Service Manage](#)
- [Azure SQL Database Deployment](#)

How do I resolve SSL errors during deployment?

To ignore SSL errors, set a variable named `VSTS_ARM_REST_IGNORE_SSL_ERRORS` to the value `true` in the build or release pipeline.

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Azure SQL database deployment

10/15/2018 • 7 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

You can automatically deploy your database updates to Azure SQL database after every successful build. Before you read this topic, you should understand the type of pipeline that you're creating: [designer](#) or [YAML](#).

DACPAC

The simplest way to deploy a database is to create [data-tier package or DACPAC](#). DACPACs can be used to package and deploy schema changes as well as data. You can create a DACPAC using the **SQL database project** in Visual Studio.

- [YAML](#)
- [Designer](#)

YAML is not supported in TFS.

To deploy a DACPAC to an Azure SQL database, add the following snippet to your azure-pipelines.yml file.

```
- task: SqlAzureDacpacDeployment@1
  displayName: Execute Azure SQL : DacpacTask
  inputs:
    azureSubscription: '<Azure service connection>'
    ServerName: '<Database server name>'
    DatabaseName: '<Database name>'
    SqlUsername: '<SQL user name>'
    SqlPassword: '<SQL user password>'
    DacpacFile: '<Location of Dacpac file in $(Build.SourcesDirectory) after compilation>'
```

SQL scripts

Instead of using a DACPAC, you can also use SQL scripts to deploy your database. Here is a simple example of a SQL script that creates an empty database.

```
USE [master]
GO
IF NOT EXISTS (SELECT name FROM master.sys.databases WHERE name = N'DatabaseExample')
CREATE DATABASE [DatabaseExample]
GO
```

To run SQL scripts as part of a pipeline, you will need Azure Powershell scripts to create and remove firewall rules in Azure. Without the firewall rules, the Azure Pipelines agent cannot communicate with Azure SQL Database.

The following Powershell script creates firewall rules. You can check-in this script as `SetAzureFirewallRule.ps1` into your repository.

```

[CmdletBinding(DefaultParameterSetName = 'None')]
param
(
    [String] [Parameter(Mandatory = $true)] $ServerName,
    [String] $AzureFirewallName = "AzureWebAppFirewall"
)

$ErrorActionPreference = 'Stop'

function New-AzureSQLServerFirewallRule {
    $agentIP = (New-Object net.webclient).downloadstring("http://checkip.dyndns.com") -replace "[^\d\.]"
    New-AzureSqlDatabaseServerFirewallRule -StartIPAddress $agentIp -EndIPAddress $agentIp -RuleName
    $AzureFirewallName -ServerName $ServerName
}
function Update-AzureSQLServerFirewallRule{
    $agentIP= (New-Object net.webclient).downloadstring("http://checkip.dyndns.com") -replace "[^\d\.]"
    Set-AzureSqlDatabaseServerFirewallRule -StartIPAddress $agentIp -EndIPAddress $agentIp -RuleName
    $AzureFirewallName -ServerName $ServerName
}

If ((Get-AzureSqlDatabaseServerFirewallRule -ServerName $ServerName -RuleName $AzureFirewallName -ErrorAction
SilentlyContinue) -eq $null)
{
    New-AzureSQLServerFirewallRule
}
else
{
    Update-AzureSQLServerFirewallRule
}

```

The following Powershell script removes firewall rules. You can check-in this script as `RemoveAzureFirewall.ps1` into your repository.

```

[CmdletBinding(DefaultParameterSetName = 'None')]
param
(
    [String] [Parameter(Mandatory = $true)] $ServerName,
    [String] $AzureFirewallName = "AzureWebAppFirewall"
)

$ErrorActionPreference = 'Stop'

If ((Get-AzureSqlDatabaseServerFirewallRule -ServerName $ServerName -RuleName $AzureFirewallName -ErrorAction
SilentlyContinue))
{
    Remove-AzureSqlDatabaseServerFirewallRule -RuleName $AzureFirewallName -ServerName $ServerName
}

```

- [YAML](#)
- [Designer](#)

Add the following to your `azure-pipelines.yml` file to run a SQL script.

```

variables:
  AzureSubscription: '<Azure service connection>'
  ServerName: '<Database server name>'
  DatabaseName: '<Database name>'
  AdminUser: '<SQL user name>'
  AdminPassword: '<SQL user password>'
  SQLFile: '<Location of SQL file in $(Build.SourcesDirectory)>'

steps:
- task: AzurePowerShell@2
  displayName: Azure PowerShell script: FilePath
  inputs:
    azureSubscription: '$(AzureSubscription)'
    ScriptPath: '$(Build.SourcesDirectory)\scripts\SetAzureFirewallRule.ps1'
    ScriptArguments: '$(ServerName)'
    azurePowerShellVersion: LatestVersion

- task: CmdLine@1
  displayName: Run Sqlcmd
  inputs:
    filename: Sqlcmd
    arguments: '-S $(ServerName) -U $(AdminUser) -P $(AdminPassword) -d $(DatabaseName) -i $(SQLFile)'

- task: AzurePowerShell@2
  displayName: Azure PowerShell script: FilePath
  inputs:
    azureSubscription: '$(AzureSubscription)'
    ScriptPath: '$(Build.SourcesDirectory)\scripts\RemoveAzureFirewallRule.ps1'
    ScriptArguments: '$(ServerName)'
    azurePowerShellVersion: LatestVersion

```

YAML is not supported in TFS.

Azure service connection

The **Azure SQL Database Deployment** task is the primary mechanism to deploy a database to Azure. This task, as with other built-in Azure tasks, requires an Azure service connection as an input. The Azure service connection stores the credentials to connect from Azure Pipelines or TFS to Azure.

The easiest way to get started with this task is to be signed in as a user that owns both the Azure DevOps organization and the Azure subscription. In this case, you won't have to manually create the service connection. Otherwise, to learn how to create an Azure service connection, see [Create an Azure service connection](#).

To learn how to create an Azure service connection, see [Create an Azure service connection](#).

Deploying conditionally

You may choose to deploy only certain builds to your Azure database.

- [YAML](#)
- [Designer](#)

To do this in YAML, you can use one of these techniques:

- Isolate the deployment steps into a separate job, and add a condition to that job.
- Add a condition to the step.

The following example shows how to use step conditions to deploy only those builds that originate from master branch.

```

- task: SqlAzureDacpacDeployment@1
  condition: and(succeeded(), eq(variables['Build.SourceBranch'], 'refs/heads/master'))
  inputs:
    azureSubscription: '<Azure service connection>'
    ServerName: '<Database server name>'
    DatabaseName: '<Database name>'
    SqlUsername: '<SQL user name>'
    SqlPassword: '<SQL user password>'
    DacpacFile: '<Location of Dacpac file in $(Build.SourcesDirectory) after compilation>'


```

To learn more about conditions, see [Specify conditions](#).

YAML builds are not yet available on TFS.

Additional SQL actions

SQL Azure Dacpac Deployment may not support all SQL server actions that you want to perform. In these cases, you can simply use Powershell or command line scripts to run the commands you need. This section shows some of the common use cases for invoking the `SqlPackage.exe` tool. As a prerequisite to running this tool, you must use a self-hosted agent and have the tool installed on your agent.

NOTE

If you execute **SQLPackage** from the folder where it is installed, you must prefix the path with `&` and wrap it in double-quotes.

Basic Syntax

```
<Path of SQLPackage.exe> <Arguments to SQLPackage.exe>
```

You can use any of the following SQL scripts depending on the action that you want to perform

Extract

Creates a database snapshot (.dacpac) file from a live SQL server or Microsoft Azure SQL Database.

Command Syntax:

```
SqlPackage.exe /TargetFile:"<Target location of dacpac file>" /Action:Extract
/SourceServerName:"<ServerName>.database.windows.net"
/SourceDatabaseName:"<DatabaseName>" /SourceUser:"<Username>" /SourcePassword:"<Password>"
```

or

```
SqlPackage.exe /action:Extract /tf:"<Target location of dacpac file>"
/SourceConnectionString:"Data Source=ServerName;Initial Catalog=DatabaseName;Integrated Security=SSPI;Persist Security Info=False;"
```

Example:

```
SqlPackage.exe /TargetFile:"C:\temp\test.dacpac" /Action:Extract
/SourceServerName:"DemoSqlServer.database.windows.net"
/SourceDatabaseName:"Testdb" /SourceUser:"ajay" /SourcePassword:"SQLPassword"
```

Help:

```
sqlpackage.exe /Action:Extract /?
```

Publish

Incrementally updates a database schema to match the schema of a source .dacpac file. If the database does not exist on the server, the publish operation will create it. Otherwise, an existing database will be updated.

Command Syntax:

```
SqlPackage.exe /SourceFile:"<Dacpac file location>" /Action:Publish /TargetServerName:"<ServerName>.database.windows.net"  
/TargetDatabaseName:"<DatabaseName>" /TargetUser:"<Username>" /TargetPassword:"<Password>"
```

Example:

```
SqlPackage.exe /SourceFile:"E:\dacpac\ajyadb.dacpac" /Action:Publish  
/TargetServerName:"DemoSqlServer.database.windows.net"  
/TargetDatabaseName:"Testdb4" /TargetUser:"ajay" /TargetPassword:"SQLPassword"
```

Help:

```
sqlpackage.exe /Action:Publish /?
```

Export

Exports a live database, including database schema and user data, from SQL Server or Microsoft Azure SQL Database to a BACPAC package (.bacpac file).

Command Syntax:

```
SqlPackage.exe /TargetFile:"<Target location for bacpac file>" /Action:Export /SourceServerName:"<ServerName>.database.windows.net"  
/SourceDatabaseName:"<DatabaseName>" /SourceUser:"<Username>" /SourcePassword:"<Password>"
```

Example:

```
SqlPackage.exe /TargetFile:"C:\temp\test.bacpac" /Action:Export  
/SourceServerName:"DemoSqlServer.database.windows.net"  
/SourceDatabaseName:"Testdb" /SourceUser:"ajay" /SourcePassword:"SQLPassword"
```

Help:

```
sqlpackage.exe /Action:Export /?
```

Import

Imports the schema and table data from a BACPAC package into a new user database in an instance of SQL Server or Microsoft Azure SQL Database.

Command Syntax:

```
SqlPackage.exe /SourceFile:"<Bacpac file location>" /Action:Import /TargetServerName:"<ServerName>.database.windows.net"  
/TargetDatabaseName:"<DatabaseName>" /TargetUser:"<Username>" /TargetPassword:"<Password>"
```

Example:

```
SqlPackage.exe /SourceFile:"C:\temp\test.bacpac" /Action:Import  
/TargetServerName:"DemoSqlServer.database.windows.net"  
/TargetDatabaseName:"Testdb" /TargetUser:"ajay" /TargetPassword:"SQLPassword"
```

Help:

```
sqlpackage.exe /Action:Import /?
```

DeployReport

Creates an XML report of the changes that would be made by a publish action.

Command Syntax:

```
SqlPackage.exe /SourceFile:<Dacpac file location> /Action:DeployReport /TargetServerName:<  
<ServerName>.database.windows.net">  
/TargetDatabaseName:<DatabaseName> /TargetUser:<Username> /TargetPassword:<Password> /OutputPath:<  
<Output XML file path for deploy report>"
```

Example:

```
SqlPackage.exe /SourceFile:"E: \dacpac\ajyadb.dacpac" /Action:DeployReport  
/TargetServerName:"DemoSqlServer.database.windows.net"  
/TargetDatabaseName:"Testdb" /TargetUser:"ajay" /TargetPassword:"SQLPassword"  
/OutputPath:"C:\temp\deployReport.xml"
```

Help:

```
sqlpackage.exe /Action:DeployReport /?
```

DriftReport

Creates an XML report of the changes that have been made to a registered database since it was last registered.

Command Syntax:

```
SqlPackage.exe /Action:DriftReport /TargetServerName:<ServerName>.database.windows.net" /TargetDatabaseName:<  
<DatabaseName>"  
/TargetUser:<Username> /TargetPassword:<Password> /OutputPath:<Output XML file path for drift report>"
```

Example:

```
SqlPackage.exe /Action:DriftReport /TargetServerName:"DemoSqlServer.database.windows.net"  
/TargetDatabaseName:"Testdb"  
/TargetUser:"ajay" /TargetPassword:"SQLPassword" /OutputPath:"C:\temp\driftReport.xml"
```

Help:

```
sqlpackage.exe /Action:DriftReport /?
```

Script

Creates a Transact-SQL incremental update script that updates the schema of a target to match the schema of a

source.

Command Syntax:

```
SqlPackage.exe /SourceFile:"<Dacpac file location>" /Action:Script /TargetServerName:<ServerName>.database.windows.net  
/TargetDatabaseName:"<DatabaseName>" /TargetUser:"<Username>" /TargetPassword:"<Password>" /OutputPath:<Output SQL script file path>"
```

Example:

```
SqlPackage.exe /Action:Script /SourceFile:"E:\dacpac\ajyadb.dacpac"  
/TargetServerName:"DemoSqlServer.database.windows.net"  
/TargetDatabaseName:"Testdb" /TargetUser:"ajay" /TargetPassword:"SQLPassword" /OutputPath:"C:\temp\test.sql"  
/Variables:StagingDatabase="Staging DB Variable value"
```

Help:

```
sqlpackage.exe /Action:Script /?
```

Azure Web App deployment

10/29/2018 • 9 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

You can automatically deploy your web app to an Azure App Service web app after every successful build. Before you read this topic, you should understand the type of pipeline that you're creating: [designer](#) or [YAML](#).

NOTE

This guidance applies to Team Foundation Server (TFS) version 2017.3 and later.

Example

If you want some sample code that works with this guidance, import (into Azure Repos or TFS) or fork (into GitHub) this repo:

```
https://github.com/MicrosoftDocs/pipelines-dotnet-core
```

Follow the guidance in [.NET Core](#) to build the sample code.

- [YAML](#)
- [Designer](#)

The preceding sample code includes an `azure-pipelines.yml` file at the root of the repository. This file contains code to build, test, and publish the source as an artifact. To learn more about building .NET Core apps, see [Build .NET Core projects with Azure Pipelines or Team Foundation Server](#).

YAML builds are not yet available on TFS.

Azure App Service Deploy task

- [YAML](#)
- [Designer](#)

The simplest way to deploy to an Azure Web App is to use the **Azure App Service Deploy** (`AzureRmWebAppDeployment`) task.

Deploy a Web Deploy package (ASP.NET)

To deploy a .zip Web Deploy package (for example, from an [ASP.NET web app](#)) to an Azure Web App, add the following snippet to your `azure-pipelines.yml` file:

```
- task: AzureRmWebAppDeployment@3
  inputs:
    azureSubscription: '<Azure service connection>'
    WebAppName: '<Name of web app>'
    Package: '$(System.ArtifactsDirectory)/**/*.zip'
```

The snippet assumes that the build steps in your YAML file produce the zip archive in the `$(System.ArtifactsDirectory)` folder on your agent.

For information on Azure service connections, see the [following section](#).

Deploy a Java app

If you're building a [Java app](#), use the following snippet to deploy the web archive (.war):

```
- task: AzureRmWebAppDeployment@3
  inputs:
    azureSubscription: '<Azure service connection>'
    WebAppName: '<Name of web app>'
    Package: '$(System.DefaultWorkingDirectory)/**/*.war'
```

The snippet assumes that the build steps in your YAML file produce the .war archive in one of the folders in your source code folder structure; for example, under `<project root>/build/libs`. If your build steps copy the .war file to `$(System.ArtifactsDirectory)` instead, change the last line in the snippet to `$(System.ArtifactsDirectory)/**/*.war`.

For information on Azure service connections, see the [following section](#).

Deploy a JavaScript Node.js app

If you're building a [JavaScript Node.js app](#), you publish the entire contents of your working directory to the web app. The following snippet also generates a Web.config file during deployment and starts the iisnode handler on the Azure Web App:

```
- task: AzureRmWebAppDeployment@3
  inputs:
    azureSubscription: '<Azure service connections>'
    WebAppName: '<Name of web app>'
    Package: '$(System.DefaultWorkingDirectory)'
    GenerateWebConfig: true
    WebConfigParameters: '-Handler iisnode -NodeStartFile server.js -appType node'
```

For information on Azure service connections, see the [following section](#).

YAML builds are not yet available on TFS.

Azure service connection

The **Azure App Service Deploy** task, similar to other built-in Azure tasks, requires an Azure service connection as an input. The Azure service connection stores the credentials to connect from Azure Pipelines or TFS to Azure.

- [YAML](#)
- [Designer](#)

You must supply an Azure service connection to the `AzureRmWebAppDeployment` task. The Azure service connection stores the credentials to connect from Azure Pipelines to Azure. See [Create an Azure service connection](#).

YAML builds are not yet available on TFS.

Deploy to a virtual application

- [YAML](#)
- [Designer](#)

By default, your deployment happens to the root application in the Azure Web App. You can deploy to a specific virtual application by using the following:

```
- task: AzureRmWebAppDeployment@3
  inputs:
    VirtualApplication: '<name of virtual application>'
```

YAML builds are not yet available on TFS.

Deploy to a slot

- [YAML](#)
- [Designer](#)

You can configure the Azure Web App to have multiple slots. Slots allow you to safely deploy your app and test it before making it available to your customers.

The following example shows how to deploy to a staging slot, and then swap to a production slot:

```
- task: AzureRmWebAppDeployment@3
  inputs:
    azureSubscription: '<Azure service connection>'
    WebAppName: '<name of web app>'
    DeployToSlotFlag: true
    ResourceGroupName: '<name of resource group>'
    SlotName: staging

- task: AzureAppServiceManage@0
  inputs:
    azureSubscription: '<Azure service connection>'
    WebAppName: '<name of web app>'
    ResourceGroupName: '<name of resource group>'
    SourceSlot: staging
```

YAML builds are not yet available on TFS.

Deploy to multiple web apps

- [YAML](#)
- [Designer](#)

You can use [jobs](#) in your YAML file to set up a pipeline of deployments. By using jobs, you can control the order of deployment to multiple web apps.

```

jobs:

- job: buildandtest
  pool:
    vmImage: 'ubuntu-16.04'
  steps:
    # add steps here to build the app

    # the following will publish an artifact called drop
    - task: PublishBuildArtifacts@1

    - task: AzureRmWebAppDeployment@3
      inputs:
        azureSubscription: '<Test stage Azure service connection>'
        WebAppName: '<name of test stage web app>'

- job: prod
  pool:
    vmImage: 'ubuntu-16.04'
  dependsOn: buildandtest
  condition: succeeded()
  steps:
    # step to download the artifacts from the previous job
    - task: DownloadBuildArtifacts@0
      inputs:
        artifactName: drop

    - task: AzureRmWebAppDeployment@3
      inputs:
        azureSubscription: '<Prod stage Azure service connection>'
        WebAppName: '<name of prod stage web app>'

```

YAML builds are not yet available on TFS.

Configuration changes

You might want to apply a specific configuration for your web app target before deploying to it. This is particularly useful when you deploy the same build to multiple web apps in a pipeline. For example, if your Web.config file contains a connection string named `connectionString`, you can change its value before deploying to each web app. You can do this either by applying a Web.config transformation or by substituting variables in your Web.config file.

- [YAML](#)
- [Designer](#)

The following snippet shows an example of variable substitution:

```

jobs:
- job: test
  variables:
    connectionString: <test-stage connection string>
  steps:
    - task: AzureRmWebAppDeployment@3
      inputs:
        azureSubscription: '<Test stage Azure service connection>'
        WebAppName: '<name of test stage web app>'
        enableXmlVariableSubstitution: true

- job: prod
  dependsOn: test
  variables:
    connectionString: <prod-stage connection string>
  steps:
    - task: AzureRmWebAppDeployment@3
      inputs:
        azureSubscription: '<Prod stage Azure service connection>'
        WebAppName: '<name of prod stage web app>'
        enableXmlVariableSubstitution: true

```

YAML builds are not yet available on TFS.

Deploying conditionally

You can choose to deploy only certain builds to your Azure Web App.

- [YAML](#)
- [Designer](#)

To do this in YAML, you can use one of these techniques:

- Isolate the deployment steps into a separate job, and add a condition to that job.
- Add a condition to the step.

The following example shows how to use step conditions to deploy only builds that originate from the master branch:

```

- task: AzureRmWebAppDeployment@3
  condition: and(succeeded(), eq(variables['Build.SourceBranch'], 'refs/heads/master'))
  inputs:
    azureSubscription: '<Azure service connection>'
    WebAppName: '<Name of web app>'

```

To learn more about conditions, see [Specify conditions](#).

YAML builds are not yet available on TFS.

Deploy to an Azure Government cloud or to Azure Stack

Create a suitable service connection:

- [Azure Government Cloud deployment](#)
- [Azure Stack deployment](#)

Deployment mechanisms

The preceding examples rely on the built-in [Azure App Service Deploy task](#), which provides simplified integration

with Azure.

If you use a Windows agent, this task uses Web Deploy technology to interact with the Azure Web App. Web Deploy provides several convenient deployment options, such as renaming locked files and excluding files from the App_Data folder during deployment.

If you use the Linux agent, the task relies on the [Kudu REST APIs](#).

The [Azure App Service Manage task](#) is another task that's useful for deployment. You can use this task to start, stop, or restart the web app before or after deployment. You can also use this task to swap slots, install site extensions, or enable monitoring of the web app.

If the built-in tasks don't meet your needs, you can use other methods to script your deployment. View the YAML snippets in each of the following tasks for some examples:

- [Azure PowerShell task](#)
- [Azure CLI task](#)
- [FTP task](#)

Deploy to a Linux Virtual Machine

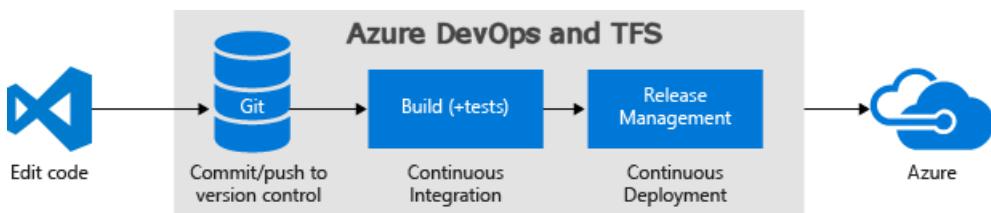
10/9/2018 • 5 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

We'll show you how to set up continuous deployment of your app to an nginx web server running on Ubuntu using Azure Pipelines or Team Foundation Server (TFS) 2018. You can use the steps in this quickstart for any app as long as your continuous integration pipeline publishes a web deployment package.



After you commit and push a code change, it is automatically built and then deployed. The results will automatically show up on your site.

Define your CI build pipeline

You'll need a continuous integration (CI) build pipeline that publishes your web application, as well as a deployment script that can be run locally on the Ubuntu server. To set up a CI build pipeline, see:

- [Build your Node.js app with gulp](#)

Make sure you follow the additional steps in that topic for creating a build to deploy to Linux.

Prerequisites

You'll need a Linux VM with Nginx web server to deploy the app. The deployment scripts used in the sample repositories have been tested on Ubuntu 16.04, and we recommend you use the same version of Linux VM for this quickstart. If you don't already have a Linux VM with Nginx, create one now in Azure using the steps in [this example](#).

Create a deployment group

Deployment groups in Azure Pipelines make it easier to organize the servers you want to use to host your app. A deployment group is a collection of machines with an Azure Pipelines agent on each of them. Each machine interacts with Azure Pipelines to coordinate deployment of your app.

1. Open a SSH session to your Linux VM. You can do this using the Cloud Shell button on the menu in the upper-right of the [Azure portal](#).



2. Initiate the session by typing the following command, substituting the IP address of your VM:

```
ssh <publicIpAddress>
```

For more information, see [SSH into your VM](#).

3. Run the following command:

```
sudo apt-get install -y libunwind8 libcurl3
```

The libraries this command installs are pre-requisites for installing the build and release agent onto a Ubuntu 16.04 VM. Pre-requisites for other versions of Linux can be found [here](#).

4. Open the Azure Pipelines web portal, navigate to **Azure Pipelines**, and choose **Deployment groups**.

5. Choose **Add Deployment group** (or **New** if you have existing deployment groups).

6. Enter a name for the group such as **myNginx** and choose **Create**.

7. In the **Register machine** section, make sure that **Ubuntu 16.04+** is selected and that **Use a personal access token in the script for authentication** is also checked. Choose **Copy script to clipboard**.

The script you've copied to your clipboard will download and configure an agent on the VM so that it can receive new web deployment packages and apply them to web server.

8. Back in the SSH session to your VM, paste and run the script.

9. When you're prompted to configure tags for the agent, press *Enter* (you don't need any tags).

10. Wait for the script to finish and display the message *Started Azure Pipelines Agent*. Type "q" to exit the file editor and return to the shell prompt.

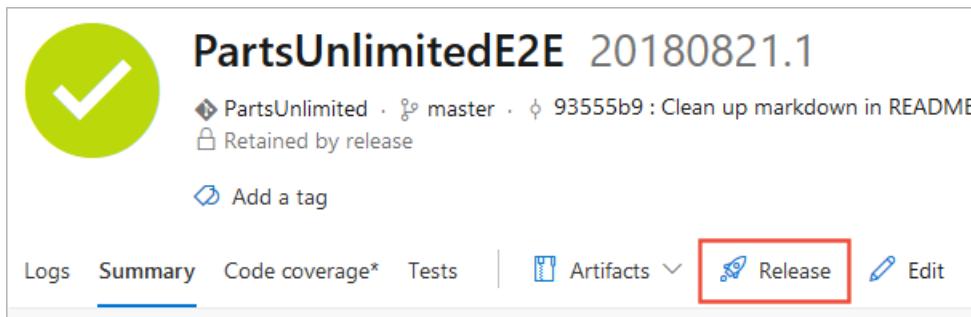
11. Back in Azure Pipelines or TFS, on the **Deployment groups** page, open the **myNginx** deployment group. On the **Targets** tab, verify that your VM is listed.

Define your CD release pipeline

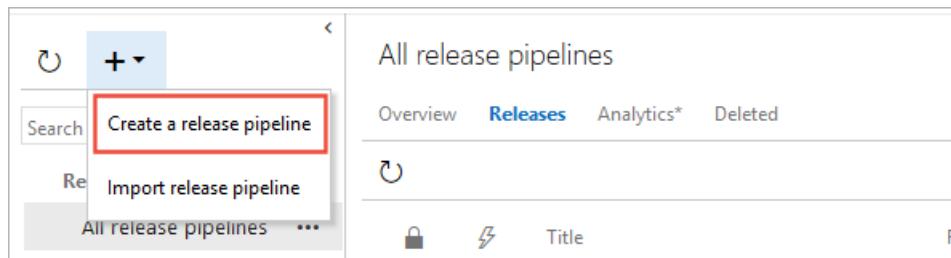
Your CD release pipeline picks up the artifacts published by your CI build and then deploys them to your nginx servers.

1. Do one of the following to start creating a release pipeline:

- If you've just completed a CI build then, in the build's **Summary** tab under **Deployments**, choose **Create release** followed by **Yes**. This starts a new release pipeline that's automatically linked to the build pipeline.



- Open the **Releases** tab of **Azure Pipelines**, open the + drop-down in the list of release pipelines, and choose **Create release pipeline**.



2. Choose **Start with an Empty job**.

3. If you created your new release pipeline from a build summary, check that the build pipeline and artifact is shown in the **Artifacts** section on the **Pipeline** tab. If you created a new release pipeline from the **Releases** tab, choose the **+ Add** link and select your build artifact.

4. Choose the **Continuous deployment** icon in the **Artifacts** section, check that the continuous deployment trigger is enabled, and add a filter that includes the **master** branch.

Continuous deployment is not enabled by default when you create a new release pipeline from the **Releases** tab.

5. Open the **Tasks** tab, select the **Agent job**, and choose **Remove** to remove this job.

The screenshot shows the 'Tasks' tab of a 'New release pipeline'. A single 'Agent job' is listed under 'Stage 1'. A red box highlights the 'Remove' button next to the job name. The 'Display name' field contains 'Agent job'.

6. Choose ... next to the **Stage 1** deployment pipeline and select **Add deployment group job**.

The screenshot shows the 'Tasks' tab of a 'New release pipeline'. Under 'Stage 1', a tooltip message says 'The stage 'Stage 1' should have at least one job.' A red box highlights the 'Add a deployment group job' option in the dropdown menu.

7. For the **Deployment Group**, select the deployment group you created earlier such as **myNginx**.

The screenshot shows the 'Tasks' tab of a 'New release pipeline'. Under 'Stage 1', a 'Deployment group job' is listed. In the 'Deployment targets' section, a red box highlights the 'Deployment group' dropdown menu, which is set to 'myNginx'.

The tasks you add to this job will run on each of the machines in the deployment group you specified.

8. Choose + next to the **Deployment group job** and, in the task catalog, search for and add a **Shell Script** task.

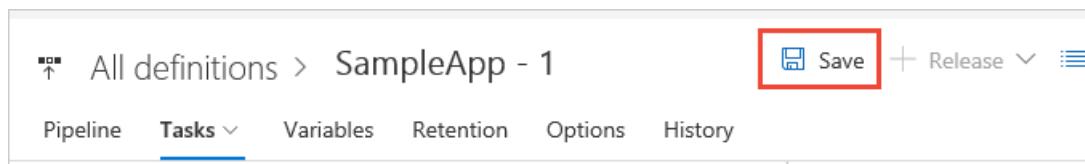
The screenshot shows the 'Tasks' tab in the Azure DevOps Pipeline interface. A 'Stage 1' deployment process is visible. In the 'Add tasks' search bar, 'shell script' is typed and highlighted with a red box. Below the search bar, a list of tasks is shown, each with a small icon and a brief description. The 'Shell Script' task, which includes a '#!' icon, is highlighted with a red box around its 'Add' button.

- In the properties of the **Shell Script** task, use the **Browse** button for the **Script Path** to select the path to the **deploy.sh** script in the build artifact. For example, when you use the **nodejs-sample** repository to build your app, the location of the script is

```
$(System.DefaultWorkingDirectory)/nodejs-sample/drop/deploy/deploy.sh
```

The screenshot shows the 'Select File Or Folder' dialog. On the left, a tree view shows the structure of a build artifact named 'myNodeJSApp (Build)'. Under the 'drop' folder, there is a 'deploy' folder containing a file named 'deploy.sh'. This 'deploy.sh' file is highlighted with a red box. At the bottom of the dialog, there is explanatory text about artifacts and a summary of the selected files. The 'Location' field at the bottom contains the path 'myNodeJSApp/drop/deploy/deploy.sh'. There are 'OK' and 'Cancel' buttons at the bottom right.

- Save the release pipeline.



Create a release to deploy your app

You're now ready to create a release, which means to start the process of running the release pipeline with the artifacts produced by a specific build. This will result in deploying the build.

1. Choose **+ Release** and select **Create a release**.
2. In the **Create a new release** panel, check that the artifact version you want to use is selected and choose **Create**.
3. Choose the release link in the information bar message. For example: "Release **Release-1** has been created".
4. In the pipeline view, choose the status link in the stages of the pipeline to see the logs and agent output.
5. After the release is complete, navigate to your app and verify its contents.

Next steps

- [Dynamically create and remove a deployment group](#)
- [Apply stage-specific configurations](#)
- [Perform a safe rolling deployment](#)
- [Deploy a database with your app](#)

Publish npm packages

11/19/2018 • 2 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

You can publish npm packages produced by your build to:

- Azure Artifacts or the TFS Package Management service
- Other registries such as <https://registry.npmjs.org/>

Before you read this topic, you should understand the kind of build pipeline you're creating:[designer](#) or [YAML](#).

- [YAML](#)
- [Designer](#)

To publish to an Azure Artifacts feed, set the **Project Collection Build Service** identity to be a **Contributor** on the feed. To learn more about permissions to Package Management feeds, see [Secure and share packages using feed permissions](#).

To publish to a external npm registry, you must first create a service connection to point to that feed. You can do this by going to **Project settings**, then choosing **Services**, and then creating a **New service connection**. Select the **npm** option for the service connection. Fill in registry URL and the credentials to connect to the registry.

To publish a package to a npm registry, add the following snippet to your azure-pipelines.yml file.

```
- task: Npm@1
  inputs:
    command: publish
    publishEndpoint: '<copy and paste the name of the service connection here>'
```

For a list of other options, see the [npm task](#).

YAML is not supported in TFS.

NOTE

Build does not support using the `publishConfig` property to specify the `registry` to which you're publishing. Ensure your working folder has a `.npmrc` file with a `registry=` line, as detailed in the Connect to feed screen in your feed.

Q&A

Where can I learn about the Azure Pipelines and TFS Package management service?

[Package Management service](#)

Publish to NuGet feeds

11/29/2018 • 7 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

You can publish NuGet packages from your build to NuGet feeds. You can publish these packages to

- Azure Artifacts or the TFS Package Management service
- Other NuGet services such as NuGet.org
- Your internal NuGet repository

Before you read this topic, you should understand the kind of build pipeline you're creating: [designer](#) or [YAML](#).

Create a NuGet package

There are a variety of ways to create NuGet packages during a build. If you're already using MSBuild or some other task to create your packages, skip this section to go ahead and [publish your packages](#). Otherwise, add a **NuGet** task:

- [YAML](#)
- [Designer](#)

To create a package, add the following snippet to your azure-pipelines.yml file.

```
- task: NuGetCommand@2
  inputs:
    command: pack
    packagesToPack: '**/*.csproj'
```

The NuGet task supports a number of options. Some of the key ones are described below. The rest are described in the [task documentation](#).

- **packagesToPack:** The path to the files that describe the package you want to create. If you don't have these, see the [NuGet documentation](#) to get started.
- **configuration:** The default is `$(BuildConfiguration)` unless you wish to always build either `Debug` or `Release` packages, or unless you have a custom build configuration.
- **packDestination:** The default is `$(Build.ArtifactStagingDirectory)`. If you set this, make a note of the location so you can use it in the [publish task](#).

YAML is not supported in TFS.

Package versioning

In NuGet, a particular package is identified by its name and version number. A recommended approach to versioning packages is to use semantic versioning. Semantic version numbers have 3 numeric components, `Major.Minor.Patch`. When you fix a bug, you increment the patch (`1.0.0` ? `1.0.1`). When you release a new

backwards-compatible feature, you increment minor and reset patch to 0 (`1.4.17` ? `1.5.0`). When you make a backwards-incompatible change, you increment major and reset minor and patch to 0 (`2.6.5` ? `3.0.0`).

In addition to `Major.Minor.Patch`, semantic versioning provides for a prerelease label. Prerelease labels are a “-” followed by whatever letters and numbers you want. Version `1.0.0-alpha`, `1.0.0-beta`, and `1.0.0-foo12345` are all prerelease versions of `1.0.0`. Even better, semantic versioning specifies that when you sort by version number, those prerelease versions fit exactly where you’d expect: `0.99.999` < `1.0.0-alpha` < `1.0.0` < `1.0.1-beta`.

When you create a package in CI, you can use semantic versioning with prerelease labels. The **NuGet** task can be used for this purpose, and supports the following formats:

- Use the same versioning scheme for your builds and packages, provided that scheme has at least three parts separated by periods. The following build pipeline formats are examples of versioning schemes that are compatible with NuGet.
 - `$(Major).$(Minor).$(rev:r)`, where `Major` and `Minor` are two variables defined in the build pipeline. This will automatically increment the build number and the package version with a new patch number keeping the major and minor versions constant, until you change them manually in the build pipeline.
 - `$(Major).$(Minor).$(Patch).$(date:yyyyMMdd)`, where `Major`, `Minor`, and `Patch` are variables defined in the build pipeline. This will create a new prerelease label for the build and package while keeping the major, minor, and patch versions constant.
- Use a version that is different from the build number. You can customize the major, minor, and patch versions for your packages in the NuGet task, and let the task generate a unique prerelease label based on date and time.
- Use a script in your build pipeline to generate the version.
- [YAML](#)
- [Designer](#)

This example shows how to use the date and time as the prerelease label.

```
variables:  
  Major: '1'  
  Minor: '0'  
  Patch: '0'  
  
steps:  
- task: NuGetCommand@2  
  inputs:  
    command: pack  
    versioningScheme: byPrereleaseNumber  
    majorVersion: '$(Major)'  
    minorVersion: '$(Minor)'  
    patchVersion: '$(Patch)'
```

For a list of other possible values for `versioningScheme`, see the [NuGet task](#).

YAML is not supported in TFS.

While semantic versioning with prerelease labels is a good solution for packages produced in CI builds, including a prerelease label is not ideal when you want to release a package to your users. The challenge is that packages once produced are [immutable](#) and so cannot be updated or replaced. When you’re producing a package in build, you can’t know whether it will be the version that you aim to release to your users or just a step along the way towards that release. While none of the following solutions are ideal, you can use one of these depending on your preference:

- Once you validate a package and decide to release it, produce another package without the prerelease label

and publish it. The drawback of this approach is that you have to validate the new package again, and it may uncover new issues.

- Publish only packages that you wish to release. In this case, you will not use a prerelease label for every build. Instead, you will reuse the same package version for all packages. Since you do not publish packages from every build, you do not cause a conflict.

Publish your packages

In the previous section, you learned how to create a package with every build. When you are ready to share the changes to your package with your users, you can publish it.

- [YAML](#)
- [Designer](#)

To publish to an Azure Artifacts feed, set the **Project Collection Build Service** identity to be a **Contributor** on the feed. To learn more about permissions to Package Management feeds, see [Secure and share packages using feed permissions](#). Add the following snippet to your `azure-pipelines.yml` file.

```
steps:  
- task: NuGetCommand@2  
  displayName: 'NuGet push'  
  inputs:  
    command: push  
    publishVstsFeed: 'af432f1b-f4b2-4a4f-bc9c-fef924f8f823'  
    allowPackageConflicts: true
```

To publish to a external NuGet feed, you must first create a service connection to point to that feed. You can do this by going to **Project settings**, then choosing **Service connections**, and then creating a **New service connection**. Select the **NuGet** option for the service connection. Fill in feed URL and the API key or token to connect to the feed.

To publish a package to a NuGet feed, add the following snippet to your `azure-pipelines.yml` file.

```
- task: NuGetCommand@2  
  inputs:  
    command: push  
    nuGetFeedType: external  
    publishFeedCredentials: '<Name of the NuGet service connection>'  
    versioningScheme: byEnvVar  
    versionEnvVar: Version
```

YAML is not supported in TFS.

Publish symbols for your packages

When you push packages to a Package Management feed, you can also [publish symbols](#).

Q&A

Where can I learn more about Azure Artifacts and the TFS Package Management service

[Package Management in Azure Artifacts and TFS](#)

2 minutes to read

SCVMM deployment

10/15/2018 • 8 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018

You can automatically provision new virtual machines in System Center Virtual Machine Manager (SCVMM) and deploy to those virtual machines after every successful build. Before this guidance, read the [web quickstart](#).

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions.

Service connections are called *service endpoints* in TFS 2018 and in older versions.

SCVMM connection

You need to first configure how Azure Pipelines connects to SCVMM. You cannot use Microsoft-hosted agents to run SCVMM tasks since the VMM Console is not installed on hosted agents. You must set up a self-hosted build and release agent on the same network as your SCVMM server.

You need to first configure how TFS connects to SCVMM. You must have a build and release agent that can communicate with the SCVMM server.

1. Install the **Virtual Machine Manager** (VMM) console on the agent machine by following [these instructions](#). Supported version: [System Center 2012 R2 Virtual Machine Manager](#).
2. Install the **System Center Virtual Machine Manager (SCVMM)** extension from Visual Studio Marketplace into TFS or Azure Pipelines:
 - If you are using **Azure Pipelines**, install the extension from [this location](#) in Visual Studio Marketplace.
 - If you are using **Team Foundation Server**, download the extension from [this location](#) in Visual Studio Marketplace, upload it to your Team Foundation Server, and install it.
3. Create an SCVMM service connection in your project:
 - In your Azure Pipelines or TFS project in your web browser, navigate to the project settings and select **Service connections**.
 - In the **Service connections** tab, choose **New service connection**, and select **SCVMM**.
 - In the **Add new SCVMM Connection** dialog, enter the values required to connect to the SCVMM Server:
 - **Connection Name:** Enter a user-friendly name for the service connection such as **MySCVMMServer**.
 - **SCVMM Server Name:** Enter the fully qualified domain name and port number of the SCVMM server, in the form **machine.domain.com:port**.
 - **Username** and **Password:** Enter the credentials required to connect to the vCenter Server. Username formats such as **username**, **domain\username**, **machine-name\username**, and **\username** are supported. UPN formats such as **username@domain.com** and built-in system accounts such as **NT Authority\System** are not supported.

Create new virtual machines from a template, VHD, or stored VM

One of the common actions that you can perform with every build is to create a new virtual machine to deploy the build to. You use the SCMVMM task from the extension to do this and configure the properties of the task as follows:

- **Display name:** The name for the task as it appears in the task list.
- **SCVMM Service Connection:** Select a SCVMM service connection you already defined, or create a new one.
- **Action:** Select **New Virtual Machine using Template/Stored VM/VHD**.
- **Create virtual machines from VM Templates:** Set this option if you want to use a template.
 - **Virtual machine names:** Enter the name of the virtual machine, or a list of the virtual machine names on separate lines. Example `FabrikamDevVM`
 - **VM template names:** Enter the name of the template, or a list of the template names on separate lines.
 - **Set computer name as defined in the VM template:** If not set, the computer name will be the same as the VM name.
- **Create virtual machines from stored VMs:** Set this option if you want to use a stored VM.
 - **Virtual machine names:** Enter the name of the virtual machine, or a list of the virtual machine names on separate lines. Example `FabrikamDevVM`
 - **Stored VMs:** Enter the name of the stored VM, or a list of the VMs on separate lines in the same order as the virtual machine names.
- **Create virtual machines from VHD:** Set this option if you want to use a stored VM.
 - **Virtual machine names:** Enter the name of the virtual machine, or a list of the virtual machine names on separate lines. Example `FabrikamDevVM`
 - **VHDs:** Enter the name of the VHD or VHDX, or a list of names on separate lines in the same order as the virtual machine names.
 - **CPU count:** Specify the number of processor cores required for the virtual machines.
 - **Memory:** Specify the memory in MB required for the virtual machines.
- **Clear existing network adapters:** Set this option if you want to remove the network adapters and specify new ones in the **Network Virtualization** options.
- **Deploy the VMs to:** Choose either **Cloud** or **Host** to select the set of virtual machines to which the action will be applied.
- **Host Name or Cloud Name:** Depending on the previous selection, enter either a cloud name or a host machine name.
- **Placement path for VM:** If you selected **Host** as the deployment target, enter the path to be used during virtual machine placement. Example `C:\ProgramData\Microsoft\Windows\Hyper-V`
- **Additional Arguments:** Enter any arguments to pass to the virtual machine creation template. Example
`-StartVM -StartAction NeverAutoTurnOnVM -StopAction SaveVM`
- **Wait Time:** The time to wait for the virtual machine to reach ready state.
- **Network Virtualization:** Set this option to enable network virtualization for your virtual machines. For more information, see [Create a virtual network isolated environment](#).
- **Show minimal logs:** Set this option if you don't want to create detailed live logs about the VM provisioning process.

Delete virtual machines

After validating your build, you would want to delete the virtual machines that you created. You use the SCMVMM task from the extension to do this and configure the properties of the task as follows:

- **Display name:** The name for the task as it appears in the task list.
- **SCVMM Service Connection:** Select a SCVMM service connection you already defined, or create a new one.
- **Action:** Select **New Virtual Machine using Template/Stored VM/VHD**.
- **VM Names:** Enter the name of the virtual machine, or a comma-separated list of the virtual machine names.

Example `FabrikamDevVM,FabrikamTestVM`

- **Select VMs From:** Choose either **Cloud** or **Host** to select the set of virtual machines to which the action will be applied.
- **Host Name or Cloud Name:** Depending on the previous selection, enter either a cloud name or a host machine name.

Start and stop virtual machines

You can start a virtual machine prior to deploying a build, and then stop the virtual machine after running tests. Use the SCVMM task as follows in order to achieve this:

- **Display name:** The name for the task as it appears in the task list.
 - **SCVMM Service Connection:** Select a SCVMM service connection you already defined, or create a new one.
 - **Action:** Select **Start Virtual Machine** or **Stop Virtual Machine**.
 - **VM Names:** Enter the name of the virtual machine, or a comma-separated list of the virtual machine names.
- Example `FabrikamDevVM,FabrikamTestVM`
- **Select VMs From:** Choose either **Cloud** or **Host** to select the set of virtual machines to which the action will be applied.
 - **Host Name or Cloud Name:** Depending on the previous selection, enter either a cloud name or a host machine name.
 - **Wait Time:** The time to wait for the virtual machine to reach ready state.

Create, restore, and delete checkpoints

A quick alternative to bringing up a virtual machine in desired state prior to running tests is to restore it to a known checkpoint. Use the SCVMM task as follows in order to do this:

- **Display name:** The name for the task as it appears in the task list.
 - **SCVMM Service Connection:** Select a SCVMM service connection you already defined, or create a new one.
 - **Action:** Select one of the checkpoint actions **Create Checkpoint**, **Restore Checkpoint**, or **Delete Checkpoint**.
 - **VM Names:** Enter the name of the virtual machine, or a comma-separated list of the virtual machine names.
- Example `FabrikamDevVM,FabrikamTestVM`
- **Checkpoint Name:** For the **Create Checkpoint** action, enter the name of the checkpoint that will be applied to the virtual machines. For the **Delete Checkpoint** or **Restore Checkpoint** action, enter the name of an existing checkpoint.
 - **Description for Checkpoint:** Enter a description for the new checkpoint when creating it.
 - **Select VMs From:** Choose either **Cloud** or **Host** to select the set of virtual machines to which the action will be applied.
 - **Host Name or Cloud Name:** Depending on the previous selection, enter either a cloud name or a host machine name.

Run custom PowerShell scripts for SCVMM

For functionality that is not available through the in-built actions, you can run custom SCVMM PowerShell scripts using the task. The task helps you with setting up the connection with SCVMM using the credentials configured in the service connection, and then runs the script.

- **Display name:** The name for the task as it appears in the task list.
- **SCVMM Service Connection:** Select a SCVMM service connection you already defined, or create a new one.
- **Action:** Select **Run PowerShell Script for SCVMM**.
- **Script Type:** Select either **Script File Path** or **Inline Script**.

- **Script Path:** If you selected **Script File Path**, enter the path of the PowerShell script to execute. It must be a fully-qualified path, or a path relative to the default working directory.
- **Inline Script:** If you selected **Inline Script**, enter the PowerShell script lines to execute.
- **Script Arguments:** Enter any arguments to be passed to the PowerShell script. You can use either ordinal parameters or named parameters.
- **Working folder:** Specify the current working directory for the script when it runs. The default if not provided is the folder containing the script.

Deploying build to virtual machines

Once you have the virtual machines set up, deploying a build to those virtual machines is no different than deploying to any other machine. For instance, you can:

- Use the [PowerShell on Target Machines](#) task to run remote scripts on those machines using Windows Remote Management.
- Use [Deployment groups](#) to run scripts and other tasks on those machines using build and release agent.

See also

- [Create a virtual network isolated environment for build-deploy-test scenarios](#)

VMware deployment

10/15/2018 • 5 minutes to read • [Edit Online](#)

[Azure Pipelines | TFS 2018 | TFS 2017](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

You can automatically provision virtual machines in a VMware environment and deploy to those virtual machines after every successful build. Before this guidance, read the [web quickstart](#).

VMware connection

You need to first configure how Azure Pipelines connects to vCenter. You cannot use Microsoft-hosted agents to run VMware tasks since the vSphere SDK is not installed on these machines. You have to set up a self-hosted agent that can communicate with the vCenter server.

You need to first configure how TFS connects to vCenter. You have to set up a self-hosted agent that can communicate with the vCenter server.

1. Install the VMware vSphere Management SDK to call VMware API functions that access vSphere web services. To install and configure the SDK on the agent machine:

- Download and install the latest version of the Java Runtime Environment from [this location](#).
- Go to [this location](#) and sign in with your existing credentials or register with the website. Then download the **vSphere 6.0 Management SDK**.
- Create a directory for the vSphere Management SDK such as **C:\vSphereSDK**. Do not include spaces in the directory names to avoid issues with some of the batch and script files included in the SDK.
- Unpack the vSphere Management SDK into the new folder you just created.
- Add the full path and name of the precompiled VMware Java SDK file **vim25.jar** to the machine's CLASSPATH environment variable. If you used the path and name **C:\vSphereSDK** for the SDK files, as shown above, the full path will be:

`C:\vSphereSDK\SDK\vsphere-ws\java\JAXWS\lib\vim25.jar`

2. Install the VMware extension from Visual Studio Marketplace into TFS or Azure Pipelines.

- If you are using **Azure Pipelines**, install the extension from [this location](#) in Visual Studio Marketplace.
- If you are using **Team Foundation Server**, download the extension from [this location](#) in Visual Studio Marketplace, upload it to your Team Foundation Server, and install it.

3. Follow these steps to create a vCenter Server service connection in your project:

- Open your Azure Pipelines or TFS project in your web browser. Choose the **Settings** icon in the menu bar and select **Services**.
- In the **Services** tab, choose **New service connection**, and select **VMware vCenter Server**.

- In the **Add new VMware vCenter Server Connection** dialog, enter the values required to connect to the vCenter Server:
 - **Connection Name:** Enter a user-friendly name for the service connection such as **Fabrikam vCenter**.
 - **vCenter Server URL:** Enter the URL of the vCenter server, in the form `https://machine.domain.com/`. Note that only **HTTPS** connections are supported.
 - **Username and Password:** Enter the credentials required to connect to the vCenter Server. Username formats such as **username**, **domain\username**, **machine-name\username**, and **.\\username** are supported. UPN formats such as **username@domain.com** and built-in system accounts such as **NT Authority\System** are not supported.

Managing VM snapshots

Use the **VMware Resource Deployment** task from the VMware extension and configure the properties as follows to take snapshot of virtual machines, or to revert or delete them:

- **VMware Service Connection:** Select the VMware vCenter Server connection you created earlier.
- **Action:** Select one of the actions: **Take Snapshot of Virtual Machines**, **Revert Snapshot of Virtual Machines**, or **Delete Snapshot of Virtual Machines**.
- **Virtual Machine Names:** Enter the names of one or more virtual machines. Separate multiple names with a comma; for example, `VM1,VM2,VM3`
- **Datacenter:** Enter the name of the datacenter where the virtual machines will be created.
- **Snapshot Name:** Enter the name of the snapshot. This snapshot must exist if you use the revert or delete action.
- **Host Name:** Depending on the option you selected for the compute resource type, enter the name of the host, cluster, or resource pool.
- **Datastore:** Enter the name of the datastore that will hold the virtual machines' configuration and disk files.
- **Description:** Optional. Enter a description for the **Take Snapshot of Virtual Machines** action, such as `$(Build.DefinitionName).$(Build.BuildNumber)`. This can be used to track the execution of the build or release that created the snapshot.
- **Skip Certificate Authority Check:** If the vCenter Server's certificate is self-signed, select this option to skip the validation of the certificate by a trusted certificate authority.

To verify if a self-signed certificate is installed on the vCenter Server, open the VMware vSphere Web Client in your browser and check for a certificate error page. The vSphere Web Client URL will be of the form `https://machine.domain/vsphere-client/`. Good practice guidance for vCenter Server certificates can be found in the [VMware Knowledge Base](#) (article 2057223).

Provisioning virtual machines

To configure the **VMware Resource Deployment** task to provision a new virtual machine from a template, use these settings:

- **VMware Service Connection:** Select the VMware vCenter Server connection you created earlier.
- **Action:** `Deploy Virtual Machines using Template`
- **Template:** The name of the template that will be used to create the virtual machines. The template must exist in the location you enter for the **Datacenter** parameter.

- **Virtual Machine Names:** Enter the names of one or more virtual machines. Separate multiple names with a comma; for example, `VM1,VM2,VM3`
- **Datacenter:** Enter the name of the datacenter where the virtual machines will be created.
- **Compute Resource Type:** Select the type of hosting for the virtual machines: `VMware ESXi Host`, `Cluster`, or `Resource Pool`
- **Host Name:** Depending on the option you selected for the compute resource type, enter the name of the host, cluster, or resource pool.
- **Datastore:** Enter the name of the datastore that will hold the virtual machines' configuration and disk files.
- **Description:** Optional. Enter a description to identify the deployment.
- **Skip Certificate Authority Check:** If the vCenter Server's certificate is self-signed, select this option to skip the validation of the certificate by a trusted certificate authority. See the note for the previous step to check for the presence of a self-signed certificate.

Deploying build to virtual machines

Once you have the virtual machines set up, deploying a build to those virtual machines is no different than deploying to any other machine. For instance, you can:

- Use the [PowerShell on Target Machines](#) task to run remote scripts on those machines using Windows Remote Management.
- Use [Deployment groups](#) to run scripts and other tasks on those machines using build and release agent.

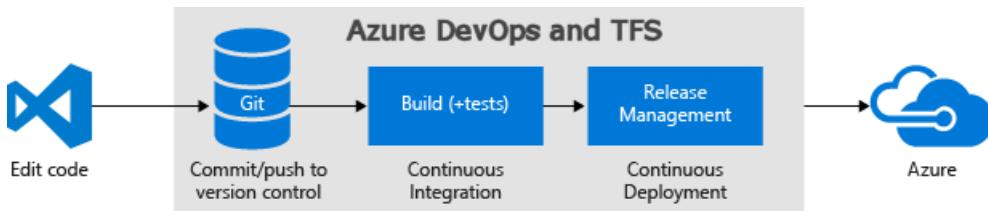
Deploy to an Azure Web App for Containers

11/15/2018 • 5 minutes to read • [Edit Online](#)

Azure Pipelines

We'll show you how to set up continuous deployment of your Docker-enabled app to an Azure Web App using Azure Pipelines.

For example, you can continuously deliver your app to a Windows VM hosted in Azure.



After you commit and push a code change, it is automatically built and then deployed. The results will automatically show up on your site.

Example

If you want some sample code that works with this guidance, import (into Azure DevOps) or fork (into GitHub) this repo:

```
https://github.com/Microsoft/devops-project-samples/tree/master/dotnet/aspnetcore/container/Application
```

Define your CI build pipeline

You'll need a continuous integration (CI) build pipeline that publishes a Docker container image. To set up a CI build pipeline, see:

- [Build and push a Docker image.](#)

Prerequisites

You'll need an Azure subscription. You can get one free through [Visual Studio Dev Essentials](#).

Create an Azure Web App to host a container

1. Sign into Azure at <https://portal.azure.com>.
2. In the Azure Portal, choose **Create a resource**, **Web**, then choose **Web App for Containers**.
3. Enter a name for your new web app, and select or create a new Resource Group. For the **OS**, choose **Linux**.
4. Choose **Configure container** and select **Azure Container Registry**. Use the drop-down lists to select the registry you created earlier, and the Docker image and tag that was generated by the build pipeline.
5. Wait until the new web app has been created. Then you can create a release pipeline as shown in the next section.

The **Docker** tasks you used in the build pipeline when you created the build artifacts push the Docker image back

into your Azure Container Registry. The web app you created here will host an instance of that image and expose it as a website.

Why use a separate release pipeline instead of the automatic deployment feature available in Web App for Containers?

You can configure Web App for Containers to automatically configure deployment as part of the CI/CD pipeline so that the web app is automatically updated when a new image is pushed to the container registry (this feature uses a [webhook](#)). However, by using a separate release pipeline in Azure Pipelines or TFS you gain extra flexibility and traceability. You can:

- Specify an appropriate tag that is used to select the deployment target for multi-stage deployments.
- Use separate container registries for different stages.
- Use parameterized start-up commands to, for example, set the values of variables based on the target stage.
- Avoid using the same tag for all the deployments. The default CD pipeline for Web App for Containers uses the same tag for every deployment. While this may be appropriate for a tag such as **latest**, you can achieve end-to-end traceability from code to deployment by using a build-specific tag for each deployment. For example, the Docker build tasks let you tag your images with the **Build.ID** for each deployment.

Create a release pipeline

1. In **Azure Pipelines**, open the build summary for your build and choose **Release** to start a new release pipeline.

If you have previously created a release pipeline that uses these build artifacts, you will be prompted to create a new release instead. In that case, go to the **Releases** tab page and start a new release pipeline from there by choosing the **+** icon.

2. Select the **Azure App Service Deployment** template and choose **Apply**.
3. Open the **Tasks** tab and select the **Stage 1** item. In the settings panel next to **Parameters**, choose **Unlink all**.

4. Select the **Deploy Azure App Service** task and configure the settings as follows:

- **Version:** Select **4.* (preview)**.
- **Azure Subscription:** Select a connection from the list under **Available Azure Service Connections** or create a more restricted permissions connection to your Azure subscription. If you are using Azure Pipelines and if you see an **Authorize** button next to the input, click on it to authorize Azure Pipelines to connect to your Azure subscription. If you are using TFS or if you do not see the desired Azure subscription in the list of subscriptions, see [Azure Resource Manager service connection](#) to manually set up the connection.
- **App Service type:** Select **Web App for Containers**.
- **App Service name:** Select the web app you created earlier from your subscription. App services based on selected app type will only be listed.

When you select the Docker-enabled app type, the task recognizes that it is a containerized app, and changes the property settings to show the following:

- **Registry or Namespace:** Enter the path to your Azure Container Registry which is a globally unique top-level domain name for your specific registry or namespace. Typically this is *your-registry-name.azurecr.io*
- **Image:** Name of the repository where the container images are stored.

- **Tag:** Tags are optional, it is the mechanism that registries use to give Docker images a version. A fully qualified image name will be of the format: '/'. For example, 'myregistry.azurecr.io/nginx:latest'.
- **Startup command:** Start up command for the container.

5. Save the release pipeline.

Create a release to deploy your app

You're now ready to create a release, which means to run the release pipeline with the artifacts produced by a specific build. This will result in deploying the build:

1. Choose **+ Release** and select **Create a release**.
2. In the **Create a new release** panel, check that the artifact version you want to use is selected and choose **Create**.
3. Choose the release link in the information bar message. For example: "Release **Release-1** has been created".
4. In the pipeline view, choose the status link in the stages of the pipeline to see the logs and agent output.
5. After the release is complete, navigate to your site running in Azure using the Web App URL
`http://{web_app_name}.azurewebsites.net`, and verify its contents.

Next steps

- [Set up multi-stage release](#)

Deploy to a Windows Virtual Machine

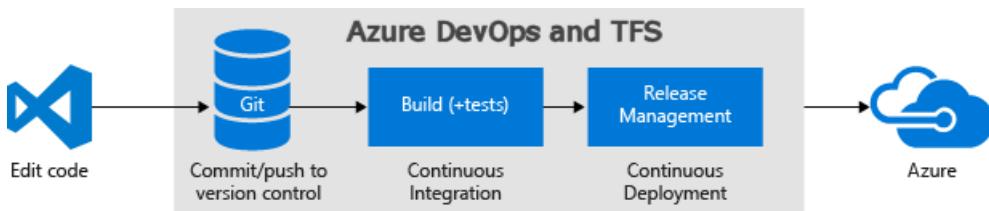
12/3/2018 • 4 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

We'll show you how to set up continuous deployment of your ASP.NET or Node.js app to an IIS web server running on Windows using Azure Pipelines. You can use the steps in this quickstart as long as your continuous integration pipeline publishes a web deployment package.



After you commit and push a code change, it is automatically built and then deployed. The results will automatically show up on your site.

Define your CI build pipeline

You'll need a continuous integration (CI) build pipeline that publishes your web deployment package. To set up a CI build pipeline, see:

- [Build ASP.NET 4 apps](#)
- [Build ASP.NET Core apps](#)
- [Build JavaScript and Node.js apps](#)

Prerequisites

IIS configuration

The configuration varies depending on the type of app you are deploying.

ASP.NET app

On your VM, open an **Administrator: Windows PowerShell** console. Install IIS:

```
# Install IIS
Install-WindowsFeature Web-Server,Web-Asp-Net45,.NET-Framework-Features
```

ASP.NET Core app

Running an ASP.NET Core app on Windows requires some dependencies.

On your VM, open an **Administrator: Windows PowerShell** console. Install IIS and the required .NET features:

```

# Install IIS
Install-WindowsFeature Web-Server,Web-Asp-Net45,NET-Framework-Features

# Install the .NET Core SDK
Invoke-WebRequest https://go.microsoft.com/fwlink/?LinkId=848827 -outfile $env:temp\dotnet-dev-win-x64.1.0.6.exe
Start-Process $env:temp\dotnet-dev-win-x64.1.0.6.exe -ArgumentList '/quiet' -Wait

# Install the .NET Core Windows Server Hosting bundle
Invoke-WebRequest https://go.microsoft.com/fwlink/?LinkId=817246 -outfile
$env:temp\DotNetCore.WindowsHosting.exe
Start-Process $env:temp\DotNetCore.WindowsHosting.exe -ArgumentList '/quiet' -Wait

# Restart the web server so that system PATH updates take effect
net stop was /y
net start w3svc

```

When `net start w3svc` appears, press **Enter** to run it.

Node.js app

Follow the instructions in [this topic](#) to install and configure IISnode on IIS servers.

Create a deployment group

Deployment groups in Azure Pipelines make it easier to organize the servers that you want to use to host your app. A deployment group is a collection of machines with an Azure Pipelines agent on each of them. Each machine interacts with Azure Pipelines to coordinate deployment of your app.

1. Open the Azure Pipelines web portal and choose **Deployment groups**.
2. Click **Add Deployment group** (or **New** if there are already deployment groups in place).
3. Enter a name for the group, such as *myIIS*, and then click **Create**.
4. In the **Register machine** section, make sure that **Windows** is selected, and that **Use a personal access token in the script for authentication** is also selected. Click **Copy script to clipboard**.

The script that you've copied to your clipboard will download and configure an agent on the VM so that it can receive new web deployment packages and apply them to IIS.

5. On your VM, in an **Administrator PowerShell** console, paste and run the script.
6. When you're prompted to configure tags for the agent, press Enter (you don't need any tags).
7. When you're prompted for the user account, press Enter to accept the defaults.

The account under which the agent runs needs **Manage** permissions for the `C:\Windows\system32\inetsrv\` directory. Adding non-admin users to this directory is not recommended. In addition, if you have a custom user identity for the application pools, the identity needs permission to read the crypto-keys. Local service accounts and user accounts must be given read access for this. For more details, see [Keyset does not exist error message](#).

8. When the script is done, it displays the message *Service vstsagent.account.computername started successfully*.
9. On the **Deployment groups** page in Azure Pipelines, open the *myIIS* deployment group. On the **Targets** tab, verify that your VM is listed.

Define your CD release pipeline

Your CD release pipeline picks up the artifacts published by your CI build and then deploys them to your IIS servers.

1. If you haven't already done so, install the [IIS Web App Deployment Using WinRM](#) extension from Marketplace. This extension contains the tasks required for this example.
2. Do one of the following:
 - If you've just completed a CI build then, in the build's **Summary** tab choose **Release**. This creates a new release pipeline that's automatically linked to the build pipeline.
 - Open the **Releases** tab of **Azure Pipelines**, open the + drop-down in the list of release pipelines, and choose **Create release pipeline**.
3. Select the **IIS Website Deployment** template and choose **Apply**.
4. If you created your new release pipeline from a build summary, check that the build pipeline and artifact is shown in the **Artifacts** section on the **Pipeline** tab. If you created a new release pipeline from the **Releases** tab, choose the + **Add** link and select your build artifact.
5. Choose the **Continuous deployment** icon in the **Artifacts** section, check that the continuous deployment trigger is enabled, and add a filter to include the **master** branch.
6. Open the **Tasks** tab and select the **IIS Deployment** job. For the **Deployment Group**, select the deployment group you created earlier (such as *myIIS*).
7. Save the release pipeline.

Create a release to deploy your app

You're now ready to create a release, which means to run the release pipeline with the artifacts produced by a specific build. This will result in deploying the build:

1. Choose + **Release** and select **Create a release**.
2. In the **Create a new release** panel, check that the artifact version you want to use is selected and choose **Create**.
3. Choose the release link in the information bar message. For example: "Release **Release-1** has been created".
4. In the pipeline view, choose the status link in the stages of the pipeline to see the logs and agent output.
5. After the release is complete, navigate to your app and verify its contents.

Next steps

- [Dynamically create and remove a deployment group](#)
- [Apply stage-specific configurations](#)
- [Perform a safe rolling deployment](#)
- [Deploy a database with your app](#)

Artifacts in Azure Pipelines

11/19/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015.3 and newer | TFS 2015 RTM (see Q&A)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

You can publish and consume many different types of packages and artifacts with Azure Pipelines. Your continuous integration (CI)/continuous deployment (CD) pipeline can publish specific package types to their respective package repositories (NuGet, npm, Python, etc.), or you can leverage Build artifacts/Pipeline Artifacts (Preview) to help store build outputs and intermediate files between build steps where they can be added onto, built, tested, or even deployed.

Supported artifact types

Below you will find a table of supported artifact types in Azure Pipelines.

SUPPORTED ARTIFACT TYPES	DESCRIPTION
Build artifacts	Build artifacts are the files that you want your build to produce. Build artifacts can be nearly anything your team needs to test or deploy your app. For example, you've got a .DLL and .EXE executable files and .PDB symbols file of a C# or C++ .NET Windows app.
Pipeline Artifacts (Preview)	You can use Pipelines Artifacts to help store build outputs and move intermediate files between jobs in your pipeline. Pipeline Artifacts are tied to the pipeline that they are created in and can be used within the pipeline and downloaded from the build as long as the build is retained. Pipelines Artifacts are the new generation of Build artifacts, they leverage existing services to dramatically reduce the time it takes to store outputs in your pipelines.
Maven	You can publish Maven artifacts to Azure Artifacts feeds or Maven repositories.
npm	You can publish npm packages to Azure Artifacts or npm registries.
NuGet	You can publish NuGet packages to Azure Artifacts, other NuGet services (like NuGet.org), or internal NuGet repositories.
PyPI	You can publish Python packages to Azure Artifacts or PyPI repositories.

SUPPORTED ARTIFACT TYPES	DESCRIPTION
Symbols	Symbol files contain debugging information for compiled executables. You can publish symbols to symbol servers, which enable debuggers to automatically retrieve the correct symbol files without knowing specific product, package, or build information.
Universal	Universal Packages store one or more files together in a single unit that has a name and version. Unlike Pipeline Artifacts that reside in the pipeline, Universal Packages reside within a feed in Azure Artifacts.

How do I publish and consume artifacts?

Each different kind of artifact has a different way of being published and consumed. Some artifacts are specific to particular development tools such as .NET, Node.js/JavaScript, Python, and Java. Other artifact types offer more generic file storage such as Pipeline Artifacts and Universal Packages. Refer to the table above for specific guidance on each kind of artifact that we support.

Pipeline Artifacts in Azure Pipelines

11/19/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

NOTE

Pipeline Artifacts are the next generation of build artifacts in Azure Pipelines and are current in Preview. For the current artifacts, see [Build artifacts](#).

Pipeline Artifacts help you store build outputs and move intermediate files between stages in your pipelines. Artifacts are the files that you want your build to produce and can be anything your team needs to test or deploy your app.

You'll see the most benefit from Pipeline Artifacts if you have a build that produces very large build outputs. If you're an existing Azure DevOps user, Pipeline Artifacts are automatically enabled for you.

Publish Pipeline Artifact

In build artifacts, it was common to first copy files to `$(Build.ArtifactStagingDirectory)` and then use the Publish Build Artifacts task to publish that directory. With Pipeline Artifacts, we recommend pointing the Publish Pipeline Artifacts tasks directly to the paths to be published. This saves your build the time of creating a copy of the files you wish to publish.

- [YAML](#)
- [Designer](#)

```
steps:  
- task: PublishPipelineArtifact@0  
  inputs:  
    artifactName: 'artifactName'  
    targetPath: 'src/MyWebApp/bin/Release/netcoreapp2.0/linux-x64/publish'
```

Download Pipeline Artifact

If you are using Pipeline Artifacts to deliver artifacts into a release pipeline you don't need to add the task, release pipelines will automatically inject the task into your stages. If you want more control over how files are placed on disk, you can manually add the Download Pipeline Artifact task yourself. You can also use the task to download artifacts from a different pipeline.

- [YAML](#)
- [Designer](#)

```
steps:  
- task: DownloadPipelineArtifact@0  
  inputs:  
    targetPath: $(System.DefaultWorkingDirectory)
```

Artifacts in Azure Pipelines

11/26/2018 • 5 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015.3 and newer | TFS 2015 RTM (see Q&A)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

NOTE

We recommend upgrading from Build artifacts to [Pipeline Artifacts \(Preview\)](#) for drastically faster output storage speeds.

Artifacts are the files that you want your build to produce. Artifacts can be anything your team needs to test or deploy your app.

Azure Pipelines can pick up and use your build artifacts as part of a continuous integration (CI)/continuous deployment (CD) pipeline. In this scenario, you're automatically building a web app with each commit using your CI build. Your CD release pipeline picks up the .ZIP (ASP.NET or Node.js) or .WAR (Java) web deployment file. Your changes are automatically deployed to a test environment in Azure.

You can publish your artifacts to other tasks in your pipeline where they can be added onto, built, tested, or even deploy.

How do I publish artifacts?

Artifacts can be published at any stage of pipeline. What to publish as an artifact and when to publish it can be configured using two different methods, alongside your code with **YAML**, or in the Azure Pipelines UI with the **visual designer**.

Example: Publish a text file as an artifact

- [YAML](#)
- [Designer](#)

```
- powershell: gci env:* | sort-object name | Format-Table -AutoSize | Out-File $env:BUILD_ARTIFACTSTAGINGDIRECTORY/environment-variables.txt

- task: PublishBuildArtifacts@1
  inputs:
    pathToPublish: '$(Build.ArtifactStagingDirectory)'
    artifactName: drop
```

Example: Publish two sets of artifacts

- [YAML](#)
- [Designer](#)

```

- powershell: gci env:* | sort-object name | Format-Table -AutoSize | Out-File
$env:BUILD_ARTIFACTSTAGINGDIRECTORY/environment-variables.txt

- task: PublishBuildArtifacts@1
  inputs:
    pathToPublish: '$(Build.ArtifactStagingDirectory)'
    artifactName: drop1
- task: PublishBuildArtifacts@1
  inputs:
    pathToPublish: '$(Build.ArtifactStagingDirectory)'
    artifactName: drop2

```

Example: Assemble C++ artifacts into one location and publish as an artifact

- [YAML](#)
- [Designer](#)

```

- powershell: gci env:* | sort-object name | Format-Table -AutoSize | Out-File
$env:BUILD_ARTIFACTSTAGINGDIRECTORY/environment-variables.txt

- task: CopyFiles@2
  inputs:
    sourceFolder: '$(Build.SourcesDirectory)'
    contents: '**/$(BuildConfiguration)/**/?(*.exe|*.dll|*.pdb)'
- task: PublishBuildArtifacts@1
  inputs:
    pathToPublish: '$(Build.ArtifactStagingDirectory)'
    artifactName: drop

```

How do I consume artifacts?

Consume artifacts in Release Management

You can download artifacts to Release Management and deploy them to the target of your choice.

In the next job of your build

You can consume an artifact from your build in a subsequent step of the build. This can be useful to build or test your artifact.

Download to debug

You can download an artifact directly from a pipeline for use in debugging.

- [YAML](#)
- [Designer](#)

```

- powershell: gci env:* | sort-object name | Format-Table -AutoSize | Out-File
$env:BUILD_ARTIFACTSTAGINGDIRECTORY/environment-variables.txt

- task: DownloadBuildArtifacts@0
  inputs:
    buildType: 'current'
    downloadType: 'single'
    artifactName: 'drop'
    downloadPath: '$(System.ArtifactsDirectory)'

```

Tips

- **Artifact publish location** argument: **Azure Pipelines/TFS (TFS 2018 RTM and older)**: Artifact type: Server) is the best and simplest choice in most cases. This choice causes the artifacts to be stored in Azure Pipelines or TFS. But if you're using a private Windows agent, you've got the option [drop to a UNC file share](#).
- **Artifact name** argument: Just enter a name that's meaningful to you.
- Use forward slashes in file path arguments so that they work for all agents. Backslashes don't work for macOS and Linux agents.
- On Azure Pipelines and some versions of TFS there are two different [variables](#) that point to the staging directory: `Build.ArtifactStagingDirectory` and `Build.StagingDirectory`. These are interchangeable.
- The directory referenced by `Build.ArtifactStagingDirectory` is cleaned up after each build.
- You can [get build artifacts from the REST API](#).

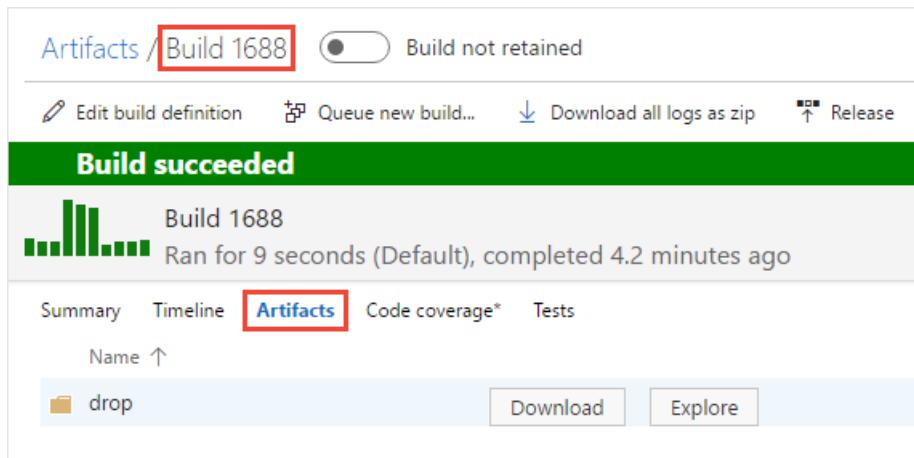
Related tasks for publishing artifacts

Use these tasks to publish artifacts:

-  [Utility: Copy Files](#) By copying files to `$(Build.ArtifactStagingDirectory)` you can publish multiple files of different types from different places specified by your [matching patterns](#).
-  [Utility: Delete Files](#) Handy to prune unnecessary files that you copied to the staging directory.
-  [Utility: Publish Build Artifacts](#)

Explore, download, and deploy your artifacts

When the build is done, if you watched it run, click the name of the completed build and then click the artifacts tab to see your artifact.



The screenshot shows the Azure Pipelines build results for 'Build 1688'. The top navigation bar includes 'Artifacts / Build 1688', a toggle switch for 'Build not retained', and buttons for 'Edit build definition', 'Queue new build...', 'Download all logs as zip', and 'Release'. A green banner at the top says 'Build succeeded'. Below it, the build summary shows 'Build 1688' and 'Ran for 9 seconds (Default), completed 4.2 minutes ago'. The 'Artifacts' tab is highlighted with a red box. Other tabs include 'Summary', 'Timeline', 'Code coverage*', and 'Tests'. At the bottom, there is a table with one row labeled 'drop' and buttons for 'Download' and 'Explore'.

From here you can explore or download the artifacts.

You can also use Azure Pipelines to deploy your app using the artifacts that you've published. See [Artifacts in Azure Pipelines releases](#).

Publish from TFS to UNC file share

If you're using a private Windows agent, you can set the **artifact publish location** option (**TFS 2018 RTM and older**: artifact type) to publish your files to a UNC **file share**.

NOTE

Use a Windows build agent. This option doesn't work for macOS and Linux agents.

Choose file share to copy the artifact to a file share. Some common reasons to do this:

- The size of your drop is large and consumes too much time and bandwidth to copy.
- You need to run some custom scripts or other tools against the artifact.

If you use a file share, specify the UNC file path to the folder. You can control how the folder is created for each build using [variables](#). For example `\my\share\$(Build.DefinitionName)\$(Build.BuildNumber)`.

Publish artifacts from TFS 2015 RTM

If you're using TFS 2015 RTM, then the steps in the above examples are not available. Instead, you copy and publish your artifacts using a single task: [Build: Publish Build Artifacts](#).

Set up Azure Pipelines and Maven

11/19/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

This guide covers the basics of using Azure Pipelines to work with Maven artifacts in Package Management feeds.

This walkthrough assumes that you've already added the correct build service identity to your feed.

1. Create a new build pipeline and select the **Maven** template.
2. Fill in the path to your `pom.xml`, configure the Maven goal you'd like for your build. Authentication to your Azure Artifacts feed should happen automatically.

Publish npm packages

11/19/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

You can publish npm packages produced by your build to:

- Azure Artifacts or the TFS Package Management service
- Other registries such as <https://registry.npmjs.org/>

Before you read this topic, you should understand the kind of build pipeline you're creating:[designer](#) or [YAML](#).

- [YAML](#)
- [Designer](#)

To publish to an Azure Artifacts feed, set the **Project Collection Build Service** identity to be a **Contributor** on the feed. To learn more about permissions to Package Management feeds, see [Secure and share packages using feed permissions](#).

To publish to a external npm registry, you must first create a service connection to point to that feed. You can do this by going to **Project settings**, then choosing **Services**, and then creating a **New service connection**. Select the **npm** option for the service connection. Fill in registry URL and the credentials to connect to the registry.

To publish a package to a npm registry, add the following snippet to your azure-pipelines.yml file.

```
- task: Npm@1
  inputs:
    command: publish
    publishEndpoint: '<copy and paste the name of the service connection here>'
```

For a list of other options, see the [npm task](#).

YAML is not supported in TFS.

NOTE

Build does not support using the `publishConfig` property to specify the `registry` to which you're publishing. Ensure your working folder has a `.npmrc` file with a `registry=` line, as detailed in the Connect to feed screen in your feed.

Q&A

Where can I learn about the Azure Pipelines and TFS Package management service?

[Package Management service](#)

Publish to NuGet feeds

11/29/2018 • 7 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

You can publish NuGet packages from your build to NuGet feeds. You can publish these packages to

- Azure Artifacts or the TFS Package Management service
- Other NuGet services such as NuGet.org
- Your internal NuGet repository

Before you read this topic, you should understand the kind of build pipeline you're creating: [designer](#) or [YAML](#).

Create a NuGet package

There are a variety of ways to create NuGet packages during a build. If you're already using MSBuild or some other task to create your packages, skip this section to go ahead and [publish your packages](#). Otherwise, add a **NuGet** task:

- [YAML](#)
- [Designer](#)

To create a package, add the following snippet to your azure-pipelines.yml file.

```
- task: NuGetCommand@2
  inputs:
    command: pack
    packagesToPack: '**/*.csproj'
```

The NuGet task supports a number of options. Some of the key ones are described below. The rest are described in the [task documentation](#).

- **packagesToPack:** The path to the files that describe the package you want to create. If you don't have these, see the [NuGet documentation](#) to get started.
- **configuration:** The default is `$(BuildConfiguration)` unless you wish to always build either `Debug` or `Release` packages, or unless you have a custom build configuration.
- **packDestination:** The default is `$(Build.ArtifactStagingDirectory)`. If you set this, make a note of the location so you can use it in the [publish task](#).

YAML is not supported in TFS.

Package versioning

In NuGet, a particular package is identified by its name and version number. A recommended approach to versioning packages is to use semantic versioning. Semantic version numbers have 3 numeric components, `Major.Minor.Patch`. When you fix a bug, you increment the patch (`1.0.0` ? `1.0.1`). When you release a new

backwards-compatible feature, you increment minor and reset patch to 0 (`1.4.17` ? `1.5.0`). When you make a backwards-incompatible change, you increment major and reset minor and patch to 0 (`2.6.5` ? `3.0.0`).

In addition to `Major.Minor.Patch`, semantic versioning provides for a prerelease label. Prerelease labels are a “-” followed by whatever letters and numbers you want. Version `1.0.0-alpha`, `1.0.0-beta`, and `1.0.0-foo12345` are all prerelease versions of `1.0.0`. Even better, semantic versioning specifies that when you sort by version number, those prerelease versions fit exactly where you’d expect: `0.99.999` < `1.0.0-alpha` < `1.0.0` < `1.0.1-beta`.

When you create a package in CI, you can use semantic versioning with prerelease labels. The **NuGet** task can be used for this purpose, and supports the following formats:

- Use the same versioning scheme for your builds and packages, provided that scheme has at least three parts separated by periods. The following build pipeline formats are examples of versioning schemes that are compatible with NuGet.
 - `$(Major).$(Minor).$(rev:.r)`, where `Major` and `Minor` are two variables defined in the build pipeline. This will automatically increment the build number and the package version with a new patch number keeping the major and minor versions constant, until you change them manually in the build pipeline.
 - `$(Major).$(Minor).$(Patch).$(date:yyyyMMdd)`, where `Major`, `Minor`, and `Patch` are variables defined in the build pipeline. This will create a new prerelease label for the build and package while keeping the major, minor, and patch versions constant.
- Use a version that is different from the build number. You can customize the major, minor, and patch versions for your packages in the NuGet task, and let the task generate a unique prerelease label based on date and time.
- Use a script in your build pipeline to generate the version.
- [YAML](#)
- [Designer](#)

This example shows how to use the date and time as the prerelease label.

```
variables:  
  Major: '1'  
  Minor: '0'  
  Patch: '0'  
  
steps:  
- task: NuGetCommand@2  
  inputs:  
    command: pack  
    versioningScheme: byPrereleaseNumber  
    majorVersion: '$(Major)'  
    minorVersion: '$(Minor)'  
    patchVersion: '$(Patch)'
```

For a list of other possible values for `versioningScheme`, see the [NuGet task](#).

YAML is not supported in TFS.

While semantic versioning with prerelease labels is a good solution for packages produced in CI builds, including a prerelease label is not ideal when you want to release a package to your users. The challenge is that packages once produced are [immutable](#) and so cannot be updated or replaced. When you’re producing a package in build, you can’t know whether it will be the version that you aim to release to your users or just a step along the way towards that release. While none of the following solutions are ideal, you can use one of these depending on your preference:

- Once you validate a package and decide to release it, produce another package without the prerelease label and publish it. The drawback of this approach is that you have to validate the new package again, and it may uncover new issues.
- Publish only packages that you wish to release. In this case, you will not use a prerelease label for every build. Instead, you will reuse the same package version for all packages. Since you do not publish packages from every build, you do not cause a conflict.

Publish your packages

In the previous section, you learned how to create a package with every build. When you are ready to share the changes to your package with your users, you can publish it.

- [YAML](#)
- [Designer](#)

To publish to an Azure Artifacts feed, set the **Project Collection Build Service** identity to be a **Contributor** on the feed. To learn more about permissions to Package Management feeds, see [Secure and share packages using feed permissions](#). Add the following snippet to your `azure-pipelines.yml` file.

```
steps:
- task: NuGetCommand@2
  displayName: 'NuGet push'
  inputs:
    command: push
    publishVstsFeed: 'af432f1b-f4b2-4a4f-bc9c-fef924f8f823'
    allowPackageConflicts: true
```

To publish to a external NuGet feed, you must first create a service connection to point to that feed. You can do this by going to **Project settings**, then choosing **Service connections**, and then creating a **New service connection**. Select the **NuGet** option for the service connection. Fill in feed URL and the API key or token to connect to the feed.

To publish a package to a NuGet feed, add the following snippet to your `azure-pipelines.yml` file.

```
- task: NuGetCommand@2
  inputs:
    command: push
    nuGetFeedType: external
    publishFeedCredentials: '<Name of the NuGet service connection>'
    versioningScheme: byEnvVar
    versionEnvVar: Version
```

YAML is not supported in TFS.

Publish symbols for your packages

When you push packages to a Package Management feed, you can also [publish symbols](#).

Q&A

Where can I learn more about Azure Artifacts and the TFS Package Management service

[Package Management in Azure Artifacts and TFS](#)

Publish Python packages in Azure Pipelines

11/29/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

NOTE

Python package publishing in Azure Pipelines is currently in public preview.

You can publish Python packages produced by your build to:

- Azure Artifacts
- Other repositories such as <https://pypi.org/>

Before you read this topic, you should understand the kind of build pipeline you're creating:[visual designer](#) or [YAML](#).

To publish Python packages produced by your build, you will use `twine`, a widely-used utility for publishing Python packages. This guide covers how to do the following in your pipeline:

1. Authenticate `twine` with your Azure Artifacts feeds
2. Use a custom task that invokes `twine` to publish your Python packages

Authenticate Azure Artifacts with `twine`

To use `twine` to publish Python packages, you first need to set up authentication. The [Python Twine Authenticate](#) task stores your authentication credentials in an environment variable (`PYPIRC_PATH`). `twine` will reference this variable later.

- [YAML](#)
- [Designer](#)

To authenticate with `twine`, add the following snippet to your `azure-pipelines.yml` file.

```
- task: TwineAuthenticate@0
  inputs:
    artifactFeeds: 'feed_name1, feed_name2'
    externalFeeds: 'feed_name1, feed_name2'
```

- **artifactFeeds**: the name of one or more Azure Artifacts feeds within your organization
- **externalFeeds**: the name of one or more [external connection endpoints](#), including PyPI or feeds in other Azure DevOps organizations

Use a custom `twine` task to publish

After you've set up authentication with the snippet above, you can use `twine` to publish your Python packages. Below is an example using a custom command line task.

- [YAML](#)
- [Designer](#)

```
- script: 'twine -r {feedName/EndpointName} --config-file ${PYPIRC_PATH} {package path to publish}'
```

Check out the [script YAML task reference](#) for the schema for this command.

Tips and FAQs

1. The authentication credentials written into the `PYPIRC_PATH` environment variable supersede those in your ini/conf files.
2. If you add multiple Python Twine Authenticate tasks at different times in your pipeline steps, each additional build task execution will extend (not override) the existing `PYPIRC_PATH` environment variable.
3. Lastly, we strongly recommend **NOT** checking into source control any credentials or tokens.

Publish symbols for debugging

11/28/2018 • 4 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

NOTE

A symbol server is available with **Azure Artifacts** in Azure DevOps Services and works best with **Visual Studio 2017 Update 4 or later**. **Team Foundation Server** users and users without the **Azure Artifacts** extension can publish symbols to a file share using a [build task](#).

Symbol servers enable debuggers to automatically retrieve the correct symbol files without knowing product names, build numbers or package names. To learn more about symbols, read the [concept page](#); to consume symbols, see [this page for Visual Studio](#) or [this page for WinDbg](#).

Publish symbols

In order to publish symbols to the Package Management symbol server in Azure Pipelines, include the [Index Sources and Publish Symbols](#) task in your build pipeline. Configure the task as follows:

- For **Version**, select 2.*.
- For **Symbol Server Type**, select **Azure Pipelines**.
- Use the **Path to symbols folder** argument to specify the root directory that contains the .pdb files to be published.
- Use the **Search pattern** argument to specify search criteria to find the .pdb files in the folder that you specify in **Path to symbols folder**. You can use a single-folder wildcard (`*`) and recursive wildcards (`**`). For example, `**\bin***.pdb` searches for all .pdb files in all subdirectories named `bin`.

Publish symbols for NuGet packages

To publish symbols for NuGet packages, include the above task in the build pipeline that produces the NuGet packages. Then the symbols will be available to all users in the Azure DevOps organization.

Publish symbols to a file share

You can also publish symbols to a file share using the [Index Sources and Publish Symbols](#) task. When you use this method, the task will copy the PDB files over and put them into a specific layout. When Visual Studio is pointed to the UNC share, it can find the symbols related to the binaries that are currently loaded.

Add the task to your build pipeline and configure as follows:

- For **Version**, select 2.*.
- For **Symbol Server Type**, select **File share**.
 - When you select **File share** as your **Symbol Server Type**, you get the option to *Compress Symbols*, this option will compress your symbols to save space.

- Use the **Path to symbols folder** argument to specify the root directory that contains the .pdb files to be published.
- Use the **Search pattern** argument to specify search criteria to find the .pdb files in the folder that you specify in **Path to symbols folder**. You can use a single-folder wildcard (`*`) and recursive wildcards (`**`). For example, `**\bin***.pdb` searches for all .pdb files in all subdirectories named `bin`.

Portable PDBs

If you're using [Portable PDBs](#), you don't need to use the **Index Sources and Publish Symbols** task. For Portable PDBs, indexing is done by the build. This is a design feature of Portable PDBs and .NET.

Use indexed symbols to debug your app

You can use your indexed symbols to debug an app on a different machine from where the sources were built.

Enable your dev machine

In Visual Studio you may need to enable the following two options:

- Debug -> Options -> Debugging -> General
 - -> Enable source server support
 - -> Allow source server for partial trust assemblies (Managed only)

Advanced usage: overriding at debug time

The mapping information injected into the PDB files contains variables that can be overridden at debugging time. Overriding the variables may be required if the collection URL has changed. When overriding the mapping information, the goals are to construct:

- A command (SRCSRVCMD) that the debugger can use to retrieve the source file from the server.
- A location (SRCRVTRG) where the debugger can find the retrieved source file.

The mapping information may look something like the following:

```

SRCSRV: variables -----
TFS_EXTRACT_TARGET=%target%\%var5%\fnvar(%var6%)%fnbks1(%var7%)
TFS_EXTRACT_CMD=tf.exe git view /collection:%fnvar(%var2%) /teamproject:"%fnvar(%var3%)"
/repository:"%fnvar(%var4%)" /commitId:%fnvar(%var5%) /path:"%var7%" /output:%SRCRVTRG% %fnvar(%var8%)
TFS_COLLECTION=http://SERVER:8080/tfs/DefaultCollection
TFS_TEAM_PROJECT=93fc2e4d-0f0f-4e40-9825-01326191395d
TFS_REPO=647ed0e6-43d2-4e3d-b8bf-2885476e9c44
TFS_COMMIT=3a9910862e22f442cd56ff280b43dd544d1ee8c9
TFS_SHORT_COMMIT=3a991086
TFS_APPLY_FILTERS=/applyfilters
SRCRVVERCTRL=git
SRCRVERRDESC=access
SRCRVERRVAR=var2
SRCRVTRG=%TFS_EXTRACT_TARGET%
SRCRVCMD=%TFS_EXTRACT_CMD%
SRCSRV: source files -----
C:\BuildAgent\_work\1\src\MyApp\Program.cs*TFS_COLLECTION*TFS_TEAM_PROJECT*TFS_REPO*TFS_COMMIT*TFS_SHORT_COMMIT*/MyApp/Program.cs*TFS_APPLY_FILTERS
C:\BuildAgent\_work\1\src\MyApp\SomeHelper.cs*TFS_COLLECTION*TFS_TEAM_PROJECT*TFS_REPO*TFS_COMMIT*TFS_SHORT_COMMIT*/MyApp/SomeHelper.cs*TFS_APPLY_FILTERS

```

The above example contains two sections: 1) the variables section and 2) the source files section. The information in the variables section is what can be overridden. The variables can leverage other variables, and can leverage information from the source files section.

To override one or more of the variables while debugging with Visual Studio, create an ini file

`%LOCALAPPDATA%\SourceServer\srcsrv.ini`. Set the content of the INI file to override the variables. For example:

```
[variables]
TFS_COLLECTION=http://DIFFERENT_SERVER:8080/tfs/DifferentCollection
```

Q&A

Q: What's the retention policy for the symbols stored in the Azure Pipelines symbol server?

A: Symbols will have the same retention as the build. When you delete a build, you also delete the symbols produced by that build.

Q: Can I use source indexing on a portable PDB created from a .NET Core assembly?

A: No, source indexing is currently not enabled for Portable PDBs as SourceLink doesn't support authenticated source repositories. The workaround at the moment is to configure the build to generate full PDBs.

Q: Is this available in TFS?

A: In TFS, you can bring your own file share and set it up as a symbol server as described in [this blog](#).

Publish and download Universal Packages in Azure Pipelines

11/19/2018 • 4 minutes to read • [Edit Online](#)

Azure Pipelines

NOTE

Universal Packages are currently in public preview.

When you want to publish a set of related files from a pipeline as a single package, you can use [Universal Packages](#) hosted in Azure Artifacts feeds.

Before you read this topic, you should understand the kind of build pipeline you're creating: [designer](#) or [YAML](#).

Prepare your Universal Package

[Universal Packages](#) are created from a directory of files. By default, the Universal Packages task will publish all files in `$(Build.ArtifactStagingDirectory)`. To prepare your Universal Package for publishing, either configure preceding tasks to place output files in that directory, or use the [Copy Files utility task](#) to assemble the files you want to publish.

Publish your packages

- [YAML](#)
- [Designer](#)

To publish a Universal Package to your feed, add the following snippet to your `azure-pipelines.yml` file.

```
- task: UniversalPackages@0
  displayName: Universal Publish
  inputs:
    command: publish
    publishDirectory: '$(Build.ArtifactStagingDirectory)'
    vstsFeedPublish: '<Feed name>'
    vstsFeedPackagePublish: '<Package name>'
    packagePublishDescription: '<Package description>'
```

To publish to an Azure Artifacts feed, set the **Project Collection Build Service** identity to be a **Contributor** on the feed. To learn more about permissions to Package Management feeds, see [Secure and share packages using feed permissions](#).

To publish to a external Universal Packages feed, you must first create a [service connection](#) to point to that feed. You can do this by going to **Project settings**, then choosing **Service connections**, and then creating a **New Service Connection**. Select the **Team Foundation Server/Team Services** option for the service connection. Fill in the feed URL and a [Personal Access Token](#)) to connect to the feed.

Package versioning

In Universal Packages, a particular package is identified by its name and version number. Currently, Universal

Packages require **semantic versioning**. Semantic version numbers have 3 numeric components, `Major.Minor.Patch`. When you fix a bug, you increment the patch (`1.0.0` → `1.0.1`). When you release a new backwards-compatible feature, you increment minor and reset patch to 0 (`1.4.17` → `1.5.0`). When you make a backwards-incompatible change, you increment major and reset minor and patch to 0 (`2.6.5` → `3.0.0`).

The Universal Packages task will automatically select the next major, minor, or patch version for you when you publish a new package - just set the appropriate radio button

- [YAML](#)
- [Designer](#)

In the **Universal Packages** snippet you added above, add the `versionOption` key with the `major`, `minor`, `patch`, or `custom` value. If you enter the `custom` value, you must also provide the `versionPublish` key.

```
- task: UniversalPackages@0
  displayName: Universal Publish
  inputs:
    command: publish
    publishDirectory: '$(Build.ArtifactStagingDirectory)'
    vstsFeedPublish: '<Feed GUID>'
    vstsFeedPackagePublish: '<Package name>'
    versionOption: custom
    versionPublish: <Package version>
    packagePublishDescription: '<Package description>'
```

Download Universal Package

You can also download a Universal Package from your pipeline.

- [YAML](#)
- [Designer](#)

To download a Universal Package from a feed in your organization, use the following snippet:

```
steps:
- task: UniversalPackages@0
  displayName: 'Universal download'
  inputs:
    command: download
    vstsFeed: 'fabrikamFeed'
    vstsFeedPackage: 'fabrikam-package'
    vstsPackageVersion: 1.0.0
```

ARGUMENT	DESCRIPTION
vstsFeed	Feed that the package is to be downloaded from.
vstsFeedPackage	Name of the package to be downloaded.
vstsPackageVersion	Version of the package to be downloaded.

To download a Universal Package from an external source, use the following snippet:

```

steps:
- task: UniversalPackages@0
  displayName: 'Universal download'
  inputs:
    command: download
    feedsToUse: external
    externalFeedCredentials: MSENG2
    feedDownloadExternal: `fabrikamFeedExternal`
    packageDownloadExternal: `fabrikam-package`
    versionDownloadExternal: 1.0.0

```

ARGUMENT	DESCRIPTION
feedsToUse	Value should be <code>external</code> when downloading from an external source.
externalFeedCredentials	The name of a service connection to another Azure DevOps organization or server. See service connections .
feedDownloadExternal	Feed that the package is to be downloaded from.
packageDownloadExternal	Name of the package to be downloaded.
versionDownloadExternal	Version of the package to be downloaded.
CONTROL OPTIONS	

Q&A

Where can I learn more about Azure Artifacts and the TFS Package Management service

[Package Management in Azure Artifacts and TFS](#)

Build and release agents

12/3/2018 • 12 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

To build your code or deploy your software you need at least one agent. As you add more code and people, you'll eventually need more.

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

Microsoft-hosted agents

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

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

TIP

You can try a Microsoft-hosted agent for no charge. If your build or release doesn't succeed, the issues will be reported in the logs.

[Learn more about Microsoft-hosted agents.](#)

Self-hosted agents

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

TIP

Before you install a self-hosted agent you might want to see if a Microsoft-hosted agent pool will work for you. In many cases this is the simplest way to get going. [Give it a try](#).

You can install the agent on Linux, macOS, or Windows machines. You can also install an agent on a Linux Docker container. See the following topics for additional information on installing a self-hosted agent:

- [macOS agent](#)

- [Red Hat agent](#)
- [Ubuntu 14.04 agent](#)
- [Ubuntu 16.04 agent](#)
- [Windows agent v2](#)

- [macOS agent](#)
- [Red Hat agent](#)
- [Ubuntu 14.04 agent](#)
- [Ubuntu 16.04 agent](#)
- [Windows agent v1](#)

After you've installed the agent on a machine, you can install any other software on that machine as required by your build or deployment jobs.

Parallel jobs

You might need more parallel jobs to use multiple Microsoft-hosted or self-hosted agents at the same time:

- [Parallel jobs in Azure Pipelines](#)

Parallel jobs

You might need more parallel jobs to use multiple agents at the same time:

- [Parallel jobs in TFS](#)

Capabilities

Every agent has a set of capabilities that indicate what it can do. Capabilities are name-value pairs that are either automatically discovered by the agent software, in which case they are called **system capabilities**, or those that you define, in which case they are called **user capabilities**.

The agent software automatically determines various system capabilities such as the name of the machine, type of operating system, and versions of certain software installed on the machine. Also, environment variables defined in the machine automatically appear in the list of system capabilities.

When you author a build or release pipeline, or when you queue a build or deployment, you specify certain **demands** of the agent. The system sends the job only to agents that have capabilities matching the demands specified in the pipeline. As a result, agent capabilities allow you to direct builds and deployments to specific agents.

You can view the system capabilities of an agent, and manage its user capabilities by navigating to **Agent pools** and selecting the **Capabilities** tab for the desired agent.

- Azure Pipelines: https://dev.azure.com/{your_organization}/_admin/_AgentPool
- TFS 2018: https://{your_server}/DefaultCollection/_admin/_AgentPool
- TFS 2017: https://{your_server}/tfs/DefaultCollection/_admin/_AgentPool
- TFS 2015: http://{your_server}:8080/tfs/_admin/_AgentPool

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

TIP

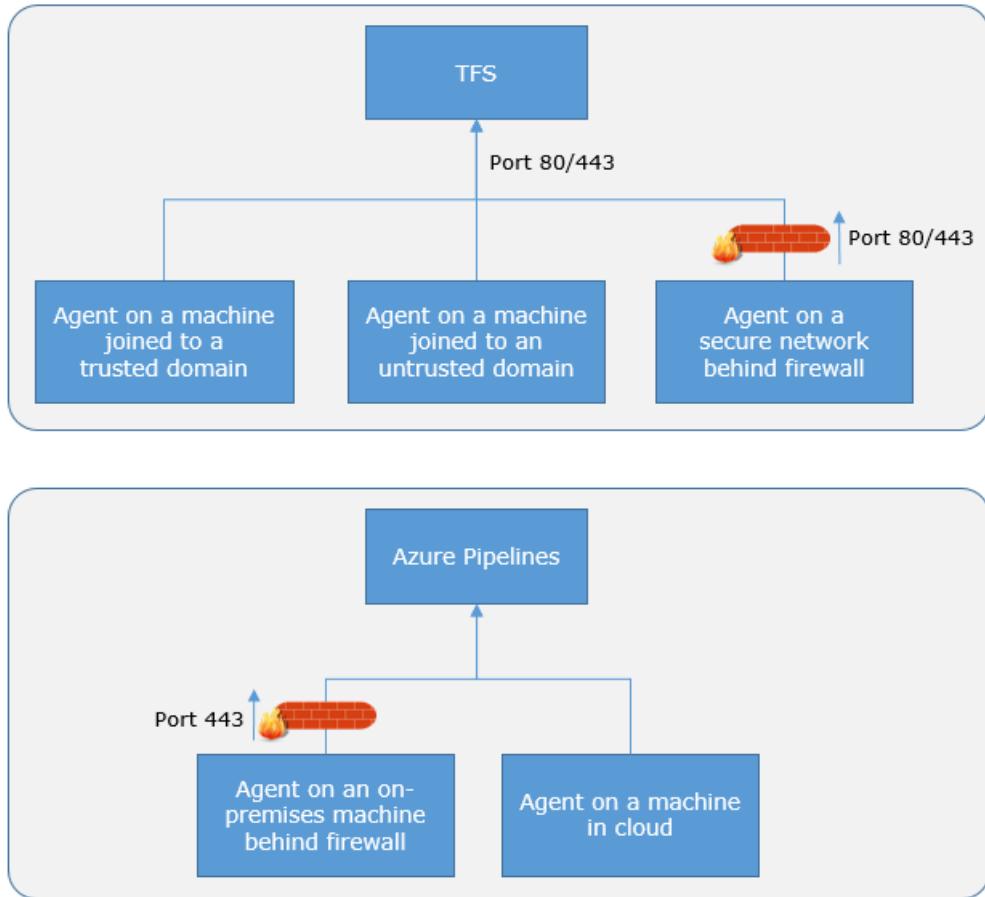
After you install new software on a agent, you must restart the agent for the new capability to show up.

Communication

Communication with Azure Pipelines

Communication with TFS

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



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

1. The user registers an agent with Azure Pipelines or TFS by adding it to an [agent pool](#). You need to be an [agent pool administrator](#) to register an agent in that agent pool. The identity of agent pool administrator is needed only at the time of registration and is not persisted on the agent, nor is used in any further communication between the agent and Azure Pipelines or TFS. Once the registration is complete, the agent downloads a *listener OAuth token* and uses it to listen to the job queue.
2. Periodically, the agent checks to see if a new job request has been posted for it in the job queue in Azure Pipelines/TFS. When a job is available, the agent downloads the job as well as a *job-specific OAuth token*. This token is generated by Azure Pipelines/TFS for the scoped identity [specified in the pipeline](#). That token is short lived and is used by the agent to access resources (e.g., source code) or modify resources (e.g., upload test results) on Azure Pipelines or TFS within that job.
3. Once the job is completed, the agent discards the job-specific OAuth token and goes back to checking if there is a new job request using the listener OAuth token.

The payload of the messages exchanged between the agent and Azure Pipelines/TFS are secured using asymmetric encryption. Each agent has a public-private key pair, and the public key is exchanged with the server during registration. The server uses the public key to encrypt the payload of the job before sending it to the agent. The agent decrypts the job content using its private key. This is how secrets stored in build pipelines, release pipelines, or variable groups are secured as they are exchanged with the agent.

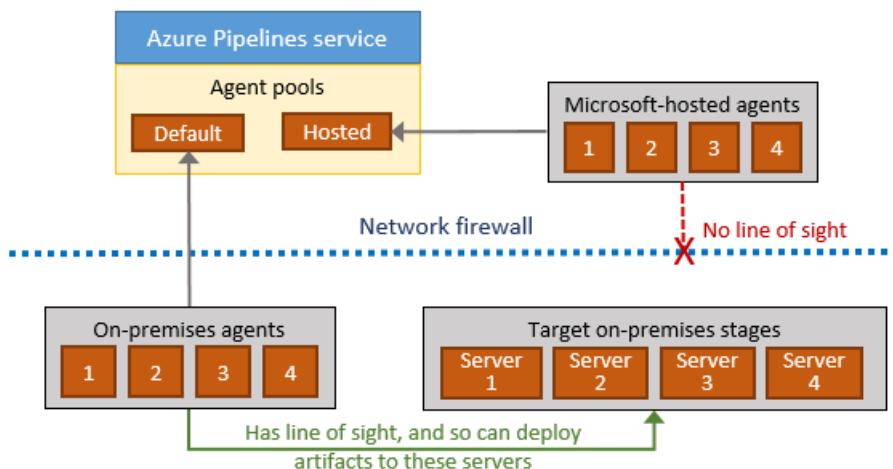
Here is a common communication pattern between the agent and TFS.

- An agent pool administrator joins the agent to an agent pool, and the credentials of the service account (for Windows) or the saved user name and password (for Linux and macOS) are used to initiate communication with TFS. The agent uses these credentials to listen to the job queue.
- The agent does not use asymmetric key encryption while communicating with the server. However, you can [use HTTPS to secure the communication](#) between the agent and TFS.

Communication to deploy to target servers

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

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



Authentication

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

Personal Access Token (PAT):

[Generate](#) and use a PAT to connect an agent with Azure Pipelines or TFS 2017 and newer. PAT is the only scheme that works with Azure Pipelines. Also, as explained above, this PAT is used only at the time of registering the agent, and not for subsequent communication.

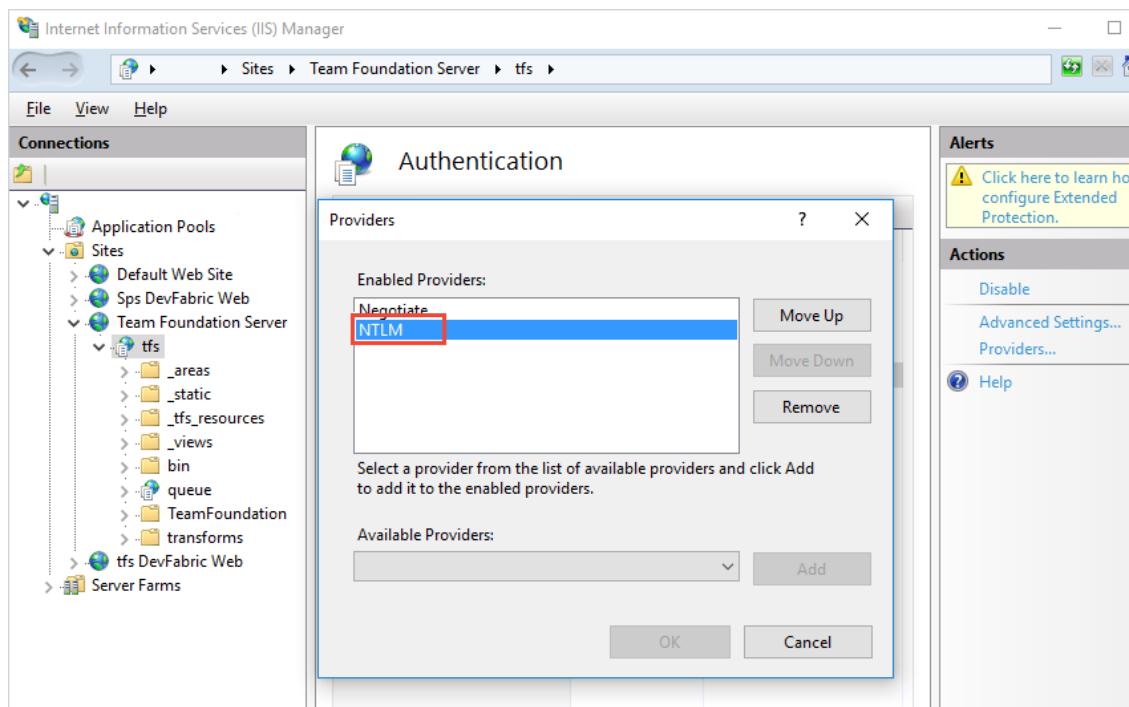
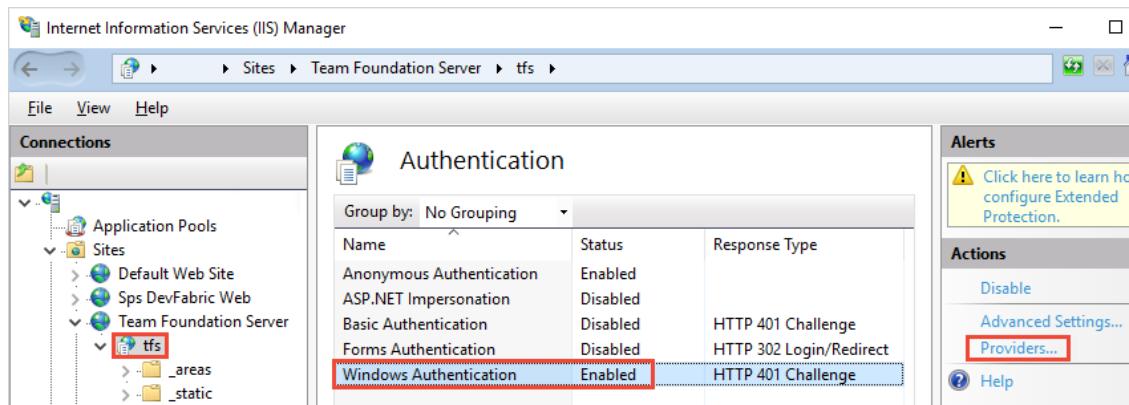
To use a PAT with TFS, your server must be configured with HTTPS. See [Web site settings and security](#).

Integrated

Connect a Windows agent to TFS using the credentials of the signed-in user through a Windows authentication scheme such as NTLM or Kerberos.

To use this method of authentication, you must first configure your TFS server.

1. Sign into the machine where you are running TFS.
2. Start Internet Information Services (IIS) Manager. Select your TFS site and make sure Windows Authentication is enabled with a valid provider such as NTLM or Kerberos.



Negotiate

Connect to TFS as a user other than the signed-in user through a Windows authentication scheme such as NTLM or Kerberos.

To use this method of authentication, you must first configure your TFS server.

1. Log on to the machine where you are running TFS.
2. Start Internet Information Services (IIS) Manager. Select your TFS site and make sure Windows Authentication is enabled with the Negotiate provider and with another method such as NTLM or Kerberos.

The top screenshot shows the 'Authentication' configuration page. The 'Windows Authentication' provider is selected and highlighted with a red box. The bottom screenshot shows the 'Providers' configuration dialog box, where 'Negotiate' and 'NTLM' are listed under 'Enabled Providers'.

Name	Status	Response Type
Anonymous Authentication	Enabled	
ASP.NET Impersonation	Disabled	
Basic Authentication	Disabled	HTTP 401 Challenge
Forms Authentication	Disabled	HTTP 302 Login/Redirect
Windows Authentication	Enabled	HTTP 401 Challenge

Alternate

Connect to TFS using Basic authentication. To use this method you must first [configure HTTPS on TFS](#).

To use this method of authentication, you must configure your TFS server as follows:

1. Log on to the machine where you are running TFS.
2. Configure basic authentication. See [Using tfx against Team Foundation Server 2015 using Basic Authentication](#).

Interactive vs. service

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

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

the agent account.

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

1. **As a service.** You can leverage the service manager of the operating system to manage the lifecycle of the agent. In addition, the experience for auto-upgrading the agent is better when it is run as a service.
2. **As an interactive process with auto-logon enabled.** In some cases, you might need to run the agent interactively for production use - such as to run UI tests. When the agent is configured to run in this mode, the screen saver is also disabled. Some domain policies may prevent you from enabling auto-logon or disabling the screen saver. In such cases, you may need to seek an exemption from the domain policy, or run the agent on a workgroup computer where the domain policies do not apply.

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

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

Agent version and upgrades

We update the agent software every few weeks in Azure Pipelines, and with every update in TFS. We indicate the agent version in the format `{major}.{minor}`. For instance, if the agent version is `2.1`, then the major version is 2 and the minor version is 1. When a newer version of the agent is only different in minor version, it is automatically upgraded by Azure Pipelines or TFS. This upgrade happens when one of the tasks requires a newer version of the agent.

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

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

- Azure Pipelines: https://dev.azure.com/{your_organization}/_admin/_AgentPool
- TFS 2018: https://{your_server}/DefaultCollection/_admin/_AgentPool
- TFS 2017: https://{your_server}/tfs/DefaultCollection/_admin/_AgentPool
- TFS 2015: http://{your_server}:8080/tfs/_admin/_AgentPool

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

Q & A

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

In many cases, yes. Specifically:

- If you use a self-hosted agent you can run incremental builds. For example, you define a CI build pipeline that does not clean the repo and does not perform a clean build, your builds will typically run faster. When you use a Microsoft-hosted agent, you don't get these benefits because the agent is destroyed after the build or release pipeline is completed.
- A Microsoft-hosted agent can take longer to start your build. While it often takes just a few seconds for your job to be assigned to a Microsoft-hosted agent, it can sometimes take several minutes for an agent to be allocated depending on the load on our system.

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

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

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

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

Agent pools

12/3/2018 • 8 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

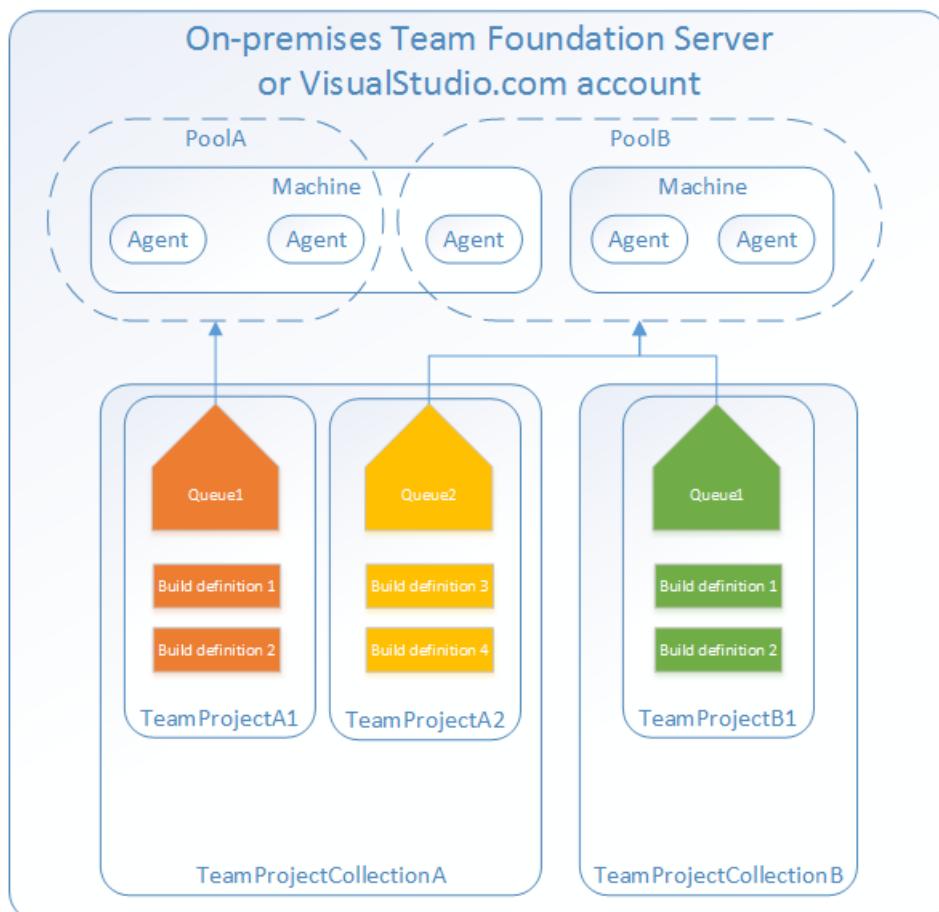
Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Instead of managing each [agent](#) individually, you organize agents into **agent pools**. An agent pool defines the sharing boundary for all agents in that pool. In TFS, pools are scoped across all of Team Foundation Server (TFS); so you can share an agent pool across project collections and projects. In Azure Pipelines, agent pools are scoped to the Azure DevOps organization; so you can share an agent pool across projects.

A **project agent pool** provides access to an **organization agent pool**. When you create a build or release pipeline, you specify which pool it uses. Pools are scoped to your project in TFS 2017 and newer and in Azure Pipelines, so you can only use them across build and release pipelines within a project.

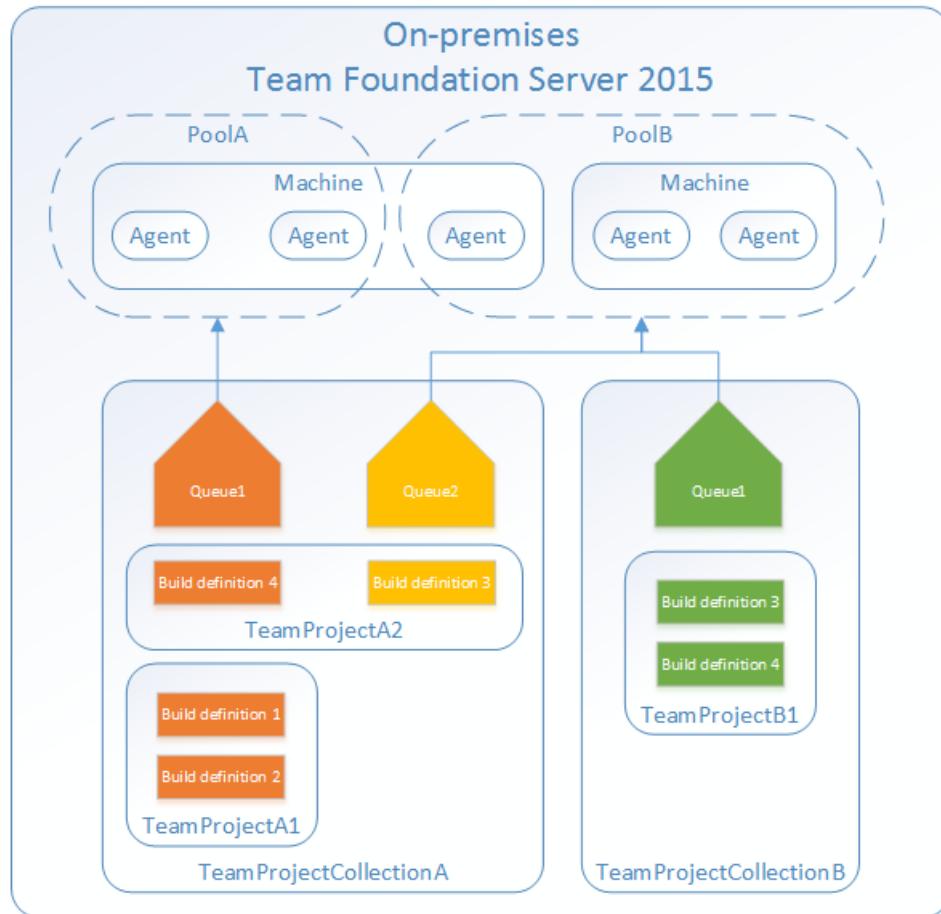
To share an agent pool with multiple projects, in each of those projects, you create a project agent pool pointing to an organization agent pool. While multiple pools across projects can use the same organization agent pool, multiple pools within a project cannot use the same organization agent pool.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

[Azure Pipelines and TFS 2017 and newer](#)



TFS 2015

In TFS 2015 agent pools are scoped to project collections.



You create and manage organization agent pools from the agent pools tab in admin settings.

- Azure Pipelines: https://dev.azure.com/{your_organization}/_admin/_AgentPool
- TFS 2018: https://{{your_server}}/DefaultCollection/_admin/_AgentPool
- TFS 2017: https://{{your_server}}/tfss/DefaultCollection/_admin/_AgentPool
- TFS 2015: http://{{your_server}}:8080/tfs/_admin/_AgentPool

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

You create and manage project agent pools from the agent pools tab in project settings.

- Azure Pipelines: https://dev.azure.com/{your_organization}/{project-name}/_admin/_AgentQueue
- TFS 2017 and newer:
https://{{your_server}}/tfss/{collection-name}/{project-name}/_admin/_AgentQueue
- TFS 2015 RTM: http://{{your_server}}:8080/tfs/_admin/_buildQueue
- TFS 2015.3: http://{{your_server}}:8080/tfs/{collection-name}/_admin/_AgentQueue

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

Default agent pools

The following organization agent pools are provided by default:

- **Default** pool: Use it to register [self-hosted agents](#) that you've set up.
- **Hosted Ubuntu 1604** pool (Azure Pipelines only): Enables you to build and release on Linux

machines without having to configure a self-hosted Linux agent. Agents in this pool do not run in a container, but the Docker tools are available for you to use if you want to run [container jobs](#).

- **Hosted Linux** pool (Azure Pipelines only): Enables you to build and release on Linux machines without having to configure a self-hosted Linux agent. The agents in this pool run on an Ubuntu Linux host inside the [vsts-agent-docker container](#). *Note: this pool has been superceded by the Hosted Ubuntu 1604 pool. It will be removed from the service on December 1, 2018. Learn more about [migrating](#).*
- **Hosted macOS** pool (Azure Pipelines only): Enables you to build and release on macOS without having to configure a self-hosted macOS agent. This option affects where your data is stored. [Learn more](#)
- **Hosted VS2017** pool (Azure Pipelines only): The **Hosted VS2017** pool is another built-in pool in Azure Pipelines. Machines in this pool have Visual Studio 2017 installed on Windows Server 2016 operating system. For a complete list of software installed on these machines, see [Microsoft-hosted agents](#).
- **Hosted** pool (Azure Pipelines only): The **Hosted** pool is the built-in pool that is a collection of Microsoft-hosted agents. For a complete list of software installed on Microsoft-hosted agents, see [Microsoft-hosted agents](#).
- **Hosted Windows Container** pool (Azure Pipelines only): Enabled you to build and release inside [Windows containers](#). Unless you're building using containers, Windows builds should run in the **Hosted VS2017** or **Hosted** pools.

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

Pools are used to run jobs. Learn about [specifying pools for jobs](#).

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

Creating agent pools

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

- You're a member of a project and you want to use a set of machines owned by your team for running build and deployment jobs. First, make sure you're a member of a group in **All Pools** with the **Administrator** role. Next create a **New project agent pool** in your project settings and select the option to **Create a new organization agent pool**. As a result, both an organization and project-level agent pool will be created. Finally [install](#) and configure agents to be part of that agent pool.
- You're a member of the infrastructure team and would like to set up a pool of agents for use in all projects. First make sure you're a member of a group in **All Pools** with the **Administrator** role. Next create a **New organization agent pool** in your admin settings and select the option to **Auto-provision corresponding project agent pools in all projects** while creating the pool. This setting ensures all projects have a pool pointing to the organization agent pool. The system creates a pool for existing projects, and in the future it will do so whenever a new project is created. Finally [install](#) and configure agents to be part of that agent pool.
- You want to share a set of agent machines with multiple projects, but not all of them. First create a project agent pool in one of the projects and select the option to **Create a new organization**

agent pool while creating that pool. Next, go to each of the other projects, and create a pool in each of them while selecting the option to **Use an existing organization agent pool**. Finally, [install](#) and configure agents to be part of the shared agent pool.

Security of agent pools

Understanding how security works for agent pools helps you control sharing and use of agents.

Azure Pipelines and TFS 2017 and newer

In Azure Pipelines and TFS 2017 and newer, **roles** are defined on each agent pool, and **membership** in these roles governs what operations you can perform on an agent pool.

ROLE ON AN ORGANIZATION AGENT POOL	PURPOSE
Reader	Members of this role can view the organization agent pool as well as agents. You typically use this to add operators that are responsible for monitoring the agents and their health.
Service Account	Members of this role can use the organization agent pool to create a project agent pool in a project. If you follow the guidelines above for creating new project agent pools, you typically do not have to add any members here.
Administrator	In addition to all the above permissions, members of this role can register or unregister agents from the organization agent pool. They can also refer to the organization agent pool when creating a project agent pool in a project. Finally, they can also manage membership for all roles of the organization agent pool. The user that created the organization agent pool is automatically added to the Administrator role for that pool.

The **All agent pools** node in the Agent Pools tab is used to control the security of *all* organization agent pools. Role memberships for individual organization agent pools are automatically inherited from those of the 'All agent pools' node. By default, TFS administrators are also administrators of the 'All agent pools' node.

Roles are also defined on each organization agent pool, and memberships in these roles govern what operations you can perform on an agent pool.

ROLE ON A PROJECT AGENT POOL	PURPOSE
Reader	Members of this role can view the project agent pool. You typically use this to add operators that are responsible for monitoring the build and deployment jobs in that project agent pool.
User	Members of this role can use the project agent pool when authoring build or release pipelines.
Administrator	In addition to all the above operations, members of this role can manage membership for all roles of the project agent pool. The user that created the pool is automatically added to the Administrator role for that pool.

ROLE ON A PROJECT AGENT POOL	PURPOSE

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

TFS 2015

In TFS 2015, special **groups** are defined on agent pools, and membership in these groups governs what operations you can perform.

Members of **Agent Pool Administrators** can register new agents in the pool and add additional users as administrators or service accounts.

Add people to the Agent Pool Administrators group to grant them permission manage all the agent pools. This enables people to create new pools and modify all existing pools. Members of Team Foundation Administrators group can also perform all these operations.

Users in the **Agent Pool Service Accounts** group have permission to listen to the message queue for the specific pool to receive work. In most cases you should not have to manage members of this group. The agent registration process takes care of it for you. The service account you specify for the agent (commonly Network Service) is automatically added when you register the agent.

Q & A

I'm trying to create a project agent pool that uses an existing organization agent pool, but the controls are grayed out. Why?

On the 'Create a project agent pool' dialog box, you can't use an existing organization agent pool if it is already referenced by another project agent pool. Each organization agent pool can be referenced by only one project agent pool within a given project collection.

I can't select a Microsoft-hosted pool and I can't queue my build. How do I fix this?

Ask the owner of your Azure DevOps organization to grant you permission to use the pool. See [Security of agent pools](#).

I need more hosted build resources. What can I do?

A: The Microsoft-hosted pools provide all Azure DevOps organizations with cloud-hosted build agents and free build minutes each month. If you need more Microsoft-hosted build resources, or need to run more jobs in parallel, then you can either:

- [Host your own agents on infrastructure that you manage.](#)
- [Buy additional parallel jobs.](#)

Microsoft-hosted agents

11/19/2018 • 8 minutes to read • [Edit Online](#)

Azure Pipelines

NOTE

The Hosted Linux Preview Pool is deprecated and will be removed in December 2018. [Learn more below.](#)

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

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

TIP

You can try a Microsoft-hosted agent for no charge. If your build or release doesn't succeed, the issues will be reported in the logs.

Use a Microsoft-hosted agent

The Microsoft-hosted agent pool provides 5 virtual machine images to choose from:

- Ubuntu 16.04 (`ubuntu-16.04`)
- Visual Studio 2017 on Windows Server 2016 (`vs2017-win2016`)
- macOS 10.13 (`macOS-10.13`)
- Windows Server 1803 (`win1803`) - for running Windows containers
- Visual Studio 2015 on Windows Server 2012R2 (`vs2015-win2012r2`)

IF YOUR DEVELOPMENT TEAM USES...	...THEN CHOOSE THIS IMAGE...	...OR POOL IN WEB DESIGNER
Docker containers	<code>ubuntu-16.04</code> or <code>win1803</code>	Hosted Ubuntu 1604 or Hosted Windows Container
Development tools on Ubuntu	<code>ubuntu-16.04</code>	Hosted Ubuntu 1604
Development tools on macOS	<code>macOS-10.13</code> (see notes below)	Hosted macOS
.NET Core	<code>ubuntu-16.04</code> or <code>vs2017-win2016</code>	Hosted Ubuntu 1604 or Hosted VS2017
Visual Studio 2017	<code>vs2017-win2016</code>	Hosted VS2017
Visual Studio 2015	<code>vs2015-win2012r2</code>	Hosted

- [YAML](#)

- [Designer](#)

YAML-based pipelines will default to the Microsoft-hosted agent pool. You simply need to specify which virtual machine image you want to use.

```
jobs:  
- job: Linux  
  pool:  
    vmImage: 'ubuntu-16.04'  
  steps:  
    - script: echo hello from Linux  
- job: macOS  
  pool:  
    vmImage: 'macOS-10.13'  
  steps:  
    - script: echo hello from macOS  
- job: Windows  
  pool:  
    vmImage: 'vs2017-win2016'  
  steps:  
    - script: echo hello from Windows
```

Notes on choosing "Hosted macOS"

This option affects where your data is stored. [Learn more](#). To disable the Hosted macOS agent pool for all projects, disable the `Hosted Agent` checkbox under **Admin settings > Agent pools > Hosted macOS**. To disable the Hosted macOS agent pool for a specific project, disable the `Hosted Agent` checkbox under **Project settings > Agent pools > Hosted macOS**.

You can manually select from tool versions on macOS images. [See below](#).

Software

Software on Microsoft-hosted agents is updated once each month.

- [Visual Studio 2017 on Windows Server 2016 \(Hosted VS2017\)](#).
- [Ubuntu 16.04 \(Hosted Ubuntu 1604\)](#).
- [Xcode 8, 9, and 10 on macOS 10.13 \(Hosted macOS\)](#).
- [Windows Server 1803 \(Hosted Windows Container\)](#)
- [Visual Studio 2015 on Windows Server 2012r2 \(Hosted\)](#).

Capabilities and limitations

Microsoft-hosted agents:

- Have [the above software](#). You can also add software during your build or release using [tool installer tasks](#).
- Provide at least 10 GB of storage for your source and build outputs.
- Can run jobs for up to 360 minutes (6 hours).
- Run on Microsoft Azure general purpose virtual machines [Standard_DS2_v2](#)
- Run as an administrator on Windows and a passwordless sudo user on Linux

Microsoft-hosted agents do not offer:

- The ability to log on.
- The ability to [drop artifacts to a UNC file share](#).
- The ability to run [XAML builds](#).

- Potential performance advantages that you might get by using self-hosted agents which might start and run builds faster. [Learn more](#)

If Microsoft-hosted agents don't meet your needs, then you can [deploy your own self-hosted agents](#).

Avoid hard-coded references

When you use a Microsoft-hosted agent, always use [variables](#) to refer to the build environment and agent resources. For example, don't hardcode the drive letter or folder that contains the repository. The precise layout of the hosted agents is subject to change without warning.

Agent IP ranges

In some setups, you may need to know the range of IP addresses where agents are deployed. For instance, if you need to grant the hosted agents access through a firewall, you may wish to restrict that access by IP address.

NOTE

Because Azure DevOps uses the Azure global network, IP ranges vary over time. We recommend that you check back frequently to ensure you keep an up-to-date list. If agent jobs begin to fail, a key first troubleshooting step is to make sure your configuration matches the latest list of IP addresses.

We publish a [weekly XML file](#) listing IP ranges for Azure datacenters, broken out by region. This file is published every Wednesday (US Pacific time) with new planned IP ranges. The new IP ranges become effective the following Monday. Hosted agents run in the same region as your Azure DevOps organization. You can check your region by navigating to https://dev.azure.com/<your_organization>/_settings/overview. Under **Organization information**, you will see a field indicating your region.

This information is maintained by the [Azure DevOps support team](#).

Q & A

I can't select a Microsoft-hosted agent and I can't queue my build or deployment. How do I fix this?

By default, all project contributors in an organization have access to the Microsoft-hosted agents. But, your organization administrator may limit the access of Microsoft-hosted agents to select users or projects. Ask the owner of your Azure DevOps organization to grant you permission to use a Microsoft-hosted agent. See [agent pool security](#).

I need more agents. What can I do?

A: All Azure DevOps organizations are provided with several free parallel jobs for open source projects, and one free parallel job and limited minutes each month for private projects. If you need more minutes, or need to run additional builds or releases in parallel, then you can buy more [parallel jobs](#) for private projects.

I'm looking for the Microsoft-hosted XAML build controller. Where did it go?

The Microsoft-hosted XAML build controller is no longer supported. If you have an organization in which you still need to run [XAML builds](#), you should set up a [self-hosted build server](#) and a [self-hosted build controller](#).

TIP

We strongly recommend that you [migrate your XAML builds to new builds](#).

How can I manually select versions of tools on the Hosted macOS agent?

Xamarin

To manually select a Xamarin SDK version to use on the **Hosted macOS** agent, before your Xamarin build task, execute this command line as part of your build, replacing the Mono version number 5.4.1 as needed (also replacing '.' characters with underscores: '_'). Choose the Mono version that is associated with the Xamarin SDK version that you need.

```
/bin/bash -c "sudo $AGENT_HOMEDIRECTORY/scripts/select-xamarin-sdk.sh 5_4_1"
```

Mono versions associated with Xamarin SDK versions on the **Hosted macOS** agent can be found [here](#).

Note that this command does not select the Mono version beyond the Xamarin SDK. To manually select a Mono version, see instructions below.

Xcode

If you use the [Xcode task](#) included with Azure Pipelines and TFS, you can select a version of Xcode in that task's properties. Otherwise, to manually set the Xcode version to use on the **Hosted macOS** agent pool, before your `xcodebuild` build task, execute this command line as part of your build, replacing the Xcode version number 8.3.3 as needed:

```
/bin/bash -c "sudo xcode-select -s /Applications/Xcode_8.3.3.app/Contents/Developer"
```

Xcode versions on the **Hosted macOS** agent pool can be found [here](#).

Mono

To manually select a Mono version to use on the **Hosted macOS** agent pool, before your Mono build task, execute this script in each job of your build, replacing the Mono version number 5.4.1 as needed:

```
SYMLINK=5_4_1
MONOPREFIX=/Library/Frameworks/Mono.framework/Versions/$SYMLINK
echo ##vso[task.setvariable
variable=DYLD_FALLBACK_LIBRARY_PATH;]$MONOPREFIX/lib:/lib:/usr/lib:$DYLD_LIBRARY_FALLBACK_PATH"
echo ##vso[task.setvariable
variable=PKG_CONFIG_PATH;]$MONOPREFIX/lib/pkgconfig:$MONOPREFIX/share/pkgconfig:$PKG_CONFIG_PATH"
echo ##vso[task.setvariable variable=PATH;]$MONOPREFIX/bin:$PATH"
```

Hosted Linux Preview pool deprecation

We're deprecating the **Hosted Linux Preview** pool. We recommend that you begin moving your pipelines to the **Hosted Ubuntu 1604** pool now. We plan to remove the **Hosted Linux Preview** from Azure Pipelines on December 1, 2018.

How do I move my builds?

You'll need to change the pool or queue. You might need to also change a few other things.

If you have scripts that assume that they're running as root, then you'll need to change them to use `sudo`. If your pipeline assumes that its agent is running in a container, then you'll need to remove that assumption.

Point at the new pool

- [YAML](#)
- [Designer](#)

Pipelines that use `queue: 'Hosted Linux Preview'` need to be changed to the new `pool` syntax. `queue` has been split into two focused sections: `pool` and `strategy`. `pool` specifies which agent pool receives the build. `strategy` specifies if and how the system should create multiple instances of the jobs you specify.

Example: Pool name only

```

# Before
queue: 'Hosted Linux Preview'

# After
pool:
  vmImage: 'ubuntu-16.04'

```

Example: Pool name with a matrix

```

# Before
queue: 'Hosted Linux Preview'
matrix:
  entry1:
    var: value1
  entry2:
    var: value2

# After
## The matrix syntax has not changed, but it has moved under a `strategy` keyword
pool:
  vmImage: 'ubuntu-16.04'
strategy:
  matrix:
    entry1:
      var: value1
    entry2:
      var: value2

```

Example: Pool name with a matrix and parallelism

```

# Before
queue: 'Hosted Linux Preview'
parallel: 2
matrix:
  entry1:
    var: value1
  entry2:
    var: value2

# After
## The keyword `parallel` has been replaced by `maxParallel` and moved under the strategy node
pool:
  vmImage: 'ubuntu-16.04'
strategy:
  maxParallel: 2
  matrix:
    entry1:
      var: value1
    entry2:
      var: value2

```

Remove user assumptions

If you run tools that require root access, then you'll instead need to run them using `sudo`. For example, if you have a script like this:

```
apt-get update && apt-get install foo
```

Then you'll need to update it to invoke `sudo`:

```
sudo apt-get update && sudo apt-get install foo
```

Remove container assumptions

This is not common.

If your pipeline works with Docker containers, then you may be running commands to account for the agent itself running in a container. On the **Hosted Ubuntu 1604** pool, the agent no longer runs in a container. Typically, this means you can remove parameters or even entire commands from your Docker interactions.

Why is this changing?

The **Hosted Linux Preview** pool ran agents inside a container. If you weren't aware that the agent was in a container, your pipelines would fail in surprising ways if you tried to operate on containers. For example, you could successfully instantiate another container, but because the agent and new containers weren't networked together, you couldn't communicate between them.

The **Hosted Ubuntu 1604** pool uses a more typical model that matches the other hosted pools. The agent runs on the host VM. When you start a container from the agent, the host is networked to the container automatically.

What will happen if I don't move?

On December 1, 2018, your pipelines that attempt to queue for the **Hosted Linux Preview** pool will fail.

Videos

Deploy an agent on Linux

12/3/2018 • 13 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

To build or deploy you'll need at least one agent. A Linux agent can build and deploy different kinds of apps, including Java and Android apps. We support Ubuntu, Red Hat, and CentOS.

Before you begin:

- If your build and release pipelines are in [Azure Pipelines](#) and a [Microsoft-hosted agent](#) meets your needs, you can skip setting up a private Linux agent.
- Otherwise, you've come to the right place to set up an agent on Linux. Continue to the next section.

Learn about agents

If you already know what an agent is and how it works, feel free to jump right in to the following sections. But if you'd like some more background about what they do and how they work, see [Build and release agents](#).

Check prerequisites

Where are your builds and releases running?

Azure Pipelines: The agent is based on CoreCLR 2.0. You can run this agent on several Linux distributions. Make sure your machine is prepared with [our prerequisites](#).

TFS 2018 RTM and older: The agent is based on CoreCLR 1.0. Make sure your machine is prepared with our prerequisites for either of the supported distributions:

- [Ubuntu systems](#)
- [Red Hat/CentOS systems](#)

Subversion

If you're building from a Subversion repo, you must install the Subversion client on the machine.

Prepare permissions

Decide which user you'll use

As a one-time step, you must register the agent. Someone with permission to [administer the agent queue](#) must complete these steps. The agent will not use this person's credentials in everyday operation, but they're required to complete registration. Learn more about [how agents communicate](#).

Authenticate with a personal access token (PAT)

1. Sign in with the user account you plan to use in either your Azure DevOps organization (https://dev.azure.com/{your_organization}) or your Team Foundation Server web portal (<http://your-tfs-site:8080/tfs>)

<https://{{your-server}}:8080/tfs/>).

- From your home page, open your profile. Go to your security details.

The screenshot shows the Azure DevOps user profile page for 'FabrikamFiber'. On the left, there's a sidebar with links like Overview, Summary, Dashboards, Wiki, and Boards. The main area shows a profile picture for 'Jamal Hartnett' and an email address 'fabrikamfiber4@hotmail.com'. A red box highlights the 'Security' link in the dropdown menu under 'My profile'.

- Create a personal access token.

The screenshot shows the 'Personal Access Tokens' page under 'User settings > Personal access tokens'. On the left, there's a sidebar with sections like General, Notifications, Usage, Security (which is expanded), and Personal access tokens (which is selected and highlighted). The main area shows a section titled 'Personal Access Tokens' with a sub-section 'These can be used instead of a password for applica...'. A red box highlights the '+ New Token' button. Below it, there's a table with one row for 'Jamal Hartnett' with scopes 'Code (Read, write, & manage); Code (Read & write); Pac...'.

- For the scope select **Agent Pools (read, manage)** and make sure all the other boxes are cleared. If it's a **deployment group** agent, for the scope select **Deployment group (read, manage)** and make sure all the other boxes are cleared.

- Copy the token. You'll use this token when you configure the agent.

Authenticate as a Windows user (TFS 2015 and TFS 2017)

As an alternative, on TFS 2017, you can use either a domain user or a local Windows user on each of your TFS application tiers.

On TFS 2015, for macOS and Linux only, we recommend that you create a local Windows user on each of your TFS application tiers and dedicate that user for the purpose of deploying build agents.

Confirm the user has permission

Make sure the user account that you're going to use has permission to register the agent.

Is the user an Azure DevOps organization owner or TFS server administrator? **Stop here**, you have permission.

Otherwise:

1. Open a browser and navigate to the **Agent pools** tab for your Azure Pipelines organization or TFS server:

- Azure Pipelines: https://dev.azure.com/{your_organization}/_admin/_AgentPool
- TFS 2018: https://{your_server}/DefaultCollection/_admin/_AgentPool
- TFS 2017: https://{your_server}/tfss/DefaultCollection/_admin/_AgentPool
- TFS 2015: http://{your_server}:8080/tfs/_admin/_AgentPool

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

2. Click the pool on the left side of the page and then click **Roles**.

3. If the user account you're going to use is not shown, then get an administrator to add it. The administrator can be an agent pool administrator, an [Azure DevOps organization owner](#), or a [TFS server administrator](#). If it's a [deployment group](#) agent, the administrator can be an deployment group administrator, an [Azure DevOps organization owner](#), or a [TFS server administrator](#). You can add a user to the deployment group administrator role in the **Security** tab on the **Deployment Groups** page in [Azure Pipelines](#).

If you see a message like this: *Sorry, we couldn't add the identity. Please try a different identity.*, you probably followed the above steps for an organization owner or TFS server administrator. You don't need to do anything; you already have permission to administer the agent queue.

Download and configure the agent

Azure Pipelines

1. Log on to the machine using the account for which you've prepared permissions as explained above.

2. In your web browser, sign in to Azure Pipelines, and navigate to the **Agent pools** tab:

- Azure Pipelines: https://dev.azure.com/{your_organization}/_admin/_AgentPool
- TFS 2018: https://{your_server}/DefaultCollection/_admin/_AgentPool
- TFS 2017: https://{your_server}/tfss/DefaultCollection/_admin/_AgentPool
- TFS 2015: http://{your_server}:8080/tfs/_admin/_AgentPool

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

3. Click **Download agent**.

4. On the **Get agent** dialog box, click **Linux**.

5. On the left pane, select the processor architecture (x64 or ARM).

6. On the right pane, click the **Download** button.

7. Follow the instructions on the page.

8. Unpack the agent into the directory of your choice. `cd` to that directory and run `./config.sh`.

TFS 2017 and TFS 2018

1. Log on to the machine using the account for which you've prepared permissions as explained above.

2. In your web browser, sign in to TFS, and navigate to the **Agent pools** tab:

- Azure Pipelines: https://dev.azure.com/{your_organization}/_admin/_AgentPool
- TFS 2018: https://{your_server}/DefaultCollection/_admin/_AgentPool
- TFS 2017: https://{your_server}/tfss/DefaultCollection/_admin/_AgentPool
- TFS 2015: http://{your_server}:8080/tfs/_admin/_AgentPool

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

3. Click **Download agent**.

4. On the **Get agent** dialog box, click **Linux**.

5. Click the **Download** button.
6. Follow the instructions on the page.
7. Unpack the agent into the directory of your choice. `cd` to that directory and run `./config.sh`. Make sure that the path to the directory contains no spaces because tools and scripts don't always properly escape spaces.

TFS 2015

1. Browse to the [latest release on GitHub](#).
2. Follow the instructions on that page to download the agent.
3. Configure the agent.

```
./config.sh
```

Server URL

Azure Pipelines: `https://dev.azure.com/{your-organization}`

TFS 2017 and newer: `https://{your_server}/tfss`

TFS 2015: `http://{your_server}:8080/tfs`

Authentication type

Azure Pipelines

Choose **PAT**, and then paste the [PAT token you created](#) into the command prompt window.

NOTE

When using PAT as the authentication method, the PAT token is used only for the initial configuration of the agent. Learn more at [Communication with Azure Pipelines or TFS](#).

TFS

IMPORTANT

Make sure your server is [configured to support the authentication method](#) you want to use.

When you configure your agent to connect to TFS, you've got the following options:

- **Alternate** Connect to TFS using Basic authentication. After you select Alternate you'll be prompted for your credentials.
- **Integrated** Not supported on macOS or Linux.
- **Negotiate** (Default) Connect to TFS as a user other than the signed-in user via a Windows authentication scheme such as NTLM or Kerberos. After you select Negotiate you'll be prompted for credentials.
- **PAT** Supported only on Azure Pipelines and TFS 2017 and newer. After you choose PAT, paste the [PAT token you created](#) into the command prompt window. Use a personal access token (PAT) if your TFS instance and the agent machine are not in a trusted domain. PAT authentication is handled by your TFS instance instead of the domain controller.

NOTE

When using PAT as the authentication method, the PAT token is used only for the initial configuration of the agent on newer versions of TFS. Learn more at [Communication with Azure Pipelines or TFS](#).

Run interactively

For guidance on whether to run the agent in interactive mode or as a service, see [Agents: Interactive vs. service](#).

To run the agent interactively:

1. If you have been running the agent as a service, [uninstall the service](#).
2. Run the agent.

```
./run.sh
```

To use your agent, run a [job](#) using the agent's pool. If you didn't choose a different pool, your agent will be in the **Default** pool.

Run as a systemd service

If your agent is running on these operating systems you can run the agent as a systemd service:

- Ubuntu 16 LTS or newer
- Red Hat 7.1 or newer

We provide the `./svc.sh` script for you to run and manage your agent as a systemd service. This script will be generated after you configure the agent.

NOTE

If you have a different distribution, or if you prefer other approaches, you can use whatever kind of service mechanism you prefer. See [Service files](#).

Commands

Change to the agent directory

For example, if you installed in the `myagent` subfolder of your home directory:

```
cd ~/myagent$
```

Install

Command:

```
sudo ./svc.sh install
```

This command creates a service file that points to `./runsvcs.sh`. This script sets up the environment (more details below) and starts the agents host.

Start

```
sudo ./svc.sh start
```

Status

```
sudo ./svc.sh status
```

Stop

```
sudo ./svc.sh stop
```

Uninstall

You should stop before you uninstall.

```
sudo ./svc.sh uninstall
```

Update environment variables

When you configure the service, it takes a snapshot of some useful environment variables for your current logon user such as PATH, LANG, JAVA_HOME, ANT_HOME, and MYSQL_PATH. If you need to update the variables (for example, after installing some new software):

1.

```
./env.sh
```

2.

```
sudo ./svc.sh stop
```

3.

```
sudo ./svc.sh start
```

The snapshot of the environment variables is stored in `.env` file under agent root directory, you can also change that file directly to apply environment variable changes.

Run instructions before the service starts

You can also run your own instructions and commands to run when the service starts. For example, you could set up the environment or call scripts.

1. Edit `runsvc.sh`.

2. Replace the following line with your instructions:

```
# insert anything to setup env when running as a service
```

Service files

When you install the service, some service files are put in place.

systemd service file

A systemd service file is created:

```
/etc/systemd/system/vsts.agent.{tfs-name}.{agent-name}.service
```

For example, you have configured an agent (see above) with the name `our-linux-agent`. The service file will be either:

- Azure Pipelines: the name of your organization. For example if you connect to

```
https://dev.azure.com/fabrikam, then the service name would be  
/etc/systemd/system/vsts.agent.fabrikam.our-linux-agent.service
```

- TFS: the name of your on-premises TFS AT server. For example if you connect to

```
http://our-server:8080/tfs, then the service name would be  
/etc/systemd/system/vsts.agent.our-server.our-linux-agent.service
```

```
sudo ./svc.sh install generates this file from this template: ./bin/vsts.agent.service.template
```

.service file

`sudo ./svc.sh start` finds the service by reading the `.service` file, which contains the name of systemd service file described above.

Alternative service mechanisms

We provide the `./svc.sh` script as a convenient way for you to run and manage your agent as a systemd service. But you can use whatever kind of service mechanism you prefer (for example: initd or upstart).

You can use the template described above as to facilitate generating other kinds of service files.

Replace an agent

To replace an agent, follow the *Download and configure the agent* steps again.

When you configure an agent using the same name as an agent that already exists, you're asked if you want to replace the existing agent. If you answer `y`, then make sure you remove the agent (see below) that you're replacing. Otherwise, after a few minutes of conflicts, one of the agents will shut down.

Remove and re-configure an agent

To remove the agent:

1. Stop and uninstall the service as explained above.
2. Remove the agent.

```
./config.sh remove
```

3. Enter your credentials.

After you've removed the agent, you can [configure it again](#).

Unattended config

The agent can be set up from a script with no human intervention. You must pass `--unattended` and the answers to all questions. Consult `./config.sh --help` for details about the required responses.

Help on other options

To learn about other options:

```
./config.sh --help
```

The help provides information on authentication alternatives and unattended configuration.

Capabilities

Your agent's capabilities are cataloged and advertised in the pool so that only the builds and releases it can handle are assigned to it. See [Build and release agent capabilities](#).

In many cases, after you deploy an agent, you'll need to install software or utilities. Generally you should install on your agents whatever software and tools you use on your development machine.

For example, if your build includes the [npm task](#), then the build won't run unless there's a build agent in the pool that has npm installed.

IMPORTANT

After you install new software on an agent, you must restart the agent for the new capability to show up in the pool so that the build can run.

Q & A

How do I make sure I have the latest v2 agent version?

1. Go to the *Agent pools* control panel tab:

- Azure Pipelines: https://dev.azure.com/{your_organization}/_admin/_AgentPool
- TFS 2018: https://{your_server}/DefaultCollection/_admin/_AgentPool
- TFS 2017: https://{your_server}/tfss/DefaultCollection/_admin/_AgentPool
- TFS 2015: http://{your_server}:8080/tfs/_admin/_AgentPool

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

2. Click the pool that contains the agent.

3. Make sure the agent is enabled.

4. Click **Agents**.

5. Click **Capabilities**.

6. Look for the `Agent.Version` capability.

You can check this value against the latest published agent version. See [Azure Pipelines Agent](#) and check the page for the highest version number listed.

7. Each agent automatically updates itself when it runs a task that requires a newer version of the agent. But if you want to manually update some agents, right-click the pool, and then choose **Update all agents**.

Can I update my v2 agents that are part of an Azure DevOps Server pool?

Yes. You can configure your Azure DevOps Server to look for the agent package files on a local disk, overriding the default version that came with the server (at the time of its release). This scenario also applies when the server does not have access to the Internet.

1. From a computer with Internet access, download the latest version of the agent package files (in .zip or .tar.gz form) from the [Azure Pipelines Agent GitHub Releases page](#).
2. Transfer the downloaded package files to each Azure DevOps Server Application Tier, via a method of your choice (e.g. USB drive, Network transfer). Place the agent files under the

`%ProgramData%\Microsoft\Azure DevOps\Agents` folder. **Note:** in Azure DevOps Server 2019 RC1, the folder is called `%ProgramData%\Microsoft\DevOps\Agents`.

3. You're all set! Your Azure DevOps Server will now use the local files whenever the agents need to be updated. Each agent automatically updates itself when it runs a task that requires a newer version of the agent. But if you want to manually update some agents, right-click the pool, and then choose **Update all agents**.

Why is sudo needed to run the service commands?

`./svc.sh` uses `systemctl`, which requires `sudo`.

Source code: [systemd.svc.sh.template](#) on GitHub

I'm running a firewall and my code is in Azure Repos. What URLs does the agent need to communicate with?

If you're running an agent in a secure network behind a firewall, make sure the agent can initiate communication with the following URLs and IP addresses.

For organizations using the `*.visualstudio.com` domain:

```
https://login.microsoftonline.com  
https://app.vssps.visualstudio.com  
https://{{organization_name}}.visualstudio.com  
https://{{organization_name}}.vsrm.visualstudio.com  
https://{{organization_name}}.pkgs.visualstudio.com  
https://{{organization_name}}.vssps.visualstudio.com
```

For organizations using the `dev.azure.com` domain:

```
https://dev.azure.com  
https://*.dev.azure.com  
https://login.microsoftonline.com  
https://management.core.windows.net
```

To ensure your organization works with any existing firewall or IP restrictions, ensure that `dev.azure.com` is open and the `13.107.6.183` and `13.107.9.183` IP addresses are allowed.

How do I run the agent with self-signed certificate?

[Run the agent with self-signed certificate](#)

How do I run the agent behind a web proxy?

[Run the agent behind a web proxy](#)

How do I configure the agent to bypass a web proxy and connect to Azure Pipelines?

If you want the agent to bypass your proxy and connect to Azure Pipelines directly, then you should configure your web proxy to enable the agent to access the following URLs.

For organizations using the `*.visualstudio.com` domain:

```
https://login.microsoftonline.com  
https://app.vssps.visualstudio.com  
https://{{organization_name}}.visualstudio.com  
https://{{organization_name}}.vsrm.visualstudio.com  
https://{{organization_name}}.pkgs.visualstudio.com  
https://{{organization_name}}.vssps.visualstudio.com
```

For organizations using the `dev.azure.com` domain:

```
https://dev.azure.com  
https://*.dev.azure.com  
https://login.microsoftonline.com  
https://management.core.windows.net
```

To ensure your organization works with any existing firewall or IP restrictions, ensure that `dev.azure.com` is open and the `13.107.6.183` and `13.107.9.183` IP addresses are allowed.

NOTE

This procedure enables the agent to bypass a web proxy. Your build pipeline and scripts must still handle bypassing your web proxy for each task and tool you run in your build.

For example, if you are using a NuGet task, you must configure your web proxy to support bypassing the URL for the server that hosts the NuGet feed you're using.

I'm using TFS and the URLs in the sections above don't work for me. Where can I get help?

[Web site settings and security](#)

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Deploy an agent on macOS

12/3/2018 • 13 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

To build and deploy Xcode apps or Xamarin.iOS projects, you'll need at least one macOS agent. This agent can also build and deploy Java and Android apps.

Before you begin:

- If your build and release pipelines are in [Azure Pipelines](#) and a [Microsoft-hosted agent](#) meets your needs, you can skip setting up a private macOS agent.
- Otherwise, you've come to the right place to set up an agent on macOS. Continue to the next section.

Learn about agents

If you already know what an agent is and how it works, feel free to jump right in to the following sections. But if you'd like some more background about what they do and how they work, see [Build and release agents](#).

Check prerequisites

Make sure your machine is prepared with our [macOS system prerequisites](#).

Prepare permissions

If you're building from a Subversion repo, you must install the Subversion client on the machine.

Decide which user you'll use

As a one-time step, you must register the agent. Someone with permission to [administer the agent queue](#) must complete these steps. The agent will not use this person's credentials in everyday operation, but they're required to complete registration. Learn more about [how agents communicate](#).

Authenticate with a personal access token (PAT)

1. Sign in with the user account you plan to use in either your Azure DevOps organization (
`https://dev.azure.com/{your_organization}`) or your Team Foundation Server web portal (
`https://{your-server}:8080/tfs/`).
2. From your home page, open your profile. Go to your security details.

The screenshot shows the Azure DevOps interface for the 'FabrikamFiber' project. On the left, there's a sidebar with links like 'Overview', 'Summary', 'Dashboards', 'Wiki', and 'Boards'. The 'Summary' link is currently selected. On the right, there's a profile section for 'Jamal Hartnett' with an email address 'fabrikamfiber4@hotmail.com'. Below the profile, there's a 'About this project' section and a 'Languages' section. A navigation bar at the top has a 'Security' link, which is highlighted with a red box.

3. Create a personal access token.

The screenshot shows the 'User settings > Personal access tokens' page. On the left, there's a sidebar with sections like 'General', 'Notifications', 'Usage', 'Security', and 'SSH public keys'. The 'Personal access tokens' link under 'Security' is currently selected. On the right, there's a 'Personal Access Tokens' section with a sub-section for 'Jamal Hartnett'. A large blue button labeled '+ New Token' is highlighted with a red box. Below it, there's a 'Token name' input field and a list of scopes for the token.

4. For the scope select **Agent Pools (read, manage)** and make sure all the other boxes are cleared. If it's a **deployment group** agent, for the scope select **Deployment group (read, manage)** and make sure all the other boxes are cleared.

5. Copy the token. You'll use this token when you configure the agent.

Authenticate as a Windows user (TFS 2015 and TFS 2017)

As an alternative, on TFS 2017, you can use either a domain user or a local Windows user on each of your TFS application tiers.

On TFS 2015, for macOS and Linux only, we recommend that you create a local Windows user on each of your TFS application tiers and dedicate that user for the purpose of deploying build agents.

Confirm the user has permission

Make sure the user account that you're going to use has permission to register the agent.

Is the user an Azure DevOps organization owner or TFS server administrator? **Stop here**, you have permission.

Otherwise:

1. Open a browser and navigate to the **Agent pools** tab for your Azure Pipelines organization or TFS server:

- Azure Pipelines: https://dev.azure.com/{your_organization}/_admin/_AgentPool
- TFS 2018: https://{your_server}/DefaultCollection/_admin/_AgentPool
- TFS 2017: https://{your_server}/tfss/DefaultCollection/_admin/_AgentPool
- TFS 2015: http://{your_server}:8080/tfs/_admin/_AgentPool

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

2. Click the pool on the left side of the page and then click **Roles**.

3. If the user account you're going to use is not shown, then get an administrator to add it. The administrator can be an agent pool administrator, an [Azure DevOps organization owner](#), or a [TFS server administrator](#). If it's a [deployment group](#) agent, the administrator can be a deployment group administrator, an [Azure DevOps organization owner](#), or a [TFS server administrator](#). You can add a user to the deployment group administrator role in the **Security** tab on the **Deployment Groups** page in [Azure Pipelines](#).

If you see a message like this: *Sorry, we couldn't add the identity. Please try a different identity.*, you probably followed the above steps for an organization owner or TFS server administrator. You don't need to do anything; you already have permission to administer the agent queue.

Download and configure the agent

Azure Pipelines and TFS 2017 and newer

1. Log on to the machine using the account for which you've prepared permissions as explained above.

2. In your web browser, sign in to Azure Pipelines or TFS, and navigate to the **Agent pools** tab:

- Azure Pipelines: https://dev.azure.com/{your_organization}/_admin/_AgentPool
- TFS 2018: https://{your_server}/DefaultCollection/_admin/_AgentPool
- TFS 2017: https://{your_server}/tfss/DefaultCollection/_admin/_AgentPool
- TFS 2015: http://{your_server}:8080/tfs/_admin/_AgentPool

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

3. Click **Download agent**.

4. On the **Get agent** dialog box, click **macOS**.

5. Click the **Download** button.

6. Follow the instructions on the page.

7. Unpack the agent into the directory of your choice. `cd` to that directory and run `./config.sh`. Make sure that the path to the directory contains no spaces because tools and scripts don't always properly escape spaces.

TFS 2015

1. Browse to the [latest release on GitHub](#).

2. Follow the instructions on that page to download the agent.

3. Configure the agent.

```
./config.sh
```

Server URL

Azure Pipelines: <https://dev.azure.com/{your-organization}>

TFS 2017 and newer: https://{your_server}/tfss

TFS 2015: http://{your_server}:8080/tfs

Authentication type

Azure Pipelines

Choose **PAT**, and then paste the [PAT token you created](#) into the command prompt window.

NOTE

When using PAT as the authentication method, the PAT token is used only for the initial configuration of the agent. Learn more at [Communication with Azure Pipelines or TFS](#).

TFS

IMPORTANT

Make sure your server is [configured to support the authentication method](#) you want to use.

When you configure your agent to connect to TFS, you've got the following options:

- **Alternate** Connect to TFS using Basic authentication. After you select Alternate you'll be prompted for your credentials.
- **Integrated** Not supported on macOS or Linux.
- **Negotiate** (Default) Connect to TFS as a user other than the signed-in user via a Windows authentication scheme such as NTLM or Kerberos. After you select Negotiate you'll be prompted for credentials.
- **PAT** Supported only on Azure Pipelines and TFS 2017 and newer. After you choose PAT, paste the [PAT token you created](#) into the command prompt window. Use a personal access token (PAT) if your TFS instance and the agent machine are not in a trusted domain. PAT authentication is handled by your TFS instance instead of the domain controller.

NOTE

When using PAT as the authentication method, the PAT token is used only for the initial configuration of the agent on newer versions of TFS. Learn more at [Communication with Azure Pipelines or TFS](#).

Run interactively

For guidance on whether to run the agent in interactive mode or as a service, see [Agents: Interactive vs. service](#).

To run the agent interactively:

1. If you have been running the agent as a service, [uninstall the service](#).
2. Run the agent.

```
./run.sh
```

To use your agent, run a **job** using the agent's pool. If you didn't choose a different pool, your agent will be in the **Default** pool.

Run as a launchd service

We provide the `./svc.sh` script for you to run and manage your agent as a launchd LaunchAgent service. This script will be generated after you configure the agent. The service has access to the UI to run your UI tests.

NOTE

If you prefer other approaches, you can use whatever kind of service mechanism you prefer. See [Service files](#).

Tokens

In the section below, these tokens are replaced:

- `{agent-name}`
- `{tfs-name}`

For example, you have configured an agent (see above) with the name `our-osx-agent`. In the following examples, `{tfs-name}` will be either:

- Azure Pipelines: the name of your organization. For example if you connect to
`https://dev.azure.com/fabrikam`, then the service name would be `vsts.agent.fabrikam.our-osx-agent`
- TFS: the name of your on-premises TFS AT server. For example if you connect to
`http://our-server:8080/tfs`, then the service name would be `vsts.agent.our-server.our-osx-agent`

Commands

Change to the agent directory

For example, if you installed in the `myagent` subfolder of your home directory:

```
cd ~/myagent$
```

Install

Command:

```
./svc.sh install
```

This command creates a launchd plist that points to `./runsvc.sh`. This script sets up the environment (more details below) and starts the agent's host.

Start

Command:

```
./svc.sh start
```

Output:

```
starting vsts.agent.{tfs-name}.{agent-name}
status vsts.agent.{tfs-name}.{agent-name}:

/Users/{your-name}/Library/LaunchAgents/vsts.agent.{tfs-name}.{agent-name}.plist

Started:
13472 0 vsts.agent.{tfs-name}.{agent-name}
```

The left number is the pid if the service is running. If second number is not zero, then a problem occurred.

Status

Command:

```
./svc.sh status
```

Output:

```
status vsts.agent.{tfss-name}.{agent-name}:
/Users/{your-name}/Library/LaunchAgents/vsts.{tfss-name}.{agent-name}.testsrv.plist

Started:
13472 0 vsts.agent.{tfss-name}.{agent-name}
```

The left number is the pid if the service is running. If second number is not zero, then a problem occurred.

Stop

Command:

```
./svc.sh stop
```

Output:

```
stopping vsts.agent.{tfss-name}.{agent-name}
status vsts.agent.{tfss-name}.{agent-name}:
/Users/{your-name}/Library/LaunchAgents/vsts.{tfss-name}.{agent-name}.testsrv.plist

Stopped
```

Uninstall

You should stop before you uninstall.

Command:

```
./svc.sh uninstall
```

Automatic login and lock

Normally, the agent service runs only after the user logs in. If you want the agent service to automatically start when the machine restarts, you can configure the machine to automatically log in and lock on startup. See [Set your Mac to automatically log in during startup - Apple Support](#).

Update environment variables

When you configure the service, it takes a snapshot of some useful environment variables for your current logon user such as PATH, LANG, JAVA_HOME, ANT_HOME, and MYSQL_PATH. If you need to update the variables (for example, after installing some new software):

1.

```
./env.sh
```

2.

```
./svc.sh stop
```

3.

```
./svc.sh start
```

The snapshot of the environment variables is stored in `.env` file under agent root directory, you can also change that file directly to apply environment variable changes.

Run instructions before the service starts

You can also run your own instructions and commands to run when the service starts. For example, you could set up the environment or call scripts.

1. Edit `runsvc.sh`.

2. Replace the following line with your instructions:

```
# insert anything to setup env when running as a service
```

Service Files

When you install the service, some service files are put in place.

.plist service file

A .plist service file is created:

```
~/Library/LaunchAgents/vsts.agent.{tfs-name}.{agent-name}.plist
```

For example:

```
~/Library/LaunchAgents/vsts.agent.fabrikam.our-osx-agent.plist
```

```
sudo ./svc.sh install generates this file from this template: ./bin/vsts.agent.plist.template
```

.service file

`./svc.sh start` finds the service by reading the `.service` file, which contains the path to the plist service file described above.

Alternative service mechanisms

We provide the `./svc.sh` script as a convenient way for you to run and manage your agent as a launchd LaunchAgent service. But you can use whatever kind of service mechanism you prefer.

You can use the template described above as to facilitate generating other kinds of service files. For example, you modify the template to generate a service that runs as a launch daemon if you don't need UI tests and don't want to configure automatic log on and lock. See [Apple Developer Library: Creating Launch Daemons and Agents](#).

Replace an agent

To replace an agent, follow the *Download and configure the agent* steps again.

When you configure an agent using the same name as an agent that already exists, you're asked if you want to replace the existing agent. If you answer `y`, then make sure you remove the agent (see below) that you're replacing. Otherwise, after a few minutes of conflicts, one of the agents will shut down.

Remove and re-configure an agent

To remove the agent:

1. Stop and uninstall the service as explained above.
2. Remove the agent.

```
./config.sh remove
```

3. Enter your credentials.

After you've removed the agent, you can [configure it again](#).

Unattended config

The agent can be set up from a script with no human intervention. You must pass `--unattended` and the answers to all questions. Consult `./config.sh --help` for details about the required responses.

Help on other options

To learn about other options:

```
./config.sh --help
```

The help provides information on authentication alternatives and unattended configuration.

Capabilities

Your agent's capabilities are cataloged and advertised in the pool so that only the builds and releases it can handle are assigned to it. See [Build and release agent capabilities](#).

In many cases, after you deploy an agent, you'll need to install software or utilities. Generally you should install on your agents whatever software and tools you use on your development machine.

For example, if your build includes the [npm task](#), then the build won't run unless there's a build agent in the pool that has npm installed.

IMPORTANT

After you install new software on an agent, you must restart the agent for the new capability to show up in the pool so that the build can run.

Q & A

How do I make sure I have the latest v2 agent version?

1. Go to the *Agent pools* control panel tab:

- Azure Pipelines: `https://dev.azure.com/{your_organization}/_admin/_AgentPool`
- TFS 2018: `https://{your_server}/DefaultCollection/_admin/_AgentPool`
- TFS 2017: `https://{your_server}/tfss/DefaultCollection/_admin/_AgentPool`
- TFS 2015: `http://{your_server}:8080/tfs/_admin/_AgentPool`

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

2. Click the pool that contains the agent.
3. Make sure the agent is enabled.

4. Click **Agents**.

5. Click **Capabilities**.

6. Look for the `Agent.Version` capability.

You can check this value against the latest published agent version. See [Azure Pipelines Agent](#) and check the page for the highest version number listed.

7. Each agent automatically updates itself when it runs a task that requires a newer version of the agent. But if you want to manually update some agents, right-click the pool, and then choose **Update all agents**.

Can I update my v2 agents that are part of an Azure DevOps Server pool?

Yes. You can configure your Azure DevOps Server to look for the agent package files on a local disk, overriding the default version that came with the server (at the time of its release). This scenario also applies when the server does not have access to the Internet.

1. From a computer with Internet access, download the latest version of the agent package files (in .zip or .tar.gz form) from the [Azure Pipelines Agent GitHub Releases page](#).
2. Transfer the downloaded package files to each Azure DevOps Server Application Tier, via a method of your choice (e.g. USB drive, Network transfer). Place the agent files under the `%ProgramData%\Microsoft\Azure DevOps\Agents` folder. **Note:** in Azure DevOps Server 2019 RC1, the folder is called `%ProgramData%\Microsoft\DevOps\Agents`.
3. You're all set! Your Azure DevOps Server will now use the local files whenever the agents need to be updated. Each agent automatically updates itself when it runs a task that requires a newer version of the agent. But if you want to manually update some agents, right-click the pool, and then choose **Update all agents**.

Where can I learn more about how the launchd service works?

[Apple Developer Library: Creating Launch Daemons and Agents](#)

I'm running a firewall and my code is in Azure Repos. What URLs does the agent need to communicate with?

If you're running an agent in a secure network behind a firewall, make sure the agent can initiate communication with the following URLs and IP addresses.

For organizations using the `*.visualstudio.com` domain:

```
https://login.microsoftonline.com  
https://app.vssps.visualstudio.com  
https://{{organization_name}}.visualstudio.com  
https://{{organization_name}}.vsrm.visualstudio.com  
https://{{organization_name}}.pkgs.visualstudio.com  
https://{{organization_name}}.vssps.visualstudio.com
```

For organizations using the `dev.azure.com` domain:

```
https://dev.azure.com  
https://*.dev.azure.com  
https://login.microsoftonline.com  
https://management.core.windows.net
```

To ensure your organization works with any existing firewall or IP restrictions, ensure that `dev.azure.com` is open and the `13.107.6.183` and `13.107.9.183` IP addresses are allowed.

How do I run the agent with self-signed certificate?

[Run the agent with self-signed certificate](#)

How do I run the agent behind a web proxy?

[Run the agent behind a web proxy](#)

How do I configure the agent to bypass a web proxy and connect to Azure Pipelines?

If you want the agent to bypass your proxy and connect to Azure Pipelines directly, then you should configure your web proxy to enable the agent to access the following URLs.

For organizations using the `*.visualstudio.com` domain:

```
https://login.microsoftonline.com  
https://app.vssps.visualstudio.com  
https://{organization_name}.visualstudio.com  
https://{organization_name}.vsrm.visualstudio.com  
https://{organization_name}.pkgs.visualstudio.com  
https://{organization_name}.vssps.visualstudio.com
```

For organizations using the `dev.azure.com` domain:

```
https://dev.azure.com  
https://*.dev.azure.com  
https://login.microsoftonline.com  
https://management.core.windows.net
```

To ensure your organization works with any existing firewall or IP restrictions, ensure that `dev.azure.com` is open and the `13.107.6.183` and `13.107.9.183` IP addresses are allowed.

NOTE

This procedure enables the agent to bypass a web proxy. Your build pipeline and scripts must still handle bypassing your web proxy for each task and tool you run in your build.

For example, if you are using a NuGet task, you must configure your web proxy to support bypassing the URL for the server that hosts the NuGet feed you're using.

I'm using TFS and the URLs in the sections above don't work for me. Where can I get help?

[Web site settings and security](#)

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Deploy an agent on Windows

12/3/2018 • 12 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#) | [Previous versions \(XAML builds\)](#)

To build and deploy Windows, Azure, and other Visual Studio solutions you'll need at least one Windows agent. Windows agents can also build Java and Android apps.

Before you begin:

- If your code is in [Azure Pipelines](#) and a [Microsoft-hosted agent](#) meets your needs, you can skip setting up a private Windows agent.
- If your code is in an on-premises Team Foundation Server (TFS) 2015 server, see [Deploy an agent on Windows for on-premises TFS 2015](#).
- Otherwise, you've come to the right place to set up an agent on Windows. Continue to the next section.

Learn about agents

If you already know what an agent is and how it works, feel free to jump right in to the following sections. But if you'd like some more background about what they do and how they work, see [Build and release agents](#).

Check prerequisites

Make sure your machine is prepared with our [Windows system prerequisites](#).

If you're building from a Subversion repo, you must install the Subversion client on the machine.

Hardware specs

The hardware specs for your agents will vary with your needs, team size, etc. It's not possible to make a general recommendation that will apply to everyone. As a point of reference, the Azure DevOps team builds its hosted agents using the [hosted agents](#). On the other hand, the bulk of the Azure DevOps code is built by 24-core server class machines running 4 agents apiece.

Prepare permissions

Decide which user you'll use

As a one-time step, you must register the agent. Someone with permission to [administer the agent queue](#) must complete these steps. The agent will not use this person's credentials in everyday operation, but they're required to complete registration. Learn more about [how agents communicate](#).

Authenticate with a personal access token (PAT)

1. Sign in with the user account you plan to use in either your Azure DevOps organization (https://dev.azure.com/{your_organization}) or your Team Foundation Server web portal (<https://{your-server}:8080/tfs/>).
2. From your home page, open your profile. Go to your security details.

The screenshot shows the Azure DevOps interface for the 'FabrikamFiber' project. On the left, there's a sidebar with links for Overview, Summary, Dashboards, Wiki, and Boards. The main area has a header with the project name 'Fabrikam' and a profile picture for 'Jamal Hartnett'. Below the header, there are several navigation links: 'My profile', 'Security' (which is highlighted with a red box), 'Usage', 'Notification settings', 'Preview features', and 'Help'. The 'About this project' section is also visible.

3. Create a personal access token.

The screenshot shows the 'User settings > Personal access tokens' page. On the left, there's a sidebar with sections for General (Notifications, Usage), Security (Personal access tokens, Alternate credentials, SSH public keys), and a collapsed 'Agent pools' section. The 'Personal access tokens' section is currently selected. The main area is titled 'Personal Access Tokens' and contains a note: 'These can be used instead of a password for application access.' Below this is a form with a 'Token name' field and a 'New Token' button, which is highlighted with a red box. To the right, there's a table showing a single token entry for 'Jamal Hartnett' with scopes like 'Code (Read, write, & manage); Code (Read & write); Pac...'.

4. For the scope select **Agent Pools (read, manage)** and make sure all the other boxes are cleared. If it's a **deployment group** agent, for the scope select **Deployment group (read, manage)** and make sure all the other boxes are cleared.

5. Copy the token. You'll use this token when you configure the agent.

Authenticate as a Windows user (TFS 2015 and TFS 2017)

As an alternative, on TFS 2017, you can use either a domain user or a local Windows user on each of your TFS application tiers.

On TFS 2015, for macOS and Linux only, we recommend that you create a local Windows user on each of your TFS application tiers and dedicate that user for the purpose of deploying build agents.

Confirm the user has permission

Make sure the user account that you're going to use has permission to register the agent.

Is the user an Azure DevOps organization owner or TFS server administrator? **Stop here**, you have permission.

Otherwise:

1. Open a browser and navigate to the **Agent pools** tab for your Azure Pipelines organization or TFS server:

- Azure Pipelines: https://dev.azure.com/{your_organization}/_admin/_AgentPool
- TFS 2018: https://{your_server}/DefaultCollection/_admin/_AgentPool
- TFS 2017: https://{your_server}/tfs/DefaultCollection/_admin/_AgentPool
- TFS 2015: http://{your_server}:8080/tfs/_admin/_AgentPool

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

2. Click the pool on the left side of the page and then click **Roles**.

3. If the user account you're going to use is not shown, then get an administrator to add it. The administrator can be an agent pool administrator, an [Azure DevOps organization owner](#), or a [TFS server administrator](#). If it's a [deployment group](#) agent, the administrator can be an deployment group administrator, an [Azure DevOps organization owner](#), or a [TFS server administrator](#). You can add a user to the deployment group administrator role in the **Security** tab on the **Deployment Groups** page in [Azure Pipelines](#).

If you see a message like this: *Sorry, we couldn't add the identity. Please try a different identity.*, you probably followed the above steps for an organization owner or TFS server administrator. You don't need to do anything; you already have permission to administer the agent queue.

Download and configure the agent

Azure Pipelines

1. Log on to the machine using the account for which you've prepared permissions as explained above.

2. In your web browser, sign in to Azure Pipelines, and navigate to the **Agent pools** tab:

- Azure Pipelines: https://dev.azure.com/{your_organization}/_admin/_AgentPool
- TFS 2018: https://{your_server}/DefaultCollection/_admin/_AgentPool
- TFS 2017: https://{your_server}/tfs/DefaultCollection/_admin/_AgentPool
- TFS 2015: http://{your_server}:8080/tfs/_admin/_AgentPool

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

3. Click **Download agent**.

4. On the **Get agent** dialog box, click **Windows**.

5. On the left pane, select the processor architecture of your machine (x86 or x64).

6. On the right pane, click the **Download** button.

7. Follow the instructions on the page to download the agent.

8. Unpack the agent into the directory of your choice. Then run `config.cmd`.

TIP

Choose the processor according to the installed Windows OS version on your machine. The x64 agent version is intended for 64-bit Windows, whereas the x86 version is intended for 32-bit Windows. You can find which version of Windows your machine is running [by following these instructions](#).

TFS 2017 and TFS 2018

1. Log on to the machine using the account for which you've prepared permissions as explained above.

2. In your web browser, sign in to TFS, and navigate to the **Agent pools** tab:

- Azure Pipelines: https://dev.azure.com/{your_organization}/_admin/_AgentPool
- TFS 2018: https://{your_server}/DefaultCollection/_admin/_AgentPool
- TFS 2017: https://{your_server}/tfs/DefaultCollection/_admin/_AgentPool
- TFS 2015: http://{your_server}:8080/tfs/_admin/_AgentPool

The TFS URL doesn't work for me. How can I get the correct URL?

3. Click **Download agent**.
4. On the **Get agent** dialog box, click **Windows**.
5. Click the **Download** button.
6. Follow the instructions on the page to download the agent.
7. Unpack the agent into the directory of your choice. Then run `config.cmd`. Make sure that the path to the directory contains no spaces because tools and scripts don't always properly escape spaces.

NOTE

We recommend you configure the agent from an elevated Command Prompt.

Server URL and authentication

When setup asks for your server URL, for Azure DevOps Services, answer

`https://dev.azure.com/{your-organization}`.

When setup asks for your server URL, for TFS, answer `https://{your_server}/tfss`.

When setup asks for your authentication type, choose **PAT**. Then paste the [PAT token you created](#) into the command prompt window.

IMPORTANT

Make sure your server is [configured to support the authentication method](#) you want to use.

When you configure your agent to connect to TFS, you've got the following options:

- **Alternate** Connect to TFS using Basic authentication. After you select Alternate you'll be prompted for your credentials.
- **Negotiate** Connect to TFS as a user other than the signed-in user via a Windows authentication scheme such as NTLM or Kerberos. After you select Negotiate you'll be prompted for credentials.
- **Integrated** (Default) Connect a Windows agent to TFS using the credentials of the signed-in user via a Windows authentication scheme such as NTLM or Kerberos. You won't be prompted for credentials after you choose this method.
- **PAT** Supported only on Azure Pipelines and TFS 2017 and newer. After you choose PAT, paste the [PAT token you created](#) into the command prompt window. Use a personal access token (PAT) if your TFS instance and the agent machine are not in a trusted domain. PAT authentication is handled by your TFS instance instead of the domain controller.

NOTE

When using PAT as the authentication method, the PAT token is used only for the initial configuration of the agent. Learn more at [Communication with Azure Pipelines or TFS](#).

Choose interactive or service mode

For guidance on whether to run the agent in interactive mode or as a service, see [Agents: Interactive vs. service](#).

Note that if you configure as a service, the username you choose to run as should be 20 characters or less.

Run the agent

If you configured the agent to run interactively, to run it:

```
.\run.cmd
```

If you configured the agent to run as a service, it starts automatically. You can view and control the agent running status from the services snap-in. Run `services.msc` and look for "Azure Pipelines Agent (*name of your agent*)".

NOTE

If you need to change the agent's logon account, don't do it from the Services snap-in. Instead, see the information below to re-configure the agent.

To use your agent, run a [job](#) using the agent's pool. If you didn't choose a different pool, your agent will be in the **Default** pool.

Replace an agent

To replace an agent, follow the *Download and configure the agent* steps again.

When you configure an agent using the same name as an agent that already exists, you're asked if you want to replace the existing agent. If you answer `y`, then make sure you remove the agent (see below) that you're replacing. Otherwise, after a few minutes of conflicts, one of the agents will shut down.

Remove and re-configure an agent

To remove the agent:

```
.\config remove
```

After you've removed the agent, you can [configure it again](#).

Unattended config

The agent can be set up from a script with no human intervention. You must pass `--unattended` and the answers to all questions. Consult `.\config --help` for details about the required responses.

Help on other options

To learn about other options:

```
.\config --help
```

The help provides information on authentication alternatives and unattended configuration.

Capabilities

Your agent's capabilities are cataloged and advertised in the pool so that only the builds and releases it can handle are assigned to it. See [Build and release agent capabilities](#).

In many cases, after you deploy an agent, you'll need to install software or utilities. Generally you should install on your agents whatever software and tools you use on your development machine.

For example, if your build includes the [npm task](#), then the build won't run unless there's a build agent in the pool that has npm installed.

IMPORTANT

After you install new software on an agent, you must restart the agent for the new capability to show up in the pool so that the build can run.

Q & A

How do I make sure I have the latest v2 agent version?

1. Go to the *Agent pools* control panel tab:

- Azure Pipelines: https://dev.azure.com/{your_organization}/_admin/_AgentPool
- TFS 2018: https://{your_server}/DefaultCollection/_admin/_AgentPool
- TFS 2017: https://{your_server}/tfss/DefaultCollection/_admin/_AgentPool
- TFS 2015: http://{your_server}:8080/tfs/_admin/_AgentPool

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

2. Click the pool that contains the agent.

3. Make sure the agent is enabled.

4. Click **Agents**.

5. Click **Capabilities**.

6. Look for the `Agent.Version` capability.

You can check this value against the latest published agent version. See [Azure Pipelines Agent](#) and check the page for the highest version number listed.

7. Each agent automatically updates itself when it runs a task that requires a newer version of the agent. But if you want to manually update some agents, right-click the pool, and then choose **Update all agents**.

Can I update my v2 agents that are part of an Azure DevOps Server pool?

Yes. You can configure your Azure DevOps Server to look for the agent package files on a local disk, overriding the default version that came with the server (at the time of its release). This scenario also applies when the server does not have access to the Internet.

1. From a computer with Internet access, download the latest version of the agent package files (in .zip or .tar.gz form) from the [Azure Pipelines Agent GitHub Releases page](#).

2. Transfer the downloaded package files to each Azure DevOps Server Application Tier, via a method of your choice (e.g. USB drive, Network transfer). Place the agent files under the `%ProgramData%\Microsoft\Azure DevOps\Agents` folder. **Note:** in Azure DevOps Server 2019 RC1, the folder is called `%ProgramData%\Microsoft\DevOps\Agents`.

3. You're all set! Your Azure DevOps Server will now use the local files whenever the agents need to be updated. Each agent automatically updates itself when it runs a task that requires a newer version of the agent. But if you want to manually update some agents, right-click the pool, and then choose **Update all agents**.

What version of the agent runs with TFS 2017?

TFS VERSION	MINIMUM AGENT VERSION
2017 RTM	2.105.7
2017.3	2.112.0

I'm running a firewall and my code is in Azure Repos. What URLs does the agent need to communicate with?

If you're running an agent in a secure network behind a firewall, make sure the agent can initiate communication with the following URLs and IP addresses.

For organizations using the `*.visualstudio.com` domain:

```
https://login.microsoftonline.com
https://app.vssps.visualstudio.com
https://{organization_name}.visualstudio.com
https://{organization_name}.vsrm.visualstudio.com
https://{organization_name}.pkgs.visualstudio.com
https://{organization_name}.vssps.visualstudio.com
```

For organizations using the `dev.azure.com` domain:

```
https://dev.azure.com
https://*.dev.azure.com
https://login.microsoftonline.com
https://management.core.windows.net
```

To ensure your organization works with any existing firewall or IP restrictions, ensure that `dev.azure.com` is open and the `13.107.6.183` and `13.107.9.183` IP addresses are allowed.

How do I run the agent with self-signed certificate?

[Run the agent with self-signed certificate](#)

How do I run the agent behind a web proxy?

[Run the agent behind a web proxy](#)

How do I set different environment variables for each individual agent?

Create a `.env` file under agent's root directory and put environment variables you want to set into the file as following format:

```
MyEnv0=MyEnvValue0
MyEnv1=MyEnvValue1
MyEnv2=MyEnvValue2
MyEnv3=MyEnvValue3
MyEnv4=MyEnvValue4
```

How do I configure the agent to bypass a web proxy and connect to Azure Pipelines?

If you want the agent to bypass your proxy and connect to Azure Pipelines directly, then you should configure your web proxy to enable the agent to access the following URLs.

For organizations using the `*.visualstudio.com` domain:

```
https://login.microsoftonline.com  
https://app.vssps.visualstudio.com  
https://{{organization_name}}.visualstudio.com  
https://{{organization_name}}.vsrm.visualstudio.com  
https://{{organization_name}}.pkgs.visualstudio.com  
https://{{organization_name}}.vssps.visualstudio.com
```

For organizations using the `dev.azure.com` domain:

```
https://dev.azure.com  
https://*.dev.azure.com  
https://login.microsoftonline.com  
https://management.core.windows.net
```

To ensure your organization works with any existing firewall or IP restrictions, ensure that `dev.azure.com` is open and the `13.107.6.183` and `13.107.9.183` IP addresses are allowed.

NOTE

This procedure enables the agent to bypass a web proxy. Your build pipeline and scripts must still handle bypassing your web proxy for each task and tool you run in your build.

For example, if you are using a NuGet task, you must configure your web proxy to support bypassing the URL for the server that hosts the NuGet feed you're using.

I'm using TFS and the URLs in the sections above don't work for me. Where can I get help?

[Web site settings and security](#)

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Deploy an agent on Windows for TFS 2015

11/19/2018 • 6 minutes to read • [Edit Online](#)

[Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015 | Previous versions \(XAML builds\)](#)

To build and deploy Windows, Azure, and other Visual Studio solutions you may need a Windows agent. Windows agents can also build and deploy Java and Android apps.

Before you begin:

- If you use [Azure Pipelines](#) or TFS 2017 and newer, then you need to use a newer agent. See [Deploy an agent on Windows](#).
- If you use TFS, you might already have a build and release agent running. An agent is automatically or optionally deployed in some cases when you [set up Team Foundation Server](#).
- Otherwise, you've come to the right place to set up an agent on Windows for TFS 2015. Continue to the next section.

Learn about agents

If you already know what an agent is and how it works, feel free to jump right in to the following sections. But if you'd like some more background about what they do and how they work, see [Build and release agents](#).

Check prerequisites

Before you begin, make sure your agent machine is prepared with these prerequisites:

- An [operating system that is supported by Visual Studio 2013](#) or newer
- Visual Studio 2013 or Visual Studio 2015
- PowerShell 3 or newer ([Where can I get a newer version of PowerShell?](#))

Download and configure the agent

1. Make sure you're logged on the machine as an agent pool administrator. See [Agent pools](#).
2. Navigate to the **Agent pools** tab: `http://{your_server}:8080/tfs/_admin/_AgentPool`
3. Click **Download agent**.
4. Unzip the .zip file into the folder on disk from which you would like to run the agent. To avoid "path too long" issues on the file system, keep the path short. For example: `C:\Agent\`
5. Run Command Prompt as Administrator, and then run:

```
ConfigureAgent.cmd
```

6. Respond to the prompts.

Choose interactive or service mode

For guidance on whether to run the agent in interactive mode or as a service, see [Agents: Interactive vs. service](#).

[Run as a service](#)

If you chose to run the agent as a Windows service, then the agent running status can be controlled from the Services snap-in. Run services.msc and look for "VSO Agent (<name of your agent>)".

If you need to change the logon account, don't do it from the services snap-in. Instead, from an elevated Command Prompt, run:

```
C:\Agent\Agent\VsoAgent.exe /ChangeWindowsServiceAccount
```

Run interactively

If you chose to run interactively, then to run the agent:

```
C:\Agent\Agent\VsoAgent.exe
```

Command-line parameters

You can use command-line parameters when you configure the agent (`ConfigureAgent.cmd`) and when you run the agent (`Agent\VsoAgent.exe`). These are useful to avoid being prompted during unattended installation scripts and for power users.

Common parameters

`/Login:UserName,Password[;AuthType=(AAD|Basic|PAT)]`

Used for configuration commands against an Azure DevOps organization. The parameter is used to specify the pool administrator credentials. The credentials are used to perform the pool administration changes and are not used later by the agent.

When using personal access tokens (PAT) authentication type, specify anything for the user name and specify the PAT as the password.

If passing the parameter from PowerShell, be sure to escape the semicolon or encapsulate the entire argument in quotes. For example: '/Login:user,password;AuthType=PAT'. Otherwise the semicolon will be interpreted by PowerShell to indicate the end of one statement and the beginning of another.

`/NoPrompt`

Indicates not to prompt and to accept the default for any values not provided on the command-line.

`/WindowsServiceLogonAccount:WindowsServiceLogonAccount`

Used for configuration commands to specify the identity to use for the Windows service. To specify a domain account, use the form Domain\SAMAccountName or the user principal name (for example user@fabrikam.com). Alternatively a built-in account can be provided, for example `/WindowsServiceLogonAccount:"NT AUTHORITY\NETWORK SERVICE"`.

`/WindowsServiceLogonPassword:WindowsServiceLogonPassword`

Required if the `/WindowsServiceLogonAccount` parameter is provided.

`/Configure`

Configure supports the `/NoPrompt` switch for automated installation scenarios and will return a non-zero exit code on failure.

For troubleshooting configuration errors, detailed logs can be found in the _diag folder under the agent installation directory.

`/ServerUrl:ServerUrl`

The server URL should not contain the collection name. For example, `http://example:8080/tfs` or `https://dev.azure.com/example`

/Name:AgentName

The friendly name to identify the agent on the server.

/PoolName:PoolName

The pool that will contain the agent, for example: */PoolName:Default*

/WorkFolder:WorkFolder

The default work folder location is a _work folder directly under the agent installation directory. You can change the location to be outside of the agent installation directory, for example: */WorkFolder:C:_work*. One reason you may want to do this is to avoid "path too long" issues on the file system.

/Force

Replaces the server registration if a conflicting agent exists on the server. A conflict could be encountered based on the name. Or a conflict could be encountered if based on the ID a previously configured agent is being reconfigured in-place without unconfiguring first.

/NoStart

Used when configuring an interactive agent to indicate the agent should not be started after the configuration completes.

/RunningAsService

Used to indicate the agent should be configured to run as a Windows service.

/StartMode:(Automatic|Manual|Disabled)

/ChangeWindowsServiceAccount

Change Windows service account supports the */NoPrompt* switch for automated installation scenarios and will return a non-zero exit code on failure.

For troubleshooting errors, detailed logs can be found in the _diag folder under the agent installation directory.

/Unconfigure

/Version

Prints the version number.

/?

Prints usage information.

Capabilities

Your agent's capabilities are cataloged and advertised in the pool so that only the builds and releases it can handle are assigned to it. See [Build and release agent capabilities](#).

In many cases, after you deploy an agent, you'll need to install software or utilities. Generally you should install on your agents whatever software and tools you use on your development machine.

For example, if your build includes the [npm task](#), then the build won't run unless there's a build agent in the pool that has npm installed.

IMPORTANT

After you install new software on an agent, you must restart the agent for the new capability to show up in the pool so that the build can run.

Q & A

What version of PowerShell do I need? Where can I get a newer version?

The Windows Agent requires PowerShell version 3 or later. To check your PowerShell version:

```
$PSVersionTable.PSVersion
```

If you need a newer version of PowerShell, you can download it:

- PowerShell 3: Windows Management Framework 3.0
- PowerShell 4: Windows Management Framework 4.0
- PowerShell 5: Windows Management Framework 5.0

Why do I get this message when I try to queue my build?

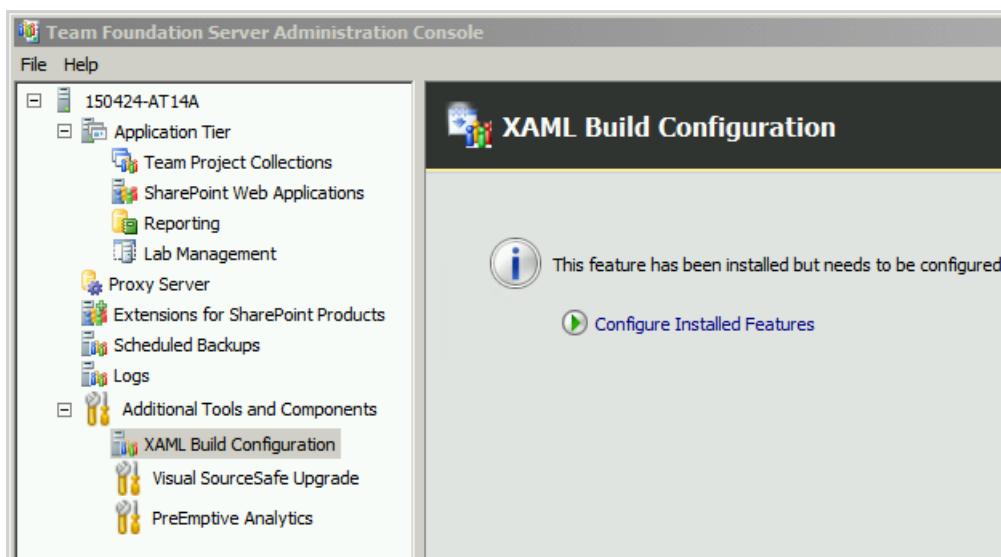
When you try to queue a build or a deployment, you might get a message such as: "No agent could be found with the following capabilities: msbuild, vsvisualstudio, vstest." One common cause of this problem is that you need to install prerequisite software such as Visual Studio on the machine where the build agent is running.

What version of the agent runs with my version of TFS?

TFS VERSION	AGENT VERSION
2015 RTM	1.83.2
2015.1	1.89.0
2015.2	1.95.1
2015.3	1.95.3

Can I still configure and use XAML build controllers and agents?

Yes. If you are an existing customer with custom build processes you are not yet ready to migrate, you can continue to use XAML builds, controllers, and agents.



I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Run a self-hosted agent behind a web proxy

10/24/2018 • 3 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called service endpoints in TFS 2018 and in older versions.

This topic explains how to run a v2 self-hosted agent behind a web proxy.

Azure Pipelines, TFS 2018 RTM and newer

(Applies to agent version 2.122 and newer.)

When your self-hosted agent is behind a web proxy, you can:

- Allow your agent to connect to Azure Pipelines or TFS behind a web proxy.
- Allow your agent to get sources in the build job and download artifacts in the release job.
- Develop custom vsts-task-lib tasks that leverage the proxy agent configuration.

To enable the agent to run behind a web proxy, pass `--proxyurl`, `--proxyusername` and `--proxypassword` during agent configuration.

For example:

- [Windows](#)
- [macOS and Linux](#)

```
./config.cmd --proxyurl http://127.0.0.1:8888 --proxyusername "myuser" --proxypassword "mypass"
```

We store your proxy credential securely on each platform. On Linux, the credential is encrypted with a symmetric key based on the machine ID. On macOS, we use the Keychain. On Windows, we use the Credential Store.

NOTE

Agent version 122.0, which shipped with TFS 2018 RTM, has a known issue configuring as a service on Windows. Because the Windows Credential Store is per user, you must configure the agent using the same user the service is going to run as. For example, in order to configure the agent service run as `mydomain\buildadmin`, you must launch `config.cmd` as `mydomain\buildadmin`. You can do that by logging into the machine with that user or using `Run as a different user` in the Windows shell.

How the agent handles the proxy within a build or release job

The agent will talk to Azure DevOps/TFS service through the web proxy specified in the `.proxy` file.

Since the code for the `Get Source` task in builds and `Download Artifact` task in releases are also baked into the agent, those tasks will follow the agent proxy configuration from the `.proxy` file.

The agent exposes proxy configuration via environment variables for every task execution. Task authors need to

use [\[vsts-task-lib\]\(https://github.com/Microsoft/azure-pipelines-task-lib\)](https://github.com/Microsoft/azure-pipelines-task-lib) methods to retrieve proxy configuration and [handle the proxy](#) within their task.

TFS 2017.2 and older

(You also can use this method for Azure Pipelines and newer versions of TFS, but we recommend the above method in those cases.)

In the agent root directory, create a .proxy file with your proxy server url.

- [Windows](#)
- [macOS and Linux](#)

```
echo http://name-of-your-proxy-server:8888 | Out-File .proxy
```

If your proxy doesn't require authentication, then you're ready to configure and run the agent. See [Deploy an agent on Windows](#).

NOTE

For backwards compatibility, if the proxy is not specified as described above, the agent also checks for a proxy URL from the VSTS_HTTP_PROXY environment variable.

Proxy authentication

If your proxy requires authentication, the simplest way to handle it is to grant permissions to the user under which the agent runs. Otherwise, you can provide credentials through environment variables. When you provide credentials through environment variables, the agent keeps the credentials secret by masking them in job and diagnostic logs. To grant credentials through environment variables, set the following variables:

- [Windows](#)
- [macOS and Linux](#)

```
$env:VSTS_HTTP_PROXY_USERNAME = "proxyuser"  
$env:VSTS_HTTP_PROXY_PASSWORD = "proxypassword"
```

NOTE

This procedure enables the agent infrastructure to operate behind a web proxy. Your build pipeline and scripts must still handle proxy configuration for each task and tool you run in your build. For example, if you are using a task that makes a REST API call, you must configure the proxy for that task.

Specify proxy bypass URLs

Create a [.proxybypass](#) file in the agent's root directory that specifies regular expressions (in ECMAScript syntax) to match URLs that should bypass the proxy. For example:

```
github\.com  
bitbucket\.com
```

Run the agent with a self-signed certificate

10/24/2018 • 3 minutes to read • [Edit Online](#)

[TFS 2018](#) | [TFS 2017](#)

This topic explains how to run a v2 self-hosted agent with self-signed certificate.

Work with SSL server certificate

```
Enter server URL > https://corp.tfs.com/tfs
Enter authentication type (press enter for Integrated) >
Connecting to server ...
An error occurred while sending the request.
```

Agent diagnostic log shows:

```
[2017-11-06 20:55:33Z ERR AgentServer] System.Net.Http.HttpRequestException: An error occurred while sending
the request. ---> System.Net.Http.WinHttpException: A security error occurred
```

This error may indicate the server certificate you used on your TFS server is not trusted by the build machine. Make sure you install your self-signed ssl server certificate into the OS certificate store.

```
Windows: Windows certificate store
Linux: OpenSSL certificate store
macOS: OpenSSL certificate store for agent version 2.124.0 or below
        Keychain for agent version 2.125.0 or above
```

You can easily verify whether the certificate has been installed correctly by running few commands. You should be good as long as SSL handshake finished correctly even you get a 401 for the request.

```
Windows: PowerShell Invoke-WebRequest -Uri https://corp.tfs.com/tfs -UseDefaultCredentials
Linux: curl -v https://corp.tfs.com/tfs
macOS: curl -v https://corp.tfs.com/tfs (agent version 2.124.0 or below, curl needs to be built for OpenSSL)
       curl -v https://corp.tfs.com/tfs (agent version 2.125.0 or above, curl needs to be built for Secure
Transport)
```

If somehow you can't successfully install certificate into your machine's certificate store due to various reasons, like: you don't have permission or you are on a customized Linux machine. The agent version 2.125.0 or above has the ability to ignore SSL server certificate validation error.

IMPORTANT

This is not secure and not recommended, we highly suggest you to install the certificate into your machine certificate store.

Pass `--sslskipcertvalidation` during agent configuration

```
./config.cmd/sh --sslskipcertvalidation
```

NOTE

There is limitation of using this flag on Linux and macOS

The libcurl library on your Linux or macOS machine needs to be built with OpenSSL, [More Detail](#)

Git get sources fails with SSL certificate problem (Windows agent only)

We ship command-line Git as part of the Windows agent. We use this copy of Git for all Git related operations. When you have a self-signed SSL certificate for your on-premises TFS server, make sure to configure the Git we shipped to allow that self-signed SSL certificate. There are 2 approaches to solve the problem.

1. Set the following git config in global level by the agent's run as user.

```
git config --global http."https://tfss.com/".sslCAInfo certificate.pem
```

NOTE

Setting system level Git config is not reliable on Windows. The system .gitconfig file is stored with the copy of Git we packaged, which will get replaced whenever the agent is upgraded to a new version.

2. Enable git to use SChannel during configuration with 2.129.0 or higher version agent Pass `--gituseschannel` during agent configuration

```
./config.cmd --gituseschannel
```

NOTE

Git SChannel has more restrictive requirements for your self-signed certificate. Self-signed certificate generated by IIS or PowerShell command may not be capable with SChannel.

Work with SSL client certificate

IIS has a SSL setting that requires all incoming requests to TFS must present client certificate in addition to the regular credential.

When that IIS SSL setting is enabled, you need to use `2.125.0` or above version agent and follow these extra steps in order to configure the build machine against your TFS server.

- Prepare all required certificate informations
 - CA certificate(s) in `.pem` format (This should contain the public key and signature of the CA certificate, you need put the root CA certificate and all your intermediate CA certificates into one `.pem` file)
 - Client certificate in `.pem` format (This should contain the public key and signature of the Client certificate)
 - Client certificate private key in `.pem` format (This should contain only the private key of the Client certificate)
 - Client certificate archive package in `.pfx` format (This should contain the signature, public key and private key of the Client certificate)
 - Use `SAME` password to protect Client certificate private key and Client certificate archive package, since they both have client certificate's private key
- Install CA certificate(s) into machine certificate store

- Linux: OpenSSL certificate store
- macOS: System or User Keychain
- Windows: Windows certificate store
- Pass `--sslcacert`, `--sslclientcert`, `--sslclientcertkey`, `--sslclientcertarchive` and `--sslclientcertpassword` during agent configuration.

```
.\"config.cmd/sh --sslcacert ca.pem --sslclientcert clientcert.pem --sslclientcertkey clientcert-key-pass.pem --sslclientcertarchive clientcert-archive.pfx --sslclientcertpassword "mypassword"
```

Your client certificate private key password is securely stored on each platform.

Linux: Encrypted with a symmetric key based on the machine ID
macOS: macOS Keychain
Windows: Windows Credential Store

Learn more about [agent client certificate support](#).

Build pipeline history

9/10/2018 • 2 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

From the **History** tab you can see a list of changes that includes who made the change and when the change occurred.

TFS 2017.3 and newer

To work with a change, select it, click  , and then click **Compare Difference** or **Revert Pipeline**.

TFS 2017 RTM

After you've viewed the history, if you want details about a change, select it and then choose **Diff**. If you want to roll back to an earlier version, select it, and then click **Rollback**.

Q & A

Can I edit the JSON source directly?

No

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Build pipeline options

10/29/2018 • 7 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

Here we explain settings you can change on the build pipeline **Options** tab.

Description

Team Foundation Server (TFS) 2017.1 and older

This section is available under **General tab**.

If you specify a description here, it is shown near the name of the build pipeline when you select it in the Build area of your project.

Build number format

- [YAML](#)
- [Designer](#)

In YAML, this property is called `name`. If not specified, your completed build is given a unique integer as its name. You can give completed builds much more useful names that are meaningful to your team. You can use a combination of tokens, variables, and underscore characters.

```
name: $(TeamProject)_$(BuildDefinitionName)_$(SourceBranchName)_$(Date:yyyyMMdd)$(Rev:.r)
steps:
- script: echo hello world
```

YAML builds are not yet available on TFS.

Example

At the time the build is queued:

- Project name: Fabrikam
- Build pipeline name: CIBuild
- Branch: master
- Build ID: 752
- Date: August 5, 2009.
- Time: 9:07:03 PM.
- The build ran once earlier today.

If you specify this build number format:

```
$(TeamProject)_$(BuildDefinitionName)_$(SourceBranchName)_$(Date:yyyyMMdd)$(Rev:.r)
```

Then the second completed build on this day would be named: **Fabrikam_CIBuild_master_20090805.2**

Tokens

The following table shows how each token is resolved based on the previous example.

TOKEN	EXAMPLE REPLACEMENT VALUE
<code>\$(BuildDefinitionName)</code>	CIBuild Note: The build pipeline name must not contain invalid or whitespace characters.
<code>\$(BuildID)</code>	752 \$(BuildID) is an internal immutable ID.
<code>\$(DayOfMonth)</code>	5
<code>\$(DayOfYear)</code>	217
<code>\$(Hours)</code>	21
<code>\$(Minutes)</code>	7
<code>\$(Month)</code>	8
<code>\$(Rev:.r)</code>	2 (The third build on this day will be 3, and so on.) Use \$(Rev:.r) to ensure that every completed build has a unique name. When a build is completed, if nothing else in the build number has changed, the Rev integer value is incremented by one. If you want to show prefix zeros in the number, you can add additional 'r' characters. For example, specify \$(rev:.rr) if you want the Rev number to begin with 01, 02, and so on.
<code>\$(Date:yyyyMMdd)</code>	20090824 You can specify other date formats such as \$(Date:MMddyy)
<code>\$(Seconds)</code>	3
<code>\$(SourceBranchName)</code>	master
<code>\$(TeamProject)</code>	Fabrikam
<code>\$(Year:yy)</code>	09
<code>\$(Year:yyyy)</code>	2009

TOKEN	EXAMPLE REPLACEMENT VALUE
-------	---------------------------

Variables

You can also use user-defined and predefined variables that have a scope of "All" in your build number format. For example, if you've defined `My.Variable`, you could specify the following build number format:

```
$(Build.DefinitionName)_$(Build.DefinitionVersion)_$(Build.RequestedFor)_$(Build.BuildId)_$(My.Variable)
```

The first four variables are predefined. `My.Variable` is defined by you on the [variables tab](#).

Badge enabled

TFS 2017.1 and older

This section is available under **General tab**.

Select if you want to show the latest outcome of this build on external web sites.

1. Select the **Badge enabled** check box.
2. Save the pipeline.
3. When the **Show URL** link appears, click it and copy the URL to your clipboard.
4. Use the URL as the source of an image in the HTML of the page of the external web site. For example:

```

```

Create a work item on failure

If the build pipeline fails, you can automatically create a work item to track getting the problem fixed. You can specify the work item type.

You can also select if you want to assign the work item to the requestor. For example, if this is a CI build, and a team member checks in some code that breaks the build, then the work item is assigned to that person.

Additional Fields: You can set the value of work item fields. For example:

FIELD	VALUE
<code>System.Title</code>	<code>Build \$(Build.BuildNumber) failed</code>
<code>System.Reason</code>	<code>Build failure</code>

Q: What other work item fields can I set? **A:** [Work item field index](#)

Allow scripts to access the OAuth token

Select this check box if you want to enable your script to use the build pipeline OAuth token.

For an example, see [Use a script to customize your build pipeline](#).

Default agent pool

TFS 2017.1 and older

This section is available under **General tab**.

Select the [pool](#) that's attached to the pool that contains the agents you want to run this pipeline.

Tip: If your code is in Azure Pipelines and you run your builds on Windows, in many cases the simplest option is to use the [Hosted pool](#).

Build job authorization scope

TFS 2017.1 and older

This section is available under **General tab**.

Specify the authorization scope for a build job. Select:

- **Project Collection** if the build needs access to multiple projects.
- **Current Project** if you want to restrict this build to have access only the resources in the current project.

Build job timeout in minutes

TFS 2017.1 and older

This section is available under **General tab**.

Specify the maximum time a build job is allowed to execute on an agent before being canceled by the server. Leave it empty or at zero if you want the job to never be canceled by the server.

Build job cancel timeout in minutes

Specify the maximum time a build job is allowed to respond after a user cancels the build. You can specify a value from 1 to 60 minutes.

Set this value to allow sufficient time for tasks to complete in cases where you've specified to **Run this task** as **Even if a previous task has failed, even if the build was cancelled** or as **Custom conditions** that allow a task to [always run](#) after cancellation.

Whatever value you set here, the **Build job timeout in minutes** limit still applies. For example:

- Build job timeout in minutes = 30
- Build job cancel timeout in minutes = 15
- A user clicks **Cancel** after the build has run for 25 minutes.
- The build is given 5 minutes to cancel instead of the 15 you specified.

NOTE

The system typically consumes about 10 seconds of this time allotment for messaging before your tasks run.

Demands

TFS 2017.1 and older

This section is available under **General tab**.

Use demands to make sure that the capabilities your build needs are present on the build agents that run it. Demands are asserted automatically by build tasks or manually by you.

Build task demands

Some build tasks won't run unless one or more demands are met by the build agent. For example, the [Visual Studio Build](#) task demands that `msbuild` and `visualstudio` are installed on the build agent. If your build includes tasks that have demands, they are listed first.

Manually entered demands

You might need to use on-premises build agents with special capabilities. For example, your build pipeline requires `SpecialSoftware`.

Add the demand to your build pipeline.

NAME	TYPE
SpecialSoftware	exists

Register each build agent that has the capability.

1. Go to the Agent pools tab at the root of the control panel:

- Azure Pipelines: https://dev.azure.com/{your_organization}/_admin/_AgentPool
- TFS 2018: https://{your_server}/DefaultCollection/_admin/_AgentPool
- TFS 2017: https://{your_server}/tfss/DefaultCollection/_admin/_AgentPool
- TFS 2015: http://{your_server}:8080/tfs/_admin/_AgentPool

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

2. Click the agent, and then click **Capabilities**

3. Add something like the following entry:

FIRST BOX	SECOND BOX
SpecialSoftware	C:\Program Files (x86)\SpecialSoftware

Tip: When you manually queue a build you can change the demands for that run.

Multi-configuration

Select this option to build multiple configurations. For example, you could build a C++ app for both debug and release configurations on both x86 and x64 platforms. In Azure Pipelines and TFS 2018, you set this option in the **Tasks** tab for each **Job** in your pipeline (not in the **Options** tab). To learn about multi-configuration, see the example [Build your C++ app for Windows](#).

Q & A

In what time zone are the build number time values expressed?

If you are using Azure Pipelines, then the time zone is UTC.

If you are using an on-premises Team Foundation Server, then the time zone is the same as the time zone of the operating system of the machine where you are running your application tier server.

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Build pipeline triggers

11/15/2018 • 9 minutes to read • [Edit Online](#)

[Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015 | Previous versions \(XAML builds\)](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

On the **Triggers** tab you specify the events that will trigger the build. You can use the same build pipeline for both CI and scheduled builds.

Continuous integration (CI)

- [YAML](#)
- [Designer](#)

YAML builds are configured by default with a CI trigger on all branches.

You can control which branches get CI triggers with a simple syntax:

```
trigger:  
- master  
- releases/*
```

You can specify the full name of the branch (for example, `master`) or a prefix-matching wildcard (for example, `releases/*`). You cannot put a wildcard in the middle of a value. For example, `releases/*2018` is invalid.

You can specify branches to include and exclude. For example:

```
# specific branch build  
trigger:  
branches:  
  include:  
    - master  
    - releases/*  
  exclude:  
    - releases/old*
```

If you have a lot of team members uploading changes often, then you might want to reduce the number of builds you're running. If you set `batch` to `true`, when a build is running, the system waits until the build is completed, then queues another build of all changes that have not yet been built.

```
# specific branch build with batching  
trigger:  
  batch: true  
  branches:  
    include:  
      - master
```

You can specify file paths to include or exclude.

```
# specific path build
trigger:
  branches:
    include:
      - master
      - releases/*
  paths:
    include:
      - docs/*
    exclude:
      - docs/README.md
```

You can opt out of CI builds entirely by specifying `trigger: none`.

```
# a build with no CI
trigger: none
```

YAML builds are not yet available on TFS.

Pull request validation

- [YAML](#)
- [Designer](#)

Unless you specify `pr` triggers in your YAML file, pull request builds are automatically enabled for all branches. You can specify the target branches for your pull request builds. For example, to run pull request builds only for branches that target: `master` and `releases/*`:

```
pr:
- master
- releases/*
```

You can specify the full name of the branch (for example, `master`) or a prefix-matching wildcard (for example, `releases/*`). You cannot put a wildcard in the middle of a value. For example, `releases/*2018` is invalid.

You can specify branches to include and exclude. For example:

```
# specific branch build
pr:
  branches:
    include:
      - master
      - releases/*
    exclude:
      - releases/old*
```

You can specify file paths to include or exclude. For example:

```
# specific path build
pr:
  branches:
    include:
      - master
      - releases/*
  paths:
    include:
      - docs/*
    exclude:
      - docs/README.md
```

You can opt out of pull request builds entirely by specifying `pr: none`.

```
# no PR builds
pr: none
```

YAML builds are not yet available on TFS.

Scheduled

- [YAML](#)
- [Designer](#)

Scheduled builds are not yet supported in YAML syntax. After you create your YAML build pipeline, you can use the designer to specify a scheduled trigger.

YAML builds are not yet available on TFS.

TFVC gated check-in

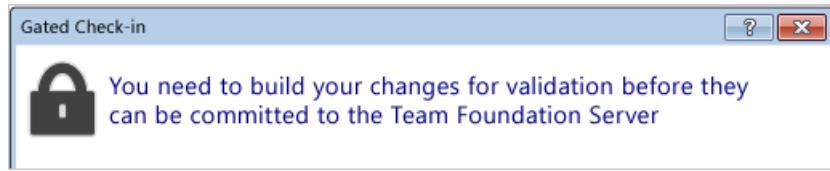
If your code is in a [Team Foundation version control \(TFVC\)](#) repo, use gated check-in to protect against breaking changes.

By default **Use workspace mappings for filters** is selected. Builds are triggered whenever a change is checked in under a path specified in your mappings in the [source repository settings](#).

Otherwise, you can clear this check box and specify the paths in the trigger.

How it affects your developers

When a developer tries to check-in, they are prompted to build their changes.



The system then creates a shelveset and builds it.

For details on the gated check-in experience, see [Check in to a folder that is controlled by a gated check-in build pipeline](#).

Option to run CI builds

By default, CI builds are not run after the gated check-in process is complete and the changes are checked in.

However, if you **do** want CI builds to run after a gated check-in, select the **Run CI triggers for committed changes** check box. When you do this, the build pipeline does not add *****NO_CI***** to the changeset

description. As a result, CI builds that are affected by the check-in are run.

A few other things to know

- Make sure the folders you include in your trigger are also included in your mappings on the [Repository tab](#).
- You can run gated builds on either a [Microsoft-hosted agent](#) or a [self-hosted agent](#).

Build completion triggers

- [YAML](#)
- [Designer](#)

Build completion triggers are not yet supported in YAML syntax. After you create your YAML build pipeline, you can use the designer to specify a build completion trigger.

Q & A

How do I protect my Git codebase from build breaks?

If your code is in a Git repo on Azure Repos or Team Foundation Server, you can create a branch policy that runs your build. See [Improve code quality with branch policies](#). This option is not available for GitHub repos.

My build didn't run. What happened?

Someone must view a page in your Azure DevOps organization regularly for CI and scheduled builds to run. It can be any page, including, for example, [Azure Pipelines](#).

Your Azure DevOps organization goes dormant five minutes after the last user signed out. After that, each of your build pipelines will run one more time. For example, while your organization is dormant:

- A nightly build of code in your Azure DevOps organization will run only one night until someone signs in again.
- CI builds of an external Git repo will stop running until someone signs in again.

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Predefined build variables

10/15/2018 • 46 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Some features and predefined variables are not available in certain versions of TFS. We're working on updating this topic to call out these differences.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Variables give you a convenient way to get key bits of data into various parts of your build pipeline. This is the comprehensive list of predefined build variables.

These variables are automatically set by the system and read-only. (The exceptions are `Build.Clean` and `System.Debug`.) Learn more about [working with variables](#).

Build.Clean

This is a deprecated variable that modifies how the build agent cleans up source. To learn how to clean up source, see [source repositories](#).

This variable modifies how the build agent cleans up source. To learn more, see [source repositories](#).

System.Debug

For more detailed logs to debug pipeline problems, define `System.Debug` and set it to `true`.

Agent variables

NOTE

You can use agent variables as environment variables in your scripts and as parameters in your build tasks. You cannot use them to customize the build number or to apply a version control label or tag.

VARIABLE	DESCRIPTION
Agent.BuildDirectory	The local path on the agent where all folders for a given build pipeline are created. For example: <code>c:\agent_work\1</code>
Agent.HomeDirectory	The directory the agent is installed into. This contains the agent software. For example: <code>c:\agent</code>
Agent.Id	The ID of the agent.
Agent.JobName	The display name of the running job.

Agent.JobStatus	<p>The status of the build.</p> <ul style="list-style-type: none"> • <code>Canceled</code> • <code>Failed</code> • <code>Succeeded</code> • <code>SucceededWithIssues</code> (partially successful) <p>The environment variable should be referenced as <code>AGENT_JOBSTATUS</code>. The older <code>agent.jobstatus</code> is available for backwards compatibility.</p>
Agent.MachineName	The name of the machine on which the agent is installed.
Agent.Name	<p>The name of the agent that is registered with the pool.</p> <p>If you are using a self-hosted agent, then this name is specified by you. See agents.</p>
Agent.OS	<p>The operating system of the agent host. Valid values are:</p> <ul style="list-style-type: none"> • Windows_NT • Darwin • Linux <p>If you're running in a container, the agent host and container may be running different operating systems.</p>
Agent.OSArchitecture	<p>The operating system processor architecture of the agent host. Valid values are:</p> <ul style="list-style-type: none"> • X86 • X64 • ARM
Agent.ToolsDirectory	<p>The directory used by tasks such as Node Tool Installer and Use Python Version to switch between multiple versions of a tool. These tasks will add tools from this directory to <code>PATH</code> so that subsequent build steps can use them.</p> <p>Learn about managing this directory on a self-hosted agent.</p>
Agent.WorkFolder	<p>The working directory for this agent. For example: <code>c:\agent_work</code>.</p>

Build variables

VARIABLE	DESCRIPTION

Build.ArtifactStagingDirectory	<p>The local path on the agent where any artifacts are copied to before being pushed to their destination. For example:</p> <pre>c:\agent_work\1\a</pre> <p>A typical way to use this folder is to publish your build artifacts with the Copy files and Publish build artifacts tasks.</p> <p>Note: Build.ArtifactStagingDirectory and Build.StagingDirectory are interchangeable. This directory is purged before each new build, so you don't have to clean it up yourself.</p> <p>See Artifacts in Azure Pipelines.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.BuildId	<p>The ID of the record for the completed build.</p>
Build.BuildNumber	<p>The name of the completed build. You can specify the build number format that generates this value in the pipeline options.</p> <p>A typical use of this variable is to make it part of the label format, which you specify on the repository tab.</p> <p>Note: This value can contain whitespace or other invalid label characters. In these cases, the label format will fail.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.BuildUri	<p>The URI for the build. For example:</p> <pre>vstfs:///Build/Build/1430</pre> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.BinariesDirectory	<p>The local path on the agent you can use as an output folder for compiled binaries.</p> <p>By default, new build pipelines are not set up to clean this directory. You can define your build to clean it up on the Repository tab.</p> <p>For example: <pre>c:\agent_work\1\b</pre>.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.DefinitionName	<p>The name of the build pipeline.</p> <p>Note: This value can contain whitespace or other invalid label characters. In these cases, the label format will fail.</p>
Build.DefinitionVersion	<p>The version of the build pipeline.</p>
Build.QueuedBy	<p>See "How are the identity variables set?".</p> <p>Note: This value can contain whitespace or other invalid label characters. In these cases, the label format will fail.</p>

Build.QueuedById	See " How are the identity variables set? ".
Build.Reason	<p>The event that caused the build to run.</p> <ul style="list-style-type: none"> • <code>Manual</code> : A user manually queued the build. • <code>IndividualCI</code> : Continuous integration (CI) triggered by a Git push or a TFVC check-in. • <code>BatchedCI</code> : Continuous integration (CI) triggered by a Git push or a TFVC check-in, and the Batch changes was selected. • <code>Schedule</code> : Scheduled trigger. • <code>ValidateShelveset</code> : A user manually queued the build of a specific TFVC shelveset. • <code>CheckInShelveset</code> : Gated check-in trigger. • <code>PullRequest</code> : The build was triggered by a Git branch policy that requires a build. • <code>BuildCompletion</code> : The build was triggered by another build <p>See Build pipeline triggers, Improve code quality with branch policies.</p>
Build.Repository.Clean	<p>The value you've selected for Clean in the source repository settings.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.Repository.LocalPath	<p>The local path on the agent where your source code files are downloaded. For example: <code>c:\agent_work\1\s</code></p> <p>By default, new build pipelines update only the changed files. You can modify how files are downloaded on the Repository tab.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p> <p>This variable is synonymous with <code>Build.SourcesDirectory</code>.</p>
Build.Repository.Name	<p>The name of the repository.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.Repository.Provider	<p>The type of repository you selected.</p> <ul style="list-style-type: none"> • <code>TfsGit</code> : TFS Git repository • <code>TfsVersionControl</code> : Team Foundation Version Control • <code>git</code> : Git repository hosted on an external server • <code>GitHub</code> • <code>Svn</code> : Subversion <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>

Build.Repository.Tfvc.Workspace	<p>Defined if your repository is Team Foundation Version Control. The name of the TFVC workspace used by the build agent.</p> <p>For example, if the Agent.BuildDirectory is <code>c:\agent_work\12</code> and the Agent.Id is <code>8</code>, the workspace name could be: <code>ws_12_8</code></p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.Repository.Uri	<p>The URL for the repository. For example:</p> <ul style="list-style-type: none"> Git: <code>https://dev.azure.com/fabrikamfiber/_git/Scripts</code> TFVC: <code>https://dev.azure.com/fabrikamfiber/</code> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.RequestedFor	<p>See "How are the identity variables set?".</p> <p>Note: This value can contain whitespace or other invalid label characters. In these cases, the label format will fail.</p>
Build.RequestedForEmail	<p>See "How are the identity variables set?".</p>
Build.RequestedForId	<p>See "How are the identity variables set?".</p>
Build.SourceBranch	<p>The branch the build was queued for. Some examples:</p> <ul style="list-style-type: none"> Git repo branch: <code>refs/heads/master</code> Git repo pull request: <code>refs/pull/1/merge</code> TFVC repo branch: <code>\$/teamproject/main</code> TFVC repo gated check-in: <code>Gated_2016-06-06_05.20.51.4369;username@live.com</code> TFVC repo shelveset build: <code>myshelveset;username@live.com</code> <p>When you use this variable in your build number format, the forward slash characters (<code>/</code>) are replaced with underscore characters (<code>_</code>).</p> <p>Note: In TFVC, if you are running a gated check-in build or manually building a shelveset, you cannot use this variable in your build number format.</p>
Build.SourceBranchName	<p>The name of the branch the build was queued for.</p> <ul style="list-style-type: none"> Git repo branch or pull request: The last path segment in the ref. For example, in <code>refs/heads/master</code> this value is <code>master</code>. TFVC repo branch: The last path segment in the root server path for the workspace. For example in <code>\$/teamproject/main</code> this value is <code>main</code>. TFVC repo gated check-in or shelveset build is the name of the shelveset. For example, <code>Gated_2016-06-06_05.20.51.4369;username@live.com</code> or <code>myshelveset;username@live.com</code>. <p>Note: In TFVC, if you are running a gated check-in build or manually building a shelveset, you cannot use this variable in your build number format.</p>

Build.SourcesDirectory	<p>The local path on the agent where your source code files are downloaded. For example: <code>c:\agent_work\1\s</code></p> <p>By default, new build pipelines update only the changed files. You can modify how files are downloaded on the Repository tab.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p> <p>This variable is synonymous with <code>Build.Repository.LocalPath</code>.</p>
Build.SourceVersion	<p>The latest version control change that is included in this build.</p> <ul style="list-style-type: none"> • Git: The commit ID. • TFVC: the changeset. <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.SourceVersionMessage	<p>The comment of the commit or changeset. Note: This variable is available in TFS 2015.4.</p>
Build.StagingDirectory	<p>The local path on the agent where any artifacts are copied to before being pushed to their destination. For example: <code>c:\agent_work\1\a</code></p> <p>A typical way to use this folder is to publish your build artifacts with the Copy files and Publish build artifacts tasks.</p> <p>Note: <code>Build.ArtifactStagingDirectory</code> and <code>Build.StagingDirectory</code> are interchangeable. This directory is purged before each new build, so you don't have to clean it up yourself.</p> <p>See Artifacts in Azure Pipelines.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.Repository.Git.SubmoduleCheckout	<p>The value you've selected for Checkout submodules on the repository tab.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.SourceTfcShelveset	<p>Defined if your repository is Team Foundation Version Control.</p> <p>If you are running a gated build or a shelveset build, this is set to the name of the shelveset you are building.</p> <p>Note: This variable yields a value that is invalid for build use in a build number format.</p>
Build.TriggeredBy.BuildId	<p>If the build was triggered by another build, then this variable is set to the <code>BuildID</code> of the triggering build.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>

Build.TriggeredBy.DefinitionId	If the build was triggered by another build, then this variable is set to the DefinitionID of the triggering build. This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.
Build.TriggeredBy.DefinitionName	If the build was triggered by another build, then this variable is set to the name of the triggering build pipeline. This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.
Build.TriggeredBy.BuildNumber	If the build was triggered by another build, then this variable is set to the number of the triggering build. This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.
Build.TriggeredBy.ProjectID	If the build was triggered by another build, then this variable is set to ID of the project that contains the triggering build. This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.
Common.TestResultsDirectory	The local path on the agent where the test results are created. For example: <code>c:\agent_work\1\TestResults</code> This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.

System variables

VARIABLE	DESCRIPTION
System.AccessToken	Use the OAuth token to access the REST API. Use System.AccessToken from YAML scripts. This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.
System.CollectionId	The GUID of the TFS collection or Azure DevOps organization
System.DefaultWorkingDirectory	The local path on the agent where your source code files are downloaded. For example: <code>c:\agent_work\1\s</code> By default, new build pipelines update only the changed files. You can modify how files are downloaded on the Repository tab . This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.
System.DefinitionId	The ID of the build pipeline.

System.HostType	Set to <code>build</code> if the pipeline is a build. For a release, the values are <code>deployment</code> for a Deployment group job and <code>release</code> for an Agent job.
System.PullRequest.IsFork	If the pull request is from a fork of the repository, this variable is set to <code>True</code> . Otherwise, it is set to <code>False</code> .
System.PullRequest.PullRequestId	The ID of the pull request that caused this build. For example: <code>17</code> . (This variable is initialized only if the build ran because of a Git PR affected by a branch policy .)
System.PullRequest.PullRequestNumber	The number of the pull request that caused this build. This variable is populated for pull requests from GitHub which have a different pull request ID and pull request number.
System.PullRequest.SourceBranch	The branch that is being reviewed in a pull request. For example: <code>refs/heads/users/raisa/new-feature</code> . (This variable is initialized only if the build ran because of a Git PR affected by a branch policy .)
System.PullRequest.SourceRepositoryURI	The URL to the repo that contains the pull request. For example: <code>https://dev.azure.com/ouraccount/_git/OurProject</code> . (This variable is initialized only if the build ran because of a Azure Repos Git PR affected by a branch policy . It is not initialized for GitHub PRs.)
System.PullRequest.TargetBranch	The branch that is the target of a pull request. For example: <code>refs/heads/master</code> . This variable is initialized only if the build ran because of a Git PR affected by a branch policy .
System.TeamFoundationCollectionUri	<p>The URL of the team foundation collection. For example: <code>https://dev.azure.com/fabrikamfiber/</code>.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
System.TeamProject	The name of the project that contains this build.
System.TeamProjectId	The ID of the project that this build belongs to.
TF_BUILD	<p>Set to <code>True</code> if the script is being run by a build task.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>

Agent variables

NOTE

You can use agent variables as environment variables in your scripts and as parameters in your build tasks. You cannot use them to customize the build number or to apply a version control label or tag.

VARIABLE	DESCRIPTION
----------	-------------

Agent.BuildDirectory	The local path on the agent where all folders for a given build pipeline are created. For example: <code>c:\agent_work\1</code>
Agent.HomeDirectory	The directory the agent is installed into. This contains the agent software. For example: <code>c:\agent</code> .
Agent.Id	The ID of the agent.
Agent.JobStatus	The status of the build. <ul style="list-style-type: none">• <code>Canceled</code>• <code>Failed</code>• <code>Succeeded</code>• <code>SucceededWithIssues</code> (partially successful) The environment variable should be referenced as <code>AGENT_JOBSTATUS</code> . The older <code>agent.jobstatus</code> is available for backwards compatibility.
Agent.MachineName	The name of the machine on which the agent is installed.
Agent.Name	The name of the agent that is registered with the pool. This name is specified by you. See agents .
Agent.ToolsDirectory	The directory used by tasks such as Node Tool Installer and Use Python Version to switch between multiple versions of a tool. These tasks will add tools from this directory to <code>PATH</code> so that subsequent build steps can use them. Learn about managing this directory on a self-hosted agent .
Agent.WorkFolder	The working directory for this agent. For example: <code>c:\agent_work</code> .

Build variables

VARIABLE	DESCRIPTION
Build.ArtifactStagingDirectory	The local path on the agent where any artifacts are copied to before being pushed to their destination. The local path on the agent where any artifacts are copied to before being pushed to their destination. For example: <code>c:\agent_work\1\a</code> A typical way to use this folder is to publish your build artifacts with the Copy files and Publish build artifacts tasks. Note: Build.ArtifactStagingDirectory and Build.StagingDirectory are interchangeable. This directory is purged before each new build, so you don't have to clean it up yourself. See Artifacts in Azure Pipelines . This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.

Build.BuildId	The ID of the record for the completed build.
Build.BuildNumber	<p>The name of the completed build. You can specify the build number format that generates this value in the pipeline options.</p> <p>A typical use of this variable is to make it part of the label format, which you specify on the repository tab.</p> <p>Note: This value can contain whitespace or other invalid label characters. In these cases, the label format will fail.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as a version control tag.</p>
Build.BuildUri	<p>The URI for the build. For example:</p> <pre>vstfs:///Build/Build/1430</pre> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.BinariesDirectory	<p>The local path on the agent you can use as an output folder for compiled binaries.</p> <p>By default, new build pipelines are not set up to clean this directory. You can define your build to clean it up on the Repository tab.</p> <p>For example:</p> <pre>c:\agent_work\1\b</pre> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.DefinitionName	The name of the build pipeline. Note: This value can contain whitespace or other invalid label characters. In these cases, the label format will fail.
Build.DefinitionVersion	The version of the build pipeline.
Build.QueuedBy	See " How are the identity variables set? ". Note: This value can contain whitespace or other invalid label characters. In these cases, the label format will fail.
Build.QueuedById	See " How are the identity variables set? ".

Build.Reason	<p>The event that caused the build to run.</p> <ul style="list-style-type: none"> • <code>Manual</code> : A user manually queued the build. • <code>IndividualCI</code> : Continuous integration (CI) triggered by a Git push or a TFVC check-in. • <code>BatchedCI</code> : Continuous integration (CI) triggered by a Git push or a TFVC check-in, and the Batch changes was selected. • <code>Schedule</code> : Scheduled trigger. • <code>ValidateShelveset</code> : A user manually queued the build of a specific TFVC shelveset. • <code>CheckInShelveset</code> : Gated check-in trigger. • <code>PullRequest</code> : The build was triggered by a Git branch policy that requires a build. <p>See Build pipeline triggers, Improve code quality with branch policies.</p>
Build.Repository.Clean	<p>The value you've selected for Clean in the source repository settings.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.Repository.LocalPath	<p>The local path on the agent where your source code files are downloaded. For example: <code>c:\agent_work\1\s</code></p> <p>By default, new build pipelines update only the changed files. You can modify how files are downloaded on the Repository tab.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p> <p>This variable is synonymous with <code>Build.SourcesDirectory</code>.</p>
Build.Repository.Name	<p>The name of the repository.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.Repository.Provider	<p>The type of repository you selected.</p> <ul style="list-style-type: none"> • <code>TfsGit</code> : TFS Git repository • <code>TfsVersionControl</code> : Team Foundation Version Control • <code>Git</code> : Git repository hosted on an external server • <code>Svn</code> : Subversion <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.Repository.Tfvc.Workspace	<p>Defined if your repository is Team Foundation Version Control. The name of the TFVC workspace used by the build agent.</p> <p>For example, if the <code>Agent.BuildDirectory</code> is <code>c:\agent_work\12</code> and the <code>Agent.Id</code> is <code>8</code>, the workspace name could be: <code>ws_12_8</code></p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>

Build.Repository.Uri	<p>The URL for the repository. For example:</p> <ul style="list-style-type: none"> • Git: <code>https://fabrikamfiber/tfs/DefaultCollection/Scripts/_git/Sc</code> • TFVC: <code>https://fabrikamfiber/tfs/DefaultCollection/</code> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.RequestedFor	<p>See "How are the identity variables set?". Note: This value can contain whitespace or other invalid label characters. In these cases, the label format will fail.</p>
Build.RequestedForEmail	<p>See "How are the identity variables set?".</p>
Build.RequestedForId	<p>See "How are the identity variables set?".</p>
Build.SourceBranch	<p>The branch the build was queued for. Some examples:</p> <ul style="list-style-type: none"> • Git repo branch: <code>refs/heads/master</code> • Git repo pull request: <code>refs/pull/1/merge</code> • TFVC repo branch: <code>\$/teamproject/main</code> • TFVC repo gated check-in: <code>Gated_2016-06-06_05.20.51.4369;username@live.com</code> • TFVC repo shelveset build: <code>myshelveset;username@live.com</code> <p>When you use this variable in your build number format, the forward slash characters (<code>/</code>) are replaced with underscore characters (<code>_</code>).</p> <p>Note: In TFVC, if you are running a gated check-in build or manually building a shelveset, you cannot use this variable in your build number format.</p>
Build.SourceBranchName	<p>The name of the branch the build was queued for.</p> <ul style="list-style-type: none"> • Git repo branch or pull request: The last path segment in the ref. For example, in <code>refs/heads/master</code> this value is <code>master</code>. • TFVC repo branch: The last path segment in the root server path for the workspace. For example in <code>\$/teamproject/main</code> this value is <code>main</code>. • TFVC repo gated check-in or shelveset build is the name of the shelveset. For example, <code>Gated_2016-06-06_05.20.51.4369;username@live.com</code> or <code>myshelveset;username@live.com</code>. <p>Note: In TFVC, if you are running a gated check-in build or manually building a shelveset, you cannot use this variable in your build number format.</p>

Build.SourcesDirectory	<p>The local path on the agent where your source code files are downloaded. For example: <code>c:\agent_work\1\s</code></p> <p>By default, new build pipelines update only the changed files. You can modify how files are downloaded on the Repository tab.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p> <p>This variable is synonymous with <code>Build.Repository.LocalPath</code>.</p>
Build.SourceVersion	<p>The latest version control change that is included in this build.</p> <ul style="list-style-type: none"> • Git: The commit ID. • TFVC: the changeset. <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.SourceVersionMessage	<p>The comment of the commit or changeset. Note: This variable is available in TFS 2015.4.</p>
Build.StagingDirectory	<p>The local path on the agent where any artifacts are copied to before being pushed to their destination. For example: <code>c:\agent_work\1\a</code></p> <p>A typical way to use this folder is to publish your build artifacts with the Copy files and Publish build artifacts tasks.</p> <p>Note: <code>Build.ArtifactStagingDirectory</code> and <code>Build.StagingDirectory</code> are interchangeable. This directory is purged before each new build, so you don't have to clean it up yourself.</p> <p>See Artifacts in Azure Pipelines.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.Repository.Git.SubmoduleCheckout	<p>The value you've selected for Checkout submodules on the repository tab.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.SourceTfcShelveset	<p>Defined if your repository is Team Foundation Version Control.</p> <p>If you are running a gated build or a shelveset build, this is set to the name of the shelveset you are building.</p> <p>Note: This variable yields a value that is invalid for build use in a build number format.</p>
Common.TestResultsDirectory	<p>The local path on the agent where the test results are created. For example: <code>c:\agent_work\1\TestResults</code></p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>

System variables

VARIABLE	DESCRIPTION
System.AccessToken	<p>Use the OAuth token to access the REST API.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
System.CollectionId	The GUID of the TFS collection or Azure DevOps organization
System.DefaultWorkingDirectory	<p>The local path on the agent where your source code files are downloaded. For example: <code>c:\agent_work\1\s</code></p> <p>By default, new build pipelines update only the changed files. You can modify how files are downloaded on the Repository tab.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
System.DefinitionId	The ID of the build pipeline.
System.HostType	Set to <code>build</code> if the pipeline is a build or <code>release</code> if the pipeline is a release.
System.PullRequest.IsFork	If the pull request is from a fork of the repository, this variable is set to <code>True</code> . Otherwise, it is set to <code>False</code> . Available in TFS 2018.2 .
System.PullRequest.PullRequestId	The ID of the pull request that caused this build. For example: <code>17</code> . (This variable is initialized only if the build ran because of a Git PR affected by a branch policy .)
System.PullRequest.SourceBranch	The branch that is being reviewed in a pull request. For example: <code>refs/heads/users/raisa/new-feature</code> . (This variable is initialized only if the build ran because of a Git PR affected by a branch policy .)
System.PullRequest.SourceRepositoryURI	The URL to the repo that contains the pull request. For example: <code>http://our-server:8080/tfs/DefaultCollection/_git/OurProject</code> . (This variable is initialized only if the build ran because of a Azure Repos Git PR affected by a branch policy .)
System.PullRequest.TargetBranch	The branch that is the target of a pull request. For example: <code>refs/heads/master</code> . This variable is initialized only if the build ran because of a Git PR affected by a branch policy .
System.TeamFoundationCollectionUri	<p>The URI of the team foundation collection. For example: <code>http://our-server:8080/tfs/DefaultCollection/</code>.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
System.TeamProject	The name of the project that contains this build.
System.TeamProjectId	The ID of the project that this build belongs to.

TF_BUILD

Set to `True` if the script is being run by a build task.

This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.

Agent variables

NOTE

You can use agent variables as environment variables in your scripts and as parameters in your build tasks. You cannot use them to customize the build number or to apply a version control label or tag.

VARIABLE	DESCRIPTION
Agent.BuildDirectory	The local path on the agent where all folders for a given build pipeline are created. For example: <code>c:\agent_work\1</code>
Agent.HomeDirectory	The directory the agent is installed into. This contains the agent software. For example: <code>c:\agent</code> .
Agent.Id	The ID of the agent.
Agent.JobStatus	The status of the build. <ul style="list-style-type: none">• <code>Canceled</code>• <code>Failed</code>• <code>Succeeded</code>• <code>SucceededWithIssues</code> (partially successful) The environment variable should be referenced as <code>AGENT_JOBSTATUS</code> . The older <code>agent.jobstatus</code> is available for backwards compatibility.
Agent.MachineName	The name of the machine on which the agent is installed.
Agent.Name	The name of the agent that is registered with the pool. This name is specified by you. See agents .
Agent.WorkFolder	The working directory for this agent. For example: <code>c:\agent_work</code> .

Build variables

VARIABLE	DESCRIPTION
----------	-------------

Build.ArtifactStagingDirectory	<p>The local path on the agent where any artifacts are copied to before being pushed to their destination.</p> <p>The local path on the agent where any artifacts are copied to before being pushed to their destination. For example:</p> <pre>c:\agent_work\1\a</pre> <p>A typical way to use this folder is to publish your build artifacts with the Copy files and Publish build artifacts tasks.</p> <p>Note: Build.ArtifactStagingDirectory and Build.StagingDirectory are interchangeable. This directory is purged before each new build, so you don't have to clean it up yourself.</p> <p>See Artifacts in Azure Pipelines.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.BuildId	The ID of the record for the completed build.
Build.BuildNumber	<p>The name of the completed build. You can specify the build number format that generates this value in the pipeline options.</p> <p>A typical use of this variable is to make it part of the label format, which you specify on the repository tab.</p> <p>Note: This value can contain whitespace or other invalid label characters. In these cases, the label format will fail.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as a version control tag.</p>
Build.BuildUri	<p>The URI for the build. For example:</p> <pre>vstfs:///Build/Build/1430</pre> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.BinariesDirectory	<p>The local path on the agent you can use as an output folder for compiled binaries.</p> <p>By default, new build pipelines are not set up to clean this directory. You can define your build to clean it up on the Repository tab.</p> <p>For example: <code>c:\agent_work\1\b</code>.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.DefinitionName	The name of the build pipeline. Note: This value can contain whitespace or other invalid label characters. In these cases, the label format will fail.
Build.DefinitionVersion	The version of the build pipeline.
Build.QueuedBy	See " How are the identity variables set? ". Note: This value can contain whitespace or other invalid label characters. In these cases, the label format will fail.

Build.QueuedById	See " How are the identity variables set? ".
Build.Reason	<p>The event that caused the build to run. Available in TFS 2017.3.</p> <ul style="list-style-type: none"> • <code>Manual</code> : A user manually queued the build. • <code>IndividualCI</code> : Continuous integration (CI) triggered by a Git push or a TFVC check-in. • <code>BatchedCI</code> : Continuous integration (CI) triggered by a Git push or a TFVC check-in, and the Batch changes was selected. • <code>Schedule</code> : Scheduled trigger. • <code>ValidateShelveset</code> : A user manually queued the build of a specific TFVC shelveset. • <code>CheckInShelveset</code> : Gated check-in trigger. • <code>PullRequest</code> : The build was triggered by a Git branch policy that requires a build. <p>See Build pipeline triggers, Improve code quality with branch policies.</p>
Build.Repository.Clean	<p>The value you've selected for Clean in the source repository settings.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.Repository.LocalPath	<p>The local path on the agent where your source code files are downloaded. For example: <code>c:\agent_work\1\s</code></p> <p>By default, new build pipelines update only the changed files. You can modify how files are downloaded on the Repository tab.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p> <p>This variable is synonymous with <code>Build.SourcesDirectory</code>.</p>
Build.Repository.Name	<p>The name of the repository.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.Repository.Provider	<p>The type of repository you selected.</p> <ul style="list-style-type: none"> • <code>TfsGit</code> : TFS Git repository • <code>TfsVersionControl</code> : Team Foundation Version Control • <code>Git</code> : Git repository hosted on an external server • <code>Svn</code> : Subversion <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>

Build.Repository.Tfvc.Workspace	<p>Defined if your repository is Team Foundation Version Control. The name of the TFVC workspace used by the build agent.</p> <p>For example, if the Agent.BuildDirectory is <code>c:\agent_work\12</code> and the Agent.Id is <code>8</code>, the workspace name could be: <code>ws_12_8</code></p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.Repository.Uri	<p>The URL for the repository. For example:</p> <ul style="list-style-type: none"> • Git: <code>https://fabrikamfiber/tfs/DefaultCollection/Scripts/_git/Scri</code> • TFVC: <code>https://fabrikamfiber/tfs/DefaultCollection/</code> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.RequestedFor	<p>See "How are the identity variables set?". Note: This value can contain whitespace or other invalid label characters. In these cases, the label format will fail.</p>
Build.RequestedForEmail	<p>See "How are the identity variables set?".</p>
Build.RequestedForId	<p>See "How are the identity variables set?".</p>
Build.SourceBranch	<p>The branch the build was queued for. Some examples:</p> <ul style="list-style-type: none"> • Git repo branch: <code>refs/heads/master</code> • Git repo pull request: <code>refs/pull/1/merge</code> • TFVC repo branch: <code>\$/teamproject/main</code> • TFVC repo gated check-in: <code>Gated_2016-06-06_05.20.51.4369;username@live.com</code> • TFVC repo shelveset build: <code>myshelveset;username@live.com</code> <p>When you use this variable in your build number format, the forward slash characters (<code>/</code>) are replaced with underscore characters (<code>_</code>).</p> <p>Note: In TFVC, if you are running a gated check-in build or manually building a shelveset, you cannot use this variable in your build number format.</p>
Build.SourceBranchName	<p>The name of the branch the build was queued for.</p> <ul style="list-style-type: none"> • Git repo branch or pull request: The last path segment in the ref. For example, in <code>refs/heads/master</code> this value is <code>master</code>. • TFVC repo branch: The last path segment in the root server path for the workspace. For example in <code>\$/teamproject/main</code> this value is <code>main</code>. • TFVC repo gated check-in or shelveset build is the name of the shelveset. For example, <code>Gated_2016-06-06_05.20.51.4369;username@live.com</code> or <code>myshelveset;username@live.com</code>. <p>Note: In TFVC, if you are running a gated check-in build or manually building a shelveset, you cannot use this variable in your build number format.</p>

Build.SourcesDirectory	<p>The local path on the agent where your source code files are downloaded. For example: <code>c:\agent_work\1\s</code></p> <p>By default, new build pipelines update only the changed files. You can modify how files are downloaded on the Repository tab.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p> <p>This variable is synonymous with <code>Build.Repository.LocalPath</code>.</p>
Build.SourceVersion	<p>The latest version control change that is included in this build.</p> <ul style="list-style-type: none"> • Git: The commit ID. • TFVC: the changeset. <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.SourceVersionMessage	<p>The comment of the commit or changeset. Note: This variable is available in TFS 2015.4.</p>
Build.StagingDirectory	<p>The local path on the agent where any artifacts are copied to before being pushed to their destination. For example: <code>c:\agent_work\1\a</code></p> <p>A typical way to use this folder is to publish your build artifacts with the Copy files and Publish build artifacts tasks.</p> <p>Note: <code>Build.ArtifactStagingDirectory</code> and <code>Build.StagingDirectory</code> are interchangeable. This directory is purged before each new build, so you don't have to clean it up yourself.</p> <p>See Artifacts in Azure Pipelines.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.Repository.Git.SubmoduleCheckout	<p>The value you've selected for Checkout submodules on the repository tab.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.SourceTfcShelveset	<p>Defined if your repository is Team Foundation Version Control.</p> <p>If you are running a gated build or a shelveset build, this is set to the name of the shelveset you are building.</p> <p>Note: This variable yields a value that is invalid for build use in a build number format.</p>
Common.TestResultsDirectory	<p>The local path on the agent where the test results are created. For example: <code>c:\agent_work\1\TestResults</code></p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>

System variables

VARIABLE	DESCRIPTION
System.AccessToken	Use the OAuth token to access the REST API.
System.CollectionId	The GUID of the TFS collection or Azure DevOps organization
System.DefaultWorkingDirectory	<p>The local path on the agent where your source code files are downloaded. For example: <code>c:\agent_work\1\s</code></p> <p>By default, new build pipelines update only the changed files. You can modify how files are downloaded on the Repository tab.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
System.DefinitionId	The ID of the build pipeline.
System.HostType	Set to <code>build</code> if the pipeline is a build or <code>release</code> if the pipeline is a release.
System.PullRequest.PullRequestId	The ID of the pull request that caused this build. For example: <code>17</code> . (This variable is initialized only if the build ran because of a Git PR affected by a branch policy .)
System.PullRequest.SourceBranch	The branch that is being reviewed in a pull request. For example: <code>refs/heads/users/raisa/new-feature</code> . (This variable is initialized only if the build ran because of a Git PR affected by a branch policy .)
System.PullRequest.SourceRepositoryURI	The URL to the repo that contains the pull request. For example: <code>http://our-server:8080/tfs/DefaultCollection/_git/OurProject</code> . (This variable is initialized only if the build ran because of a Azure Repos Git PR affected by a branch policy .)
System.PullRequest.TargetBranch	The branch that is the target of a pull request. For example: <code>refs/heads/master</code> . This variable is initialized only if the build ran because of a Git PR affected by a branch policy .
System.TeamFoundationCollectionUri	<p>The URI of the team foundation collection. For example: <code>http://our-server:8080/tfs/DefaultCollection/</code>.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
System.TeamProject	The name of the project that contains this build.
System.TeamProjectId	The ID of the project that this build belongs to.
TF_BUILD	<p>Set to <code>True</code> if the script is being run by a build task.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>

Agent variables

NOTE

You can use agent variables as environment variables in your scripts and as parameters in your build tasks. You cannot use them to customize the build number or to apply a version control label or tag.

VARIABLE	DESCRIPTION
Agent.BuildDirectory	<p>The local path on the agent where all folders for a given build pipeline are created.</p> <p>For example:</p> <ul style="list-style-type: none"> • TFS 2015.4: C:\TfsData\Agents\Agent-MACHINENAME_work\1 • TFS 2015 RTM user-installed agent: C:\Agent_work\6c3842c6 • TFS 2015 RTM built-in agent: C:\TfsData\Build_work\6c3842c6
Agent.HomeDirectory	<p>The directory the agent is installed into. This contains the agent software.</p> <p>For example:</p> <ul style="list-style-type: none"> • TFS 2015.4: C:\TfsData\Agents\Agent-MACHINENAME • TFS 2015 RTM user-installed agent: C:\Agent • TFS 2015 RTM built-in agent: C:\Program Files\Microsoft Team Foundation Server 14.0\Build
Agent.Id	The ID of the agent.
Agent.JobStatus	<p>The status of the build.</p> <ul style="list-style-type: none"> • Canceled • Failed • Succeeded • SucceededWithIssues (partially successful) <p>Note: The environment variable can be referenced only as <code>agent.jobstatus</code>. <code>AGENT_JOBSTATUS</code> was not present in TFS 2015.</p>
Agent.MachineName	The name of the machine on which the agent is installed. This variable is available in TFS 2015.4 , not in TFS 2015 RTM .
Agent.Name	<p>The name of the agent that is registered with the pool.</p> <p>This name is specified by you. See agents.</p>
Agent.WorkFolder	The working directory for this agent. For example: c:\agent_work

Build variables

VARIABLE	DESCRIPTION
----------	-------------

Build.ArtifactStagingDirectory	<p>The local path on the agent where any artifacts are copied to before being pushed to their destination.</p> <p>A typical way to use this folder is to publish your build artifacts with the Copy files and Publish build artifacts tasks. See Artifacts in Azure Pipelines.</p> <p>For example:</p> <ul style="list-style-type: none"> • TFS 2015.4: C:\TfsData\Agents\Agent-MACHINENAME_work\1\artifacts • TFS 2015 RTM default agent: C:\TfsData\Build_work\6c3842c6\artifacts • TFS 2015 RTM agent installed by you: C:\Agent_work\6c3842c6\artifacts <p>This directory is purged before each new build, so you don't have to clean it up yourself.</p> <p>In TFS 2015.4, Build.ArtifactStagingDirectory and Build.StagingDirectory are interchangeable.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.BuildId	The ID of the record for the completed build.
Build.BuildNumber	<p>The name of the completed build. You can specify the build number format that generates this value in the pipeline options. A typical use of this variable is to make it part of the label format, which you specify on the repository tab. Note: This value can contain whitespace or other invalid label characters. In these cases, the label format will fail.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of a version control tag.</p>
Build.BuildUri	<p>The URI for the build. For example: vstfs:///Build/Build/1430 .</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.BinariesDirectory	<p>The local path on the agent you can use as an output folder for compiled binaries. Available in TFS 2015.4.</p> <p>By default, new build pipelines are not set up to clean this directory. You can define your build to clean it up on the Repository tab.</p> <p>For example: C:\TfsData\Agents\Agent-MACHINENAME_work\1\b</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.DefinitionName	The name of the build pipeline. Note: This value can contain whitespace or other invalid label characters. In these cases, the label format will fail.
Build.DefinitionVersion	The version of the build pipeline.

Build.QueuedBy	See " How are the identity variables set? ". Note: This value can contain whitespace or other invalid label characters. In these cases, the label format will fail.
Build.QueuedById	See " How are the identity variables set? ".
Build.Repository.Clean	<p>The value you've selected for Clean in the source repository settings.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.Repository.LocalPath	<p>The local path on the agent where your source code files are downloaded. For example: <code>c:\agent_work\1\s</code></p> <p>By default, new build pipelines update only the changed files. You can modify how files are downloaded on the Repository tab.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p> <p>This variable is synonymous with Build.SourcesDirectory.</p>
Build.Repository.Name	<p>The name of the repository.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.Repository.Provider	<p>The type of repository you selected.</p> <ul style="list-style-type: none"> • <code>TfsGit</code> : TFS Git repository • <code>TfsVersionControl</code> : Team Foundation Version Control • <code>Git</code> : Git repository hosted on an external server • <code>Svn</code> : Subversion (available on TFS 2015.4) <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.Repository.Tfvc.Workspace	<p>Defined if your repository is Team Foundation Version Control. The name of the TFVC workspace used by the build agent.</p> <p>For example, if the Agent.BuildDirectory is <code>c:\agent_work\12</code> and the Agent.Id is <code>8</code>, the workspace name could be: <code>ws_12_8</code></p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.Repository.Uri	<p>The URL for the repository. For example:</p> <ul style="list-style-type: none"> • Git: <code>https://fabrikamfiber/tfs/DefaultCollection/Scripts/_git/Sci</code> • TFVC: <code>https://fabrikamfiber/tfs/DefaultCollection/</code> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>

Build.RequestedFor	See " How are the identity variables set? ". Note: This value can contain whitespace or other invalid label characters. In these cases, the label format will fail.
Build.RequestedForId	See " How are the identity variables set? ".
Build.SourceBranch	<p>The branch the build was queued for. Some examples:</p> <ul style="list-style-type: none"> Git repo branch: <code>refs/heads/master</code> Git repo pull request: <code>refs/pull/1/merge</code> TFVC repo branch: <code>\$/teamproject/main</code> TFVC repo gated check-in: <code>Gated_2016-06-06_05.20.51.4369;username@live.com</code> TFVC repo shelveset build: <code>myshelveset;username@live.com</code> <p>When you use this variable in your build number format, the forward slash characters (/) are replaced with underscore characters (_).</p> <p>Note: In TFVC, if you are running a gated check-in build or manually building a shelveset, you cannot use this variable in your build number format.</p>
Build.SourceBranchName	<p>The name of the branch the build was queued for.</p> <ul style="list-style-type: none"> Git repo branch or pull request: The last path segment in the ref. For example, in <code>refs/heads/master</code> this value is <code>master</code>. TFVC repo branch: The last path segment in the root server path for the workspace. For example in <code>\$/teamproject/main</code> this value is <code>main</code>. TFVC repo gated check-in or shelveset build is the name of the shelveset. For example, <code>Gated_2016-06-06_05.20.51.4369;username@live.com</code> or <code>myshelveset;username@live.com</code>. <p>Note: In TFVC, if you are running a gated check-in build or manually building a shelveset, you cannot use this variable in your build number format.</p>
Build.SourcesDirectory	<p>The local path on the agent where your source code files are downloaded. For example: <code>c:\agent_work\1\s</code></p> <p>By default, new build pipelines update only the changed files. You can modify how files are downloaded on the Repository tab.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p> <p>This variable is synonymous with <code>Build.Repository.LocalPath</code>.</p>
Build.SourcesDirectoryHash	Note: This variable is available in TFS 2015 RTM, but not in TFS 2015.4.
Build.SourceVersion	<p>The latest version control change that is included in this build.</p> <ul style="list-style-type: none"> Git: The commit ID. TFVC: the changeset. <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>

Build.SourceVersionMessage	The comment of the commit or changeset. Note: This variable is available in TFS 2015.4.
Build.StagingDirectory	<p>TFS 2015 RTM</p> <p>The local path on the agent you can use as an output folder for compiled binaries. For example: <code>C:\TfsData\Build_work\6c3842c6\staging</code>.</p> <p>By default, new build pipelines are not set up to clean this directory. You can define your build to clean it up on the Repository tab.</p> <p>TFS 2015.4</p> <p>The local path on the agent where any artifacts are copied to before being pushed to their destination. For example: <code>C:\TfsData\Agents\Agent-MACHINENAME_work\1\a</code></p> <p>This directory is purged before each new build, so you don't have to clean it up yourself.</p> <p>A typical way to use this folder is to publish your build artifacts with the Copy files and Publish build artifacts tasks. See Artifacts in Azure Pipelines.</p> <p>In TFS 2015.4, Build.ArtifactStagingDirectory and Build.StagingDirectory are interchangeable.</p> <p>All versions of TFS 2015</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.Repository.Git.SubmoduleCheckout	<p>The value you've selected for Checkout submodules on the repository tab.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
Build.SourceTfvcShelveset	<p>Defined if your repository is Team Foundation Version Control.</p> <p>If you are running a gated build or a shelveset build, this is set to the name of the shelveset you are building.</p> <p>Note: This variable yields a value that is invalid for build use in a build number format.</p>
Common.TestResultsDirectory	<p>The local path on the agent where the test results are created. For example: <code>c:\agent_work\1\TestResults</code>. Available in TFS 2015.4.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>

System variables

VARIABLE	DESCRIPTION

System.AccessToken	<p>Available in TFS 2015.4. Use the OAuth token to access the REST API.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
System.CollectionId	The GUID of the TFS collection or Azure DevOps organization
System.DefaultWorkingDirectory	<p>The local path on the agent where your source code files are downloaded. For example: <code>c:\agent_work\1\s</code></p> <p>By default, new build pipelines update only the changed files. You can modify how files are downloaded on the Repository tab.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
System.DefinitionId	The ID of the build pipeline.
System.HostType	Set to <code>build</code> if the pipeline is a build or <code>release</code> if the pipeline is a release.
System.PullRequest.PullRequestId	The ID of the pull request that caused this build. For example: <code>17</code> . (This variable is initialized only if the build ran because of a Git PR affected by a branch policy .)
System.PullRequest.SourceBranch	The branch that is being reviewed in a pull request. For example: <code>refs/heads/users/raisa/new-feature</code> . (This variable is initialized only if the build ran because of a Git PR affected by a branch policy .)
System.PullRequest.SourceRepositoryURI	The URL to the repo that contains the pull request. For example: <code>http://our-server:8080/tfs/DefaultCollection/_git/OurProject</code> . (This variable is initialized only if the build ran because of a Azure Repos Git PR affected by a branch policy .)
System.PullRequest.TargetBranch	The branch that is the target of a pull request. For example: <code>refs/heads/master</code> . This variable is initialized only if the build ran because of a Git PR affected by a branch policy .
System.TeamFoundationCollectionUri	<p>The URI of the team foundation collection. For example: <code>http://our-server:8080/tfs/DefaultCollection/</code>.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>
System.TeamProject	The name of the project that contains this build.
System.TeamProjectId	The ID of the project that this build belongs to.
TF_BUILD	<p>Set to <code>True</code> if the script is being run by a build task.</p> <p>This variable is agent-scoped. It can be used as an environment variable in a script and as a parameter in a build task, but not as part of the build number or as a version control tag.</p>

How are the identity variables set?

The value depends on what caused the build.

IF THE BUILD IS TRIGGERED...	THEN THE BUILD.QUEUEDBY AND BUILD.QUEUEDBYID VALUES ARE BASED ON...	THEN THE BUILD.REQUESTEDFOR AND BUILD.REQUESTEDFORID VALUES ARE BASED ON...
In Git or TFVC by the Continuous integration (CI) triggers	The system identity, for example: [DefaultCollection]\Project Collection Service Accounts	The person who pushed or checked in the changes.
In Git or by a branch policy build .	The system identity, for example: [DefaultCollection]\Project Collection Service Accounts	The person who checked in the changes.
In TFVC by a gated check-in trigger	The person who checked in the changes.	The person who checked in the changes.
In Git or TFVC by the Scheduled triggers	The system identity, for example: [DefaultCollection]\Project Collection Service Accounts	The system identity, for example: [DefaultCollection]\Project Collection Service Accounts
Because you clicked the Queue build button	You	You

What is Azure Pipelines release service?

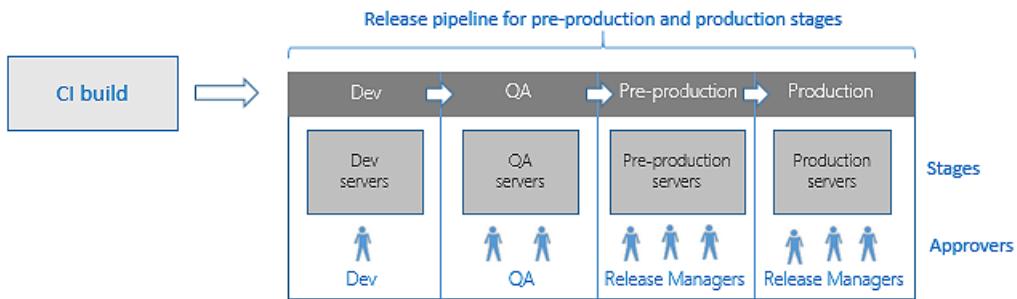
10/9/2018 • 4 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

Azure Pipelines releases is a service in Azure Pipelines and Team Foundation Server (TFS 2015.2 and later) and an essential element of DevOps CI/CD that helps your team **continuously deliver** software to your customers at a faster pace and with lower risk. You can **fully automate** the testing and delivery of your software in multiple stages all the way to production, or set up semi-automated processes with **approvals** and **on-demand deployments**.



1. [Watch this video](#) - see Azure Pipelines releases in action.

<https://www.youtube.com/embed/zSPuRXTeZW8>

2. [Decide if it suits your scenarios](#) - use the simple checklist.
3. [See how it works](#) - get a basic understanding of the process.
4. [Get started now](#) - follow the steps to deploy your apps.

Is Azure Pipelines release service for you?

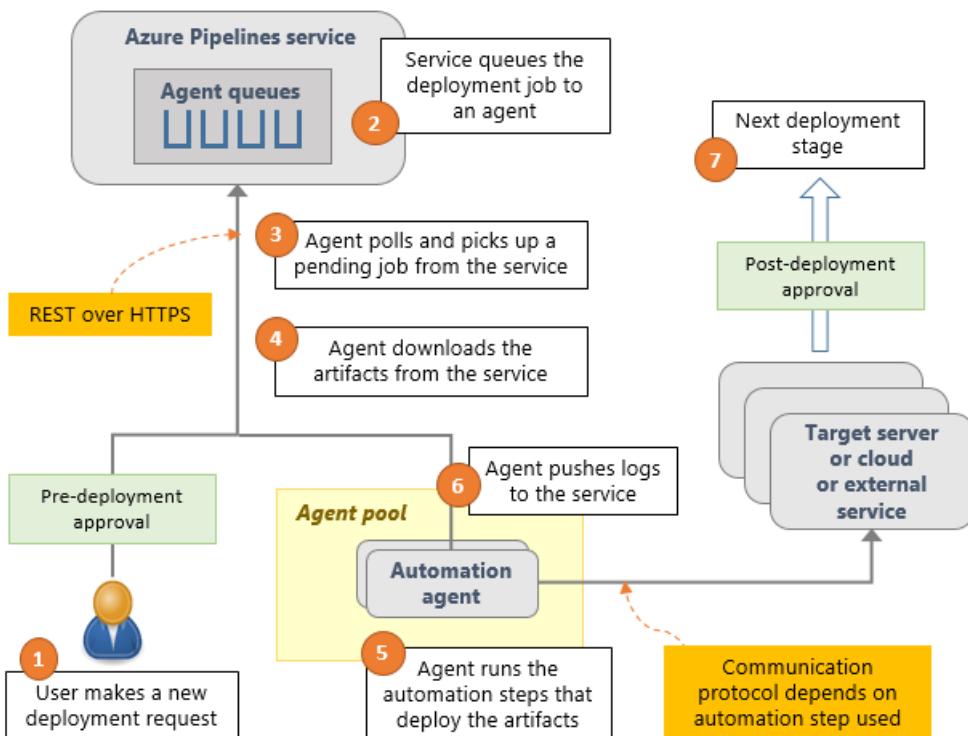
Consider using Azure Pipelines releases if:

- **You develop applications and need to deploy them regularly to any platform**, public or private cloud services, or App stores. Azure Pipelines has many out-of-the-box tasks to deploy a variety of applications. If you cannot find an out-of-the-box task to deploy your application using Azure Pipelines, consider this: if you can script the deployment of your application using Shell scripts or PowerShell scripts, utilities such as Ant or Maven, batch files or EXE utilities, then you can deploy it using Azure Pipelines. It also integrates with third party deployment systems such as Chef and Docker.
- **You use a continuous integration (CI) system** and are looking for a fully-fledged continuous delivery management or release system. Whether you use Azure Pipelines or TFS, or Jenkins as your CI system, you can set up Azure Pipelines releases to automatically deploy new builds to multiple stages. Even if we do not yet support integration with your favorite CI system or artifact repository, you can still write custom tasks to download and deploy artifacts from it.

- **You need to track the progress of releases.** If you use several stages for your tests, Azure Pipelines release service helps you monitor whether a release has been deployed and tested on each of these stages. It also tracks whether an issue fixed by a developer, or a product backlog item completed by your team, has been deployed to a specific stage.
- **You need control of the deployments.** Azure Pipelines release service lets you specify which users can change the configuration of a stage, or approve the release to be deployed into a particular stage. If there is a problem with your deployment, Azure Pipelines helps you roll back to a previous deployment, and provide all the logs in one place to help you debug the problem.
- **You need audit history for all releases and their deployments.** Azure Pipelines release service provides a history of all changes to the pipelines, configurations, and deployments. It also provides a history of all the activity performed during each deployment. Each release is accompanied by a listing of new features and developer commits that went into that release.

How does Azure Pipelines release service work?

The Azure Pipelines release service stores the data about your release pipelines, stages, tasks, releases, and deployments in Azure Pipelines or TFS.



Azure Pipelines runs the following steps as part of every deployment:

- Pre-deployment approval:** When a new deployment request is triggered, Azure Pipelines checks whether a pre-deployment approval is required before deploying a release to a stage. If it is required, it sends out email notifications to the appropriate approvers.
- Queue deployment job:** Azure Pipelines schedules the deployment job on an available [automation agent](#). An agent is a piece of software that is capable of running tasks in the deployment.
- Agent selection:** An automation agent picks up the job. The agents for Azure Pipelines releases are exactly the same as those that run your builds in Azure Pipelines and TFS. A release pipeline can contain settings to select an appropriate agent at runtime.
- Download artifacts:** The agent downloads all the artifacts specified in that release (provided you have not opted to skip the download). The agent currently understands two types of artifacts: Azure Pipelines artifacts

and Jenkins artifacts.

5. **Run the deployment tasks:** The agent then runs all the tasks in the deployment job to deploy the app to the target servers for a stage.
6. **Generate progress logs:** The agent creates detailed logs for each step while running the deployment, and pushes these logs back to Azure Pipelines or TFS.
7. **Post-deployment approval:** When deployment to a stage is complete, Azure Pipelines checks if there is a post-deployment approval required for that stage. If no approval is required, or upon completion of a required approval, it proceeds to trigger deployment to the next stage.

Get started now!

Simply follow these steps:

1. [Create your first pipeline](#)
2. [Set up a multi-stage managed release pipeline](#)
3. [Manage deployments by using approvals and gates](#)

Related topics

- [Download Team Foundation Server](#)
- [Install and configure Team Foundation Server](#)
- [Sign up for Azure Pipelines](#)

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Release pipelines and release names

10/9/2018 • 3 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called service endpoints in TFS 2018 and in older versions.

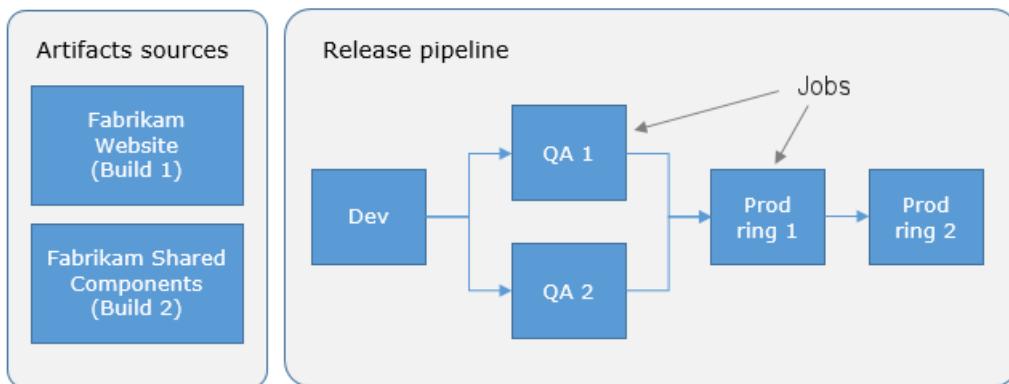
A **release pipeline** is one of the fundamental concepts in Azure Pipelines for your DevOps CI/CD processes. It defines the end-to-end release pipeline for an application to be deployed across various stages.

You start using Azure Pipelines releases by authoring a release pipeline for your application. To author a release pipeline, you must specify the [artifacts](#) that make up the application and the **release pipeline**.

An **artifact** is a deployable component of your application. It is typically produced through a Continuous Integration or a build pipeline. Azure Pipelines releases can deploy artifacts that are produced by a [wide range of artifact sources](#) such as Azure Pipelines build, Jenkins, or Team City.

You define the **release pipeline** using [stages](#), and restrict deployments into or out of a stage using [approvals](#). You define the automation in each stage using [jobs](#) and [tasks](#). You use [variables](#) to generalize your automation and [triggers](#) to control when the deployments should be kicked off automatically.

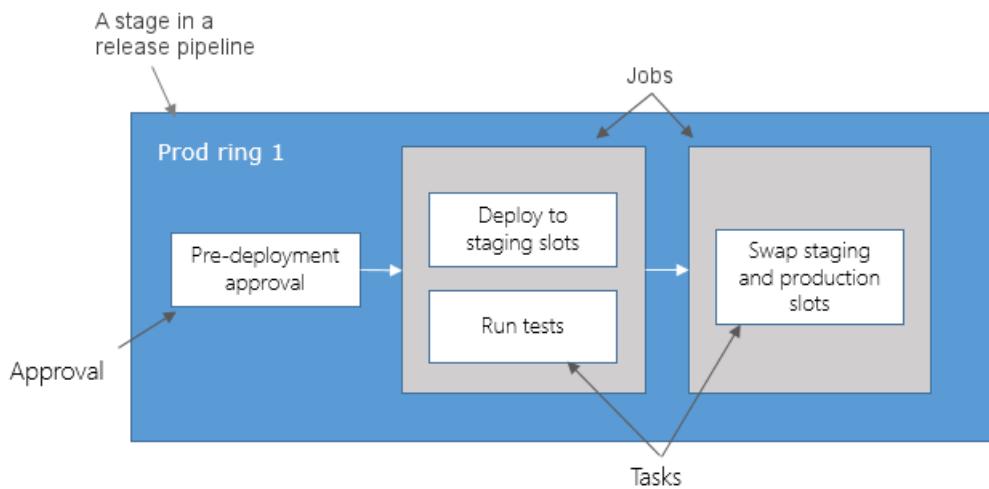
An example of a release pipeline that can be modeled through a release pipeline is shown below:



[What's the difference between a release pipeline and a release?](#)

In this example, a release of a website is created by collecting specific versions of two builds (artifacts), each from a different build pipeline. The release is first deployed to a Dev stage and then forked to two QA stages in parallel. If the deployment succeeds in both the QA stages, the release is deployed to Prod ring 1 and then to Prod ring 2. Each production ring represents multiple instances of the same website deployed at various locations around the globe.

An example of how deployment automation can be modeled within a stage is shown below:



In this example, a **job** is used to deploy the app to websites across the globe in parallel within production ring 1. After all those deployments are successful, a second job is used to switch traffic from the previous version to the newer version.

TFS 2015: Jobs, and fork and join deployments, are not available in TFS 2015.

Besides the release pipeline, release pipelines have a few options that can be customized: [release names](#) and [retention policies](#).

Release names

The names of releases for a release pipeline are, by default, sequentially numbered. The first release is named **Release-1**, the next release is **Release-2**, and so on. You can change this naming scheme by editing the release name format mask. In the **Options** tab of a release pipeline, edit the **Release name format** property.

When specifying the format mask, you can use the following pre-defined variables.

VARIABLE	DESCRIPTION
Rev:r	An auto-incremented number with at least the specified number of digits.
Date / Date:MMddyy	The current date, with the default format MMddyy . Any combinations of M/MM/MMM/MMMM, d/dd/ddd/ddd, y/yy/yyyy/yyy, h/hh/H/HH, m/mm, s/ss are supported.
System.TeamProject	The name of the project to which this build belongs.
Release.ReleaseId	The ID of the release, which is unique across all releases in the project.
Release.DefinitionName	The name of the release pipeline to which the current release belongs.
Build.BuildNumber	The number of the build contained in the release. If a release has multiple builds, this is the number of the primary build .
Build.DefinitionName	The pipeline name of the build contained in the release. If a release has multiple builds, this is the pipeline name of the primary build .

VARIABLE	DESCRIPTION
Artifact.ArtifactType	The type of the artifact source linked with the release. For example, this can be Azure Pipelines or Jenkins .
Build.SourceBranch	The branch of the primary artifact source . For Git, this is of the form master if the branch is refs/heads/master . For Team Foundation Version Control, this is of the form branch if the root server path for the workspace is \$/teamproject/branch . This variable is not set for Jenkins or other artifact sources.
<i>Custom variable</i>	The value of a global configuration property defined in the release pipeline.

For example, the release name format `Release $(Rev:rrr) for build $(Build.BuildNumber) $(Build.DefinitionName)` will create releases with names such as **Release 002 for build 20170213.2 MySampleAppBuild**.

Release retention

You can customize how long releases of this pipeline must be retained. For more information, see [release retention](#).

Release history

Every time you save a release pipeline, Azure Pipelines keeps a copy of the changes. This allows you to compare the changes at a later point, especially when you are debugging a deployment failure.

Related topics

- [Artifacts](#)
- [Stages](#)
- [Triggers](#)
- [Variables](#)
- [Release retention](#)
- [Release security](#)

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Release stages, queuing policies, and options

10/9/2018 • 7 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

A **stage** is a *logical* and *independent* entity that represents where you want to deploy a release generated from a release pipeline. We'll examine these two characteristics in more detail to help you understand how to divide your release pipeline into stages in your DevOps CI/CD processes.

First, a stage in a release pipeline is a **logical** entity. 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.

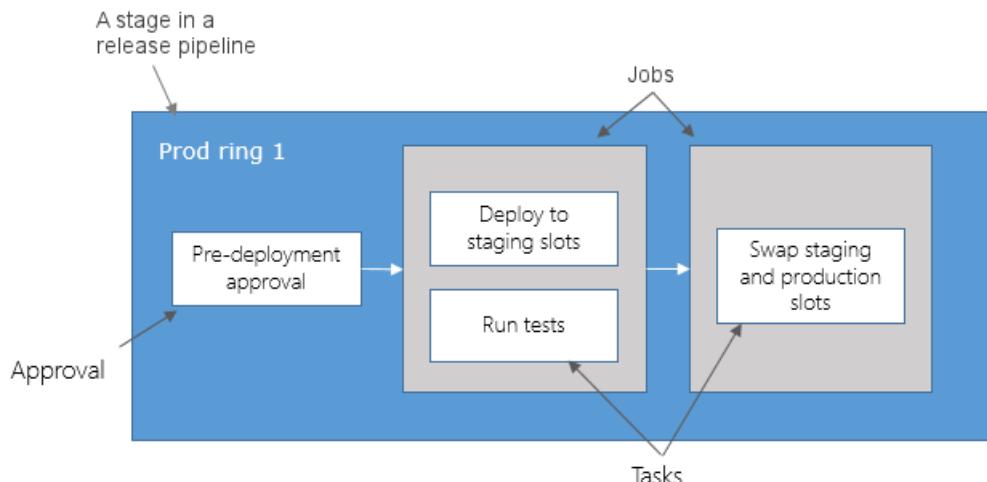
Second, you must be able to deploy to a stage **independently** of other stages in the pipeline. For example, your pipeline might consist of two stages A and B, and Azure Pipelines could deploy Release 2 to A and Release 1 to B. If you make any assumptions in B about the existence of a certain release 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.

The deployment pipeline of a release to a stage is defined in terms of [jobs](#) and [tasks](#). The physical deployment of a release to a stage is controlled through [approvals and gates](#), [deployment conditions and triggers](#), and [queuing policies](#).



Queuing policies

In some cases, you may be generating builds more quickly than they can be deployed. Alternatively, you may configure multiple [agents](#) and, for example, be creating releases from the same release pipeline for deployment of different artifacts. In such cases, it's useful to be able to control how multiple releases are queued into a stage. **Queuing policies** give you that control.

Stages | + Add ▾

Deployment queue settings ▾
Define behavior when multiple releases are queued for deployment

Number of parallel deployments ⓘ

Specific Unlimited

Maximum number of parallel deployments
5

Subsequent releases

Deploy all in sequence
 Deploy latest and cancel the others

The options you can choose for a queuing policy are:

- **Number of parallel deployments:** Use this option if you dynamically provision new resources in your stage and it is physically capable of handling the deployment of multiple releases in parallel, but you want to limit the number of parallel deployments.
- If you specify a maximum number of deployments, two more options appear:
 - **Deploy all in sequence:** Use this option if you want to deploy all the releases sequentially into the same shared physical resources. By deploying them in turn, one after the other, you ensure that two

deployment jobs do not target the same physical resources concurrently, even if there are multiple build and release agents available. You also ensure that pre-deployment approval requests for the stage are sent out in sequence.

- **Deploy latest and cancel the others:** Use this option if you are producing releases faster than builds, and you only want to deploy the latest build.

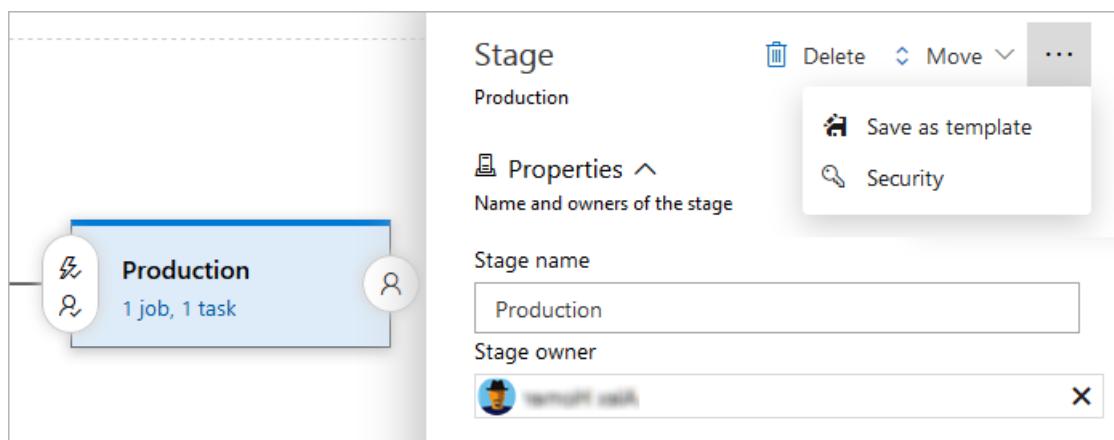
To understand how these options work, consider a scenario where releases **R1, R2, ..., R5** of a single release pipeline are created in quick succession due to new builds being produced rapidly. Assume that the first stage in this pipeline is named **QA** and has both pre-deployment and post-deployment approvers defined.

- If you do not specify a limit for the number of parallel deployments, all five approval requests will be sent out as soon as the releases are created. If the approvers grant approval for all of the releases, they will all be deployed to the **QA** stage in parallel. (if the **QA** stage did not have any pre-deployment approvers defined, all the five releases will automatically be deployed in parallel to this stage).
- If you specify a limit and **Deploy all in sequence**, and the limit has already been reached, the pre-deployment approval for release **R1** will be sent out first. After this approval is completed, the deployment of release **R1** to the **QA** stage begins. Next, a request for post-deployment approval is sent out for release **R1**. It is only after this post-deployment approval is completed that execution of release **R2** begins and its pre-deployment approval is sent out. The process continues like this for all of the releases in turn.
- If you specify a limit and **Deploy latest and cancel the others**, and the limit has already been reached, releases **R2, R3, and R4** will be skipped, and the pre-deployment approval for **R5** in the **QA** stage will be sent out immediately after the post-deployment approval for release **R1** is completed.

Stage general options

While the most important part of defining a stage is the automation tasks, you can also configure several properties and options for a stage in a release pipeline. You can:

- Edit the name of the stage here if required.
- Designate a single user or a single group to be the stage owner. Stage owners are notified whenever a deployment of a release is completed to that stage. Stage owners are not automatically assigned any additional permissions.
- Delete the stage from the pipeline.
- Change the order of stages.
- Save a copy of the stage as a template.
- Manage the security settings for the stage.



Q & A

I need to deploy two Azure resource groups in order to deploy my application. Is that one stage or two?

A stage is a logical entity that represents an independent unit of deployment for your application, so you can deploy both the resource groups using a single stage. For more guidance on stages see the [introductory section](#) above.

At the end of my pipeline, I update the binaries in an app store. I really do not have any stage in this case. How do I model this in a release pipeline?

A stage is a logical entity that can be used to perform any automation. It doesn't need to map to any physical resources. Therefore, you can add a stage to your release pipeline and add tasks to it to upload your binaries to the app store.

Related topics

- [Stage triggers](#)
- [Tasks](#)
- [Stage security](#)

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Deployment groups

10/9/2018 • 4 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

A deployment group is a logical set of deployment target machines that have agents installed on each one. Deployment groups represent the physical environments; for example, "Dev", "Test", "UAT", and "Production". In effect, a deployment group is just another grouping of agents, much like an [agent pool](#).

When authoring an Azure Pipelines or TFS Release pipeline, you can specify the deployment targets for a [job](#) using a deployment group. This makes it easy to define [parallel execution](#) of deployment tasks.

Deployment groups:

- Specify the security context and runtime targets for the agents. As you create a deployment group, you add users and give them appropriate permissions to administer, manage, view, and use the group.
- Let you view live logs for each server as a deployment takes place, and download logs for all servers to track your deployments down to individual machines.
- Enable you to use machine tags to limit deployment to specific sets of target servers.

Create a deployment group

You define groups on the **Deployment Groups** tab of the **Azure Pipelines** section, and install the agent on each server in the group. After you prepare your target servers, they appear in the **Deployment Groups** tab. The list indicates if a server is available, the tags you assigned to each server, and the latest deployment to each server.

The tags you assign allow you to limit deployment to specific servers when the deployment group is used in a [Deployment group job](#). You manage the security for a deployment group by [assigning security roles](#).

Deploy agents to a deployment group

Every target machine in the deployment group requires the build and release agent to be installed. You can do this using the script that is generated in the **Deployment Groups** tab of **Azure Pipelines**. You can choose the type of agent to suit the target operating system and platform; such as Windows and Linux.

If the target machines are Azure VMs, you can quickly and easily prepare them by installing the **Azure Pipelines Agent** Azure VM extension on each of the VMs, or by using the **Azure Resource Group Deployment** task in your release pipeline to create a deployment group dynamically.

You can force the agents on the target machines to be upgraded to the latest version without needing to redeploy them by choosing the **Upgrade targets** command on the shortcut menu for a deployment group.

For more information, see [Provision agents for deployment groups](#).

Monitor releases for deployment groups

When release is executing, you see an entry in the live logs page for each server in the deployment group. After a release has completed, you can download the log files for every server to examine the deployments and resolve issues. To navigate quickly to a release pipeline or a release, use the links in the **Releases** tab.

Share a deployment group

Each deployment group is a member of a **deployment pool**, and you can share the deployment pool and groups across projects provided that:

- The user sharing the deployment pool has [User permission](#) for the pool containing the group.
- The user sharing the deployment pool has permission to create a deployment group in the project where it is being shared.
- The project does not already contain a deployment group that is a member of the same deployment pool.

The tags you assign to each machine in the pool are scoped at project level, so you can specify a different tag for the same machine in each deployment group.

Add a deployment pool and group to another project

To manage a deployment pool, or to add an existing deployment pool and the groups it contains to another project, choose the **Manage** link in the **Agent Pool** section of the **Deployment Group** page. In the **Deployment Pools** page, select the projects for which you want the deployment group to be available, then save the changes.

When you navigate to the **Deployment Groups** page in the target project(s), you will see the deployment group you added and you can assign project-specific machine tags as required.

Create a new deployment pool

You can add a new deployment pool, share it amongst your projects, and then add deployment groups to it. In the **Deployment Pools** page, choose **+ New**. In the **New deployment pool** panel, enter a name for the pool and then select the projects for which you want it to be available.

When you navigate to the **Deployment Groups** page in the target project(s), you will see the deployment group you added and you can assign project-specific machine tags as required.

Related topics

- [Run on machine group job](#)
- [Deploy an agent on Windows](#)
- [Deploy an agent on macOS](#)
- [Deploy an agent on Linux](#)

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

How To: Provision agents for deployment groups

10/31/2018 • 6 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called service endpoints in TFS 2018 and in older versions.

Deployment groups make it easy to define logical groups of target machines for deployment, and install the required agent on each machine. This topic explains how to create a deployment group, and install and provision the agent on each virtual or physical machine in your deployment group.

You can install the agent in any one of these ways:

- [Run the script](#) that is generated automatically when you create a deployment group.
- [Install the Azure Pipelines Agent Azure VM extension](#) on each of the VMs.
- [Use the Azure Resource Group Deployment task](#) in your release pipeline.

For information about agents and pipelines, see:

- [Parallel jobs in Team Foundation Server](#).
- [Parallel jobs in Azure Pipelines](#).
- [Pricing for Azure Pipelines features](#)

Run the installation script on the target servers

1. In the **Deployment groups** tab of **Azure Pipelines**, choose **+New** to create a new group.
2. Enter a name for the group, and optionally a description, then choose **Create**.
3. In the **Register machines using command line** section of the next page, select the target machine operating system.
4. Choose **Use a personal access token in the script for authentication**. [Learn more](#).
5. Choose **Copy the script to clipboard**.
6. Log onto each target machine in turn using the account with the [appropriate permissions](#) and:
 - Open an Administrator PowerShell command prompt, paste in the script you copied, then execute it to register the machine with this group.
 - If you get an error when running the script that a secure channel could not be created, execute this command at the Administrator PowerShell prompt:

```
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12
```

- When prompted to configure tags for the agent, press Y and enter any tags you will use to identify subsets of the machines in the group for partial deployments.

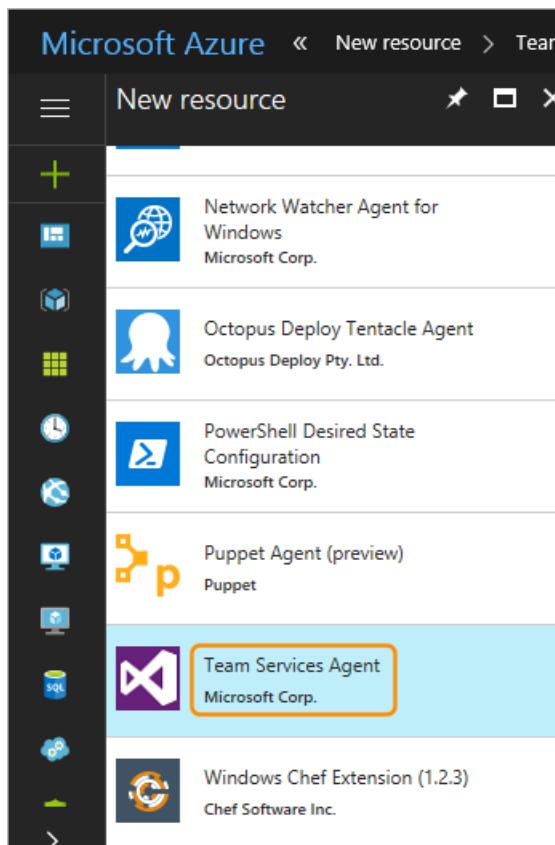
Tags you assign allow you to limit deployment to specific servers when the deployment group is used in a [Run on machine group job](#).

- When prompted for the user account, press *Return* to accept the defaults.
 - Wait for the script to finish with the message

```
Service vstsagent.{organization-name}.{computer-name} started successfully.
```
7. In the **Deployment groups** page of **Azure Pipelines**, open the **Machines** tab and verify that the agents are running. If the tags you configured are not visible, refresh the page.

Install the Azure Pipelines Agent Azure VM extension

1. In the **Deployment groups** tab of **Azure Pipelines**, choose **+New** to create a new group.
2. Enter a name for the group, and optionally a description, then choose **Create**.
3. In the Azure portal, for each VM that will be included in the deployment group open the **Extension** blade, choose **+ Add** to open the **New resource** list, and select **Azure Pipelines Agent**.



4. In the **Install extension** blade, specify the name of the Azure Pipelines subscription to use. For example, if the URL is `https://dev.azure.com/contoso`, just specify **contoso**.
5. Specify the project name and the deployment group name.
6. Optionally, specify a name for the agent. If not specified, it uses the VM name appended with `-DG`.
7. Enter the **Personal Access Token (PAT)** to use for authentication against Azure Pipelines.
8. Optionally, specify a comma-separated list of tags that will be configured on the agent. Tags are not case-sensitive, and each must no more than 256 characters.
9. Choose **OK** to begin installation of the agent on this VM.
10. Add the extension to any other VMs you want to include in this deployment group.

Use the Azure Resource Group Deployment task

You can use the [Azure Resource Group Deployment task](#) to deploy an Azure Resource Manager (ARM) template that installs the Azure Pipelines Agent Azure VM extension as you create a virtual machine, or to update the resource group to apply the extension after the virtual machine has been created. Alternatively, you can use the advanced deployment options of the Azure Resource Group Deployment task to deploy the agent to deployment groups.

Install the "Azure Pipelines Agent" Azure VM extension using an ARM template

An ARM template is a JSON file that declaratively defines a set of Azure resources. The template can be automatically read and the resources provisioned by Azure. In a single template, you can deploy multiple services along with their dependencies.

For a Windows VM, create an ARM template and add a resources element under the

`Microsoft.Compute/virtualMachine` resource as shown here:

```
"resources": [
  {
    "name": "[concat(parameters('vmNamePrefix'),copyIndex(),'/TeamServicesAgent')]",
    "type": "Microsoft.Compute/virtualMachines/extensions",
    "location": "[parameters('location')]",
    "apiVersion": "2015-06-15",
    "dependsOn": [
      "[resourceId('Microsoft.Compute/virtualMachines/',
      concat(parameters('vmNamePrefix'),copyindex()))]"
    ],
    "properties": {
      "publisher": "Microsoft.VisualStudio.Services",
      "type": "TeamServicesAgent",
      "typeHandlerVersion": "1.0",
      "autoUpgradeMinorVersion": true,
      "settings": {
        "VSTSAccountName": "[parameters('VSTSAccountName')]",
        "TeamProject": "[parameters('TeamProject')]",
        "DeploymentGroup": "[parameters('DeploymentGroup')]",
        "AgentName": "[parameters('AgentName')]",
        "Tags": "[parameters('Tags')]"
      },
      "protectedSettings": {
        "PATToken": "[parameters('PATToken')]"
      }
    }
  }
]
```

where:

- **VSTSAccountName** is required. The Azure Pipelines subscription to use. Example: If your URL is `https://dev.azure.com/contoso`, just specify `contoso`
- **TeamProject** is required. The project that has the deployment group defined within it
- **DeploymentGroup** is required. The deployment group against which deployment agent will be registered
- **AgentName** is optional. If not specified, the VM name with `-DG` appended will be used
- **Tags** is optional. A comma-separated list of tags that will be set on the agent. Tags are not case sensitive and each must be no more than 256 characters
- **PATToken** is required. The Personal Access Token that will be used to authenticate against Azure Pipelines to download and configure the agent

Note: If you are deploying to a Linux VM, ensure that the `type` parameter in the code is `TeamServicesAgentLinux`.

For more information about ARM templates, see [Define resources in Azure Resource Manager templates](#).

To use the template:

1. In the **Deployment groups** tab of **Azure Pipelines**, choose **+New** to create a new group.
2. Enter a name for the group, and optionally a description, then choose **Create**.
3. In the **Releases** tab of **Azure Pipelines**, create a release pipeline with a stage that contains the **Azure Resource Group Deployment** task.
4. Provide the parameters required for the task such as the Azure subscription, resource group name, location, and template information, then save the release pipeline.
5. Create a release from the release pipeline to install the agents.

Install agents using the advanced deployment options

1. In the **Deployment groups** tab of **Azure Pipelines**, choose **+New** to create a new group.
2. Enter a name for the group, and optionally a description, then choose **Create**.
3. In the **Releases** tab of **Azure Pipelines**, create a release pipeline with a stage that contains the **Azure Resource Group Deployment** task.
4. Select the task and expand the **Advanced deployment options for virtual machines** section. Configure the parameters in this section as follows:
 - **Enable Prerequisites:** select **Configure with Deployment Group Agent**.
 - **Azure Pipelines/TFS endpoint:** Select an existing Team Foundation Server/TFS service connection that points to your target. Agent registration for deployment groups requires access to your Visual Studio project. If you do not have an existing service connection, choose **Add** and create one now. Configure it to use a [Personal Access Token \(PAT\)](#) with scope restricted to **Deployment Group**.
 - **Project:** Specify the project containing the deployment group.
 - **Deployment Group:** Specify the name of the deployment group against which the agents will be registered.
 - **Copy Azure VM tags to agents:** When set (ticked), any tags already configured on the Azure VM will be copied to the corresponding deployment group agent. By default, all [Azure tags](#) are copied using the format `Key: Value`. For example, `Role: Web`.
5. Provide the other parameters required for the task such as the Azure subscription, resource group name, and location, then save the release pipeline.
6. Create a release from the release pipeline to install the agents.

Related topics

- [Run on machine group job](#)
- [Deploy an agent on Windows](#)
- [Deploy an agent on macOS](#)
- [Deploy an agent on Linux](#)

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#)

page.

Release artifacts and artifact sources

11/29/2018 • 18 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

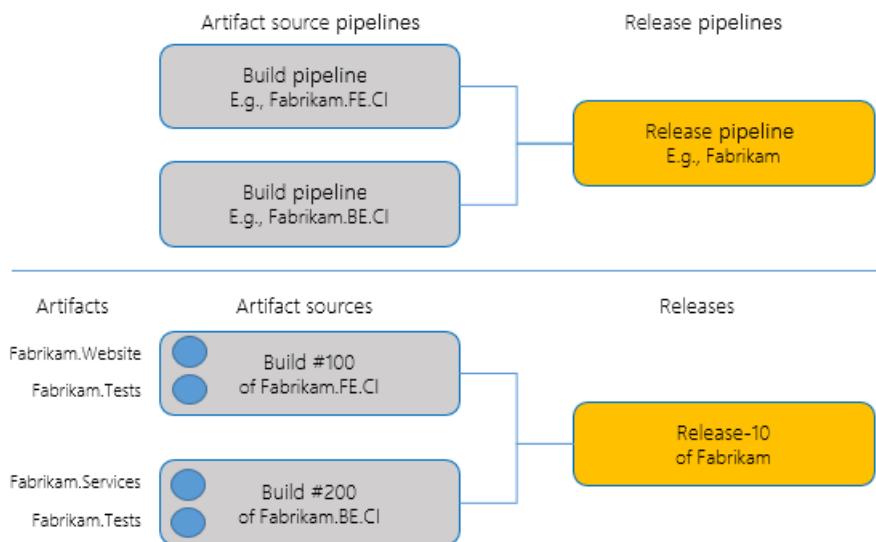
A release is a collection of artifacts in your DevOps CI/CD processes. An **artifact** is a deployable component of your application. Azure Pipelines can deploy artifacts that are produced by a [wide range of artifact sources](#), and stored in different types of artifact repositories.

When **authoring a release pipeline**, you link the appropriate **artifact sources** to your release pipeline. For example, you might link an Azure Pipelines build pipeline or a Jenkins project to your release pipeline.

When **creating a release**, you specify the exact version of these artifact sources; for example, the number of a build coming from Azure Pipelines, or the version of a build coming from a Jenkins project.

After a release is created, you cannot change these versions. A release is fundamentally defined by the versioned artifacts that make up the release. As you deploy the release to various stages, you will be deploying and validating the same artifacts in all stages.

A single release pipeline can be linked to **multiple artifact sources**, of which one is the [primary source](#). In this case, when you create a release, you specify individual versions for each of these sources.



Artifacts are central to a number of features in Azure Pipelines. Some of the features that depend on the linking of artifacts to a release pipeline are:

- **Auto-trigger releases.** You can configure new releases to be automatically created whenever a new version of an artifact is produced. For more details, see [Continuous deployment triggers](#). Note that the ability to automatically create releases is available for only some artifact sources.
- **Trigger conditions.** You can configure a release to be created automatically, or the deployment of a release to a stage to be triggered automatically, when only specific conditions on the artifacts are met. For

example, you can configure releases to be automatically created only when a new build is produced from a certain branch.

- **Artifact versions.** You can configure a release to automatically use a specific version of the build artifacts, to always use the latest version, or to allow you to specify the version when the release is created.
- **Artifact variables.** Every artifact that is part of a release has metadata associated with it, exposed to [tasks](#) through [variables](#). This metadata includes the version number of the artifact, the branch of code from which the artifact was produced (in the case of build or source code artifacts), the pipeline that produced the artifact (in the case of build artifacts), and more. This information is accessible in the deployment tasks. For more details, see [Artifact variables](#).
- **Work items and commits.** The work items or commits that are part of a release are computed from the versions of artifacts. For example, each build in Azure Pipelines is associated with a set of work items and commits. The work items or commits in a release are computed as the union of all work items and commits of all builds between the current release and the previous release. Note that Azure Pipelines is currently able to compute work items and commits for only certain artifact sources.
- **Artifact download.** Whenever a release is deployed to a stage, by default Azure Pipelines automatically downloads all the artifacts in that release to the [agent](#) where the deployment job runs. The procedure to download artifacts depends on the type of artifact. For example, Azure Pipelines artifacts are downloaded using an algorithm that downloads multiple files in parallel. Git artifacts are downloaded using Git library functionality. For more details, see [Artifact download](#).

Artifact sources

There are several types of tools you might use in your application lifecycle process to produce or store artifacts. For example, you might use continuous integration systems such as Azure Pipelines, Jenkins, or TeamCity to produce artifacts. You might also use version control systems such as Git or TFVC to store your artifacts. Or you can use repositories such as Package Management in Visual Studio Team Services or a NuGet repository to store your artifacts. You can configure Azure Pipelines to deploy artifacts from all these sources.

By default, a release created from the release pipeline will use the latest version of the artifacts. At the time of linking an artifact source to a release pipeline, you can change this behavior by selecting one of the options to use the latest build from a specific branch by specifying the tags, a specific version, or allow the user to specify the version when the release is created from the pipeline.

All definitions > **Fabrika**

Pipeline Tasks Variables Ref

Artifacts **+ Add**

Build Azure Repos... GitHub TFVC

3 more artifact types ▾

Project * ⓘ
Fabrikam

Source (Build definition) * ⓘ
Fabrikam.CI

Default version * ⓘ
Latest

Latest
Latest from build definition default branch with tags
Latest from specific branch with tags
Specific version
Specify at the time of release creation

Add

If you link more than one set of artifacts, you can specify which is the primary (default).

Artifact
Build - TestsWebAppBuild

Project
Fabrikam

Source (Build definition)
TestsWebAppBuild

Delete **...**

Mark primary

The following sections describe how to work with the different types of artifact sources.

- [Azure Pipelines](#)
- [TFVC, Git, and GitHub](#)
- [Jenkins](#)
- [Azure Container Registry, Docker, and Kubernetes](#)
- [Azure Artifacts \(NuGet, Maven, and npm\)](#)
- [External or on-premises TFS](#)
- [TeamCity](#)
- [Other sources](#)

Azure Pipelines

You can link a release pipeline to any of the build pipelines in Azure Pipelines or TFS project collection.

NOTE

You must include a **Publish Artifacts** task in your build pipeline. For XAML build pipelines, an artifact with the name **drop** is published implicitly.

Some of the differences in capabilities between different versions of TFS and Azure Pipelines are:

- **TFS 2015:** You can link build pipelines only from the same project of your collection. You can link multiple definitions, but you cannot specify default versions. You can set up a continuous deployment trigger on only one of the definitions. When multiple build pipelines are linked, the latest builds of all the other definitions are used, along with the build that triggered the release creation.
- **TFS 2017 and newer and Azure Pipelines:** You can link build pipelines from any of the projects in Azure Pipelines or TFS. You can link multiple build pipelines and specify default values for each of them. You can set up continuous deployment triggers on multiple build sources. When any of the builds completes, it will trigger the creation of a release.

The following features are available when using Azure Pipelines sources:

FEATURE	BEHAVIOR WITH AZURE PIPELINES SOURCES
Auto-trigger releases	New releases can be created automatically when new builds (including XAML builds) are produced. See Continuous Deployment for details. You do not need to configure anything within the build pipeline. See the notes above for differences between version of TFS.
Artifact variables	A number of artifact variables are supported for builds from Azure Pipelines.
Work items and commits	Azure Pipelines integrates with work items in TFS and Azure Pipelines. These work items are also shown in the details of releases. Azure Pipelines integrates with a number of version control systems such as TFVC and Git, GitHub, Subversion, and external Git repositories. Azure Pipelines shows the commits only when the build is produced from source code in TFVC or Git.
Artifact download	By default, build artifacts are downloaded to the agent. You can configure an option in the stage to skip the download of artifacts.
Deployment section in build	The build summary includes a Deployment section, which lists all the stages to which the build was deployed.

TFVC, Git, and GitHub

There are scenarios in which you may want to consume artifacts stored in a version control system directly, without passing them through a build pipeline. For example:

- You are developing a PHP or a JavaScript application that does not require an explicit build pipeline.
- You manage configurations for various stages in different version control repositories, and you want to consume these configuration files directly from version control as part of the deployment pipeline.
- You manage your infrastructure and configuration as code (such as Azure Resource Manager templates) and you want to manage these files in a version control repository.

Because you can configure multiple artifact sources in a single release pipeline, you can link both a build pipeline that produces the binaries of the application as well as a version control repository that stores the configuration files into the same pipeline, and use the two sets of artifacts together while deploying.

Azure Pipelines integrates with Team Foundation Version Control (TFVC) repositories, Git repositories, and GitHub repositories.

You can link a release pipeline to any of the Git or TFVC repositories in any of the projects in your collection (you will need read access to these repositories). No additional setup is required when deploying version control artifacts within the same collection.

When you link a **Git** or **GitHub** repository and select a branch, you can edit the default properties of the artifact types after the artifact has been saved. This is particularly useful in scenarios where the branch for the stable version of the artifact changes, and continuous delivery releases should use this branch to obtain newer versions of the artifact. You can also specify details of the checkout, such as whether check out submodules and LFS-tracked files, and the shallow fetch depth.

When you link a **TFVC branch**, you can specify the changeset to be deployed when creating a release.

The following features are available when using TFVC, Git, and GitHub sources:

FEATURE	BEHAVIOR WITH TFVC, GIT, AND GITHUB SOURCES
Auto-trigger releases	You can configure a continuous deployment trigger for pushes into the repository in a release pipeline. This can automatically trigger a release when a new commit is made to a repository. See Triggers .
Artifact variables	A number of artifact variables are supported for version control sources.
Work items and commits	Azure Pipelines cannot show work items or commits associated with releases when using version control artifacts.
Artifact download	By default, version control artifacts are downloaded to the agent. You can configure an option in the stage to skip the download of artifacts.

Jenkins

To consume Jenkins artifacts, you must create a service connection with credentials to connect to your Jenkins server. For more details, see [service connections](#) and [Jenkins service connection](#). You can then link a Jenkins project to a release pipeline. The Jenkins project must be configured with a post build action to publish the artifacts.

The following features are available when using Jenkins sources:

FEATURE	BEHAVIOR WITH JENKINS SOURCES
Auto-trigger releases	You can configure a continuous deployment trigger for pushes into the repository in a release pipeline. This can automatically trigger a release when a new commit is made to a repository. See Triggers .
Artifact variables	A number of artifact variables are supported for builds from Jenkins.
Work items and commits	Azure Pipelines cannot show work items or commits for Jenkins builds.
Artifact download	By default, Jenkins builds are downloaded to the agent. You can configure an option in the stage to skip the download of artifacts.

Artifacts generated by Jenkins builds are typically propagated to storage repositories for archiving and sharing. Azure blob storage is one of the supported repositories, allowing you to consume Jenkins projects that publish to Azure storage as artifact sources in a release pipeline. Deployments download the artifacts automatically from Azure to the agents. In this configuration, connectivity between the agent and the Jenkins server is not required. Microsoft-hosted agents can be used without exposing the server to internet.

NOTE

Azure Pipelines may not be able to contact your Jenkins server if, for example, it is within your enterprise network. In this case you can integrate Azure Pipelines with Jenkins by setting up an on-premises agent that can access the Jenkins server. You will not be able to see the name of your Jenkins projects when linking to a build, but you can type this into the link dialog field.

For more information about Jenkins integration capabilities, see [Azure Pipelines Integration with Jenkins Jobs, Pipelines, and Artifacts](#).

Azure Container Registry, Docker, Kubernetes

When deploying containerized apps, the container image is first pushed to a container registry. After the push is complete, the container image can be deployed to the Web App for Containers service or a Docker/Kubernetes cluster. You must create a service connection with credentials to connect to your service to deploy images located there, or to Azure. For more details, see [service connections](#).

The following features are available when using Azure Container Registry, Docker, Kubernetes sources:

FEATURE	BEHAVIOR WITH DOCKER SOURCES
Auto-trigger releases	You can configure a continuous deployment trigger for images. This can automatically trigger a release when a new commit is made to a repository. See Triggers .
Artifact variables	A number of artifact variables are supported for builds.
Work items and commits	Azure Pipelines cannot show work items or commits.
Artifact download	By default, builds are downloaded to the agent. You can configure an option in the stage to skip the download of artifacts.

Azure Artifacts (NuGet packages only)

To use NuGet packages from Azure Artifacts in your deployment, you must first [assign licenses for the Azure Artifacts](#). For more information, see the [Azure Artifacts](#) overview.

Scenarios where you may want to consume these artifacts are:

1. You have your application build (such as TFS, Azure Pipelines, TeamCity, Jenkins) published as a package to Azure Artifacts and you want to consume the artifact in a release.
2. As part of your application deployment, you need additional packages stored in Azure Artifacts.

When you link such an artifact to your release pipeline, you must select the Feed, Package, and the Default version for the package. You can choose to pick up the latest version of the package, use a specific version, or select the version at the time of release creation. During deployment, the package is downloaded to the agent folder and the contents are extracted as part of the job execution.

The following features are available when using Azure Artifacts sources:

FEATURE	BEHAVIOR WITH AZURE ARTIFACTS SOURCES
Auto-trigger releases	You can configure a continuous deployment trigger for packages. This can automatically trigger a release when a package is updated. See Triggers .
Artifact variables	A number of artifact variables are supported for packages.
Work items and commits	Azure Pipelines cannot show work items or commits.
Artifact download	By default, packages are downloaded to the agent. You can configure an option in the stage to skip the download of artifacts.

NOTE

Only NuGet packages are currently supported in the Azure Artifacts release artifact type. Support for the other package types supported in Azure Artifacts is coming soon.

External or on-premises TFS

You can use Azure Pipelines to deploy artifacts published by an on-premises TFS server. You don't need to make the TFS server visible on the Internet; you just set up an on-premises automation agent. Builds from an on-premises TFS server are downloaded directly into the on-premises agent, and then deployed to the specified target servers. They will not leave your enterprise network. This allows you to leverage all of your investments in your on-premises TFS server, and take advantage of the release capabilities in Azure Pipelines.

Using this mechanism, you can also deploy artifacts published in one Azure Pipelines subscription in another Azure Pipelines, or deploy artifacts published in one Team Foundation Server from another Team Foundation Server.

To enable these scenarios, you must install the [TFS artifacts for Azure Pipelines](#) extension from Visual Studio Marketplace. Then create a service connection with credentials to connect to your TFS server (see [service connections](#) for details).

You can then link a TFS build pipeline to your release pipeline. Choose **External TFS Build** in the **Type** list.

The following features are available when using external TFS sources:

FEATURE	BEHAVIOR WITH EXTERNAL TFS SOURCES
Auto-trigger releases	You cannot configure a continuous deployment trigger for external TFS sources in a release pipeline. To automatically create a new release when a build is complete, you would need to add a script to your build pipeline in the external TFS server to invoke Azure Pipelines REST APIs and to create a new release.
Artifact variables	A number of artifact variables are supported for external TFS sources.
Work items and commits	Azure Pipelines cannot show work items or commits for external TFS sources.

FEATURE	BEHAVIOR WITH EXTERNAL TFS SOURCES
Artifact download	By default, External TFS artifacts are downloaded to the agent. You can configure an option in the stage to skip the download of artifacts.

NOTE

Azure Pipelines may not be able to contact an on-premises TFS server if, for example, it is within your enterprise network. In this case you can integrate Azure Pipelines with TFS by setting up an on-premises agent that can access the TFS server. You will not be able to see the name of your TFS projects or build pipelines when linking to a build, but you can type these into the link dialog fields. In addition, when you create a release, Azure Pipelines may not be able to query the TFS server for the build numbers. Instead, type the **Build ID** (not the build number) of the desired build into the appropriate field, or select the **Latest** build.

TeamCity

To integrate with TeamCity, you must first install the [TeamCity artifacts for Azure Pipelines](#) extension from Marketplace.

To consume TeamCity artifacts, start by creating a service connection with credentials to connect to your TeamCity server (see [service connections](#) for details).

You can then link a TeamCity build configuration to a release pipeline. The TeamCity build configuration must be configured with an action to publish the artifacts.

The following features are available when using TeamCity sources:

FEATURE	BEHAVIOR WITH TEAMCITY SOURCES
Auto-trigger releases	You cannot configure a continuous deployment trigger for TeamCity sources in a release pipeline. To create a new release automatically when a build is complete, add a script to your TeamCity project that invokes the Azure Pipelines REST APIs to create a new release.
Artifact variables	A number of artifact variables are supported for builds from TeamCity.
Work items and commits	Azure Pipelines cannot show work items or commits for TeamCity builds.
Artifact download	By default, TeamCity builds are downloaded to the agent. You can configure an option in the stage to skip the download of artifacts.

NOTE

Azure Pipelines may not be able to contact your TeamCity server if, for example, it is within your enterprise network. In this case you can integrate Azure Pipelines with TeamCity by setting up an on-premises agent that can access the TeamCity server. You will not be able to see the name of your TeamCity projects when linking to a build, but you can type this into the link dialog field.

Other sources

Your artifacts may be created and exposed by other types of sources such as a NuGet repository. While we continue to expand the types of artifact sources supported in Azure Pipelines, you can start using it without waiting for support for a specific source type. Simply skip the linking of artifact sources in a release pipeline, and add custom tasks to your stages that download the artifacts directly from your source.

Artifact download

When you deploy a release to a stage, the versioned artifacts from each of the sources are, by default, downloaded to the automation agent so that tasks running within that stage can deploy these artifacts. The artifacts downloaded to the agent are not deleted when a release is completed. However, when you initiate the next release, the downloaded artifacts are deleted and replaced with the new set of artifacts.

A new unique folder in the agent is created for every release pipeline when you initiate a release, and the artifacts are downloaded into that folder. The `$(System.DefaultWorkingDirectory)` variable maps to this folder.

Note that, at present, Azure Pipelines does not perform any optimization to avoid downloading the unchanged artifacts if the same release is deployed again. In addition, because the previously downloaded contents are always deleted when you initiate a new release, Azure Pipelines cannot perform incremental downloads to the agent.

You can, however, instruct Azure Pipelines to [skip the automatic download](#) of artifacts to the agent for a specific job and stage of the deployment if you wish. Typically, you will do this when the tasks in that job do not require any artifacts, or if you implement custom code in a task to download the artifacts you require.

In Azure Pipelines, you can, however, [select which artifacts you want to download](#) to the agent for a specific job and stage of the deployment. Typically, you will do this to improve the efficiency of the deployment pipeline when the tasks in that job do not require all or any of the artifacts, or if you implement custom code in a task to download the artifacts you require.

The screenshot shows the 'Artifact download' section of the Azure Pipelines interface. At the top, there's a header with a dropdown arrow, the project name 'HotelFinder-CI', the build version 'Latest', and a status indicating 'Selected 1/3 artifact(s)'. Below this, a message says 'Following are the artifacts published in the latest version of the build.' A list of artifacts is shown with checkboxes: 'TestFile' (unchecked), 'SeleniumTests' (unchecked), and 'Web' (checked). Below this, there are three artifact source sections: 'thelisland' (selected), 'Fabrikam' (not selected), and another 'thelisland' entry (not selected). Each section has a dropdown arrow and a 'Selected all artifacts' button.

Artifact source alias

To ensure the uniqueness of every artifact download, each artifact source linked to a release pipeline is automatically provided with a specific download location known as the *source alias*. This location can be accessed through the variable:

```
$(System.DefaultWorkingDirectory)\[source alias]
```

This uniqueness also ensures that, if you later rename a linked artifact source in its original location (for example,

rename a build pipeline in Azure Pipelines or a project in Jenkins), you don't need to edit the task properties because the download location defined in the agent does not change.

The source alias is, by default, the name of the source selected when you linked the artifact source, prefixed with an underscore; depending on the type of the artifact source this will be the name of the build pipeline, job, project, or repository. You can edit the source alias from the artifacts tab of a release pipeline; for example, when you change the name of the build pipeline and you want to use a source alias that reflects the name of the build pipeline.

The source alias can contain only alphanumeric characters and underscores, and must start with a letter or an underscore

Primary source

When you link multiple artifact sources to a release pipeline, one of them is designated as the primary artifact source. The primary artifact source is used to set a number of pre-defined [variables](#). It can also be used in [naming releases](#).

Artifact variables

Azure Pipelines exposes a set of pre-defined variables that you can access and use in tasks and scripts; for example, when executing PowerShell scripts in deployment jobs. When there are multiple artifact sources linked to a release pipeline, you can access information about each of these. For a list of all pre-defined artifact variables, see [variables](#).

Contributed links and additional information

- [Code repo sources in Azure Pipelines](#)
- [TeamCity extension for Azure Pipelines](#)
- [Jenkins artifacts in Azure Pipelines](#)
- [External TFS artifacts for Azure Pipelines](#)

Related topics

- [Release pipelines](#)
- [Stages](#)

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Release, branch, and stage triggers

11/19/2018 • 5 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

You can configure when releases should be created, and when those releases should be deployed to stages, in your DevOps CI/CD processes. The former is configured through [release triggers](#), and the latter through [stage triggers](#) - both in a release pipeline.

Continuous deployment triggers

If you specify [certain types](#) of artifacts in a release pipeline, you can enable continuous deployment. This instructs Azure Pipelines to create new releases automatically when it detects new artifacts are available. At present this option is available only for Team Foundation Build artifacts and Git-based sources such as Team Foundation Git, GitHub, and other Git repositories.

The screenshot shows the 'Continuous deployment trigger' configuration for a release pipeline named 'DotNetSample-ASP.NET Core-CI'. On the left, under 'Artifacts', there is a list with one item: 'DotNetSample-ASP.NET Core-CI', which has a red box around its 'Edit' icon. Below this is a note: 'Schedule not set'. On the right, the 'Continuous deployment trigger' section is shown with the following details:

- Enabled:** A toggle switch is turned on, labeled 'Enabled'. Below it, the text reads 'Creates a release every time a new build is available.'
- Build branch filters:** A table with columns 'Type', 'Build branch', and 'Build tags'.
 - Type:** 'Include' dropdown set to 'The build pipeline's default bra...'.
 - Build branch:** 'The build pipeline's default branch'.
 - Build tags:** An empty input field.
 - Add:** A button with '+ Add'.
- Branch filter:** A note: 'The build pipeline's default branch'

If you have linked multiple Team Foundation Build artifacts to a release pipeline, you can configure continuous deployment for each of them. In other words, you can choose to have a release created automatically when a new build of any of those artifacts is produced.

You add build branch filters if you want to create the release only when the build is produced by compiling code from certain branches (only applicable when the code is in a TFVC, Git, or GitHub repository) or when the build has certain tags. These can be both include and exclude filters. For example, use **features/*** to include all builds under the **features** branch. You can also include [custom variables](#) in a filter value.

Alternatively, you can specify a filter to use the default branch specified in the build pipeline. This is useful when, for example, the default build branch changes in every development sprint. It means you don't need to update the trigger filter across all release pipelines for every change - instead you just change the default branch in the build pipeline.

Note that, even though a release is automatically created, it might not be deployed automatically to any stages. The [stage triggers](#) govern when and if a release should be deployed to a stage.

Scheduled release triggers

If you want to create and start a release at specific times, define one or more scheduled release triggers. Choose the schedule icon in the **Artifacts** section of your pipeline and enable scheduled release triggers. You can configure multiple schedules.

The screenshot shows the 'Scheduled release trigger' configuration for the 'FabrikamFiber' pipeline. On the left, under 'Artifacts', there is a 'FabrikamCode' artifact and a 'Schedule set' item highlighted with a red box. The main panel is titled 'Scheduled release trigger' and contains the following fields:

- Enabled:** A toggle switch is turned on and highlighted with a red box.
- Create a new release at the specified times:** A dropdown menu shows 'Mon through Fri at 3:00'. Days from Monday to Friday are checked, while Saturday is unchecked. The time is set to '03h 00m' in '(UTC) Coordinated Universal Time'.
- + Add a new time:** A button highlighted with a red box.

See also [stage scheduled triggers](#).

Pull request triggers

You can configure a pull request trigger that will create a new release when a pull request uploads a new version of the artifact. Enable the trigger and add the branches targeted by pull requests that you want to activate this trigger.

The screenshot shows the 'Pull request trigger' configuration for the 'DotNetSample-ASP.NET Core-CI' build. On the left, under 'Artifacts', there is a 'DotNetSample-ASP.NET Core-CI' artifact highlighted with a red box. The main panel is titled 'Pull request trigger' and contains the following fields:

- Enabled:** A toggle switch is turned on and highlighted with a red box.
- Creates a release every time a new version of the selected artifact is available as part of a pull request workflow:** A descriptive text field.
- Target Branch Filters:** A dropdown menu shows 'master' selected.
- Stages:** A section indicating '0 of 2 stages are enabled for pull request deployments. You can enable a stage for pull request based deployments in the pre-deployment conditions of that stage.'

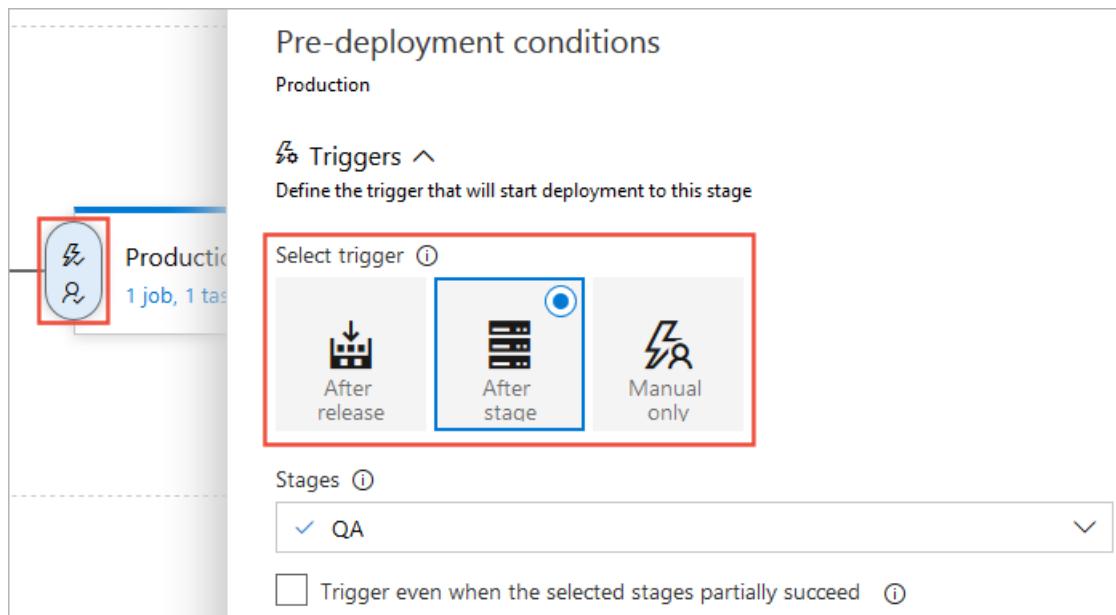
However, to use a pull request trigger, you must also enable it for specific stages of the pipeline. Do this in the stage [triggers panel](#) for the required stage(s). You may also want to set up a [branch policy](#) for the branch.

Note that, even though a release is automatically created, it might not be deployed automatically to any stages. The [stage triggers](#) govern when and if a release should be deployed to a stage.

Stage triggers

You can choose to have the deployment to each stage triggered automatically when a release is created by a continuous deployment trigger, based on:

- **The result of deploying to a previous stage in the pipeline.** Use this setting if you want the release to be first deployed and validated in another stage(s) before it is deployed to this stage. Triggers are configured for each stage, but the combination of these allows you to orchestrate the overall deployment - such as the sequence in which automated deployments occur across all the stages in a release pipeline. For example, you can set up a linear pipeline where a release is deployed first to the **Test** and **QA** stages. Then, if these two deployments succeed, it will be deployed to a **Staging** stage. In addition, you can configure the trigger to fire for partially succeeded (but not failed) deployments.



- **Filters based on the artifacts.** You can add one or more filters for each artifact linked to the release pipeline, and specify if you want to include or exclude particular branches of the code. Deployment will be triggered to this stage only if all the artifact conditions are successfully met. Unlike [build branch filters](#), variables *cannot* be used in artifact filter conditions.

The screenshot shows the 'Triggers' configuration for the 'QA' stage. On the left, the pipeline stages are listed: 'QA' (2 jobs, 2 tasks) and 'Production' (1 job, 1 task). The 'QA' stage icon is highlighted with a red box. In the center, the 'Select trigger' section shows three options: 'After release' (selected), 'After stage', and 'Manual only'. Below this are 'Artifact filters' for the artifact 'DotNetSample-ASP.NET Core-CI', which includes 'Type: Include', 'Build branch: master', and a '+ Add' button. To the right is an 'Enabled' toggle switch, also highlighted with a red box.

- **A predefined schedule.** When you select this option, you can select the days of the week and the time of day that Azure Pipelines will automatically start a new deployment. Unlike scheduled release triggers, you cannot configure multiple schedules for stage triggers. Note that, with scheduled triggers, a new deployment is created even if a newer version of artifact is not available.

The screenshot shows the 'Triggers' configuration for the 'Production' stage. The 'QA' stage icon is highlighted with a red box. In the 'Select trigger' section, 'After release' is selected. Below it, the 'Stages' dropdown shows 'QA' selected. A checkbox for 'Trigger even when the selected stages partially succeed' is unchecked. Under 'Artifact filters', the toggle switch is set to 'Disabled'. At the bottom, the 'Schedule' section is expanded, showing a repeating schedule from Monday through Friday at 3:00 UTC. The days of the week are checked, and the time is set to 03h 00m UTC. The 'Enabled' toggle switch for the schedule is highlighted with a red box.

- **A pull request that updates the artifacts.** If you have enabled [pull request triggers](#) for your pipeline, you must also enable pull request deployment for the specific stages where you want the release to be deployed. You may also want to set up a [branch policy](#) for the branch.

The screenshot shows the Azure DevOps Pipeline configuration for the 'SampleApp - 1' pipeline. On the left, there's a sidebar with 'Stages' and a '+ Add' button. The 'QA' stage is highlighted with a red box. To the right, under 'Pre-deployment conditions', there's a section for 'Triggers' with three options: 'After release' (selected), 'After stage', and 'Manual only'. Below that are sections for 'Artifact filters' (disabled), 'Schedule' (disabled), and 'Pull request deployment' (enabled, with a red box around the toggle switch).

- **Manually by a user.** Releases are not automatically deployed to the stage. To deploy a release to this stage, you must manually start a release and deployment from the release pipeline or from a build summary.

You can combine the automated settings to have deployments created automatically either when a new build is available or according to a schedule.

TFS 2015: The following features are not available in TFS 2015 - continuous deployment triggers for multiple artifact sources, multiple scheduled triggers, combining scheduled and continuous deployment triggers in the same pipeline, continuous deployment based on the branch or tag of a build.

Parallel forked and joined deployments

The **Triggering stage** list lets you select more than one stage. This allows you to configure parallel (*forked* and *joined*) deployment pipelines where the deployment to a stage occurs only when deployment to **all** the selected stages succeeds.

For example, the following schematic shows a pipeline where deployment occurs in parallel to the **QA** and **Pre-prod** stages after deployment to the **Dev** stage succeeds. However, deployment to the **Production** stage occurs only after successful deployment to both the **QA** and **Pre-prod** stages.



In combination with the ability to define [pre- and post-deployment approvals](#), this capability enables the configuration of complex and fully managed deployment pipelines to suit almost any release scenario.

Note that you can always deploy a release directly to any of the stages in your release pipeline by selecting the **Deploy** action when you create a new release. In this case, the stage triggers you configure, such as a trigger on successful deployment to another stage, do not apply. The deployment occurs irrespective of these settings. This gives you the ability to override the release pipeline. Performing such direct deployments requires the **Manage deployments** permission, which should only be given to selected and approved users.

TFS 2015: Parallel fork and joined deployments are not available in TFS 2015

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Default and custom release variables and debugging

11/19/2018 • 12 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

As you compose the tasks for deploying your application into each stage in your DevOps CI/CD processes, variables will help you to:

- Define a more generic deployment pipeline once, and then customize it easily for each stage. For example, a variable can be used to represent the connection string for web deployment, and the value of this variable can be changed from one stage to another. These are **custom variables**.
- Use information about the context of the particular release, [stage](#), [artifacts](#), or [agent](#) in which the deployment pipeline is being run. For example, your script may need access to the location of the build to download it, or to the working directory on the agent to create temporary files. These are **default variables**.

You can view the [current values of all variables](#) for a release, and use a default variable to [run a release in debug mode](#).

Custom variables

Custom variables can be defined at various scopes.

- Share values across all of the definitions in a project by using [variable groups](#). Choose a variable group when you need to use the same values across all the definitions, stages, and tasks in a project, and you want to be able to change the values in a single place. You define and manage variable groups in the **Library** tab.
- Share values across all of the stages by using **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. You define and manage these variables in the **Variables** tab in a release pipeline.
- Share values across all of the tasks within one specific stage by using **stage variables**. Use an stage-level variable for values that vary from stage to stage (and are the same for all the tasks in an stage). You define and manage these variables in the **Variables** tab of an stage in a release pipeline.

Using custom variables at project, release pipeline, and stage scope helps you to:

- Avoid duplication of values, making it easier to update all occurrences as one operation.
- Store sensitive values in a way that they cannot be seen or changed by users of the release pipelines. Designate a configuration property to be a secure (secret) variable by selecting the  (padlock) icon next to the variable.

The values of hidden (secret) variables are stored securely on the server and cannot be viewed by users after they are saved. During a deployment, the Azure Pipelines release service decrypts these values when referenced by the tasks and passes them to the agent over a secure HTTPS channel.

Using custom variables

To use custom variables in your build and release tasks, simply enclose the variable name in parentheses and precede it with a \$ character. For example, if you have a variable named **adminUserName**, you can insert the current value of that variable into a parameter of a task as `$(adminUserName)`.

Note: At present, variables in different groups that are linked to a pipeline in the same scope (such as a release or stage scope) will collide and the result may be unpredictable. Ensure that you use different names for variables across all your variable groups.

You can use custom variables to prompt for values during the execution of a release. For more details, see [Approvals](#).

Define and modify your variables in a script

To define or modify a variable from a script, use the `task.setvariable` logging command. Note that the updated variable value is scoped to the job being executed, and does not flow across jobs or stages. Variable names are transformed to uppercase, and the characters "." and " " are replaced by "_".

For example, `Agent.WorkFolder` becomes `AGENT_WORKFOLDER`. On Windows, you access this as `%AGENT_WORKFOLDER%` or `$env:AGENT_WORKFOLDER`. On Linux and macOS, you use `$AGENT_WORKFOLDER`.

TIP

You can run a script on a:

- Windows agent using either a [Batch script task](#) or [PowerShell script task](#).
- macOS or Linux agent using a [Shell script task](#).

- [Batch](#)
- [PowerShell](#)
- [Shell](#)

Batch script



Set the `sauce` and `secret.Sauce` variables

```
@echo ##vso[task.setvariable variable=sauce]crushed tomatoes
@echo ##vso[task.setvariable variable=secret.Sauce;issecret=true]crushed tomatoes with garlic
```



Read the variables

Arguments

```
"$(sauce)" "$(secret.Sauce)"
```

Script

```

@echo off
set sauceArgument=%~1
set secretSauceArgument=%~2
@echo No problem reading %sauceArgument% or %SAUCE%
@echo But I cannot read %SECRET_SAUCE%
@echo But I can read %secretSauceArgument% (but the log is redacted so I do not spoil
the secret)

```

Console output from reading the variables:

```

No problem reading crushed tomatoes or crushed tomatoes
But I cannot read
But I can read ***** (but the log is redacted so I do not spoil the secret)

```

Default variables

Information about the execution context is made available to running tasks through default variables. Your tasks and scripts can use these variables to find information about the system, release, stage, or agent they are running in. With the exception of **System.Debug**, these variables are read-only and their values are automatically set by the system. Some of the most significant variables are described in the following tables.

System variables

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
System.TeamFoundationServerUri	The URL of the service connection in TFS or Azure Pipelines. Use this from your scripts or tasks to call Azure Pipelines REST APIs.	https://fabrikam.vssrm.visualstudio.com/	
System.TeamFoundationCollectionUri	The URL of the Team Foundation collection or Azure Pipelines. Use this from your scripts or tasks to call REST APIs on other services such as Build and Version control.	https://dev.azure.com/fabrikam/	
System.CollectionId	The ID of the collection to which this build or release belongs.	6c6f3423-1c84-4625-995a-f7f143a1e43d	TFS 2015
System.TeamProject	The name of the project to which this build or release belongs.	Fabrikam	
System.TeamProjectId	The ID of the project to which this build or release belongs.	79f5c12e-3337-4151-be41-a268d2c73344	TFS 2015

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
System.ArtifactsDirectory	The directory to which artifacts are downloaded during deployment of a release. The directory is cleared before every deployment if it requires artifacts to be downloaded to the agent. Same as Agent.ReleaseDirectory and System.DefaultWorkingDirectory.	C:\agent_work\r1\a	
System.DefaultWorkingDirectory	The directory to which artifacts are downloaded during deployment of a release. The directory is cleared before every deployment if it requires artifacts to be downloaded to the agent. Same as Agent.ReleaseDirectory and System.ArtifactsDirectory.	C:\agent_work\r1\a	
System.WorkFolder	The working directory for this agent, where subfolders are created for every build or release. Same as Agent.RootDirectory and Agent.WorkFolder.	C:\agent_work	
System.Debug	This is the only system variable that can be <i>set</i> by the users. Set this to true to run the release in debug mode to assist in fault-finding.	true	

Release variables

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
Release.DefinitionName	The name of the release pipeline to which the current release belongs.	fabrikam-cd	
Release.DefinitionId	The ID of the release pipeline to which the current release belongs.	1	TFS 2015
Release.ReleaseName	The name of the current release.	Release-47	
Release.ReleaseId	The identifier of the current release record.	118	
Release.ReleaseUri	The URI of current release.	vstfs:///ReleaseManagement/Release/118	

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
Release.ReleaseDescription	The text description provided at the time of the release.	Critical security patch	
Release.RequestedFor	The display name of identity that triggered the release.	Mateo Escobedo	
Release.RequestedForEmail	The email address of identity that triggered the release.	mateo@fabrikam.com	
Release.RequestedForId	The ID of identity that triggered the release.	2f435d07-769f-4e46-849d-10d1ab9ba6ab	
Release.EnvironmentName	The name of stage to which deployment is currently in progress.	Dev	
Release.EnvironmentId	The ID of the stage instance in a release to which the deployment is currently in progress.	276	
Release.EnvironmentUri	The URI of the stage instance in a release to which deployment is currently in progress.	vstfs://ReleaseManagement/Environment/276	
Release.DefinitionEnvironmentId	The ID of the stage in the corresponding release pipeline.	1	TFS 2015
Release.AttemptNumber	The number of times this release is deployed in this stage.	1	TFS 2015
Release.Deployment.RequestedFor	The display name of the identity that triggered (started) the deployment currently in progress.	Mateo Escobedo	TFS 2015
Release.Deployment.RequestedForId	The ID of the identity that triggered (started) the deployment currently in progress.	2f435d07-769f-4e46-849d-10d1ab9ba6ab	TFS 2015
Release.DeploymentID	The ID of the deployment. Unique per job.	254	
Release.DeployPhaseID	The ID of the phase where deployment is running.	127	
Release.Environments.{stage-name}.status	The deployment status of the stage.	InProgress	

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
Release.ReleaseWebURL	The URL for this release.	https://dev.azure.com/fabrika m/f3325c6c/_release? releaseId=392&_a=release- summary	
Release.SkipArtifactDownload	Boolean value that specifies whether or not to skip downloading of artifacts to the agent.	FALSE	
Release.TriggeringArtifact.Alias	The alias of the artifact which triggered the release. This is empty when the release was scheduled or triggered manually.	fabrikam_app	

Release stage variables

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
Release.Environments.{stage name}.Status	The status of deployment of this release within a specified stage.	NotStarted	TFS 2015

Agent variables

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
Agent.Name	The name of the agent as registered with the agent pool . This is likely to be different from the computer name.	fabrikam-agent	
Agent.MachineName	The name of the computer on which the agent is configured.	fabrikam-agent	
Agent.Version	The version of the agent software.	2.109.1	
Agent.JobName	The name of the job that is running, such as Release or Build.	Release	
Agent.HomeDirectory	The folder where the agent is installed. This folder contains the code and resources for the agent.	C:\agent	

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
Agent.ReleaseDirectory	The directory to which artifacts are downloaded during deployment of a release. The directory is cleared before every deployment if it requires artifacts to be downloaded to the agent. Same as System.ArtifactsDirectory and System.DefaultWorkingDirectory.	C:\agent_work\r1\a	
Agent.RootDirectory	The working directory for this agent, where subfolders are created for every build or release. Same as Agent.WorkFolder and System.WorkFolder.	C:\agent_work	
Agent.WorkFolder	The working directory for this agent, where subfolders are created for every build or release. Same as Agent.RootDirectory and System.WorkFolder.	C:\agent_work	
Agent.DeploymentGroupId	The ID of the deployment group the agent is registered with. This is available only in deployment group jobs.	1	TFS 2018 U1

General artifact variables

For each artifact that is referenced in a release, you can use the following artifact variables. Not all variables are meaningful for each artifact type. The table below lists the default artifact variables and provides examples of the values that they have depending on the artifact type. If an example is empty, it implies that the variable is not populated for that artifact type.

Replace **{alias}** with the value you specified for the [artifact alias](#), or with the default value generated for the release pipeline.

VARIABLE NAME	DESCRIPTION	AZURE PIPELINES EXAMPLE	JENKINS/TEAMCITY EXAMPLE	TFVC/GIT EXAMPLE	GITHUB EXAMPLE
Release.Artifacts.{alias}.DefinitionId	The identifier of the build pipeline or repository.	1			fabrikam/asp
Release.Artifacts.{alias}.DefinitionName	The name of the build pipeline or repository.	fabrikam-ci		TFVC: \$/fabrikam, Git: fabrikam	fabrikam/asp (master)
Release.Artifacts.{alias}.BuildNumber	The build number or the commit identifier.	20170112.1	20170112.1	TFVC: Changeset 3, Git: 38629c964	38629c964

VARIABLE NAME	DESCRIPTION	AZURE PIPELINES EXAMPLE	JENKINS/TEAMCITY EXAMPLE	TFVC/GIT EXAMPLE	GITHUB EXAMPLE
---------------	-------------	-------------------------	--------------------------	------------------	----------------

Release.Artifacts.{alias}.BuildId	The build identifier.	130	130		38629c964d21fe 405ef830b7d022 0966b82c9e11
Release.Artifacts.{alias}.BuildURI	The URL for the build.	vstfs:///build-release /Build/130			
Release.Artifacts.{alias}.SourceBranch	The full path and name of the branch from which the source was built.	refs/heads/master			
Release.Artifacts.{alias}.SourceBranchName	The name only of the branch from which the source was built.	master			
Release.Artifacts.{alias}.SourceVersion	The commit that was built.	bc0044458ba1d9 298 cdc649cb5dcf013 180706f7			
Release.Artifacts.{alias}.Repository.Provider	The type of repository from which the source was built	Git			
Release.Artifacts.{alias}.RequestedForID	The identifier of the account that triggered the build.	2f435d07-769f-4e46-849d-10d1ab9ba6ab			
Release.Artifacts.{alias}.RequestedFor	The name of the account that requested the build.	Mateo Escobedo			
Release.Artifacts.{alias}.Type	The type of artifact source, such as Build.	Build	Jenkins: Jenkins, TeamCity: TeamCity	TFVC: TFVC, Git: Git	GitHub

See also [Artifact source alias](#)

Primary artifact variables

You designate one of the artifacts as a primary artifact in a release pipeline. For the designated primary artifact, Azure Pipelines populates the following variables.

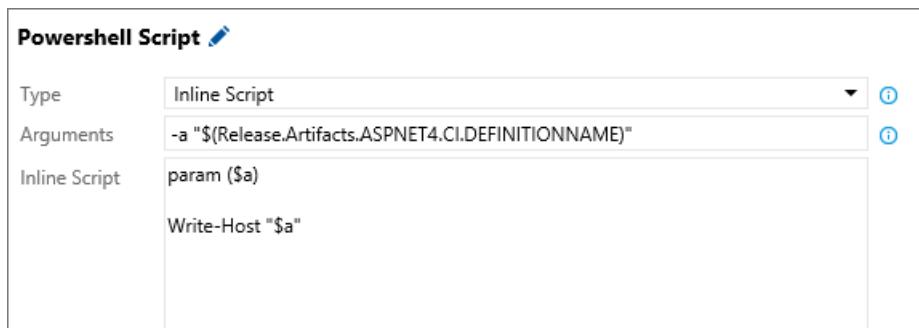
VARIABLE NAME	SAME AS
Build.DefinitionId	Release.Artifacts.{Primary artifact alias}.DefinitionId
Build.DefinitionName	Release.Artifacts.{Primary artifact alias}.DefinitionName
Build.BuildNumber	Release.Artifacts.{Primary artifact alias}.BuildNumber
Build.BuildId	Release.Artifacts.{Primary artifact alias}.BuildId
Build.BuildURI	Release.Artifacts.{Primary artifact alias}.BuildURI
Build.SourceBranch	Release.Artifacts.{Primary artifact alias}.SourceBranch
Build.SourceBranchName	Release.Artifacts.{Primary artifact alias}.SourceBranchName
Build.SourceVersion	Release.Artifacts.{Primary artifact alias}.SourceVersion
Build.Repository.Provider	Release.Artifacts.{Primary artifact alias}.Repository.Provider
Build.RequestedForID	Release.Artifacts.{Primary artifact alias}.RequestedForID
Build.RequestedFor	Release.Artifacts.{Primary artifact alias}.RequestedFor
Build.Type	Release.Artifacts.{Primary artifact alias}.Type

Using default variables

You can use the default variables in two ways - as parameters to tasks in a release pipeline or in your scripts.

You can directly use a default variable as an input to a task. For example, to pass

`Release.Artifacts.{Artifact alias}.DefinitionName` for the artifact source whose alias is **ASPNET4.CI** to a task, you would use `$(Release.Artifacts.ASPNET4.CI.DefinitionName)`.



To use a default variable in your script, you must first replace the `.` in the default variable names with `_`. For example, to print the value of artifact variable `Release.Artifacts.{Artifact alias}.DefinitionName` for the artifact source whose alias is **ASPNET4.CI** in a Powershell script, you would use

`$env:RELEASE_ARTIFACTS_ASPNET4_CI_DEFINITIONNAME`.

Powershell Script

Type	Inline Script	?
Arguments		?
Inline Script	Write-Host "\$env:RELEASE_ARTIFACTS_ASPNET4_CI_DEFINITIONNAME"	?

Note that the original name of the artifact source alias, `ASPNET4.CI`, is replaced by `ASPNET4_CI`.

View the current values of all release variables

1. Open the pipelines view of the summary for the release, and choose the stage you are interested in. In the list of steps, choose **Initialize job**.

The screenshot shows the Azure DevOps Release pipeline interface. At the top, it displays the path: SampleApp - 1 > Release-2 > QA, with a green checkmark indicating 'Succeeded'. Below this, there are tabs for Pipeline, Tasks, Variables, Logs (which is selected), Tests, Deploy, Cancel, Refresh, and Download. On the left, under 'Deployment process', there's a section for 'Run on agent' which also shows 'Succeeded'. On the right, under 'Run on agent', it says 'Pool: Hosted VS2017 · Agent: Hosted Agent'. Below this, a list of steps is shown: 'Initialize Agent' (succeeded), 'Initialize job' (succeeded, highlighted with a red box), and 'Download artifact - DotNetSample-ASP.NET Core-CI' (succeeded).

2. This opens the log for this step. Scroll down to see the values used by the agent for this job.

The screenshot shows the log output for the 'Initialize job' step. The log entries are as follows:

```

1 2018-08-23T10:44:08.4457681Z ##[section]Starting: Initialize job
2 2018-08-23T10:44:08.4458248Z Current agent version: '2.139.0'
3 2018-08-23T10:44:08.4487538Z Prepare release directory.
4 2018-08-23T10:44:08.4501806Z ReleaseId=2, TeamProjectId=57633bf3-e6f1-4d73-8e33-89b718255fc0
5 2018-08-23T10:44:08.4679318Z Release folder: D:\a\r1\a
6 2018-08-23T10:44:08.4837852Z Environment variables available are below. Note that these env
7   [AGENT_HOMEDIRECTORY] --> [C:\agents\2.139.0]
8   [AGENT_ID] --> [3]
9   [AGENT_JOBNAME] --> [Release]
10  [AGENT_MACHINENAME] --> [factoryvm-az24]
11  [AGENT_NAME] --> [Hosted Agent]
12  [AGENT_OS] --> [Windows_NT]
13  [AGENT_RELEASEDIRECTORY] --> [D:\a\r1\a]
14  [AGENT_ROOTDIRECTORY] --> [D:\a]
15  [AGENT_SERVEROMDIRECTORY] --> [C:\agents\2.139.0\externals\vstsom]
16  [AGENT_TEMPDIRECTORY] --> [D:\a\_temp]
17  [AGENT_TOOLSDIRECTORY] --> [C:/hostedtoolcache/windows]

```

Run a release in debug mode

Show additional information as a release executes and in the log files by running the entire release, or just the tasks in an individual release stage, in debug mode. This can help you resolve issues and failures.

- To initiate debug mode for an entire release, add a variable named `System.Debug` with the value `true` to the **Variables** tab of a release pipeline.
- To initiate debug mode for a single stage, open the **Configure stage** dialog from the shortcut menu of the stage and add a variable named `System.Debug` with the value `true` to the **Variables** tab.

- Alternatively, create a [variable group](#) containing a variable named `System.Debug` with the value `true` and link this variable group to a release pipeline.

If you get an error related to an Azure RM service connection, see [How to: Troubleshoot Azure Resource Manager service connections](#).

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Release approvals and gates overview

11/15/2018 • 2 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

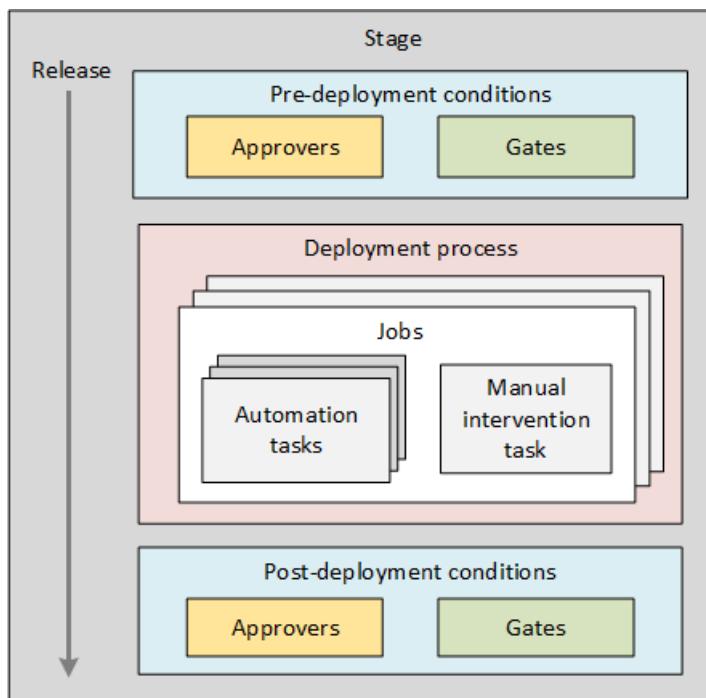
Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

A release pipeline specifies the end-to-end release pipeline for an app to be deployed across a range of stages. Deployments to each stage are fully automated by using [jobs](#) and [tasks](#).

Approvals and **gates** give you additional control over the start and completion of the deployment pipeline. Each stage in a release pipeline can be configured with pre-deployment and post-deployment conditions that can include waiting for users to manually approve or reject deployments, and checking with other automated systems until specific conditions 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.

At present, gates are available only in Azure Pipelines.

The following diagram shows how these features are combined in a stage of a release pipeline.



By using approvals, gates, and manual intervention you can take full control of your releases to meet a wide range of deployment requirements. Typical scenarios where approvals, gates, and manual intervention are useful include the following.

SCENARIO	FEATURE(S) TO USE
Some users must manually validate the change request and approve the deployment to a stage.	Pre-deployment approvals

SCENARIO	FEATURE(S) TO USE
Some users must manually sign off the app after deployment before the release is promoted to other stages.	Post-deployment approvals
You want to ensure there are no active issues in the work item or problem management system before deploying a build to a stage.	Pre-deployment gates
You want to ensure there are no incidents from the monitoring or incident management system for the app after it's been deployed, before promoting the release.	Post-deployment gates
After deployment you want to wait for a specified time before prompting some users for a manual sign-off.	Post-deployment gates and post-deployment approvals
During the deployment pipeline a user must manually follow specific instructions and then resume the deployment.	Manual Intervention
During the deployment pipeline you want to prompt the user to enter a value for a parameter used by the deployment tasks, or allow the user to edit the details of this release.	Manual Intervention
During the deployment pipeline you want to wait for monitoring or information portals to detect any active incidents, before continuing with other deployment jobs.	Planned

You can, of course, combine all three techniques within a release pipeline to fully achieve your own deployment requirements.

Related topics

- [Approvals](#)
- [Gates](#)
- [Manual intervention](#)
- [Stages](#)
- [Triggers](#)
- [Release pipelines and releases](#)

See also

- [Video: Deploy quicker and safer with gates in Azure Pipelines](#)
- [Configure your release pipelines for safe deployments](#)

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Release deployment control using approvals

10/9/2018 • 2 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

When a release is created from a release pipeline that defines approvals, the deployment stops at each point where approval is required until the specified approver grants approval or rejects the release (or re-assigns the approval to another user). You can enable manual deployment approvals for each stage in a release pipeline.

Define a deployment approval

You can define approvals at the start of a stage (pre-deployment approvers), at the end of a stage (post-deployment approvers), or both. For details of how to define and use approvals, see [Add approvals within a release pipeline](#).

- For a **pre-deployment** approval, choose the icon at the entry point of the stage and enable pre-deployment approvers.
- For a **post-deployment** approval, choose the icon at the exit point of the stage and enable post-deployment approvers.

You can add multiple approvers for both pre-deployment and post-deployment settings. These approvers can be individual users or groups of users. When a group is specified as an approver, only one of the users in that group needs to approve for the deployment to occur or the release to move forward.

- If you are using **Azure Pipelines**, you can use local groups managed in Azure Pipelines or Azure Active Directory (Azure AD) groups if they have been added into Azure Pipelines.
- If you are using **Team Foundation Server** (TFS), you can use local groups managed in TFS or Active Directory (AD) groups if they have been added into TFS.

The creator of a deployment is considered to be a separate user role for deployments. For more details, see [Release permissions](#). Either the release creator or the deployment creator can be restricted from approving deployments.

If no approval is granted within the **Timeout** specified for the approval, the deployment is rejected.

Use the **Approval policies** to:

- Specify that the user who requested (initiated or created) the release cannot approve it. If you are experimenting with approvals, uncheck this option so that you can approve or reject your own deployments.
- Force a revalidation of the user identity to take into account recently changed permissions.
- Reduce user workload by automatically approving subsequent prompts if the specified user has already approved the deployment to a previous stage in the pipeline (applies to pre-deployment approvals only). Take care when using this option; for example, you may want to require a user to physically approve a deployment to production even though that user has previously approved a deployment to a QA stage in the same release pipeline.

For information about approving or rejecting deployments, and viewing approval logs, see [Create a release](#), [View the logs for approvals](#), and [Monitor and track deployments](#).

Approval notifications

Notifications such as an email message can be sent to the approver(s) defined for each approval step. Configure recipients and settings in the **Notifications** section of the [project settings page](#).

Build	
Build completes Build completes	Build completed (any project)
Code (Git)	
Pull request reviewers added or removed Notifies the team when it is added or removed as a reviewer for a pull request	Pull request (any project)
Pull request changes Notifies the team when changes are made to a pull request the team is a reviewer for	Pull request (any project)
Release	
Manual intervention pending Notifies the team when a manual intervention is pending on the team	Deployment pending (any project)
Deployment to an owned environment failed Notifies the team when a deployment to an environment team owns fails to complete	Deployment comple... (any project)
Deployment to an approved environment failed Notifies the team when a deployment team approved fails to complete successfully	Deployment comple... (any project)
Deployment completion failures Notifies the team when a deployment team requested fails to complete successfully	Deployment comple... (any project)
Deployment approval pending Notifies the team when an approval for a deployment is pending on the team	Release approval pe... (any project)

The link in the email message opens the **Summary** page for the release where the user can approve or reject the release.

Related topics

- [Approvals and gates overview](#)
- [Manual intervention](#)
- [Stages](#)
- [Triggers](#)

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Release deployment control using gates

11/6/2018 • 4 minutes to read • [Edit Online](#)

Azure Pipelines

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.

Scenarios for gates

Some scenarios and use cases for gates are:

- **Incident and issues management.** Ensure the required status for work items, incidents, and issues. For example, ensure deployment occurs only if no priority zero bugs exist, and validation that there are no active incidents takes place after deployment.
- **Seek approvals outside Azure Pipelines.** Notify non-Azure Pipelines 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 deploy only 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 just 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 healthy resource utilization and a positive security report.

Most of the health parameters vary over time, regularly changing their status from healthy to unhealthy and back to healthy. To account for such variations, all the gates are periodically re-evaluated until all of them are successful at the same time. The release execution and deployment does not proceed if all gates do not succeed in the same interval and before the configured timeout.

Define a gate for a stage

You can enable gates at the start of a stage (in the **Pre-deployment conditions**) or at the end of a stage (**Post-deployment conditions**), or both. For details of how to enable gates, see [Configure a gate](#).

The **Delay before evaluation** is a time delay at the beginning of the gate evaluation process that allows the gates to initialize, stabilize, and begin providing accurate results for the current deployment (see [Gate evaluation flows](#)). For example:

- For **pre-deployment gates**, the delay would be the time required for all bugs to be logged against the artifacts being deployed.
- For **post-deployment gates**, the delay would be the maximum of the time taken for the deployed app to reach a steady operational state, the time taken for execution of all the required tests on the deployed stage,

and the time it takes for incidents to be logged after the deployment.

The following gates are available by default:

- **Invoke Azure function**: Trigger execution of an Azure function and ensure a successful completion. For more details, see [Azure function task](#).
- **Query Azure monitor alerts**: Observe the configured Azure monitor alert rules for active alerts. For more details, see [Azure monitor task](#).
- **Invoke REST API**: Make a call to a REST API and continue if it returns a successful response. For more details, see [HTTP REST API task](#).
- **Query Work items**: Ensure the number of matching work items returned from a query is within a threshold. For more details, see [Work item query task](#).

You can [create your own gates](#) with Marketplace extensions.

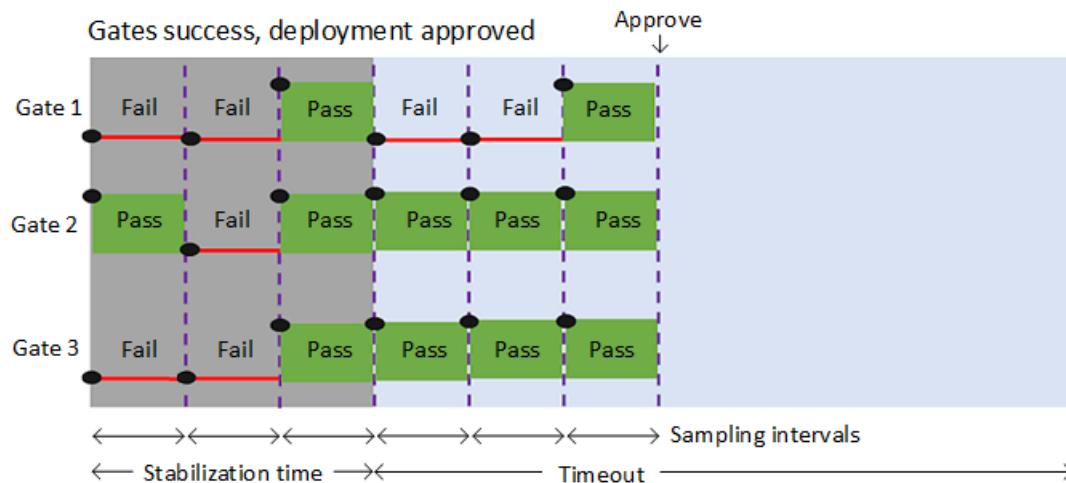
The evaluation options that apply to all the gates you've added are:

- **Time between re-evaluation of gates**. The time interval between successive evaluations of the gates. At each sampling interval, new requests are sent concurrently to each gate and the new results are evaluated. It is recommended that the sampling interval is greater than the longest typical response time of the configured gates to allow time for all responses to be received for evaluation.
- **Timeout after which gates fail**. The maximum evaluation period for all gates. The deployment will be rejected if the timeout is reached before all gates succeed during the same sampling interval.
- **Gates and approvals**. Select the required order of execution for gates and approvals if you have configured both. For pre-deployment conditions, the default is to prompt for manual (user) approvals first, then evaluate gates afterwards. This saves the system from evaluating the gate functions if the release is rejected by the user. For post-deployment conditions, the default is to evaluate gates and prompt for manual approvals only when all gates are successful. This ensures the approvers have all the information required for a sign-off.

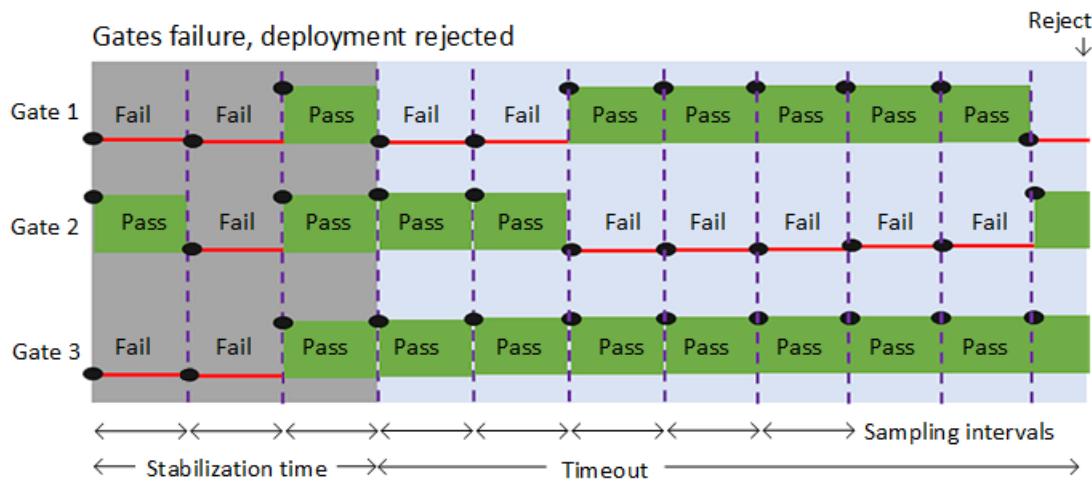
For information about viewing gate results and logs, see [View the logs for approvals](#) and [Monitor and track deployments](#).

Gate evaluation flow examples

The following diagram illustrates the flow of gate evaluation where, after the initial stabilization delay period and three sampling intervals, the deployment is approved.



The following diagram illustrates the flow of gate evaluation where, after the initial stabilization delay period, not all gates have succeeded at each sampling interval. In this case, after the timeout period expires, the deployment is rejected.



Related topics

- [Approvals and gates overview](#)
- [Manual intervention](#)
- [Use approvals and gates to control your deployment](#)
- [Stages](#)
- [Triggers](#)

See also

- [Video: Deploy quicker and safer with gates in Azure Pipelines](#)
- [Configure your release pipelines for safe deployments](#)
- [Tutorial: Use approvals and gates to control your deployment](#)
- [Twitter sentiment as a release gate](#)
- [GitHub issues as a release gate](#)
- [Author custom gates. Library with examples](#)

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Videos

Stage templates in Azure Pipelines

10/9/2018 • 2 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions.

Service connections are called *service endpoints* in TFS 2018 and in older versions.

When you start a new release pipeline, or when you add a stage to an existing release pipeline, you can choose from a list of templates for each stage. These templates pre-populate the stage with the appropriate tasks and settings, which can considerably reduce the time and effort required to create a release pipeline for your DevOps CI/CD processes.

A set of pre-defined stage templates are available in Azure Pipelines and in each version of TFS. You can use these templates when you create a new release pipeline or add a new stage to a pipeline. You can also create your own custom stage templates from a stage you have populated and configured.

Templates do not have any additional security capability. There is no way to restrict the use of a template to specific users. All templates, pre-defined and custom, are available for use by all users who have permission to create release pipelines.

When a stage is created from a template, the tasks in the template are copied over to the stage. Any further updates to the template have no impact on existing stages. If you want a way to easily insert a number of stages into release pipelines (perhaps to keep the definitions consistent) and to enable these stages to all be updated in one operation, use [task groups](#) instead of stage templates.

Q & A

Can I export templates or share them with other subscriptions, enterprises, or projects?

Custom templates that you create are scoped to the project that you created them in. Templates cannot be exported or shared with another project, collection, server, or organization. You can, however, export a release pipeline and import it into another project, collection, server, or subscription. Then you can re-create the template for use in that location.

Can I publish or consume new templates through extensions in VS Marketplace?

Yes. See [Adding release stage templates to your VSS extension](#) for more details.

How do I delete a custom stage template?

You can delete an existing custom template from the list of templates that is displayed when you add a new stage to our pipeline.

Select a Template

Or start with an [Empty process](#)

applications to IIS website.

IIS IIS Website and SQL Database offline upgrade

Deployment Group: Upgrade ASP.Net, ASP.Net core based websites. Upgrade SQL database using SQL scripts executed when web application is offline.

IIS IIS Website and SQL Database online upgrade

Deployment Group: Upgrade ASP.Net, ASP.Net core based websites. Upgrade SQL database using SQL scripts executed when web application is online.

Custom



DevTemplate

Custom Dev environment template

Apply



How do I update a custom stage template?

To update an stage template, delete the existing template in a release pipeline and then save the stage as a template with the same name.

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Azure policy compliance

11/15/2018 • 3 minutes to read • [Edit Online](#)

Azure Pipelines

Azure Policy helps you manage and prevent IT issues by using policy definitions that enforce rules and effects for your resources. When you use Azure Policy, resources stay compliant with your corporate standards and service level agreements. Policies can be applied to an entire subscription, a management group, or a resource group.

This topic describes how you can apply Azure policies in your build and release pipelines. For example, you could apply an Azure policy that limits deployments to a specific set of virtual machines. There are several pre-defined policies that you can apply, or you can define your own policies in Azure.

For more information, see [What is Azure Policy?](#) and [Create and manage policies to enforce compliance](#).

Use a release pipeline to create and apply an Azure policy

Start a new release and choose the **Azure Policy Deployment** template. This adds two instances of the [Azure PowerShell task](#) to the **Run on agent** job. These tasks contain pre-defined inline scripts and a set of parameters.

Create an Azure policy

The **Create Azure Policy** task helps you [create a new Azure policy programmatically](#) using Azure PowerShell cmdlets. There are example policy definitions in JSON format available in the [Azure policy documentation](#) and on [GitHub](#). A reference to the PowerShell cmdlets for Azure policy is available at [AzureRM.Resources - Policies](#).

The script for the **Create Azure Policy** task has the following parameters:

PARAMETER	DESCRIPTION
policyName	Required. Policy definition name.
policyDisplayName	Optional. Policy definition display name.
policyDescription	Optional. Policy definition description.
subscriptionId	Optional. ID of the subscription in which the definition is available.
managementGroupName	Optional. Name of management group in which the definition is available.
policyRule	Required. Policy definition rule in JSON string format, or path to a file containing the JSON policy definition rule . The path can be fully qualified or relative to \$(System.DefaultWorkingDirectory).
policyParameters	Optional. Policy parameter values in JSON string format.

Assign an Azure policy

After you create an Azure policy, you can assign it by using the **Assign Azure Policy** task. Alternatively, you can create - or select an existing or a pre-defined policy - in the Azure portal as described in [this topic](#) and then assign it using the task.

The script for the **Assign Azure Policy** task has the following parameters:

PARAMETER	DESCRIPTION
AssignmentName	Required. Policy assignment name.
AssignmentDisplayName	Optional. Policy assignment display name.
AssignmentDescription	Optional. Policy assignment description.
PolicyName	Optional. Name of policy definition to assign.
PolicySetName	Optional. Name of policy set definition to assign.
ResourceGroupName	Optional. Name of resource group to which the policy [set] definition will be applied.
SubscriptionId	Optional. ID of the subscription to which the policy [set] definition will be applied.
ManagementGroupName	Optional. Name of management group to which the policy [set] definition will be applied.
PolicyParameters	Optional. Policy parameter values in JSON string format.

The screenshot shows the Azure DevOps pipeline editor. On the left, there is a sidebar with a tree view of the pipeline stages: 'Staging' (Deployment process), 'Run on agent' (Run on agent), 'Assign Azure Policy' (selected, Azure CLI), 'Deploy Infrastructure' (Azure Resource Group Deployment), and 'Deploy Application' (Azure App Service Deploy). The 'Assign Azure Policy' task is highlighted with a blue selection bar. To the right of the sidebar, the task configuration pane is open, showing the following fields:

- Azure CLI**: Version 1.*
- Display name ***: Assign Azure Policy
- Azure subscription ***: RMPM (afc11291-9826-46be-b852-70349146ddf8)
- Script Location ***: Script path
- Script Path ***: \$(System.DefaultWorkingDirectory)/arm-website/policyAssignmentScript.json
- Arguments**: (empty)

Run a release pipeline to confirm Azure policy

When a release pipeline runs and attempts to perform an action disallowed by a defined policy, the deployment is marked as **Failed**. The error message contains a link to view policy violations.

The screenshot shows the 'Environments' section of the Azure Pipelines interface. On the left, a card for the 'Staging' environment is displayed with a red border and a large red 'X' icon labeled 'Failed'. Below the card, it says '2 issues' and 'on 5/7/2018 2:28 PM'. On the right, the 'Overview' tab is selected under the 'Environment' header. It shows a timeline of events:

- A red circle with a white 'X' and the text 'Deployment failed at 19:00, 18/8/17'.
- A green circle with a checkmark and the text 'Pre-deployment approved at 18:06, 18/8/17'.
- A green circle with a checkmark and the text 'Automatically triggered at 18:00, 18/8/17'.
- A green circle with a checkmark and the text 'Work items and Commits' followed by a detailed list: 'Changes compared to Contosoapp_VSO.RM.CI_m123_Sep10.21.4 (last successful deployment) will be deployed', 'VSO.RM.CI', '9 commits and 2 workitems', and 'VSO.RM.CIALTER'.

Below the timeline, there is a section titled '3 issues (2 errors)' with two items listed:

- The template deployment failed because of policy violation. Please see details for more information.
- Request disallowed by Azure Policy. [Click here](#) for more information on Azure Portal

An error message is written to the logs and displayed in the stage status panel in the releases page of Azure pipelines.

The screenshot shows the 'Run on agent' log for a pipeline run on the 'Hosted VS2017' queue. The log entries are as follows:

- Initialize Agent · succeeded
- Initialize Job · succeeded
- Download Artifacts · succeeded
- Download artifact - html-website-static-CI · succeeded
- Azure Deployment:Create Or Update Resource Group action on aloka · 4 errors

The '4 errors' entry is expanded to show the detailed error message:

```
✖ The template deployment failed because of policy violation. Please see details for more information.
✖ Details:
✖ RequestDisallowedByPolicy: Resource 'AlokSampleHTMLCodeFlowAppHello' was disallowed by policy. Policy is 'Microsoft.Authorization/policyDefinitions/6c112d4e-5bc7-47ae-a041-ea2d9dccd749', "policyAssignmentId" is '9146ddf8/resourceGroups/aloka/providers/Microsoft.Authorization/policyAssignments/8ee0ce511faa409fa9380'.
✖ [More information on Azure Portal](URL)
✖ Task failed while creating or updating the template deployment.
```

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Releases in Azure Pipelines

10/9/2018 • 2 minutes to read • [Edit Online](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions.

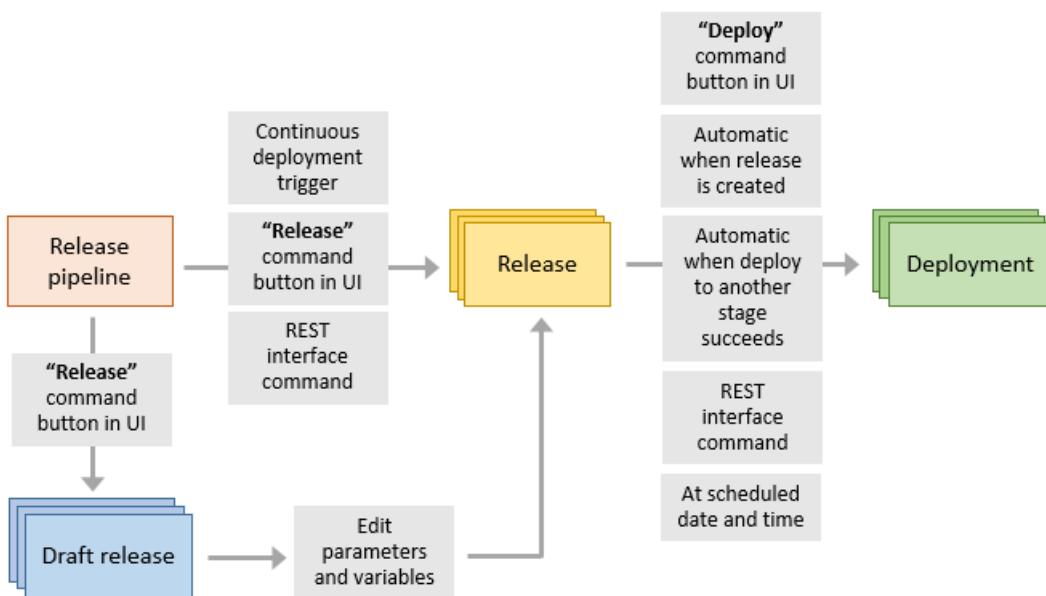
Service connections are called *service endpoints* in TFS 2018 and in older versions.

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015

A **release** is the package or container that holds a versioned set of artifacts specified in a **release pipeline** in your DevOps CI/CD processes. 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**, 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 released pipeline, and information about each one is stored and displayed in Azure Pipelines for the specified **retention period**.

A **deployment** is the action of running the **tasks** for one stage, which results in the application **artifacts** being deployed, tests being run, and whatever other actions 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.

The following schematic shows the relationship between release pipelines, releases, and deployments.



Releases (and, in some cases, draft releases) can be created from a release pipeline in several ways:

- By a **continuous deployment trigger** that creates a release when a new version of the source build artifacts is available.
- By using the **Release** command in the UI to create a release manually from the Releases or the Builds summary.
- By sending a command over the network to the **REST interface**.

However, the action of creating a release **does not** mean it will automatically or immediately start a deployment. For example:

- There may be [deployment triggers](#) defined for a stage, which force the deployment to wait; this could be for a manual deployment, until a scheduled day and time, or for successful deployment to another stage.
- A deployment started manually from the **[Deploy]** command in the UI, or from a network command sent to the [REST interface](#), may specify a final target stage other than the last stage in a release pipeline. For example, it may specify that the release is deployed only as far as the QA stage and not to the production stage.
- There may be [queuing policies](#) defined for a stage, which specify which of multiple deployments will occur, or the order in which releases are deployed.
- There may be [pre-deployment approvers or gates](#) defined for a stage, and the deployment will not occur until all necessary approvals have been granted.
- Approvers may defer the release to a stage until a specified date and time using a [scheduled trigger](#).

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Build source repositories

11/8/2018 • 2 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#) | [Previous versions \(XAML builds\)](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called service endpoints in TFS 2018 and in older versions.

Configure a repository

At the beginning of a pipeline, the agent downloads files from your repository into a local sources directory.

If your pipeline consists of multiple jobs, the agent downloads source files at the beginning of each job. You can specify only one source repository for your entire pipeline.

Azure Pipelines: To specify the source repository, while editing your pipeline, click the **YAML** or **Tasks** tab, then click **Get sources**, and then select the type of repo that contains your source files.

TFS 2018: To specify the source repository, while editing your pipeline, click the **Tasks** tab, then click **Get sources**, and then select the type of repo that contains your source files.

TFS 2017: To specify the source repository:

- **TFS 2017.3** Click the **Tasks** tab, then click **Get sources**, and then select the type of repo that contains your source files.
- **TFS 2017 RTM** Click the **Repository** tab, and then select the type of repo that contains your source files.

TFS 2015: To specify the source repository, click the **Repository** tab, and then select the type of repo that contains your source files.

Supported repository types

You can choose from the following repository types:

REPOSITORY TYPE	AZURE PIPELINES (YAML)	AZURE PIPELINES (VISUAL DESIGNER)	TFS 2018, TFS 2017, TFS 2015.4	TFS 2015 RTM
Azure Repos Git	Yes	Yes	Yes	Yes
Azure Repos TFVC	No	Yes	Yes	Yes
Bitbucket Cloud	No	Yes	No	No
External Git (generic)	No	Yes	Yes	Yes
GitHub	Yes	Yes	No	No
GitHub Enterprise	Yes	Yes	TFS 2018.2 or 2018.3	No

REPOSITORY TYPE	AZURE PIPELINES (YAML)	AZURE PIPELINES (VISUAL DESIGNER)	TFS 2018, TFS 2017, TFS 2015.4	TFS 2015 RTM
Subversion	No	Yes	Yes	No

Specific repository details

See details about building specific repository types:

- [Build Azure Repos Git repositories](#)
- [Build GitHub repositories](#)
- [Build TFVC repositories](#)

NOTE

To build code from Subversion, you must install a Subversion client (`svn`) on your [self-hosted build agents](#) or use [Microsoft-hosted build agents](#).

Options for Git repositories

See [Pipeline options for Git repositories](#) for options available to pipelines that use a Git repository.

Q & A

Why are some repository types not supported by on-premises installations?

When a pipeline uses a remote, 3rd-party repository host such as Bitbucket Cloud, the repository is configured with webhooks that notify Azure Pipelines Server or TFS when code has changed and a build should be triggered. Since on-premises installations are normally protected behind a firewall, 3rd-party webhooks are unable to reach the on-premises server. As a workaround, you can use the **External Git** repository type which uses polling instead of webhooks to trigger a build when code has changed.

How do I reference the directories on the build agent?

Reference directories using build variables such as `$(Build.SourcesDirectory)` and `$(Build.BinariesDirectory)`. To learn more, see [Build variables](#).

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Build Azure Repos Git or TFS Git repositories

11/19/2018 • 3 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#) | [Previous versions \(XAML builds\)](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Azure Pipelines can automatically build and validate every pull request and commit to your Azure Repos Git repository.

Introduction to creating pipelines

For an introduction to creating a pipeline, follow the steps in [Create your first pipeline](#).

Options for Git repositories

See [Pipeline options for Git repositories](#) for options available to pipelines that use a Git repository.

Make open source projects public

If your Azure Repos Git repository is open source, you can make your Azure DevOps project **public** so that anyone can view your pipeline's build results, logs, and test results without signing in. When users submit pull requests, they can view the status of builds that automatically validate those pull requests. See [Create a public project](#).

NOTE

Azure Repos Git repositories do not support forks by users who do not have explicit access to the project.

Access restrictions

Be aware of the following access restrictions when you're running builds in Azure Pipelines public projects:

- **Cross-project access:** All builds in a Azure DevOps public project run with an access token restricted to the project. Builds in a public project can access resources such as build artifacts or test results only within the project and not from other projects of the Azure DevOps organization.
- **Azure Artifacts packages:** If your builds need access to packages from Azure Artifacts, you must explicitly grant permission to the **Project Build Service** account to access the package feeds.

Choose a repository to build

While creating a pipeline, to choose the repository to build, first select the project to which the repository belongs. Then, select the repository.

Authorize access to your repositories

Azure Pipelines must be granted access to your repositories to display them, trigger their builds, and fetch their

code during builds.

If the repository that you wish to build is in the same project as your pipeline, you're all set. Your builds will automatically have access to the repository. If the repository is in a different project than your pipeline, you must have read access to the project and repository.

Protect branches with validation builds

You can run a validation build with each commit or pull request that targets a branch, and even prevent pull requests from merging until a validation build succeeds.

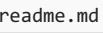
To configure validation builds for an Azure Repos Git repository, you must be a project administrator of its project.

1. First, from the Azure Repos **Branches** page, make sure that your repository is selected.
2. Next, hover over the branch you wish to protect, click  to display its context menu, and then select **Branch policies**.
3. Finally, click **Add build policy** and choose the pipeline and methods of protecting the branch as detailed in the Azure Repos documentation [here](#).

Validate contributions from forks

Building pull requests from Azure Repos forks is no different from building pull requests within the same repository or project. You can create forks only within the same Azure DevOps organization that your project is part of.

Add a build badge

To add a build badge to the  file at the root of your repository, follow the steps in [Get the status badge](#).

Parallel jobs and time limits

If your pipeline and Azure Repos Git repository are both in an Azure DevOps Services public project, then Azure Pipelines jobs are free. These free jobs have a maximum timeout of 360 minutes (6 hours) each.

If either your pipeline or Azure Repos Git repository is in a private project, then you can run up to 1,800 minutes (30 hours) of jobs for free every month. These free jobs have a maximum timeout of 30 minutes each. Purchasing jobs for private projects or private repositories removes any monthly time limit and allows jobs to have a maximum timeout of 360 minutes (6 hours) each.

To adjust the timeout of jobs, see [Timeouts](#).

Learn more about pricing based on [parallel jobs](#).

Build GitHub repositories

11/19/2018 • 17 minutes to read • [Edit Online](#)

Azure Pipelines

Azure Pipelines can automatically build and validate every pull request and commit to your GitHub repository.

Introduction to creating pipelines

For an introduction to creating a pipeline for a GitHub repo, follow the steps in [Create your first pipeline](#).

Options for Git repositories

See [Pipeline options for Git repositories](#) for options available to pipelines that use a Git repository.

Make open source projects public

If your GitHub repository is open source, you can make your Azure DevOps project **public** so that anyone can view your pipeline's build results, logs, and test results without signing in. When users outside your organization fork your repository and submit pull requests, they can view the status of builds that automatically validate those pull requests. See [Create a public project](#).

Access restrictions

Be aware of the following access restrictions when you're running builds in Azure Pipelines public projects:

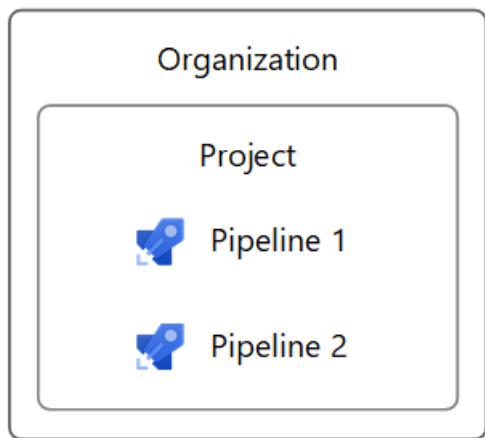
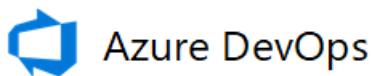
- **Build secrets:** By default, secrets associated with your build pipeline are not made available to pull request builds of forks. See [Validate contributions from forks](#).
- **Cross-project access:** All builds in a Azure DevOps public project run with an access token restricted to the project. Builds in a public project can access resources such as build artifacts or test results only within the project and not from other projects of the Azure DevOps organization.
- **Azure Artifacts packages:** If your builds need access to packages from Azure Artifacts, you must explicitly grant permission to the **Project Build Service** account to access the package feeds.

Map GitHub organizations to Azure DevOps

GitHub's structure consists of **organizations and user accounts** that contain **repositories**. See [GitHub's documentation](#).

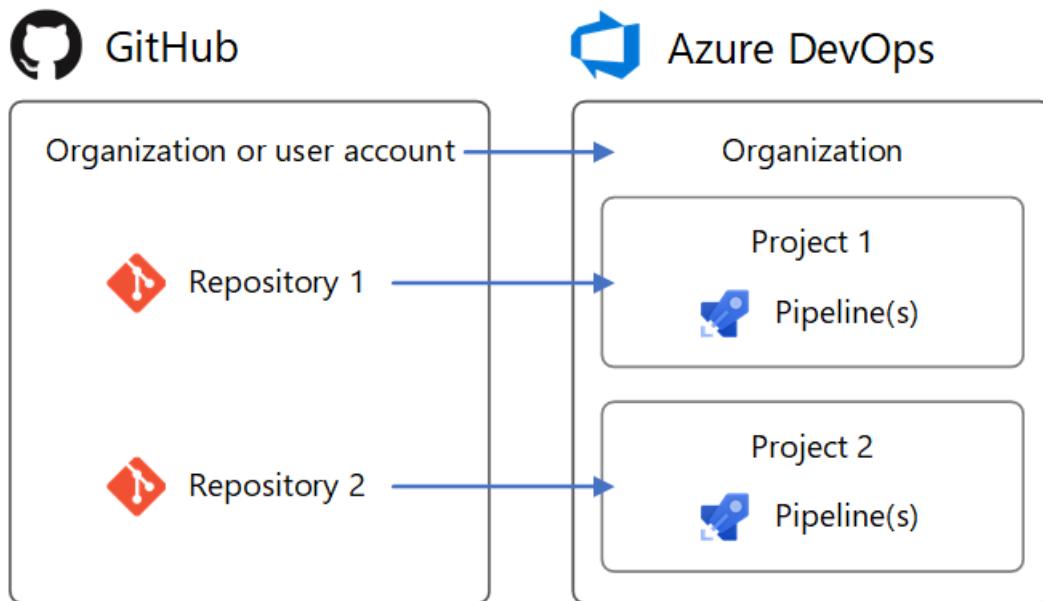


Azure DevOps' structure consists of **organizations** that contain **projects**. See [Plan your Azure DevOps organizational structure](#).



Azure DevOps can reflect your GitHub structure with:

- An Azure DevOps **organization** for your GitHub **organization or user account**
- Azure DevOps **projects** for your GitHub **repositories**



To set up this mapping between GitHub and Azure DevOps:

1. Create an Azure DevOps organization named after your GitHub organization or user account. It will have a URL like <https://dev.azure.com/your-organization>.
2. In the Azure DevOps organization, create projects named after your repositories. They will have URLs like <https://dev.azure.com/your-organization/your-repository>.
3. In the Azure DevOps projects, create pipelines named after the GitHub organization and repository they build, such as `your-organization.your-repository`. Then, it's clear which repositories they're for.

Following this pattern, your GitHub repositories and Azure DevOps projects will have matching URL paths. For example:

GitHub	https://github.com/python/cpython
Azure DevOps	https://dev.azure.com/your-organization/your-repository

Mapping GitHub permissions to Azure DevOps

GitHub permissions on GitHub organizations, user accounts, and repositories can be reflected in Azure DevOps.

Mapping GitHub organization roles

GitHub organization member roles are found at <https://github.com/orgs/your-organization/people> (replace `your-organization`).

Azure DevOps organization member permissions are found at

https://dev.azure.com/your-organization/_settings/security (replace `your-organization`).

GitHub organization roles map to Azure DevOps organization permissions as follows.

GITHUB ORGANIZATION ROLE	AZURE DEVOPS ORGANIZATION EQUIVALENT
Owner	Member of <code>Project Collection Administrators</code>
Billing manager	Member of <code>Project Collection Administrators</code>
Member	Member of <code>Project Collection Valid Users</code> . By default, this group lacks permission to create new projects. To change this, set the group's <code>Create new projects</code> permission to <code>Allow</code> , or create a new group with permissions you need.

Mapping GitHub user account roles

A GitHub user account has 1 role, which is ownership of the account.

Azure DevOps organization member permissions are found at

https://dev.azure.com/your-organization/_settings/security (replace `your-organization`).

The GitHub user account role maps to Azure DevOps organization permissions as follows.

GITHUB USER ACCOUNT ROLE	AZURE DEVOPS ORGANIZATION EQUIVALENT
Owner	Member of <code>Project Collection Administrators</code>

Mapping GitHub repository permissions

GitHub repository permissions are found at

<https://github.com/your-organization/your-repository/settings/collaboration> (replace `your-organization` and `your-repository`).

Azure DevOps project permissions are found at

https://dev.azure.com/your-organization/your-project/_settings/security (replace `your-organization` and `your-project`).

GitHub repository permissions map to Azure DevOps project permissions as follows.

GITHUB REPOSITORY PERMISSION	AZURE DEVOPS PROJECT EQUIVALENT
Admin	Member of <code>Project Administrators</code>
Write	Member of <code>Contributors</code>
Read	Member of <code>Readers</code>

If your GitHub repository grants permission to teams, you can create matching teams in the **Teams** section of your Azure DevOps project settings. Then, add the teams to the security groups above, just like users.

Pipeline-specific permissions

To grant permissions to users or teams for specific pipelines in an Azure DevOps project, follow these steps:

1. Visit the project's Builds page (for example, https://dev.azure.com/your-organization/your-project/_build).
2. Select the pipeline for which to set specific permissions.
3. From the '...' context menu, select **Security**.
4. Click **Add...** to add a specific user, team, or group and customize their permissions for the pipeline.

Authorize access to your repositories

Azure Pipelines must be granted access to your repositories to display them, trigger their builds, and fetch their code during builds.

There are 3 authentication types for granting Azure Pipelines access to your GitHub repositories while creating a pipeline.

AUTHENTICATION TYPE	BUILDS RUN USING	WORKS WITH THE GITHUB CHECKS API
1. GitHub App	The Azure Pipelines identity	Yes
2. OAuth	Your personal GitHub identity	No
3. Personal access token (PAT)	Your personal GitHub identity	No

Integrate using the GitHub App

The Azure Pipelines GitHub App is the **recommended** authentication type. By installing it in your GitHub account or organization, your pipeline can run without using your personal GitHub identity. Builds and GitHub status updates will be performed using the Azure Pipelines identity. The GitHub App works with the [GitHub Checks API](#) to display build, test, and code coverage results in GitHub.

Install the GitHub App

To use the GitHub App, install it in your GitHub organization or user account. The app can be installed and uninstalled from 2 locations:

1. The app's [homepage](#) - **This is recommended** when no parallel jobs are being purchased, or when your organization pays GitHub by purchase order (PO) or invoice.
2. The app's [GitHub Marketplace listing](#) where additional parallel jobs can be purchased for private repositories, but where cancelation of the price plan may delay uninstallation until the end of your GitHub billing period, even for the free plan.

To install the GitHub App, you must be a repository admin or GitHub organization owner.

If you install the app for all repositories in a GitHub organization, you don't need to worry about Azure Pipelines sending mass emails or automatically setting up pipelines on your behalf. As an alternative to installing the app for all repositories, repo admins can install it one at a time for individual repositories. This requires more work for admins, but has no advantage nor disadvantage.

The app will become Azure Pipelines' default method of authentication to GitHub (instead of OAuth) when organization members create pipelines for their repositories. This is recommended so that pipelines run as "Azure Pipelines" instead of a user's GitHub identity which may lose access to the repository.

GitHub App permissions

The GitHub App requests the following permissions during installation:

PERMISSION	WHAT AZURE PIPELINES DOES WITH IT
Write access to code	Only upon your deliberate action, Azure Pipelines will simplify creating a pipeline by committing a YAML file to a selected branch of your GitHub repository.
Read access to metadata	Azure Pipelines will retrieve GitHub metadata for displaying the repository, branches, and issues associated with a build in the build's summary.
Read and write access to checks	Azure Pipelines will read and write its own build, test, and code coverage results to be displayed in GitHub.
Read and write access to pull requests	Only upon your deliberate action, Azure Pipelines will simplify creating a pipeline by creating a pull request for a YAML file that was committed to a selected branch of your GitHub repository. Azure Pipelines will retrieve pull request metadata to display in build summaries associated with pull requests.

GitHub Marketplace purchases

Additional, Microsoft-hosted parallel jobs can be purchased through the [Azure DevOps Marketplace](#) (recommended) or [GitHub Marketplace](#). Pricing is the same in both marketplaces. Unless you prefer using an existing GitHub billing account, it's recommended that purchases be made in the Azure DevOps Marketplace to simplify associating purchases with different Azure DevOps organizations.

The **first time** the app is installed in a GitHub organization or user account, the Azure DevOps organization that is created or selected during installation will be where GitHub Marketplace purchases are applied. Currently, the only way to change where GitHub Marketplace purchases are applied is to uninstall and reinstall the app (which will disable existing pipelines), or purchase parallel jobs through the [Azure DevOps Marketplace](#) instead of GitHub.

Create pipelines in multiple Azure DevOps organizations and projects

Once the app is installed, pipelines can be created for the organization's repositories in different Azure DevOps organizations and projects. However, if you create pipelines for a single repository in multiple Azure DevOps organizations, only the first organization's pipelines can be automatically triggered by GitHub commits or pull requests. Manual or scheduled builds are still possible in secondary Azure DevOps organizations.

Temporarily (this is actively being improved), follow these steps to create a pipeline using the GitHub App in an Azure DevOps organization or project that is different than the one chosen during the app's installation.

1. Visit <https://github.com/apps/azure-pipelines/installations/new> and click **Configure** on the GitHub organization for which you want to create a pipeline in another Azure DevOps organization or project.
 - If Configure is not shown next to the GitHub organization, that means the GitHub App is not yet installed for the GitHub organization. Click the organization name to install it and skip the remaining steps below.
2. Under "Repository Access," make sure that access is granted to the repository you wish to build. Optionally, toggle the selection between "All repositories" and "Only select repositories" so that the Save button is enabled. Then, click the **Save** button.
3. You'll be redirected to Azure DevOps to choose the organization, project, and repository for the new pipeline.

Integrate using OAuth

OAuth is the simplest authentication type to get started with for repositories in your personal GitHub account. Builds and GitHub status updates will be performed on behalf of your personal GitHub identity. For builds to keep working, your repository access must remain.

Using OAuth

To use OAuth, click **Authorize** on the repository step while creating a pipeline. The OAuth connection will be saved in your Azure DevOps project for later use.

Revoking OAuth access

After authorizing Azure Pipelines to use OAuth, to later revoke it and prevent further use, visit [OAuth Apps](#) in your GitHub settings. You can also delete it from the list of GitHub [service connections](#) in your Azure DevOps project settings.

Integrate using a personal access token (PAT)

PATs are effectively the same as OAuth, but allow you to control which permissions are granted to Azure Pipelines. Builds and GitHub status updates will be performed on behalf of your personal GitHub identity. For builds to keep working, your repository access must remain.

Using a PAT

To create a PAT, visit [Personal access tokens](#) in your GitHub settings. The required permissions are `repo`, `admin:repo_hook`, `read:user`, and `user:email`. These are the same permissions required when using OAuth, above. Copy the generated PAT to the clipboard and paste it into a new GitHub [service connection](#) in your Azure DevOps project settings. For future recall, name the service connection after your GitHub username. It will be available in your Azure DevOps project for later use.

Revoking PAT access

After authorizing Azure Pipelines to use a PAT, to later delete it and prevent further use, visit [Personal access tokens](#) in your GitHub settings. You can also delete it from the list of GitHub [service connections](#) in your Azure DevOps project settings.

Choose a repository to build

To create a pipeline for your repository with continuous integration and pull request triggers, you must have the required GitHub permissions configured. Otherwise, **the repository will not appear** in the repository list while creating a pipeline. Depending on the authentication type and ownership of the repository, ensure that the following access is configured.

Repository permissions for GitHub App authentication

IF THE REPO IS IN YOUR PERSONAL GITHUB ACCOUNT	IF THE REPO IS IN SOMEONE ELSE'S PERSONAL GITHUB ACCOUNT	IF THE REPO IS IN A GITHUB ORGANIZATION THAT YOU OWN	IF THE REPO IS IN A GITHUB ORGANIZATION THAT SOMEONE ELSE OWNS
--	--	--	--

IF THE REPO IS IN YOUR PERSONAL GITHUB ACCOUNT	IF THE REPO IS IN SOMEONE ELSE'S PERSONAL GITHUB ACCOUNT	IF THE REPO IS IN A GITHUB ORGANIZATION THAT YOU OWN	IF THE REPO IS IN A GITHUB ORGANIZATION THAT SOMEONE ELSE OWNS
Install the Azure Pipelines GitHub App in your personal GitHub account. You can do so from here .	<p>1. The other person must install the Azure Pipelines GitHub App in their personal GitHub account. They can do so from here.</p> <p>2. You must be added as a collaborator in the repository's settings under "Collaborators". Accept the invitation to be a collaborator using the link that is emailed to you.</p>	<p>1. Install the Azure Pipelines GitHub App in the GitHub organization. You can do so from here.</p> <p>2. You must be added as a collaborator, or your team must be added, in the repository's settings under "Collaborators and teams".</p>	<p>1. A GitHub organization owner or repository admin must install the Azure Pipelines GitHub App in the organization. The app can be installed from here.</p> <p>2. You must be added as a collaborator, or your team must be added, in the repository's settings under "Collaborators and teams". Accept the invitation to be a collaborator using the link that is emailed to you.</p>

Repository permissions for OAuth authentication

IF THE REPO IS IN YOUR PERSONAL GITHUB ACCOUNT	IF THE REPO IS IN SOMEONE ELSE'S PERSONAL GITHUB ACCOUNT	IF THE REPO IS IN A GITHUB ORGANIZATION THAT YOU OWN	IF THE REPO IS IN A GITHUB ORGANIZATION THAT SOMEONE ELSE OWNS
<p>1. At least once, authenticate to GitHub with OAuth using your personal GitHub account credentials. This can be done in Azure DevOps project settings under Pipelines > Service connections > New service connection > GitHub > Authorize.</p> <p>2. Grant Azure Pipelines access to your repositories under "Permissions" here.</p>	<p>1. At least once, the other person must authenticate to GitHub with OAuth using their personal GitHub account credentials. This can be done in Azure DevOps project settings under Pipelines > Service connections > New service connection > GitHub > Authorize.</p> <p>2. The other person must grant Azure Pipelines access to their repositories under "Permissions" here.</p> <p>3. You must be added as a collaborator in the repository's settings under "Collaborators". Accept the invitation to be a collaborator using the link that is emailed to you.</p>	<p>1. At least once, authenticate to GitHub with OAuth using your personal GitHub account credentials. This can be done in Azure DevOps project settings under Pipelines > Service connections > New service connection > GitHub > Authorize.</p> <p>2. Grant Azure Pipelines access to your organization under "Organization access" here.</p> <p>3. You must be added as a collaborator, or your team must be added, in the repository's settings under "Collaborators and teams".</p>	<p>1. At least once, a GitHub organization owner must authenticate to GitHub with OAuth using their personal GitHub account credentials. This can be done in Azure DevOps project settings under Pipelines > Service connections > New service connection > GitHub > Authorize.</p> <p>2. The organization owner must grant Azure Pipelines access to the organization under "Organization access" here.</p> <p>3. You must be added as a collaborator, or your team must be added, in the repository's settings under "Collaborators and teams". Accept the invitation to be a collaborator using the link that is emailed to you.</p>

Repository permissions for Personal access token (PAT) authentication

IF THE REPO IS IN YOUR PERSONAL GITHUB ACCOUNT	IF THE REPO IS IN SOMEONE ELSE'S PERSONAL GITHUB ACCOUNT	IF THE REPO IS IN A GITHUB ORGANIZATION THAT YOU OWN	IF THE REPO IS IN A GITHUB ORGANIZATION THAT SOMEONE ELSE OWNS
The PAT must have the required access scopes under Personal access tokens : <code>repo</code> , <code>admin:repo_hook</code> , <code>read:user</code> , and <code>user:email</code> .	<p>1. The PAT must have the required access scopes under Personal access tokens: <code>repo</code>, <code>admin:repo_hook</code>, <code>read:user</code>, and <code>user:email</code>.</p> <p>2. You must be added as a collaborator in the repository's settings under "Collaborators". Accept the invitation to be a collaborator using the link that is emailed to you.</p>	<p>1. The PAT must have the required access scopes under Personal access tokens: <code>repo</code>, <code>admin:repo_hook</code>, <code>read:user</code>, and <code>user:email</code>.</p> <p>2. You must be added as a collaborator, or your team must be added, in the repository's settings under "Collaborators and teams". Accept the invitation to be a collaborator using the link that is emailed to you.</p>	<p>1. The PAT must have the required access scopes under Personal access tokens: <code>repo</code>, <code>admin:repo_hook</code>, <code>read:user</code>, and <code>user:email</code>.</p> <p>2. You must be added as a collaborator, or your team must be added, in the repository's settings under "Collaborators and teams". Accept the invitation to be a collaborator using the link that is emailed to you.</p>

Protect branches with validation builds

You can run a validation build with each commit or pull request that targets a branch, and even prevent pull requests from merging until a validation build succeeds.

To configure validation builds for a GitHub repository, you must be the owner or have admin access to the repository.

1. First, create a pipeline for the repository and build it at least once so that its status is posted to GitHub, thereby making GitHub aware of the pipeline's name.
2. Next, follow GitHub's documentation for [configuring protected branches](#) in the repository's settings.

Trigger builds for GitHub tags

To trigger a build for a specific tag name or pattern, create a branch filter for your pipeline's continuous integration trigger. For the branch name, use the fully-qualified name of the tag ref, such as the following examples. See [Build pipeline triggers](#).

- `refs/tags/myTagName`
- `refs/tags/partialTagName*`
- `refs/tags/*`

Validate contributions from forks

IMPORTANT

These settings affect the security of your build.

When you create a build pipeline, your pipeline is automatically triggered for pull requests from forks of your repository. You can change this behavior, carefully considering how it affects security. To enable or disable this behavior:

1. Go to your Azure DevOps project. Select **Pipelines**, and then select **Builds**. Locate your build pipeline, and select **Edit**.
2. Select the **Triggers** tab. After enabling the **Pull request trigger**, enable or disable the **Build pull requests**

from forks of this repository check box.

By default with GitHub pipelines, secrets associated with your build pipeline are not made available to pull request builds of forks. These secrets are enabled by default with GitHub Enterprise pipelines. Secrets include:

- A security token with access to your GitHub repository.
- These items, if your build uses them:
 - [Service connection](#) credentials
 - Files from the [secure files library](#)
 - Build [variables](#) marked **secret**

To bypass this precaution on GitHub pipelines, enable the **Make secrets available to builds of forks** check box. Be aware of this setting's effect on security.

Important security considerations

A GitHub user can fork your repository, change it, and create a pull request to propose changes to your repository. This pull request could contain malicious code to run as part of your triggered build. For example, an ill-intentioned script or unit test change might leak secrets or compromise the agent machine that's performing the build. We recommend the following actions to address this risk:

- Do not enable the **Make secrets available to builds of forks** check box if your repository is public or untrusted users can submit pull requests that automatically trigger builds. Otherwise, secrets might leak during a build.
- Use a [Microsoft-hosted agent pool](#) to build pull requests from forks. Microsoft-hosted agent machines are immediately deleted after they complete a build, so there is no lasting impact if they're compromised.
- If you must use a [self-hosted agent](#), do not store any secrets or perform other builds and releases that use secrets on the same agent, unless your repository is private and you trust pull request creators. Otherwise, secrets might leak, and the repository contents or secrets of other builds and releases might be revealed.

Add a build badge

To add a build badge to the `README.md` file at the root of your repository, follow the steps in [Get the status badge](#).

Parallel jobs and time limits

If you use a public project with a public repository, then Azure Pipelines jobs are free. These free jobs have a maximum timeout of 360 minutes (6 hours) each.

If you use a private project or a private repository, then you can run up to 1,800 minutes (30 hours) of jobs for free every month. These free jobs have a maximum timeout of 30 minutes each. Purchasing jobs for private projects or private repositories removes any monthly time limit and allows jobs to have a maximum timeout of 360 minutes (6 hours) each.

To adjust the timeout of jobs, see [Timeouts](#).

Learn more about pricing based on [parallel jobs](#).

Q & A

Why isn't a GitHub repository displayed for me to choose in Azure Pipelines?

Depending on the authentication type and ownership of the repository, specific permissions are required. See [Choose a repository to build](#) above.

Build TFVC repositories

11/8/2018 • 3 minutes to read • [Edit Online](#)

[Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015 | Previous versions \(XAML builds\)](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

TFVC options

While editing a pipeline that uses a TFVC repo, you have the following options.

FEATURE	AZURE PIPELINES, TFS 2018, TFS 2017, TFS 2015.4	TFS 2015 RTM
Clean	Yes	Yes
Specify local path	Yes	No
Label sources	Yes	No

NOTE

Azure Pipelines, TFS 2017.2 and newer: Click **Advanced settings** to see some of the following options.

Repository name

Ignore this text box (**TFS 2017 RTM** or older).

Mappings (workspace)

Include with a type value of **Map** only the folders that your build pipeline requires. If a subfolder of a mapped folder contains files that the build pipeline does not require, map it with a type value of **Cloak**.

Make sure that you **Map** all folders that contain files that your build pipeline requires. For example, if you add another project, you might have to add another mapping to the workspace.

Cloak folders you don't need. By default the root folder of project is mapped in the workspace. This configuration results in the build agent downloading all the files in the version control folder of your project. If this folder contains lots of data, your build could waste build system resources and slow down your build pipeline by downloading large amounts of data that it does not require.

When you remove projects, look for mappings that you can remove from the workspace.

If this is a CI build, in most cases you should make sure that these mappings match the filter settings of your CI trigger on the [Triggers tab](#).

For more information on how to optimize a TFVC workspace, see [Optimize your workspace](#).

Clean the local repo on the agent

You can perform different forms of cleaning the working directory of your self-hosted agent before a build runs.

In general, for faster performance of your self-hosted agents, don't clean the repo. In this case, to get the best performance, make sure you're also building incrementally by disabling any **Clean** option of the task or tool you're using to build.

If you do need to clean the repo (for example to avoid problems caused by residual files from a previous build), your options are below.

NOTE

Cleaning is not relevant if you are using a [Microsoft-hosted agent](#) because you get a new agent every time in that case.

Azure Pipelines, TFS 2018, TFS 2017.2

If you want to clean the repo, then select **true**, and then select one of the following options:

- **Sources:** The build pipeline performs an undo of any changes and scorches the current workspace under `$(Build.SourcesDirectory)`.
- **Sources and output directory:** Same operation as **Sources** option above, plus: Deletes and recreates `$(Build.BinariesDirectory)`.
- **Sources directory:** Deletes and recreates `$(Build.SourcesDirectory)`.
- **All build directories:** Deletes and recreates `$(Agent.BuildDirectory)`.

TFS 2017 RTM, TFS 2015.4

If you select **True** then the build pipeline performs an undo of any changes and scorches the workspace.

If you want the Clean switch described above to work differently, then on the **Variables** tab, define the `Build.Clean` variable and set its value to:

- `all` if you want to delete `$(Agent.BuildDirectory)`, which is the entire working folder that contains the sources folder, binaries folder, artifacts folder, and so on.
- `source` if you want to delete `$(Build.SourcesDirectory)`.
- `binary` If you want to delete `$(Build.BinariesDirectory)`.

TFS 2015 RTM

Select **true** to delete the repository folder.

Label sources

You may want to label your source code files to enable your team to easily identify which version of each file is included in the completed build. You also have the option to specify whether the source code should be labeled for all builds or only for successful builds.

NOTE

You can only use this feature when the source repository in your build is a Git or TFVC repository from your project.

In the **Label format** you can use user-defined and predefined variables that have a scope of "All." For example:

```
$(Build.DefinitionName)_$(Build.DefinitionVersion)_$(Build.BuildId)_$(Build.BuildNumber)_$(My.Variable)
```

The first four variables are predefined. `My.Variable` can be defined by you on the [variables tab](#).

The build pipeline labels your sources with a [TFVC label](#).

Q & A

What is scorch?

Schorch is a TFVC power tool that ensures source control on the server and the local disk are identical. See [Microsoft Visual Studio Team Foundation Server 2015 Power Tools](#).

Pipeline options for Git repositories

11/28/2018 • 12 minutes to read • [Edit Online](#)

[Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015 | Previous versions \(XAML builds\)](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

While editing a pipeline that uses a Git repo (in an Azure DevOps or TFS project, GitHub, GitHub Enterprise, Bitbucket Cloud, or external Git repo), you have the following options.

FEATURE	AZURE PIPELINES	TFS 2018	TFS 2017.2	TFS 2017 RTM	TFS 2015.4	TFS 2015 RTM
Branch	Yes	Yes	Yes	Yes	Yes	Yes
Clean	Yes	Yes	Yes	Yes	Yes	Yes
Tag or label sources	Project; Designer only	Team project	Team project	Team project	Team project	No
Report build status	Yes	Yes	Yes	Yes	No	No
Checkout submodules	Yes	Yes	Yes	Yes	Yes	Yes
Checkout files from LFS	Yes	Yes	Yes	Linux and macOS agents	Linux and macOS agents	Linux and macOS agents
Don't sync sources	Yes	Yes	Yes	No	No	No
Shallow fetch	Yes	Yes	Yes	Linux and macOS agents	Linux and macOS agents	Linux and macOS agents

NOTE

Azure Pipelines, TFS 2017.2 and newer: Click **Advanced settings** in the **Get Sources** task to see some of the above options.

Branch

TFS 2017 RTM and TFS 2015: This field is called **Default branch**.

This is the branch that you want to be the default when you manually queue this build. If you set a scheduled trigger for the build, this is the branch from which your build will get the latest sources. The default branch has no bearing when the build is triggered through continuous integration (CI). Usually you'll set this to be the same as

the default branch of the repository (for example, "master").

Clean the local repo on the agent

You can perform different forms of cleaning the working directory of your self-hosted agent before a build runs.

In general, for faster performance of your self-hosted agents, don't clean the repo. In this case, to get the best performance, make sure you're also building incrementally by disabling any **Clean** option of the task or tool you're using to build.

If you do need to clean the repo (for example to avoid problems caused by residual files from a previous build), your options are below.

NOTE

Cleaning is not effective if you're using a [Microsoft-hosted agent](#) because you'll get a new agent every time.

Azure Pipelines, TFS 2018, TFS 2017.2, TFS 2017.3

Select one of the following options:

- **Sources:** The build pipeline performs an undo of any changes in `$(Build.SourcesDirectory)`. More specifically, the following Git commands are executed prior to fetching the source.

```
git clean -fdx  
git reset --hard HEAD
```

- **Sources and output directory:** Same operation as **Sources** option above, plus: Deletes and recreates `$(Build.BinariesDirectory)`. Note that the `$(Build.ArtifactStagingDirectory)` and `$(Common.TestResultsDirectory)` are always deleted and recreated prior to every build regardless of any of these settings.

- **Sources directory:** Deletes and recreates `$(Build.SourcesDirectory)`. This results in initializing a new, local Git repository for every build.
- **All build directories:** Deletes and recreates `$(Agent.BuildDirectory)`. This results in initializing a new, local Git repository for every build.

TFS 2017 RTM

If you select **True** then the build pipeline performs an undo of any changes. If errors occur, then it deletes the contents of `$(Build.SourcesDirectory)`.

If you want the Clean switch described above to work differently, then on the **Variables** tab, define the `Build.Clean` variable and set its value to:

- `all` if you want to delete `$(Agent.BuildDirectory)`, which is the entire working folder that contains the sources folder, binaries folder, artifacts folder, and so on.
- `source` if you want to delete `$(Build.SourcesDirectory)`.
- `binary` If you want to delete `$(Build.BinariesDirectory)`.

TFS 2015.4

If you select **True** then the build pipeline performs an undo of any changes. If errors occur, then it deletes the contents of `$(Build.SourcesDirectory)`.

If you want the Clean switch described above to work differently, then on the **Variables** tab, define the

`Build.Clean` variable and set its value to:

- `all` if you want to delete `$(Agent.BuildDirectory)`, which is the entire working folder that contains the sources folder, binaries folder, artifacts folder, and so on.
- `source` if you want to delete `$(Build.SourcesDirectory)`.
- `binary` If you want to delete `$(Build.BinariesDirectory)`.

TFS 2015 RTM

Select **true** to delete the repository folder.

If you want the Clean switch described above to work differently, then on the **Variables** tab, define the `Build.Clean` variable and set its value to:

- `all` if you want to delete `$(Agent.BuildDirectory)`, which is the entire working folder that contains the sources folder, binaries folder, artifacts folder, and so on.
- `source` if you want to delete `$(Build.SourcesDirectory)`.
- `binary` If you want to delete `$(Build.BinariesDirectory)`.

Label sources

You may want to label your source code files to enable your team to easily identify which version of each file is included in the completed build. You also have the option to specify whether the source code should be labeled for all builds or only for successful builds.

NOTE

You can only use this feature when the source repository in your build is a Git or TFVC repository from your project.

In the **Label format** you can use user-defined and predefined variables that have a scope of "All." For example:

```
$(Build.DefinitionName)_$(Build.DefinitionVersion)_$(Build.BuildId)_$(Build.BuildNumber)_$(My.Variable)
```

The first four variables are predefined. `My.Variable` can be defined by you on the [variables tab](#).

The build pipeline labels your sources with a [Git tag](#).

Some build variables might yield a value that is not a valid label. For example, variables such as `$(Build.RequestedFor)` and `$(Build.DefinitionName)` can contain white space. If the value contains white space, the tag is not created.

After the sources are tagged by your build pipeline, an artifact with the Git ref `refs/tags/{tag}` is automatically added to the completed build. This gives your team additional traceability and a more user-friendly way to navigate from the build to the code that was built.

Report build status (Azure Pipelines, TFS 2017 and newer)

You've got the option to give your team a view of the build status from your remote source repository.

If your sources are in an Azure Repos Git repository in your project, then this option displays a badge on the **Code** page to indicate whether the build is passing or failing. The build status is displayed in the following tabs:

- **Files:** Indicates the status of the latest build for the selected branch.
- **Commits:** Indicates the build status of the each commit (this requires the continuous integration (CI) trigger to

be enabled for your builds).

- **Branches:** Indicates the status of the latest build for each branch.

If you use multiple build pipelines for the same repository in your project, then you may choose to enable this option for one or more of the pipelines. In the case when this option is enabled on multiple pipelines, the badge on the **Code** page indicates the status of the latest build across all the pipelines. Your team members can click the build status badge to view the latest build status for each one of the build pipelines.

GitHub

If your sources are in GitHub, then this option publishes the status of your build to GitHub using GitHub [Checks](#) or [Status](#) APIs. If your build is triggered from a GitHub pull request, then you can view the status on the GitHub pull requests page. This also allows you to set status policies within GitHub and automate merges. If your build is triggered by continuous integration (CI), then you can view the build status on the commit or branch in GitHub.

Other types of Git remote repositories

If your source is in any other type of remote repository, then you cannot use Azure Pipelines or TFS to automatically publish the build status to that repository. However, you can use a [build badge](#) as a way to integrate and show build status within your version control experiences.

Checkout submodules

Select if you want to download files from [submodules](#). You can either choose to get the immediate submodules or all submodules nested to any depth of recursion.

The build pipeline will check out your Git submodules as long as they are:

- **Unauthenticated:** A public, unauthenticated repo with no credentials required to clone or fetch.

- **Authenticated:**

- Contained in the same project, GitHub organization, or Bitbucket account as the Git repo specified above.

- Added by using a URL relative to the main repository. For example, this one would be checked out:

```
git submodule add ../../submodule.git mymodule
```

This one would not be checked out:

```
git submodule add https://dev.azure.com/fabrikamfiber/_git/ConsoleApp mymodule
```

NOTE

If you're running **TFS 2017.1, TFS 2017 RTM, or TFS 2015**, then the submodules must be children (immediate submodules)** of the Git repo you've selected for this build pipeline. In effect, the build pipeline runs

```
git submodule update --init
```

(not `git submodule update -init --recursive`).

Authenticated submodules

NOTE

Make sure that you have registered your submodules using HTTPS and not using SSH.

The same credentials that are used by the agent to get the sources from the main repository are also used to get the sources for submodules.

If your main repository and submodules are in an Azure Repos Git repository in your Azure DevOps project, then you can select the account used to access the sources. On the **Options** tab, on the **Build job authorization scope** menu, select either:

- **Project collection** to use the Project Collection Build service account
- **Current project** to use the Project Build Service account.

Make sure that whichever account you use has access to both the main repository as well as the submodules.

If your main repository and submodules are in the same GitHub organization, then the token stored in the GitHub service connection is used to access the sources.

Alternative to using the **Checkout submodules** option

In some cases you can't use the **Checkout submodules** option. You might have a scenario where a different set of credentials are needed to access the submodules. This can happen, for example, if your main repository and submodule repositories aren't stored in the same Azure DevOps organization or Git service.

If you can't use the **Checkout submodules** option, then you can instead use a custom script step to fetch submodules. First, get a personal access token (PAT) and prefix it with "pat:". Next, Base64-encode this string to create a basic auth token. Finally, add this script to your pipeline:

```
git -c http.https://<url of submodule repository>.extraheader="AUTHORIZATION: basic <BASIC_AUTH_TOKEN>"  
submodule update --init --recursive
```

Be sure to replace "<BASIC_AUTH_TOKEN>" with your Base64-encoded token.

Use a secret variable in your project or build pipeline to store the basic auth token that you generated. Use that variable to populate the secret in the above Git command.

NOTE

Q: Why can't I use a Git credential manager on the agent? A: Storing the submodule credentials in a Git credential manager installed on your private build agent is usually not effective as the credential manager may prompt you to re-enter the credentials whenever the submodule is updated. This isn't desirable during automated builds when user interaction isn't possible.

Checkout files from LFS

Select if you want to download files from [large file storage \(LFS\)](#).

- **Azure Pipelines, TFS 2017.3 and newer:** Select the check box to enable this option.
- **TFS 2017 RTM and TFS 2015 (macOS and Linux only):** On the **Variables** tab, set `Agent.Source.Git.Lfs` to `true`.

If you're using TFS, or if you're using Azure Pipelines with a self-hosted agent, then you must install git-lfs on the agent for this option to work.

Don't sync sources (TFS 2017 and newer only)

Use this option if you want to skip fetching new commits. This option can be useful in cases when you want to:

- Git init, config, and fetch using your own custom options.
- Use a build pipeline to just run automation (for example some scripts) that do not depend on code in version control.

If you want to disable downloading sources:

- **Azure Pipelines, TFS 2017.2, and newer:** Click **Advanced settings**, and then select **Don't sync**

sources.

- **TFS 2017 RTM:** Define `Build.SyncSources` on the **Variables** and set its value to false.

NOTE

When you use this option, the agent also skips running Git commands that clean the repo.

Shallow fetch

Select if you want to limit how far back in history to download. Effectively this results in `git fetch --depth=n`. If your repository is large, this option might make your build pipeline more efficient. Your repository might be large if it has been in use for a long time and has sizeable history. It also might be large if you added and later deleted large files.

In these cases this option can help you conserve network and storage resources. It might also save time. The reason it doesn't always save time is because in some situations the server might need to spend time calculating the commits to download for the depth you specify.

NOTE

When the build is queued, the branch to build is resolved to a commit ID. Then, the agent fetches the branch and checks out the desired commit. There is a small window between when a branch is resolved to a commit ID and when the agent performs the checkout. If the branch updates rapidly and you set a very small value for shallow fetch, the commit may not exist when the agent attempts to check it out. If that happens, increase the shallow fetch depth setting.

Azure Pipelines, TFS 2018, TFS 2017.2

After you select the check box to enable this option, in the **Depth** box specify the number of commits.

Tip: The `Agent.Source.Git.ShallowFetchDepth` variable mentioned below also works and overrides the check box controls. This way you can modify the setting when you queue the build.

TFS 2017 RTM, TFS 2015 (macOS and Linux only)

On the **Variables** tab, define `Agent.Source.Git.ShallowFetchDepth` and set its value to the number of commits in history you want to download. Specify 0 to set no limit.

Prefer Git from path

The Windows agent comes with its own copy of Git. If you prefer to supply your own Git rather than use the included copy, set `System.PreferGitFromPath` to `true`. This setting is always true on non-Windows agents.

Trigger Options for External Git

When using an external Git repository, CI builds require that the repository is accessible from the internet. If the repository is behind a firewall or proxy, then only scheduled and manual builds will work.

Q & A

What protocols can the build agent use with Git?

The agent supports HTTPS.

The agent does not yet support SSH. See [User Voice: Allow build to use SSH authentication while checking out Git submodules](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Tasks for builds and releases

11/27/2018 • 6 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

A **task** is the building block for defining automation in a build pipeline, or in a stage of a release pipeline. A task is simply a packaged script or procedure that has been abstracted with a set of inputs.

When you add a task to your build or release pipeline, it may also add a set of **demands** to the pipeline. The demands define the prerequisites that must be installed on the [agent](#) for the task to run. When you run the build or deployment, an agent that meets these demands will be chosen.

When you queue a build or a deployment, all the tasks are run in sequence, one after the other, on an agent. To run the same set of tasks in parallel on multiple agents, or to run some tasks without using an agent, see [jobs](#).

Custom tasks

We provide some [built-in tasks](#) to enable fundamental build and deployment scenarios. We have also provided guidance for [creating your own custom task](#).

In addition, [Visual Studio Marketplace](#) offers a number of extensions; each of which, when installed to your subscription or collection, extends the task catalog with one or more tasks. Furthermore, you can write your own [custom extensions](#) to add tasks to Azure Pipelines or TFS.

In YAML pipelines, you refer to tasks by name. If a name matches both an in-box task and a custom task, the in-box task will take precedence. You can use a fully-qualified name for the custom task to avoid this risk:

```
steps:  
- task: myPublisherId.myExtensionId.myTaskName@1
```

Task versions

Tasks are versioned, and you must specify the major version of the task used in your pipeline. This can help to prevent issues when new versions of a task are released. Tasks are typically backwards compatible, but in some scenarios you may encounter unpredictable errors when a task is automatically updated.

When a new minor version is released (for example, 1.2 to 1.3), your build or release will automatically use the new version. However, if a new major version is released (for example 2.0), your build or release will continue to use the major version you specified until you edit the pipeline and manually change to the new major version. The build or release log will include an alert that a new major version is available.

- [YAML](#)
- [Designer](#)

In YAML, you specify the major version using `@` in the task name. For example, to pin to version 2 of the `PublishTestResults` task:

```
steps:  
- task: PublishTestResults@2
```

Task control options

Each task offers you some **Control Options**.

Enabled

Clear this check box to disable a task. This is useful when you want to temporarily take task out of the process for testing or for specific deployments.

TIP

You can also right-click the task to toggle this setting.

Timeout

The timeout for this task in minutes. The default is zero (infinite timeout). Setting a value other than zero overrides the setting for the parent task job.

Azure Pipelines options

Continue on error (partially successful)

Select this option if you want subsequent tasks in the same job to possibly run even if this task fails. The build or deployment will be no better than partially successful. Whether subsequent tasks run depends on the **Run this task** setting.

Run this task

Select the condition for running this task:

- Only when all previous tasks have succeeded
- Even if a previous task has failed, unless the build or release was canceled
- Even if a previous task has failed, even if the build was canceled
- Only when a previous task has failed
- [Custom conditions](#) which are composed of [expressions](#)

NOTE

If you're running tasks in cases when the build is canceled, then make sure you specify sufficient time for these tasks to run the [pipeline options](#).

TFS 2015 and newer options

Continue on error (partially successful)

Select this option if you want subsequent tasks in the same job to run even if this task fails. The build or deployment will be no better than partially successful.

Always run

Select this check box if you want the task to run even if the build or deployment is failing.

Build tool installers (Azure Pipelines)

Azure Pipelines preview feature

To use this capability you must be working on Azure Pipelines and enable the [Task tool installers preview feature](#).

Tool installers enable your build pipeline to install and control your dependencies. Specifically, you can:

- Install a tool or runtime on the fly (even on [Microsoft-hosted agents](#)) just in time for your CI build.
- Validate your app or library against multiple versions of a dependency such as Node.js.

For example, you can set up your build pipeline to run and validate your app for multiple versions of Node.js.

Example: Test and validate your app on multiple versions of Node.js

TIP

Want a visual walkthrough? See [our April 19 news release](#).

- [YAML](#)
- [Designer](#)

Create an `azure-pipelines.yml` file in your project's base directory with the following contents.

```
pool:  
  vmImage: 'Ubuntu 16.04'  
  
steps:  
  # Node install  
  - task: NodeTool@0  
    displayName: Node install  
    inputs:  
      versionSpec: '6.x' # The version we're installing  
    # Write the installed version to the command line  
    - script: which node
```

[Create a new build pipeline](#) and run it. Observe how the build is run. The [Node.js Tool Installer](#) downloads the Node.js version if it is not already on the agent. The [Command Line](#) script logs the location of the Node.js version on disk.

Tool installer tasks

For a list of our tool installer tasks, see [Tool installer tasks](#).

Related topics

- [Task jobs](#)
- [Task groups](#)
- [Built-in task catalog](#)

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Jobs

12/3/2018 • 10 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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

NOTE

You must install TFS 2018.2 to use jobs in build processes. In TFS 2018 RTM you can use jobs in release deployment processes.

You can organize your deployment process into jobs. Every deployment process has at least one job.

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

NOTE

You must install Update 2 to use jobs in TFS 2017, and they are available only in release deployment processes. Jobs in build pipelines are available in Azure Pipelines, TFS 2018.2, and newer versions.

In a build pipeline, the most common target is an agent. The other kind of target is the [Azure Pipelines server](#).

In a build pipeline, the most common target is an agent. The other kind of target is the server (your TFS instance).

In a deployment pipeline, the target can be either an agent, a [deployment group](#), or the server.

When the target is an agent, the tasks are run on the computer that hosts the agent.

- [YAML](#)
- [Designer](#)

The full syntax to specify an agent job is:

```

jobs:
- job: string
  timeoutInMinutes: number
  cancelTimeoutInMinutes: number
  strategy:
    maxParallel: number
    # note: `parallel` and `matrix` are mutually exclusive
    # you may specify one or the other; including both is an error
    parallel: number
    matrix: { string: { string: string } }
  pool:
    name: string
    demands: string | [ string ]

  container: string

steps:
- script: echo Hello world

```

The above syntax is necessary only if you want to define multiple jobs or change the properties for a job. You can skip the job syntax if you need only a single job with the standard options. For example, the following YAML file runs a single job to print `Hello world`.

```

steps:
- script: echo Hello world

```

If you want to specify just the pool, you can do that and skip the other properties. For example:

```

jobs:
- job: Run this job on a Linux agent
  pool: Default # the default self-hosted agent pool in your organization
  steps:
    ...

```

It's possible to re-use some or all of a pipeline through [templates](#).

YAML is not yet supported in TFS.

Demands

Use demands to specify what capabilities an agent must have to run jobs from your job.

- [YAML](#)
- [Designer](#)

```

pool:
  name: myPrivateAgents
  demands: agent.os -equals Windows_NT
steps:
- script: echo hello world

```

Or multiple demands:

```
pool:  
  name: myPrivateAgents  
  demands:  
    - agent.os -equals Darwin  
    - anotherCapability -equals somethingElse  
  steps:  
    - script: echo hello world
```

YAML is not yet supported in TFS.

Learn more about [agent capabilities](#).

Container image

If you are using YAML, you can specify a Docker container to use for your agent job.

- [YAML](#)
- [Designer](#)

For example, to run a script in a container:

```
resources:  
  containers:  
    - container: dev1  
      image: ubuntu:16.04  
jobs:  
  - job: job_container  
    pool:  
      name: default  
    container: dev1  
    steps:  
      - script: printenv
```

To run your job as four jobs on four different containers:

```

resources:
  containers:
    - container: dev1
      image: ubuntu:14.04
    - container: dev2
      image: private:ubuntu14
      endpoint: privatedockerhub
    - container: dev3
      image: ubuntu:16.04
      options: --cpu-count 4
    - container: dev4
      image: ubuntu:16.04
      options: --hostname container-test --ip 192.168.0.1
      localImage: true
  env:
    envVariable1: envValue1
    envVariable2: envValue2
jobs:
  - job: job_container
    container: ${variables['runtimeContainer']}
pool:
  name: default
  matrix:
    container_1:
      runtimeContainer: dev1
    container_2:
      runtimeContainer: dev2
    container_3:
      runtimeContainer: dev3
    container_4:
      runtimeContainer: dev4
steps:
  - script: printenv

```

YAML is not yet supported in TFS.

Timeouts

To avoid taking up resources when your job is hung or waiting too long, it's a good idea to set a limit on how long your job is allowed to run. Use the job timeout setting to specify the limit in minutes for running the job. Setting the value to **zero** means that the job can run:

- Forever on self-hosted agents
- For 360 minutes (6 hours) on Microsoft-hosted agents with a public project and public repository
- For 30 minutes on Microsoft-hosted agents with a private project or private repository
- [YAML](#)
- [Designer](#)

The `timeoutInMinutes` allows a limit to be set for the job execution time. When not specified, the default is 60 minutes. When `0` is specified, the maximum limit is used (described above).

The `cancelTimeoutInMinutes` allows a limit to be set for the job cancel time. When not specified, the default is 5 minutes.

```

jobs:
  - job: Test
    timeoutInMinutes: 10
    cancelTimeoutInMinutes: 2

```

YAML is not yet supported in TFS.

Jobs targeting Microsoft-hosted agents have [additional restrictions](#) on how long they may run.

You can also set the timeout for each task individually - see [task control options](#).

Multi-configuration

From a single job you can run multiple jobs and multiple agents in parallel. Some examples include:

- **Multi-configuration builds:** An agent job can be used in a build pipeline to build multiple configurations in parallel. For example, you could build a Visual C++ app for both `debug` and `release` configurations on both `x86` and `x64` platforms. To learn more, see [Visual Studio Build - multiple configurations for multiple platforms](#).
- **Multi-configuration deployments:** An agent job can be used in a stage of a release pipeline to run multiple deployment jobs in parallel, for example, to different geographic regions.
- **Multi-configuration testing:** An agent job can be used in a build pipeline or in a stage of a release pipeline to run a set of tests in parallel - once for each test configuration.
- [YAML](#)
- [Designer](#)

The `matrix` strategy enables a job to be dispatched multiple times, with different variable sets. The `maxParallel` tag restricts the amount of parallelism. The following job will be dispatched three times with the values of Location and Browser set as specified. However, only two jobs will run at the same time.

```
jobs:
- job: Test
  strategy:
    maxParallel: 2
    matrix:
      US_IE:
        Location: US
        Browser: IE
      US_Chrome:
        Location: US
        Browser: Chrome
      Europe_Chrome:
        Location: Europe
        Browser: Chrome
```

YAML is not yet supported in TFS.

Slicing

An agent job can be used to run a suite of tests in parallel. For example, you can run a large suite of 1000 tests on a single agent. Or, you can use two agents and run 500 tests on each one in parallel. To leverage slicing, the tasks in the job should be smart enough to understand the slice they belong to. The Visual Studio Test task is one such task that supports test slicing. If you have installed multiple agents, you can specify how the Visual Studio Test task will run in parallel on these agents.

- [YAML](#)
- [Designer](#)

The `parallel` strategy enables a job to be duplicated many times. The `maxParallel` tag restricts the amount

of parallelism. Variables `System.JobPositionInPhase` and `System.TotalJobsInPhase` are added to each job. The variables can then be used within your scripts to divide work among the jobs. See [Parallel and multiple execution using agent jobs](#).

The following job will be dispatched 5 times with the values of `System.JobPositionInPhase` and `System.TotalJobsInPhase` set appropriately. However, only two jobs will run at the same time.

```
jobs:
- job: Test
  strategy:
    parallel: 5
    maxParallel: 2
```

YAML is not yet supported in TFS.

Job variables

If you are using YAML, variables can be specified on the job. The variables can be passed to task inputs using the macro syntax `$(variableName)`, or accessed within a script using the stage variable.

- [YAML](#)
- [Designer](#)

Here's an example of defining variables in a job and using them within tasks.

```
variables:
  mySimpleVar: simple var value
  "my.dotted.var": dotted var value
  "my var with spaces": var with spaces value

steps:
- script: echo Input macro = $(mySimpleVar). Env var = %MYSIMPLEVAR%
  condition: eq(variables['agent.os'], 'Windows_NT')
- script: echo Input macro = $(mySimpleVar). Env var = $MYSIMPLEVAR
  condition: in(variables['agent.os'], 'Darwin', 'Linux')
- bash: echo Input macro = $(my.dotted.var). Env var = $MY_DOTTED_VAR
- powershell: Write-Host "Input macro = $(my var with spaces). Env var = $env:MY_VAR_WITH_SPACES"
```

YAML is not yet supported in TFS.

Artifact download

- [YAML](#)
- [Designer](#)

```

# test and upload my code as an artifact named WebSite
jobs:
- job: Build
  pool:
    vmImage: 'ubuntu-16.04'
  steps:
  - script: npm test
  - task: PublishBuildArtifacts@1
    inputs:
      pathToPublish: '$(System.DefaultWorkingDirectory)'
      artifactName: WebSite

# download the artifact and deploy it only if the build job succeeded
- job: Deploy
  pool:
    vmImage: 'ubuntu-16.04'
  steps:
  - checkout: none #skip checking out the default repository resource
  - task: DownloadBuildArtifacts@0
    displayName: 'Download Build Artifacts'
    inputs:
      artifactName: WebSite
      downloadPath: $(System.DefaultWorkingDirectory)

dependsOn: Build
condition: succeeded()

```

Access to OAuth token

You can allow tasks running in this job to access current Azure Pipelines or TFS OAuth security token. The token can be used to authenticate to the Azure Pipelines REST API.

- [YAML](#)
- [Designer](#)

The OAuth token is always available to YAML pipelines. It must be explicitly mapped into the task or step using `env`. Here's an example:

```

steps:
- powershell: |
  $url =
"$(System.TeamFoundationCollectionUri)$env:SYSTEM_TEAMPROJECTID/_apis/build/definitions/$($env:SYSTEM_DEFINITIONID)?api-version=4.1-preview"
  Write-Host "URL: $url"
  $pipeline = Invoke-RestMethod -Uri $url -Headers @{
    Authorization = "Bearer $env:TOKEN"
  }
  Write-Host "Pipeline = $($pipeline | ConvertTo-Json -Depth 100)"
env:
  TOKEN: $(system.accesstoken)

```

YAML is not yet supported in TFS.

Related topics

- [Multiple jobs](#)
- [Server jobs](#)
- [Deployment group jobs](#)

Container jobs

11/26/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

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.

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.

Requirements

The Azure Pipelines system requires a few things in Linux-based containers:

- Bash
- `which`
- glibc
- Can run Nodejs (which the agent provides)
- Does not define an `ENTRYPOINT`, or if it does, that `ENTRYPOINT` is a shell

Be sure your container has each of these tools available. Some of the extremely stripped-down containers available on Docker Hub, especially those based on Alpine Linux, don't satisfy these minimum requirements. Also, containers with a non-shell `ENTRYPOINT` don't work, since Azure Pipelines will `docker exec` a series of commands which expect to be run by a shell.

Azure Pipelines can also run [Windows Containers](#). [Windows Server version 1803](#) or higher is required.

The Hosted macOS pool does not support running containers.

- [YAML](#)
- [Designer](#)

Single job

A simple example:

```
resources:
  containers:
    - container: my_container
      image: ubuntu:16.04

  pool:
    vmImage: 'ubuntu-16.04'

  container: my_container

steps:
  - script: printenv
```

This tells the system to fetch the `ubuntu` image tagged `16.04` from [Docker Hub](#) and then start the container.

When the `printenv` command runs, it will happen inside the `ubuntu:16.04` container.

NOTE

You must specify "Hosted Ubuntu 1604" as the pool name in order to run containers. Other pools won't work. In the future, we intend to remove the need to specify a pool.

Multiple jobs

Containers are also useful for running the same steps in [multiple jobs](#). In the following example, the same steps run in multiple versions of Ubuntu Linux.

```
resources:
  containers:
    - container: u14
      image: ubuntu:14.04

    - container: u16
      image: ubuntu:16.04

    - container: u18
      image: ubuntu:18.04

  pool:
    vmImage: 'ubuntu-16.04'

  strategy:
    matrix:
      ubuntu14:
        containerResource: u14
      ubuntu16:
        containerResource: u16
      ubuntu18:
        containerResource: u18

    container: ${ variables['containerResource'] }

  steps:
    - script: printenv
```

Other settings

Endpoints

Containers can be hosted on registries other than Docker Hub. To host an image on [Azure Container Registry](#) or another private container registry, add a [service connection](#) to the private registry. Then you can reference it in a container spec:

```
resources:
  containers:
    - container: private_ubuntu1604
      image: myprivate/registry:ubuntu1604
      endpoint: private_dockerhub_connection

    - container: acr_win1803
      image: myprivate.azurecr.io/windowsservercore:1803
      endpoint: my_acr_connection
```

Options

If you need to control container startup, you can specify `options`.

```
resources:  
  containers:  
    - container: my_container  
      image: ubuntu:16.04  
      options: --hostname container-test --ip 192.168.0.1
```

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

Server jobs

Azure Pipelines | TFS 2018

Tasks in a server 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.

- [YAML](#)
- [Designer](#)

The full syntax to specify an agent job is:

```
jobs:  
- job: string  
  timeoutInMinutes: number  
  cancelTimeoutInMinutes: number  
  strategy:  
    maxParallel: number  
    matrix: { string: { string: string } }  
  
  pool: server
```

You can also use the simplified syntax:

```
jobs:  
- job: string  
  server: true
```

YAML is not yet supported in TFS.

Timeouts

Use the job timeout setting to specify the limit in minutes for running the job. A zero value for this option means that the timeout is effectively infinite and so, by default, jobs run until they complete or fail. You can also set the timeout for each task individually. See [task control options](#). Jobs targeting Microsoft-hosted agents have [additional restrictions](#) on how long they may run.

- [YAML](#)
- [Designer](#)

The `timeoutInMinutes` allows a limit to be set for the job execution time. When not specified, the default is 60 minutes. The `cancelTimeoutInMinutes` allows a limit to be set for the job cancel time. When not specified, the

default is 5 minutes.

```
pool:  
  timeoutInMinutes: number  
  cancelTimeoutInMinutes: number
```

YAML is not yet supported in TFS.

Multi-configuration

If you are using YAML, you can dispatch multiple server jobs from a single server job.

- [YAML](#)
- [Designer](#)

The `matrix` setting enables a job to be dispatched multiple times, with different variable sets. The `parallel` tag restricts the amount of parallelism. The following job will be dispatched three times with the values of Location and Browser set as specified. However, only two jobs will run in parallel at a time.

```
jobs:  
- job: Test  
  strategy:  
    maxParallel: 2  
    matrix:  
      US_IE:  
        Location: US  
        Browser: IE  
      US_Chrome:  
        Location: US  
        Browser: Chrome  
      Europe_Chrome:  
        Location: Europe  
        Browser: Chrome  
  
  pool: server
```

YAML is not yet supported in TFS.

Job variables

If you are using YAML, variables can be specified on the job. The variables can be passed to task inputs using the macro syntax `$(variableName)`, or accessed within a script using the stage variable.

- [YAML](#)
- [Designer](#)

Here is an example of defining variables in a job and using them within tasks.

```
variables:
  mySimpleVar: simple var value
  "my.dotted.var": dotted var value
  "my var with spaces": var with spaces value

steps:
- script: echo Input macro = $(mySimpleVar). Env var = %MYSIMPLEVAR%
  condition: eq(variables['agent.os'], 'Windows_NT')
- script: echo Input macro = $(mySimpleVar). Env var = $MYSIMPLEVAR
  condition: in(variables['agent.os'], 'Darwin', 'Linux')
- bash: echo Input macro = $(my.dotted.var). Env var = $MY_DOTTED_VAR
- powershell: Write-Host "Input macro = $(my var with spaces). Env var = $env:MY_VAR_WITH_SPACES"
```

YAML is not yet supported in TFS.

Related topics

- [Jobs](#)
- [Multiple jobs](#)
- [Deployment group jobs](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

Deployment group jobs

Azure Pipelines | TFS 2018

[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.

- [YAML](#)
- [Designer](#)

NOTE

Deployment group jobs are not yet supported in YAML.

YAML builds are not yet available on TFS.

Timeouts

Use the job timeout to specify the timeout in minutes for jobs in this job. A zero value for this option means that the timeout is effectively infinite and so, by default, jobs run until they complete or fail. You can also set the timeout for each task individually - see [task control options](#). Jobs targeting Microsoft-hosted agents have [additional restrictions](#) on how long they may run.

Related topics

- [Jobs](#)
- [Server jobs](#)
- [Multiple jobs](#)

Multiple jobs

11/15/2018 • 6 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

You can add multiple [jobs](#) to a pipeline. By using different jobs in a pipeline, you can:

- Partition your pipeline into sections targeting different agent pools
 - Partition your pipeline into sections targeting different sets of self-hosted agents using different demands
 - Partition your pipeline into sections that run on agents and those that run without an agent
 - Publish build artifacts in one job and consume those in subsequent jobs
 - Pause the deployment in the middle using a manual intervention task
 - Reduce build time by running multiple jobs in parallel
 - Reduce deployment time by selectively downloading different artifacts in different jobs of a deployment pipeline
 - [Conditionally execute](#) a set of tasks
-
- Partition your process into sections targeting different agent pools
 - Partition your process into sections targeting different sets of self-hosted agents using different demands
 - Partition your process into sections that run on agents and those that run without an agent
 - Publish build artifacts in one job and consume those in subsequent jobs
 - Pause the deployment in the middle using a manual intervention task
 - Reduce deployment time by selectively downloading different artifacts in different jobs of a deployment process
 - [Conditionally execute](#) a set of tasks
-
- Partition your process into sections targeting different agent pools
 - Partition your process into sections targeting different sets of self-hosted agents using different demands
 - Partition your process into sections that run on agents and those that run without an agent
 - Pause the deployment in the middle using a manual intervention task

NOTE

Jobs are not supported in build pipelines in TFS 2017.

NOTE

Each agent can run only one job at a time. To run multiple jobs in parallel you must configure multiple agents. You also need sufficient [parallel jobs](#).

- [YAML](#)
- [Designer](#)

The syntax for defining multiple jobs and their dependencies is:

```
jobs:  
- job: string  
  dependsOn: string  
  condition: string
```

YAML builds are not yet available on TFS.

Dependencies

- [YAML](#)
- [Designer](#)

Example jobs that build sequentially:

```
jobs:  
- job: Debug  
  steps:  
    - script: echo hello from the Debug build  
- job: Release  
  dependsOn: Debug  
  steps:  
    - script: echo hello from the Release build
```

Example jobs that build in parallel (no dependencies):

```
jobs:  
- job: Windows  
  pool:  
    vmImage: 'vs2017-win2016'  
  steps:  
    - script: echo hello from Windows  
- job: macOS  
  pool:  
    vmImage: 'macOS-10.13'  
  steps:  
    - script: echo hello from macOS  
- job: Linux  
  pool:  
    vmImage: 'ubuntu-16.04'  
  steps:  
    - script: echo hello from Linux
```

Example of fan out:

```
jobs:  
- job: InitialJob  
  steps:  
    - script: echo hello from initial job  
- job: SubsequentA  
  dependsOn: InitialJob  
  steps:  
    - script: echo hello from subsequent A  
- job: SubsequentB  
  dependsOn: InitialJob  
  steps:  
    - script: echo hello from subsequent B
```

Example of fan in:

```
jobs:
- job: InitialA
  steps:
  - script: echo hello from initial A
- job: InitialB
  steps:
  - script: echo hello from initial B
- job: Subsequent
  dependsOn:
  - InitialA
  - InitialB
  steps:
  - script: echo hello from subsequent
```

YAML is not yet supported in TFS.

Conditions

You can specify the conditions under which each job runs. By default, a job runs if it does not depend on any other job, or if all of the jobs that it depends on have completed and succeeded. You can customize this behavior by forcing a job to run even if a previous job fails or by specifying a custom condition.

- [YAML](#)
- [Designer](#)

YAML builds are not yet available on TFS.

Example to run a job based upon the status of running a previous job:

```
jobs:
- job: A
  steps:
  - script: exit 1

- job: B
  dependsOn: A
  condition: failed()
  steps:
  - script: echo this will run when A fails

- job: C
  dependsOn:
  - A
  - B
  condition: succeeded('B')
  steps:
  - script: echo this will run when B runs and succeeds
```

Example of using a [custom condition](#):

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

- job: B
  dependsOn: A
  condition: and(succeeded(), eq(variables['build.sourceBranch'], 'refs/heads/master'))
  steps:
    - script: echo this only runs for master
```

You can specify that a job run based on the value of an output variable set in a previous job. In this case, you can only use variables set in directly dependent jobs:

```
jobs:
- job: A
  steps:
    - script: "echo ##vso[task.setvariable variable=skipsubsequent;isOutput=true]false"
      name: printvar

- job: B
  condition: and(succeeded(), ne(dependencies.A.outputs['printvar.skipsubsequent'], 'true'))
  dependsOn: A
  steps:
    - script: echo hello from B
```

Related topics

- [Jobs](#)
- [Server jobs](#)
- [Deployment group jobs](#)

Specify conditions

10/15/2018 • 4 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017.3

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

- [YAML](#)
- [Designer](#)

On each step and job, you can specify the conditions under which the step or job will run.

- Only when all previous tasks have succeeded
- Even if a previous task has failed, unless the build or release was canceled
- Even if a previous task has failed, even if the build was canceled
- Only when a previous task has failed
- Custom conditions

```
jobs:  
- job: Foo  
  
  steps:  
    - script: echo Hello!  
      condition: always() # this step will always run, even if the pipeline is cancelled  
  
    - job: Bar  
      dependsOn: Foo  
      condition: failed() # this job will only run if Foo fails
```

YAML is not yet supported in TFS.

Enable a custom condition

If the built-in conditions don't meet your needs, then you can specify **custom conditions**.

In TFS 2017.3, custom task conditions are available in the user interface only for Build pipelines. You can use the Release [REST APIs](#) to establish custom conditions for Release pipelines.

Conditions are written as expressions. The agent evaluates the expression beginning with the innermost function and works its way out. The final result is a boolean value that determines if the task is run or not. See the [expressions](#) topic for a full guide to the syntax.

Do any of your conditions make it possible for the task to run even after the build is canceled by a user? If so, then specify a reasonable value for **Build job cancel timeout in minutes** in the [options](#) so that these kinds of tasks have enough time to complete after the user clicks **Cancel**.

Examples

Run for the master branch, if succeeding

```
and(succeeded(), eq(variables['Build.SourceBranch'], 'refs/heads/master'))
```

Run if the branch is not master, if succeeding

```
and(succeeded(), ne(variables['Build.SourceBranch'], 'refs/heads/master'))
```

Run for user topic branches, if succeeding

```
and(succeeded(), startsWith(variables['Build.SourceBranch'], 'refs/heads/users/'))
```

Run for continuous integration (CI) builds if succeeding

```
and(succeeded(), in(variables['Build.Reason'], 'IndividualCI', 'BatchedCI'))
```

Run if the build is run by a branch policy for a pull request, if failing

```
and(failed(), eq(variables['Build.Reason'], 'PullRequest'))
```

Run if the build is scheduled, even if failing, even if canceled

```
and(always(), eq(variables['Build.Reason'], 'Schedule'))
```

Release.Artifacts.{artifact-alias}.SourceBranch is equivalent to **Build.SourceBranch**.

Job status functions

In addition to the [general functions](#) available in expressions, you can use the following as shortcuts for common job status checks.

always

- Always evaluates to `True` (even when canceled). Note: A critical failure may still prevent a task from running. For example, if getting sources failed.

canceled

- Evaluates to `True` if the pipeline was canceled.

failed

- For a step, equivalent to `eq(variables['Agent.JobStatus'], 'Failed')`.
- For a job:
 - With no arguments, evaluates to `True` only if any previous job in the dependency graph failed.
 - With job names as arguments, evaluates to `True` only if any of those jobs failed.

succeeded

- For a step, equivalent to `in(variables['Agent.JobStatus'], 'Succeeded', 'SucceededWithIssues')`.
- For a job:
 - With no arguments, evaluates to `True` only if all previous jobs in the dependency graph succeeded or

partially succeeded.

- With job names as arguments, evaluates to `True` if all of those jobs succeeded or partially succeeded.

succeededOrFailed

- For a step, equivalent to `in(variables['Agent.JobStatus'], 'Succeeded', 'SucceededWithIssues', 'Failed')`
- For a job:
 - With no arguments, evaluates to `True` regardless of whether any jobs in the dependency graph succeeded or failed.
 - With job names as arguments, evaluates to `True` whether any of those jobs succeeded or failed.

This is like `always()`, except it will evaluate `False` when the pipeline is canceled.

Variables

Build or Release variables are available. For agent jobs, variables marked agent-scoped are available.

Some of the more useful predefined variables include:

- `Build.Reason` which you can use to check whether the build was the result of a [build trigger](#), a [Git PR affected by a branch policy](#), or a [TFVC gated check-in](#).
- `Build.SourceBranch`
- `Release.Artifacts.{artifact-alias}.SourceBranch`

Dependencies

For jobs which depend on other jobs, expressions may also use context about previous jobs in the dependency graph. The context is called `dependencies` and works much like `variables`.

Structurally, the `dependencies` object is a map of job names to `results` and `outputs`. Expressed as JSON, it would look like:

```
"dependencies": {  
    "<JOB_NAME>": {  
        "result": "Succeeded|SucceededWithIssues|Skipped|Failed|Canceled",  
        "outputs": { // only variables explicitly made outputs will appear here  
            "variable1": "value1",  
            "variable2": "value2"  
        }  
    },  
    "...": {  
        // another job  
    }  
}
```

For instance, in a YAML pipeline, you could use it like this:

```
jobs:
- job: A
  steps:
  - script: "echo ##vso[task.setvariable variable=skipsubsequent;isOutput=true]false"
    name: printvar

- job: B
  condition: and(succeeded(), ne(dependencies.A.outputs['printvar.skipsubsequent'], 'true'))
  dependsOn: A
  steps:
  - script: echo hello from B
```

Q&A

I've got a condition that runs even when build was cancelled. Does this affect a build that I cancelled in the queue?

No. If you cancel a build while it's in the queue, then the entire build is canceled, including tasks like this.

I've got a task condition that runs even when the deployment was canceled. How do I specify this?

This scenario is not yet supported for release pipelines.

Expressions

11/8/2018 • 5 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017.3

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called service endpoints in TFS 2018 and in older versions.

Expressions let you describe decisions the system should make, such as whether to run a step or the value of a variable. The most common use of expressions is in [conditions](#) to determine whether a job or step should run. Expressions are typically a nested set of functions evaluated from the innermost function out. Expressions always evaluate to strings, though the strings may be treated as booleans or other data types depending on where they're used.

Types

Although expressions evaluate to strings, they're coerced to other data types as needed. For instance, if an inner function evaluates to `"true"` and then is used as an input to `and()`, it will be coerced to Boolean `True`.

Boolean

`True` and `False`

Null

Null is a special type that's returned from a dictionary miss only, e.g. (`variables['noSuch']`).

Number

Starts with `'-`, `'.'`, or `'0'` through `'9'`.

String

Must be single-quoted. For example: `'this is a string'`.

To express a literal single-quote, escape it with a single quote. For example:

`'It''s OK if they're using contractions.'`.

Version

A version number with up to four segments. Must start with a number and contain two or three period (`.`) characters. For example: `1.2.3.4`.

Type Casting

Conversion chart

Detailed conversion rules are listed further below.

		TO				
		Boolean	Null	Number	String	Version

		TO					
From	Boolean	-	-	Yes	Yes	-	
	Null	Yes	-	Yes	Yes	-	
	Number	Yes	-	-	Yes	Partial	
	String	Yes	Partial	Partial	-	Partial	
	Version	Yes	-	-	Yes	-	

Boolean

To number:

- `False` → `0`
- `True` → `1`

To string:

- `False` → `'false'`
- `True` → `'true'`

Null

- To Boolean: `False`
- To number: `0`
- To string: `''` (the empty string)

Number

- To Boolean: `0` → `False`, any other number → `True`
- To version: Must be greater than zero and must contain a non-zero decimal. Must be less than `Int32.MaxValue` (decimal component also).
- To string: Converts the number to a string with no thousands separator and no decimal separator.

String

- To Boolean: `''` (the empty string) → `False`, any other string → `True`
- To null: `''` (the empty string) → `Null`, any other string not convertible
- To number: `''` (the empty string) → 0, otherwise, runs C#'s `Int32.TryParse` using `InvariantCulture` and the following rules: `AllowDecimalPoint` | `AllowLeadingSign` | `AllowLeadingWhite` | `AllowThousands` | `AllowTrailingWhite`. If `TryParse` fails, then it's not convertible.
- To version: runs C#'s `Version.TryParse`. Must contain Major and Minor component at minimum. If `TryParse` fails, then it's not convertible.

Version

- To Boolean: `True`
- To string: Major.Minor or Major.Minor.Build or Major.Minor.Build.Revision.

Variables

Depending on the execution context, different variables are available. For instance, in a condition on a pipeline, `Build` or `Release` variables are available.

You may access variables using one of two syntaxes:

- Index syntax: `variables['MyVar']`
- Property dereference syntax: `variables.MyVar`.

In order to use property dereference syntax, the property name must:

- Start with `a-z` or `_`
- Be followed by `a-z` `0-9` or `_`

Functions

Expressions may include functions. These functions are always available. Depending on context, other functions may be available as well.

and

- Evaluates `True` if all parameters are `True`
- Min parameters: 2. Max parameters: N
- Casts parameters to Boolean for evaluation
- Short-circuits after first `False`

contains

- Evaluates `True` if left parameter String contains right parameter
- Min parameters: 2. Max parameters: 2
- Casts parameters to String for evaluation
- Performs ordinal ignore-case comparison

endsWith

- Evaluates `True` if left parameter String ends with right parameter
- Min parameters: 2. Max parameters: 2
- Casts parameters to String for evaluation
- Performs ordinal ignore-case comparison

eq

- Evaluates `True` if parameters are equal
- Min parameters: 2. Max parameters: 2
- Converts right parameter to match type of left parameter. Returns `False` if conversion fails.
- Ordinal ignore-case comparison for Strings

ge

- Evaluates `True` if left parameter is greater than or equal to the right parameter
- Min parameters: 2. Max parameters: 2
- Converts right parameter to match type of left parameter. Errors if conversion fails.
- Ordinal ignore-case comparison for Strings

gt

- Evaluates `True` if left parameter is greater than the right parameter
- Min parameters: 2. Max parameters: 2
- Converts right parameter to match type of left parameter. Errors if conversion fails.
- Ordinal ignore-case comparison for Strings

in

- Evaluates `True` if left parameter is equal to any right parameter
- Min parameters: 1. Max parameters: N

- Converts right parameters to match type of left parameter. Equality comparison evaluates `False` if conversion fails.
- Ordinal ignore-case comparison for Strings
- Short-circuits after first match

le

- Evaluates `True` if left parameter is less than or equal to the right parameter
- Min parameters: 2. Max parameters: 2
- Converts right parameter to match type of left parameter. Errors if conversion fails.
- Ordinal ignore-case comparison for Strings

lt

- Evaluates `True` if left parameter is less than the right parameter
- Min parameters: 2. Max parameters: 2
- Converts right parameter to match type of left parameter. Errors if conversion fails.
- Ordinal ignore-case comparison for Strings

ne

- Evaluates `True` if parameters are not equal
- Min parameters: 2. Max parameters: 2
- Converts right parameter to match type of left parameter. Returns `True` if conversion fails.
- Ordinal ignore-case comparison for Strings

not

- Evaluates `True` if parameter is `False`
- Min parameters: 1. Max parameters: 1
- Converts value to Boolean for evaluation

notIn

- Evaluates `True` if left parameter is not equal to any right parameter
- Min parameters: 1. Max parameters: N
- Converts right parameters to match type of left parameter. Equality comparison evaluates `False` if conversion fails.
- Ordinal ignore-case comparison for Strings
- Short-circuits after first match

or

- Evaluates `True` if any parameter is `true`
- Min parameters: 2. Max parameters: N
- Casts parameters to Boolean for evaluation
- Short-circuits after first `True`

startsWith

- Evaluates `true` if left parameter string starts with right parameter
- Min parameters: 2. Max parameters: 2
- Casts parameters to String for evaluation
- Performs ordinal ignore-case comparison

xor

- Evaluates `True` if exactly one parameter is `True`

- Min parameters: 2. Max parameters: 2
- Casts parameters to Boolean for evaluation

Build variables

12/3/2018 • 10 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called service endpoints in TFS 2018 and in older versions.

Variables give you a convenient way to get key bits of data into various parts of the pipeline. As the name suggests, the contents of a variable may change from run to run or job to job of your pipeline. Some variables are predefined by the system, and you are free to add your own as well.

Working with variables

Variables add a layer of indirection to your pipeline. Almost any place where a pipeline requires a text string or a number, you can use a variable instead of hard-coding a value. The system will replace the variable with its current value during the pipeline's execution.

Variable names consist of letters, numbers, `.`, and `_` characters. How you reference a variable depends on context. The following table indicates how you can reference a variable called `Build.DefinitionName` in each context.

CONTEXT	SYNTAX	NOTES
Version control tag applied by the build	<code>\$(Build.DefinitionName)</code>	Learn about repository version control tagging .
Custom build number	<code>\$(Build.DefinitionName)</code>	Learn about build number format options .
Designer input fields	<code>\$(Build.DefinitionName)</code>	
YAML input fields	<code>\$(Build.DefinitionName)</code>	
Windows batch script	<code>%BUILD_DEFINITIONNAME%</code>	Name is upper-cased, <code>.</code> replaced with <code>_</code> , and automatically inserted into the process environment. For more information and examples, see: Batch script , PowerShell script , or Shell script .
PowerShell script	<code>\$env:BUILD_DEFINITIONNAME</code>	
Bash script	<code>\$BUILD_DEFINITIONNAME</code>	

System-defined variables

Some variables are automatically inserted by the system. As a pipeline author or end user, you cannot set the contents of such variables. See the comprehensive lists of [build variables](#) and [release variables](#) to learn which ones are available.

[System.AccessToken](#)

Secret variables such as System.AccessToken have special behavior. Secrets aren't available to scripts and tasks by default. You must explicitly allow secret variables on a pipeline-by-pipeline basis. This reduces the chances for a malicious script or task to steal the credentials they contain.

- [YAML](#)
- [Designer](#)

In YAML, you must explicitly map System.AccessToken into the pipeline using a variable. You can do this at the step or task level:

```
steps:  
  - bash: echo This is a script that could use $SYSTEM_ACESSTOKEN  
    env:  
      SYSTEM_ACESSTOKEN: $(System.AccessToken)  
  - powershell: Write-Host "This is a script that could use $env:SYSTEM_ACESSTOKEN"  
    env:  
      SYSTEM_ACESSTOKEN: $(System.AccessToken)
```

User-defined variables

Some build templates automatically create variables for you. For example, when you [create a new .NET app build](#), `BuildConfiguration` and `BuildPlatform` are automatically defined for you. You are free to define additional variables in your pipelines. Both of these are considered user-defined variables.

- [YAML](#)
- [Designer](#)

YAML builds can have variables defined at the pipeline or [job](#) level. They can also access variables defined when the build is queued.

```
# Set variables once  
variables:  
  configuration: debug  
  platform: x64  
  
steps:  
  
  # Build solution 1  
  - task: MSBuild@1  
    inputs:  
      solution: solution1.sln  
      configuration: $(configuration) # Use the variable  
      platform: $(platform)  
  
  # Build solution 2  
  - task: MSBuild@1  
    inputs:  
      solution: solution2.sln  
      configuration: $(configuration) # Use the variable  
      platform: $(platform)
```

Parameters to YAML statements

To use a variable in a YAML statement, wrap it in `$()`. For example:

```
pool:  
  vmImage: 'ubuntu-16.04'  
steps:  
- script: ls  
  workingDirectory: $(agent.homeDirectory)
```

Scripts

To use a variables in a script, use environment variable syntax. Replace `.` and space with `_`, capitalize the letters, and then use your platform's syntax for referencing environment variables.

```
variables:  
  MY_CUSTOM: MyValue  
  
jobs:  
- job: LinuxOrMacOS  
  pool:  
    vmImage: 'ubuntu-16.04'  
  steps:  
    - bash: echo $AGENT_HOMEDIRECTORY  
  
- job: Windows  
  pool:  
    vmImage: 'vs2017-win2016'  
  steps:  
    - script: echo %AGENT_HOMEDIRECTORY%  
    - powershell: Write-Host $env:AGENT_HOMEDIRECTORY
```

Counters

You can create a counter that is automatically incremented by one in each execution of your pipeline. You can optionally provide a seed value for the counter if you need to start at a specific number. The counter can be assigned to a variable and then referenced in task inputs or scripts as you would any other variable.

```
variables:  
  major: 2  
  minor: 1  
  # creates a counter called versioncounter and seeds it at 100 and then assigns the value to a variable named  
  patch.  
  patch: $[counter('versioncounter', 100)]  
  
  # use the patch variable as part of your pipeline naming scheme  
  name: $(Date:yyyyMMdd).$(patch)  
  pool:  
    vmImage: 'ubuntu-16.04'  
  
  steps:  
  
  # use the variables as part of your versioning for your nuget package  
  - script: |  
    dotnet pack /p:PackageVersion=$(major).$(minor).$(patch)
```

Set a job-scoped variable from a script

To set a variable from a script, you use a command syntax and print to stdout. This does not update the environment variables, but it does make the new variable available to downstream steps within the same job.

```

pool:
  vmImage: 'ubuntu-16.04'

steps:
  # Create a variable
  - script: |
    echo '##vso[task.setvariable variable=myVariable]abc123'

  # Print the variable
  - script: |
    echo my variable is $(myVariable)

```

Set an output (multi-job) variable

If you want to make a variable available to future jobs, you must mark it as an output variable using `isOutput=true`. Then you can map it into future jobs using `[$[]` syntax and including the step name which set the variable.

```

jobs:

# Set an output variable from job A
- job: A
  pool:
    vmImage: 'vs2017-win2016'
  steps:
    - powershell: echo "##vso[task.setvariable variable=myOutputVar;isOutput=true]this is the value"
      name: setvarStep
    - script: echo $(setvarStep.myOutputVar)
      name: echovar

# Map the variable into job B
- job: B
  dependsOn: A
  pool:
    vmImage: 'ubuntu-16.04'
  variables:
    myVarFromJobA: $[ dependencies.A.outputs['setvarStep.myOutputVar'] ] # map in the variable
  steps:
    - script: echo $(myVarFromJobA)
      name: echovar

```

If you're setting a variable from a [matrix](#) or [slice](#), then to reference the variable, you have to include the name of the job as well as the step when you access it from a downstream job.

```

jobs:

# Set an output variable from a job with a matrix
- job: A
  pool:
    vmImage: 'ubuntu-16.04'
  strategy:
    maxParallel: 2
  matrix:
    debugJob:
      configuration: debug
      platform: x64
    releaseJob:
      configuration: release
      platform: x64
  steps:
    - script: echo "##vso[task.setvariable variable=myOutputVar;isOutput=true]this is the $(configuration) value"
      name: setvarStep
    - script: echo $(setvarStep.myOutputVar)
      name: echovar

# Map the variable from the debug job
- job: B
  dependsOn: A
  pool:
    vmImage: 'ubuntu-16.04'
  variables:
    myVarFromJobADebug: $[ dependencies.A.outputs['debugJob.setvarStep.myOutputVar'] ]
  steps:
    - script: echo $(myVarFromJobADebug)
      name: echovar

```

```

jobs:

# Set an output variable from a job with slicing
- job: A
  pool:
    vmImage: 'ubuntu-16.04'
    parallel: 2 # Two slices
  steps:
    - script: echo "##vso[task.setvariable variable=myOutputVar;isOutput=true]this is the slice $(system.jobPositionInPhase) value"
      name: setvarStep
    - script: echo $(setvarStep.myOutputVar)
      name: echovar

# Map the variable from the job for the first slice
- job: B
  dependsOn: A
  pool:
    vmImage: 'ubuntu-16.04'
  variables:
    myVarFromJobsA1: $[ dependencies.A.outputs['job1.setvarStep.myOutputVar'] ]
  steps:
    - script: "echo $(myVarFromJobsA1)"
      name: echovar

```

YAML builds are not yet supported on TFS.

Secret variables

We recommend that you make the variable **Secret** if it contains a password, keys, or some other kind of data that you need to avoid exposing.

TFS 2017.2, TFS 2017.3

Tasks	Variables	Triggers	Options	Retention	History
Process variables					
	Name	Value	Settable at queue time		
	system.collectionId	189a025d-3690-49db-91e8-e9e320834f58			
	system.teamProject	FabrikamFibre			
	system.definitionId	31			
	system.debug	false	<input checked="" type="checkbox"/>		
	BuildConfiguration	release	<input checked="" type="checkbox"/>		
	BuildPlatform	any cpu	<input checked="" type="checkbox"/>		
	password	*****	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
+ Add					

TFS 2017 RTM

Definitions / OurBuild | Builds

Build	Options	Repository	Variables	Triggers	General	Retention	History
Save		Queue build...	Undo				
List of predefined variables							
Name	Value	Allow at Queue Time					
system.collectionId	d72ef65b-fbd0-4ce1-bb0d-eb6393afc3b1	<input type="checkbox"/>					
system.teamProject	Scripts	<input type="checkbox"/>					
system.definitionId	98	<input type="checkbox"/>					
system.debug	false	<input checked="" type="checkbox"/>					
password	*****	<input type="checkbox"/>	<input type="checkbox"/>				
+ Add variable							

- [YAML](#)
- [Designer](#)

Important: By default with GitHub repositories, secret variables associated with your build pipeline are not made available to pull request builds of forks. See [Validate contributions from forks](#).

Secret variables are encrypted at rest with a 2048-bit RSA key. They are automatically masked out of any log output from the pipeline. Unlike a normal variable, they are not automatically decrypted into environment variables for scripts. You can explicitly map them in, though:

```

steps:

# Create a secret variable
- powershell: |
    Write-Host '##vso[task.setvariable variable=mySecret;issecret=true]abc'

# Attempt to output the value in various ways
- powershell: |
    # Using an input-macro:
    Write-Host "This works: $(mySecret)"

    # Using the env var directly:
    Write-Host "This does not work: $env:MYSECRET"

    # Using the mapped env var:
    Write-Host "This works: $env:MY_MAPPED_ENV_VAR"
env:
  MY_MAPPED_ENV_VAR: $(mySecret)

```

The output from the above script would look like this:

```

This works: ***
This does not work:
This works: ***

```

YAML builds are not yet available on TFS.

Allow at queue time

You can choose which variables are allowed to be set at queue time and which are fixed by the pipeline author. Continuing the .NET example from above, `BuildConfiguration` can be settable at queue time for CI builds. This way, developers can choose whether to create Debug or Release builds depending on their needs. However, on your official builds, `BuildConfiguration` should not be settable at queue time so that you don't accidentally ship Debug binaries.

Define and modify your variables in a script

To define or modify a variable from a script, use the `task.setvariable` logging command. Note that the updated variable value is scoped to the job being executed, and does not flow across jobs or stages. Variable names are transformed to uppercase, and the characters "." and " " are replaced by "_".

For example, `Agent.WorkFolder` becomes `AGENT_WORKFOLDER`. On Windows, you access this as `%AGENT_WORKFOLDER` or `$env:AGENT_WORKFOLDER`. On Linux and macOS, you use `$AGENT_WORKFOLDER`.

TIP

You can run a script on a:

- Windows agent using either a [Batch script task](#) or [PowerShell script task](#).
- macOS or Linux agent using a [Shell script task](#).

Batch

PowerShell

Shell

Batch script



Set the `sauce` and `secret.Sauce` variables

```
@echo ##vso[task.setvariable variable=sauce]crushed tomatoes  
@echo ##vso[task.setvariable variable=secret.Sauce;issecret=true]crushed tomatoes with garlic
```



Read the variables

Arguments

```
"$(sauce)" "$(secret.Sauce)"
```

Script

```
@echo off  
set sauceArgument=%~1  
set secretSauceArgument=%~2  
@echo No problem reading %sauceArgument% or %SAUCE%  
@echo But I cannot read %SECRET_SAUCE%  
@echo But I can read %secretSauceArgument% (but the log is redacted so I do not spoil  
the secret)
```

Console output from reading the variables:

```
No problem reading crushed tomatoes or crushed tomatoes  
But I cannot read  
But I can read ***** (but the log is redacted so I do not spoil the secret)
```

Environment variables

You can also pass environment variables from the agent host into build tasks. For example, on the [Build tab](#) of a build pipeline, add this task:

TASK	ARGUMENTS
Utility: Command Line	Tool: echo Arguments: \$(PATH)

NOTE

If you have defined a pipeline variable of the same name as an environment variable (for example, `PATH`), your pipeline variable value overrides the agent host's environment variable.

Q & A

What are the predefined release variables?

[Default release variables](#)

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Resources

10/19/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

A resource is anything used by a pipeline that lives outside the pipeline itself. Examples include:

- [secure files](#)
- [variable groups](#)
- [service connections](#)
- [agent pools](#)
- [other repositories](#)
- [containers](#)

Some resources must be authorized before they can be used. This ensures that only users with sufficient permissions can access potentially sensitive resources such as service connections.

Resource authorization

When you save a pipeline, resource authorization checks for new and updated resources. If you lack permission to authorize one or more resources, then saving the pipeline will fail.

If you add a new resource to an existing YAML pipeline, Azure Pipelines will pick up the change but may not be able to authorize resources. Your builds may fail until you authorize resources using the troubleshooting steps below.

Troubleshooting authorization for a YAML pipeline

When you add a new service endpoint or other resource to a pipeline, it must be authorized before it will work. Authorization happens when you save the pipeline (not the YAML file, but the pipeline configuration in Azure Pipelines).

If builds fail with an error message about resource authorization, you need to re-save your pipeline. When you add a new resource in a branch other than the one you set up in Azure Pipelines, follow these steps:

1. Navigate to the pipeline in Azure Pipelines.
2. Switch the pipeline's default branch to the branch that includes the new service endpoint reference. For example, if you've made a change in a branch called `features/add-resource`, then switch the pipeline's default branch to `features/add-resource`.
3. Save the pipeline.
4. Revert back to the prior default branch and save the pipeline again.

Why does this fix it?

The act of saving the pipeline loads the file from the default branch and authorizes discovered resources. We're working on a better experience that won't require these steps.

Job and step templates

11/19/2018 • 6 minutes to read • [Edit Online](#)

Azure Pipelines

Use templates to define your logic once and then reuse it several times. Templates combine the content of multiple YAML files into a single pipeline. You can pass parameters into a template from your parent pipeline.

Step re-use

You can reuse one or more steps across several jobs. In addition to the steps from the template, each job can define additional steps.

```
# File: templates/npm-steps.yml
steps:
- script: npm install
- script: npm test
```

```
# File: azure-pipelines.yml

jobs:
- job: Linux
  pool:
    vmImage: 'ubuntu-16.04'
  steps:
    - template: templates/npm-steps.yml # Template reference

- job: macOS
  pool:
    vmImage: 'macOS-10.13'
  steps:
    - template: templates/npm-steps.yml # Template reference

- job: Windows
  pool:
    vmImage: 'vs2017-win2016'
  steps:
    - script: echo This script runs before the template's steps, only on Windows.
    - template: templates/npm-steps.yml # Template reference
    - script: echo This step runs after the template's steps.
```

Job reuse

Much like steps, jobs can be reused.

```
# File: templates/jobs.yml
jobs:
- job: Build
  steps:
    - script: npm install

- job: Test
  steps:
    - script: npm test
```

```
# File: azure-pipelines.yml

jobs:
- template: templates/jobs.yml # Template reference
```

Passing parameters

You can pass parameters to both step and job templates. The `parameters` section defines what parameters are available in the template and their default values. Templates are expanded just before the pipeline runs so that values surrounded by `${{ }}` are replaced by the parameters it receives from the enclosing pipeline.

```
# File: templates/npm-with-params.yml

parameters:
  name: '' # defaults for any parameters that aren't specified
  vmImage: ''

jobs:
- job: ${{ parameters.name }}
  pool:
    vmImage: ${{ parameters.vmImage }}
  steps:
    - script: npm install
    - script: npm test
```

When you consume the template in your pipeline, specify values for the template parameters.

```
# File: azure-pipelines.yml

jobs:
- template: templates/npm-with-params.yml # Template reference
  parameters:
    name: Linux
    vmImage: 'ubuntu-16.04'

- template: templates/npm-with-params.yml # Template reference
  parameters:
    name: macOS
    vmImage: 'macOS-10.13'

- template: templates/npm-with-params.yml # Template reference
  parameters:
    name: Windows
    vmImage: 'vs2017-win2016'
```

The above example shows only how to use parameters with a job template. But you can also use parameters with step templates.

Using other repositories

You can keep your templates in other repositories. For example, suppose you have a core pipeline that you want all of your app pipelines to use. You can put the template in a core repo and then refer to it from each of your app repos:

```

# Repo: Contoso/BuildTemplates
# File: common.yml
parameters:
  vmImage: 'ubuntu 16.04'

jobs:
- job: Build
  pool:
    vmImage: ${{ parameters.vmImage }}
  steps:
- script: npm install
- script: npm test

```

Now you can reuse this template in multiple pipelines. Use the `resources` specification to provide the location of the core repo. When you refer to the core repo, use `@` and the name you gave it in `resources`.

```

# Repo: Contoso/LinuxProduct
# File: azure-pipelines.yml
resources:
  repositories:
    - repository: templates
      type: github
      name: Contoso/BuildTemplates

jobs:
- template: common.yml@templates # Template reference

```

```

# Repo: Contoso/WindowsProduct
# File: azure-pipelines.yml
resources:
  repositories:
    - repository: templates
      type: github
      name: Contoso/BuildTemplates

jobs:
- template: common.yml@templates # Template reference
  parameters:
    vmImage: 'vs2017-win2016'

```

Repositories are resolved only once, when the pipeline starts up. After that, the same resource is used for the duration of the pipeline. Only the template files are used. Once the templates are fully expanded, the final pipeline runs as if it were defined entirely in the source repo. This means that you can't use scripts from the template repo in your pipeline.

Template expressions

Use template [expressions](#) to specify how values are dynamically resolved during pipeline initialization. Wrap your template expression inside this syntax: `${{ }}.`

Template expressions can expand template parameters, and also variables. You can use parameters to influence how a template is expanded. The `parameters` object works like the [variables object](#) in an expression.

For example you define a template:

```
# File: steps/msbuild.yml

parameters:
  solution: '**/*.sln'

steps:
- task: msbuild@1
  inputs:
    solution: ${{ parameters['solution'] }} # index syntax
- task: vstest@2
  inputs:
    solution: ${{ parameters.solution }} # property dereference syntax
```

Then you reference the template and pass it the optional `solution` parameter:

```
# File: azure-pipelines.yml

steps:
- template: steps/msbuild.yml
  parameters:
    solution: my.sln
```

Required parameters

You can add a validation step at the beginning of your template to check for the parameters you require.

Here's an example that checks for the `solution` parameter using Bash (which enables it to work on any platform):

```
# File: steps/msbuild.yml

parameters:
  solution: ''

steps:
- bash: |
  if [ -z "$SOLUTION" ]; then
    echo ##vso[task.complete result=Failed;]Missing template parameter \"solution\"
  fi
env:
  SOLUTION: ${{ parameters.solution }}
displayName: Check for required parameters
- task: msbuild@1
  inputs:
    solution: ${{ parameters.solution }}
- task: vstest@2
  inputs:
    solution: ${{ parameters.solution }}
```

To prove that the template fails if it's missing the required parameter:

```
# File: azure-pipelines.yml

# This will fail since it doesn't set the "solution" parameter to anything,
# so the template will use its default of an empty string
steps:
- template: steps/msbuild.yml
```

Template expression functions

You can use [general functions](#) in your templates. You can also use a few template expression functions.

format

- Simple string token replacement
- Min parameters: 2. Max parameters: N
- Example: ``${{ format('{0} Build', parameters.os) }}`` → 'Windows Build'

coalesce

- Evaluates to the first non-empty, non-null string argument
- Min parameters: 2. Max parameters: N
- Example:

```
parameters:  
  restoreProjects: ''  
  buildProjects: ''  
  
steps:  
- script: echo ${{ coalesce(parameters.foo, parameters.bar, 'Nothing to see') }}
```

Insertion

You can use template expressions to alter the structure of a YAML pipeline. For instance, to insert into a sequence:

```
# File: jobs/build.yml  
  
parameters:  
  preBuild: []  
  preTest: []  
  preSign: []  
  
jobs:  
- job: Build  
  pool:  
    vmImage: 'vs2017-win2016'  
  steps:  
    - script: cred-scan  
    - ${{ parameters.preBuild }}  
    - task: msbuild@1  
    - ${{ parameters.preTest }}  
    - task: vstest@2  
    - ${{ parameters.preSign }}  
    - script: sign
```

```
# File: .vsts.ci.yml  
  
jobs:  
- template: jobs/build.yml  
  parameters:  
    preBuild:  
      - script: echo hello from pre-build  
    preTest:  
      - script: echo hello from pre-test
```

When an array is inserted into an array, the nested array is flattened.

To insert into a mapping, use the special property ``${{ insert }}``.

```

# Default values
parameters:
  variables: {}

jobs:
- job: build
  variables:
    configuration: debug
    arch: x86
    ${{ insert }}: ${{ parameters.variables }}
  steps:
- task: msbuild@1
- task: vstest@2

```

```

jobs:
- template: jobs/build.yml
parameters:
  variables:
    TEST_SUITE: L0,L1

```

Conditional insertion

If you want to conditionally insert into a sequence or a mapping, then use insertions and expression evaluation.

For example, to insert into a sequence:

```

# File: steps/build.yml

parameters:
  toolset: msbuild

steps:
# msbuild
- ${{ if eq(parameters.toolset, 'msbuild') }}:
  - task: msbuild@1
  - task: vstest@2

# dotnet
- ${{ if eq(parameters.toolset, 'dotnet') }}:
  - task: dotnet@1
    inputs:
      command: build
  - task: dotnet@1
    inputs:
      command: test

```

```

# File: azure-pipelines.yml

steps:
- template: steps/build.yml
parameters:
  toolset: dotnet

```

For example, to insert into a mapping:

```

# File: steps/build.yml

parameters:
  debug: false

steps:
- script: tool
  env:
    ${${ if eq(parameters.debug, 'true') }}:
      TOOL_DEBUG: true
      TOOL_DEBUG_DIR: _dbg

```

```

steps:
- template: steps/build.yml
  parameters:
    debug: true

```

Iterative insertion

The `each` directive allows iterative insertion based on a sequence or mapping.

For example, to wrap all steps within each job

```

parameters:
  jobs: []

jobs:
- ${${ each job in parameters.jobs }}: # Each job
  - ${${ each pair in job }}:           # Insert all properties other than "steps"
    ${${ if ne(pair.key, 'steps') }}:
      ${${ pair.key }}: ${${ pair.value }}
  steps:                      # Wrap the steps
    - task: SetupMyBuildTools@1      # Pre steps
    - ${${ job.steps }}             # Users steps
    - task: PublishMyTelemetry@1    # Post steps
      condition: always()

```

For example, to wrap all jobs with an additional dependency

```

parameters:
  jobs: []

jobs:
- job: CredScan                  # Cred scan first
  pool: MyCredScanPool
  steps:
    - task: MyCredScanTask@1
- ${${ each job in parameters.jobs }}: # Then each job
  - ${${ each pair in job }}:           # Insert all properties other than "dependsOn"
    ${${ if ne(pair.key, 'dependsOn') }}:
      ${${ pair.key }}: ${${ pair.value }}
  dependsOn:                         # Inject dependency
    - CredScan
    - ${${if job.dependsOn}}:
      - ${${ job.dependsOn }}

```

Escaping

If you need to escape a value that literally contains ``${{ }}`, then wrap the value in an expression string. For example

```
 ${{{ 'my${{value' }}}} or ${{{ 'my${{value with a '' single quote too' }}}}
```

Run cross-platform scripts

11/29/2018 • 4 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018

With Azure Pipelines and Team Foundation Server (TFS), you can run your builds on macOS, Linux, and Windows. If you develop on cross-platform technologies such as Node.js and Python, these capabilities bring benefits, and also some challenges. For example, most pipelines include one or more scripts that you want to run during the build process. But scripts often don't run the same way on different platforms. Below are some tips on how to handle this kind of challenge.

Run cross-platform tools with a script step

Some scripts just pass arguments to a cross-platform tool. For instance, calling `npm` with a set of arguments can be easily accomplished with a `script` step. `script` runs in each platform's native script interpreter: Bash on macOS and Linux, CMD on Windows.

- [YAML](#)
- [Designer](#)

```
steps:  
- script: |  
  npm install  
  npm test
```

Handle environment variables

Environment variables throw the first wrinkle into writing cross-platform scripts. Command line, PowerShell, and Bash each have different ways of reading environment variables. If you need to access an operating system-provided value like PATH, you'll need different techniques per platform.

However, Azure Pipelines offers a cross-platform way to refer to variables that it knows about. By surrounding a variable name in `$()`, it will be expanded before the platform's shell ever sees it. For instance, if you want to echo out the ID of the pipeline, the following script is cross-platform friendly:

- [YAML](#)
- [Designer](#)

```
steps:  
- script: echo This is pipeline $(System.DefinitionId)
```

This also works for variables you specify in the pipeline.

```
variables:  
  Example: 'myValue'  
  
steps:  
- script: echo The value passed in is $(Example)
```

Consider Bash

If you have more complex scripting needs than the examples shown above, then consider writing them in Bash. Most macOS and Linux agents have Bash as an available shell, and Windows agents include Git Bash.

For Azure Pipelines, the Microsoft-hosted agents always have Bash available.

For example, if you need to make a decision based on whether this is a pull request build:

- [YAML](#)
- [Designer](#)

```
trigger:  
  batch: true  
  branches:  
    include:  
    - master  
steps:  
- bash: |  
  echo "Hello world from $AGENT_NAME running on $AGENT_OS"  
  case $BUILD_REASON in  
    "Manual") echo "$BUILD_REQUESTEDFOR manually queued the build.";;  
    "IndividualCI") echo "This is a CI build for $BUILD_REQUESTEDFOR.";;  
    "BatchedCI") echo "This is a batched CI build for $BUILD_REQUESTEDFOR.";;  
    *) $BUILD_REASON;;  
  esac  
displayName: Hello world
```

Switch based on platform

In general we recommend that you avoid platform-specific scripts to avoid problems such as duplication of your pipeline logic. Duplication causes extra work and extra risk of bugs. However, if there's no way to avoid platform-specific scripting, then you can use a `condition` to detect what platform you're on.

For example, suppose that for some reason you need the IP address of the build agent. On Windows, `ipconfig` gets that information. On macOS, it's `ifconfig`. And on Ubuntu Linux, it's `ip addr`.

Set up the below pipeline, then try running it against agents on different platforms.

- [YAML](#)
- [Designer](#)

```
steps:
# Linux
- bash: |
  export IPADDR=$(ip addr | grep 'state UP' -A2 | tail -n1 | awk '{print $2}' | cut -f1 -d'/')
  echo "##vso[task.setvariable variable=IP_ADDR]$IPADDR"
  condition: eq( variables['Agent.OS'], 'Linux' )
  displayName: Get IP on Linux

# macOS
- bash: |
  export IPADDR=$(ifconfig | grep 'en0' -A3 | tail -n1 | awk '{print $2}')
  echo "##vso[task.setvariable variable=IP_ADDR]$IPADDR"
  condition: eq( variables['Agent.OS'], 'Darwin' )
  displayName: Get IP on macOS

# Windows
- powershell: |
  Set-Variable -Name IPADDR -Value ((Get-NetIPAddress | ?{ $_.AddressFamily -eq "IPv4" -and !($_.IPAddress -match "169") -and !($_.IPAddress -match "127") } | Select-Object -First 1).IPAddress)
  Write-Host "##vso[task.setvariable variable=IP_ADDR]$IPADDR"
  condition: eq( variables['Agent.OS'], 'Windows_NT' )
  displayName: Get IP on Windows

# now we use the value, no matter where we got it
- script: |
  echo The IP address is $(IP_ADDR)
```

Use a PowerShell script to customize your build pipeline

10/9/2018 • 3 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#) | [Previous versions \(XAML builds\)](#)

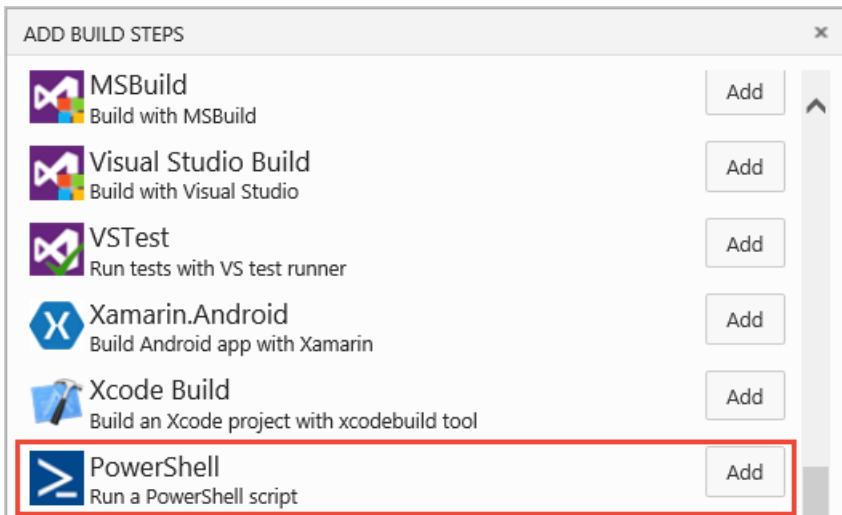
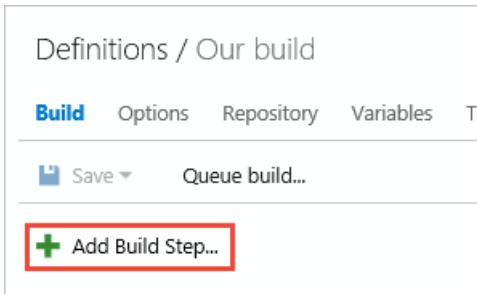
NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

When you are ready to move beyond the basics of compiling and testing your code, use a PowerShell script to add your team's business logic to your build pipeline.

You can run a PowerShell Script on a [Windows build agent](#).

1. Push your script into your repo.
2. Add a PowerShell build task.



3. Drag the build task where you want it to run.
4. Specify the name of the script.

Example: Version your assemblies

For example, to version to your assemblies, copy and upload this script to your project:

```

##-----  

## <copyright file="ApplyVersionToAssemblies.ps1">(c) Microsoft Corporation. This source is subject to the  

Microsoft Permissive License. See  

http://www.microsoft.com/resources/sharedsource/licensingbasics/sharedsourcelicenses.mspx. All other rights  

reserved.</copyright>  

##-----  

# Look for a 0.0.0.0 pattern in the build number.  

# If found use it to version the assemblies.  

#  

# For example, if the 'Build number format' build pipeline parameter  

# $(BuildDefinitionName)_$(Year:yyyy).$(Month).$(DayOfMonth)$(Rev:.r)  

# then your build numbers come out like this:  

# "Build HelloWorld_2013.07.19.1"  

# This script would then apply version 2013.07.19.1 to your assemblies.  

# Enable -Verbose option  

[CmdletBinding()]  

# Regular expression pattern to find the version in the build number  

# and then apply it to the assemblies  

$VersionRegex = "\d+\.\d+\.\d+\.\d+"  

# If this script is not running on a build server, remind user to  

# set environment variables so that this script can be debugged  

if(-not ($Env:BUILD_SOURCESDIRECTORY -and $Env:BUILDDATE))  

{  

    Write-Error "You must set the following environment variables"  

    Write-Error "to test this script interactively."  

    Write-Host '$Env:BUILD_SOURCESDIRECTORY - For example, enter something like:'  

    Write-Host '$Env:BUILD_SOURCESDIRECTORY = "C:\code\FabrikamTFVC\HelloWorld"'  

    Write-Host '$Env:BUILDDATE - For example, enter something like:'  

    Write-Host '$Env:BUILDDATE = "Build HelloWorld_0000.00.00.0"'  

    exit 1
}  

# Make sure path to source code directory is available  

if (-not $Env:BUILD_SOURCESDIRECTORY)  

{  

    Write-Error ("BUILD_SOURCESDIRECTORY environment variable is missing.")  

    exit 1
}  

elseif (-not (Test-Path $Env:BUILD_SOURCESDIRECTORY))  

{  

    Write-Error "BUILD_SOURCESDIRECTORY does not exist: $Env:BUILD_SOURCESDIRECTORY"  

    exit 1
}  

Write-Verbose "BUILD_SOURCESDIRECTORY: $Env:BUILD_SOURCESDIRECTORY"  

# Make sure there is a build number  

if (-not $Env:BUILDDATE)  

{  

    Write-Error ("BUILD_BUILDDATE environment variable is missing.")  

    exit 1
}  

Write-Verbose "BUILD_BUILDDATE: $Env:BUILDDATE"  

# Get and validate the version data  

$VersionData = [regex]::matches($Env:BUILDDATE,$VersionRegex)  

switch($VersionData.Count)  

{  

    0
    {
        Write-Error "Could not find version number data in BUILD_BUILDDATE."
        exit 1
    }
    1 {}
    default
    {
        Write-Warning "Found more than instance of version data in BUILD_BUILDDATE."
    }
}

```

```

        Write-Warning "Will assume first instance is version."
    }
}

$NewVersion = $VersionData[0]
Write-Verbose "Version: $NewVersion"

# Apply the version to the assembly property files
$files = gci $Env:BUILD_SOURCESDIRECTORY -recurse -include "*Properties*","My Project" |
?{ $_.PSIsContainer } |
foreach { gci -Path $_.FullName -Recurse -include AssemblyInfo.* }

if($files)
{
    Write-Verbose "Will apply $NewVersion to $($files.count) files."

    foreach ($file in $files) {
        $filecontent = Get-Content($file)
        attrib $file -r
        $filecontent -replace $VersionRegex, $NewVersion | Out-File $file
        Write-Verbose "$file.FullName - version applied"
    }
}
else
{
    Write-Warning "Found no files."
}

```

Add the build task to your build pipeline.

The screenshot shows the 'Definitions / Our build' screen in VSTS. The 'Build' tab is selected. A 'PowerShell' step is added, titled 'Powershell: Scripts/ApplyVersionToAssemblies.ps1'. The 'Enabled' checkbox is checked. The 'Script filename' is set to 'Scripts/ApplyVersionToAssemblies.ps1'. The 'Arguments' field is empty. Below the step, there are icons for 'Visual Studio Build' and 'VSTest'.

Specify your build number with something like this:

The screenshot shows the 'Definitions / Our build' screen in VSTS, focusing on the 'General' tab. The 'Build number format' field contains the placeholder '\$(BuildDefinitionName)_\$(Year:yyyy).\$(Month).\$(DayOf'. This field is highlighted with a red border.

\$(BuildDefinitionName)_\$(Year:yyyy).\$(Month).\$(DayOfMonth)\$(Rev:.r)

Use the OAuth token to access the REST API

To enable your script to use the build pipeline OAuth token, go to the **Options** tab of the build pipeline and select **Allow Scripts to Access OAuth Token**.

After you've done that, your script can use the `SYSTEM_ACESSTOKEN` environment variable to access the [Azure Pipelines REST API](#). For example:

```
$url = "($env:SYSTEM_TEAMFOUNDATIONCOLLECTIONURI)$env:SYSTEM_TEAMPROJECTID/_apis/build-release/definitions/($env:SYSTEM_DEFINITIONID)?api-version=2.0"
Write-Host "URL: $url"
$pipeline = Invoke-RestMethod -Uri $url -Headers @{
    Authorization = "Bearer $env:SYSTEM_ACESSTOKEN"
}
Write-Host "Pipeline = $($pipeline | ConvertTo-Json -Depth 1000)"
```

Q&A

What variables are available for me to use in my scripts?

[Use variables](#)

How do I set a variable so that it can be read by subsequent scripts and tasks?

[Define and modify your build variables in a script](#)

[Define and modify your release variables in a script](#)

Which branch of the script does the build run?

The build runs the script same branch of the code you are building.

What kinds of parameters can I use?

You can use named parameters. Other kinds of parameters, such as switch parameters, are not yet supported and will cause errors.

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Run Git commands in a script

10/29/2018 • 3 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

For some workflows you need your build pipeline to run Git commands. For example, after a CI build on a feature branch is done, the team might want to merge the branch to master.

Git is available on [Microsoft-hosted agents](#) and on [on-premises agents](#).

Enable scripts to run Git commands

Grant version control permissions to the build service

Go to the [Version Control](#) control panel tab ▾

- Azure Repos:

https://dev.azure.com/{your-organization}/DefaultCollection/{your-project}/_admin/_versioncontrol

- On-premises: https:///{your-server}:8080/tfs/DefaultCollection/{your-project}/_admin/_versioncontrol



If you see this page, select the repo, and then click the link:

A screenshot of the 'Overview' tab in the 'fabrikam' project control panel. The top navigation bar shows 'Control panel > fabrikam'. Below it is a horizontal tab bar with 'Overview' (highlighted with a red box), 'Settings', 'Security', 'Process', 'Build', 'Agent pools', and 'Extensions*'. The main area shows the 'Account / fabrikam' section with a 'Description' field containing 'New team project'. To the right is a 'Projects' section with a table. The first row shows 'Project name: Fabrikam' and 'Process: Agile'. The second row shows 'Project name: OurProject' (highlighted with a red box) and 'Process: Agile'.

Project name	Process
Fabrikam	Agile
OurProject	Agile

A screenshot of the 'Version Control' tab in the 'OurProject' repository control panel. The top navigation bar shows 'Control panel > fabrikam > OurProject'. Below it is a horizontal tab bar with 'Overview', 'Work', 'Security', 'Alerts', and 'Version Control' (highlighted with a red box).

On the **Version Control** tab, select the repository in which you want to run Git commands, and then select **Project Collection Build Service**.

Repositories

New repository

Git repositories

- OurProject
- Portal-Client
- Portal-Server
- Store-Client
- Store-Server

Security for all Git repositories

Add... Inheritance ▾

Search P

VSTS Groups

- Project Collection Administrators
- Project Collection Build Service Accounts
- Project Collection Service Accounts
- Build Administrators
- Contributors
- Project Administrators
- Readers
- Project Collection Build Service (Application-...)

Users

Grant permissions needed for the Git commands you want to run. Typically you'll want to grant:

- **Branch creation:** Allow
- **Contribute:** Allow
- **Read:** Inherited allow
- **Tag creation:** Inherited allow

When you're done granting the permissions, make sure to click **Save changes**.

Enable your pipeline to run command-line Git

On the [variables tab](#) set this variable:

NAME	VALUE
system.prefergit	true

Allow scripts to access the system token

- [YAML](#)
- [Designer](#)

Add a `checkout` section with `persistCredentials` set to `true`.

```
steps:  
- checkout: self  
  persistCredentials: true
```

Learn more about [checkout](#).

On the [options tab](#) select **Allow scripts to access OAuth token**.

Make sure to clean up the local repo

Certain kinds of changes to the local repository are not automatically cleaned up by the build pipeline. So make sure to:

- Delete local branches you create.

- Undo git config changes.

If you run into problems using an on-premises agent, make sure the repo is clean:

- [YAML](#)
- [Designer](#)

Make sure `checkout` has `clean` set to `true`.

```
steps:
- checkout: self
  clean: true
```

- On the [repository tab](#) set **Clean** to true.
- On the [variables tab](#) create or modify the `Build.Clean` variable and set it to `source`

Examples

List the files in your repo

Make sure to follow the above steps to [enable Git](#).

On the [build tab](#) add this task:

TASK	ARGUMENTS
 Utility: Command Line List the files in the Git repo.	Tool: <code>git</code> Arguments: <code>ls-files</code>

Merge a feature branch to master

You want a CI build to merge to master if the build succeeds.

Make sure to follow the above steps to [enable Git](#).

On the [Triggers tab](#) select **Continuous integration (CI)** and include the branches you want to build.

Create `merge.bat` at the root of your repo:

```
@echo off
ECHO SOURCE BRANCH IS %BUILD_SOURCEBRANCH%
IF %BUILD_SOURCEBRANCH% == refs/heads/master (
  ECHO Building master branch so no merge is needed.
  EXIT
)
SET sourceBranch=origin/%BUILD_SOURCEBRANCH:refs/heads/=%
ECHO GIT CHECKOUT MASTER
git checkout master
ECHO GIT STATUS
git status
ECHO GIT MERGE
git merge %sourceBranch% -m "Merge to master"
ECHO GIT STATUS
git status
ECHO GIT PUSH
git push origin
ECHO GIT STATUS
git status
```

On the [build tab](#) add this as the last task:

TASK	ARGUMENTS
 Utility: Batch Script Run merge.bat.	Path: <code>merge.bat</code>

Q & A

Can I run Git commands if my remote repo is in GitHub or an external Git service such as Bitbucket?

Yes

Which tasks can I use to run Git commands?

[Batch Script](#)

[Command Line](#)

[PowerShell](#)

[Shell Script](#)

How do I avoid triggering a CI build when the script pushes?

Add `***NO_CI***` to your commit message. For example,

```
git merge origin/features/hello-world -m "Merge to master ***NO_CI***"
```

How does enabling scripts to run Git commands affect how the build pipeline gets build sources?

When you set `system.preferrgit` to `true`, the build pipeline uses command-line Git instead of LibGit2Sharp to clone or fetch the source files.

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Library

10/9/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

Library is a collection of *shared* build and release assets for a project. Assets defined in a library can be used in multiple build and release pipelines of the project. The **Library** tab can be accessed directly in Azure Pipelines and Team Foundation Server (TFS).

At present, the library contains two types of assets: [variable groups](#) and [secure files](#).

Variable groups are available to only release pipelines in Azure Pipelines and TFS 2017 and newer at present. Task groups and service connections are available to build and release pipelines in TFS 2015 and newer, and Azure Pipelines.

Library Security

All assets defined in the **Library** tab share a common security model. You can control who can define new items in a library, and who can use an existing item. **Roles** are defined for library items, and **membership** of these roles governs the operations you can perform on those items.

ROLE ON A LIBRARY ITEM	PURPOSE
Reader	Members of this role can view the item.
User	Members of this role can use the item when authoring build or release pipelines. For example, you must be a 'User' for a variable group to be able to use it in a release pipeline.
Administrator	In addition to all the above operations, members of this role can manage membership of all other roles for the item. The user that created an item is automatically added to the Administrator role for that item.

The security settings for the **Library** tab control access for *all* items in the library. Role memberships for individual items are automatically inherited from those of the **Library** node. In addition to the three roles listed above, the **Creator** role on the library defines who can create new items in the library, but cannot be used to manage permissions for other users. By default, the following groups are added to the **Administrator** role of the library: **Build Administrators**, **Release Administrators**, and **Project Administrators**.

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Variable groups for builds and releases

11/19/2018 • 5 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Use a variable group to store values that you want to make available across multiple build and release pipelines. Variable groups are defined and managed in the **Library** tab of the **Pipelines** hub.

NOTE

Variable groups can be used in a build pipeline in only Azure DevOps and TFS 2018. They cannot be used in a build pipeline in earlier versions of TFS.

Create a variable group

1. Open the **Library** tab to see a list of existing variable groups for your project. Choose **+ Variable group**.
2. Enter a name and description for the group. Then enter the name and value for each **variable** you want to include in the group, choosing **+ Add** for each one. If you want to encrypt and securely store the value, choose the "lock" icon at the end of the row. When you're finished adding variables, choose **Save**.

Properties

Variable group name

HO Website Configuration

Description

Variables used to configure head office website

 Link secrets from an Azure key vault as variables
Variables

Name ↑	Value	Lock
app-location	Head_Office	
app-name	Fabrikam	
password	*****	
requires-login	True	
user-name	*****	
+ Add		

Variable groups follow the [library security model](#).

Link secrets from an Azure key vault as variables

Link an existing Azure key vault to a variable group and map selective vault secrets to the variable group.

1. In the **Variable groups** page, enable **Link secrets from an Azure key vault as variables**. You'll need an existing key vault containing your secrets. You can create a key vault using the [Azure portal](#).

Link secrets from an Azure key vault as variables [\(i\)](#)

Azure subscription * | [Manage](#)

! This setting is required.

Key vault name * [Manage](#)

! This setting is required.

2. Specify your Azure subscription end point and the name of the vault containing your secrets.

Ensure the Azure service connection has at least **Get** and **List** management permissions on the vault for secrets. You can enable Azure Pipelines to set these permissions by choosing **Authorize** next to the vault name. Alternatively, you can set the permissions manually in the [Azure portal](#):

- Open the **Settings** blade for the vault, choose **Access policies**, then **Add new**.
- In the **Add access policy** blade, choose **Select principal** and select the service principal for your client account.
- In the **Add access policy** blade, choose **Secret permissions** and ensure that **Get** and **List** are checked

(ticked).

- Choose **OK** to save the changes.
3. In the **Variable groups** page, choose **+ Add** to select specific secrets from your vault that will be mapped to this variable group.

Secrets management notes

- Only the secret *names* are mapped to the variable group, not the secret values. The latest version of the value of each secret is fetched from the vault and used in the pipeline linked to the variable group during the build or release.
- Any changes made to *existing* secrets in the key vault, such as a change in the value of a secret, will be made available automatically to all the definitions in which the variable group is used.
- When new secrets are added to the vault, or a secret is deleted from the vault, the associated variable groups are not updated automatically. The secrets included in the variable group must be explicitly updated in order for the definitions using the variable group to execute correctly.
- Azure Key Vault supports storing and managing cryptographic keys and secrets in Azure. Currently, Azure Pipelines variable group integration supports mapping only secrets from the Azure key vault. Cryptographic keys and certificates are not yet supported.

Use a variable group

- [Schema](#)
- [Example](#)

To use a variable group, reference it in your variables mapping:

```
variables:  
- group: my-variable-group
```

YAML builds are not yet available on TFS.

You access the value of the variables in a linked variable group in exactly the same way as [variables you define within the pipeline itself](#). For example, to access the value of a variable named **customer** in a variable group linked to the pipeline, use `$(customer)` in a task parameter or a script. However, secret variables (encrypted variables and key vault variables) cannot be accessed directly in scripts - instead they must be passed as arguments to a task.

Note: At present, variables in different groups that are linked to a pipeline in the same scope (such as a release or stage scope) will collide and the result may be unpredictable. Ensure that you use different names for variables across all your variable groups.

Any changes made centrally to a variable group, such as a change in the value of a variable or the addition of new variables, will automatically be made available to all the definitions or stages to which the variable group is linked.

Variable groups in a build or release

- When a new instance of a build or release is created from a pipeline definition, the values of the variables from the linked variable group are copied to the build or release.
- To override the values of variables in the variable group you must create a variable with the same name within the build or release pipeline. A variable in the pipeline overrides a variable with the same name in the variable group.

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Task groups for builds and releases

11/19/2018 • 5 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

NOTE

Task groups are not supported in YAML pipelines. Instead, in that case you can use templates. See [YAML schema reference](#).

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.

The new task group is automatically added to the task catalogue, ready to be added to other release and build pipelines. Task groups are stored at the project level, and are not accessible outside the project scope.

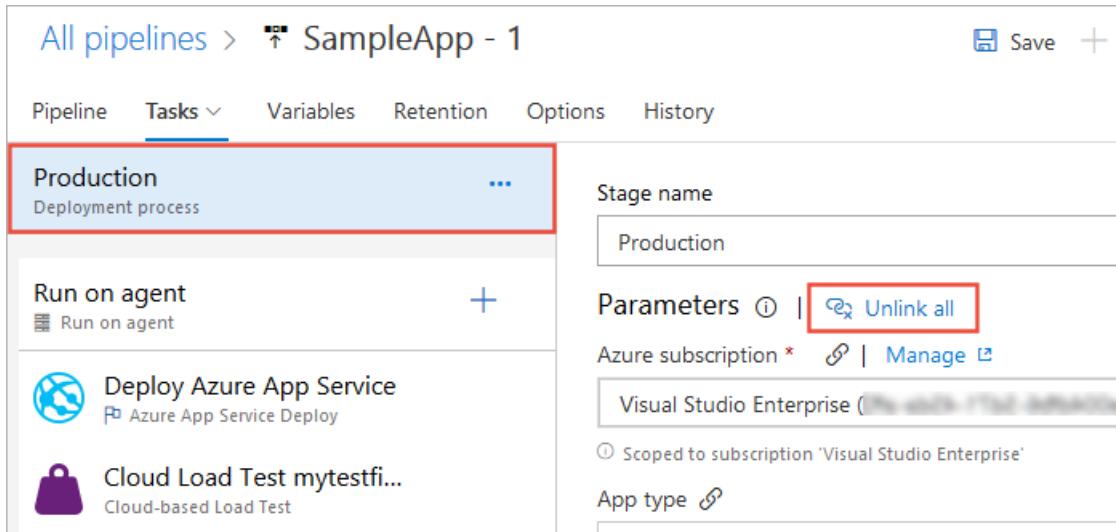
Task groups are a way to standardize and centrally manage deployment steps for all your applications. When you include a task group in your definitions, and then make a change centrally to the task group, the change is automatically reflected in all the definitions that use the task group. There is no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

Before you create a task group...

- Ensure that all of the tasks you want to include in a task group have their parameters defined as variables, such as **\$(MyVariable)**, where you want to be able to configure these parameters when you use the task group. Variables used in the tasks are automatically extracted and converted into parameters for the task group. Values of these configuration variables will be converted into default values for the task group.
- If you specify a value (instead of a variable) for a parameter, that value becomes a fixed parameter value and cannot be exposed as a parameter to the task group.
- Parameters of the encapsulated tasks for which you specified a value (instead of a variable), or you didn't provide a value for, are not configurable in the task group when added to a build or release pipeline.
- Task conditions (such as "Run this task only when a previous task has failed" for a **PowerShell Script** task) can be configured in a task group and these settings are persisted with the task group.
- When you save the task group, you can provide a name and a description for the new task group, and select a category where you want it to appear in the **Task catalog** dialog. You can also change the default values for each of the parameters.
- When you queue a build or a release, the encapsulated tasks are extracted and the values you entered for the task group parameters are applied to the tasks.
- Changes you make to a task group are reflected in every instance of the task group.

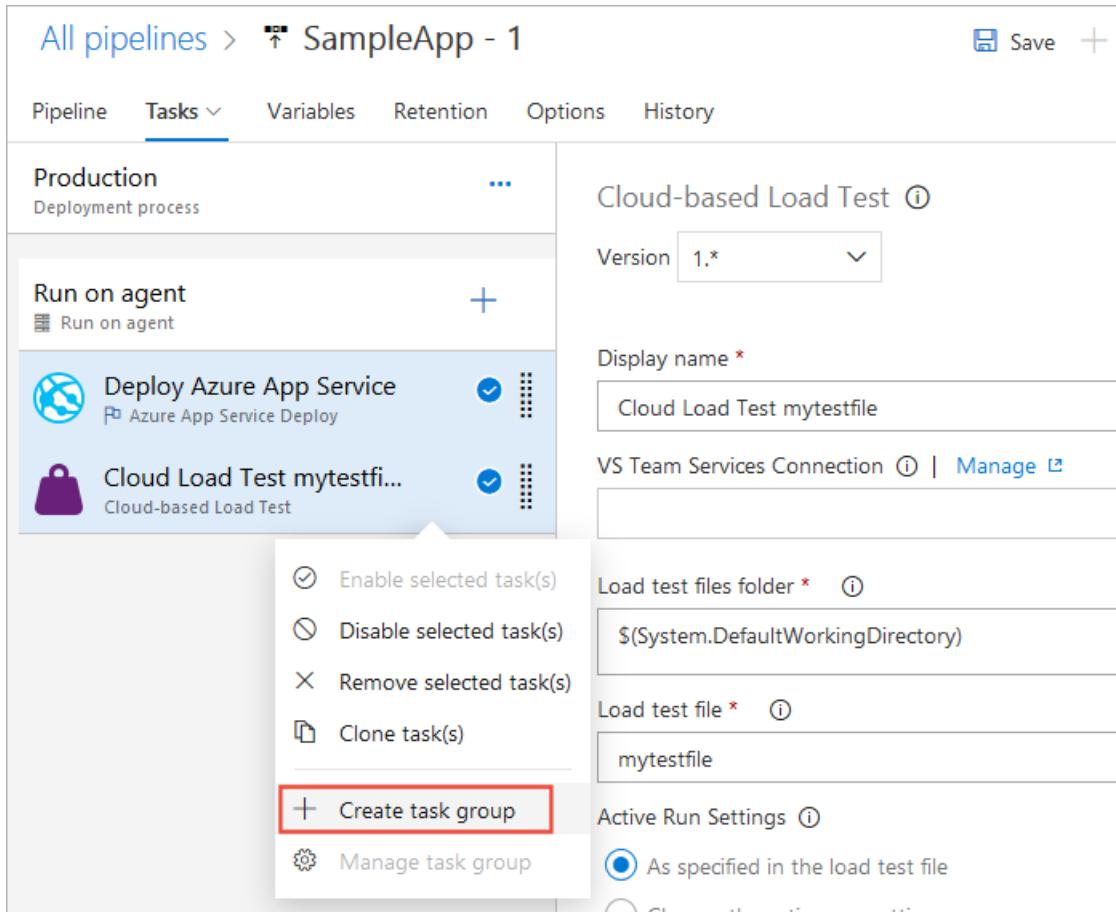
Create a task group

1. Ensure that all the tasks you intend to include do not contain any linked parameters. The easy way to do this is to choose **Unlink all** in the settings panel for the entire process.



The screenshot shows the 'Tasks' tab of a pipeline named 'Production'. A specific task, 'Deploy Azure App Service', is selected. In the settings panel on the right, the 'Parameters' section has an 'Unlink all' button highlighted with a red box.

2. Select a sequence of tasks in a build or release pipeline (when using a mouse, click on the checkmarks of each one). Then open the shortcut menu and choose **Create task group**.



The screenshot shows the 'Tasks' tab of a pipeline named 'Production'. Two tasks, 'Deploy Azure App Service' and 'Cloud Load Test mytestfile...', are selected. A context menu is open, and the '+ Create task group' option is highlighted with a red box. The task group configuration pane on the right shows fields for 'Display name' (set to 'Cloud Load Test mytestfile'), 'VS Team Services Connection' (set to 'Manage'), 'Load test files folder' (set to '\$(System.DefaultWorkingDirectory)'), 'Load test file' (set to 'mytestfile'), and 'Active Run Settings' (set to 'As specified in the load test file').

3. Specify a name and description for the new task group, and the category (tab in the Add tasks panel) you want to add it to.
4. After you choose **Create**, the new task group is created and replaces the selected tasks in your pipeline.
5. Save your updated pipeline.

Manage task groups

All the task groups you create in the current project are listed in the **Task Groups** page of **Azure Pipelines**.

The screenshot shows the 'Task Groups' page in Azure Pipelines. A task group named 'Standard deployment tasks' is selected. A context menu is open over this group, with the 'Export' option highlighted. Other options in the menu include 'Delete' and 'Security'. The page also lists other task groups like 'FabrikamTests'.

Use the **Export** shortcut command to save a copy of the task group as a JSON pipeline, and the **Import** icon to import previously saved task group definitions. Use this feature to transfer task groups between projects and enterprises, or replicate and save copies of your task groups.

Select a task group name to open the details page.

The screenshot shows the details page for the 'Standard deployment tasks' task group. On the left, there's a list of tasks including 'Deploy Azure App Service' and 'Quick Web Performance...'. On the right, the task group properties are displayed: Version 1.*, Name: Standard deployment tasks, Description: Fabrikam standard set of tasks for deployment, and Category: Deploy.

- In the **Tasks** page you can edit the tasks that make up the task group. For each encapsulated task you can change the parameter values for the non-variable parameters, edit the existing parameter variables, or convert parameter values to and from variables. When you save the changes, all definitions that use this task group will pick up the changes.
- In the **History** tab you can see the history of changes to the group.
- In the **References** tab you can expand lists of all the build and release pipelines, and other task groups, that use (reference) this task group. This is useful to ensure changes do not have unexpected effects on other processes.

Create previews and updated versions of task groups

All of the built-in tasks in Azure Pipelines and TFS are **versioned**. This allows build and release pipelines to continue to use the existing version of a task while new versions are developed, tested, and released. In Azure Pipelines, you can version your own custom task groups so that they behave in the same way and provide the same advantages.

1. After you finish editing a task group, choose **Save as draft** instead of **Save**.

The screenshot shows the 'Task Groups' interface for 'Standard deployment tasks'. The 'Save' button in the top right has a dropdown menu with two options: 'Save' and 'Save as draft'. The 'Save as draft' option is highlighted with a red box. To the right of the dropdown, there's a 'Version' field set to '3.*'. Below the save buttons, there are fields for 'Display name' (set to 'Deploy Azure App Service'), 'Azure subscription' (set to 'FabrikamRM'), 'App type' (set to 'Web App'), and 'App Service name' (set to 'NewWebAppWithTests'). A checkbox for 'Deploy to slot' is also present.

2. The string **-test** is appended to the task group version number. When you are happy with the changes, choose **Publish draft**. You can choose whether to publish it as a preview or as a production-ready version.

The screenshot shows the 'Task Groups' interface for 'Standard deployment tasks (Draft)'. The 'Publish draft' button in the top right is highlighted with a red box. A modal dialog box titled 'Publish draft task group Standard deployment tasks' is open. It contains a warning message: 'Changes made to the draft version will overwrite the parent task group version and the draft task group will be deleted. Are you sure you want to proceed?'. Below the message, there's a note: 'If your changes are not backward compatible, it is recommended that you publish the draft as a preview version'. A checkbox for 'Publish as preview' is checked and highlighted with a red box. There's a 'Comment' input field and two buttons at the bottom: 'Publish' (blue) and 'Cancel'.

3. You can now use the updated task group in your build and release processes; either by changing the version number of the task group in an existing pipeline or by adding it from the **Add tasks** panel.

The screenshot shows the 'Tasks' tab in the 'All definitions > Fabrikam' pipeline. A task group named 'Task group: Standard de...' is selected. On the right, the 'Standard deployment tasks' panel is open. A red box highlights the 'Version' dropdown menu, which lists '1.*' and '2.* (preview)'.

As with the built-in tasks, the default when you add a task group is the highest non-preview version.

4. After you have finished testing the updated task group, choose **Publish preview**. The **Preview** string is removed from the version number string. It will now appear in definitions as a "production-ready" version.

The screenshot shows the 'Task Groups' page with 'Standard deployment tasks (Preview)' selected. A red box highlights the 'Publish preview' button. The 'Version' dropdown is set to '2.*'. On the right, the task group properties are shown, including 'Name *' set to 'Standard deployment tasks'.

5. In a build or release pipeline that already contains this task group, you can now select the new "production-ready" version. When you add the task group from the **Add tasks** panel, it automatically selects the new "production-ready" version.

The screenshot shows the 'Tasks' tab in the 'All definitions > Fabrikam' pipeline. The 'Task group: Standard deployment tasks' task is selected. A red box highlights the 'Version' dropdown menu, which lists '2.*' and '2.*'.

Related topics

- [Tasks](#)
- [Task jobs](#)

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Service connections for builds and releases

11/29/2018 • 22 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

You will typically need to connect to external and remote services to execute tasks for a build or deployment. For example, you may need to connect to your Microsoft Azure subscription, to a different build server or file server, to an online continuous integration environment, or to services you install on remote computers.

You can define service connections in Azure Pipelines or Team Foundation Server (TFS) that are available for use in all your tasks. For example, you can create a service connection for your Azure subscription and use this service connection name in an Azure Web Site Deployment task in a release pipeline.

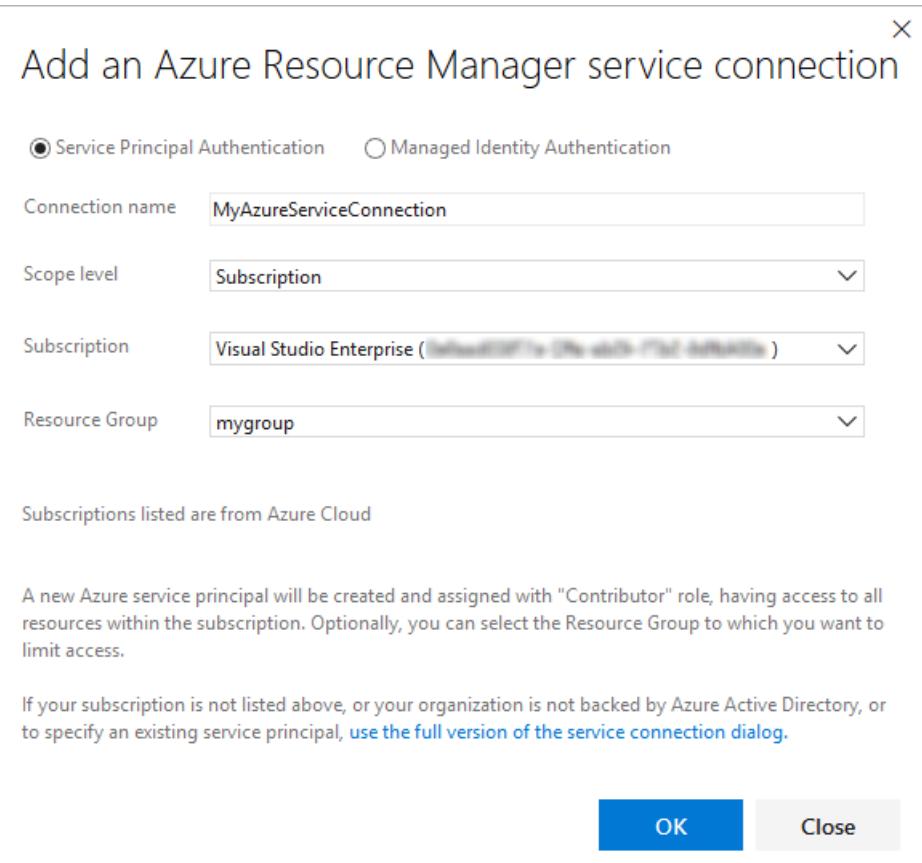
You define and manage service connections from the Admin settings of your project:

- Azure DevOps: https://dev.azure.com/{organization}/{project}/_admin/_services
- TFS: https://{tfsserver}/{collection}/{project}/_admin/_services

Service connections are created at project scope. A service connection created in one project is not visible in another project.

Create a service connection

1. In Azure DevOps, open the **Service connections** page from the [project settings page](#). In TFS, open the **Services** page from the "settings" icon in the top menu bar.
2. Choose **+ New service connection** and select the type of service connection you need.
3. Fill in the parameters for the service connection. The list of parameters differs for each type of service connection - see the [following list](#). For example, this is the default **Azure Resource Manager** connection dialog:



4. Choose **OK** to create the connection.

You can also create your own [custom service connections](#).

Secure a service connection

You can control who can define new service connections in a library, and who can use an existing service connection. **Roles** are defined for service connections, and **membership** in these roles governs the operations you can perform on those service connections.

ROLE ON A LIBRARY SERVICE CONNECTION	PURPOSE
User	Members of this role can use the service connection when authoring build or release pipelines.
Administrator	In addition to using the service connection, members of this role can manage membership of all other roles for the service connection. The user that created the service connection is automatically added to the Administrator role for that service connection.

Two special groups called **Service connection administrators** and **Service connection creators** are added to every project. Members of the Service connection administrators group can manage all service connections. By default, project administrators are added as members of this group. This group is also added as an administrator to every service connection created. Members of the Service connection creators group can create new service connections. By default, project contributors are added as members of this group.

To modify the security for a connection:

1. In Azure DevOps, open the **Service connections** page from the [project settings page](#). In TFS, open the **Services** page from the "settings" icon in the top menu bar.

- Choose the **Roles** link to open the security tab.

User	Role	Access
[Project]\Endpoint Administrators	Administrator	Inherited
SB	Administrator	Assigned

- Add users or groups, turn on and off inheritance, or change the role for existing users and groups as required.

Use a service connection

After the new service connection is created:

- If you are using it in the UI, select the connection name you assigned in the **Azure subscription** (or the equivalent connection name) setting of your pipeline.

- If you are using it in YAML, copy the connection name into your code as the **azureSubscription** (or the equivalent connection name) value.

```

DotNetCore
  deploy
  dotnetcore-sample
  dotnetcore-tests
  .gitattributes
  .gitignore
  .vsts-ci.yml *
  ...
  Dockerfile
  dotnetcore-sample.sln
  k8config.yml
  README.md

Contents  Highlight changes
25  displayName: dotnet build
26
27 - task: dotNetCoreCLI@1
28   inputs:
29     command: publish
30     arguments: --configuration release --output $(Build.ArtifactStagingDirectory)
31     zipAfterPublish: true
32   displayName: dotnet publish
33
34 - task: publishBuildArtifacts@1
35   inputs:
36     PathToPublish: $(Build.ArtifactStagingDirectory)
37     ArtifactName: drop
38     ArtifactType: Container
39   displayName: Publish the artifacts
40
41 - task: AzureRmWebAppDeployment@3
42   inputs:
43     azureSubscription: 'MyARMConnection'
44     WebAppName: 'MyWebApp'
45

```

You can also create your own [custom service connections](#).

Common service connection types

Azure Pipelines and TFS support a variety of service connection types by default. Some of these are described below:

- [Azure Classic service connection](#)
- [Azure Resource Manager service connection](#)
- [Azure Service Bus service connection](#)
- [Bitbucket Cloud service connection](#)
- [Chef service connection](#)
- [Docker Host service connection](#)
- [Docker Registry service connection](#)
- [External Git service connection](#)
- [Generic service connection](#)
- [GitHub service connection](#)
- [GitHub Enterprise service connection](#)
- [Jenkins service connection](#)
- [Kubernetes service connection](#)
- [npm service connection](#)
- [NuGet service connection](#)
- [Python package download service connection](#)
- [Python package upload service connection](#)
- [Service Fabric service connection](#)
- [SSH service connection](#)
- [Subversion service connection](#)
- [Team Foundation Server / Azure Pipelines service connection](#)
- [Visual Studio App Center service connection](#)

After you enter the parameters when creating a service connection, validate the connection. The validation link uses a REST call to the external service with the information you entered, and indicates if the call succeeded.

Azure Classic service connection

Defines and secures a connection to a Microsoft Azure subscription using Azure credentials or an Azure

management certificate. [How do I create a new service connection?](#)

PARAMETER	DESCRIPTION
[authentication type]	Required. Select Credentials or Certificate based .
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the azureSubscription or the equivalent subscription name value in the script.
Environment	Required. Select Azure Cloud , Azure Stack , or one of the pre-defined Azure Government Clouds where your subscription is defined.
Subscription ID	Required. The GUID-like identifier for your Azure subscription (not the subscription name). You can copy this from the Azure portal.
Subscription Name	Required. The name of your Microsoft Azure subscription (account).
User name	Required for Credentials authentication. User name of a work or school account (for example @fabrikam.com). Microsoft accounts (for example @live or @hotmail) are not supported.
Password	Required for Credentials authentication. Password for the user specified above.
Management Certificate	Required for Certificate based authentication. Copy the value of the management certificate key from your publish settings XML file or the Azure portal.

If your subscription is defined in an [Azure Government Cloud](#), ensure your application meets the relevant compliance requirements before you configure a service connection.

Azure Resource Manager service connection

Defines and secures a connection to a Microsoft Azure subscription using Service Principal Authentication (SPA). The dialog offers two modes:

- **Automated subscription detection.** In this mode, Azure Pipelines and TFS will attempt to query Azure for all of the subscriptions and instances to which you have access using the credentials you are currently logged on with in Azure Pipelines or TFS (including Microsoft accounts and School or Work accounts). If no subscriptions are shown, or subscriptions other than the one you want to use, you must sign out of Azure Pipelines or TFS and sign in again using the appropriate account credentials.
- **Manual subscription pipeline.** In this mode, you must specify the service principal you want to use to connect to Azure. The service principal specifies the resources and the access levels that will be available over the connection. Use this approach when you need to connect to an Azure account using different credentials from those you are currently logged on with in Azure Pipelines or TFS. This is also a useful way to maximize security and limit access.

For more information, see [Create an Azure service connection](#)

NOTE: If you don't see any Azure subscriptions or instances, or you have problems validating the connection, see [Troubleshoot Azure Resource Manager service connections](#).

Azure Service Bus service connection

Defines and secures a connection to a Microsoft Azure Service Bus queue.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the azureSubscription or the equivalent subscription name value in the script.
Service Bus ConnectionString	The URL of your Azure Service Bus instance. More information .
Service Bus Queue Name	The name of an existing Azure Service Bus queue.

[How do I create a new service connection?](#)

Bitbucket service connection

Defines a connection to a Bitbucket server.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the azureSubscription or the equivalent subscription name value in the script.
User name	Required. The username to connect to the service.
Password	Required. The password for the specified username.

[How do I create a new service connection?](#)

Chef service connection

Defines and secures a connection to a [Chef](#) automation server.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the azureSubscription or the equivalent subscription name value in the script.
Server URL	Required. The URL of the Chef automation server.
Node Name (Username)	Required. The name of the node to connect to. Typically this is your username.

PARAMETER	DESCRIPTION
Client Key	Required. The key specified in the Chef .pem file.

[How do I create a new service connection?](#)

Docker Host service connection

Defines and secures a connection to a Docker host.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the azureSubscription or the equivalent subscription name value in the script.
Server URL	Required. The URL of the Docker host.
CA Certificate	Required. A trusted certificate authority certificate to use to authenticate with the host.
Certificate	Required. A client certificate to use to authenticate with the host.
Key	Required. The key specified in the Docker key.pem file.

Ensure you protect your connection to the Docker host. [Learn more](#).

[How do I create a new service connection?](#)

Docker Registry service connection

Defines and secures a connection to a Docker registry.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the azureSubscription , endpoint , or the equivalent name value in the script.
Docker Registry	Required. The URL of the Docker registry. A default value is provided.
Docker ID	Required. The identifier of the Docker account user. For Azure Container Registry, this is likely to be a service principal.
Password	Required. The password for the account user identified above.
Email	Optional. An email address to receive notifications.

[How do I create a new service connection?](#)

External Git service connection

Defines and secures a connection to a Git repository server. Note that there is a specific service connection for [GitHub](#) and [GitHub Enterprise](#) connections.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the azureSubscription or the equivalent subscription name value in the script.
Server URL	Required. The URL of the Git repository server.
User name	Required. The username to connect to the Git repository server.
Password/Token Key	Required. The password or access token for the specified username.

Also see [Artifact sources](#).

[How do I create a new service connection?](#)

Generic service connection

Defines and secures a connection to any other type of service or application.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the azureSubscription or the equivalent subscription name value in the script.
Server URL	Required. The URL of the service.
User name	Required. The username to connect to the service.
Password/Token Key	Required. The password or access token for the specified username.

[How do I create a new service connection?](#)

GitHub service connection

Defines a connection to a GitHub repository. Note that there is a specific service connection for [External Git servers](#) and [GitHub Enterprise](#) connections.

PARAMETER	DESCRIPTION
Choose authorization	Required. Either Grant authorization or Personal access token . See notes below.

PARAMETER	DESCRIPTION
Token	Required for Personal access token authorization. See notes below.
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the azureSubscription or the equivalent subscription name value in the script.

How do I create a new service connection?

NOTE

If you select **Grant authorization** for the **Choose authorization** option, the dialog shows an **Authorize** button that opens the GitHub login page. If you select **Personal access token** you must obtain a suitable token and paste it into the **Token** textbox. The dialog shows the recommended scopes for the token: **repo, user, admin:repo_hook**. See [this page](#) on GitHub for information about obtaining an access token. Then register your GitHub account in your profile:

- Open your profile from your organization name at the right of the Azure Pipelines page heading.
- At the top of the left column, under **DETAILS**, choose **Security**.
- In the **Security** tab, in the right column, choose **Personal access tokens**.
- Choose the **Add** link and enter the information required to create the token.

Also see [Artifact sources](#).

GitHub Enterprise service connection

Defines a connection to a GitHub repository. Note that there is a specific service connection for [External Git servers](#) and [standard GitHub service connections](#).

PARAMETER	DESCRIPTION
Choose authorization	Required. Either Personal access token , Username and Password , or OAuth2 . See notes below.
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the azureSubscription or the equivalent subscription name value in the script.
Server URL	Required. The URL of the service.
Accept untrusted SSL certificates	Set this option to allow clients to accept a self-signed certificate instead of installing the certificate in the TFS service role or the computers hosting the agent .
Token	Required for Personal access token authorization. See notes below.
User name	Required for Username and Password authentication. The username to connect to the service.

PARAMETER	DESCRIPTION
Password	Required for Username and Password authentication. The password for the specified username.
OAuth configuraton	Required for OAuth2 authorization. The OAuth configuration specified in your account.
GitHub Enterprise configuration URL	The URL is fetched from OAuth configuration.

[How do I create a new service connection?](#)

NOTE

If you select **Personal access token** you must obtain a suitable token and paste it into the **Token** textbox. The dialog shows the recommended scopes for the token: **repo, user, admin:repo_hook**. See [this page](#) on GitHub for information about obtaining an access token. Then register your GitHub account in your profile:

- Open your profile from your account name at the right of the Azure Pipelines page heading.
- At the top of the left column, under **DETAILS**, choose **Security**.
- In the **Security** tab, in the right column, choose **Personal access tokens**.
- Choose the **Add** link and enter the information required to create the token.

Jenkins service connection

Defines a connection to the Jenkins service.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the azureSubscription or the equivalent subscription name value in the script.
Server URL	Required. The URL of the service.
Accept untrusted SSL certificates	Set this option to allow clients to accept a self-signed certificate instead of installing the certificate in the TFS service role or the computers hosting the agent .
User name	Required. The username to connect to the service.
Password	Required. The password for the specified username.

[How do I create a new service connection?](#)

Also see [Azure Pipelines Integration with Jenkins](#) and [Artifact sources](#).

Kubernetes service connection

Defines and secures a connection to a [Kubernetes](#) automation account.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the azureSubscription or the equivalent subscription name value in the script.
Server URL	Required. The URL of the Kubernetes automation service.
Kubeconfig	The contents of the kubectl configuration file.

[How do I create a new service connection?](#)

npm service connection

Defines and secures a connection to an npm server.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the azureSubscription or the equivalent subscription name value in the script.
Registry URL	Required. The URL of the npm server.
Username	Required when connection type is Username and Password . The username for authentication.
Password	Required when connection type is Username and Password . The password for the username.
Personal Access Token	Required when connection type is External Azure Pipelines . The token to use to authenticate with the service. Learn more .

[How do I create a new service connection?](#)

NuGet service connection

Defines and secures a connection to a NuGet server.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the azureSubscription or the equivalent subscription name value in the script.
Feed URL	Required. The URL of the NuGet server.
ApiKey	Required when connection type is ApiKey . The authentication key.

PARAMETER	DESCRIPTION
Personal Access Token	Required when connection type is External Azure Pipelines . The token to use to authenticate with the service. Learn more .
Username	Required when connection type is Basic authentication . The username for authentication.
Password	Required when connection type is Basic authentication . The password for the username.

[How do I create a new service connection?](#)

Python package download service connection

Defines and secures a connection to a Python repository for downloading Python packages.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the azureSubscription or the equivalent subscription name value in the script.
Python repository url for download	Required. The URL of the Python repository.
Personal Access Token	Required when connection type is Authentication Token . The token to use to authenticate with the service. Learn more .
Username	Required when connection type is Username and Password . The username for authentication.
Password	Required when connection type is Username and Password . The password for the username.

[How do I create a new service connection?](#)

Python package upload service connection

Defines and secures a connection to a Python repository for uploading Python packages.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the azureSubscription or the equivalent subscription name value in the script.
Python repository url for upload	Required. The URL of the Python repository.
EndpointName	Required. Unique repository name used for twine upload. Spaces and special characters are not allowed.

PARAMETER	DESCRIPTION
Personal Access Token	Required when connection type is Authentication Token . The token to use to authenticate with the service. Learn more .
Username	Required when connection type is Username and Password . The username for authentication.
Password	Required when connection type is Username and Password . The password for the username.

[How do I create a new service connection?](#)

Service Fabric service connection

Defines and secures a connection to a Service Fabric cluster.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the azureSubscription or the equivalent subscription name value in the script.
Cluster Endpoint	Required. The TCP endpoint of the cluster.
Server Certificate Thumbprint	Required when connection type is Certificate based or Azure Active Directory .
Client Certificate	Required when connection type is Certificate based .
Password	Required when connection type is Certificate based . The certificate password.
Username	Required when connection type is Azure Active Directory . The username for authentication.
Password	Required when connection type is Azure Active Directory . The password for the username.
Use Windows security	Required when connection type is Others .
Cluster SPN	Required when connection type is Others and using Windows security.

[How do I create a new service connection?](#)

SSH service connection

Defines and secures a connection to a remote host using Secure Shell (SSH).

PARAMETER	DESCRIPTION
-----------	-------------

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the azureSubscription or the equivalent subscription name value in the script.
Host name	Required. The name of the remote host machine or the IP address.
Port number	Required. The port number of the remote host machine to which you want to connect. The default is port 22.
User name	Required. The username to use when connecting to the remote host machine.
Password or passphrase	The password or passphrase for the specified username if using a keypair as credentials.
Private key	The entire contents of the private key file if using this type of authentication.

[How do I create a new service connection?](#)

Also see [SSH task](#) and [Copy Files Over SSH](#).

Subversion service connection

Defines and secures a connection to the Subversion repository.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the azureSubscription or the equivalent subscription name value in the script.
Server repository URL	Required. The URL of the repository.
Accept untrusted SSL certificates	Set this option to allow the client to accept self-signed certificates installed on the agent computer(s).
Realm name	Optional. If you use multiple credentials in a build or release pipeline, use this parameter to specify the realm containing the credentials specified for this service connection.
User name	Required. The username to connect to the service.
Password	Required. The password for the specified username.

[How do I create a new service connection?](#)

Team Foundation Server / Azure Pipelines service connection

Defines and secures a connection to another TFS or Azure DevOps organization.

PARAMETER	DESCRIPTION
(authentication)	Select Basic or Token Based authentication.
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the azureSubscription or the equivalent subscription name value in the script.
Connection URL	Required. The URL of the TFS or Azure Pipelines instance.
User name	Required for Basic authentication. The username to connect to the service.
Password	Required for Basic authentication. The password for the specified username.
Personal Access Token	Required for Token Based authentication (TFS 2017 and newer and Azure Pipelines only). The token to use to authenticate with the service. Learn more .

How do I create a new service connection?

Use the **Verify connection** link to validate your connection information.

See also [Authenticate access with personal access tokens for Azure DevOps and TFS](#).

Visual Studio App Center service connection

Defines and secures a connection to Visual Studio App Center.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the azureSubscription or the equivalent subscription name value in the script.
API Token	Required. The token to use to authenticate with the service. Learn more .

How do I create a new service connection?

Extensions for other service connections

Other service connection types and tasks can be installed in Azure Pipelines and Team Foundation Server as extensions. Some examples of service connections currently available through extensions are:

- [TFS artifacts for Azure Pipelines](#). Deploy on-premises TFS builds with Azure Pipelines through a TFS service connection connection and the **Team Build (external)** artifact, even when the TFS machine is not reachable directly from Azure Pipelines. For more information, see [External TFS](#) and [this blog post](#).
- [TeamCity artifacts for Azure Pipelines](#). This extension provides integration with TeamCity through a TeamCity service connection, enabling artifacts produced in TeamCity to be deployed by using Azure Pipelines. See [TeamCity](#) for more details.

- [SCVMM Integration](#). Connect to a System Center Virtual Machine Manager (SCVMM) server to easily provision virtual machines and perform actions on them such as managing checkpoints, starting and stopping VMs, and running PowerShell scripts.
- [VMware Resource Deployment](#). Connect to a VMware vCenter Server from Visual Studio Team Services or Team Foundation Server to provision, start, stop, or snapshot VMware virtual machines.

You can also create your own [custom service connections](#).

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Secure files

11/6/2018 • 2 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

Use the **Secure Files** library to store files such as signing certificates, Apple Provisioning Profiles, Android Keystore files, and SSH keys on the server without having to commit them to your source repository. Secure files are defined and managed in the **Library** tab in **Azure Pipelines**.

The contents of the secure files are encrypted and can only be used during the build or release pipeline by referencing them from a task. The secure files are available across multiple build and release pipelines in the project based on the security settings. Secure files follow the [library security model](#).

There's a size limit of 10 MB for each secure file.

Q & A

How can I consume secure files in a Build or Release Pipeline?

Use the [Download Secure File](#) Utility task to consume secure files within a Build or Release Pipeline.

How can I create a custom task using secure files?

You can build your own tasks that use secure files by using inputs with type `secureFile` in the `task.json`. Learn [how to build a custom task](#).

The Install Apple Provisioning Profile task is a simple example of a task using a secure file. See the [reference documentation](#) and [source code](#).

To handle secure files during build or release, you can refer to the common module available [here](#).

My task can't access the secure files. What do I do?

Make sure your agent is running version of 2.116.0 or higher. See [Agent version and upgrades](#).

Parallel jobs

10/9/2018 • 6 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#)

You can use a *parallel job* in Azure Pipelines to run a single build job or a single release job at a time in your organization. In Azure Pipelines, you can run parallel jobs on Microsoft-hosted infrastructure or on your own (self-hosted) infrastructure.

Microsoft-hosted CI/CD

If you want to run your builds and releases on machines that Microsoft manages, use *Microsoft-hosted parallel jobs*. Your jobs run on our pool of [Microsoft-hosted agents](#).

We provide a *free tier* of service by default in your Azure DevOps Services organization:

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

When the free tier is no longer sufficient:

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

Self-hosted CI/CD

If you want Azure Pipelines to orchestrate your builds and releases, but use your own machines to run them, use *self-hosted parallel jobs*. You start by deploying our [self-hosted agents](#) on your machines. You can register any number of these self-hosted agents in your Azure DevOps Services organization. We charge based on the number of jobs you want to run at a time, not the number of agents registered.

We provide a *free tier* of service by default in your Azure DevOps Services organization:

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

When the free tier is no longer sufficient:

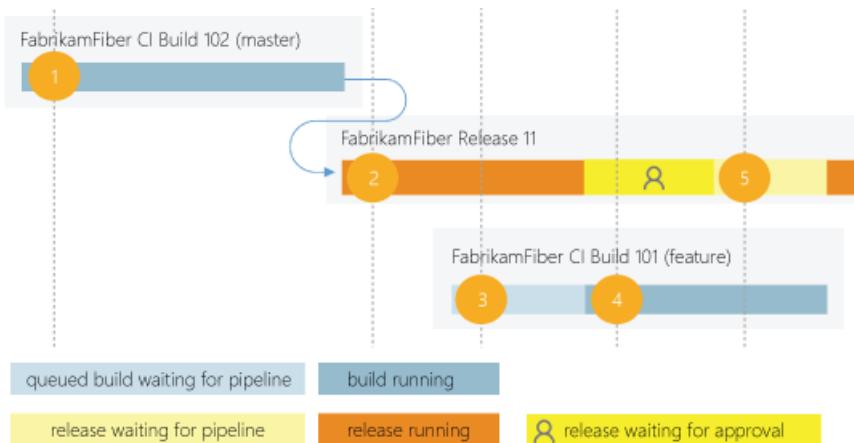
- Public project: [Contact us](#) to get your free tier limits increased.
- Private project: You can pay for additional capacity per parallel job. [Buy self-hosted parallel jobs](#).

There are no time limits on self-hosted jobs.

How a parallel job is consumed by a build or release

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

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



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

Relationship between jobs and parallel jobs

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

- When you define a build or release, you can define it as a collection of [jobs](#). When a build or release runs, you can run multiple jobs as part of that build or release.
- Each job consumes a *parallel job* that runs on an agent. When there aren't enough parallel jobs available for your organization, then the jobs are queued up and run one after the other.
- When you run a [server job](#) or deploy to a [deployment group](#), you don't consume any parallel jobs.

Determine how many parallel jobs you need

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

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

Simple estimate

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

Detailed estimate

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

- If you have multiple teams, and if each of them require a CI build, you'll likely need a parallel job for each team.

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

View available parallel jobs

1. Browse to **Organization settings > Pipelines > Retention and parallel jobs > Parallel jobs**.

Host Type	Parallel Jobs Available	Details
Microsoft-hosted	0/1800 minutes consumed	View in-progress jobs
Self-hosted	2	Parallel jobs Free parallel jobs: 1 Visual Studio Enterprise subscribers: 1 Monthly purchases: 0

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

2. View the maximum number of parallel jobs that are available in your organization.
3. Select **View in-progress jobs** to display all the builds and releases that are actively consuming an available parallel job or that are queued waiting for a parallel job to be available.

Sharing of parallel jobs across projects in a collection

Parallel jobs are purchased at the organization level, and they are shared by all projects in an organization. Currently, there isn't a way to partition or dedicate parallel job capacity to a specific project or agent pool. For example:

1. You purchase two parallel jobs in your organization.
2. You queue two builds in the first project, and both the parallel jobs are consumed.
3. You queue a build in the second project. That build won't start until one of the builds in your first project is completed.

Q&A

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

We'll automatically apply the free tier limits for public projects if you meet both of these conditions:

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

Are there limits on who can use Azure Pipelines?

You can have as many users as you want when you're using Azure Pipelines. There is no per-user charge for

using Azure Pipelines. Users with both [basic and stakeholder access](#) can author as many builds and releases as they want.

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

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

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

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

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

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

- When you're using the per-minute plan, you can run only one job at a time.
- If you run builds for more than 14 paid hours in a month, the per-minute plan might be less cost-effective than the parallel jobs model.

I use XAML build controllers with my organization. How am I charged for those?

You can register one XAML build controller for each self-hosted parallel job in your organization. Your organization gets at least one free self-hosted parallel job, so you can register one XAML build controller for no additional charge. For each additional XAML build controller, you'll need an additional self-hosted parallel job.

Parallel release jobs in Team Foundation Server

10/9/2018 • 4 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

This article describes the licensing model for Azure Pipelines in Team Foundation Server 2017 (TFS 2017) or newer. We don't charge you for Team Foundation Build (TFBuild) so long as you have a TFS Client Access License (CAL).

A TFS *parallel job* gives you the ability to run a single release at a time in a project collection. You can keep hundreds or even thousands of release jobs in your collection. But, to run more than one release at a time, you need additional parallel jobs.

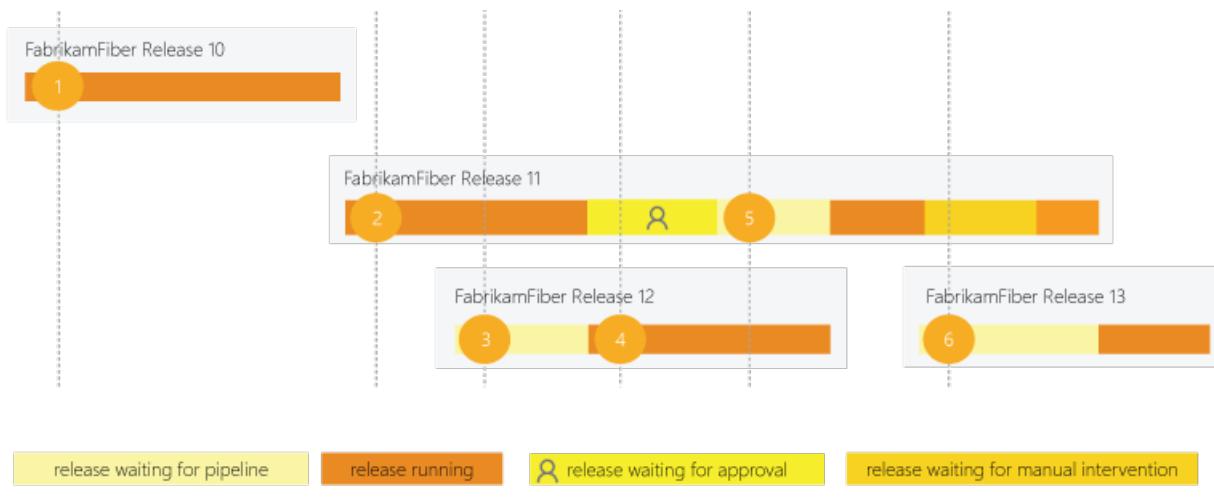
One free parallel job is included with every collection in a Team Foundation server. Every Visual Studio Enterprise subscriber in a Team Foundation server contributes one additional parallel job. You can buy additional private jobs from the Visual Studio Marketplace.

Do I need parallel jobs in TFS 2015? Short answer: no. [More details](#)

How a parallel job is consumed

For example, a collection in a Team Foundation server has one parallel job. This allows users in that collection to run only one release at a time. When additional releases are triggered, they are queued and will wait for the previous one to complete.

A release requires a parallel job only when it is being actively deployed to a stage. Waiting for an approval does not consume a parallel job. However, waiting for a manual intervention in the middle of a deployment does consume a parallel job.



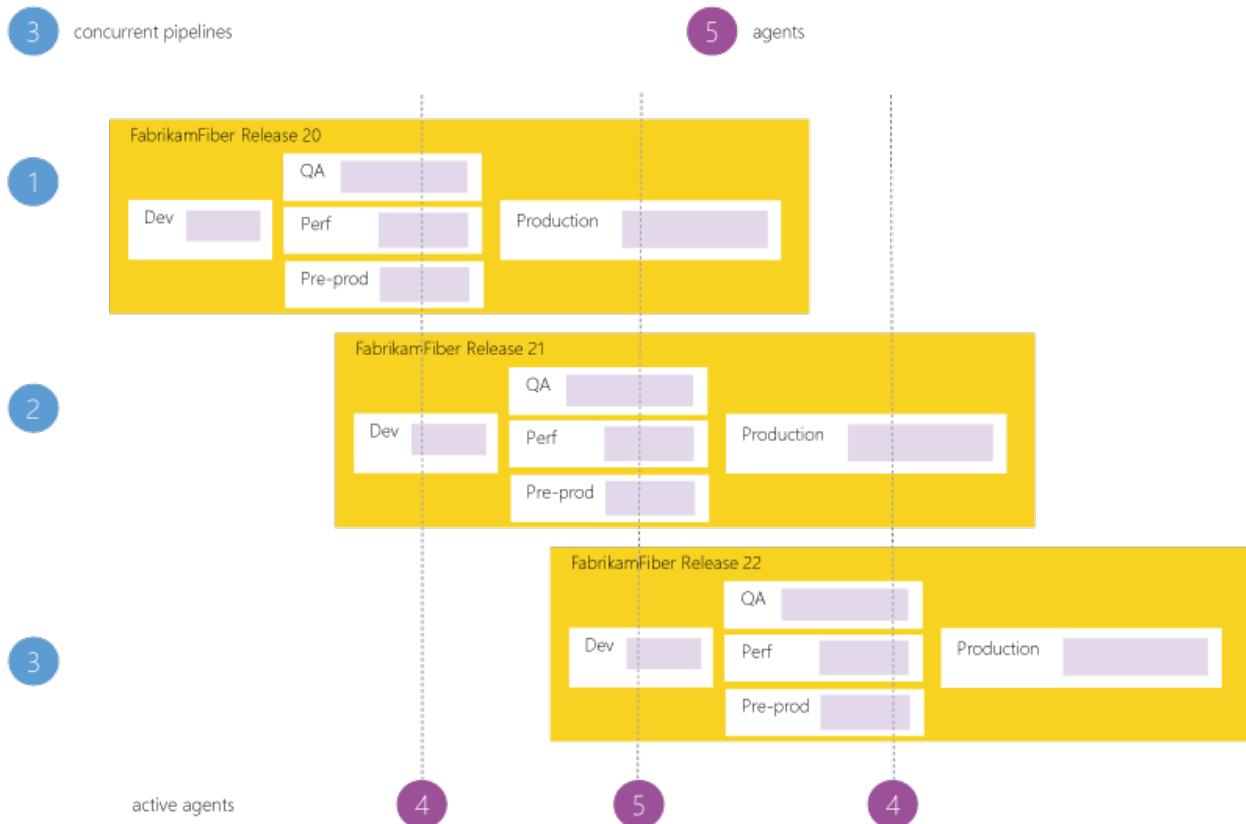
1. FabrikamFiber Release 10 is first to be deployed.
2. Deployment of FabrikamFiber Release 11 starts after Release 10's deployment is complete.
3. Release 12 is queued until Release 11's deployment is active.
4. Release 11 waits for an approval. Release 12's deployment starts because a release waiting for approvals does not consume a parallel job.
5. Even though Release 11 is approved, it resumes only after Release 12's deployment is completed.
6. Release 11 is waiting for manual intervention. Release 13 cannot start because the manual intervention state consumes a parallel job.

Manual intervention does not consume a job in TFS 2017.1 and newer.

Parallel processing within a single release

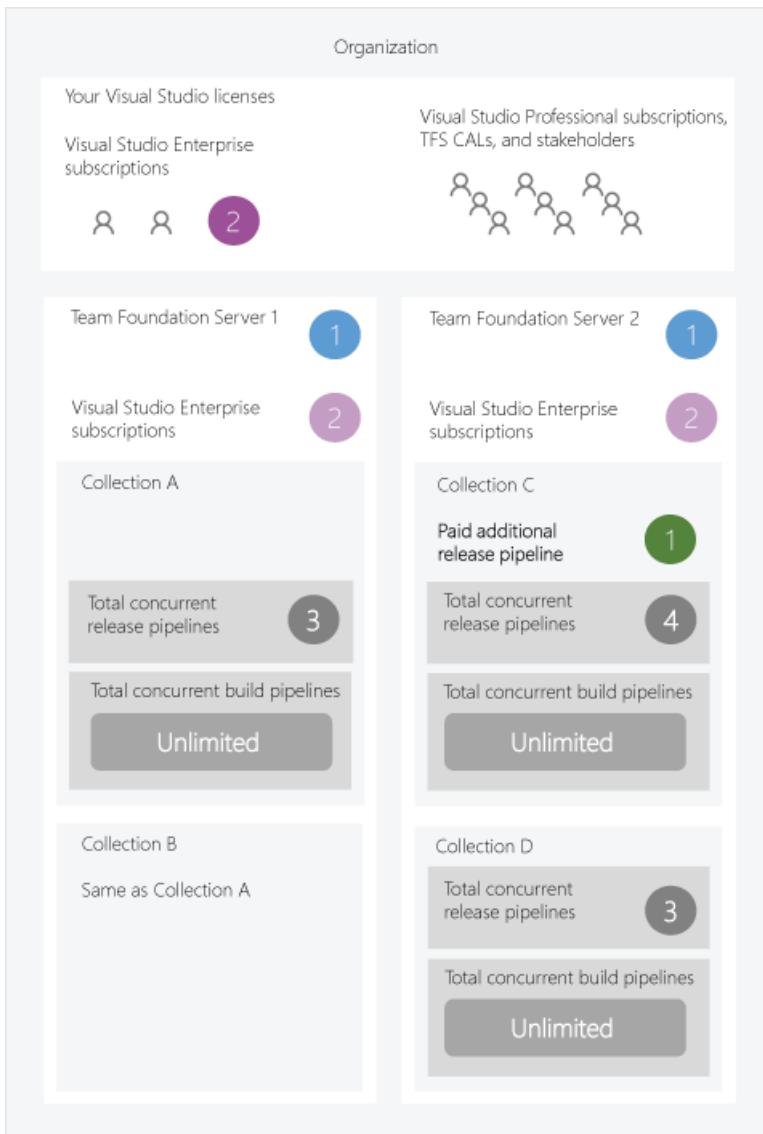
Parallel processing within a single release does not require additional parallel jobs. So long as you have enough agents, you can deploy to multiple stages in a release at the same time.

For example, suppose your collection has three parallel jobs. You can have more than three agents running at the same time to perform parallel operations within releases. For instance, notice below that four or five agents are actively running jobs from three parallel jobs.



Parallel jobs in an organization

For example, here's an organization that has multiple Team Foundation Servers. Two of their users have Visual Studio Enterprise subscriptions that they can use at the same time across all their on-premises servers and in each collection so long as the customer adds them as users to both the servers as explained below.



Determine how many parallel jobs you need

You can begin by seeing if your teams can get by with the parallel jobs you've got by default. As the number of queued releases exceeds the number of parallel jobs you have, your release queues will grow longer. When you find the queue delays are too long, you can purchase additional parallel jobs as needed.

Simple estimate

A simple rule of thumb: Estimate that you'll need one parallel job for every 10 users in your server.

Detailed estimate

In the following scenarios you might need multiple parallel jobs:

- If you have multiple teams, if each of them require a CI build, and if each of the CI builds is configured to trigger a release, then you'll likely need a parallel job for each team.
- If you develop multiple applications in one collection, then you'll likely need additional parallel jobs: one to deploy each application at the same time.

Use your Visual Studio Enterprise subscription benefit

Users who have Visual Studio Enterprise subscriptions are assigned to **VS Enterprise** access level in the Users hub of TFS instance. Each of these users contributes one additional parallel job to each collection. You can use this benefit on all Team Foundation Servers in your organization.

1. Browse to **Server settings, Access levels**.

The screenshot shows the top navigation bar of the Team Foundation Server interface. It includes the TFS logo, 'Team Foundation Server' text, a dropdown arrow, 'Home' and 'Rooms' links, and a gear icon for settings. Below the main bar, there's a secondary navigation bar with three items: 'Control panel', 'Access levels' (which is underlined, indicating it's the current page), and 'Legacy extensions'.

URL example: http://{your_server}:8080/tfs/_admin/_licenses

2. On the left side of the page, click **VS Enterprise**.
3. Add your users who have Visual Studio Enterprise subscriptions.

After you've added these users, additional licenses will appear on the resource limits page described below.

Purchase additional parallel jobs

If you need to run more parallel releases, you can [buy additional private jobs from the Visual Studio marketplace](#). Since there is no way to directly purchase parallel jobs from Marketplace for a TFS instance at present, you must first buy parallel jobs for an Azure DevOps organization. After you buy the private jobs for an Azure DevOps organization, you enter the number of purchased parallel jobs manually on the resource limits page described below.

View and manage parallel jobs

1. Browse to **Collection settings, Pipelines, Resource limits**.

The screenshot shows the top navigation bar for the 'DefaultCollection'. It includes the collection name 'DefaultCollection', a dropdown arrow, 'Home' and 'Rooms' links, and a gear icon for settings. Below the main bar, there's a secondary navigation bar with several items: 'Overview', 'Users', 'Security', 'Build and Release' (which is underlined, indicating it's the current page), 'Agent pools', 'Extensions', 'Settings', and 'Resource limits' (which is also underlined).

URL example: http://{your_server}:8080/tfs/DefaultCollection/_admin/_buildQueue?a=resourceLimits

2. View or edit the number of purchased parallel jobs.

Q&A

Who can use the system?

TFS users with a [TFS CAL](#) can author as many releases as they want.

To approve releases, a TFS CAL is not necessary; any user with [stakeholder access](#) can approve or reject releases.

Do I need parallel jobs to run builds on TFS?

No, on TFS you don't need parallel jobs to run builds. You can run as many builds as you want at the same time for no additional charge.

Do I need parallel jobs to manage releases in versions before TFS 2017?

No.

In TFS 2015, so long as your users have a TFS CAL, they can manage releases for no additional charge in trial mode. We called it "trial mode" to indicate that we would eventually charge for managing releases. Despite this label, we fully support managing releases in TFS 2015.

How are releases licensed in Azure Pipelines?

See [Parallel jobs in Azure Pipelines](#).

Build and release retention policies

10/29/2018 • 8 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#) | [Previous versions \(XAML builds\)](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Retention policies are used to configure how long builds and releases are to be retained by the system. The primary reasons to delete older builds and releases are to conserve storage and to reduce clutter. The main reasons to keep builds and releases are for audit and tracking.

Build retention

In most cases you don't need to retain completed builds longer than a certain number of days. Using build retention policies, you can control **how many days** you want to keep each build before deleting it and the **minimum number of builds** that should be retained for each pipeline.

As an author of a build pipeline, you can customize retention policies for builds of your pipeline on the **Retention** tab. You can also customize these policies on a branch-by-branch basis if you are building from [Git repositories](#).

Global build retention policy

If you are using an on-premises Team Foundation Server, you can specify build retention policy defaults and maximums for a project collection. You can also specify when builds are permanently destroyed (removed from the **Deleted** tab in the build explorer).

If you are using Azure Pipelines, you can view but not change these settings for your organization.

Global build retention policy settings can be managed from the **Pipelines** settings of your organization or project collection:

- Azure Pipelines: https://dev.azure.com/{your_organization}/_admin/_buildQueue
- TFS 2017 and newer: https://{your_server}/tfs/DefaultCollection/_admin/_buildQueue
- TFS 2015.3: http://{your_server}:8080/tfs/DefaultCollection/_admin/_buildQueue
- TFS 2015 RTM: http://{your_server}:8080/tfs/DefaultCollection/_admin/_buildQueue#a=settings

The **maximum retention policy** sets the upper limit for how long builds can be retained for all build pipelines. Authors of build pipelines cannot configure settings for their definitions beyond the values specified here.

The **default retention policy** sets the default retention values for all the build pipelines. Authors of build pipelines can override these values.

The **build destruction policy** helps you keep the builds for a certain period of time after they are deleted. This policy cannot be overridden in individual build pipelines.

Git repositories

If your [repository type](#) is one of the following, you can define multiple retention policies with branch filters:

- Azure Repos Git or TFS Git
- GitHub
- External Git

For example, your team may want to keep:

- User branch builds for five days, with a minimum of a single successful or partially successful build for each branch.
- Master and feature branch builds for 10 days, with a minimum of three successful or partially successful builds for each of these branches. You exclude a special feature branch that you want to keep for a longer period of time.
- Builds from the special feature branch and all other branches for 15 days, with a minimum of a single successful or partially successful build for each branch.

The following example retention policy for a build pipeline meets the above requirements:

The screenshot shows the 'Retention' tab in the Azure DevOps 'Builds' configuration. It displays three separate retention rules:

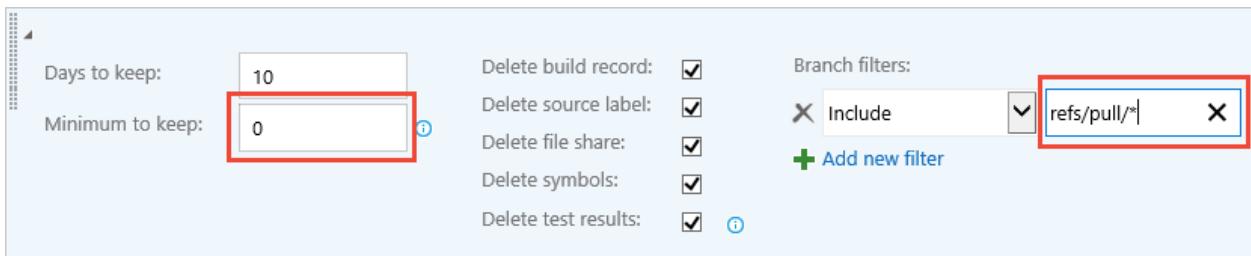
- Rule 1 (Top):** Days to keep: 5, Minimum to keep: 1. Delete build record: Delete source label: Delete test results: Branch filters: Include users/*. An 'Add new filter' button is available.
- Rule 2 (Middle):** Days to keep: 10, Minimum to keep: 3. Delete build record: Delete source label: Delete test results: Branch filters: Include master, Include features/*, Exclude features/special. An 'Add new filter' button is available.
- Rule 3 (Bottom):** Days to keep: 15, Minimum to keep: 1. Delete build record: Delete source label: Delete test results: Branch filters: Include *. An 'Add new filter' button is available.

When specifying custom policies for each pipeline, you cannot exceed the maximum limits set by administrator.

Clean up pull request builds

If you [protect your Git branches with pull request builds](#), then you can use retention policies to automatically delete the completed builds. To do it, add a policy that keeps a minimum of builds with the following branch filter:

```
refs/pull/*
```



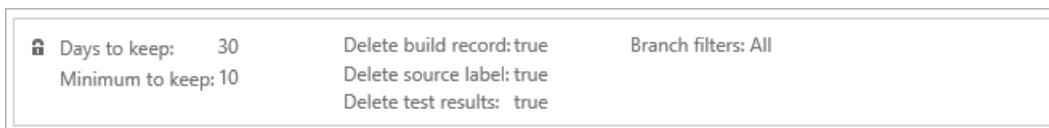
TFVC and Subversion repositories

For TFVC and Subversion [repository types](#) you can modify a single policy with the same options shown above.

Policy order

When the system is purging old builds, it evaluates each build against the policies in the order you have specified. You can drag and drop a policy lower or higher in the list to change this order.

The "All" branches policy is automatically added as the last policy in the evaluation order to enforce the maximum limits for all other branches.



What parts of the build get deleted

When the retention policies mark a build for deletion, you can control which information related to the build is deleted:

- Build record: You can choose to delete the entire build record or keep basic information about the build even after the build is deleted.
- Source label: If you label sources as part of the build, then you can choose to delete the tag (for Git) or the label (for TFVC) created by a build.
- Automated test results: You can choose to delete the automated test results associated with the build (for example, results published by the Publish Test Results build task).

The following information is deleted when a build is deleted:

- Logs
- [Published artifacts](#)
- [Published symbols](#)

When are builds deleted

Azure Pipelines

Your retention policies are processed once per day. The timing of this process varies because we spread the work throughout the day for load balancing purposes. There is no option to change this process.

TFS

Your retention policies run every day at 3:00 A.M. UTC. There is no option to change this process.

Release retention

The release retention policies for a release pipeline determine how long a release and the build linked to it are retained. Using these policies, you can control **how many days** you want to keep each release after it has been last modified or deployed and the **minimum number of releases** that should be retained for each pipeline. The retention timer on a release is reset every time a release is modified or deployed to a stage. The minimum number of releases to retain setting takes precedence over the number of days. For example, if you specify to retain a minimum of three releases, the most recent three will be retained indefinitely - irrespective of the number of days specified. However, you can manually delete these releases when you no longer require them.

As an author of a release pipeline, you can customize retention policies for releases of your pipeline on the **Retention** tab. You can also customize these policies on a [stage-by-stage basis](#).

Global release retention policy

If you are using an on-premises Team Foundation Server, you can specify release retention policy defaults and maximums for a project. You can also specify when releases are permanently destroyed (removed from the **Deleted** tab in the build explorer).

If you are using Azure Pipelines, you can view but not change these settings for your project.

Global release retention policy settings can be managed from the **Release** settings of your project:

- Azure Pipelines:

```
https://dev.azure.com/{your_organization}/{project}/_admin/_apps/hub/ms.vss-releaseManagement-web.release-project-admin-hub
```

- On-premises:

```
https://{your_server}/tfs/{collection_name}/{project}/_admin/_apps/hub/ms.vss-releaseManagement-web.release-project-admin-hub
```

The **maximum retention policy** sets the upper limit for how long releases can be retained for all release pipelines. Authors of release pipelines cannot configure settings for their definitions beyond the values specified here.

The **default retention policy** sets the default retention values for all the release pipelines. Authors of build pipelines can override these values.

The **destruction policy** helps you keep the releases for a certain period of time after they are deleted. This policy cannot be overridden in individual release pipelines.

In TFS, release retention management is restricted to specifying the number of days, and this is available only in TFS 2015.3 and newer.

Stage-specific retention

You may want to retain more releases that have been deployed to specific stages. For example, your team may want to keep:

- Releases deployed to Production stage for 60 days, with a minimum of three last deployed releases.
- Releases deployed to Pre-production stage for 15 days, with a minimum of one last deployed release.
- Releases deployed to QA stage for 30 days, with a minimum of two last deployed releases.
- Releases deployed to Dev stage for 10 days, with a minimum of one last deployed release.

The following example retention policy for a release pipeline meets the above requirements:

The screenshot shows the 'Retention' tab selected in the top navigation bar. On the left, there are three stages: 'Dev', 'Production', and 'QA', each with its own retention policy. On the right, under 'Settings for Dev', the user can specify 'Days to retain a release' (30), 'Minimum releases to keep' (3), and a checked checkbox for 'Retain associated artifacts'. A link to 'View or manage retention policy defaults' is also present.

In this example, if a release that is deployed to Dev is not promoted to QA for 10 days, it is a potential candidate for deletion. However, if that same release is deployed to QA eight days after being deployed to Dev, its retention timer is reset, and it is retained in the system for another 30 days.

When specifying custom policies per pipeline, you cannot exceed the maximum limits set by administrator.

Interaction between build and release retention

The build linked to a release has its own retention policy, which may be shorter than that of the release. If you want to retain the build for the same period as the release, set the **Retain build** checkbox for the appropriate stages. This overrides the retention policy for the build, and ensures that the artifacts are available if you need to redeploy that release.

When you delete a release pipeline, delete a release, or when the retention policy deletes a release automatically, the retention policy for the associated build will determine when that build is deleted.

In TFS, interaction between build and release retention is available in TFS 2017 and newer.

Q&A

Are manual test results deleted?

No

If I mark a build or a release to be retained indefinitely, does the retention policy still apply?

No. Neither the pipeline's retention policy nor the maximum limits set by the administrator are applied when you mark an individual build or release to be retained indefinitely. It will remain until you stop retaining it indefinitely.

How do I specify that builds deployed to production will be retained longer?

Customize the retention policy on the release pipeline. Specify the number of days that releases deployed to production must be retained. In addition, indicate that builds associated with that release are to be retained. This will override the build retention policy.

I did not mark builds to be retained indefinitely. However, I see a large number of builds being retained. How can I prevent this?

Builds that are deployed as part of releases are also governed by the release retention policy. Customize the release retention policy as explained above.

Are automated test results that are published as part of a release retained until the release is deleted?

Test results published within a stage of a release are associated with both the release and the build. These test results are retained as specified by the retention policy configured for the build and for the test results. If you are

not deploying Team Foundation or Azure Pipelines Build, and are still publishing test results, the retention of these results is governed by the retention settings of the release they belong to.

Set build and release permissions

11/1/2018 • 5 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Add your teammates

If your teammates want to edit pipelines, then have an administrator add them to your project:

1. Make sure you are a member of the Project Administrators group ([learn more](#)).
2. Go to your project summary: <https://dev.azure.com/{your-organization}/{your-project}>
3. Invite the teammates to join the project.

The screenshot shows the 'Add Users to OurProject' dialog. At the top, there's a breadcrumb navigation: OurOrg / OurProject / Overview / Summary. To the right are a search bar, a filter icon, a bag icon, and a user profile icon. Below the breadcrumb, the project name 'OurProject' is displayed with a blue square icon. To the right of the project name are three buttons: 'Private' (disabled), 'Invite' (highlighted with a red box), and a star icon. The main area of the dialog is titled 'Add Users to OurProject' with a close button 'X'. It has two input fields: 'Add users or groups *' containing 'myteammate@fabrikam.com' and 'Add to team(s) *' containing 'OurProject Team'. A note below the second field says: '(i) myteammate@fabrikam.com has not been assigned an access level, and will be assigned the best available.' At the bottom are two buttons: 'Add' (highlighted with a red box) and 'Cancel'.

4. After the teammates accept the invitation, ask them to verify that they can [create and edit pipelines](#).

Confirm that contributors have pipeline permissions

If you created your project after about October 2018, then the above procedure is probably sufficient. However, in some cases your team members might see errors or grayed-out controls when they try to work with pipelines. In these cases, make sure that your project contributors have the necessary permissions:

1. Make sure you are a member of the Build Administrators group or the Project Administrators group ([learn](#)

more).

2. Open the build security dialog box.

The screenshot shows the Azure DevOps interface for a project named 'OurProject'. On the left, there's a sidebar with 'Builds' selected (marked with a red circle 1). In the main area, under 'Pipelines', there's a list of build pipelines: 'All build pipelines' (marked with a red circle 2) and 'OurProject-CL'. A context menu is open over the 'All build pipelines' item, containing options: '...', 'Security' (marked with a red box and red circle 3), 'Rename' (marked with a red circle 4), 'Delete', and 'Copy link'.

3. On the permissions dialog box, make sure the following permissions are set to Allow.

The screenshot shows the 'Permissions for OurProject' dialog box. On the left, under 'DevOps Groups', the 'Contributors' group is selected (marked with a red box and red circle 1). In the 'ACCESS CONTROL SUMMARY' section, several permissions are listed with their current state:

Permission	State
Delete build definition	Allow
Delete builds	Allow
Destroy builds	Allow
Edit build definition	Allow
Edit build quality	Allow
Manage build qualities	Not set
Manage build queue	Not set
Override check-in validation by build	Not set
Queue builds	Allow
Retain indefinitely	Allow
Stop builds	Allow
Update build information	Allow
View build definition	Allow
View builds	Allow

At the bottom of the dialog, there are buttons: 'Remove', 'Save changes' (marked with a red box and red circle 4), 'Undo changes', and 'Close' (marked with a red box and red circle 5).

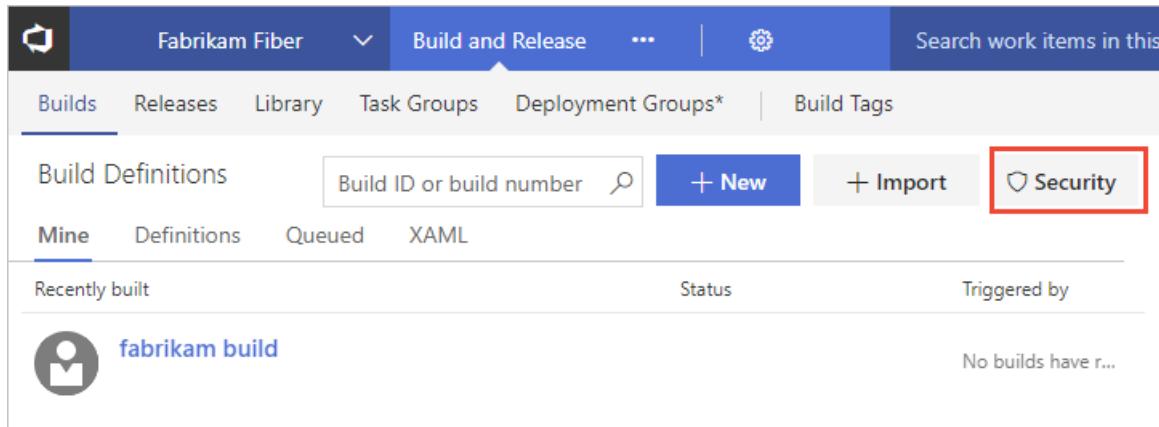
Permissions for build and release functions are primarily set at the object-level for a specific build or release, or for select tasks, at the collection level. For a simplified view of permissions assigned to built-in groups, see [Permissions and access](#).

In addition to permission assignments, you manage security for several resources—such as variable groups, secure files, and deployment groups—by adding users or groups to a role. You grant or restrict permissions by setting the [permission state to Allow or Deny](#), either for a security group or an individual user. For definitions of

each build and release permission and role, see [Build and release permissions](#).

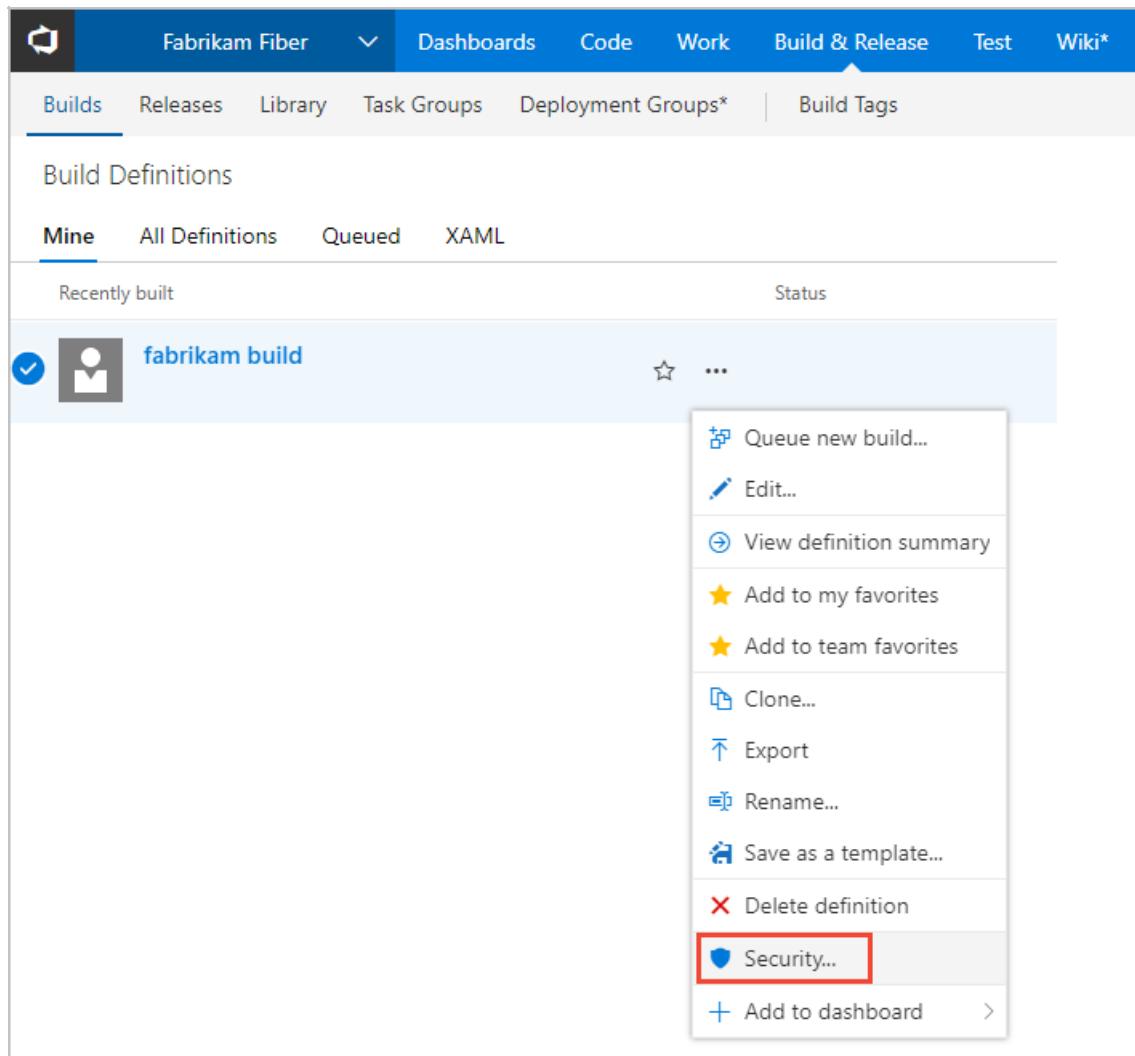
Set permissions for build pipelines

1. To set the permissions for all build pipelines, click the Security From the web portal **Build+Release** hub, **Builds** page



The screenshot shows the Microsoft DevOps Build+Release hub. In the top navigation bar, the 'Build and Release' tab is selected. Below it, the 'Builds' tab is also selected. In the top right corner of the main content area, there is a 'Security' button, which is highlighted with a red box.

To set the permissions for a specific build pipeline, open the context menu for the build and click Security.



The screenshot shows the 'Build Definitions' page for a specific build named 'fabrikam build'. A context menu is open over the build item, and the 'Security...' option is highlighted with a red box. Other options in the menu include 'Queue new build...', 'Edit...', 'View definition summary', 'Add to my favorites', 'Add to team favorites', 'Clone...', 'Export', 'Rename...', 'Save as a template...', 'Delete definition', and 'Add to dashboard'.

2. Choose the group you want to set permissions for, and then change the permission setting to Allow or Deny.

For example, here we change the permission for Edit build pipeline for the Contributors group to Allow.

Permissions for fabrikam build

[+ Add...](#) [Inheritance ▾](#)

Search [...](#)

- ▼ VSTS Groups
 - [Build Administrators](#)
 - [Contributors](#)
 - [Fabrikam Fiber Team](#)
 - [Project Administrators](#)
 - [Readers](#)
 - [Project Collection Administrators](#)
 - [Project Collection Build Administrators](#)
 - [Project Collection Build Service Accounts](#)
 - [Project Collection Test Service Accounts](#)
- ▼ Users
 - [Fabrikam Fiber Build Service \(kelliott\)](#)
 - [Project Collection Build Service \(kelliott\)](#)

ACCESS CONTROL SUMMARY
Shows information about the permissions being granted to this identity

Administer build permissions	Not set
Delete build definition	Not set
Delete builds	Not set
Destroy builds	Not set
Edit build definition	Allow
Edit build quality	Allow (inherited)
Manage build qualities	Not set
Manage build queue	Not set
Override check-in validation by build	Not set
Queue builds	Allow (inherited)
Retain indefinitely	Not set
Stop builds	Not set
Update build information	Not set
View build definition	Allow (inherited)
View builds	Allow (inherited)

[Clear explicit permissions](#)

[Remove](#) [Save changes](#) [Undo changes](#)

[Close](#)

3. Save your changes.

Set permissions for release pipelines

1. From the web portal **Build-Release** hub, **Releases** page, open the Security dialog for all release pipelines.

The screenshot shows the Microsoft DevOps web interface. The top navigation bar includes links for Fabrikam Fiber, Dashboards, Code, Work, Build & Release, Test, and Wiki*. The 'Build & Release' tab is active. Below it, the 'Releases' tab is selected. The main area displays a list titled 'All release definitions'. A tooltip labeled 'Security...' points to a small shield icon next to a '...' button in the list. Other buttons visible include a refresh icon, a plus sign for creating new definitions, and icons for locking and deleting definitions.

If you want to manage the permissions for a specific release, then open the Security dialog for that release.

2. Choose the group you want to set permissions for, and then change the permission setting to Allow or Deny.

For example, here we deny access to several permissions for the Contributors group.

Permission	Setting
Administer release permissions	Not set
Create releases	Allow
Delete release definition	Allow
Delete release environment	Allow
Delete releases	Allow
Edit release definition	Deny
Edit release environment	Allow
Manage deployments	Deny
Manage release approvers	Allow
Manage releases	Deny
View release definition	Allow
View releases	Allow

- Save your changes.

Manage Library roles for variable groups, secure files, and deployment groups

Permissions for [variable groups](#), [secure files](#), and [deployment groups](#) are managed by roles. For a description of the roles, see [About security roles](#).

NOTE

Feature availability: These features are available on Azure Pipelines and TFS 2017 and later versions.

You can set the security for all artifacts for a project, as well as set the security for individual artifacts. The method is similar for all three artifact types. You set the security for variable groups and secure files from **Azure Pipelines, Library** page, and for deployment groups, from the **Deployment groups** page.

For example, here we show how to set the security for variable groups.

- Build-Release hub, **Library** page, open the Security dialog for all variable groups.

The screenshot shows the Azure DevOps interface with the 'Library' tab selected. At the top right, there is a blue button labeled '+ Variable group' and a red-bordered button labeled 'Security'. Below the buttons, there are two links: 'Variable groups' (underlined) and 'Secure files'.

If you want to manage the permissions for a specific variable group, then open the Security dialog for that group.

This screenshot is similar to the first one, but it shows a context menu for a variable group named 'FF group'. The menu includes options for 'Edit', 'Delete', and 'Security'. The 'Security' option is highlighted with a red box.

2. Add the user or group and choose the role you want them to have.

For example, here we deny access to several permissions for the Contributors group.

The screenshot shows the 'Assign security roles for Library' dialog. A red arrow points from the 'Add' button in the main dialog to a sub-dialog titled 'Add user'. In the sub-dialog, a user named 'Raisa Pokrovskaya' is selected, and the 'Role' dropdown is set to 'User'. A note at the bottom states: 'User can use, but cannot manage the library items.' There are 'Add' and 'Close' buttons at the bottom of the sub-dialog.

3. Click **Add**.

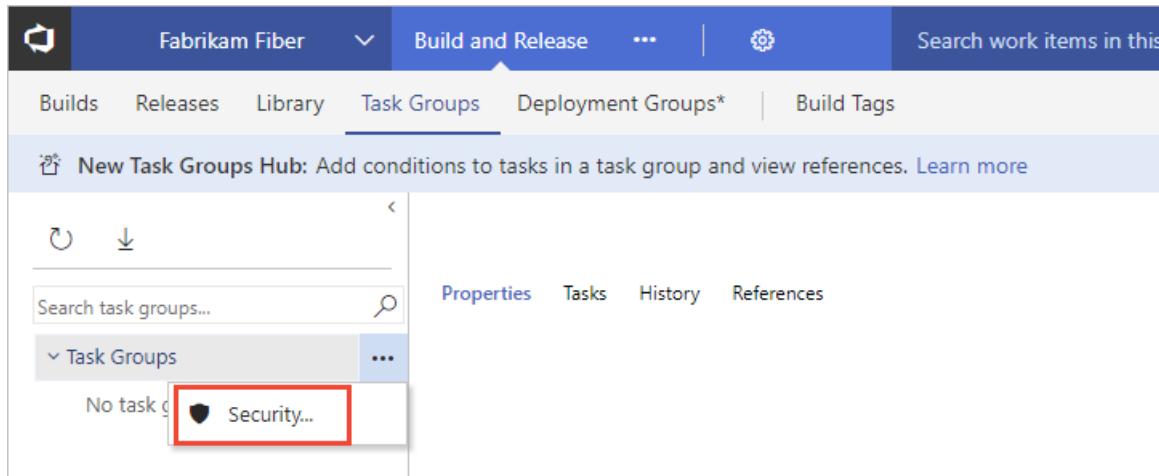
Manage task group permissions

Permissions for task groups are subject to a hierarchical model. You use task groups to encapsulate a sequence of tasks already defined in a build or a release pipeline into a single reusable task. You [define and manage task groups](#) in the **Task groups** tab of **Azure Pipelines**.

NOTE

Feature availability: These features are available on Azure Pipelines and TFS 2017 and later versions.

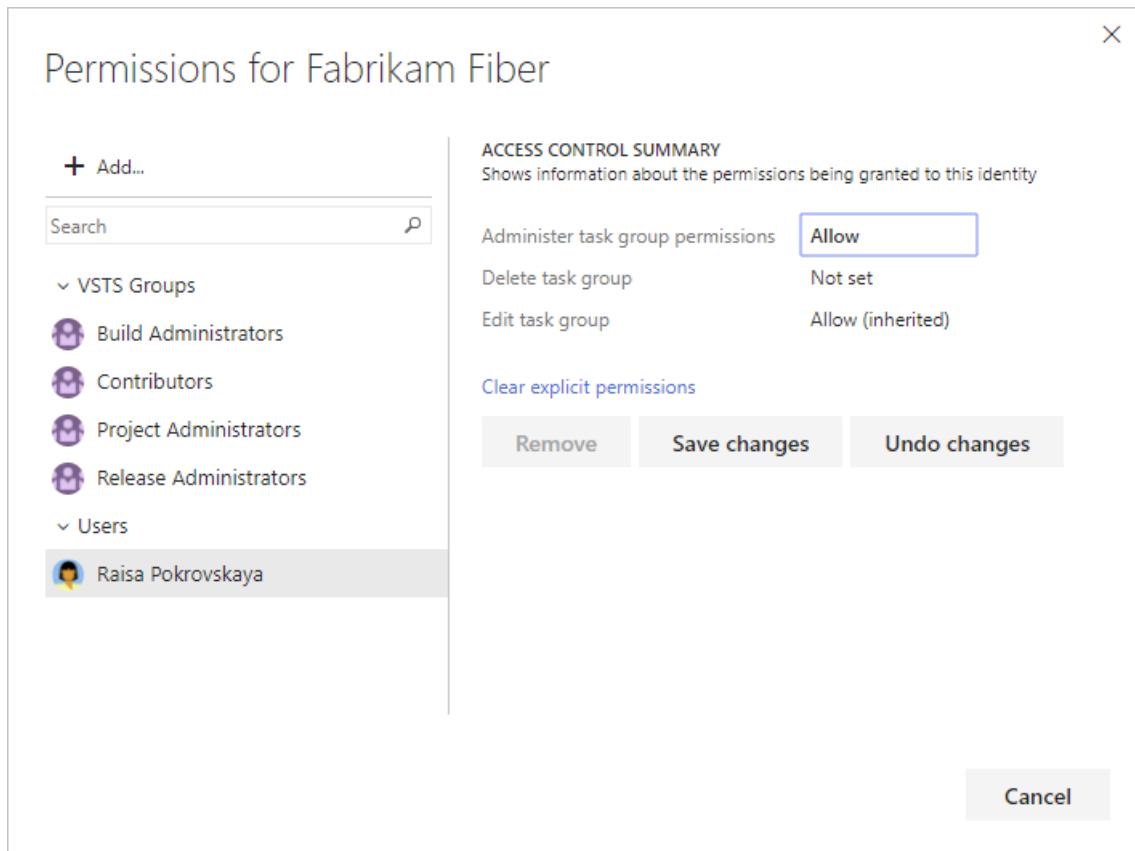
1. From the web portal **Build-Release hub**, **Task groups** page, open the Security dialog for all task groups.



If you want to manage the permissions for a specific task group, then open the Security dialog for that group.

2. Add the user or group and then set the permissions you want them to have.

For example, here we add Raisa and set her permissions to Administer all task groups.



3. Click **Add**.

Set collection-level permissions to administer build resources

1. From the web portal user context, open the admin context by clicking the gear Settings icon and choosing **Organization settings** or **Collection settings**.
2. Click **Security**, and then choose the group whose permissions you want to modify.

Here we choose the Build Administrators group and change the **Use build resources** permission. For a description of each permissions, see [Permissions and groups reference, Collection-level permissions](#).

The screenshot shows the Azure DevOps security settings interface. On the left, a sidebar lists 'VSTS Groups' with several items like 'Freedom Group', 'Project Collection Administrators', and 'Project Collection Build Administrators'. The 'Project Collection Build Administrators' item is selected and highlighted in blue. On the right, the main panel displays the 'Permissions' tab for this group. It shows a list of permissions with their current status: 'Allow' for 'Administer build resource permissions' and 'Use build resources', while other permissions like 'View build resources' and 'View system synchronization information' are set to 'Not set'. At the bottom, there are 'Save changes' and 'Undo changes' buttons.

Permission	Status
Administer build resource permissions	Allow
Administer process permissions	Not set
Administer shelved changes	Not set
Administer workspaces	Not set
Alter trace settings	Not set
Create a workspace	Allow (inherited)
Create new projects	Not set
Create process	Not set
Delete field from account	Not set
Delete process	Not set
Delete team project	Not set
Edit instance-level information	Not set
Edit process	Not set
Make requests on behalf of others	Not set
Manage build resources	Allow
Manage test controllers	Not set
Trigger events	Not set
Use build resources	Allow
View build resources	Allow
View instance-level information	Allow
View system synchronization information	Not set

3. Save your changes.

Manage permissions for agent pools and service connections

You manage the security for [agent pools](#) and [service connections](#) by adding users or groups to a role. The method is similar for both agent pools and service connections. You will need to be a member of the Project Administrator group to manage the security for these resources.

NOTE

Feature availability: These features are available on Azure Pipelines and TFS 2015 and later versions.

For example, here we show how to add a user to the Administrator role for a service connection.

1. From the web portal, click the gear Settings icon to open the project settings admin context.
2. Click **Services**, click the service connection that you want to manage, and then click **Roles**.

User	Role	Access
[Fabrikam Fiber]\Endpoint Administrators	Administrator	Inherited

3. Add the user or group and choose the role you want them to have. For a description of each role, see [About security roles](#).

For example, here we add Raisa to the Administrator role.

User or group	Role
Raisa Pokrovskaya	Administrator

4. Click **Add**.

Manage permissions for agent pools and deployment pools

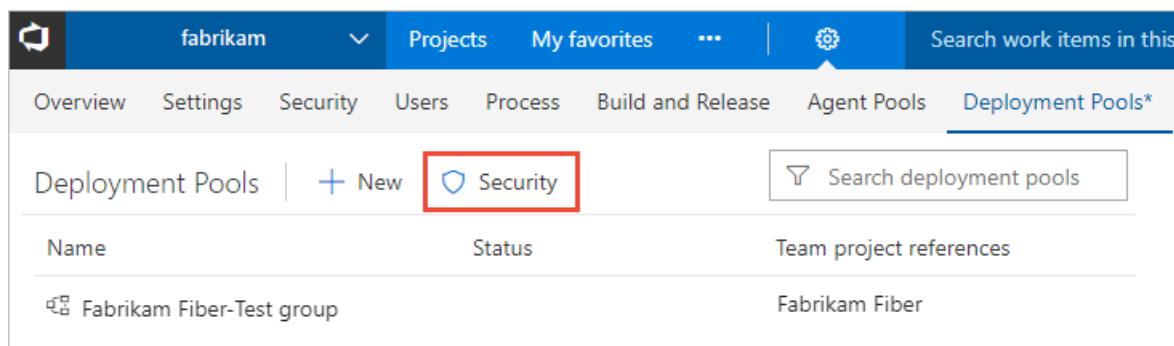
You manage the security for [agent pools](#) and [deployment pools](#) by adding users or groups to a role. The method is similar for both types of pools.

NOTE

Feature availability: These features are available on Azure Pipelines and TFS 2018 and later versions.

You will need to be a member of the Project Collection Administrator group to manage the security for a pool. Once you've been added to the Administrator role, you can then manage the pool. For a description of each role, see [About security roles](#).

1. From the web portal, click the gear icon and choose Organization settings or Collection settings to open the collection-level settings admin context.
2. Click **Deployment Pools**, and then open the **Security** dialog for all deployment pools.

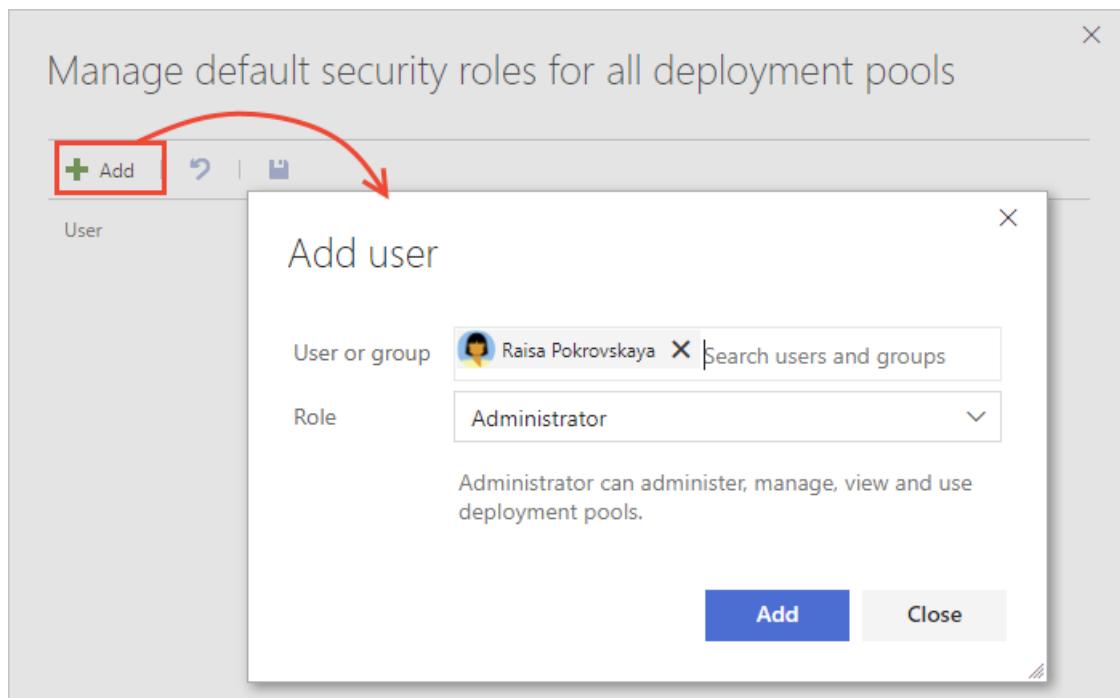


The screenshot shows the 'Deployment Pools' section of the Azure DevOps web portal. At the top, there's a navigation bar with links for Overview, Settings, Security, Users, Process, Build and Release, Agent Pools, and Deployment Pools*. Below this, there's a search bar labeled 'Search work items in this collection'. Under 'Deployment Pools', there are tabs for 'Deployment Pools' (selected), 'New', and 'Security'. A red box highlights the 'Security' tab. To the right of the tabs is a search bar labeled 'Search deployment pools'. Below these are two columns: 'Name' and 'Status'. A single row is visible, showing 'Fabrikam Fiber-Test group' and 'Fabrikam Fiber'. There's also a 'Team project references' link.

If you want to manage the permissions for a specific deployment group, then open the Security dialog for that group.

3. Add the user or group and choose the role you want them to have.

For example, here we add Raisa to the Administrator role.



The screenshot shows a modal dialog titled 'Manage default security roles for all deployment pools'. At the top left is an 'Add' button, which is highlighted with a red box and has a red arrow pointing to it from the left. The main area is titled 'Add user'. It has a 'User or group' field containing 'Raisa Pokrovskaya' with a delete icon, and a 'Role' dropdown set to 'Administrator'. Below the form is a note: 'Administrator can administer, manage, view and use deployment pools.' At the bottom are 'Add' and 'Close' buttons.

4. Click **Add**.

Related notes

[Default build and release permissions](#)

- [Default permissions and access](#)

- [Permissions and groups reference](#)

Build and release permissions and security roles

11/15/2018 • 9 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

To support security of your build and release operations, you can add users to a built-in security group, set individual permissions for a user or group, or add users to pre-defined roles. You manage security for the following objects from **Azure Pipelines** in the web portal, either from the user or admin context.

This topic provides a description of the permissions and roles used to secure operations. To learn how to set permissions or add a user or group to a role, see [Set build and release permissions](#).

For permissions, you grant or restrict permissions by setting the permission state to Allow or Deny, either for a security group or an individual user. For a role, you add a user or group to the role. To learn more about how permissions are set, including inheritance, see [About permissions and groups](#). To learn how inheritance is supported for role-based membership, see [About security roles](#).

Default permissions assigned to built-in security groups

Once you have been added as a team member, you are a member of the Contributors group. This allows you to define and manage builds and releases. The most common built-in groups include Readers, Contributors, and Project Administrators. These groups are assigned the default permissions as listed below.

TASK	STAKEHOLDERS	READERS	CONTRIBUTORS	BUILD ADMINS	PROJECT ADMINS	RELEASE ADMINS
View build and release pipelines	<input type="checkbox"/>					
Define builds with continuous integration			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Define releases and manage deployments			<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>
Approve releases	<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>
Azure Artifacts (5 users free)			<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>

Queue builds, edit build quality			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Manage build queues and build qualities				<input type="checkbox"/>	<input type="checkbox"/>	
Manage build retention policies, delete and destroy builds			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Administer build permissions				<input type="checkbox"/>	<input type="checkbox"/>	
Manage release permissions					<input type="checkbox"/>	<input type="checkbox"/>
Create and edit task groups			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Manage task group permissions				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Can view library items such as variable groups		<input type="checkbox"/>				
Use and manage library items such as variable groups				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Security of agents and library entities

You use pre-defined roles and manage membership in those roles to configure [security on agent pools](#). You can configure this in a hierarchical manner either for all pools, or for an individual pool.

Roles are also defined to help you configure security on shared [library entities](#) such as [variable groups](#) and [service connection](#). Membership of these roles can be configured hierarchically, as well as at either project level or individual entity level.

Build permissions

Permissions in Build follow a hierarchical model. Defaults for all the permissions can be set at the project level and can be overridden on an individual build pipeline.

To set the permissions at project level for all build pipelines in a project, choose **Security** from the action bar on

the main page of Builds hub.

To set or override the permissions for a specific build pipeline, choose **Security** from the context menu of the build pipeline.

The following permissions are defined in Build. All of these can be set at both the levels.

PERMISSION	DESCRIPTION
Administer build permissions	Can change any of the other permissions listed here.
Queue builds	Can queue new builds.
Delete build pipeline	Can delete build pipeline(s).
Delete builds	Can delete builds for a pipeline. Builds that are deleted are retained in the Deleted tab for a period of time before they are destroyed.
Destroy builds	Can delete builds from the Deleted tab.
Edit build pipeline	Can save any changes to a build pipeline, including configuration variables, triggers, repositories, and retention policy.
Edit build quality	Can add tags to a build.
Override check-in validation by build	Applies to TFVC gated check-in builds . This does not apply to PR builds.
Retain indefinitely	Can toggle the retain indefinitely flag on a build.
Stop builds	Can stop builds queued by other team members or by the system.
View build pipeline	Can view build pipeline(s).
View builds	Can view builds belonging to build pipeline(s).
Update build information	It is recommended to leave this alone. It's intended to enable service accounts, not team members.
Manage build qualities	<i>Only applies to XAML builds</i>
Manage build queue	<i>Only applies to XAML builds</i>

Default values for all of these permissions are set for team project collections and project groups. For example, **Project Collection Administrators**, **Project Administrators**, and **Build Administrators** are given all of the above permissions by default.

When it comes to security, there are different best practices and levels of permissiveness. While there's no one right way to handle permissions, we hope these examples help you empower your team to work securely with builds.

- By default, contributors in a project cannot create or edit build pipelines. To grant permissions to work on build pipelines, select *Contributors* and set the **Edit build pipeline** permission to *Allow*.

- In many cases you probably also want to set **Delete build pipeline** to *Allow*. Otherwise these team members can't delete even their own build pipelines.
- Without **Delete builds** permission, users cannot delete even their own completed builds. However, keep in mind that they can automatically delete old unneeded builds using [retention policies](#).
- We recommend that you do not grant these permissions directly to a person. A better practice is to add the person to the build administrator group or another group, and manage permissions on that group.

Release permissions

Permissions for releases follow a hierarchical model. Defaults for all the permissions can be set at the project level and can be overridden on an individual release pipeline. Some of the permissions can also be overridden on a specific stage within a pipeline. The hierarchical model helps you define default permissions for all definitions at one extreme, and to lock down the production stage for an application at the other extreme.

To set permissions at project level for all release definitions in a project, open the shortcut menu from the ▾ icon next to **All release pipelines** and choose **Security**.

To set or override the permissions for a specific release pipeline, open the shortcut menu from the ▾ icon next to that pipeline name. Then choose **Security** to open the **Permissions** dialog.

To specify security settings for individual stages in a release pipeline, open the **Permissions** dialog by choosing **Security** on the shortcut menu that opens from the ellipses (...) on a stage in the release pipeline editor.

The following permissions are defined for releases. The scope column explains whether the permission can be set at the project, release pipeline, or stage level.

PERMISSION	DESCRIPTION	SCOPES
Administer release permissions	Can change any of the other permissions listed here.	Project, Release pipeline, Stage
Create releases	Can create new releases.	Project, Release pipeline
Delete release pipeline	Can delete release pipeline(s).	Project, Release pipeline
Delete release stage	Can delete stage(s) in release pipeline(s).	Project, Release pipeline, Stage
Delete releases	Can delete releases for a pipeline.	Project, Release pipeline
Edit release pipeline	Can save any changes to a release pipeline, including configuration variables, triggers, artifacts, and retention policy as well as configuration within a stage of the release pipeline. To make changes to a specific stage in a release pipeline, the user also needs Edit release stage permission.	Project, Release pipeline

PERMISSION	DESCRIPTION	SCOPES
Edit release stage	Can edit stage(s) in release pipeline(s). To save the changes to the release pipeline, the user also needs Edit release pipeline permission. This permission also controls whether a user can edit the configuration inside the stage of a specific release instance. The user also needs Manage releases permission to save the modified release.	Project, Release pipeline, Stage
Manage deployments	Can initiate a direct deployment of a release to a stage. This permission is only for direct deployments that are manually initiated by selecting the Deploy or Redeploy actions in a release. If the condition on a stage is set to any type of automatic deployment, the system automatically initiates deployment without checking the permission of the user that created the release.	Project, Release pipeline, Stage
Manage release approvers	Can add or edit approvers for stage(s) in release pipeline(s). This permissions also controls whether a user can edit the approvers inside the stage of a specific release instance.	Project, Release pipeline, Stage
Manage releases	Can edit the configuration in releases. To edit the configuration of a specific stage in a release instance, the user also needs Edit release stage permission.	Project, Release pipeline
View release pipeline	Can view release pipeline(s).	Project, Release pipeline
View releases	Can view releases belonging to release pipeline(s).	Project, Release pipeline

Default values for all of these permissions are set for team project collections and project groups. For example, **Project Collection Administrators**, **Project Administrators**, and **Release Administrators** are given all of the above permissions by default. **Contributors** are given all permissions except **Administer release permissions**. **Readers**, by default, are denied all permissions except **View release pipeline** and **View releases**.

Task group permissions

Task group permissions follow a hierarchical model. Defaults for all the permissions can be set at the project level and can be overridden on an individual task group pipeline.

You use task groups to encapsulate a sequence of tasks already defined in a build or a release pipeline into a single reusable task. You [define and manage task groups](#) in the **Task groups** tab in **Azure Pipelines**.

PERMISSION	DESCRIPTION
Administer task group permissions	Can add and remove users or groups to task group security.

PERMISSION	DESCRIPTION
Delete task group	Can delete a task group.
Edit task group	Can create, modify, or delete a task group.

Library roles and permissions

Permissions for library artifacts, such as variable groups and secure files, are managed by roles. You use a variable group to store values that you want to make available across multiple build and release pipelines. You [define and manage variable groups](#) and [secure files](#) in the **Library** tab in **Azure Pipelines**.

ROLE	DESCRIPTION
Administrator	Can use and manage library items.
Reader	Can only read library items.
User	Can use library items, but not manage them.

Service connection security roles

You [add users to the following roles](#) from the project-level admin context, **Services** page. To create and manage these resources, see [Service connections for build and release](#).

ROLE	DESCRIPTION
User	Can use the endpoint when authoring build or release pipelines.
Administrator	Can manage membership of all other roles for the service connection as well as use the endpoint to author build or release pipelines. The system automatically adds the user that created the service connection to the Administrator role for that pool.

Deployment pool security roles

You [add users to the following roles](#) from the collection-level admin context, **Deployment Pools** page. To create and manage deployment pools, see [Deployment groups](#).

ROLE	DESCRIPTION
Reader	Can only view deployment pools.
Service Account	Can view agents, create sessions, and listen for jobs from the agent pool.
User	Can view and use the deployment pool for creating deployment groups.
Administrator	Can administer, manage, view and use deployment pools.

Related notes

- [Set build and release permissions](#)
- [Default permissions and access](#)
- [Permissions and groups reference](#)

Build multiple branches

11/19/2018 • 8 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

You can build every commit and pull request to your Git repository using Azure Pipelines or TFS. In this tutorial, we will discuss additional considerations when building multiple branches in your Git repository. You will learn how to:

- Set up a CI trigger for topic branches
- Automatically build a change in topic branch
- Exclude or include tasks for builds based on the branch being built
- Keep code quality high by building pull requests
- Use retention policies to clean up completed builds

Prerequisites

- You need a Git repository in Azure Pipelines, TFS, or GitHub with your app. If you do not have one, we recommend importing the [sample .NET Core app](#) into your Azure Pipelines or TFS project, or forking it into your GitHub repository. Note that you must use Azure Pipelines to build a GitHub repository. You cannot use TFS.
- You also need a working build for your repository.

Set up a CI trigger for a topic branch

A common workflow with Git is to create temporary branches from your master branch. These branches are called topic or feature branches and help you isolate your work. In this workflow, you create a branch for a particular feature or bug fix. Eventually, you merge the code back to the master branch and delete the topic branch.

- [YAML](#)
- [Designer](#)

Unless you specify a [trigger](#) in your YAML file, a change in any of the branches will trigger a build. Add the following snippet to your YAML file in the `master` branch. This will cause any changes to `master` and `features/*` branches to be automatically built.

```
trigger:  
- master  
- features/*
```

YAML builds are not yet available on TFS.

Automatically build a change in topic branch

You are now ready for CI for both the master branch and future feature branches that match the branch pattern. Every code change for the branch will use an automated build pipeline to ensure the quality of your code remains high.

Follow the steps below to edit a file and create a new topic branch.

1. Navigate to your code in Azure Repos, TFS, or GitHub.
2. Create a new branch for your code that starts with `features/`, e.g., `features/feature-123`.
3. Make a change to your code in the feature branch and commit the change.
4. Navigate to the **Pipelines** menu in Azure Pipelines or TFS and select **Builds**.
5. Select the build pipeline for this repo. You should now see a new build executing for the topic branch. This build was initiated by the trigger you created earlier. Wait for the build to finish.

Your typical development process includes developing code locally and periodically pushing to your remote topic branch. Each push you make results in a build pipeline executing in the background. The build pipeline helps you catch errors earlier and helps you to maintain a quality topic branch that can be safely merged to master. Practicing CI for your topic branches helps to minimize risk when merging back to master.

Exclude or include tasks for builds based on the branch being built

The master branch typically produces deployable artifacts such as binaries. You do not need to spend time creating and storing those artifacts for short-lived feature branches. You implement custom conditions in Azure Pipelines or TFS so that certain tasks only execute on your master branch during a build run. You can use a single build with multiple branches and skip or perform certain tasks based on conditions.

- [YAML](#)
- [Designer](#)

Edit the `azure-pipelines.yml` file in your `master` branch, locate a task in your YAML file, and add a condition to it. For example, the following snippet adds a condition to [publish artifacts](#) task.

```
- task: PublishBuildArtifacts@1
  condition: and(succeeded(), eq(variables['Build.SourceBranch'], 'refs/heads/master'))
```

YAML builds are not yet available on TFS.

Validate pull requests

Use policies to protect your branches by requiring successful builds before merging pull requests. You have options to always require a new successful build before merging changes to important branches such as the master branch. There are other branch policy settings to build less frequently. You can also require a certain number of code reviewers to help ensure your pull requests are high quality and don't result in broken builds for your branches.

GitHub repository

- [YAML](#)
- [Designer](#)

Unless you specify `pr` triggers in your YAML file, pull request builds are automatically enabled for all branches. You can specify the target branches for your pull request builds. For example, to run the build only for pull requests that target: `master` and `features/*`:

```
pr:  
- master  
- features/*
```

For more details, see [Triggers](#).

YAML builds are not yet available on TFS.

Azure Pipelines or TFS repository

1. Navigate to the **Code** hub in Azure Repos or TFS.
2. Choose your **repository** and Select **Branches**. Choose the **master branch**.
3. You will implement a branch policy to protect the master branch. Select the **ellipsis** to the right of your branch name and Select **Branch policies**.
4. Choose the checkbox for **Protect this branch**. There are several options for protecting the branch.
5. Under the **Build validation** menu choose **Add build policy**.
6. Choose the appropriate build pipeline.
7. Ensure **Trigger** is set to automatic and the **Policy requirement** is set to required.
8. Enter a descriptive **Display name** to describe the policy.
9. Select **Save** to create and enable the policy. Select **Save changes** at the top left of your screen.
10. To test the policy navigate to the **Pull request** menu in Azure Pipelines or TFS.
11. Select **New pull request**. Ensure your topic branch is set to merge into your master branch. Select **create**.
12. Your screen displays the **policy** being executed.
13. Select the **policy name** to examine the build. If the build succeeds your code will be merged to master. If the build fails the merge is blocked.

Once the work is completed in the topic branch and merged to master, you can delete your topic branch. You can then create additional feature or bug fix branches as necessary.

Use retention policies to clean up your completed builds

Retention policies allow you to control and automate the cleanup of your various builds. For shorter-lived branches like topic branches, you may want to retain less history to reduce clutter and storage costs. If you create CI builds on multiple related branches, it will become less important to keep builds for all of your branches.

1. Navigate to the **Pipelines** menu in Azure Pipelines or TFS.
2. Locate the build pipeline that you set up for your repo.
3. Select **Edit** at the top right of your screen.
4. Under the build pipeline name, Select the **Retention** tab. Select **Add** to add a new retention policy.

Policies

Policies are evaluated in order, applying the first matching policy to each build. The default rule at the bottom matches all builds.

+ Add

Keep for 1 day, 1 good build
+refs/heads/features/*

Keep for 10 days, 1 good build
+refs/heads/*

Keep for 30 days, 10 good builds
Default

Settings

Branch Filters

Type: Include Branch specification: features/*

+ Add

Days to keep: 1 Minimum to keep: 1

When cleaning up builds, delete the following:

Build record
 Source label
 File Share
 Symbols
 Automated test results ⓘ

5. Type **features/*** in the **Branch specification** dropdown. This ensures any feature branches matching the wildcard will use the policy.
6. Set **Days to keep** to 1 and **Minimum to keep** to 1.
7. Select the **Save & queue** menu and then Select **Save**.

Policies are evaluated in order, applying the first matching policy to each build. The default rule at the bottom matches all builds. The retention policy will clean up build resources each day. You retain at least one build at all times. You can also choose to keep any particular build for an indefinite amount of time.

Next steps

In this tutorial, you learned how to manage CI for multiple branches in your Git repositories using Azure Pipelines or TFS.

You learned how to:

- Set up a CI trigger for topic branches
- Automatically build a change in topic branch
- Exclude or include tasks for builds based on the branch being built
- Keep code quality high by building pull requests
- Use retention policies to clean up completed builds

Create a multi-platform pipeline

10/29/2018 • 3 minutes to read • [Edit Online](#)

Azure Pipelines

This is a step-by-step guide to using Azure Pipelines to build on macOS, Linux, and Windows.

Prerequisites

- You need an Azure DevOps organization. If you don't have one, you can [create one for free](#). If your team already has one, then make sure you're an administrator of the Azure DevOps project that you want to use. (An Azure DevOps organization is different from your GitHub organization. Give them the same name if you want alignment between them.)
- You need a GitHub account, where you can create a repository.

Get the sample code

You can use Azure Pipelines to build an app on written in any language, on multiple platforms at the same time.

1. Go to <https://github.com/MicrosoftDocs/pipelines-javascript>.
2. Fork the repo into your own GitHub account.

You should now have a sample app in your GitHub account.

Add additional platforms

In the sample repo, the Node.js app is configured to run on Ubuntu Linux. You're going to add a second job that runs on Windows.

1. Go to your fork of the sample code on GitHub.
2. Select `azure-pipelines.yml`, and then select the *Edit this file* pencil icon.
3. In GitHub's web editor, replace the existing content with the content below.

```

# Build NodeJS Express app using Azure Pipelines
# https://docs.microsoft.com/azure/devops/pipelines/languages/javascript?view=vsts
jobs:
- job: Linux

    pool:
        vmImage: 'ubuntu 16.04'

    steps:
    - task: NodeTool@0
        inputs:
            versionSpec: '8.x'

    - script: |
        npm install
        npm test

    - task: PublishTestResults@2
        inputs:
            testResultsFiles: '**/TEST-RESULTS.xml'
            testRunTitle: 'Test results for JavaScript'

    - task: PublishCodeCoverageResults@1
        inputs:
            codeCoverageTool: Cobertura
            summaryFileLocation: '$(System.DefaultWorkingDirectory)/**/*coverage.xml'
            reportDirectory: '$(System.DefaultWorkingDirectory)/**/coverage'

    - task: ArchiveFiles@2
        inputs:
            rootFolderOrFile: '$(System.DefaultWorkingDirectory)'
            includeRootFolder: false

    - task: PublishBuildArtifacts@1

- job: Windows

    pool:
        vmImage: 'vs2017-win2016'

    steps:
    - task: NodeTool@0
        inputs:
            versionSpec: '8.x'

    - script: |
        npm install
        npm test

    - task: PublishTestResults@2
        inputs:
            testResultsFiles: '**/TEST-RESULTS.xml'
            testRunTitle: 'Test results for JavaScript'

    - task: PublishCodeCoverageResults@1
        inputs:
            codeCoverageTool: Cobertura
            summaryFileLocation: '$(System.DefaultWorkingDirectory)/**/*coverage.xml'
            reportDirectory: '$(System.DefaultWorkingDirectory)/**/coverage'

    - task: ArchiveFiles@2
        inputs:
            rootFolderOrFile: '$(System.DefaultWorkingDirectory)'
            includeRootFolder: false

    - task: PublishBuildArtifacts@1

```

At the bottom of the GitHub editor, select **Commit changes**.

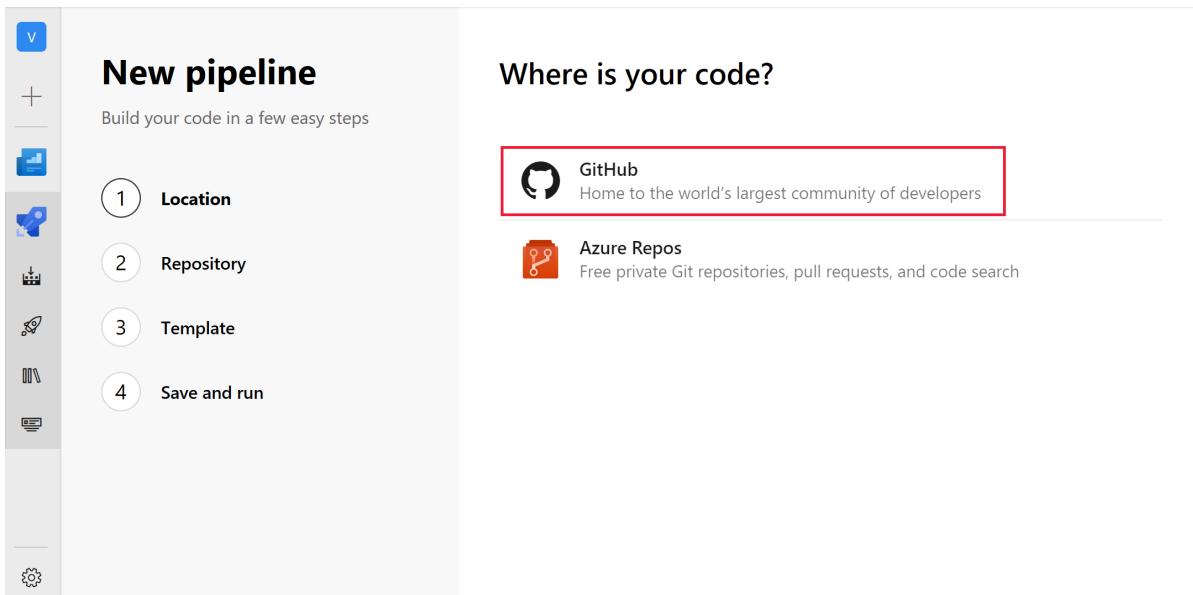
Each job in this example runs on a different VM image. By default, the Linux and Windows jobs run at the same time in parallel.

Note: To make this example clearer, we've copied the code. However, when you go to put a pipeline for your app into production, we recommend that you instead use [templates](#) to reduce code duplication. Also, `script` runs in each platform's native script interpreter: Bash on macOS and Linux, CMD on Windows. See [multi-platform scripts](#) to learn more.

Create the pipeline

Now that you've configured your GitHub repo with a pipeline, you're ready to build it.

1. Sign in to your Azure DevOps organization and navigate to your project.
2. In your project, go to the **Pipelines** page, and then select **New pipeline**.
3. Select **GitHub** as the location of your source code.



4. For **Repository**, select **Authorize** and then **Authorize with OAuth**.
5. You might be redirected to GitHub to sign in. If this happens, then enter your GitHub credentials. After you're redirected back to Azure Pipelines, select the **sample app** repository.
6. For the **Template**, Azure Pipelines analyzes the code in your repository. If your repository already contains an `azure-pipelines.yml` file (as in this case), then this step is skipped. Otherwise, Azure Pipelines recommends a starter template based on the code in your repository.
7. Azure Pipelines shows you the YAML file that it will use to create your pipeline.
8. Select **Save and run**, and then select the option to **Commit directly to the master branch**.
9. The YAML file is pushed to your GitHub repository, and a new build is automatically started. Wait for the build to finish.

Next steps

You've just learned the basics of using multiple platforms with Azure Pipelines. From here, you can learn more about:

- Running [multiple jobs](#)

- [Cross-platform scripting](#)
- [Templates](#) to remove the duplication
- Building [Nodejs](#) apps
- Building [.NET Core](#), [Go](#), [Java](#), or [Python](#) apps

For details about building GitHub repositories, see [Build GitHub repositories](#).

Build a repo using the visual designer

11/2/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

TIP

We recommend that you use YAML instead of the visual designer that is explained below. YAML allows you to use the same branching and code review practices for your pipeline as you would for your application code. See [Create your first pipeline](#).

Prerequisites

- You need an Azure DevOps organization. If you don't have one, you can [create one for free](#). If your team already has one, then make sure you're an administrator of the Azure DevOps project that you want to use. (An Azure DevOps organization is different from your GitHub organization. Give them the same name if you want alignment between them.)
- You need a GitHub account, where you can create a repository.

Get the sample code

Azure Pipelines can be used to build an app written in any language. For this guide, we will use a .NET Core sample app. Fork the following repository into your own GitHub account.

<https://github.com/MicrosoftDocs/pipelines-dotnet-core>

You should now have a sample app in your GitHub account.

Set up CI for your repository

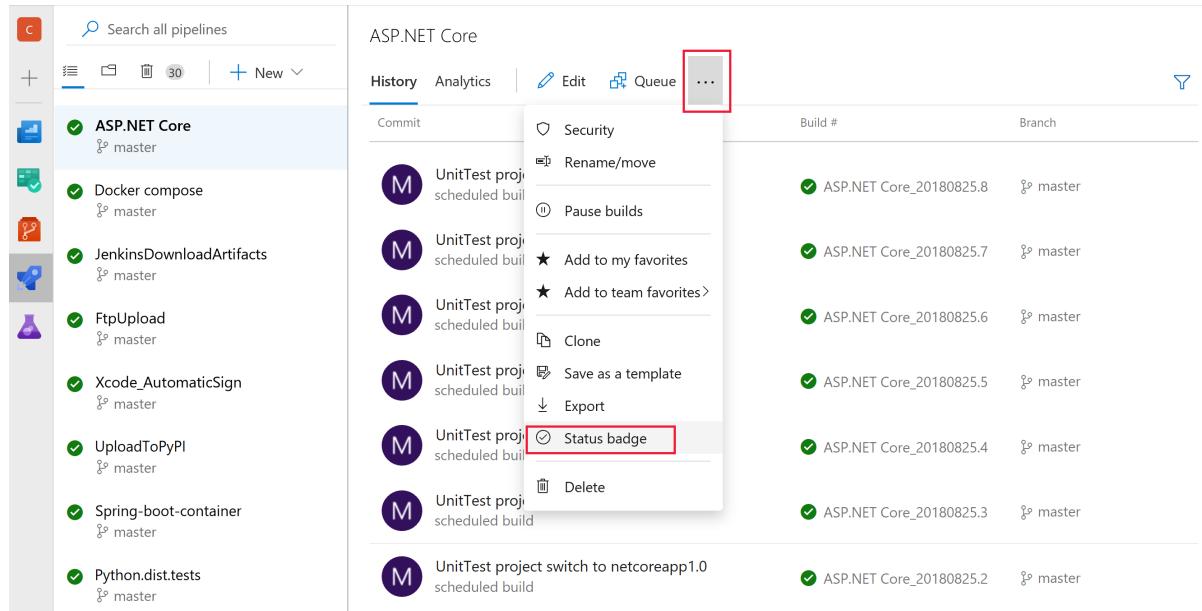
Follow the steps below to configure GitHub as a source for your Azure Pipelines build.

1. Login to your Azure DevOps organization and navigate to your project.
2. In your project, navigate to the **Pipelines** page, and then choose **New pipeline**.
3. Select **GitHub** for the type of repository.
4. Give your connection a name, and then select the **Authorize using OAuth** button. Optionally you can use a GitHub **personal access token** instead of OAuth.
5. When prompted, sign in to your **GitHub account**. Then select **Authorize** to grant access to your Azure DevOps organization. If you already are signed into GitHub in another browser tab, you may not see this step.
6. Choose the repository that contains the sample you forked earlier and select **Continue**.
7. Select the **ASP.NET Core** build template or a template that is appropriate for your application.
8. Choose **Hosted Ubuntu 1604** for Agent pool.
9. Select **Triggers**. Enable **Continuous integration** for your builds. Ensure you include the `master` branch under **Branch filters**. This setting ensures each commit to `master` in GitHub will trigger a build via a GitHub webhook.

10. Select **Save & queue** to save your build pipeline and create the first build.
11. Once the build completes, select the name of the pipeline in the build results page to navigate to the history of builds for that pipeline. Take a note of the `definitionId` in the URL. You will need this to set up the build badge in upcoming steps.

Get the status badge

1. In the Azure Pipelines page, navigate to the list of pipelines.
2. Select the pipeline that was created for you.
3. In the context menu for the pipeline, select **Status badge**.



4. Copy the sample markdown from the status badge panel.

Add a status badge to your repository

In GitHub:

1. Select the `Readme.md` file, and then choose **Edit**.
2. Copy the status badge Markdown that you copied in the previous section at the beginning of the `readme.md` file.
3. Commit the change to the master branch.
4. Notice that the status badge appears in the description of your repository.

In Azure Pipelines:

1. Observe that a new build is queued; its status could be either not started or running.

Q & A

How do I use a personal access token to authorize the Azure Pipelines to GitHub connection?

See this [article](#) for creating a GitHub personal access token. You can use the token in the Azure Pipelines **Get sources** task of your build or release pipelines by creating a GitHub [service connection](#) and entering the token.

Create an Azure service connection

10/9/2018 • 4 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

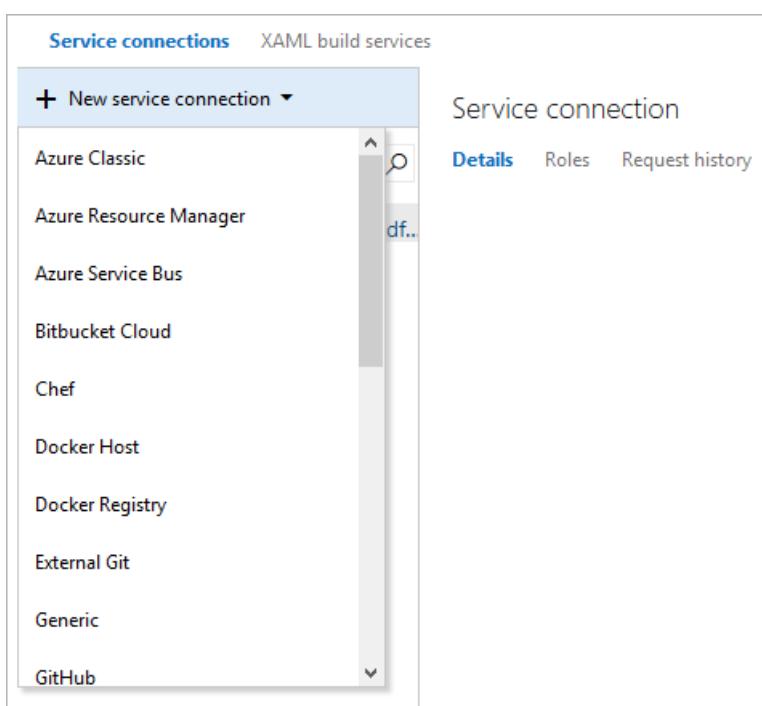
This topic explains how to create an Azure Resource Manager service connection for connecting to Microsoft Azure resources. It starts by showing the simple case where you select the subscription, and optionally the Azure Resource Group, to which you want to connect. Use this approach:

- If you are connecting from Azure Pipelines, and not from TFS.
- If you are the owner of both the Azure and the Azure DevOps subscriptions you are connecting from, and both accept the same credentials as you are currently signed into Azure Pipelines with.
- You do not need to further limit the permissions for Azure resources accessed through the service connection.
- You are not connecting to [Azure Stack](#) or an [Azure Government Cloud](#).

If you have problems using this simple approach (such as no subscriptions being shown in the drop-down list), or if you want to further limit users' permissions, you can do so by using a service principal as shown [here](#).

Create an Azure Resource Manager service connection

1. In Azure DevOps, open the **Service connections** page from the [project settings page](#). In TFS, open the **Services** page from the "settings" icon in the top menu bar.
2. Choose **+ New service connection** and select **Azure Resource Manager**.



3. Fill in the following parameters for the service connection.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure subscription.
Scope level	Select Subscription or Management Group. Management groups are containers that help you manage access, policy, and compliance across multiple subscriptions.
Subscription	If you selected Subscription for the scope, select an existing Azure subscription. If you don't see any Azure subscriptions or instances, see Troubleshoot Azure Resource Manager service connections .
Management Group	If you selected Management Group for the scope, select an existing Azure management group. See Create management groups .
Resource Group	Leave empty to allow users to access all resources defined within the subscription, or select a resource group to which you want to restrict the users' access (users will be able to access only the resources defined within that group).

4. After the new service connection is created:

- If you are using it in the UI, select the connection name you assigned in the **Azure subscription** setting of your pipeline.
- If you are using it in YAML, copy the connection name into your code as the **azureSubscription** value.

See also: [Troubleshoot Azure Resource Manager service connection](#).

Create an Azure Resource Manager service connection with an existing service principal

1. If you want to use a pre-defined set of access permissions, and you don't already have a suitable service principal defined, follow one of these tutorials to create a new service principal:
 - [Use the portal to create an Azure Active Directory application and service principal that can access resources](#)
 - [How to create and test Azure Service Principal using Azure CLI](#)
2. In Azure DevOps, open the **Service connections** page from the [project settings page](#). In TFS, open the **Services** page from the "settings" icon in the top menu bar.
3. Choose **+ New service connection** and select **Azure Resource Manager**.

The screenshot shows the 'Service connections' page in the Azure DevOps interface. On the left, there's a sidebar with a search bar and a list of connection types: Azure Classic, Azure Resource Manager, Azure Service Bus, Bitbucket Cloud, Chef, Docker Host, Docker Registry, External Git, Generic, and GitHub. The 'GitHub' option is highlighted with a blue selection bar at the bottom.

4. Switch from the simplified version of the dialog to the full version using the link in the dialog.

This screenshot shows the 'Add an Azure Resource Manager service connection' dialog. It includes fields for 'Connection name' (highlighted in yellow), 'Scope level' (set to 'Subscription'), 'Subscription' (set to 'Visual Studio Enterprise'), and 'Resource Group'. A note below states: 'Subscriptions listed are from Azure Cloud'. A descriptive text block explains that a new service principal will be created with 'Contributor' role. A red box highlights the link 'use the full version of the service connection dialog.' at the bottom.

5. Enter a user-friendly **Connection name** to use when referring to this service connection.
6. Select the **Environment** name (such as Azure Cloud, Azure Stack, or an Azure Government Cloud).
7. If you *do not* select **Azure Cloud**, enter the Environment URL. For Azure Stack, this will be something like `https://management.local.azurestack.external`
8. Select the **Scope level** you require:
 - If you choose **Subscription**, select an existing Azure subscription. If you don't see any Azure subscriptions or instances, see [Troubleshoot Azure Resource Manager service connections](#).

- If you choose **Management Group**, select an existing Azure management group. See [Create management groups.](#) |
9. Download and run [this PowerShell script](#) in an Azure PowerShell window. When prompted, enter your subscription name, password, role (optional), and the type of cloud such as Azure Cloud (the default), Azure Stack, or an Azure Government Cloud.
 10. Copy these fields from the output of the PowerShell script into the Azure subscription dialog textboxes:
 - Subscription ID
 - Subscription Name
 - Service Principal ID
 - Service Principal Key
 - Tenant ID
 11. Choose **Verify connection** to ensure the information you entered is valid, then choose **OK**.
 12. After the new service connection is created:
 - If you are using it in the UI, select the connection name you assigned in the **Azure subscription** setting of your pipeline.
 - If you are using it in YAML, copy the connection name into your code as the **azureSubscription** value.
 13. If required, modify the service principal to expose the appropriate permissions. For more details, see [Use Role-Based Access Control to manage access to your Azure subscription resources](#). This blog post also contains more information about using service principal authentication.

See also: [Troubleshoot Azure Resource Manager service connections](#).

Connect to an Azure Government Cloud

For information about connecting to an Azure Government Cloud, see:

- [Connecting from Azure Pipelines \(Azure Government Cloud\)](#)

Connect to Azure Stack

For information about connecting to Azure Stack, see:

- [Connect to Azure Stack](#)
- [Connect Azure Stack to Azure using VPN](#)
- [Connect Azure Stack to Azure using ExpressRoute](#)

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Deploy a web app to Azure App Services

10/9/2018 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017.2

We'll show you how to set up continuous deployment of your ASP.NET or Node.js app to an Azure Web App using Azure Pipelines or Team Foundation Server (TFS). You can use the steps in this quickstart as long as your continuous integration pipeline publishes a Web Deploy package.

Prerequisites

Before you begin, you'll need a CI build that publishes your Web Deploy package. To set up CI for your specific type of app, see:

- [Build your ASP.NET 4 app](#)
- [Build your ASP.NET Core app](#)
- [Build your Node.js app with gulp](#)

You'll also need an Azure Web App where you will deploy the app.

Define your CD release pipeline

Your CD release pipeline picks up the artifacts published by your CI build and then deploys them to your Azure web site.

1. Do one of the following to start creating a release pipeline:
 - If you've just completed a CI build (see above), choose the link (for example, *Build 20170815.1*) to open the build summary. Then choose **Release** to start a new release pipeline that's automatically linked to the build pipeline.
 - Open the **Releases** tab in **Azure Pipelines**, open the + drop-down in the list of release pipelines, and choose **Create release pipeline**.
2. The easiest way to create a release pipeline is to use a template. If you are deploying a Node.js app, select the **Deploy Node.js App to Azure App Service** template. Otherwise, select the **Azure App Service Deployment** template. Then choose **Apply**.

The only difference between these templates is that Node.js template configures the task to generate a **web.config** file containing a parameter that starts the **iisnode** service.

3. If you created your new release pipeline from a build summary, check that the build pipeline and artifact is shown in the **Artifacts** section on the **Pipeline** tab. If you created a new release pipeline from the **Releases** tab, choose the + **Add** link and select your build artifact.
4. Choose the **Continuous deployment** icon in the **Artifacts** section, check that the continuous deployment trigger is enabled, and add a filter to include the **master** branch.

Continuous deployment is not enabled by default when you create a new release pipeline from the **Releases** tab.

5. Open the **Tasks** tab and, with **Stage 1** selected, configure the task property variables as follows:

- **Azure Subscription:** Select a connection from the list under **Available Azure Service Connections** or create a more restricted permissions connection to your Azure subscription. If you are using Azure Pipelines and if you see an **Authorize** button next to the input, click on it to authorize Azure Pipelines to connect to your Azure subscription. If you are using TFS or if you do not see the desired Azure subscription in the list of subscriptions, see [Azure Resource Manager service connection](#) to manually set up the connection.
- **App Service Name:** Select the name of the web app from your subscription.

NOTE: Some settings for the tasks may have been automatically defined as [stage variables](#) when you created a release pipeline from a template. These settings cannot be modified in the task settings; instead you must select the parent stage item in order to edit these settings.

6. Save the release pipeline.

Create a release to deploy your app

You're now ready to create a release, which means to run the release pipeline with the artifacts produced by a specific build. This will result in deploying the build:

1. Choose **+ Release** and select **Create a release**.
2. In the **Create a new release** panel, check that the artifact version you want to use is selected and choose **Create**.
3. Choose the release link in the information bar message. For example: "Release **Release-1** has been created".
4. In the pipeline view, choose the status link in the stages of the pipeline to see the logs and agent output.
5. After the release is complete, navigate to your site running in Azure using the Web App URL
`http://{web_app_name}.azurewebsites.net`, and verify its contents.

Next step

- [Customize web app deployment](#)

Deploy to an Azure Kubernetes Service

11/19/2018 • 6 minutes to read • [Edit Online](#)

Azure Pipelines

We'll show you how to set up continuous deployment of your containerized application to an Azure Kubernetes Service (AKS) using Azure Pipelines.

After you commit and push a code change, it will be automatically built and deployed to the target Kubernetes cluster.

Example

If you want some sample code that works with this guidance, import (into Azure DevOps) or fork (into GitHub) this repo:

```
https://github.com/Microsoft/devops-project-samples/tree/master/dotnet/aspnetcore/kubernetes/Application
```

Define your CI build process

You'll need a continuous integration (CI) build process that publishes a container image to a container registry (for example: Azure Container Registry) and packages a Helm chart.

To set up a CI build process, see:

- [Build a Docker image](#) and [create a Helm chart](#).

Prerequisites

You'll need an Azure subscription. You can get one free through [Visual Studio Dev Essentials](#).

Create an AKS to host your app

1. Sign into Azure at <https://portal.azure.com>.
2. In the Azure Portal, choose **Create a resource**, **New**, **Containers**, then choose **Kubernetes Service**.
3. Select or create a new Resource Group, enter name for your new Kubernetes Service cluster and DNS name prefix.
4. Choose **Review + Create** and then, after validation, choose **Create**.
5. Wait until the new AKS cluster has been created.

Configure authentication

When you use Azure Container Registry (ACR) with Azure Kubernetes Service (AKS), you must establish an authentication mechanism. This can be achieved in two ways:

1. Grant AKS access to ACR. See [Authenticate with Azure Container Registry from Azure Kubernetes Service](#).
2. Use a [Kubernetes image pull secret](#). An image pull secret can be created by using the [Kubernetes deployment task](#).

Create a release pipeline

The build pipeline used to set up CI has already built a Docker image and pushed it to an Azure Container Registry. It also packaged and published a Helm chart as an artifact. In the release pipeline we'll deploy the container image as a Helm application to the AKS cluster.

1. In **Azure Pipelines**, or the **Build & Release** hub in TFS, open the summary for your build.

2. In the build summary, choose the **Release** icon to start a new release pipeline.

If you have previously created a release pipeline that uses these build artifacts, you will be prompted to create a new release instead. In that case, go to the **Releases** page and start a new release pipeline from there by choosing the + icon.

3. Select the **Empty job** template.

4. Open the **Tasks** page and select **Agent job**.

5. Choose + to add a new task and add a **Helm tool installer** task. This ensures the agent that runs the subsequent tasks has Helm and Kubectl installed on it.

6. Choose + again and add a **Package and deploy Helm charts** task. Configure the settings for this task as follows:

- **Connection Type:** Select **Azure Resource Manager** to connect to an AKS cluster by using an Azure service connection. If, alternatively, you want to connect to any Kubernetes cluster by using kubeconfig or a service account, you can select **Kubernetes Service Connection**. In this case, you will need to create and select a Kubernetes service connection instead of an Azure subscription for the following setting.
- **Azure subscription:** Select a connection from the list under **Available Azure Service Connections** or create a more restricted permissions connection to your Azure subscription. If you see an **Authorize** button next to the input, use it to authorize the connection to your Azure subscription. If you do not see the required Azure subscription in the list of subscriptions, see [Create an Azure service connection](#) to manually set up the connection.
- **Resource group:** Enter or select the resource group containing your AKS cluster.
- **Kubernetes cluster:** Enter or select the AKS cluster you created.
- **Command:** Select **init** as the Helm command. This will install Tiller to your running Kubernetes cluster. It will also set up any necessary local configuration. Tick **Use canary image version** to install the latest pre-release version of Tiller. You could also choose to upgrade Tiller if it is pre-installed by ticking **Upgrade Tiller**. If these options are enabled, the task will run

```
helm init --canary-image --upgrade
```

7. Choose + in the **Agent job** and add another **Package and deploy Helm charts** task. Configure the settings for this task as follows:

- **Kubernetes cluster:** Enter or select the AKS cluster you created.
- **Namespace:** Enter your Kubernetes cluster namespace where you want to deploy your application. Kubernetes supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called *namespaces*. You can use namespaces to create different environments such as dev, test, and staging in the same cluster.
- **Command:** Select **upgrade** as the Helm command. You can run any Helm command using this task and pass in command options as arguments. When you select the **upgrade**, the task shows some additional fields:

- **Chart Type:** Select **File Path**. Alternatively, you can specify **Chart Name** if you want to specify a URL or a chart name. For example, if the chart name is `stable/mysql`, the task will execute `helm upgrade stable/mysql`
 - **Chart Path:** This can be a path to a packaged chart or a path to an unpacked chart directory. In this example you are publishing the chart using a CI build, so select the file package using file picker or enter `$(System.DefaultWorkingDirectory)/**/*.tgz`
 - **Release Name:** Enter a name for your release; for example `azuredevops`
 - **Recreate Pods:** Tick this checkbox if there is a configuration change during the release and you want to replace a running pod with the new configuration.
 - **Reset Values:** Tick this checkbox if you want the values built into the chart to override all values provided by the task.
 - **Force:** Tick this checkbox if, should conflicts occur, you want to upgrade and rollback to delete, recreate the resource, and reinstall the full release. This is useful in scenarios where applying patches can fail (for example, for services because the cluster IP address is immutable).
 - **Arguments:** Enter the Helm command arguments and their values; for this example
`--set image.repository=$(imageRepoName) --set image.tag=$(Build.BuildId)` See [this section](#) for a description of why we are using these arguments.
 - **Enable TLS:** Tick this checkbox to enable strong TLS-based connections between Helm and Tiller.
 - **CA certificate:** Specify a CA certificate to be uploaded and used to issue certificates for Tiller and Helm client.
 - **Certificate:** Specify the Tiller certificate or Helm client certificate
 - **Key:** Specify the Tiller Key or Helm client key
8. In the **Variables** page of the pipeline, add a variable named **imageRepoName** and set the value to the name of your Helm image repository. Typically, this is in the format `name.azurecr.io/coderepository`
9. Save the release pipeline.

Arguments used in the Helm upgrade task

In the build pipeline, the container image is tagged with `$(Build.BuildId)` and this is pushed to an Azure Container Registry. In a Helm chart you can parameterize the container image details such as the name and tag because the same chart can be used to deploy to different environments. These values can also be specified in the **values.yaml** file or be overridden by a user-supplied values file, which can in turn be overridden by `--set` parameters during the Helm install or upgrade.

In this example, we pass the following arguments:

```
--set image.repository=$(imageRepoName) --set image.tag=$(Build.BuildId)
```

The value of `$(imageRepoName)` was set in the **Variables** page (or the **variables** section of your YAML file). Alternatively, you can directly replace it with your image repository name in the `--set` arguments value or **values.yaml** file. For example:

```
image:
  repository: VALUE_TO_BE_OVERRIDDEN
  tag: latest
```

Another alternative is to set the **Set Values** option of the task to specify the argument values as comma separated key-value pairs.

Create a release to deploy your app

You're now ready to create a release, which means to start the process of running the release pipeline with the artifacts produced by a specific build. This will result in deploying the build:

1. Choose **+ Release** and select **Create a release**.
2. In the **Create a new release** panel, check that the artifact version you want to use is selected and choose **Create**.
3. Choose the release link in the information bar message. For example: "Release **Release-1** has been created".
4. In the pipeline view, choose the status link in the stages of the pipeline to see the logs and agent output.

Next steps

- [Set up multi-stage release](#)

Define your multi-stage continuous deployment (CD) pipeline

10/15/2018 • 8 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Azure Pipelines and Team Foundation Server (TFS) provide a highly configurable and manageable pipeline for releases to multiple stages such as development, staging, QA, and production stages; including requiring approvals at specific stages.

In this tutorial, you learn about:

- Configuring triggers within the release pipeline
- Extending a release pipeline by adding stages
- Configuring the stages as a multi-stage release pipeline
- Adding approvals to your release pipeline
- Creating a release and monitoring the deployment to each stage

Prerequisites

You'll need:

- A release pipeline that contains at least one stage. If you don't already have one, you can create it by working through any of the following quickstarts and tutorials:
 - [Deploy to an Azure Web App](#)
 - [Azure DevOps Project](#)
 - [Deploy to IIS web server on Windows](#)
- Two separate targets where you will deploy the app. These could be virtual machines, web servers, on-premises physical deployment groups, or other types of deployment target. In this example, we are using Azure App Services website instances. If you decide to do the same, you will have to choose names that are unique, but it's a good idea to include "QA" in the name of one, and "Production" in the name of the other so that you can easily identify them. Use the Azure portal to create a new web app.

Configure the triggers in your release pipeline

In this section, you will check that the triggers you need for continuous deployment are configured in your release pipeline.

1. In **Azure Pipelines**, open the **Releases** tab. Select your release pipeline and, in the right pane, choose **Edit**.

The screenshot shows the 'SampleApp - 1' release pipeline in the 'Releases' tab. The 'Edit' button at the top left is highlighted with a red box.

2. Choose the **Continuous deployment trigger** icon in the **Artifacts** section to open the trigger panel. Make sure this is enabled so that a new release is created after every new successful build is completed.

The screenshot shows the 'Pipeline' tab of the 'SampleApp - 1' pipeline. In the 'Artifacts' section, the 'Continuous deployment trigger' icon (a blue gear with a lightning bolt) is highlighted with a red box. The trigger is enabled, and its details are shown on the right: it creates a release every time a new build is available, using the 'DotNetSample-ASP.NET Core-CI' build definition. The 'Build branch filters' section shows that the build branch is the pipeline's default branch, which is included by default.

For more information, see [Release triggers](#).

3. Choose the **Pre-deployment conditions** icon in the **Stages** section to open the conditions panel. Make sure that the trigger for deployment to this stage is set to **After release**. This means that a deployment will be initiated automatically when a new release is created from this release pipeline.

The screenshot shows the 'Stages' section of the pipeline. Stage 1 is selected and its details are shown on the right. Under 'Pre-deployment conditions', the 'Triggers' section is expanded, showing the 'Select trigger' dropdown with 'After release' selected. Below it, 'Artifact filters' and 'Schedule' are both disabled.

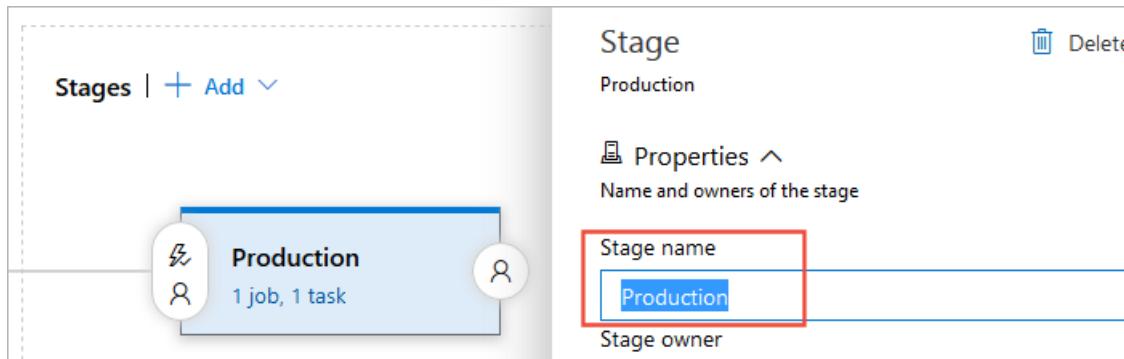
Notice that you can also define artifact filters that determine a condition for the release to proceed, and set up a schedule for deployments. You can use features to, for example, specify a branch from which the build artifacts must have been created, or a specific time of day when you know the app will not be heavily used.

For more information, see [Stage triggers](#).

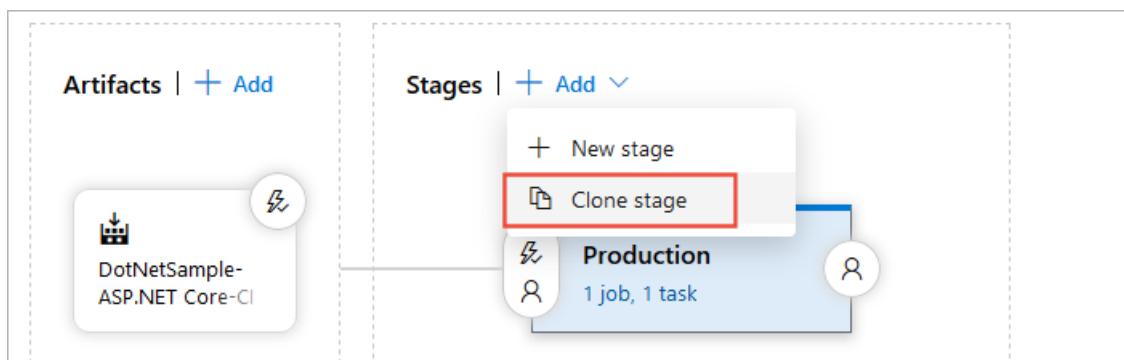
Extend a release pipeline by adding stages

In this section, you will add a new stage to the release pipeline. The two stages will deploy your app to the "QA" and the "Production" targets (in our example, two Azure App Services websites). This is a typical scenario where you deploy initially to a test or staging server, and then to a live or production server. Each **stage** represents one deployment target, though that target could be a physical or virtual server, a groups of servers, or any other legitimate physical or virtual deployment target.

1. In the **Pipeline** tab of your release pipeline, select the existing stage and rename it to **Production**.

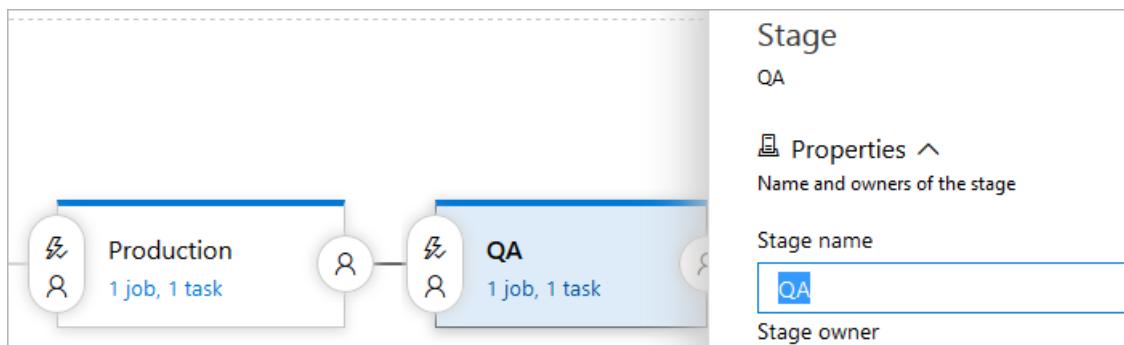


2. Open the **+ Add** drop-down list and choose **Clone stage** (the clone option is available only when an existing stage is selected).



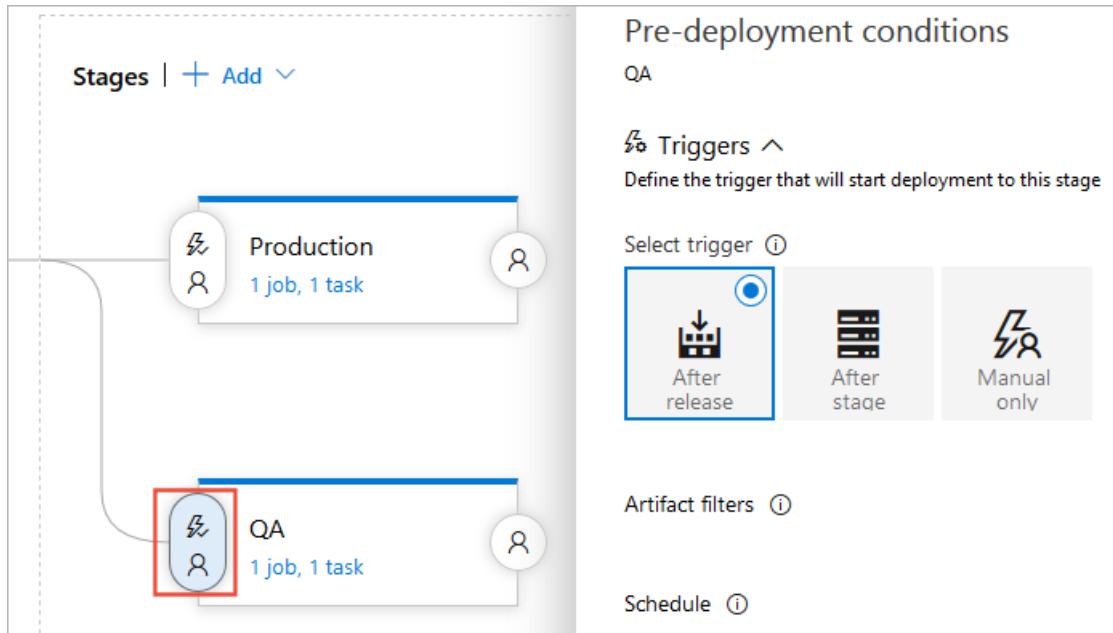
Typically, you want to use the same deployment methods with a test and a production stage so that you can be sure the deployed apps will behave in exactly the same way. Therefore, cloning an existing stage is a good way to ensure you have the same settings for both. Then you just need to change the deployment targets (the websites where each copy of the app will be deployed).

3. The clone of the stage appears after the existing stage in the pipeline, and has the name **Copy of Production**. Select this stage and, in the **Stages** panel, change the name to **QA**.

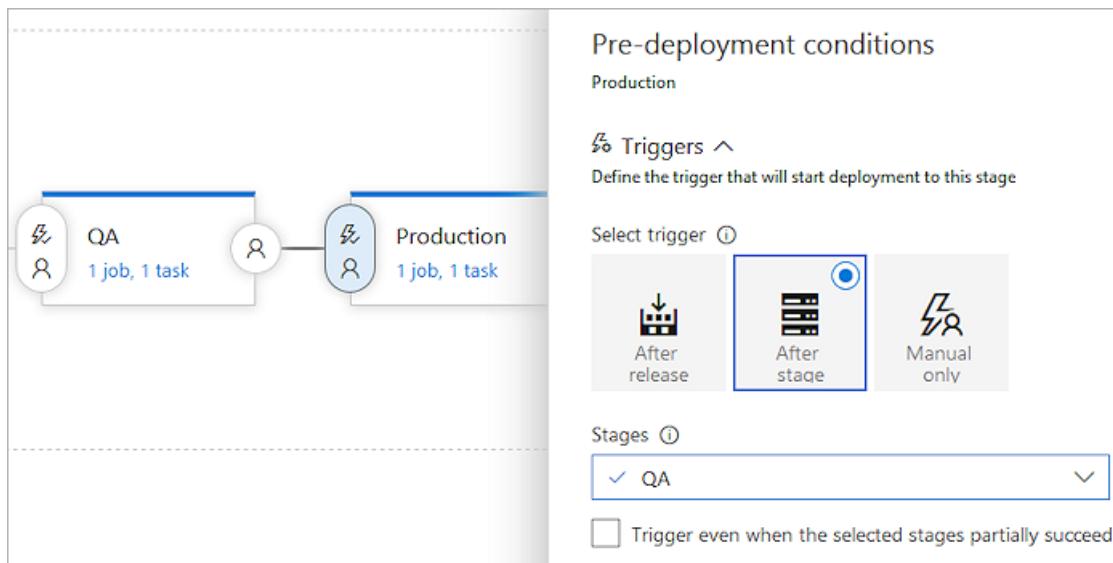


4. To reorganize the stages in the pipeline, choose the **Pre-deployment conditions** icon for the **QA** stage and set the trigger to **After release**. The pipeline diagram changes to show that the deployment to the

two stages will now execute in parallel.



5. Choose the **Pre-deployment conditions** icon for the **Production** stage and set the trigger to **After stage**, then select **QA** in the **Stages** drop-down list. The pipeline diagram changes to show that the deployment to the two stages will now execute in the required order.



Notice that you can specify deployment to start when a deployment to the previous stage is *partially* successful. Usually, this means the deployment tasks were set to continue the deployment even if a specific non-critical task failed (the default is that all tasks must succeed). You're most likely to set this option if you create a pipeline containing [fork and join deployments](#) that deploy to different stages in parallel.

6. Open the **Tasks** drop-down list and choose the **QA** stage. Recall that this stage is a clone of the original **Production** stage in the release pipeline. Therefore, currently, it will deploy the app to the same target as the **Production** stage.

All pipelines > SampleApp - 1

Pipeline	Tasks	Variables	Retention	Options	History
QA	Production				
	QA				
Run on agent	Run on agent				
 Deploy Azure App Service	Azure App Service Deploy				

Stage name: QA

Parameters: Azure subscription: Visual Studio Enterprise, App type: Web App, App service name: MySampleApp

- Depending on the tasks that you are using, change the settings so that this stage deploys to your "QA" target. In our example, using Azure App Services websites, we just need to select the **Deploy Azure App Service** task and select the "QA" website instead of the "Production" website.

All pipelines > SampleApp - 1

Pipeline	Tasks	Variables	Retention	Options	History
QA	Deployment process				
	QA				
Run on agent	Run on agent				
 Deploy Azure App Service	Azure App Service Deploy				

Stage name: QA

Parameters: Azure subscription: Visual Studio Enterprise, App type: Web App, App service name: MySampleApp-QAstage

If you are using a different type of task to deploy your app, the way you change the target for the deployment may differ. For example, if you are using deployment groups, you may be able to select a different deployment group, or a different set of tags within the same deployment group.

NOTE: Some settings for the tasks may have been automatically defined as **stage variables** when you created a release pipeline from a template. These settings cannot be modified in the task settings; instead you must select the parent stage item in order to edit these settings.

Add approvals within a release pipeline

The release pipeline you have modified deploys to test and then to production. If the deployment to test fails, the

trigger on the production stage does not fire, and so it is not deployed to production. However, it is typically the case that you want the deployment to pause after *successful* deployment to the test website so that you can verify the app is working correctly before you deploy to production. In this section, you will add an approval step to the release pipeline to achieve this.

1. Back in the **Pipeline** tab of the release pipeline, choose the **Pre-deployment conditions** icon in the **Stages** section to open the conditions panel. Scroll down to the **Pre-deployment approvers** section and enable pre-deployment approvers.

The screenshot shows the 'Pre-deployment conditions' panel for the 'Production' stage. On the left, there's a sidebar with a 'Production' section showing '1 job, 1 task'. The main panel has a heading 'Pre-deployment conditions' under 'Production'. It includes a 'Triggers' section and a 'Pre-deployment approvals' section. The 'Pre-deployment approvals' section is highlighted with a red box around its title and the 'Enabled' toggle switch. Below it, there's a note: 'Select the users who can approve or reject deployments to this stage'.

2. In the **Approvers** section, choose your user(s) from the list. You can type part of a name to search for matches. Also make sure you clear (untick) the checkbox **User requesting a release...** so that you can approve your own releases.

The screenshot shows the 'Pre-deployment conditions' panel for the 'Production' stage. The sidebar still shows '1 job, 1 task'. The main panel has the same structure as the previous screenshot, but the 'Approvers' section and the 'Approval policies' section are both highlighted with red boxes. The 'Approvers' section contains a search bar 'Search users and groups for approvers'. The 'Approval policies' section contains three checkboxes:

- The user requesting a release or deployment should not approve it
- Revalidate identity of approver before completing the approval.
- Skip approval if the same approver approved the previous stage

You can add as many approvers as you need, both individual users and organization groups. It's also possible to set up post-deployment approvals by choosing the icon at the right side of the stage item in the pipeline diagram. For more information, see [Approvals and gates overview](#).

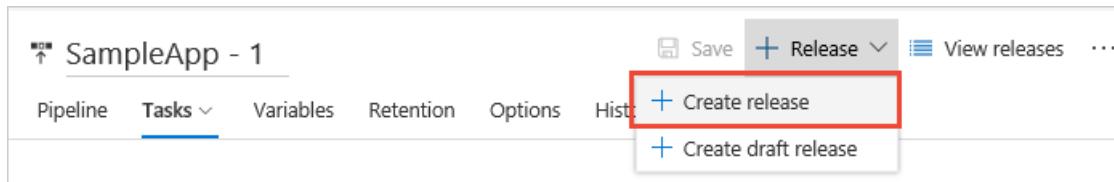
3. Save the modified release pipeline.

The screenshot shows the top navigation bar of the release pipeline interface. It includes tabs for 'Pipeline', 'Tasks', 'Variables', 'Retention', 'Options', and 'History'. A red box highlights the 'Save' button, which is located next to the '+ Release' and 'View releases' buttons.

Create a release

Now that you have completed the modifications to the release pipeline, it's time to start the deployment. To do this, you create a release from the release pipeline. A release may be created automatically; for example, the continuous deployment trigger is set in the release pipeline. This means that modifying the source code will start a new build and, from that, a new release. However, in this section you will create a new release manually.

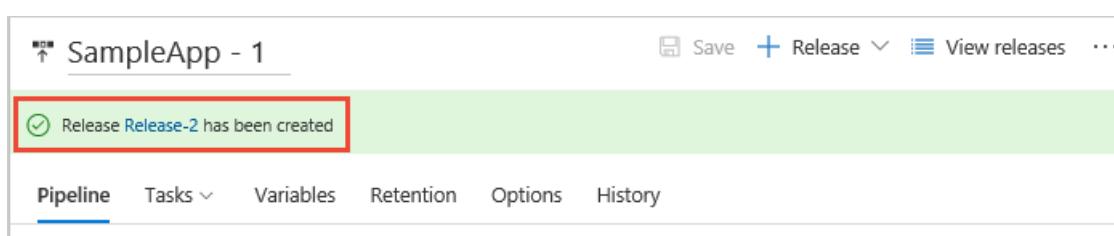
1. Open the **Release** drop-down list and choose **Create release**.



2. Enter a description for the release, check that the correct artifacts are selected, and then choose **Create**.

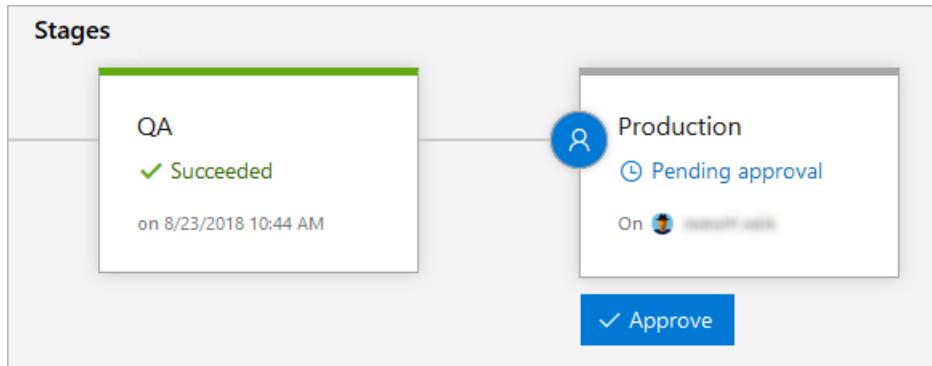
The screenshot shows the 'Create new release' dialog box for 'SampleApp - 1'.
Pipeline: QA → Production
Artifacts:
Source alias: SampleApp - 1 Version: 20170815.1
Release description: New manual release for Sample App 1
Buttons: Create, Cancel

3. After a few moments, a banner appears indicating that the new release was created. Choose the link (the name of the release).

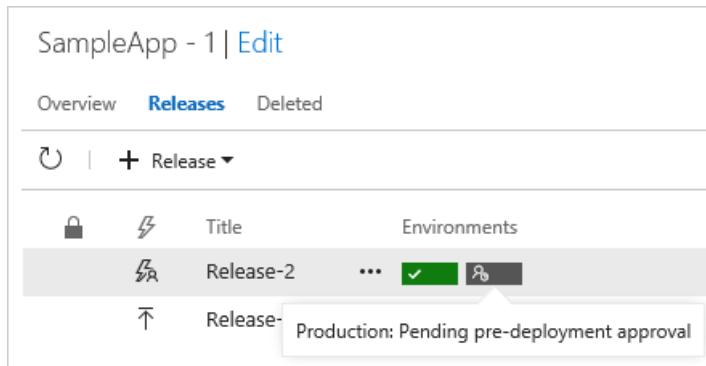


4. The release summary page opens showing details of the release. In the **Stages** section, you will see the

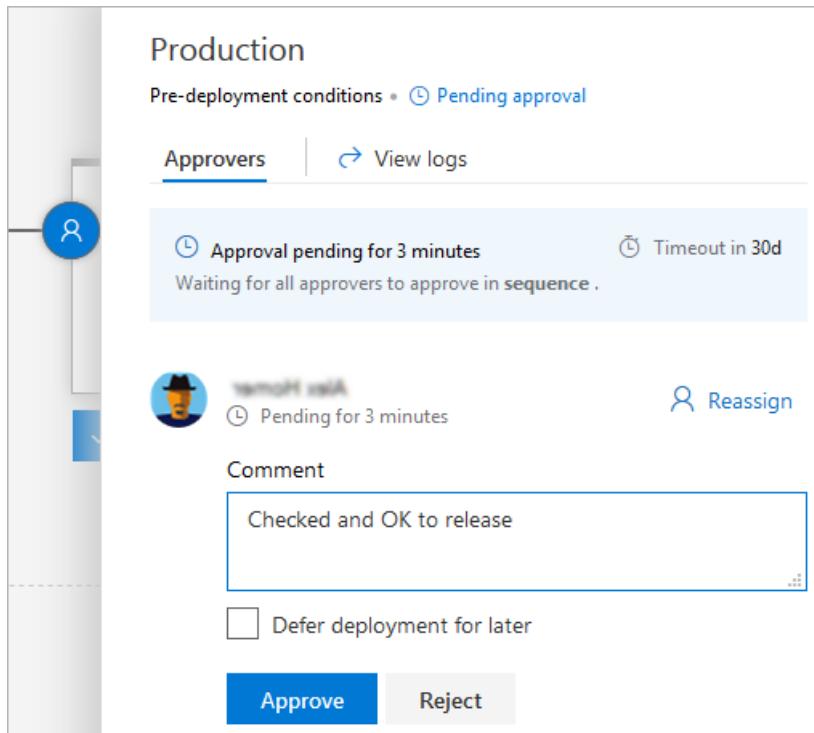
deployment status for the "QA" stage change to "Succeeded" and, at that point, an icon appears indicating that the release is now waiting for approval.



Other views, such as the list of releases, also display an icon that indicates approval is pending. The icon shows a pop-up containing the stage name and more details when you point to it. This makes it easy for an administrator to see which releases are awaiting approval, as well as the overall progress of all releases.



5. Choose the icon or link to open the approval dialog. Enter a brief note about the approval, and choose **Approve**.

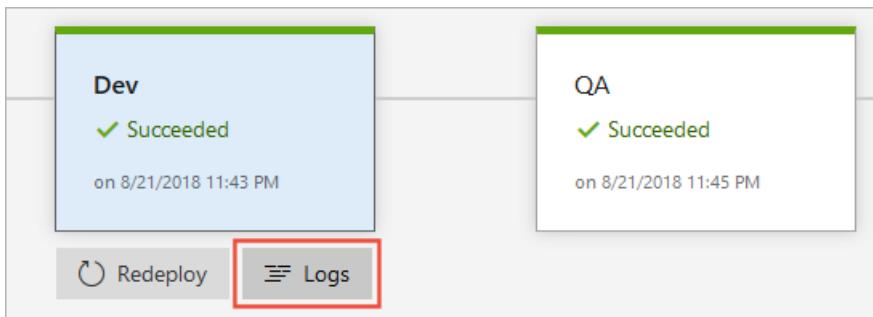


Notice that you can defer a deployment to a specific day and time; for example, a time when you expect the app to be only lightly loaded. You can also reassign the approval to another user. Release administrators can open *any* approval and over-ride it to accept or reject that deployment.

Monitor and track deployments

In this section, you will see how you can monitor and track deployments - in this example to two Azure App Services websites - from the release you created in the previous section.

1. In the release summary, hover over a stage and choose the **Logs** link that appears.



While the deployment is taking place, the logs page shows the live log from the agent. After the deployment is complete, links to the logs for each task step are displayed in the right pane.

2. Select any of the pipeline steps to show just the log file contents for that step. This makes it easier to trace and debug individual parts of the overall deployment. Alternatively, download the individual log files, or a zip of all the log files, from the icons and links in the page.

The screenshot shows the 'Logs' page for the 'Dev' stage of the 'PartsUnlimitedE2E' pipeline. The page header includes the pipeline name, release number, stage name, and status ('Succeeded'). Below the header, there are navigation links for Pipeline, Tasks, Variables, Logs (which is selected and underlined), Tests, and a set of global actions (Deploy, Refresh, Download all logs). The main content area is titled 'Dev' and shows the deployment process details: 'Pool: Hosted · Agent: Hosted Agent'. To the left is a sidebar showing the deployment process steps: 'Deployment process' (Succeeded) and 'Dev' (Succeeded). The main content area lists the task steps with their outcomes: Initialize Agent (succeeded), Initialize job (succeeded), Download artifact - PartsUnlimitedE2E (succeeded), Azure Deployment (succeeded), and Azure App Service Deploy (succeeded). A 'Download all logs' button is highlighted with a red box.

3. If you are having problems with a deployment, you can get more information from the log files by [running the release in debug mode](#).

Next step

[Use approvals and gates to control your deployment](#)

Use approvals and gates to control your deployment

10/15/2018 • 5 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

By using a combination of manual deployment approvals, gates, and manual intervention within a release pipeline in Azure Pipelines and Team Foundation Server (TFS), you can quickly and easily configure a release pipeline with all the control and auditing capabilities you require for your DevOps CI/CD processes.

In this tutorial, you learn about:

- Extending the approval process with gates
- Extending the approval process with manual intervention
- Viewing and monitoring approvals and gates

Prerequisites

This tutorial extends the tutorial [Define your multi-stage continuous deployment \(CD\) pipeline](#). **You must have completed that tutorial first.**

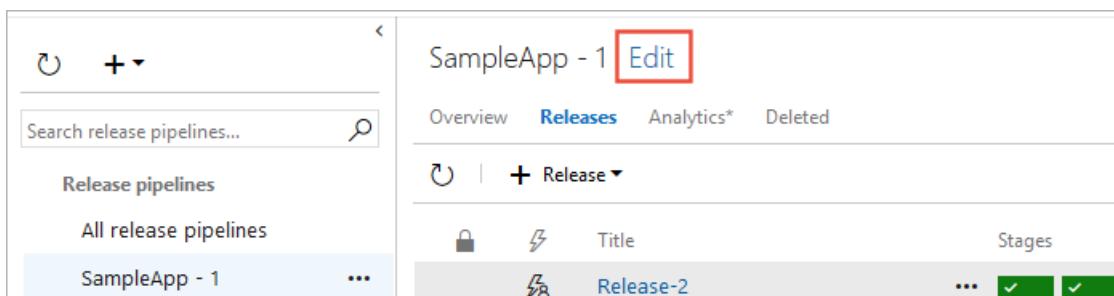
You'll also need a **work item query** that returns some work items from Azure Pipelines or TFS. This query is used in the gate you will configure. You can use one of the built-in queries, or create a new one just for this gate to use. For more information, see [Create managed queries with the query editor](#).

In the previous tutorial, you saw a simple use of manual approvals to allow an administrator to confirm that a release is ready to deploy to the production stage. In this tutorial, you'll see some additional and more powerful ways to configure approvals for releases and deployments by using manual intervention and gates. For more information about the ways you can configure approvals for a release, see [Approvals and gates overview](#).

Configure a gate

First, you will extend the approval process for the release by adding a gate. Gates allow you to configure automated calls to external services, where the results are used to approve or reject a deployment. You can use gates to ensure that the release meets a wide range of criteria, without requiring user intervention.

1. In the **Releases** tab of **Azure Pipelines**, select your release pipeline and choose **Edit** to open the pipeline editor.



The screenshot shows the Azure Pipelines web interface. On the left, there's a sidebar with a search bar labeled 'Search release pipelines...' and a list of 'Release pipelines' including 'All release pipelines' and 'SampleApp - 1'. The main area is titled 'SampleApp - 1' with an 'Edit' button highlighted by a red box. Below this, there are tabs for 'Overview', 'Releases' (which is selected), 'Analytics*', and 'Deleted'. The 'Releases' section shows a table with columns for 'Title' and 'Stages'. One row is visible for 'Release-2'. At the bottom right of the table, there are three green checkmark icons.

2. Choose the pre-deployment conditions icon for the **Production** stage to open the conditions panel. Enable gates by using the switch control in the **Gates** section.

Pre-deployment conditions

Production

Triggers ↴
Define the trigger that will start deployment to this stage

Pre-deployment approvals ↴
Select the users who can approve or reject deployments to this stage
Enabled

Gates ↴
Define gates to evaluate before the deployment. [Learn more](#)
Enabled

3. To allow gate functions to initialize and stabilize (it may take some time for them to begin returning accurate results), you configure a delay before the results are evaluated and used to determine if the deployment should be approved or rejected. For this example, so that you can see a result reasonably quickly, set the delay to a short period such as one minute.

Gates ↴
Define gates to evaluate before the deployment. [Learn more](#)
Enabled

The delay before evaluation ⓘ
1 Minutes

Deployment gates ⓘ + Add ↴

4. Choose **+ Add** and select the **Query Work Items** gate.

Pre-deployment approvals ↴
Select the users who can approve or reject deployments to this stage

Gates ↴
Define gates to evaluate before the deployment. [Learn more](#)

The delay before evaluation ⓘ
1

Deployment gates ⓘ + Add ↴

Query Work Items
Executes a work item query and checks for the number of items returned.

5. Configure the gate by selecting an existing work item query. You can use one of the built-in Azure Pipelines and TFS queries, or [create your own query](#). Depending on how many work items you expect it to return, set the maximum and minimum thresholds (run the query in the **Work** hub if you're not sure what to expect).

Deployment gates i

+ Add ▼

T **Query Work Items** Enabled Delete

Query Work Items i

Display name *

Query * i

Upper threshold * i

Advanced ^

Lower threshold * i

Evaluation options ▼

You'll need to open the **Advanced** section to see the **Maximum Threshold** setting. For more details about the gate arguments, see [Work Item Query task](#).

6. Open the **Options for all gates** section and specify the timeout and the sampling interval. For this example, choose short periods so that you can see the results reasonably quickly. The minimum values you can specify are 6 minutes timeout and 5 minutes sampling interval.

Advanced ^

Lower threshold * i

Evaluation options ▼

The time between re-evaluation of gates i Minutes ▼

Minimum duration for steady results after a successful gates evaluation i Minutes ▼

The timeout after which gates fail i Hours ▼

Gates and approvals i

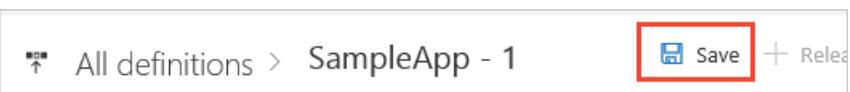
Before gates, ask for approvals

On successful gates, ask for approvals

Ignore gates outcome and ask for approvals

The sampling interval and timeout work together so that the gates will call their functions at suitable intervals, and reject the deployment if they don't all succeed during the same sampling interval and within the timeout period. For more details, see [Gates](#).

7. Save your release pipeline.

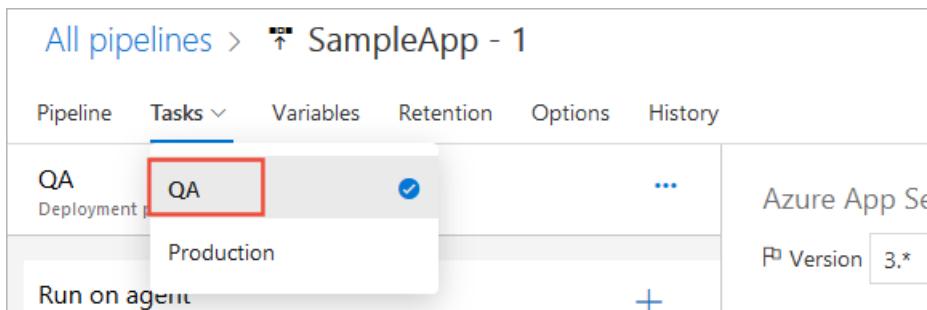


For more information about using other types of approval gates, see [Approvals and gates](#).

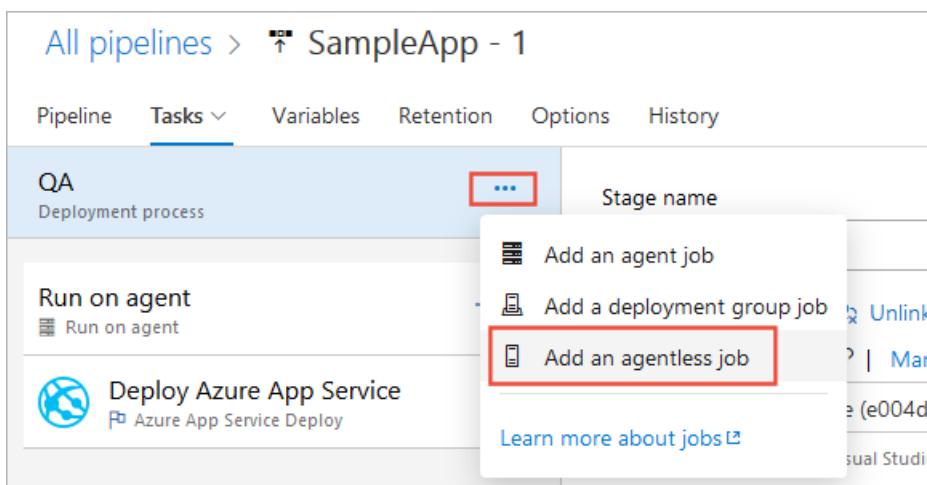
Configure a manual intervention

Sometimes, you may need to introduce manual intervention into a release pipeline. For example, there may be tasks that cannot be accomplished automatically such as confirming network conditions are appropriate, or that specific hardware or software is in place, before you approve a deployment. You can do this by using the **Manual Intervention** task in your pipeline.

1. In the release pipeline editor, open the **Tasks** editor for the **QA** stage.



2. Choose the ellipses (...) in the **QA** deployment pipeline bar and then choose **Add agentless job**.



Several tasks, including the **Manual Intervention** task, can be used only in an [agentless job](#).

3. Drag and drop the new agentless job to the start of the QA process, before the existing agent job. Then choose + in the **Agentless job** bar and add a **Manual Intervention** task to the job.

The screenshot shows the 'All pipelines > SampleApp - 1' pipeline configuration. The 'Tasks' tab is selected. On the left, a list of tasks includes 'QA Deployment process', 'Agentless job (Run on server)', 'Run on agent (Run on agent)', and 'Deploy Azure App Service (Azure App Service Deploy)'. A red box highlights the '+' button next to the 'Agentless job' task. On the right, a modal window titled 'Add tasks' lists several options under 'All': 'Delay', 'Invoke Azure Function', 'Invoke REST API', and 'Manual Intervention'. The 'Manual Intervention' option is highlighted with a blue box. A large blue 'Add' button is at the bottom right of the modal.

- Configure the task by entering a message (the **Instructions**) to display when it executes and pauses the release pipeline.

The screenshot shows the 'All pipelines > SampleApp - 1' pipeline configuration. The 'Tasks' tab is selected. The 'Manual Intervention' task is selected, indicated by a blue border and a checkmark icon. The task configuration pane on the right shows the following details:

- Version:** 8.*
- Display name:** Manual Intervention
- Instructions:** Ensure QA database updates have completed before continuing deployment
- Notify users:** Mateo Escobedo (selected), Search users and groups
- On timeout:** Reject (selected), Resume

Notice that you can specify a list of users who will receive a notification that the deployment is waiting for

manual approval. You can also specify a timeout and the action (approve or reject) that will occur if there is no user response within the timeout period. For more details, see [Manual Intervention task](#).

- Save the release pipeline and then start a new release.

The screenshot shows the 'SampleApp - 1' release definition in the 'All definitions' view. The 'Tasks' tab is selected. In the top right, there's a 'Release' dropdown menu with two options highlighted by a red box: '+ Create release' and '+ Create draft release'.

View the logs for approvals

You typically need to validate and audit a release and the associated deployments after it has completed, or even during the deployment pipeline. This is useful when debugging a problematic deployment, or when checking when and by whom approvals were granted. The comprehensive logging capabilities provide this information.

- Open the release summary for the release you just created. You can do this by choosing the link in the information bar in the release editor after you create the release, or directly from the **Releases** tab of [Azure Pipelines](#).

The screenshot shows the 'Releases' tab with two releases listed: 'Release-2' and 'Release-1'. A context menu is open over 'Release-2', with the 'Open in new tab' option highlighted by a red box. Other options in the menu include 'Open', 'Start', 'Retain indefinitely', 'Abandon', and 'Delete'.

- You'll see the live status for each step in the release pipeline. It indicates that a manual intervention is pending (this pre-deployment approval was configured in the previous tutorial [Define your multi-stage continuous deployment pipeline](#)). Choose the **Resume** link.

The screenshot shows the release pipeline stages. The 'QA' stage is currently active, displaying a 'Pending intervention' status. Below it, the 'Production' stage is shown as 'Not deployed'. At the bottom of the screen, there is a large blue 'Resume' button.

3. You see the intervention message, and can choose to resume or reject the deployment. Enter some text response to the intervention and choose **Resume**.

QA

Manual Intervention | [View logs](#)

⌚ Manual Intervention pending.

Instructions

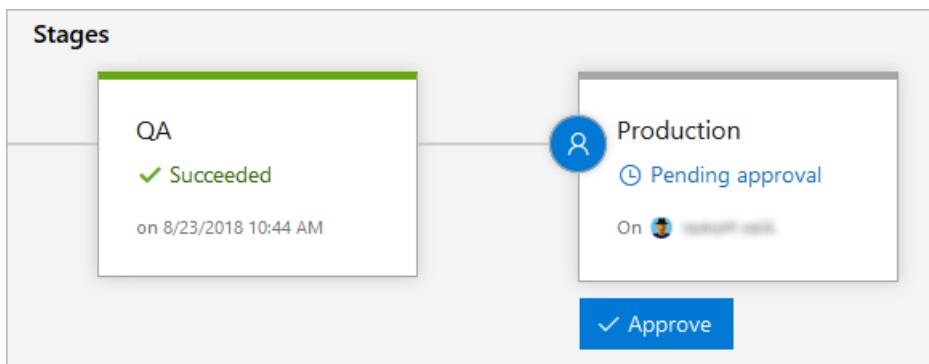
Ensure QA database updates have completed before continuing deployment

Comment

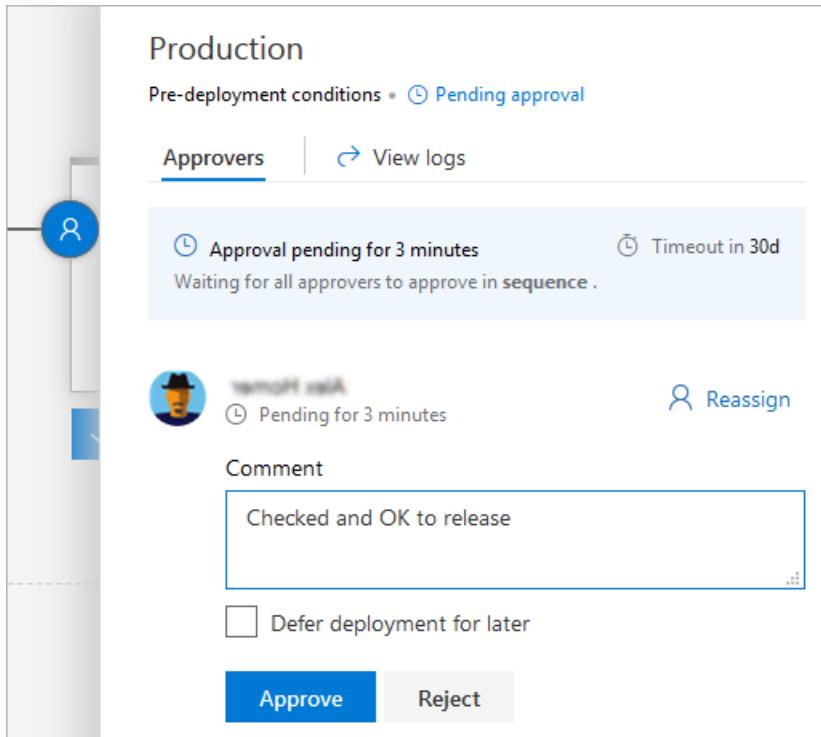
Databases checked, OK to deploy

Resume Reject

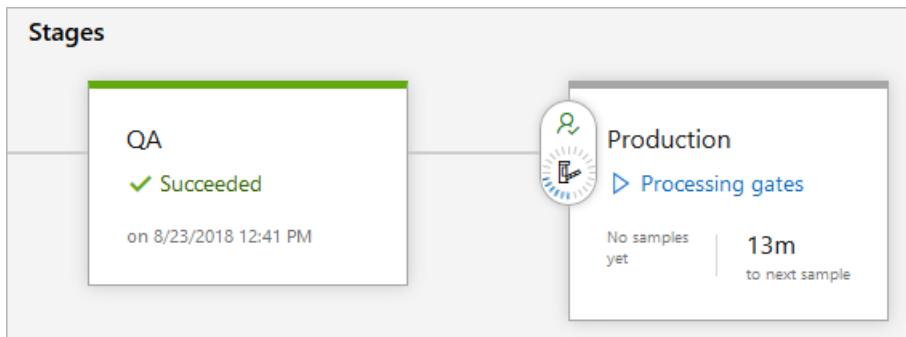
4. Go back to the pipeline view of the release. After deployment to the QA stage succeeds, you see the pre-deployment approval pending message for the **Production** environment.



5. Enter your approval message and choose **Approve** to continue the deployment.



6. Go back to the pipeline view of the release. Now you see that the gates are being processed before the release continues.



7. After the gate evaluation has successfully completed, the deployment occurs for the Production stage. Choose the **Production** stage icon in the release summary to see more details of the approvals and gate evaluations.

Altogether, by using a combination of manual approvals, approval gates, and the manual intervention task, you've seen how can configure a release pipeline with all the control and auditing capabilities you may require.

Next step

[Deploy to IIS web servers on Windows](#)

Tutorial: Integrate your Jenkins CI jobs with Azure Pipelines CD to Azure

11/2/2018 • 12 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Azure Pipelines provides integration with Jenkins so that you can use Jenkins for Continuous Integration (CI) while gaining several DevOps benefits from an Azure Pipelines release pipeline to Azure such as:

- Reusing your existing investments in Jenkins Build jobs
- Tracking work items and related code changes
- View the entire CI/CD pipeline from a central location
- Deploy various Azure services consistently via Azure Pipelines
- Enforce quality of builds to gate deployments
- Define work flows such as manual approval processes and CI triggers

In this tutorial, you use Jenkins to build a **Java web app** that uses Maven and Azure Pipelines or Team Foundation Server (TFS) to deploy to an **Azure App Service**.

You will:

- Get the Sample App
- Configure Jenkins credentials and the Azure Pipelines plugin
- Configure a Jenkins CI build with Azure Pipelines integration
- Create a Jenkins service connection and Service Hooks in Azure Pipelines
- Create an Azure Pipelines release pipeline for CD to Azure
- Test the CI/CD pipeline with a pull request
- (Optionally) Create an Azure Pipelines build pipeline to wrap the Jenkins CI job

Before you begin

- An Azure subscription. You can get one free through [Visual Studio Dev Essentials](#).
- A Tomcat and Java 8 based Azure Web App. You can follow the steps for creating one [here](#)
- Basic Jenkins and Git knowledge.
- You need access to a Jenkins server with Maven configured. If you have not yet created a Jenkins server, see [Create a Jenkins master on an Azure Virtual Machine](#).
- Sign in to your Azure DevOps organization (`https://dev.azure.com/{your-organization}`). You can get a [free Azure DevOps organization](#).

NOTE

For more information, see [Connect to Azure Pipelines](#).

Get the sample app

You need an app stored in a Git repository. You use this app to build and deploy. For this tutorial, we recommend you use [this sample Java app available from GitHub](#). This tutorial uses a Java and Maven sample application that is configured for deployment to Azure App Service. If you want to work with your own repository, you should configure a similar sample.

1. In Azure Pipelines, on the **Code** page for your Azure DevOps project, select the option to **Import repository**.
2. In the **Import a Git repository** dialog box, paste the following URL into the **Clone URL** text box.

```
https://github.com/Azure-Samples/app-service-maven
```

3. Click **Import** to copy the sample code into your Git repo.
4. Select **Clone** at the top right, and keep the **clone url** for future steps in this tutorial.

Configure Jenkins credentials and the Azure Pipelines plugin

You must configure credentials for connecting to Azure Pipelines, and a plug in for **VS Team Services Continuous Deployment** on your Jenkins server. When using credentials to connect to Azure Pipelines, it is a best practice to use a **personal access token (PAT)**. You also need to create a Jenkins credential to use in your Jenkins build jobs.

NOTE

Ensure the PAT you use for the following steps contains the **Release (read, write, execute and manage)**, **Code (read)**, **Build (read and execute) permissions in Azure Pipelines**.

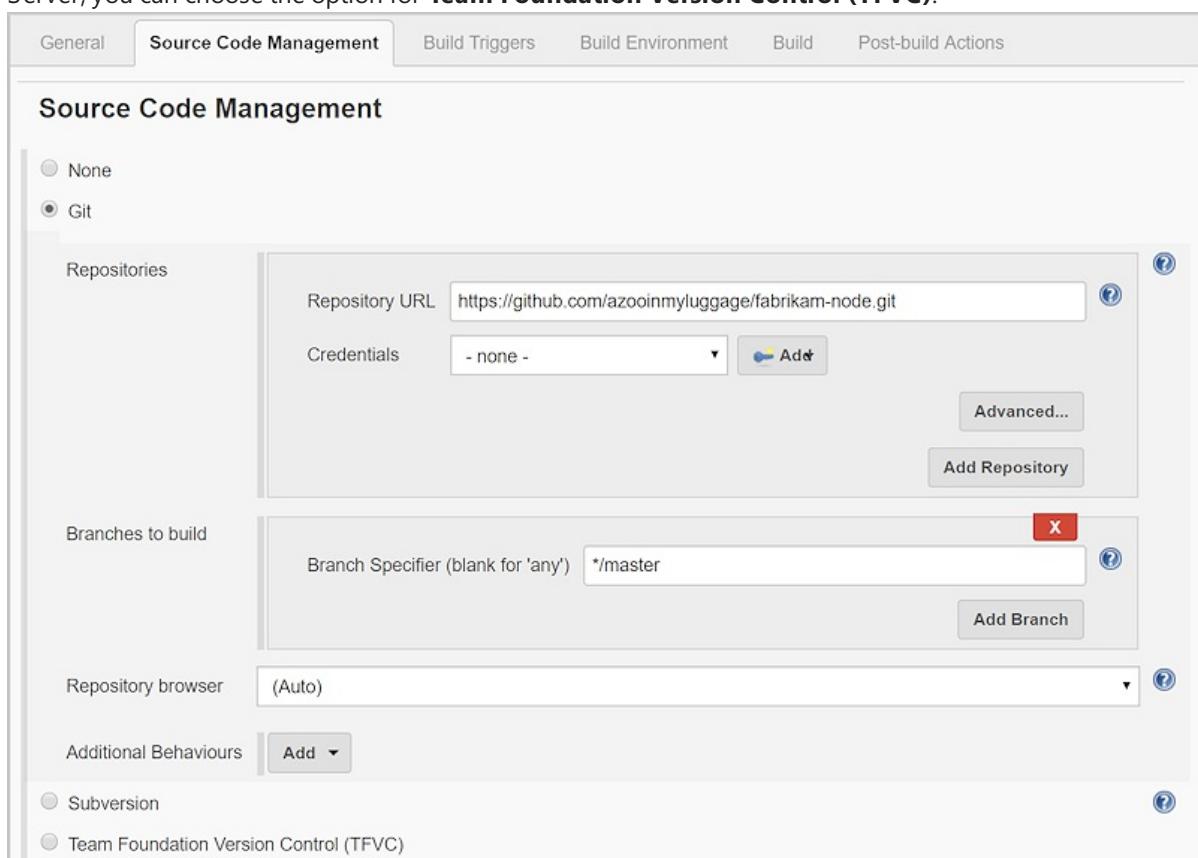
1. Create a PAT in your Azure DevOps account. Jenkins requires this information to access your Azure DevOps organization. Ensure you **store** the token information for upcoming steps in this section. Read [How do I create a personal access token for Azure Pipelines and TFS](#) to learn how to generate a PAT, or use an existing PAT if you have one.
2. Open your Jenkins account and select **Credentials, System**, and then choose **Add Domain**.
3. Enter a **name, description**, and then select **Add hostname**.
4. In the **Include** text box, enter `\dev.azure.com/*` to enable this credential for all Azure DevOps organizations. You can optionally enter your specific Azure DevOps organization.
5. Select **Ok**. You will see a message about empty credentials. Select the link for **adding some credentials**.
6. For **Kind**, choose **Username with password**. Enter the **Username** and the **PAT** you created earlier for **Password**. Select **Ok**.
7. **Navigate** back to the **Jenkins** home page.
8. Choose **Manage Jenkins**.
9. In the **Manage Jenkins** page, choose **Manage Plugins**.

10. Filter the **available list** to find the **VS Team Services Continuous Deployment** plug in and choose the **Install without restart** option. You can find additional information about the [Azure Pipelines plugin here](#).

Configure a Jenkins CI build with Azure Pipelines integration

You create a Jenkins build job to use the source code stored in your Azure Pipelines repository and execute a basic build. Ensure your Jenkins server has **Maven** installed and configured. You also create triggers to enable CI builds when code changes are pushed to the repository, and a CD trigger to initiate an Azure Pipelines release after the Jenkins build completes via a post build action.

1. Navigate to your Jenkins server. Click **New Item**. Enter an **item name**.
2. Choose **Freestyle project**. Select **OK**.
3. In the **Source Code Management** tab, select **Git** and enter the **clone url** you saved earlier for the Azure Pipelines **Repository URL** and the branch containing your app code. If you are using Team Foundation Server, you can choose the option for **Team Foundation Version Control (TFVC)**.



4. Select the **Credentials** drop down and choose the credential you created earlier. You should successfully authenticate to your Azure Pipelines repository and not receive errors before continuing. If you see errors, you likely have an issue with your credentials and Azure Pipelines **PAT**.
5. Select the **Build Triggers** tab, then **tick** the check boxes for **Build when a change is pushed to TFS/Team Services** and **Build when a change is pushed to a TFS pull request**. These two options rely on **Azure Pipelines Service Hooks** which you create in the next section.
6. Select the **Build** tab, **Add build step**, and then choose **Invoke top-level Maven targets**.
7. Select **Advanced...**. Enter **clean package** for **Goals**, and then enter **pom.xml** for **POM**.
8. Select the **Post-build Actions** tab. Choose **Add post-build action**, then select **Archive the artifacts**.
9. For **Files to archive** enter the following:

```
target/*.war
```

10. Select the **Post-build Actions** tab. Choose **Add post-build action**, then select **Trigger release in TFS/Team Services**.
11. Enter a **collection URL**. An example is `http://dev.azure.com/{your-organization}/DefaultCollection`
12. Enter the **project** and choose a **release pipeline** name. Store the **release pipeline** name since it is needed in later steps of this tutorial.
13. Enter the **username** and **PAT** you created earlier.
14. **Save** the Jenkins project.

Create a Jenkins service connection and Service Hooks in Azure Pipelines

You configure a Jenkins service connection to allow Azure Pipelines to connect to your Jenkins server. You must also configure two Jenkins service hooks so you can execute CI builds via automated triggers for both simple commits as well as pull requests to your Azure Repos Git repository.

1. Open the **Services** page in Azure Pipelines, open the **New service connection** list, and choose **Jenkins**.

The screenshot shows the Azure Pipelines Services page. At the top, there's a navigation bar with tabs: Fabrikam (selected), Home, Code, Work, Build & Release, Test, and a gear icon. Below the navigation bar, there are several tabs: Overview, Work, Security, Version Control, Policies, Agent queues, Notifications, Service Hooks, and Services (which is underlined). Under the Services tab, there are two sub-tabs: Endpoints and XAML Build Services. A large button labeled '+ New Service Endpoint' is visible. Below this, a list of service endpoints is shown: Generic, GitHub, Jenkins (which is highlighted with a red box), and Kubernetes. The Jenkins entry is also highlighted with a red box.

2. Enter a name for the connection.
3. Enter the URL of your Jenkins server, and if using **http** tick the **Accept untrusted SSL certificates** option.
An example URL is: `http://{YourJenkinsURL}.westcentralus.cloudapp.azure.com`
4. Enter the **user name and password** for your Jenkins account.
5. Choose **Verify connection** to check that the information is correct.
6. Choose **OK** to create the service connection.
7. From the **Service Hooks** page in Azure Pipelines, Select the + to add a new service and choose **Jenkins**. Select **next**.
8. Choose **Code pushed** for the type of event trigger. Select your **code repository** that contains the sample application you imported earlier. Select **Next**.
9. Enter the URL of your Jenkins server. An example is below.
`http://{YourJenkinsURL}.westcentralus.cloudapp.azure.com`
10. Enter the **user name and password** for your Jenkins account.

11. Select the Jenkins build you created earlier.
12. Choose **Test** to check that the information is correct.
13. Choose **Finish** to create the service connection.
14. Repeat the steps in this section to create another **service hook** for the **pull request merge attempted** trigger type. This will allow either simple commits to the repository as well as pull requests to both trigger the Jenkins build.

Create an Azure Pipelines release pipeline for CD to Azure

A release pipeline specifies the process Azure Pipelines executes to deploy the app. In this example, you deploy your app that originates from the Jenkins CI system. You deploy to an Azure App Service running Tomcat and Java.

To create the release pipeline in Azure Pipelines:

1. Open **Releases** in Azure Pipelines, and choose **Create release pipeline**.
2. Select the **Empty** template by choosing **Start with an empty pipeline**.
3. In the **Artifacts** section, click on **+ Add Artifact** and choose **Jenkins** for **Source type**. Select your Jenkins service connection. Then select the Jenkins source job and choose **Add**.
4. Next to the **1 job, 0 stages** link, select the **+** on the **Agent Job** to add a task to the job.
5. Search for the **Azure App Service Deploy** task. Select **Add** to add the task.
6. Choose your **Azure Subscription**. If you do not have an existing Azure connection in Azure Pipelines, you can follow the steps [here](#) to create one.
7. Enter a name for your existing Web App for the **App Service name**. This **name** must match your existing **Web App name** from the prerequisites.
8. For the **Package or folder** setting, select the ellipsis to browse, and then select your **.war artifact**.
9. Ensure the **name** for your release pipeline matches the same name you chose earlier during the **Create a Jenkins service connection and Service Hooks in Azure Pipelines** steps.
10. Click **Save**, and then click **OK** to save the release pipeline.

Test the CI/CD pipeline with a pull request

You can initiate the CI build and the subsequent CD deployment to Azure by completing a pull request into your master branch. The Jenkins build will initiate due to the service hook you set up earlier, and the Jenkins post build action will initiate an Azure Pipelines release which will deploy your app to the Azure App Service.

1. Navigate to **Code** in Azure Repos, then select your **repository**.
2. Select **New branch** to create a branch from the master branch. Enter a **name** for your new branch, and then select **Create branch**.
3. Select your new branch, then navigate to the **src/main/webapp/index.jsp** file.
4. Select **Edit**, then make a change to the **Title** for the **index.jsp page**, and then **Commit** your changes.
5. Select **Pull Requests**, then select **New Pull Request**. Select **Create** to issue a pull request from your branch to master.
6. Select **Approve**, then select **Complete**, then **Complete Merge**. This code change will initiate a CI build in Jenkins.

7. Navigate to your **Jenkins dashboard** and examine the build that is executing. Once it finishes, you can navigate to Azure Pipelines to watch the **Release Pipeline** execute. In a few moments your latest changes will be deployed to the **Azure App service**.

You are now using Jenkins CI builds with an Azure Repos code repository and an Azure Pipelines release pipeline to perform CI/CD to Azure App Service. You can easily track your code changes and deployments via the rich reporting capabilities of Azure Pipelines, and leverage Jenkins to execute CI builds.

(Optionally) Create an Azure Pipelines build pipeline to wrap the Jenkins CI job

There is an additional approach (pattern) possible when integrating Jenkins and Azure Pipelines CI/CD. You can optionally create an Azure Pipelines build pipeline to wrap the Jenkins build job, which will help you monitor the Jenkins job from Azure Pipelines. This pattern works slightly differently by using the built-in Azure Pipelines build capabilities to trigger the pipeline when new code is pushed to the repository. The approach for using both Azure Pipelines build and Jenkins together requires both Azure Pipelines build and Jenkins agents. To enable this pattern, you will modify a few items for the Azure Pipelines and Jenkins CI/CD pipeline that you created in earlier steps.

1. Navigate to the Jenkins build job you created earlier.
2. Select the **Build Triggers** tab, then remove the **tick** for the check boxes for **Build when a change is pushed to TFS/Team Services** and **Build when a change is pushed to a TFS pull request**. These two options are not needed for this pattern since the Azure Pipelines build pipeline has the ability to trigger the CI/CD pipeline when new code is pushed to the repo.
3. Navigate to your **Azure DevOps organization**.
4. Open **Builds** in Azure Pipelines, and choose **+ New** to create a new build pipeline.
5. Ensure **Azure Repos Git** is selected for the source, and then select **Continue**.
6. Search for the **Jenkins** template, and then select **Apply**.
7. Select **Pipeline**, and then select **Hosted VS 2017** for the **Agent pool**.
8. For the **Job name** parameter, enter the **Jenkins Job name** you created in the earlier steps of this tutorial. This name must match exactly. This will allow you to queue the Jenkins job and download the artifacts it produces.
9. Select the **Jenkins service connection**.
10. Select **Get sources**, and then check the **Don't sync sources** option. Since the Jenkins job is handling fetching the sources from Azure Pipelines, we do not need the Azure Pipelines build pipeline to perform this step. However, this step will configure a local working directory on the Azure Pipelines build agent.
11. Ensure the **Report build status** option is enabled so the build status will be reported on the code tab for the Azure Pipelines repository.
12. Select the **Jenkins** build tasks under **Job 1**, and then examine the additional options provided by the tasks. There is a task to queue the Jenkins job, download the artifacts produced by the job, and then finally to publish the artifacts which can later be used by the Azure Pipelines release pipeline.
13. Select the **Queue Jenkins Job** task. **Capture console output and wait for completion** is enabled to help provide the Jenkins logging information to Azure Pipelines, and also it will cause the Azure Pipelines build to wait for this step to complete successfully in Jenkins before proceeding. This causes the Azure Pipelines build to succeed or fail based on the Jenkins result. **Capture pipeline output and wait for completion** is similarly enabled to ensure Azure Pipelines succeeds or fails based on the entire Jenkins pipeline result. These two options help you view Jenkins logging information in a single place (Azure

Pipelines) for the entire CI/CD pipeline.

14. Open **Releases** in Azure Pipelines, and choose the **release pipeline** you created earlier. Select the **ellipsis** and choose **Edit**.
15. **Delete** the artifact you used earlier. **Add** a new artifact by choosing your Azure Pipelines build pipeline. The Azure Pipelines build pipeline will now provide the artifact for the CD part of the pipeline.
16. In the **Artifacts** section, click on **+ Add Artifact** and choose **Jenkins** for **Source type**. Select your Jenkins service connection. Then select the Jenkins source job and choose **Add**.

You can now test the pipeline (as you did on earlier steps) with a pull request or by pushing new code to your branch. An Azure Pipelines build will initiate, a Jenkins job executes, and an Azure Pipelines release pipeline will deploy your changes to Azure.

Next Steps

In this tutorial, you automated the deployment of an app to Azure using Jenkins for the CI build and Azure Pipelines for release. You learned how to:

- Get the Sample App
- Configure Jenkins credentials and Azure Pipelines plugin
- Configure a Jenkins CI build with Azure Pipelines integration
- Create a Jenkins service connection and service hooks in Azure Pipelines
- Create an Azure Pipelines release pipeline for CD to Azure
- Test the CI/CD pipeline with a pull request
- (Optionally) Create an Azure Pipelines build pipeline to wrap the Jenkins CI job

[Improve code quality with branch policies](#)

Tutorial: Deploy to Kubernetes on Azure Container Service (AKS) with Jenkins CI and Azure Pipelines CD

11/19/2018 • 16 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Azure Pipelines provides integration with Jenkins so that you can:

- Continue to use your existing investments in Jenkins.
- Create a single release pipeline using Azure Pipelines that deploys to a variety of Azure targets.

In this tutorial, you use Jenkins for Continuous Integration (CI) and Azure Pipelines for Continuous Delivery (CD) to deploy a **Spring Boot app** to an **Azure Container Service (AKS) Kubernetes cluster**.

You will:

- Get the sample app
- Configure the sample app to build and push a Docker image to your ACR
- Configure the sample app to include a YAML file used for deploying to your AKS cluster
- Configure Jenkins credentials to connect to Azure Pipelines
- Configure Jenkins Maven global settings
- Create Jenkins Service Hooks in Azure Pipelines
- Configure a Jenkins CI build with Azure Pipelines integration
- Install the Release Management Utility tasks Azure Pipelines extension
- Create an Azure Pipelines release pipeline for CD to Azure
- Test the CI/CD pipeline with a pull request
- View the deployed sample app
- Delete your AKS cluster

Prerequisites

- An Azure DevOps organization. If you don't have one, you can [create one for free](#). If your team already has one, then make sure you are an administrator of the project you want to use.
- An Azure subscription. You can get one free from [Visual Studio Dev Essentials](#).
- The [Azure Command-Line Interface \(CLI\)](#).
- You need a Spring Boot app. You can fork the sample app found [here](#).
- You need an Azure Container Registry (ACR). You can follow steps to deploy an ACR and login to the registry using the Azure CLI via the steps [here](#).
- An AKS cluster. You can follow the steps for creating this [here](#).
- Access to a Jenkins server with Maven and the VSTS plugin configured. If you have not yet created a

Jenkins server, see [Create a Jenkins master on an Azure Virtual Machine](#). Also, the following Jenkins plugins must be installed:

- **VS Team Services Continuous Deployment** plugin. You can find additional information about the [TFS plugin here](#).
- **Config File Provider** plugin. You can find additional information about the [Config File plugin here](#).
- **Maven Integration** plugin. You can find additional information about the [Maven Integration plugin here](#).

Get the sample app

You need an app stored in a Git repository. You use this app to build and deploy. For this tutorial, we recommend you use [this Spring Boot sample app available from GitHub](#). This tutorial uses a Spring Boot sample application that is configured for deployment to an AKS cluster. If you want to work with your own repository, you should configure a similar sample.

1. In Azure Repos, on the **Code** page for your Azure Repos project, select the option to **Import repository**.
2. In the **Import a Git repository** dialog box, paste the following URL into the **Clone URL** text box.

```
https://github.com/spring-guides/gs-spring-boot-docker
```

3. Click **Import** to copy the sample code into your Git repo.
4. Select **Clone** at the top right, and keep the **clone URL** for future steps in this tutorial.

Configure the sample app to build and push a Docker image to your ACR

The [Spotify Dockerfile Maven plugin](#) is used to build and push a Docker image to a registry, such as ACR. The Spring Boot sample app's Maven **pom.xml** file is already configured to use this plugin.

You will find the **pom.xml** file in your repository in the folder named **complete**. You must update the **pom.xml** file with your ACR's login server:

1. Navigate to your Azure Repos project. Select the **Code** tab.
2. In the code explorer, expand the **complete** folder and select the **pom.xml** file.
3. In the **Contents** pane that shows the contents of the **pom.xml** file, select the **Edit** button.
4. Update the `<properties>` collection in the **pom.xml** with the login server value for your ACR. The login server is the name of your ACR appended with `.azurecr.io`. For example, `yourACRName.azurecr.io`.

```
<properties>
  <docker.image.prefix>wingtiptoysregistry.azurecr.io</docker.image.prefix>
  <java.version>1.8</java.version>
</properties>
```

5. Update the **com.spotify** plugin in the **pom.xml** file so that the `<configuration>` section includes the tag for the Docker image that will be built and pushed to your ACR. The tag will be set to the current Jenkins build id. Also, add the **useMavenSettingsForAuth** setting to the configuration so that in a later step, you can configure Maven to authenticate with your ACR which is a private registry.

```

<plugin>
  <groupId>com.spotify</groupId>
  <artifactId>dockerfile-maven-plugin</artifactId>
  <version>1.3.4</version>
  <configuration>
    <repository>${docker.image.prefix}/${project.artifactId}</repository>
    <tag>${env.BUILD_NUMBER}</tag>
    <useMavenSettingsForAuth>true</useMavenSettingsForAuth>
    <buildArgs>
      <JAR_FILE>target/${project.build.finalName}.jar</JAR_FILE>
    </buildArgs>
  </configuration>
</plugin>

```

6. Select **Commit** and choose to commit these changes to the **master** branch.

Configure the sample app to include a YAML file used for deploying to your AKS cluster

The YAML file contains deployment settings for pulling the docker image from the ACR. Update this file to include the name of your ACR:

1. Navigate to your Azure Repos project. Select the **Code** tab.
2. In the code explorer, select the **complete** folder's ellipsis button to open the context menu that shows more actions.
3. From the context menu, select **New, File**.
4. Name the file **K8sDeploy.yaml** and select **Create**.
5. Copy and paste the following contents into the **K8sDeploy.yaml** file. Ensure the image prefix `<yourACRName>` is replaced with the name of your ACR appended with `.azurecr.io`. Also, notice that the image is tagged with **Build.BuildId**. This is a token that will be automatically replaced during the Azure Pipelines release so that it's set to the current Jenkins build id. For example, if the current Jenkins build id is 5, the full name of an image that will be pulled during the Azure Pipelines release will look similar to:
`yourACRName.azurecr.io/gs-spring-boot-docker:5`.

```

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: gs-spring-boot-docker
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: gs-spring-boot-docker
    spec:
      containers:
        - image: <yourACRName>.azurecr.io/gs-spring-boot-docker:__Build.BuildId__
          name: gs-spring-boot-docker
          imagePullPolicy: Always
          ports:
            - containerPort: 8080
        imagePullSecrets:
          - name: regsecret
      ---
apiVersion: v1
kind: Service
metadata:
  name: gs-spring-boot-docker
spec:
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  selector:
    app: gs-spring-boot-docker
  type: LoadBalancer

```

6. Select **Commit** and choose to commit these changes to the **master** branch.

Configure Jenkins credentials to connect to Azure Pipelines

You must configure credentials for connecting to Azure Pipelines. When using credentials to connect to Azure Pipelines, it is a best practice to use a **personal access token (PAT)**. You also need to create a Jenkins credential to use in your Jenkins build jobs.

NOTE

Ensure the PAT you use for the following steps contains the **Release (read, write, execute and manage), Code (read), Build (read and execute) permissions in Azure Pipelines**.

1. Create a PAT in your Azure Pipelines account. Jenkins requires this information to access your Azure DevOps organization. Ensure you **store** the token information for upcoming steps in this section. Read [How do I create a personal access token for Azure Pipelines and TFS](#) to learn how to generate a PAT, or use an existing PAT if you have one.
2. Open your Jenkins account and select **Credentials, System**, and then choose **Add Domain**.
3. Enter a **name, description**, and then select **Add hostname**.
4. In the **Include** text box, enter `\dev.azure.com/*` to enable this credential for all Azure DevOps organizations. You can optionally enter your specific Azure DevOps organization.
5. Select **Ok**. You will see a message about empty credentials. Select the link for **adding some credentials**.
6. For **Kind**, choose **Username with password**. Enter the **Username** and the **PAT** you created earlier for

Password. Select **Ok**.

Configure Jenkins Maven global settings

Since your ACR is a private registry, you must configure the user name and password for your ACR in Maven's global settings in Jenkins. This will authenticate Maven so that it can push images to your ACR during a build.

1. Retrieve the password for your ACR using the Azure CLI. Replace <yourACRName> with the name of your registry.

```
az acr credential show --name <acrName> --query passwords[0]
```

2. Navigate to your Jenkins server. Select **Manage Jenkins** and then select the **Managed files** link.
3. To create the Maven **settings.xml** file the first time, select the **Add a new Config** link.
4. Select the **Maven settings.xml** radio button and select the **Submit** button.
5. Enter a **Name** for the file.
6. In the **Content** window, update the <servers> section of the contents to include the name of your ACR and its password. For example:

```
<server>
<id>yourACRName.azurecr.io</id>
<username>yourACRName</username>
<password>e1M3Y3Z9vuUtusVx12+GoQvuAV3wsQFZ</password>
</server>
```

7. Select the **Submit** button to save the settings.

Configure a Jenkins CI build with Azure Pipelines integration

You create a Jenkins build job to use the source code stored in your Azure Repos repository and execute a basic build. Ensure your Jenkins server has **Maven** installed and configured. You also create triggers to enable CI builds when code changes are pushed to the repository, and a CD trigger to initiate an Azure Pipelines release after the Jenkins build completes via a post build action.

1. Navigate to your Jenkins server. Click **New Item**. Enter an **item name**.
2. Choose **Maven project**. Select **OK**.
3. In the **Source Code Management** tab, select **Git** and enter the **clone URL** you saved earlier for the Azure Repos **Repository URL** and the branch containing your app code. If you are using Team Foundation Server, you can choose the option for **Team Foundation Version Control (TFVC)**.

The screenshot shows the Jenkins configuration interface for a new job. The 'Source Code Management' tab is active. In the 'Repositories' section, 'Git' is selected, and a repository URL is set to <https://github.com/azoooinmyluggage/fabrikam-node.git>. The 'Credentials' dropdown is set to '- none -'. There are buttons for 'Advanced...', 'Add Repository', and 'Add Branch'. In the 'Branches to build' section, a branch specifier `*/master` is entered. The 'Repository browser' is set to '(Auto)'. Under 'Additional Behaviours', 'Subversion' and 'Team Foundation Version Control (TFVC)' are listed.

4. Select the **Credentials** drop down and choose the credential you created earlier. You should successfully authenticate to your Azure Repos repository and not receive errors before continuing. If you see errors, you likely have an issue with your credentials and Azure DevOps **PAT**.
5. Select the **Build Triggers** tab, then **tick** the checkboxes for **Build when a change is pushed to TFS/Team Services** and **Build when a change is pushed to a TFS pull request**. These two options rely on **Azure Pipelines Service Hooks** which you create in the next section.
6. Select the **Build** tab, set the **Root POM** to the relative path to the sample app's **pom.xml: complete/pom.xml**.
7. Enter **clean package** for **Goals**.
8. Select the **Advanced...** button. Set the **Settings file** to **provided settings.xml** and set **Provided Settings** to the name of the Maven settings.xml file that you created earlier.
9. Select the **Post-build Actions** tab. Choose **Add post-build action**, then select **Archive the artifacts**.
10. For **Files to archive** enter the relative path to the Kubernetes YAML deployment file: **complete/K8sDeploy.yaml**.
11. Select the **Post-build Actions** tab. Choose **Add post-build action**, then select **Trigger release in TFS/Team Services**.
12. Enter a **Collection URL**. An example is <http://dev.azure.com/{your-organization}/DefaultCollection>
13. Enter the **project** and choose a **release pipeline** name. Store the **release pipeline** name since it is needed in later steps of this tutorial.
14. Enter the **username** and **PAT** you created earlier.
15. **Save** the Jenkins project.

Create a Jenkins and AKS service connection in Azure Pipelines

You configure a Jenkins service connection to allow Azure Pipelines to connect to your Jenkins server. You also need to configure an AKS service connection to allow Azure Pipelines to access your AKS cluster to configure deployment.

1. Open the **Services** page in Azure Pipelines, open the **New service connection** list, and choose **Jenkins**.

The screenshot shows the Azure Pipelines Services page. At the top, there's a navigation bar with 'Fabrikam' and dropdown menus for Home, Code, Work, Build & Release, Test, and a gear icon for settings. Below the navigation is a secondary navigation bar with links for Overview, Work, Security, Version Control, Policies, Agent queues, Notifications, Service Hooks, and Services (which is underlined). Under the Services link, there are tabs for Endpoints and XAML Build Services. A modal window is open, titled '+ New Service Endpoint'. It contains a dropdown menu with options: Generic, GitHub, Jenkins (which is highlighted with a red box), and Kubernetes. The Jenkins option is currently selected.

2. Enter a name for the connection.
3. Enter the URL of your Jenkins server, and if using **http** tick the **Accept untrusted SSL certificates** option.
An example URL is: `http://{YourJenkinsURL}.westcentralus.cloudapp.azure.com`
4. Enter the **user name and password** for your Jenkins account.
5. Choose **Verify connection** to check that the information is correct.
6. Choose **OK** to create the service connection.

Add a second service connection for the AKS cluster.

1. Opening the **New service connection** list, and choose **Kubernetes**.
2. Enter a name for the connection.
3. Set the **Server URL** to the cluster's fully qualified domain name, such as **http://yourAKSCluster-yourResourceGroup-e65186-1e0d187e.hcp.eastus.azmk8s.io**. Retrieve the fully qualified domain name using the Azure CLI. Replace `<yourResourceGroup>` with the name of the resource group that contains the AKS cluster. Replace `<yourAKSCluster>` with the name of the AKS cluster.

```
az aks show --resource-group <yourResourceGroup> --name <yourAKSCluster> --query fqdn
```

4. Set the **KubeConfig** to the access credentials for the AKS cluster. Use the Azure CLI to get these credentials:

```
az aks get-credentials --resource-group <yourResourceGroup> --name <yourAKSCluster>
```

This command will get the access credentials for the AKS cluster. Navigate to the **.kube** folder under your home directory, such as **C:\Users\YOUR_HOMEDIR.kube**. Copy the contents of the **config** file and paste it in the **Kubernetes Connection** window. Select **OK** to create the service connection.

Create Jenkins Service Hooks in Azure Pipelines

You must also configure two Jenkins service hooks so you can execute CI builds via automated triggers for both simple commits as well as pull requests to your Azure Repos Git repository.

- From the **Service Hooks** page in Azure Pipelines, Select the + to add a new service and choose **Jenkins**. Select **next**.
- Choose **Code pushed** for the type of event trigger. Select your **code repository** that contains the sample application you imported earlier. Select **Next**.
- Enter the URL of your Jenkins server. An example is below.

http://{YourJenkinsURL}.westcentralus.cloudapp.azure.com
- Enter the **user name and password** for your Jenkins account.
- Select the Jenkins build you created earlier.
- Choose **Test** to check that the information is correct.
- Choose **Finish** to create the service connection.
- Repeat the steps in this section to create another **service hook** for the **pull request merge attempted** trigger type. This will allow either simple commits to the repository as well as pull requests to both trigger the Jenkins build.

Install the Release Management Utility tasks Azure Pipelines extension

A release pipeline specifies the steps that Azure Pipelines executes to deploy the app. In this example, you deploy your app that originates from the Jenkins CI system. You deploy to a Docker image running Tomcat and a Spring Boot app to an AKS cluster.

Before you create the release pipeline, you need to install an Azure Pipelines extension that will be used to replace the **K8sDeploy.yaml** file's **Build.BuildId** token with the current Jenkins build id.

- In your Azure DevOps organization, on the top right-hand side of the browser, Select the **Browse Marketplace** menu item. (The icon appears as a shopping bag.)
- Search for the **Release Management Utility Tasks** extension provided by **Microsoft DevLabs**. The extension includes the **Tokenizer** utility.
- Select the **Get it free** button.
- Select your Azure DevOps organization and select the **Install** button to install the extension.
- After the extension is installed, select the **Proceed to Organization** button and navigate back to your Azure DevOps project.

Create an Azure Pipelines release pipeline for CD to Azure

- Open **Releases in Azure Pipelines**, and choose **Create release pipeline**.
- Select the **Empty** template by choosing **Start with an empty pipeline**.
- In the **Artifacts** section, click on + **Add Artifact** and choose **Jenkins** for **Source type**. Select your Jenkins service connection. Then select the Jenkins source job and choose **Add**.

Add two tasks to the release pipeline. The first task updates the **K8sDeploy.yaml** file to pull the image tagged with the current Jenkins build id.

- Next to the **1 job, 0 stages** link, select the + on the **Agent Job** to add a task to the job.
- Search for the **Tokenize with XPath/Regular expressions** task which was added with the extension that was installed in the previous step. Select **Add** to add the task.
- Set the **Source filename** to the **K8sDeploy.yaml** that is archived by the Jenkins job during the build. This

task automatically replaces the **Build.BuildId** token with the current Jenkins build id.

The second task deploys to the AKS cluster:

1. Select the + button to add a second task. Search for the **Deploy to Kubernetes** task. Select **Add** to add the task.
2. Set the **Kubernetes Service Connection** to the name of the service connection that you created for the AKS cluster.
3. Set the **Command to apply**.
4. Select the **Use Configuration files** check box and set the **Configuration File** to the **K8sDeploy.yaml** file.
5. Expand the **Container Registry Details** section of the task.
6. Set **Container Registry type** to **Azure Container Registry**.
7. Set **Azure subscription** to your subscription. If you do not have an existing Azure connection in Azure Pipelines, you can follow the steps [here](#) to create one.
8. Set **Azure Container Registry** to the name of your ACR.
9. Set **Secret name** to the secret provided in the **K8sDeploy.yaml** file which is named **regsecret**. This is the name of an object in the AKS cluster that is used to store an authentication token. The cluster uses this token to authenticate to the ACR to pull images.
10. Ensure the **name** for your release pipeline matches the same name you chose earlier during the **Create a Jenkins service connection and service hooks in Azure Pipelines** steps.
11. Click **Save**, and then click **OK** to save the release pipeline.

Test the CI/CD pipeline with a pull request

You can initiate the CI build and the subsequent CD deployment to Azure by completing a pull request into your master branch. The Jenkins build will initiate due to the service hook you set up earlier, and the Jenkins post build action will initiate an Azure Pipelines release which will deploy your app to the Azure App Service.

1. Navigate to **Code** in Azure Repos, then select your **repository**.
2. Select **New branch** to create a branch from the master branch. Enter a **name** for your new branch, and then select **Create branch**.
3. Select your new branch, then navigate to the **complete/src/main/java/hello/Application.java** file.
4. Select **Edit**, then make a change to the message displayed in the **home()** method and **Commit** your changes.
5. Select **Pull Requests**, then select **New Pull Request**. Select **Create** to issue a pull request from your branch to master.
6. Select **Approve**, then select **Complete**, then **Complete Merge**. This code change will initiate a CI build in Jenkins.
7. Navigate to your **Jenkins dashboard** and examine the build that is executing. Once it finishes, a new Docker image will be pushed to your ACR that is tagged with the current Jenkins build id. You can then navigate to Azure Pipelines to watch the **Release Pipeline** execute. In a few moments, the Docker image for the current Jenkins build will be deployed to your AKS cluster.

You are now using Jenkins CI builds with an Azure Repos code repository and an Azure Pipelines release pipeline to perform CI/CD to **Azure Container Services (AKS)**. You can easily track your code changes and deployments

via the rich reporting capabilities of Azure Pipelines, and leverage Jenkins to execute CI builds.

View the deployed sample app

Once the app is deployed to the AKS cluster, you can query the external IP address using **kubectl**, the Kubernetes command-line client. You can learn how to install and connect **kubectl** to your AKS Cluster by following [these steps](#).

1. Use the following command for querying the external IP address for the deployed app:

```
kubectl get svc -w
```

kubectl will display the internal and external IP addresses; for example:

NAME	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	10.0.0.1	<none>	443/TCP	19h
gs-spring-boot-docker	10.0.242.8	13.65.196.3	80:31215/TCP	3m

1. Use the external IP address to open the sample app in a web browser.

Delete your AKS cluster

When your AKS cluster is no longer needed, you can use the `az group delete` command to remove the resource group, which will remove all of its related resources; for example:

```
az group delete --name <yourAKSCluster> --yes --now-wait
```

Next Steps

In this tutorial, you automated the deployment of an app to Azure using Jenkins build and Azure Pipelines for release. You learned how to:

- Get the sample app
- Configure the sample app to build and push a Docker image to your ACR
- Configure the sample app to include a YAML file used for deploying to your AKS cluster
- Configure Jenkins credentials to connect to Azure Pipelines
- Configure Jenkins Maven global settings
- Create Jenkins Service Hooks in Azure Pipelines
- Configure a Jenkins CI build with Azure Pipelines integration
- Install the Release Management Utility tasks Azure Pipelines extension
- Create an Azure Pipelines release pipeline for CD to Azure
- Test the CI/CD pipeline with a pull request
- View the deployed sample app
- Delete your AKS cluster

[Integrate your Jenkins CI jobs with Azure Pipelines Deploy to Kubernetes with Fabric8](#)

How To: Extend your deployments to IIS Deployment Groups

10/15/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

You can quickly and easily deploy your ASP.NET or Node.js app to an IIS Deployment Group using Azure Pipelines or Team Foundation Server (TFS), as demonstrated in [this example](#). In addition, you can extend your deployment in a range of ways depending on your scenario and requirements. This topic shows you how to:

- [Dynamically create and remove a deployment group](#)
- [Apply stage-specific configurations](#)
- [Perform a safe rolling deployment](#)
- [Deploy a database with your app](#)

Prerequisites

You should have worked through the example [CD to an IIS Deployment Group](#) before you attempt any of these steps. This ensures that you have the release pipeline, build artifacts, and websites required.

Dynamically create and remove a deployment group

You can create and remove deployment groups dynamically if you prefer by using the [Azure Resource Group Deployment task](#) to install the agent on the machines in a deployment group using ARM templates. See [Provision deployment group agents](#).

Apply stage-specific configurations

If you deploy releases to multiple stages, you can substitute configuration settings in **Web.config** and other configuration files of your website using these steps:

1. Define stage-specific configuration settings in the **Variables** tab of a stage in a release pipeline; for example,
`<connectionStringKeyName> = <value>`.
2. In the **IIS Web App Deploy** task, select the checkbox for **XML variable substitution** under **File Transforms and Variable Substitution Options**.

If you prefer to manage stage configuration settings in your own database or Azure keyvault, add a task to the stage to read and emit those values using

```
##vso[task.setvariable variable=connectionString;issecret=true]<value> .
```

At present, you cannot apply a different configuration to individual IIS servers.

Perform a safe rolling deployment

If your deployment group consists of many IIS target servers, you can deploy to a subset of servers at a time. This ensures that your application is available to your customers at all times. Simply select the **Deployment group job** and use the slider to configure the **Maximum number of targets in parallel**.

The screenshot shows the 'Tasks' tab of a pipeline named 'SampleApp - 1'. On the left, there's a sidebar with an 'IIS' button and 'Manage IISWebsite IIS Web App Manage' text. The main area has a 'Production Deployment process' section and a 'Deployment group job' section highlighted with a red box. The 'Deployment group job' section contains fields for 'Display name' (set to 'Deployment group job'), 'Deployment targets' (set to 'SampleGroup'), 'Required tags' (empty), and 'Targets to deploy to in parallel' (set to 'Multiple'). A slider for 'Maximum number of targets in parallel' is set to 50% (0 targets). Below these are fields for 'Timeout' (0) and 'Deployment job cancel timeout in minutes' (1).

Deploy a database with your app

To deploy a database with your app:

1. Add both the IIS target servers and database servers to your deployment group. Tag all the IIS servers as `web` and all database servers as `database`.
2. Add two machine group jobs to stages in the release pipeline, and a task in each job as follows:

First Run on deployment group job for configuration of web servers.

- **Deployment group:** Select the deployment group you created in the [previous example](#).
- **Machine tags:** `web`

Then add an **IIS Web App Deploy** task to this job.

Second Run on deployment group job for configuration of database servers.

- **Deployment group:** Select the deployment group you created in the [previous example](#).
- **Machine tags:** `database`

Then add a **SQL Server Database Deploy** task to this job.

Deploy your Web Deploy package to IIS servers using WinRM

11/15/2018 • 7 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

A simpler way to deploy web applications to IIS servers is by using [deployment groups](#) instead of WinRM. Deployment groups are currently in preview for some users in Azure Pipelines. They are not yet available in TFS.

Continuous deployment means starting an automated deployment pipeline whenever a new successful build is available. Here we'll show you how to set up continuous deployment of your ASP.NET or Node.js app to one or more IIS servers using Azure Pipelines. A task running on the [Build and Release agent](#) opens a WinRM connection to each IIS server to run Powershell scripts remotely in order to deploy the Web Deploy package.

Get set up

Begin with a CI build

Before you begin, you'll need a CI build that publishes your Web Deploy package. To set up CI for your specific type of app, see:

- [Build your ASP.NET 4 app](#)
- [Build your ASP.NET Core app](#)
- [Build your Node.js app with gulp](#)

WinRM configuration

Windows Remote Management (WinRM) requires target servers to be:

- Domain-joined or workgroup-joined
- Able to communicate using the HTTP or HTTPS protocol
- Addressed by using a fully-qualified domain name (FQDN) or an IP address

This table shows the supported scenarios for WinRM.

JOINED TO A	PROTOCOL	ADDRESSING MODE
Workgroup	HTTPS	FQDN
Workgroup	HTTPS	IP address
Domain	HTTPS	IP address

JOINED TO A	PROTOCOL	ADDRESSING MODE
Domain	HTTPS	FQDN
Domain	HTTP	FQDN

Ensure that your IIS servers are set up in one of these configurations. For example, do not use WinRM over HTTP to communicate with a Workgroup machine. Similarly, do not use an IP address to access the target server(s) when you use HTTP. Instead, in both scenarios, use HTTPS.

If you need to deploy to a server that is not in the same workgroup or domain, add it to trusted hosts in your [WinRM configuration](#).

Follow these steps to configure each target server.

1. Enable File and Printer Sharing. You can do this by executing the following command in a Command window with Administrative permissions:

```
netsh advfirewall firewall set rule group="File and Printer Sharing" new enable=yes
```

2. Check your PowerShell version. You need PowerShell version 4.0 or above installed on every target machine. To display the current PowerShell version, execute the following command in the PowerShell console:

```
$PSVersionTable.PSVersion
```

3. Check your .NET Framework version. You need version 4.5 or higher installed on every target machine. See [How to: Determine Which .NET Framework Versions Are Installed](#).
4. Download from GitHub [this PowerShell script](#) for Windows 10 and Windows Server 2016, or [this PowerShell script](#) for previous versions of Windows. Copy them to every target machine. You will use them to configure WinRM in the following steps.
5. Decide if you want to use HTTP or HTTPS to communicate with the target machine(s).

- If you choose HTTP, execute the following in a Command window with Administrative permissions:

```
ConfigureWinRM.ps1 {FQDN} http
```

This command creates an HTTP WinRM listener and opens port 5985 inbound for WinRM over HTTP.

- If you choose HTTPS, you can use either a FQDN or an IP address to access the target machine(s). To use a FQDN to access the target machine(s), execute the following in the PowerShell console with Administrative permissions:

```
ConfigureWinRM.ps1 {FQDN} https
```

To use an IP address to access the target machine(s), execute the following in the PowerShell console with Administrative permissions:

```
ConfigureWinRM.ps1 {ipaddress} https
```

These commands create a test certificate by using **MakeCert.exe**, use the certificate to create an HTTPS WinRM listener, and open port 5986 inbound for WinRM over HTTPS. The script also increases the WinRM **MaxEnvelopeSizekb** setting. By default on Windows Server this is 500 KB, which can result in a "Request size exceeded the configured MaxEnvelopeSize quota" error.

IIS configuration

If you are deploying an ASP.NET app, make sure that you have ASP.NET 4.5 or ASP.NET 4.6 installed on each of your IIS target servers. For more information, see [this topic](#).

If you are deploying an ASP.NET Core application to IIS target servers, follow the additional instructions in [this topic](#) to install .NET Core Windows Server Hosting Bundle.

If you are deploying a Node.js application to IIS target servers, follow the instructions in [this topic](#) to install and configure IISnode on IIS servers.

In this example, we will deploy to the Default Web Site on each of the servers. If you need to deploy to another website, make sure you configure this as well.

IIS WinRM extension

Install the [IIS Web App Deployment Using WinRM](#) extension from Visual Studio Marketplace in Azure Pipelines or TFS.

Define and test your CD release pipeline

Continuous deployment (CD) means starting an automated release pipeline whenever a new successful build is available. Your CD release pipeline picks up the artifacts published by your CI build and then deploys them to your IIS servers.

1. Do one of the following:

- If you've just completed a CI build (see above) then, in the build's **Summary** tab under **Deployments**, choose **Create release** followed by **Yes**. This starts a new release pipeline that's automatically linked to the build pipeline.
- Open the **Releases** tab of **Azure Pipelines**, open the + drop-down in the list of release pipelines, and choose **Create release pipeline**.

2. Choose **Start with an empty pipeline**.

3. If you created your new release pipeline from a build summary, check that the build pipeline and artifact is shown in the **Artifacts** section on the **Pipeline** tab. If you created a new release pipeline from the **Releases** tab, choose the + **Add** link and select your build artifact.

Add an artifact

Source type

- Build
- Git
- Github

4 more artifact types ▾

Project * ⓘ

DotNetSample

Source (build pipeline) * ⓘ

DotNetSample-ASP.NET Core-Cl

Default version * ⓘ

Latest

Source alias ⓘ

_DotNetSample-ASP.NET Core-Cl

The artifacts published by each version will be available for the latest successful build of DotNetSample-ASP.NET Core-Cl.

Add

- Choose the **Continuous deployment** icon in the **Artifacts** section, check that the continuous deployment trigger is enabled, and add a filter to include the **master** branch.

All pipelines > SampleApp - 1

Pipeline Tasks Variables Retention Options History

Artifacts | + Add

Stages | + A

Continuous deployment trigger

Build: DotNetSample-ASP.NET Core-Cl

Enabled

Creates a release every time a new build is available.

Build branch filters ⓘ

Type	Build branch
Include	The build pipeline's default bra...
+ Add	

- On the **Variables** tab of the stage in release pipeline, configure a variable named **WebServers** with the list of IIS servers as its value; for example `machine1,machine2,machine3`.
- Configure the following tasks in the stage:

 Deploy: Windows Machine File Copy - Copy the Web Deploy package to the IIS servers.

- **Source:** Select the Web deploy package (zip file) from the artifact source.
- **Machines:** `$(WebServers)`
- **Admin Login:** Enter the administrator credentials for the target servers. For workgroup-joined computers, use the format `.\username`. For domain-joined computers, use the format `domain\username`.
- **Password:** Enter the administrator password for the target servers.
- **Destination Folder:** Specify a folder on the target server where the files should be copied to.

 [Deploy: WinRM - IIS Web App Deployment](#) - Deploy the package.

- **Machines:** `$(WebServers)`
- **Admin Login:** Enter the administrator credentials for target servers. For workgroup-joined computers, use the format `.\username`. For domain-joined computers, use the format `domain\username`.
- **Password:** Enter the administrator password for target servers.
- **Protocol:** Select `HTTP` or `HTTPS` (depending on how you configured the target machine earlier). Note that if the target machine is workgroup-joined, you must choose `HTTPS`. You can use HTTP only if the target machine is domain-joined and configured to use a FDQN.
- **Web Deploy Package:** Fully qualified path of the zip file you copied to the target server in the previous task.
- **Website Name:** `Default Web Site` (or the name of the website if you configured a different one earlier).

7. Edit the name of the release pipeline, click **Save**, and click **OK**. Note that the default stage is named Stage1, which you can edit by clicking directly on the name.

You're now ready to create a release, which means to run the release pipeline with the artifacts produced by a specific build. This will result in deploying the build to IIS servers:

1. Choose **+ Release** and select **Create a release**.
2. In the **Create a new release** panel, check that the artifact version you want to use is selected and choose **Create**.
3. Choose the release link in the information bar message. For example: "Release **Release-1** has been created".
4. In the pipeline view, choose the status link in the stages of the pipeline to see the logs and agent output.
5. After the release is complete, navigate to your app and verify its contents.

Q & A

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#)

page.

Deploy apps to Azure Government Cloud

10/9/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2017 | TFS 2018

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions.

Service connections are called *service endpoints* in TFS 2018 and in older versions.

[Azure Government Clouds](#) provide private and semi-isolated locations for specific Government or other services, separate from the normal Azure services. Highest levels of privacy have been adopted for these clouds, including restricted data access policies.

Azure Pipelines is not available in Azure Government Clouds, so there are some special considerations when you want to deploy apps to Government Clouds because artifact storage, build, and deployment orchestration must execute outside the Government Cloud.

To enable connection to an Azure Government Cloud, you specify it as the **Environment** parameter when you create an [Azure Resource Manager service connection](#). You must use the full version of the service connection dialog to manually define the connection. Before you configure a service connection, you should also ensure you meet all relevant compliance requirements for your application.

You can then use the service connection in your [build and release pipeline tasks](#).

Next

- [Deploy an Azure Web App](#)
- [Troubleshoot Azure Resource Manager service connections](#)

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

How-To: CI/CD with App Service and Azure Cosmos DB

11/19/2018 • 4 minutes to read • [Edit Online](#)

Azure Pipelines

Create a continuous integration (CI) and continuous delivery (CD) pipeline for Azure Comsos DB backed Azure App Service Web App. Azure Cosmos DB is Microsoft's globally distributed, multi-model database. Cosmos DB enables you to elastically and independently scale throughput and storage across any number of Azure's geographic regions.

You will:

- Clone a sample Cosmos DB and Azure Web App to your repository
- Create a Cosmos DB collection and database
- Set up CI for your app
- Set up CD to Azure for your app
- Review the CI/CD pipeline

Prerequisites

- An Azure subscription. You can get one free through [Visual Studio Dev Essentials](#).
- An Azure DevOps organization. If you don't have one, you can [create one for free](#). If your team already has one, then make sure you are an administrator of the project you want to use.
- A SQL API based Cosmos DB instance. If you don't have one, you can follow the initial steps in [this tutorial](#) to create a Cosmos DB instance and collection.

Clone a sample Cosmos DB and Azure Web App to your repository

This sample shows you how to use the Microsoft Azure Cosmos DB service to store and access data from an ASP.NET MVC application hosted on Azure App Service. The application is a simple TODO application. You can learn more about the sample application [here](#).

To import the sample app into a Git repo in Azure Repos:

1. Sign into your Azure DevOps organization.
2. on the **Code** page for your project in Azure Repos, select the drop-down and choose the option to **Import repository**.
3. On the **Import a Git repository** dialog box, paste <https://github.com/Azure-Samples/documentdb-dotnet-todo-app.git> into the **Clone URL** text box.
4. Click **Import** to copy the sample code into your Git repo.

Set up CI for your App

Set up CI for your ASP.NET application and Cosmos DB to build and create deployable artifacts.

1. In **Azure Pipelines**, select **Builds**.
2. On the right-side of the screen, select **+ NEW** to create a new build.

3. Choose the **repository** for the sample application you imported earlier in this tutorial, and then choose **continue**.

4. Search for the **ASP.NET Application** build template, and then select **Apply**.

Select a template
Or start with an [Empty process](#)

Choose a template

Choose a template that builds your kind of app.
Don't worry if it's not an exact match;
you can add and customize the tasks later.

Featured

- .NET Desktop**
Build and run tests for .NET Desktop or Windows Classic Desktop solutions. This template requires that Visual Studio be installed on the build agent.
- ASP.NET**
Build ASP.NET web applications
- ASP.NET Core**
Build ASP.NET Core web applications targeting .NET Core
- ASP.NET Core (.NET Framework)**
Build ASP.NET Core web applications targeting the full .NET Framework
- Azure Web App**

Apply

5. Select the **triggers**, and then select the checkbox for ""Enable continuous integration**. This setting ensures every commit to the repository executes a build.

6. Select **Save & Queue**, and then choose **Save and Queue** to execute a new build.

7. Select the build **hyperlink** to examine the running build. In a few minutes the build completes. The build produces artifacts which can be used to deploy to Azure.

Set up CD to Azure for your App

The CI for the sample app produces the artifacts needed for deployment to Azure. Follow the steps below to create a release pipeline which uses the CI artifacts for deploying a Cosmos DB instance.

1. Select **Release** to create a release pipeline linked to the build artifacts from the CI pipeline you created with the previous steps.

2. Choose the **Azure App Service deployment** template, and then choose **Apply**.

3. On the **Environments** section, select the **job and task** link.

4. Select the **Azure Subscription**, and then select **Authorize**.

All definitions > MyFirstProject-CI - CD

Pipeline Tasks Variables Retention Options History

Environment 1 Some settings need attention

Run on agent Run on agent

Deploy Azure App Service Some settings need attention

Environment name

Parameters | Unlink all

Azure subscription: Windows Azure MSDN - Visual Studio Ult

Authorize

Click Authorize to configure Azure service connection

5. Choose an **App Service name**.

6. Select the **Deploy Azure App Service** task, and then select the **File Transforms & Variable Substitution Options** setting.

7. Enable the checkbox for **XML Variable substitution**.

8. At the top of the menu, select **Variables**.

9. Retrieve your **endpoint** (URL) and **authKey** (primary or secondary key) for your Azure Cosmos DB account. This information can be found on the Azure portal.

10. Select **+ Add** to create a new variable named **endpoint**. Select **+ Add** to create a second variable named **authKey**.
11. Select the **padlock** icon to make the authKey variable secret.
12. Select the **Pipeline** menu.
13. Under the **Artifacts** ideas, choose the **Continuous deployment trigger** icon. On the right side of the screen, ensure **Enabled** is on.
14. Select **Save** to save changes for the release definition.

Review the CI/CD pipeline

Follow the steps below to test and review the CI/CD pipeline.

1. on the **Code** page select the **ellipsis (...)** icon next to the **web.config** file in the **src** directory, and then select **Edit**.
2. Replace the existing **value** (ToDoList) for the **database** key in the **appSettings** section of the web.config with a new value such as **NewToDoList**. You will commit this change to demonstrate creating a new Cosmos DB database as part of the CI/CD pipeline. This is a simple change to demonstrate CI/CD capabilities of Cosmos DB with Azure Pipelines. However, more **complicated code changes** can also be deployed with the same CI/CD pipeline.
3. Select **Commit**, and then choose **Commit** to save the changes directly to the repository.
4. On the **Build** page select **Builds** and you will see your CI build executing. You can follow the build execution with the interactive logging. Once the build completes, you can also monitor the release.
5. Once the release finishes, navigate to your Cosmos DB service to see your new database.

The continuous integration trigger you enabled earlier ensures a build executes for every commit that is pushed to the master branch. The build will complete and start a deployment to Azure. Navigate to Cosmos DB in the Azure portal, and you will see the CD pipeline created a new database.

Clean up resources

NOTE

Ensure you delete any unneeded resources in Azure such as the Cosmos DB instance to avoid incurring charges.

Next steps

You can optionally modify these build and release definitions to meet the needs of your team. You can also use this CI/CD pattern as a template for your other projects. You learned how to:

- Clone a sample Cosmos DB and Azure Web App to your repository
- Create a Cosmos DB collection and database
- Set up CI for your app
- Set up CD to Azure for your app
- Review the CI/CD pipeline

To learn more about Azure Pipelines, see this tutorial:

[ASP.NET MVC and Cosmos DB](#)

Glossary

10/9/2018 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015

The section lists commonly used terms for [test report](#) and [test analytics](#) in the pipeline.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

TERM	DEFINITION
Duration	Time elapsed in execution of a test , test run , or entire test execution in a build or release pipeline.
Owner	Owner of a test or test run . The test owner is typically specified as an attribute in the test code. See Publish Test Results task to view the mapping of the Owner attribute for supported test result formats.
Failing build	Reference to the build having the first occurrence of consecutive failures of a test case.
Failing release	Reference to the release having the first occurrence of consecutive failures of a test case.
Outcome	There are 13 possible outcomes for a test result: Aborted, Blocked, Error, Failed, Inconclusive, None, Not applicable, Not executed, Not impacted, Passed, Paused, Timeout, Unspecified and Warning. Some of the commonly used outcomes are: <ul style="list-style-type: none">- Aborted: Test execution terminated abruptly due to internal or external factors e.g. bad code, environment issues etc.- Failed: Test not meeting the desired outcome- Inconclusive: Test without a definitive outcome- Not executed: Test marked as skipped for execution- Not impacted: Test not impacted by the code change that triggered the pipeline- Passed: Test executed successfully- Timeout: Test execution duration exceeding the specified threshold
Flaky test	A test with non-deterministic behavior. For example, the test may result in different outcomes for the same configuration, code, or inputs.
Filter	Mechanism to search for the test results within the result set, using the available attributes. Learn more .

TERM	DEFINITION
Grouping	An aid to organizing the test results view based on available attributes such as Requirement , Test files , Priority , and more. Both test report and test analytics provide support for grouping test results.
Pass percentage	Measure of the success of test outcome for a single instance of execution or over a period of time.
Priority	Specifies the degree of importance or criticality of a test. Priority is typically specified as an attribute in the test code. See Publish Test Results task to view the mapping of the Priority attribute for supported test result formats.
Test analytics	A view of the historical test data to provide meaningful insights.
Test case	Uniquely identifies a single test within the specified branch.
Test files	Group tests based on the way they are packaged; such as files, DLLs, or other formats.
Test report	A view of single instance of test execution in the pipeline that contains details of status and help for troubleshooting, traceability, and more.
Test result	Single instance of execution of a test case with a specific outcome and details.
Test run	Logical grouping of test results based on: - Test executed using built-in tasks: All tests executed using a single task such as Visual Studio Test , Ant , Maven , Gulp , Grunt or Xcode will be reported under a single test run - Results published using Publish Test Results task: Provides an option to group all test results from one or more test results files into a single run, or individual runs per file - Tests results published using API(s): API(s) provide the flexibility to create test runs and organize test results for each run as required.
Traceability	Ability to trace forward or backward to a requirement, bug, or source code from a test result.

Help and support

Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Set up environments to run continuous test tasks with your build tasks

11/6/2018 • 13 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015

To test your app using different platforms and configurations using test automation, set up separate environments to run your app and tests with your builds in Azure Pipelines or Team Foundation Server (TFS).

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

Set up machines to run your app and tests

You'll need to set up physical or virtual machines to run your app and tests, for example:

- Windows Server 2012 R2 or higher with IIS to run your app
- Machines with the necessary browsers to run your tests

With Azure Pipelines, you can define environments that have physical and virtual machines, such as Azure VMs and Azure resource groups. With TFS, you can define environments using only physical machines. Alternatively, you can [create a virtual network isolated environment for your build-deploy-test scenarios](#).

If you want to use a PowerShell script to deploy your app, make sure to:

- Include that script in your solution or project.
- Enable PowerShell Remote on all your machines.

You'll need to install the agent that runs your tests on the machines. For more details, see [Deploy a Windows build agent](#). You might decide to [create Azure VMs](#) to install your agents.

Define a list of machines to run your app and tests

NOTE

Previous versions of Azure Pipelines and TFS included the capability to define **Machine Groups**. However, this feature is no longer available.

As an alternative, consider:

- If you use version 2.x or higher of the [Visual Studio Test](#) task you can deploy and run unit and functional tests without requiring the **Deploy Test Agent** and **Run Functional Tests** tasks, and run tests on platforms that don't have Visual Studio installed by using the [Visual Studio Test Platform](#). In this case, you can use [deployment groups](#) to define your target machines. For more details, see [Testing with unified agents and jobs](#).
- A **comma-delimited list** of machine IP addresses or fully-qualified domain names (FQDNs), together with

any port information, in all your build or release pipelines. For example:

```
dbserver.fabrikam.com,dbserver_int.fabrikam.com:5986,192.168.12.34:5986
```

- A variable that contains the list of machines. For example, a [build or release pipeline variable](#) or a variable defined within a project-wide [variable group](#). For example, you could define the variable named **MyMachines** with the value shown above, then include it in your tasks using:

```
$(MyMachines)
```

Using a variable means that you can change the list of machines in one place and have the change apply to all the tasks that use the variable.

If you don't specify a port number, the default (based on the selected protocol) will be used. If you are using HTTPS, the IP address or name of the machine should match the CN entry in the certificate. Note that you can set the **Test Certificate** option in some build, test, and deploy tasks to omit certificate checks.

Run tests in parallel

The Visual Studio Test Platform (VSTest) supports running tests in parallel. Parallel test execution is available:

- To all frameworks and within the IDE, the command line (CLI), and in Azure Pipelines.
- Within the IDE from all launch points (Test Explorer, CodeLens, various **Run** commands, and more).

Parallel test execution:

- Composes with [test adapters](#) and frameworks that already support parallel execution such as MSTest, NUnit, and xUnit.net.
- Is easy to implement, it requires no changes to existing test code and does not break existing test runs.
- Works with the test code where it already resides.
- Is OFF by default, users must explicitly opt in.
- Is supported at [assembly level](#).

Not all existing test code might be parallel-safe; for example, tests may assume exclusive use of global resources. In general, use the following iterative approach to leverage the feature:

Partition tests in terms of a taxonomy as follows:

1. Pure unit tests (typically these can run in parallel).
2. Functional tests that can run in parallel with some modifications (for example, two tests that create or delete the same folder can be adapted to use unique folders).
3. Functional tests that cannot be modified to run in parallel (for example, two Coded UI tests performing mouse actions on the desktop, or two functional tests writing to the Bluetooth or IR port).

Gradually evolve the partitioning as follows:

1. Run the tests in parallel, see which tests fail, and classify them as above.
2. Fix tests in category 2 so that they can run in parallel.
3. Move tests in category 3 into a separate test run where parallel is OFF by default.

Parallel Test Execution is **not** supported in the following cases:

- If the test run is configured using a [.testsettings](#) file.
- For test code targeting Phone, Store, UWP app platforms.

Enable parallel tests in Visual Studio 2017 Update 1 and VS Test task v1.x

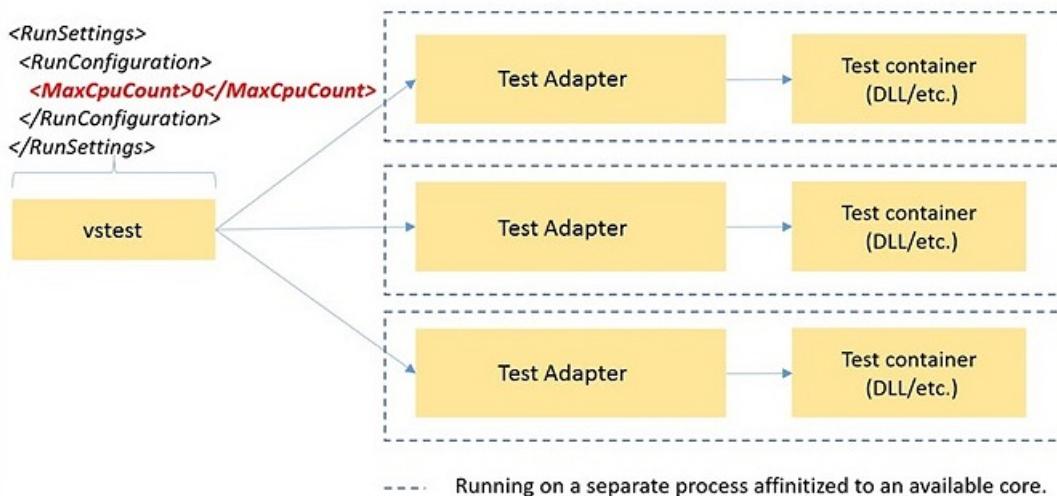
Configure a [.runsettings](#) file in the app source repository for Visual Studio IDE or the CLI, and in Azure Pipelines when using version 1.x of the [Visual Studio Test task](#).

In the [.runsettings](#) file add entry for **MaxCpuCount**, and specify or associate the file with the test run. The value for **MaxCpuCount** has the following semantics:

- **n** where $1 \leq n \leq$ number of cores: up to **n** processes will be launched.
- **n** of any other value: The number of processes launched will be as many as the available cores on the machine.
- A value of 0 (zero) indicates that up to all the available free cores may be used.

If you need to update an existing [.runsettings](#) file containing an entry for **MaxCpuCount**, see the tables in [this blog post](#) for more information.

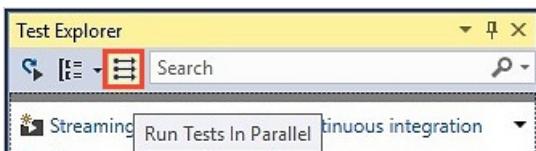
For information about in-assembly parallel test execution, see [this blog post](#).



[More information about test processes for parallel execution.](#)

Enable parallel tests in Visual Studio 2017 Update 2 and later

Enable parallel test execution by using the button on the Test Explorer toolbar. This is an ON/OFF toggle setting.



For the CLI, **vstest.console.exe** supports a **/Parallel** command line switch. Set this to enable parallel test execution.

When parallel test execution is enabled, the value of **MaxCpuCount** is set to 0 (zero) to use all the available free cores. The value is persisted with the solution and is merged with any associated [.runsettings](#) file just before a run begins. The only way to adjust the value of **MaxCpuCount** is by [explicitly adding a .runsettings file](#).

If you need to update an existing [.runsettings](#) file containing an entry for **MaxCpuCount**, see the tables in [this blog post](#) for more information.

For information about in-assembly parallel test execution, see [this blog post](#).

Enable parallel tests in Azure Pipelines with VS Test task v2.x

Enable parallel test execution by setting the **Run Tests in Parallel...** checkbox in the settings for the [Visual Studio Test task](#).

Execution options ^

Select test platform using

Version Specific location

Test platform version ⓘ

Latest

Settings file ⓘ

Override test run parameters ⓘ

Path to custom test adapters ⓘ

Run tests in parallel on multi-core machines ⓘ

Run tests in isolation ⓘ

Code coverage enabled ⓘ

When parallel test execution is enabled, the value of **MaxCpuCount** is set to 0 (zero) to use all the available free cores. The value is persisted with the solution and is merged with any associated [.runsettings file](#) just before a run begins. The only way to adjust the value of **MaxCpuCount** is by [explicitly adding a .runsettings file](#).

If you need to update an existing [.runsettings](#) file containing an entry for **MaxCpuCount**, see the tables in [this blog post](#) for more information.

For information about in-assembly parallel test execution, see [this blog post](#).

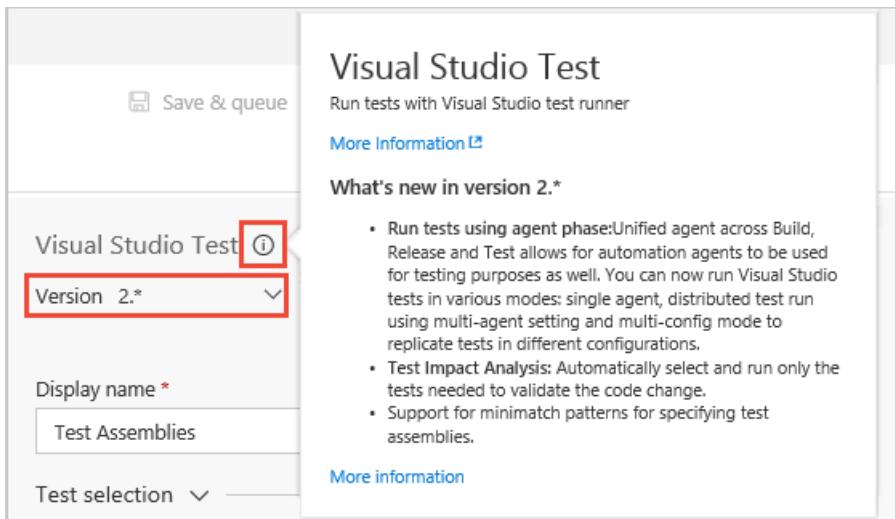
Testing with unified agents and jobs

Version 2 of the **Visual Studio Test** task uses the unified Build and Release agent, instead of a different custom agent as was the case with version 1. The new version of the task also integrates intelligently with [task jobs](#). This topic explores how you can use this task, and explains how it works in a range of scenarios.

For more information about the tasks see:

- [Visual Studio Test version 1](#)
- [Visual Studio Test version 2](#)

You select the [specific version](#) of a task you want to use in the **Version** list at the top of the task properties pane. Use the **i** icon to show more information about the task or a selected property setting.



Visual Studio Test

Run tests with Visual Studio test runner

[More Information](#)

What's new in version 2.*

- Run tests using agent phase:Unified agent across Build, Release and Test allows for automation agents to be used for testing purposes as well. You can now run Visual Studio tests in various modes: single agent, distributed test run using multi-agent setting and multi-config mode to replicate tests in different configurations.
- Test Impact Analysis: Automatically select and run only the tests needed to validate the code change.
- Support for minimatch patterns for specifying test assemblies.

[More information](#)

Visual Studio Test (i)

Version 2.*

Display name *

Test Assemblies

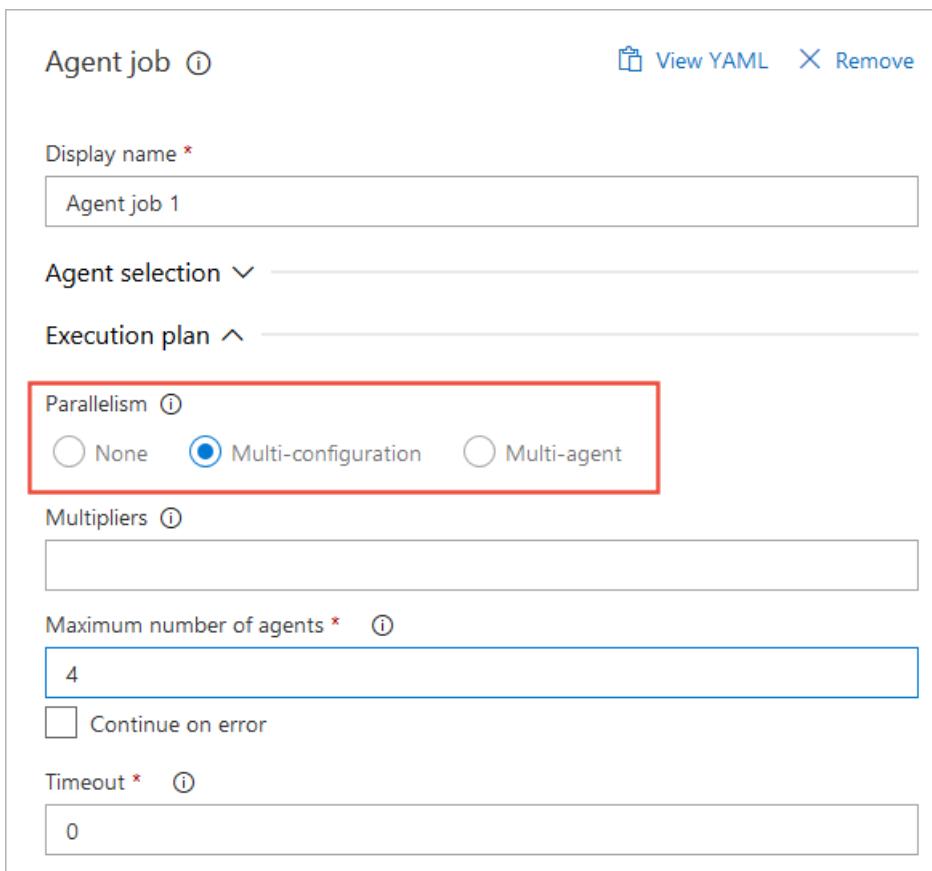
Test selection ▾

Advantages of using the unified agent

- You no longer need to use dedicated machines for testing (as was required by the **Run Functional Tests** task). With the unified agent, you can leverage the common agent pool. Administrators can set up a reusable pool of machines, making management much easier.
- You can use the unified agent for single machine as well multi-machine distributed execution.
- You no longer need the **Visual Studio Test Agent Deployment** task. This task is based on WinRM, which imposes several limitations.
- You no longer need any "copy files" tasks because all execution is now local to the automation agent, and task jobs download the artifacts to the target machines automatically. There is no requirement to copy test assemblies and their dependencies when running tests remotely using the **Run Functional Tests** task.

How test tasks run in jobs

You can add [different types of jobs](#) to a release pipeline. The properties of these jobs include settings for **Parallelism**.



Agent job (i)

[View YAML](#) X Remove

Display name *

Agent job 1

Agent selection ▾

Execution plan ▾

Parallelism (i)

None Multi-configuration Multi-agent

Multipliers (i)

Maximum number of agents * (i)

4

Continue on error

Timeout * (i)

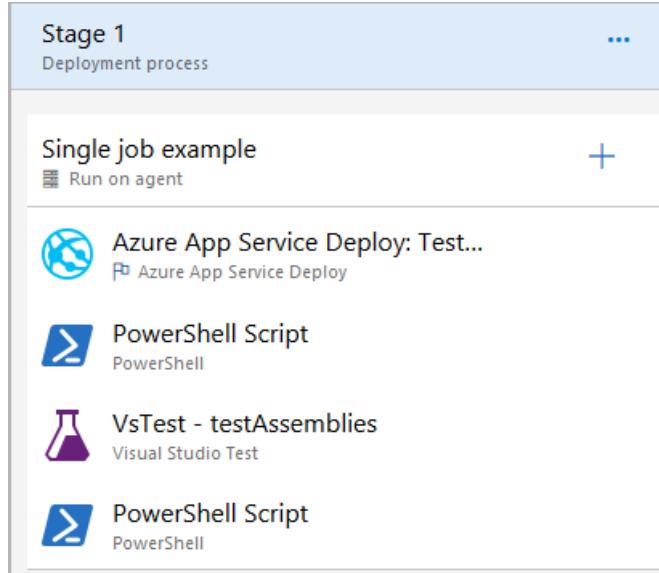
0

The following sections describe how this setting affects the operation of the **Visual Studio Test** and **Run Functional Tests** tasks. For a full description of the operation for all tasks, see [Parallel execution using agent jobs](#).

No parallelism

A single agent from the specified pool will be allocated to this job. This is the default, and all tasks in the job will run on that agent. The **Visual Studio Test** task runs in exactly the same way as version 1 with single agent test execution.

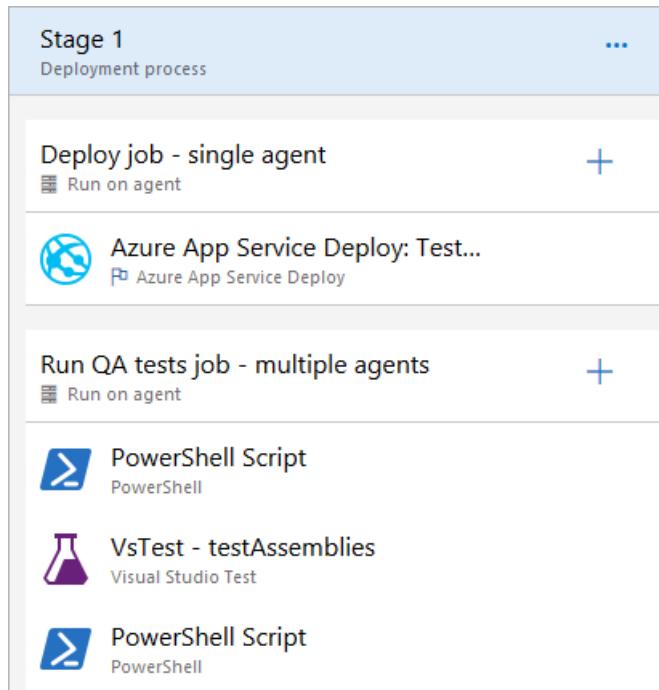
For example, you could deploy an Azure Web App and run a small number of quick tests on it (for which a single agent is sufficient), along with some pre- and post-test setup and cleanup activities, using a pipeline configured as follows:



Multiple executions

This mode is driven by 'multipliers', in much the same way as a multi-configuration Build or Release. You define the multipliers as variables. Based on the values for these variables, the various configurations are run.

In the case of Build, you typically use **BuildPlatform** and **BuildConfiguration** as multipliers. The same logic applies to testing. For example, you could deploy a web app to Azure and run cross-browser tests on IE and Firefox by configuring a pipeline to use two jobs - a deploy job and a test job:



The test job is set up as a multiple executions process using a variable named **Browser**, which has the values `Edge`

and `Firefox`. The job will run twice using these two configurations - one agent is assigned the value `Edge` for its **Browser** variable, and one with the value `Firefox`.

Agent job ⓘ

Display name *

Run QA tests job - multiple agents

Agent selection ▾

Execution plan ^

Parallelism ⓘ

None Multi-configuration Multi-agent

Multipliers ⓘ

Browser

Maximum number of agents * ⓘ

2

Continue on error

Name	Value
Browser	Edge, Firefox

In the tasks for the pipeline, the **Browser** value could be used to instantiate the appropriate browser for the tests. For example, you might pass the values as **Test Run Parameters** and access them using **TestContext** in the test code.

All pipelines > ⚡ New release pipeline

Pipeline Tasks Variables Retention Options History

Stage 1 Deployment process

Deploy job - single agent + Run on agent

Azure App Service Deploy... Azure App Service Deploy

Run QA tests job - multiple ag... + Run on agent

PowerShell Script PowerShell

VsTest - testAssemblies Visual Studio Test

Execution options ^

Select test platform using

Version Specific location

Test platform version ⓘ

Latest

Settings file ⓘ

`$(System.DefaultWorkingDirectory)/Files/SettingsFile`

Override test run parameters ⓘ

`browser=$(Browser)`

You could also use the values to provide appropriate titles for your test runs so that, if a test fails in a particular configuration, you can easily tell which run it came from.

Multiple agents

Multiple agents will be allocated to the job. You specify the number of agents to be allocated from the pool, and the set of tasks in that job will be distributed across all these agents.

In this mode, the Visual Studio Test task runs in a special way. It recognizes that it's a multiple agents job, and runs tests in a distributed manner across all the allocated agents. Because other tasks run across all agents, any pre- and post-test tasks also run equally on all the agents. Therefore, all the agents are prepared and cleaned up in a consistent manner. In addition, test execution does not require all agents to be available at the same time. If some agents are busy with another release or build, the job can still start with the available number of agents that match the demand, and test execution starts. As additional agents become available, they can pick up any remaining tests that have not yet run.

Artifacts are automatically downloaded when the job starts, so the test assemblies and other files are already located on the agent, and no "copy files" task is required. So, to publish an Azure Web App and run a large number of tests with fast test execution, you could model the pipeline as two jobs - one being the deploy job (which runs on a single agent because you don't want multiple agents to deploy the same app concurrently), and the other a test job that uses multiple agents mode to achieve test distribution.

This also means that you can use different agent pools for the two jobs, allowing you to manage agents for

different purposes separately if required.

FAQs

Q: How do I do this with Build?

A: The jobs capability is currently available only in Azure Pipelines releases. It will become available for builds soon.

Q: Does the Visual Studio Test version 1 task behave the same way as the version 2 task?

A: No, the version 1 task cannot be used for test distribution. On the single agent (the default, no parallelism) setting, the task will run in the same way as on the previous test agent. In the multiple executions and multiple agents modes, it is replicated on the agents, in the same way as all other tasks.

Q: Can I run UI tests on the Microsoft-hosted agents?

A: Yes, see [UI testing considerations](#).

Q: What is required to run UI tests?

A: See [UI testing considerations](#).

Q: What does the 'Test mix contains UI tests' checkbox do?

A: Currently, it is there only as a reminder to run agents interactively if you are running UI tests. If you are using an agent pool with a mix of interactive and 'running as service' agents, you may also want to add an 'Interactive' capability to your agents demand that in your test job to ensure the appropriate set of agents that can run UI tests are used. See [Build and Release agent capabilities](#).

Q: In multiple executions mode, do I get distribution of tests as well?

A: No, multiple executions mode assigns only one agent per configuration.

Q: How do I map the configuration in multiple executions mode to my Test Configurations using tcm.exe?

A: Currently this is not possible.

Q: How else can I use multiple executions mode?

A: This mode can be used whenever you need multiple agents to execute jobs in parallel. For more examples, see [Parallel execution using agent jobs](#).

Q: Has the Run Functional Tests task also changed?

A: No, the Run Functional Tests (RFT) task has not changed. If you are using this task you *do* need the **Deploy Test Agent** task. Note that, because tasks are replicated in the multiple agents and multiple executions mode, using the Run Functional Tests task in this mode will lead to undesirable effects.

Q: Do I need to install Visual Studio on all the machines to use the Visual Studio Test version 2 task?

A: Currently, yes. An alternate means for running tests without requiring Visual Studio on the agent machine is under consideration, which will allow you to create a separate pool of agents for testing purposes.

Q: I am using my own test runner (not the Visual Studio Test task) in the pipeline. What happens to it?

A: In the multiple agents and multiple executions mode, the task will be replicated on each of the agents. You can use the multiple executions mode to partition your tests on different configuration using the configuration variable. For example, if you have a configuration variable named **Platform** that has values `x86` and `x64`, you can run the two sets of tests on two agents in parallel by referring to your test assemblies using `**\$(Platform)*test*.dll`

Q: How does the Visual Studio Test version 2 task run on deployment groups?

A: Yes, the task can be used to run on [deployment groups](#).

If you have scenarios that necessitate running tests on machines in the deployment group where the app is deployed, you can use the Visual Studio Test version 2 task. If multiple machines are selected (using, for example, tags) in a **Run on Deployment Group** job, the tests will be replicated to each of the machines.

See Also

- [MSTest V2 in-assembly parallel test execution](#)

- Parallel Test Execution
- Create a virtual network isolated environment for build-deploy-test scenarios
- Speed up testing with Test Impact Analysis

Help and support

Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Run unit tests with your builds

10/18/2018 • 2 minutes to read • [Edit Online](#)

VS 2017 | VS 2015 | Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Make sure that your app still works after every check-in and build using Azure Pipelines or Team Foundation Server (TFS) by using test automation. Find problems earlier by running tests automatically with each build. When your build is done, review your test results to start resolving the problems that you find.

Typically you will run unit tests in your build workflow, and functional tests in your release workflow after your app is deployed (usually to a QA environment). Code coverage is available only in the build workflow.

NOTE

This example shows how to run unit tests with your build for .NET and ASP.NET apps. It uses the [Visual Studio Test](#) task. For information about running test for other app types such as [.NET Core](#) and [Python](#), see the topics in the [Languages](#) section of the Azure Pipelines documentation.

Before you start

- [Check in your solution](#) to Azure Pipelines. Include your test projects.

Create a build pipeline

Your build pipeline must include a test task that runs unit tests. For example, if you're building a Visual Studio solution in Azure Pipelines, your build pipeline should include a **Visual Studio Test** task. After your build starts, this task automatically runs all the unit tests in your solution - on the same build machine.

1. If your build pipeline does not contain a test task, add one to it.

The screenshot shows the Azure DevOps Pipeline editor. On the left, a sidebar lists the pipeline stages: 'Pipeline' (Build pipeline), 'Get sources' (DotNetSample, master), 'Agent job 1' (Run on agent), 'Use NuGet 4.4.1' (NuGet Tool Installer), and 'NuGet restore' (NuGet). A red box highlights the '+' icon next to 'Agent job 1'. On the right, there's an 'Add tasks' button, a 'Refresh' button, and a search bar with 'test'. Below the search bar, a detailed description of the 'Visual Studio Test' task is shown, including its purpose and supported frameworks.

2. Edit the Visual Studio Test task. You can add filter criteria to run specific tests, enable code coverage, run tests from [other unit test frameworks](#), and so on.

The screenshot shows the configuration of the 'Visual Studio Test' task. The task is set to version 2.*. The 'Display name' is 'VsTest - testAssemblies'. The 'Test selection' dropdown is set to 'Select tests using' and 'Test assemblies'. The 'Test assemblies' field contains the pattern '**\\$(BuildConfiguration)*test*.dll !**\obj**'. The 'Search folder' is set to '\$(System.DefaultWorkingDirectory)'. At the bottom, there are two optional checkboxes: 'Run only impacted tests' and 'Test mix contains UI tests'.

The Visual Studio Test task version 2 supports [Test Impact Analysis](#). For information about all the task settings, see [Visual Studio Test task](#).

[How do I pass parameters to my test code from a build pipeline?](#)

3. If you also want to test code coverage, set the **Code coverage enabled** checkbox in the **Execution**

options section.

The screenshot shows the 'Execution options' configuration for a build pipeline. On the left, under 'Process', there is a list of tasks: 'Get sources', 'NuGet restore', 'Build solution', 'VsTest - testAssemblies' (which is selected and highlighted with a blue border), 'Publish symbols path', 'Copy Files to', 'Publish Artifact: drop', and '+ Add Task'. To the right, under 'Execution options', there are several settings:

- 'Test mix contains UI tests': An unchecked checkbox.
- 'Execution options ^': A collapsed section header.
- 'Select test platform using': A radio button group where 'Version' is selected (radio button is filled) and 'Specific location' is unselected (radio button is empty).
- 'Test platform version': A dropdown menu showing 'Visual Studio 2015'.
- 'Settings file': A text input field.
- 'Override test run parameters': A text input field.
- 'Path to custom test adapters': A text input field.
- 'Run tests in parallel on multi-core machines': An unchecked checkbox.
- 'Run tests in isolation': An unchecked checkbox.
- 'Code coverage enabled': A checked checkbox (checkbox is filled). This checkbox is highlighted with a red rectangle.
- 'Other console options': A text input field.
- 'Rerun failed tests': An unchecked checkbox.
- 'Advanced execution options': A collapsed section header.

When tests are run with this option, code coverage information is collected dynamically and assemblies do not need to be instrumented. By default, all assemblies are profiled for collecting coverage information. If you need to [exclude specific assemblies and customize code coverage](#), use a [.runsettings file](#).

[How do I collect and publish code coverage data if I'm not using the Visual Studio Test task?](#)

4. When you're done, save your build pipeline.

The screenshot shows the 'Fabrikam-.NET Desktop-CI (1)' build pipeline in the editor. The 'Tasks' tab is selected. In the top right, there is a 'Save & queue' button with a dropdown arrow. A context menu is open, showing three options: 'Save' (which is highlighted with a red rectangle), 'Save & queue', and 'Save as draft'. The 'Save' option is the primary target of the red box.

Start the build

1. Start the build by adding it to the build queue.

Task Groups Deployment Groups*

Desktop-CI (1) Save & queue Discard Summary Queue ...

Options R Queue build for Fabrikam-.NET Desktop-CI (1)

Agent queue Hosted

Branch master

Commit

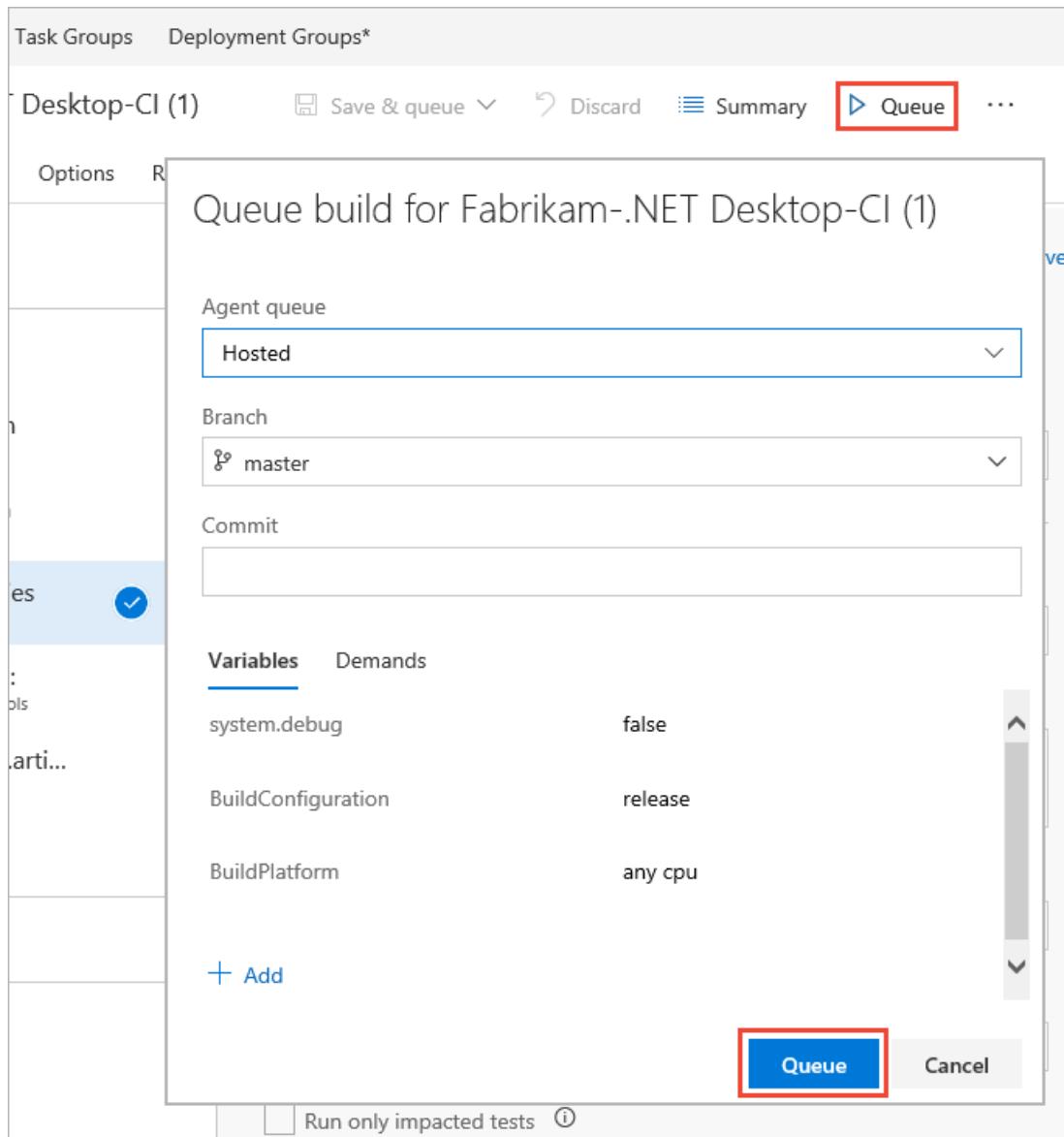
Variables Demands

system.debug	false
BuildConfiguration	release
BuildPlatform	any cpu

+ Add

Queue Cancel

Run only impacted tests ⓘ



- After the build finishes, you can review the test results to resolve any problems that happened. Go to the build summary and open the **Tests** page.

DotNetSample-ASP.NET 20180823.1

DotNetSample · master · ce44f0d : Add a multistage Dockerfile · Manual build · Retained by release

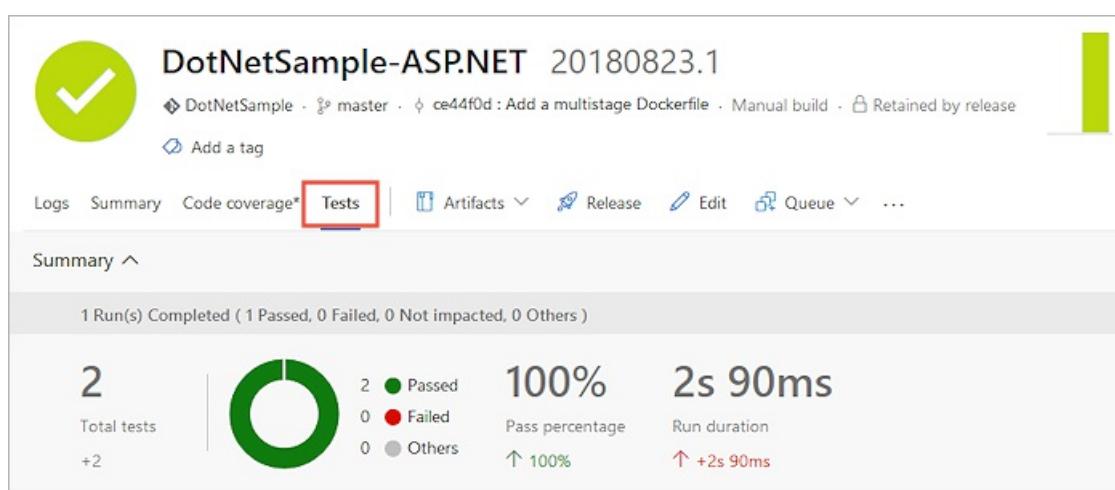
Add a tag

Logs Summary Code coverage* Tests Artifacts Release Edit Queue ...

Summary ^

1 Run(s) Completed (1 Passed, 0 Failed, 0 Not impacted, 0 Others)

2 Total tests +2	Passed 2	Failed 0	Others 0
100% Pass percentage ↑ 100%		2s 90ms Run duration ↑ +2s 90ms	



Next step

[Review your test results](#)

UI test with Selenium

10/19/2018 • 5 minutes to read • [Edit Online](#)

[VS 2017](#) | [VS 2015](#) | [Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

Performing user interface (UI) testing as part of the release pipeline is a great way of detecting unexpected changes, and need not be difficult. This topic describes using Selenium to test your website during a continuous deployment release and test automation.

Typically you will run unit tests in your build workflow, and functional (UI) tests in your release workflow after your app is deployed (usually to a QA environment).

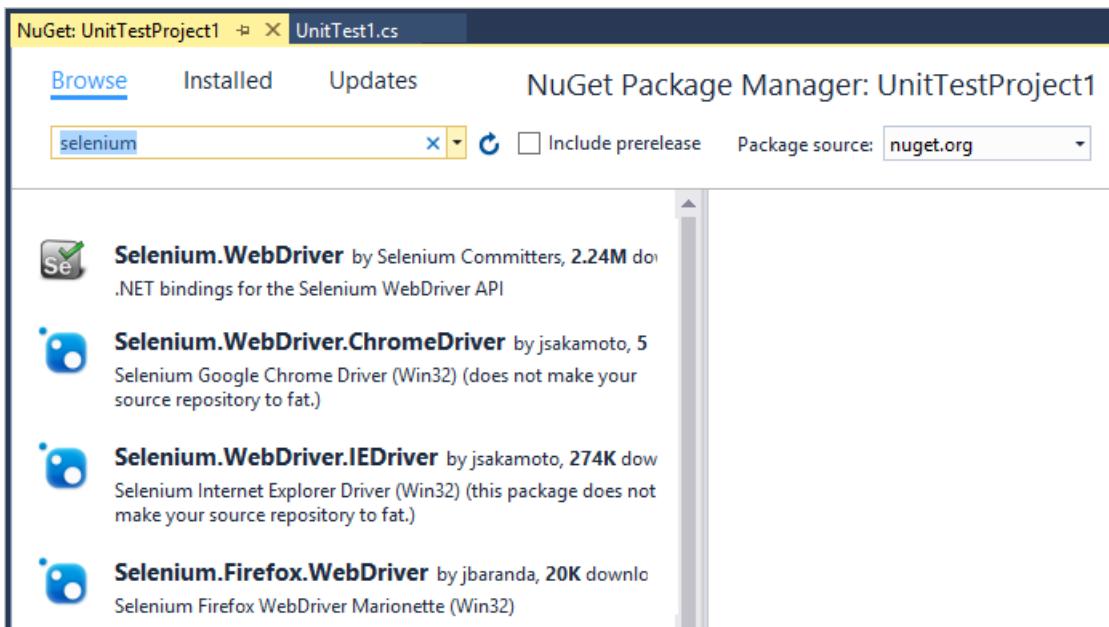
For more information about Selenium browser automation, see:

- [Selenium HQ](#)
- [Selenium documentation](#)

Create your test project

As there is no template for Selenium testing, the easiest way to get started is to use the Unit Test template. This automatically adds the test framework references and enables you run and view the results from Visual Studio Test Explorer.

1. In Visual Studio, open the **File** menu and choose **New Project**, then choose **Test** and select **Unit Test Project**. Alternatively, open the shortcut menu for the solution and choose **Add** then **New Project** and then **Unit Test Project**.
2. After the project is created, add the Selenium and browser driver references used by the browser to execute the tests. Open the shortcut menu for the Unit Test project and choose **Manage NuGet Packages**. Add the following packages to your project:
 - Selenium.WebDriver
 - Selenium.Firefox.WebDriver
 - Selenium.WebDriver.ChromeDriver
 - Selenium.WebDriver.IEDriver



3. Create your tests. For example, the following code creates a default class named **MySeleniumTests** that performs a simple test on the Bing.com website. Replace the contents of the **TheBingSearchTest** function with the [Selenium code](#) required to test your web app or website. Change the **browser** assignment in the **SetupTest** function to the browser you want to use for the test.

```
using System;
using System.Text;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using OpenQA.Selenium;
using OpenQA.Selenium.Firefox;
using OpenQA.Selenium.Chrome;
using OpenQA.Selenium.IE;

namespace SeleniumBingTests
{
    /// <summary>
    /// Summary description for MySeleniumTests
    /// </summary>
    [TestClass]
    public class MySeleniumTests
    {
        private TestContext testContextInstance;
        private IWebDriver driver;
        private string appURL;

        public MySeleniumTests()
        {
        }

        [TestMethod]
        [TestCategory("Chrome")]
        public void TheBingSearchTest()
        {
            driver.Navigate().GoToUrl(appURL + "/");
            driver.FindElement(By.Id("sb_form_q")).SendKeys("Azure Pipelines");
            driver.FindElement(By.Id("sb_form_go")).Click();
            driver.FindElement(By.XPath("//ol[@id='b_results']/li/h2/a/strong[3]")).Click();
            Assert.IsTrue(driver.Title.Contains("Azure Pipelines"), "Verified title of the page");
        }

        /// <summary>
        ///Gets or sets the test context which provides
        ///information about and functionality for the current test run.
        ///</summary>
        public TestContext TestContext
```

```

    {
        get
        {
            return testContextInstance;
        }
        set
        {
            testContextInstance = value;
        }
    }

    [TestInitialize()]
    public void SetupTest()
    {
        appURL = "http://www.bing.com/";

        string browser = "Chrome";
        switch(browser)
        {
            case "Chrome":
                driver = new ChromeDriver();
                break;
            case "Firefox":
                driver = new FirefoxDriver();
                break;
            case "IE":
                driver = new InternetExplorerDriver();
                break;
            default:
                driver = new ChromeDriver();
                break;
        }
    }

    [TestCleanup()]
    public void MyTestCleanup()
    {
        driver.Quit();
    }
}

```

- Run the Selenium test locally using Test Explorer and check that it works.

Define your CI build pipeline

You'll need a continuous integration (CI) build pipeline that builds your Selenium tests. For more details, see [Build your .NET desktop app for Windows](#).

Create your web app

You'll need a web app to test. You can use an existing app, or deploy one in your CD release pipeline. The example code above runs tests against Bing.com. For details of how to set up your own release pipeline to deploy a web app, see [Deploy to Azure Web Apps](#).

Decide how you will deploy and test your app

You can deploy and test your app using either the Microsoft-hosted agent in Azure, or a self-hosted agent that you install on the target servers.

- When using the **Microsoft-hosted agent**, you should use the Selenium web drivers that are pre-installed on the Windows agents (agents named **Hosted VS 20xx**) because they are compatible with the browser

versions installed on the Microsoft-hosted agent images. The file paths to these drivers can be obtained from the environment variables named `IEDriver` (Internet Explorer), `ChromeDriver` (Google Chrome), and `GeckoDriver` (Firefox). For example,

```
driver = new ChromeDriver(Environment.GetEnvironmentVariable("ChromeDriver"));
```

The drivers are **not** pre-installed on other agents such as Linux, Ubuntu, and macOS agents.

- When using a **self-hosted agent** that you deploy on your target servers, agents must be configured to run interactively with auto-logon enabled. See [Build and release agents](#).

Include the test in a CD release

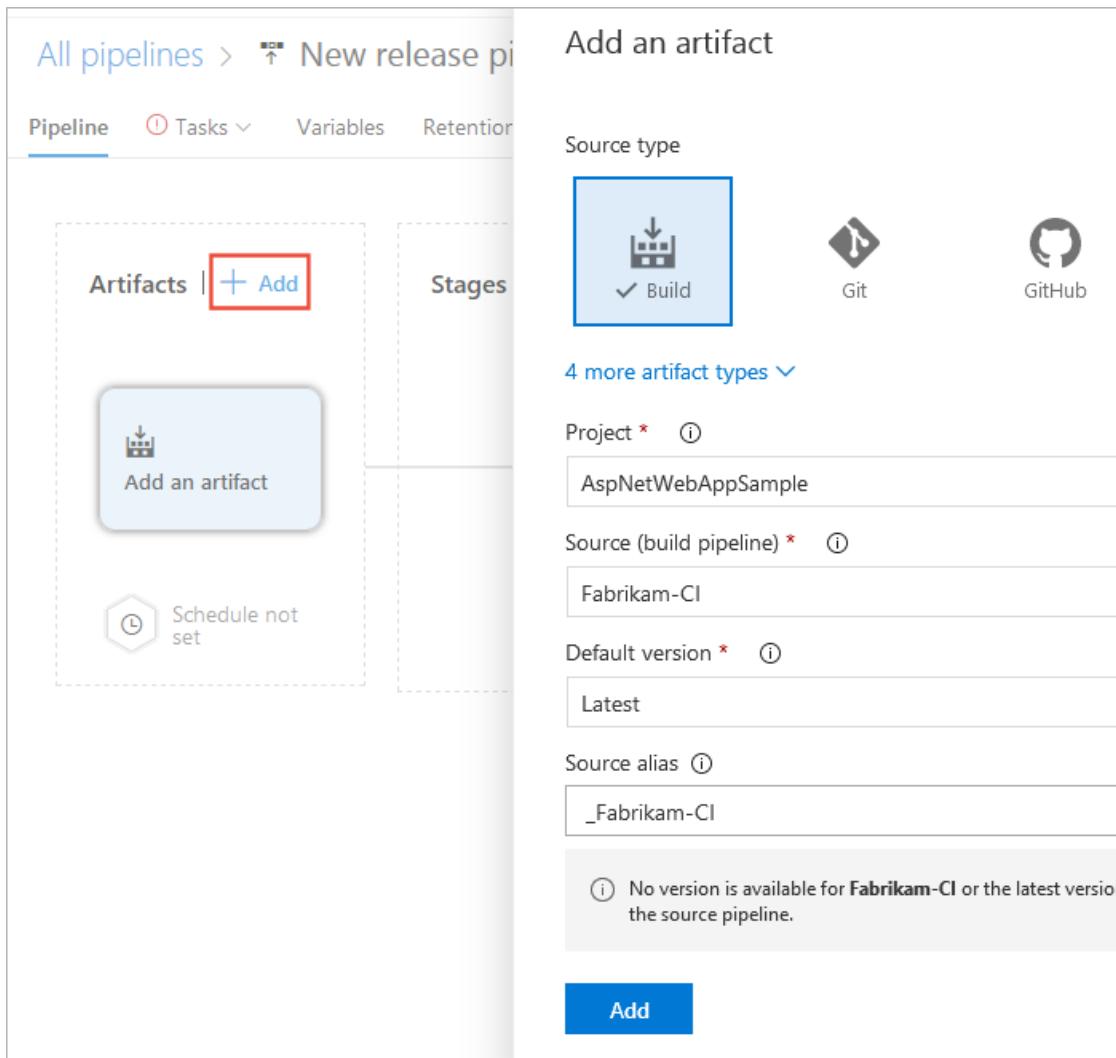
NOTE: This example uses the **Visual Studio Test Platform Installer** task and the latest version of the **Visual Studio Test** task. These tasks are not available in TFS 2015 or TFS 2017. To run Selenium tests in these versions of TFS, you must use the [Visual Studio Test Agent Deployment](#) and [Run Functional Tests](#) tasks instead.

- If you don't have an existing release pipeline that deploys your web app:

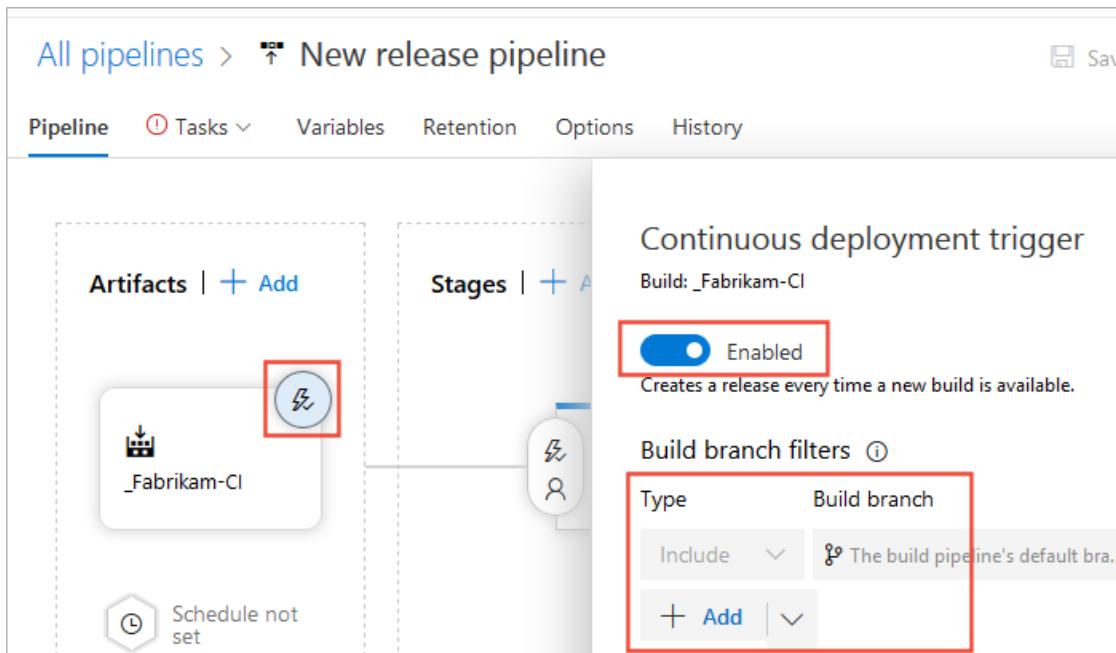
- Open the **Releases** page in the **Azure Pipelines** section in Azure DevOps or the **Build & Release** hub in TFS (see [Web portal navigation](#)) and choose the **+** icon, then choose **Create release pipeline**.



- Select the **Azure App Service Deployment** template and choose **Apply**.
- In the **Artifacts** section of the **Pipeline** tab, choose **+ Add**. Select your build artifacts and choose **Add**.



- Choose the **Continuous deployment trigger** icon in the **Artifacts** section of the **Pipeline** tab. In the Continuous deployment trigger pane, enable the trigger so that a new release is created from every build. Add a filter for the default branch.



- Open the **Tasks** tab, select the **Stage 1** section, and enter your subscription information and the name of the web app where you want to deploy the app and tests. These settings are applied to the **Deploy Azure App Service** task.

All pipelines > **New release pipeline**

Pipeline Tasks Variables Retention Options History

Stage 1
Deployment process

Run on agent +
Run on agent

Deploy Azure App Service
Azure App Service Deploy

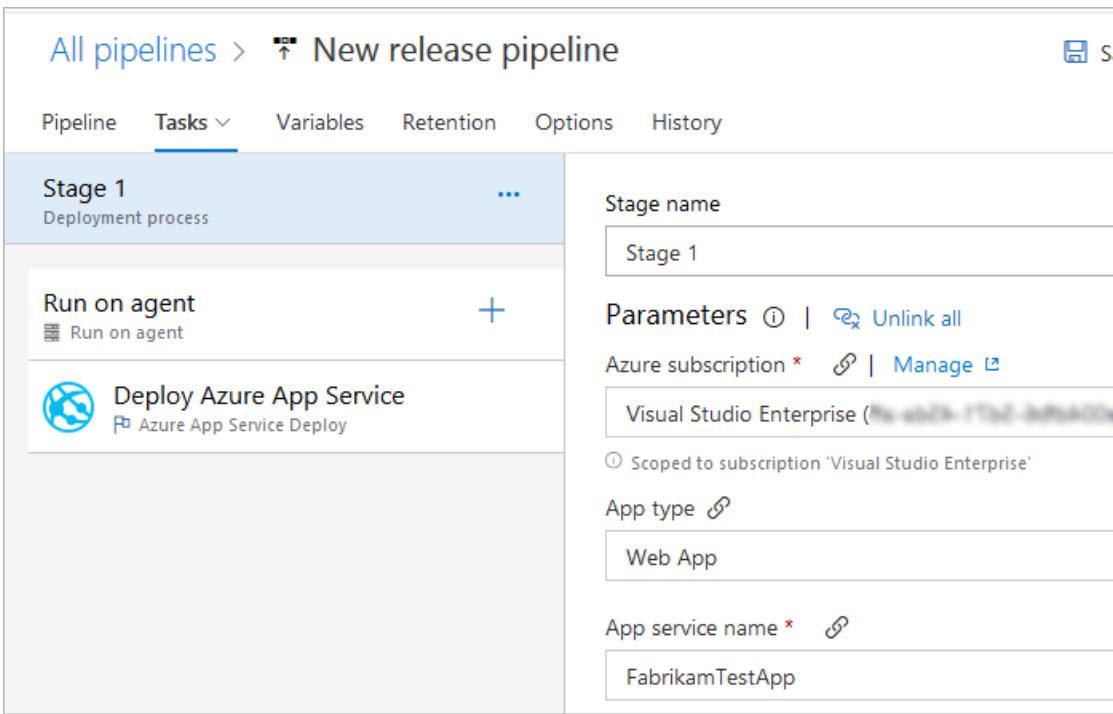
Stage name: Stage 1

Parameters: [Unlink all](#)

Azure subscription: Visual Studio Enterprise (Visual Studio Test Platform)

App type: Web App

App service name: FabrikamTestApp



- If you are deploying your app and tests to environments where the target machines that host the agents do not have Visual Studio installed:
 - In the **Tasks** tab of the release pipeline, choose the + icon in the **Run on agent** section. Select the **Visual Studio Test Platform Installer** task and choose **Add**. Leave all the settings at the default values.

All pipelines > **New release pipeline**

Pipeline Tasks Variables Retention Options History

Stage 1
Deployment process

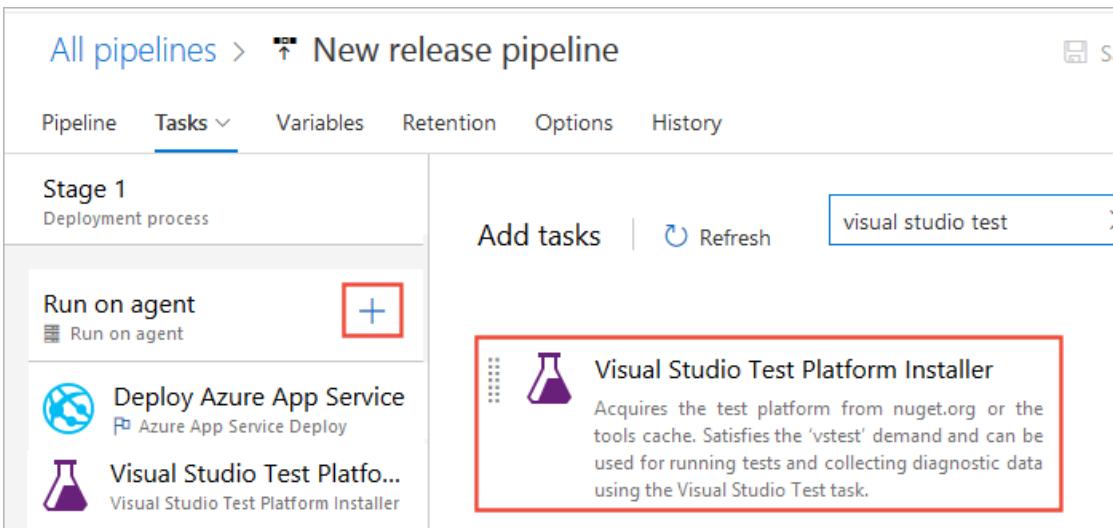
Run on agent +
Run on agent

Deploy Azure App Service
Azure App Service Deploy

Visual Studio Test Platfo...
Visual Studio Test Platform Installer

Add tasks | Refresh visual studio test

Visual Studio Test Platform Installer
Acquires the test platform from nuget.org or the tools cache. Satisfies the 'vstest' demand and can be used for running tests and collecting diagnostic data using the Visual Studio Test task.



You can find a task more easily by using the search textbox.

- In the **Tasks** tab of the release pipeline, choose the + icon in the **Run on agent** section. Select the **Visual Studio Test** task and choose **Add**.

All pipelines > **New release pipeline**

Pipeline Tasks Variables Retention Options History

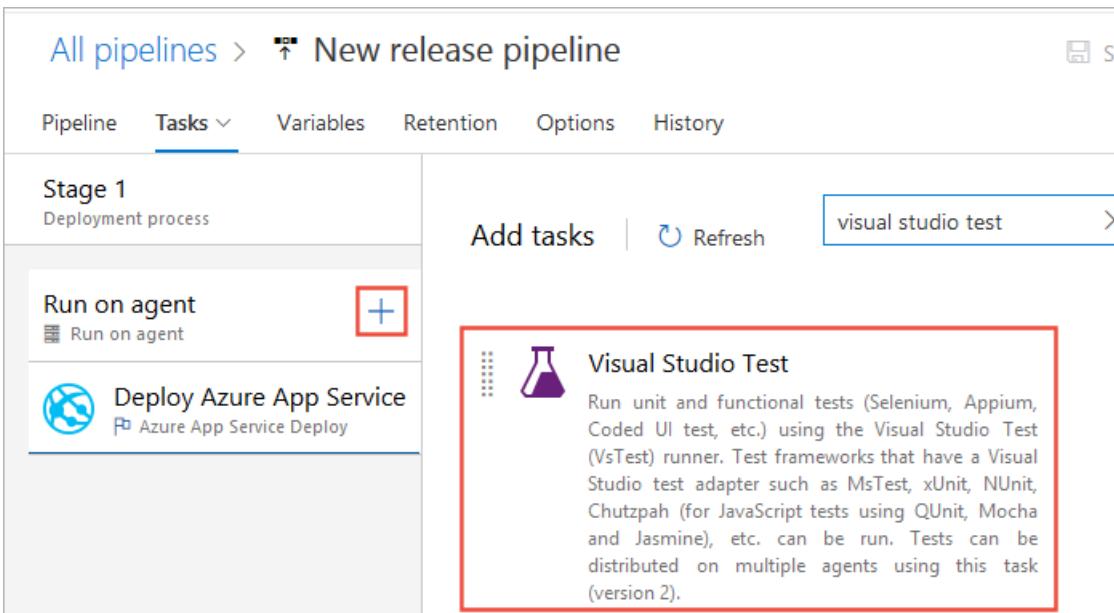
Stage 1
Deployment process

Run on agent **+ Run on agent**

Deploy Azure App Service **Azure App Service Deploy**

Add tasks Refresh visual studio test

Visual Studio Test
Run unit and functional tests (Selenium, Appium, Coded UI test, etc.) using the Visual Studio Test (VsTest) runner. Test frameworks that have a Visual Studio test adapter such as MsTest, xUnit, NUnit, Chutzpah (for JavaScript tests using QUnit, Mocha and Jasmine), etc. can be run. Tests can be distributed on multiple agents using this task (version 2).



4. If you added the **Visual Studio Test Platform Installer** task to your pipeline, change the **Test platform version** setting in the **Execution options** section of the **Visual Studio Test** task to **Installed by Tools Installer**.

All pipelines > **New release pipeline**

Pipeline Tasks Variables Retention Options History

Stage 1
Deployment process

Run on agent **+ Run on agent**

Deploy Azure App Service **Azure App Service Deploy**

Visual Studio Test Platfo... **Visual Studio Test Platform Installer**

VsTest - testAssemblies **Visual Studio Test**

Execution options

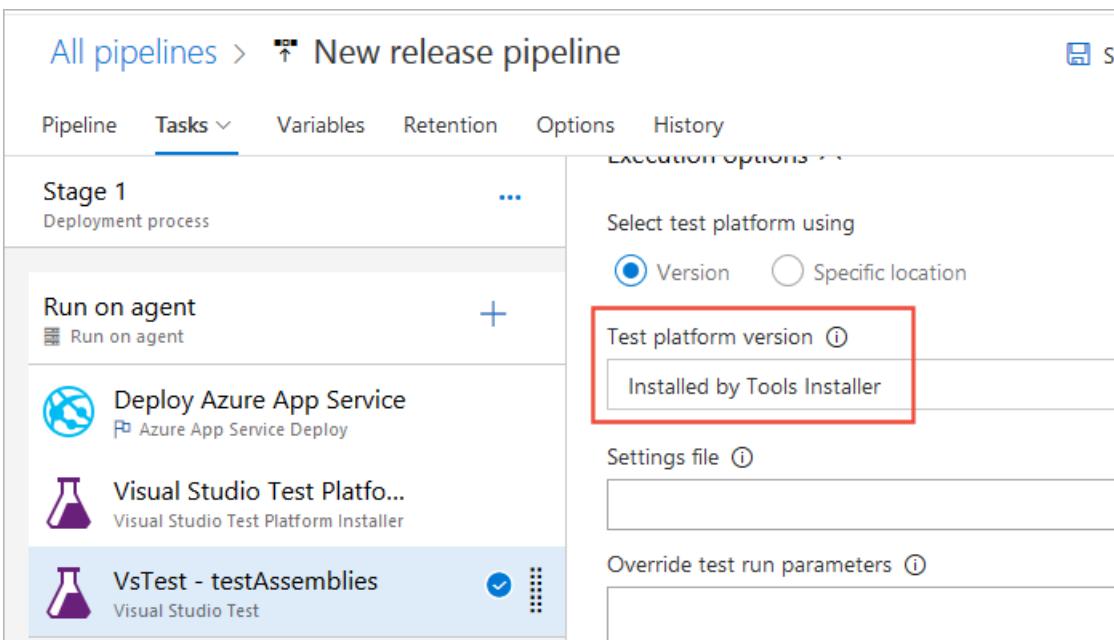
Select test platform using

Version Specific location

Test platform version **①**
Installed by Tools Installer

Settings file **①**

Override test run parameters **①**



[How do I pass parameters to my test code from a release pipeline?](#)

5. Save the release pipeline and start a new release. You can do this by queuing a new CI build, or by choosing **Create release** from the **Release** drop-down list in the release pipeline.

New release pipeline

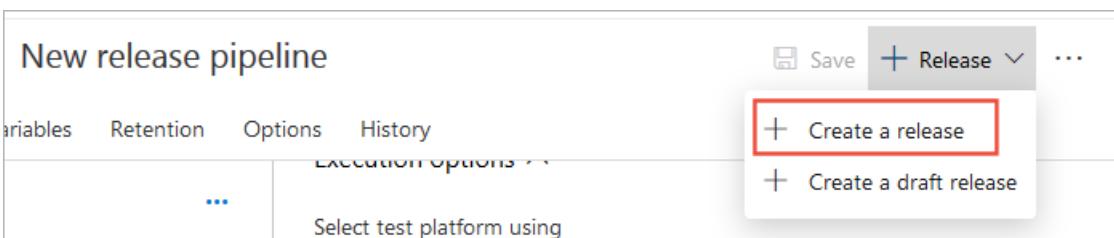
Variables Retention Options History

Execution options

Select test platform using

+ Create a release

+ Create a draft release



6. To view the test results, open the release summary from the **Releases** page and choose the **Tests** link.

Next steps

[Review your test results](#)

Review test results

10/15/2018 • 7 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

Automated tests can be configured to run as part of a build or release for various [languages](#). Test reports provide an effective and consistent way to view the tests results executed using different test frameworks, in order to measure pipeline quality, review traceability, troubleshoot failures and drive failure ownership. In addition, it provides many advanced reporting capabilities explored in the following sections.

Read the [glossary](#) to understand test report terminology.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Published test results can be viewed in the **Tests** tab in a build or release summary.

NOTE

Test report is available in TFS 2015 and above, however the new experience described in this document is available only with Azure Pipelines at present.

View test results in build

The build summary provides a timeline view of the key steps executed in the build. If tests were executed and reported as part of the build, a test milestone appears in the timeline view. The test milestone provides a summary of the test results as a measure of **pass percentage** along with indicators for **failures** and **aborts** if these exist.



VSO.PR VSO.PR_20180802.153

◆ VSO . ⌛ master . 🏷 PR 370129 : cancel autocomplete on retarget . Pull request validation

1406 1316 1314 1744 2218

Logs Timeline Code coverage* Accessibility* Scan Results* Tests Build Targets

Progression



Build artifacts published ▾

2



Tests succeeded ▾

100% passed



Build process succeeded ▲

4 error(s) / 0 warning(s)

- ✗ Toolsets\RunTests\ExecuteL0Tests.Leaf.targets(18,5): Error : Error: Error: Timeout occurred when
- ✗ Toolsets\RunTests\ExecuteL0Tests.Leaf.targets(18,5): Error : While Running:d:\v2.0\P1_work\1\s\Tfs\Service\WebAccess\Wiki\TestScripts\Tests\AdminSecuritySourceTests.ts
- ✗ Toolsets\RunTests\ExecuteL0Tests.Leaf.targets(18,5): Error :
- ✗ Toolsets\RunTests\ExecuteL0Tests.Leaf.targets(18,5): Error : Test Run Failed.

View test results in release

In the pipeline view you can see all the stages and associated tests. The view provides a summary of the test results as a measure of **pass percentage** along with indicators for **failures** and **aborts** if these exist. These indicators are same as in the build timeline view, giving a consistent experience across build and release.

The screenshot shows the VSO Features CI pipeline interface. At the top, there's a header with a back arrow, the text "Tfs.SelfHost - VSO.Features.CI > VSO.Features.CI_20170723.5.1", and a dropdown menu. Below the header are navigation links: Pipeline, Variables, History, Deploy, Cancel, Refresh, and Release (old view). The main area is divided into two sections: "Release" on the left and "Stages" on the right.

Release section:

- Manually triggered by Liang Zhu on 7/23/2017 11:51 PM
- Artifacts:
 - VSO.CI (VSO.CI_20170723.9) from master
 - VSO.Features.CI (VSO.Features.CI_20170723.5) from features/spoobpcremoval

Stages section:

- Tfs.SelfHost Set 1**: Failed (red border).
 - Validate Test Results task failed on 7/24/2017 1:23 AM.
 - 99% pass rate, 3 failures.
- Tfs.SelfHost Set 2**: Succeeded (green border).
 - on 7/24/2017 12:26 AM.
 - 100% pass rate.

Tests tab

Both the build and release summaries provide details of test execution. Choose **Test summary** to view the details in the **Tests** tab. This page has the following sections

- **Summary**: provides key quantitative metrics for the test execution such as the total test count, failed tests, pass percentage, and more. It also provides differential indicators of change compared to the previous execution.
- **Results**: lists all tests executed and reported as part of the current build or release. The default view shows only the failed and aborted tests in order to focus on tests that require attention. However, you can choose other outcomes using the filters provided.
- **Details**: A list of tests that you can sort, group, search, and filter to find the test results you need.

VSO.PR VSO.PR_20180802.610

Logs Timeline Code coverage* Accessibility* Scan Results* **Tests** Build Targets WhiteSource Bolt Build Report

Summary ^

2 Run(s) Completed (1 Passed, 1 Failed, 0 Not impacted, 0 Others)

Total tests	Passed	Failed	Others	Pass percentage
56335	56333	2	0	99.99% ↑ 99.9%
+56335				

Bug ▾ Link ▾ Test run ▾ Column Options ▾

Owner	Test	Duration
	L0 Run - VSO.PR (2/50272)	0:54:46.379
✓	FromWebApiShouldSetDefaultVersionTypeToSelectDuringReleaseCreationIfNoneExists <small>New</small>	0:00:00.050
	GetInputValuesForDefaultVersionShouldReturnAllDefaultVersionOptions <small>New</small>	0:00:00.573

Select any test run or result to view the details pane that displays additional information required for troubleshooting such as the error message, stack trace, attachments, work items, historical trend, and more.

VSO.PR VSO.PR_20180802.610

Logs Timeline Code coverage* Accessibility* Scan Results* **Tests** Build Targets WhiteSource Bolt Build Report

Summary ^

2 Run(s) Completed (1 Passed, 1 Failed, 0 Not impacted, 0 Others)

Total tests	Passed	Failed	Others	Pass percentage
56335	56333	2	0	99.99% ↑ 99.9%
+56335				

Bug ▾ Link ▾

Owner	Test	Duration
	L0 Run - VSO.PR (2/50272)	0:54:46.379
✓	FromWebApiShouldSetDefaultVersionTypeToSelectDuringReleaseCreationIfNoneExists <small>New</small>	0:00:00.050
	GetInputValuesForDefaultVersionShouldReturnAllDefaultVersionOptions <small>New</small>	0:00:00.573

FromWebApiShouldSetDefaultVersionTypeToSelectDuringReleaseCreationIfNoneExists New

✗ Failed an hour ago on TFSBUILD108 Duration 0:00:00.050

Failing build Current build Owner not available

Debug Work items Attachments History

Error message ^

```
Test method Microsoft.VisualStudio.Services.ReleaseManagement.FluentAssertions.Execution.AssertionFailedException: Expected "selectDuringReleaseCreationType" with a length of 31, but "latestType" has a length of 10.
```

Stack trace ^

```
at FluentAssertions.Execution.FallbackTestFramework.Throw()
at FluentAssertions.Execution.DefaultAssertionStrategy.HandleFailure(String message)
at FluentAssertions.Execution.AssertionScope.FailWith(String message)
at FluentAssertions.Primitives.StringEqualityValidator.Validate()
at FluentAssertions.Primitives.StringValidator.Validate()
at FluentAssertions.Primitives.StringAssertions.Be(String value)
at Microsoft.VisualStudio.Services.ReleaseManagement2.Data
```

The following capabilities of the **Tests** tab help to improve productivity and troubleshooting experience.

Filter large test results

Over time, tests accrue and, for large applications, can easily grow to tens of thousands of tests. For these applications with very many tests, it can be hard to navigate through the results to identify test failures, associate root causes, or get ownership of issues. Filters make it easy to quickly navigate to the test results of your interest. You can filter on **Test Name**, **Outcome** (failed, passed, and more), **Test Files** (files holding tests) and **Owner** (for test files). All of the filter criteria are cumulative in nature.

Handle	common.testfram...	Owner	Outcome	Clear 2
Test				
✓ L0 Run - VSO.PR (4/50272)	Duration			
✓ ExceptionHandledWithWriteInnerException	0:00:00.003			
✓ HandlesInnerExceptionWithRightMessage	0:00:00.000			
✓ HandlesPolymorphism	0:00:00.000			
✓ HandlesInnerExceptionPolymorphism	0:00:00.004			

Additionally, with multiple **Grouping** options such as **Test run**, **Test file**, **Priority**, **Requirement**, and more, you can organize the **Results** view exactly as you require.

Immersive troubleshooting experience

Error messages and stack traces are lengthy in nature and need enough real estate to view the details during troubleshooting. To provide an immersive troubleshooting experience, the **Details** view can be expanded to full page view while still being able to perform the required operations in context, such as bug creation or requirement association for the selected test result.

VSO.PR VSO.PR_20180802.610
◆ VSO · master · PR 369488 : First cut of changes for multi build support · Pull request validation build
1406 1316 1408 1314 1744 ▾

Logs Timeline Code coverage* Accessibility* Scan Results* Tests Build Targets | [Release](#) [Edit](#) [Queue](#) ...

L0 Run - VSO.PR > FromWebApiShouldSetDefaultVersionTypeToSelectDuringReleaseCreationIfNoneExist
✗ Failed an hour ago on TFSBUILD108 Duration 0:00:00.050
Failing build Current build Owner not available

[Debug](#) Work items Attachments History | [Bug](#) [Link](#)

Error message ^

```
Test method Microsoft.VisualStudio.Services.ReleaseManagement2.Data.UnitTests.Converters.ArtifactConverterTests.FromWebApiShouldSetDefaultVersionTypeToSelectDuringReleaseCreationIfNoneExist() threw an exception of type FluentAssertions.Execution.AssertionFailedException. Expected string to be "selectDuringReleaseCreationType" with a length of 31, but "latestType" has a length of 10.
```

Stack trace ^

```
at FluentAssertions.Execution.FallbackTestFramework.Throw(String message) in d:\Workspaces\FluentAssertions\FluentAssertions.Execution\FallbackTestFramework.cs:line 100
at FluentAssertions.Execution.DefaultAssertionStrategy.HandleFailure(String message) in d:\Workspaces\FluentAssertions\FluentAssertions.Execution\DefaultAssertionStrategy.cs:line 100
at FluentAssertions.Execution.AssertionScope.FailWith(String failureMessage, Object[] failureArgs) in d:\Workspaces\FluentAssertions\FluentAssertions.Execution\AssertionScope.cs:line 100
at FluentAssertions.Primitives.StringEqualityValidator.ValidateAgainstLengthDifferences() in d:\Workspaces\FluentAssertions\FluentAssertions.Primitives\StringEqualityValidator.cs:line 100
at FluentAssertions.Primitives.StringValidator.Validate() in d:\Workspaces\FluentAssertions\FluentAssertions.Primitives\StringValidator.cs:line 100
at FluentAssertions.Primitives.StringAssertions.Be(String expected, String reason, Object[] reasonArgs) in d:\Workspaces\FluentAssertions\FluentAssertions.Primitives\StringAssertions.cs:line 100
at Microsoft.VisualStudio.Services.ReleaseManagement2.Data.UnitTests.Converters.ArtifactConverterTests.FromWebApiShouldSetDefaultVersionTypeToSelectDuringReleaseCreationIfNoneExist() in d:\Workspaces\Visual Studio\Visual Studio 2017\Projects\VSO\src\VSO\Microsoft.VisualStudio.Services.ReleaseManagement2\Microsoft.VisualStudio.Services.ReleaseManagement2.Data\UnitTests\Converters\ArtifactConverterTests.cs:line 100
```

Test trends with historical data

History of test execution can provide meaningful insights into reliability or performance of tests. When troubleshooting a failure, it is valuable to know how a test has performed in the past. The **Tests** tab provides test history in context with the test results. The test history information is exposed in a progressive manner starting with the current build pipeline to other branches, or the current stage to other stages, for build and release respectively.

VSO.PR VSO.PR_20180802.610

◆ VSO · master · PR 369488 : First cut of changes for multi build support · Pull request validation build

1406 1316 1408 1314 1744 ▾

Logs Timeline Code coverage* Accessibility* Scan Results* Tests Build Targets WhiteSource Bolt Build Report | [Release](#) [Edit](#)

L0 Run - VSO.PR > FromWebApiShouldSetDefaultVersionTypeToSelectDuringReleaseCreationIfNoneExist

✖ Failed 2 hours ago on TFSBUILD108 Duration 0:00:00.050

Failing build Current build Owner not available

Debug Work items Attachments History | [Bug](#) [Link](#)

Branch [Clear](#)

Current build pipeline ^

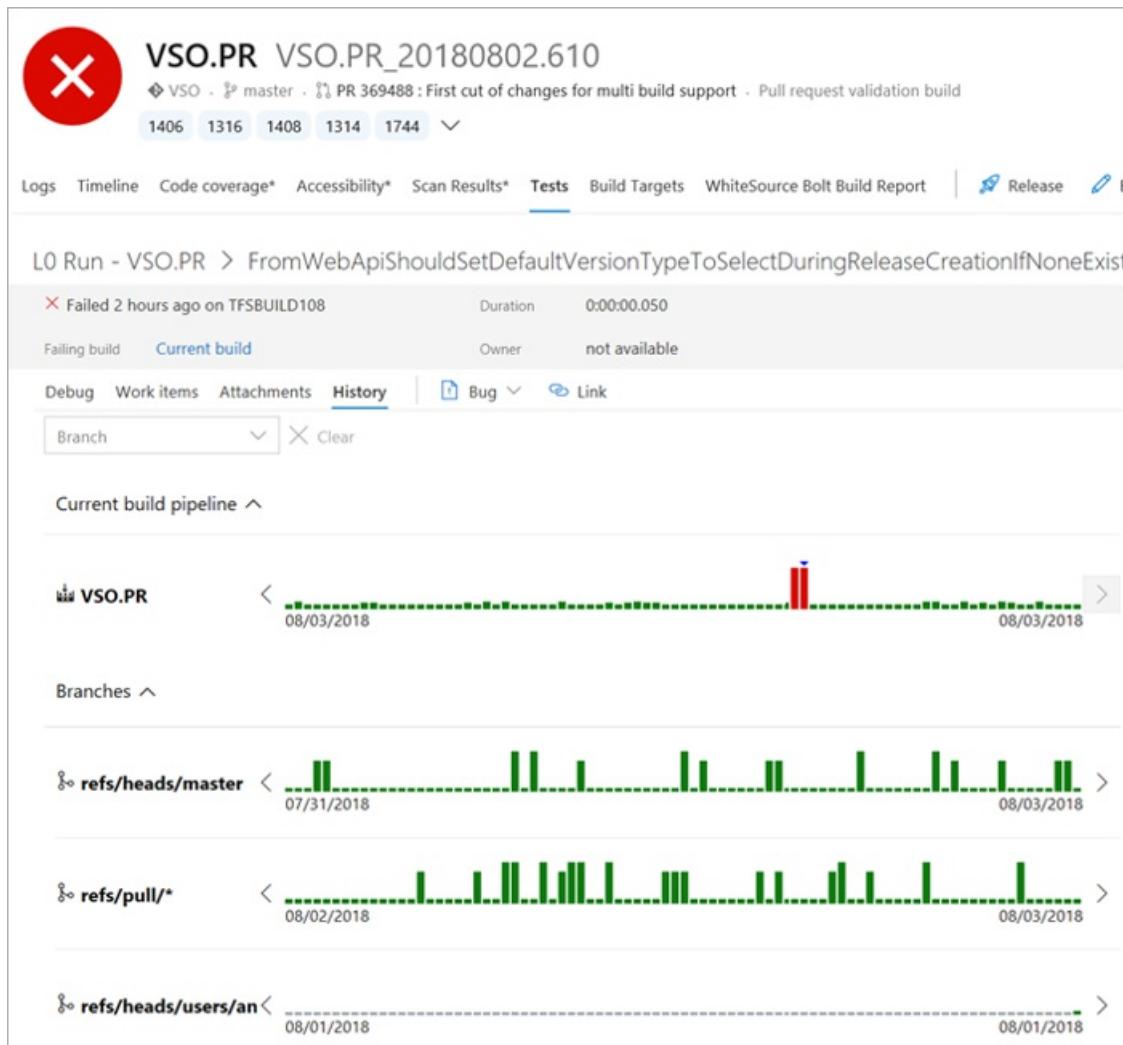
VSO.PR 08/03/2018 08/03/2018

Branches ^

refs/heads/master 07/31/2018 08/03/2018

refs/pull/* 08/02/2018 08/03/2018

refs/heads/users/an 08/01/2018 08/01/2018



View execution of in-progress tests

Tests, such as integration and functional tests, can run for a long time. Therefore, it is important to see the current or near real-time status of test execution at any given time. Even for cases where tests run quickly, it's useful to know the status of the relevant test result(s) as early as possible; especially when failures occur. The **in-progress** view eliminates the need to wait for test execution to finish. Results are available in near real-time as execution progresses, helping you to take actions faster. You can debug a failure, file a bug, or abort the pipeline.

↑ TFS.ATTPC - VS.CI > VS.CI_20180613.13.1 ▾

Pipeline Variables History | + Deploy ▾ [Cancel](#) [Refresh](#) [Release \(old view\)](#) [Edit](#)

Release

Continuous deployment for Abhijit Chakraborty 6/13/2018 3:40 AM

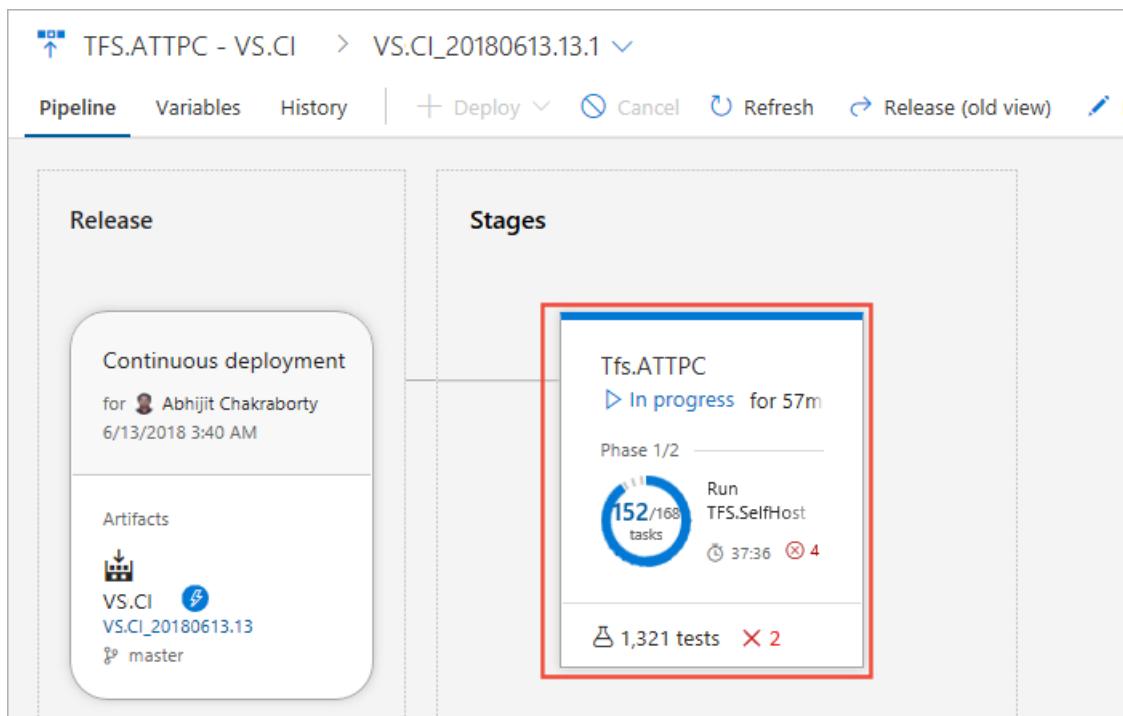
Artifacts VS.CI VS.CI_20180613.13

Stages

Tfs.ATTPC In progress for 57m

Phase 1/2 Run TFS.SelfHost 37:36 4

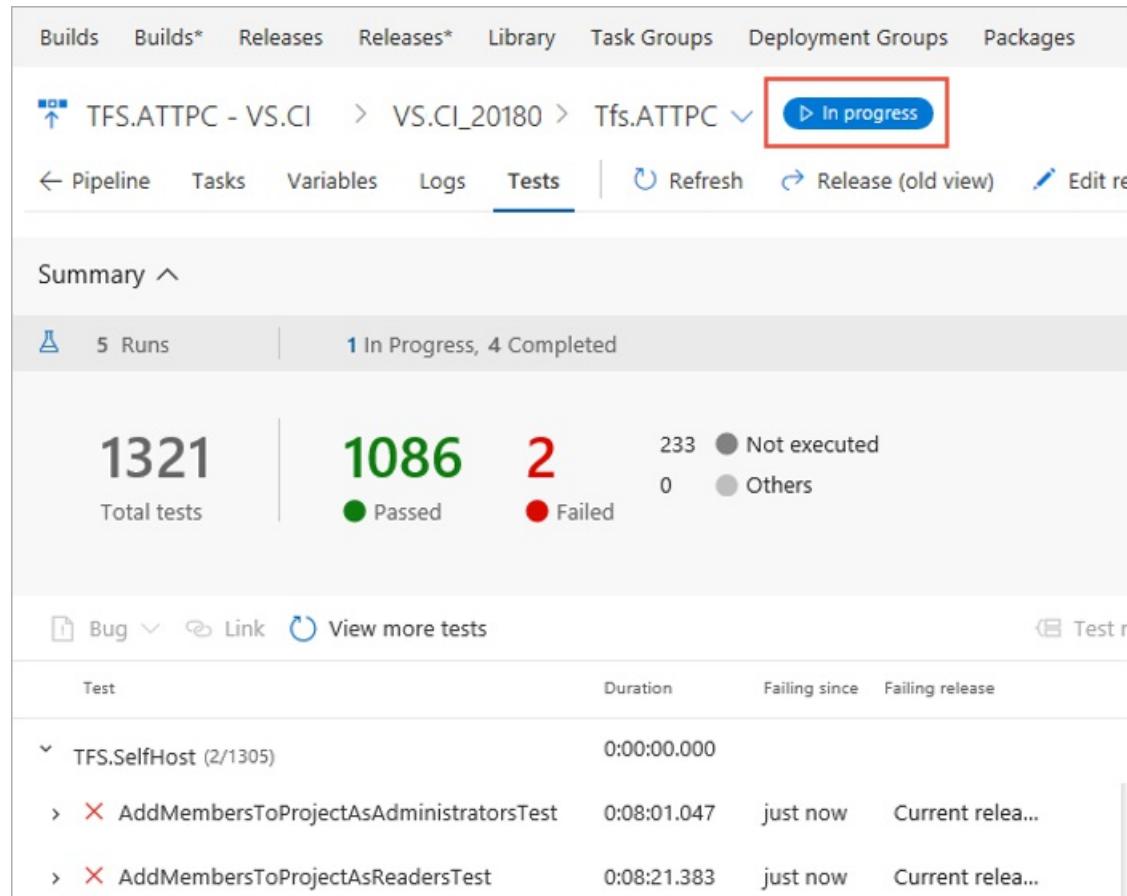
1,321 tests 2



NOTE

The feature is currently available for both build and release, using [Visual Studio Test](#) task in a Multi Agent job. It will be available for Single Agent jobs in a future release.

The view below shows the **in-progress** test summary in a release, reporting the total test count and the number of test failures at a given point in time. The test failures are available for troubleshooting, creating bug(s), or to take any other appropriate action.



The screenshot shows the TFS Test Results interface. At the top, there are tabs for Builds, Builds*, Releases, Releases*, Library, Task Groups, Deployment Groups, and Packages. Below these are navigation links for Pipeline, Tasks, Variables, Logs, Tests (which is selected and underlined), Refresh, Release (old view), and Edit. A status indicator 'In progress' is highlighted with a red box. The main area is titled 'Summary ^'. It displays '5 Runs' with '1 In Progress, 4 Completed'. Below this, it shows '1321 Total tests', '1086 Passed', and '2 Failed'. To the right, there are counts for 'Not executed' (233) and 'Others' (0). At the bottom, there are links for Bug, Link, View more tests, and Test results. The 'Test results' section lists individual test cases with columns for Test, Duration, Failing since, and Failing release. Examples include 'TFS.SelfHost (2/1305)' and two failed tests for 'AddMembersToProjectAsAdministratorsTest' and 'AddMembersToProjectAsReadersTest'.

View summarized test results

During test execution, a test might spawn multiple instances or tests that contribute to the overall outcome. Some examples are, tests that are rerun, tests composed of an ordered combination of other tests (ordered tests) or tests having different instances based on an input parameter (data driven tests).

As these tests are related, they must be reported together with the overall outcome derived from the individual instances or tests. These test results are reported as a summarized test result in the **Tests** tab:

- **Rerun failed tests:** The ability to rerun failed tests is available in the latest version of the [Visual Studio Test](#) task. During a rerun, multiple attempts can be made for a failed test, and each failure could have a different root cause due to the non-deterministic behavior of the test. Test reports provide a combined view for all the attempts of a rerun, along with the overall test outcome as a summarized unit. Additionally the [Test Management API\(s\)](#) now support the ability to publish and query summarized test results.

Tfs.SelfHost.CodeDev - VSO.CI > VSO.CI_20180608.154.1 > Tfs.SelfHost.CodeDev

← Pipeline Tasks Variables Logs Tests | → Release (old view) Edit release

Summary ^

4 Run(s) Completed (4 Passed, 0 Failed, 0 Not impacted, 1 Skipped)

1063 Total tests



Status	Count
Passed	1063
Failed	0
Others	0

Bug ▾ Link Test ru ▾

Filter by test name Container

Test

▼ TFS.SelfHost.CodeDev (2/1306)

- ▼ ExportWorkItemTypeDefinitionFuzzTest
 - ✓ Attempt# 1 - ExportWorkItemTypeDefinitionFuzzTest
 - ✗ Attempt# 0 - ExportWorkItemTypeDefinitionFuzzTest
- ▼ UpdateWorkItemTypeDefinitionFuzzTest

UpdateWorkItemTypeDefinitionFuzzTest

✓ Passed 2 days ago on RICP0POOL4-0070

Debug Work items Attachments History

Error message ^

```
Test method MS.TF.Test.WIT.REST.Tests.Security.WorkItem
System.Exception: Microsoft.VisualStudio.TestTools.UnitTesting.
at Microsoft.VisualStudio.TestTools.UnitTesting.Assert.Inco
at MS.TF.Test.WIT.REST.Tests.Security.WorkItemTypeDefin
at Common.TestFramework.L2.Fuzz.FuzzTest.<>c__Display
at Common.TestFramework.L2.Fuzz.SingleThreadFuzzItera
```

Stack trace ^

```
at Common.TestFramework.L2.Fuzz.SingleThreadFuzzItera
at Common.TestFramework.L2.Fuzz.SingleThreadFuzzItera
at Common.TestFramework.L2.Fuzz.FuzzTest.Start[T](Int32
at Common.TestFramework.L2.Fuzz.FuzzTest.Start[Int32]
at MS.TF.Test.WIT.REST.Tests.Security.WorkItemTypeDefin
```

- **Data driven tests:** Similar to the rerun of failed tests, all iterations of data driven tests are reported under that test. The summarized result view for data driven tests depends on the behavior of the test framework. If the framework produces a hierarchy of results (for example, MSTest v1 and v2) they will be reported in a summarized view. If the framework produces individual results for each iteration (for example, xUnit) they will not be grouped together. The summarized view is also available for ordered tests (**.orderedtest** in Visual Studio).

Partsunlimited.CD > Release-37 > QA Environment

← Pipeline Tasks Variables Logs Tests → Release (old view) Edit release

Summary ^

3 Run(s) Completed (0 Passed, 3 Failed, 0 Not imp.)

56 Total tests



Status	Count
Passed	46
Failed	10
Others	0

Bug ▾ Link

Filter by test name Container

Test

- > Multi config tests Chrome_1 (2/7)
- > Multi config tests Firefox_1 (1/7)
- ▼ TestRun_Partsunlimited.CD_Release-37 (7/42)
 - ✗ AdditionTest
 - ✗ AdditionTest (Data Row 2)
 - ✓ AdditionTest (Data Row 1)
 - ✓ AdditionTest (Data Row 0)
 - ✗ E2ETest_Search

AdditionTest (Data Row 2)

✗ Failed 3 hours ago on autologon4 (auto)

Failing release Release-36

Debug Work items Attachments

Error message ^

Assert.Fail failed. Row cant be 6

Stack trace ^

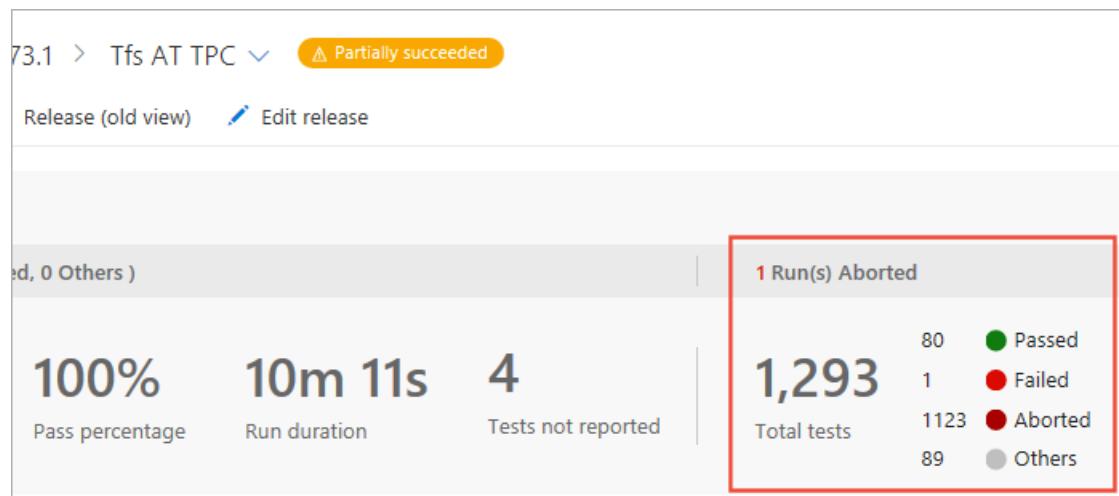
at PartsUnlimitedServiceTest.PartsUnlimi

NOTE

Metrics in the test summary section, such as the total number of tests, passed, failed, or other are computed using the root level of the summarized test result.

View aborted tests

Test execution can abort due to several reasons such as bad test code, errors in the source under test, or environmental issues. Irrespective of the reason for the abort, it is important to be able to diagnose the behavior and identify the root cause. The aborted tests and test runs can be viewed alongside the completed runs in the **Tests** tab.



NOTE

The feature is currently available for both build and release, using the [Visual Studio Test task](#) in a Multi Agent job or publishing test results using the [Test Management API\(s\)](#). It will be available for Single Agent jobs in a future release.

Surface test results in the Tests tab

Tests results can be surfaced in the **Tests** tab using one of the following options:

- **Test execution tasks:** built-in test execution tasks such as [Visual Studio Test](#) that automatically publish test results to the pipeline, or others such as [Ant](#), [Maven](#), [Gulp](#), [Grunt](#), and [Xcode](#) that provide this capability as an option within the task.
- **Publish Test Results task:** publishes test results to Azure Pipelines or TFS when tests are executed using your choice of runner, and results are available in any of the [supported test result formats](#).
- **API(s):** test results published directly using the [Test Management API\(s\)](#).

Surface test information beyond the Tests tab

The **Tests** tab provides a detailed summary of the test execution. This is helpful in tracking the quality of the pipeline, as well as troubleshooting failures. Azure DevOps also provides other ways to surface the test information:

- The [Dashboard](#) provides visibility into your team's progress. Add one or more widgets that surface test related information:
 - [Requirements quality](#)
 - [Test results trend](#)
 - [Deployment status](#)
- [Test analytics](#) provides rich insights into test results measured over a period of time. It can help identify problematic areas in your test by providing data such as the top failing tests, and more.

Help and support

Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Analyze test results

10/9/2018 • 3 minutes to read • [Edit Online](#)

Azure Pipelines

Tracking test quality over time and improving test collateral is key to maintaining a healthy DevOps pipeline. Test analytics provides near real-time visibility into your test data for builds and releases. It helps improve the efficiency of your pipeline by identifying repetitive, high impact quality issues.

NOTE

Test analytics is currently available only with Azure Pipelines.

Read the [glossary](#) to understand test reports terminology.

NOTE

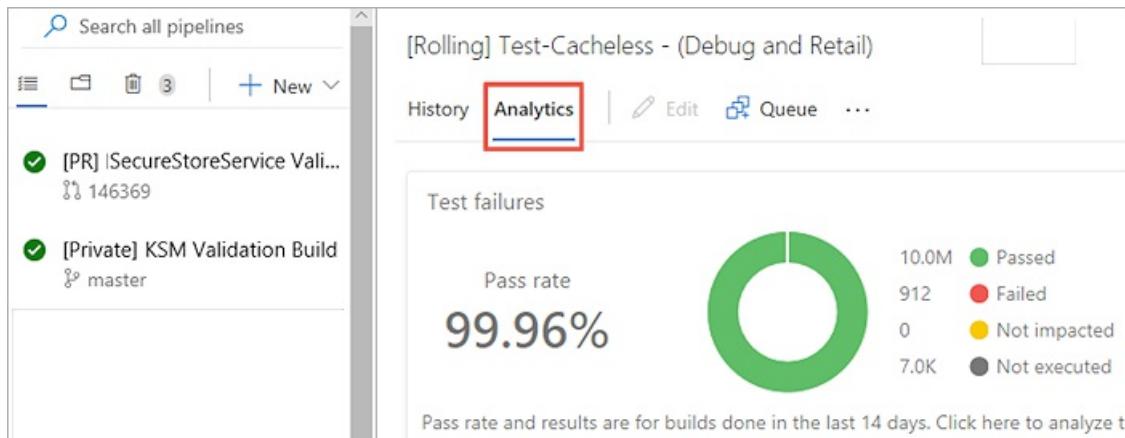
Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Install the Analytics extension

1. Go to the [Analytics extension](#) in Marketplace.
2. Choose to install the Analytics extension. You might need to sign in with your Azure DevOps Services organization credentials.
3. Select the Azure DevOps Services organization where you would like to install this extension and confirm.
[Learn more about installing extensions.](#)

View test analytics for builds

To help teams find and fix tests that fail frequently or intermittently, use the **top failing tests** report. The build summary includes the **Analytics** page that hosts this report. The top-level view provides a summary of the test pass rate and results for the selected build pipeline, for the specified period. The default range is 14 days.



View test analytics for releases

For tests executing as part of release, access test analytics from the **Analytics** link at the top right corner. As with

build, the summary provides an aggregated view of the test pass rate and results for the specified period.

The screenshot shows the 'Release pipelines' summary page for the 'Tfs.SelfHost.CodeDev' pipeline. It displays a list of recent releases and their stages. A large 'Analysis' section on the right provides performance indicators, including a 'Pass rate and results - last 14 days' chart showing a 99.95% pass rate with 2.6M total tests, 986 failed, 0 not impacted, and 566K not executed. A search bar and various navigation links are visible at the top.

Test Failures

Open a build or release summary to view the top failing tests report. This report provides a granular view of the top failing tests in the pipeline, along with the failure details.

The screenshot shows the 'Test Failures Report' for the 'Tfs.SelfHost.CodeDev' pipeline. It includes a 'Summary' section with a pass rate of 99.95%, a donut chart of test outcomes, and a bar chart of failed result counts over time. Below this is a table of failing test details, showing metrics like Failed, Pass rate, Total count, and Average duration for specific tests like ValidateRetentionTabExperienceForTfcProject and CachingGms_RemoteCacheInvalidation.

Test	Failed	Pass rate	Total count	Average duration
ValidateRetentionTabExperienceForTfcProject	368	75.46%	1500	42.35s
CachingGms_RemoteCacheInvalidation	87	94.2%	1500	20.31s
RemoteSecurityNamespacesServiceBus	25	98.33%	1500	10.32s
ValidateOptionsTabExperience	15	99%	1500	62.9s

The detailed view contains two sections:

- **Summary:** Provides key quantitative metrics for the tests executed in build or release over the specified period. The default view shows data for 14 days.
 - Pass rate and results: Shows the [pass percentage](#), along with the distribution of tests across various outcomes.

Pass rate and results

99.95%



2.6M	Passed
986	Failed
0	Not impacted
566K	Not executed

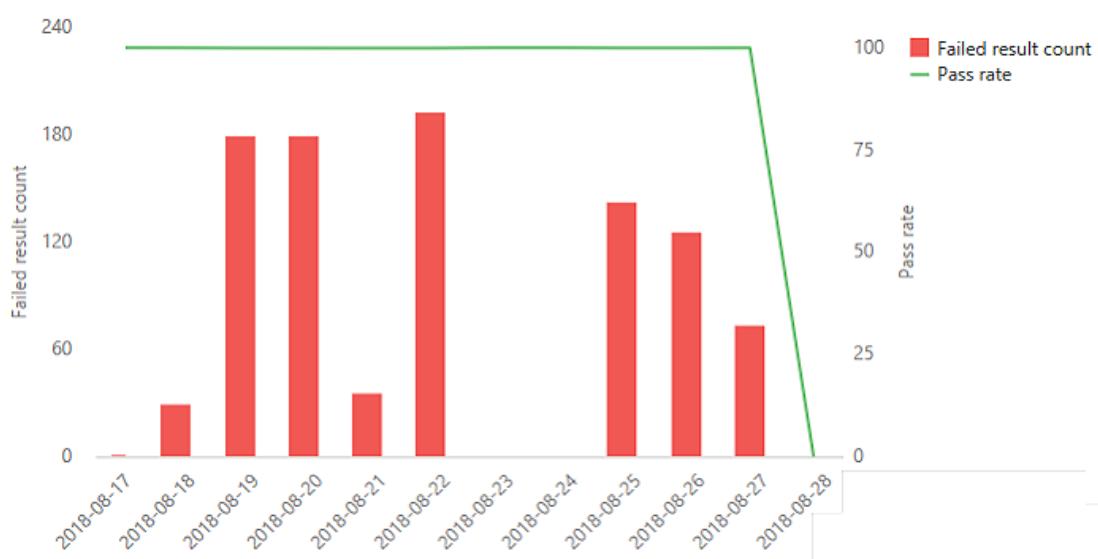
- Failing tests: Provides a distinct count of tests that failed during the specified period. In the example above, 986 test failures originated from 124 tests.

Failing tests

124 tests

- Chart view: A trend of the total test failures and average pass rate on each day of the specified period.

Result count and pass rate



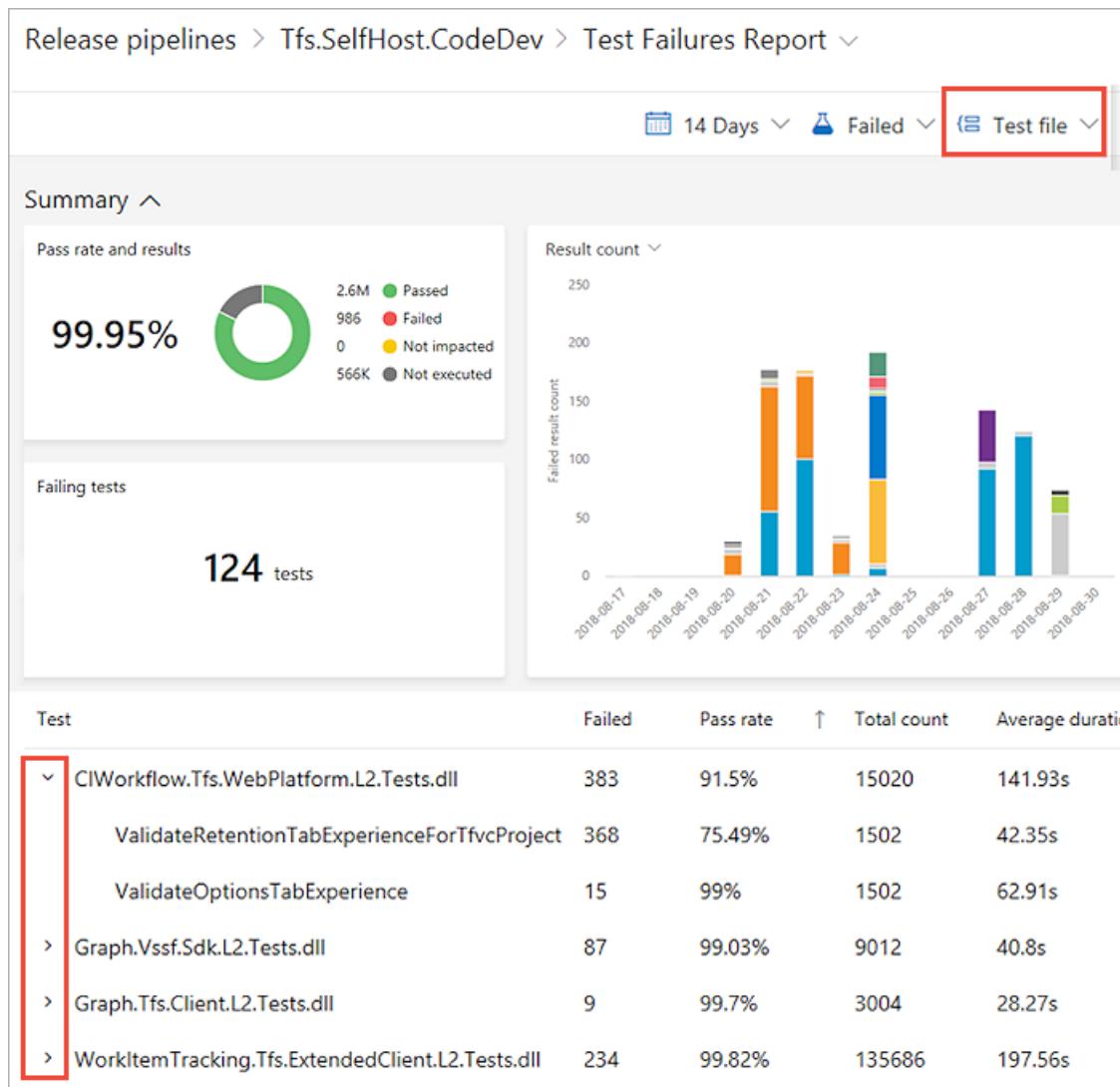
- **Results:** List of top failed tests based on the total number of failures. Helps to identify problematic tests and lets you drill into a detailed summary of results.

Test	Failed	Pass rate	↑	Total count	Average duration
ValidateRetentionTabExperienceForTfvcProject	368	75.46%		1500	42.35s
CachingGms_RemoteCacheInvalidation	87	94.2%		1500	20.31s
RemoteSecurityNamespacesServiceBus	25	98.33%		1500	10.32s
ValidateOptionsTabExperience	15	99%		1500	62.9s
ValidatePreCreateAuditNotifications	9	99.28%		1282	15.83s
ProjectRenameTest	10	99.33%		1500	7.86s

Group test failures

The report view can be organized in several different ways using the **group by** option. Grouping test results can provide deep insights into various aspects of the top failing tests. In the example below, the test results are

grouped based on the [test files](#) they belong to. It shows the test files and their respective contribution towards the total of test failures, during the specified period to help you easily identify and prioritize your next steps. Additionally, for each test file, it shows the tests that contribute to these failures.

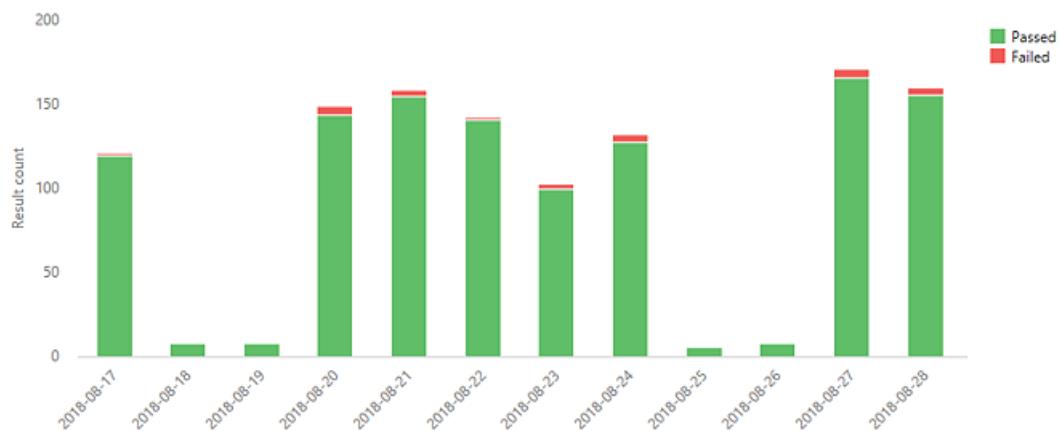


Drill down to individual tests

After you have identified one or more tests in the **Details** section, select the individual test you want to analyze. This provides a drill-down view of the selected test with a stacked chart of various outcomes such as passed or failed instances of the test, for each day in the specified period. This view helps you infer hidden patterns and take actions accordingly.

Summary ^

Result count ▾



Outcome	Date	Duration	Branch	Stage
✓ Passed	8/29/2018 8:58 PM	25.73s	refs/heads/master	Tfs.SelfHost.CodeDev
✓ Passed	8/29/2018 8:50 PM	29.04s	refs/heads/master	Tfs.SelfHost.CodeDev
✓ Passed	8/29/2018 8:42 PM	12.47s	refs/heads/master	Tfs.SelfHost.CodeDev
✓ Passed	8/29/2018 8:22 PM	10.41s	refs/heads/master	Tfs.SelfHost.CodeDev
✓ Passed	8/29/2018 8:10 PM	27.72s	refs/heads/master	Tfs.SelfHost.CodeDev
✓ Passed	8/29/2018 8:09 PM	12.4s	refs/heads/master	Tfs.SelfHost.CodeDev
✓ Passed	8/29/2018 8:06 PM	34.26s	refs/heads/master	Tfs.SelfHost.CodeDev
✗ Failed	8/29/2018 8:02 PM	103.95s	refs/heads/master	Tfs.SelfHost.CodeDev

The corresponding grid view lists all instances of execution of the selected test during that period.

Release pipelines > Tfs.SelfHost.CodeDev > Test Failures Report ▾

CachingGms_RemoteCacheInvalidation ⏪ Back					14 Days ▾
Outcome	Date	Duration	Branch	Stage	
✓ Passed	8/29/2018 8:58 PM	25.73s	refs/heads/master	Tfs.SelfHost.CodeDev	
✓ Passed	8/29/2018 8:50 PM	29.04s	refs/heads/master	Tfs.SelfHost.CodeDev	
✓ Passed	8/29/2018 8:42 PM	12.47s	refs/heads/master	Tfs.SelfHost.CodeDev	
✓ Passed	8/29/2018 8:22 PM	10.41s	refs/heads/master	Tfs.SelfHost.CodeDev	
✓ Passed	8/29/2018 8:10 PM	27.72s	refs/heads/master	Tfs.SelfHost.CodeDev	
✓ Passed	8/29/2018 8:09 PM	12.4s	refs/heads/master	Tfs.SelfHost.CodeDev	
✓ Passed	8/29/2018 8:06 PM	34.26s	refs/heads/master	Tfs.SelfHost.CodeDev	
✗ Failed	8/29/2018 8:02 PM	103.95s	refs/heads/master	Tfs.SelfHost.CodeDev	
✓ Passed	8/29/2018 7:58 PM	31.2s	refs/heads/master	Tfs.SelfHost.CodeDev	
✗ Failed	8/29/2018 7:54 PM	104.53s	refs/heads/master	Tfs.SelfHost.CodeDev	
✓ Passed	8/29/2018 7:51 PM	25.99s	refs/heads/master	Tfs.SelfHost.CodeDev	
✓ Passed	8/29/2018 7:44 PM	12.42s	refs/heads/master	Tfs.SelfHost.CodeDev	
✓ Passed	8/29/2018 7:32 PM	12.43s	refs/heads/master	Tfs.SelfHost.CodeDev	
✓ Passed	8/29/2018 7:28 PM	29.61s	refs/heads/master	Tfs.SelfHost.CodeDev	
✗ Failed	8/29/2018 7:04 PM	109.01s	refs/heads/master	Tfs.SelfHost.CodeDev	
✗ Failed	8/29/2018 7:01 PM	111.89s	refs/heads/master	Tfs.SelfHost.CodeDev	
✓ Passed	8/29/2018 6:54 PM	28.01s	refs/heads/master	Tfs.SelfHost.CodeDev	
✓ Passed	8/29/2018 6:54 PM	14.48s	refs/heads/master	Tfs.SelfHost.CodeDev	
✗ Failed	8/29/2018 6:50 PM	111.31s	refs/heads/master	Tfs.SelfHost.CodeDev	
✗ Failed	8/29/2018 6:48 PM	110.51s	refs/heads/master	Tfs.SelfHost.CodeDev	
✓ Passed	8/29/2018 6:47 PM	34.44s	refs/heads/master	Tfs.SelfHost.CodeDev	

Failure analysis

To perform failure analysis for root causes, choose one or more instances of test execution in the drill-down view to see failure details in context.

Release pipelines > Tfs.SelfHost.CodeDev - VSO.CI > Test Failures Report ▾

CachingGms_RemoteCacheInvalidation | ⏪ Back

14 Days ▾

Result count

Date	Result count
2018-08-17	~120
2018-08-18	~5
2018-08-19	~5
2018-08-20	~145
2018-08-21	~155
2018-08-22	~140
2018-08-23	~100
2018-08-24	~130
2018-08-25	~5
2018-08-26	~5
2018-08-27	~5

Debug Work items Attachments

Error message ▾

```
Test method Graph.Vssf.Sdk.L2.Tests.C
System.TimeoutException: Retry reache
```

Stack trace ▾

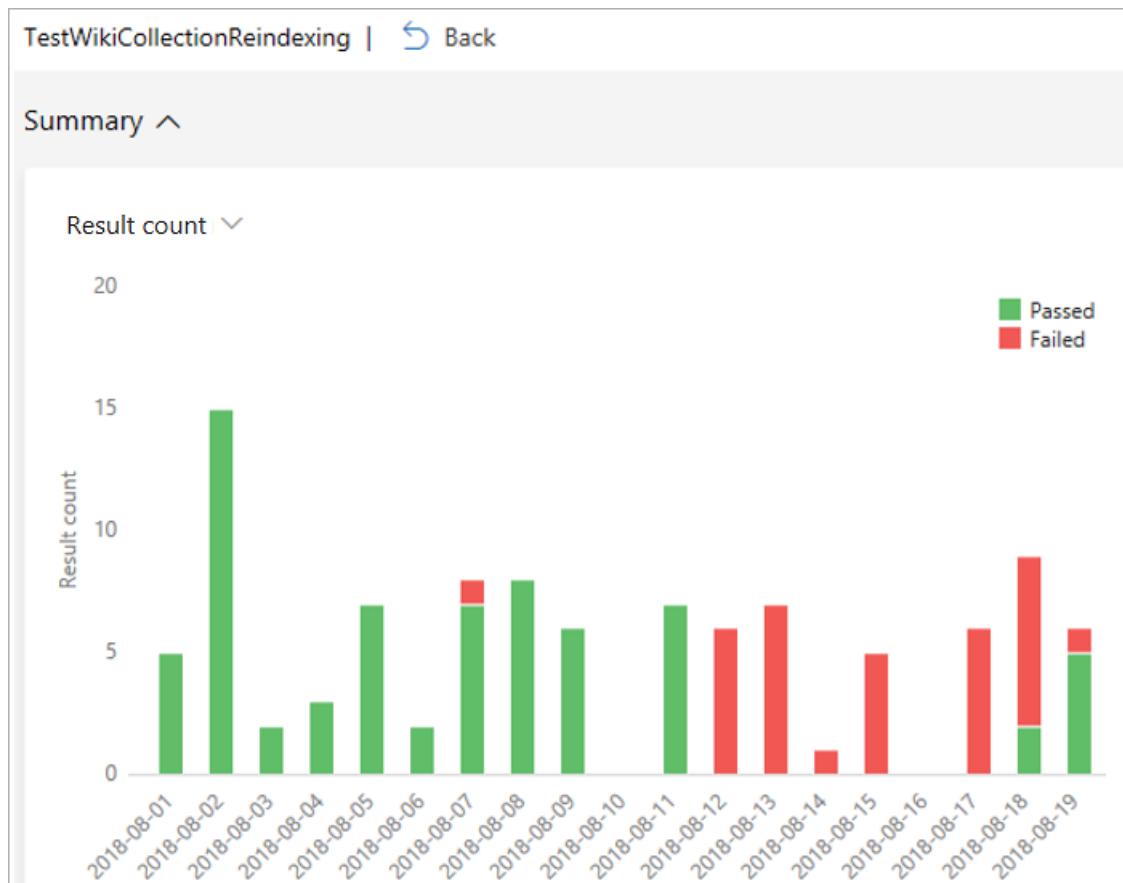
```
at Vssf.TestFramework.Client.Comm
at Graph.Vssf.Sdk.L2.Tests.Caching
```

Outcome Date Duration Branch

✓ Passed	8/29/2018 8:58 PM	25.73s	refs/
✓ Passed	8/29/2018 8:50 PM	29.04s	refs/
✓ Passed	8/29/2018 8:42 PM	12.47s	refs/
✓ Passed	8/29/2018 8:22 PM	10.41s	refs/
✓ Passed	8/29/2018 8:10 PM	27.72s	refs/
✓ Passed	8/29/2018 8:09 PM	12.4s	refs/
✓ Passed	8/29/2018 8:06 PM	34.26s	refs/
✗ Failed	8/29/2018 8:02 PM	103.95s	refs/
✓ Passed	8/29/2018 7:58 PM	31.2s	refs/
✗ Failed	8/29/2018 7:54 PM	104.53s	refs/

Infer hidden patterns

When looking at the test failures for a single instance of execution, it is often difficult to infer any pattern. In the example below, the test failures occurred during a specific period, and knowing this can help narrow down the scope of investigation.



Another example is tests that exhibit non-deterministic behavior (often referred to as [flaky tests](#)). Looking at an individual instance of test execution may not provide any meaningful insights into the behavior. However, observing test execution trends for a period can help infer hidden patterns, and help you resolve the failures.

Report information source

The source of information for test analytics is the set of [published test results](#) for the build or release pipeline. These results are accrued over a period of time, and form the basis of the rich insights that test analytics provides.

Help and support

Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Review code coverage results

11/19/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015

Code coverage helps you determine the proportion of your project's code that is actually being tested by tests such as unit tests. To increase your confidence of the code changes, and guard effectively against bugs, your tests should exercise - or cover - a large proportion of your code.

Reviewing the code coverage result helps to identify code path(s) that are not covered by the tests. This information is important to improve the test collateral over time by reducing the test debt.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Example

To view an example of publishing code coverage results for your choice of language, see the **Languages** section of the Pipelines topics. For example, collect and publish code coverage for [JavaScript](#) using Istanbul.

View results

The code coverage summary can be viewed in the build timeline view. The summary shows the overall percentage of line coverage.

 **VSO.PR** 20180831.4

MyFirstProject · master · eeb7030 : Updated UnitTest2.cs · Manual build

Add a tag

Logs Summary Tests Code Coverage Artifacts Edit Queue ...

Progression

Build artifacts published ▾
1

Code coverage succeeded ▾
97.14% code covered, 2 flavor(s)
Configuration: release | Platform: any cpu
0% 100%
Configuration: Release | Platform: x64
0% 100%

Tests succeeded ▾
100% passed

Build pipeline succeeded
0 error(s) / 0 warning(s)
queued a minute ago Ran for 56 seconds

Manually queued
SP requested a minute ago

Associated changes
50 commit(s)
Updated UnitTest2.cs

NOTE

Merging code coverage results from multiple [test runs](#) is limited to .NET and .NET Core at present. This will be supported for other formats in a future release.

Artifacts

The code coverage artifacts published during the build can be viewed under the **Build artifacts published** milestone in the timeline view.

Progression

Build artifacts published ▾
2

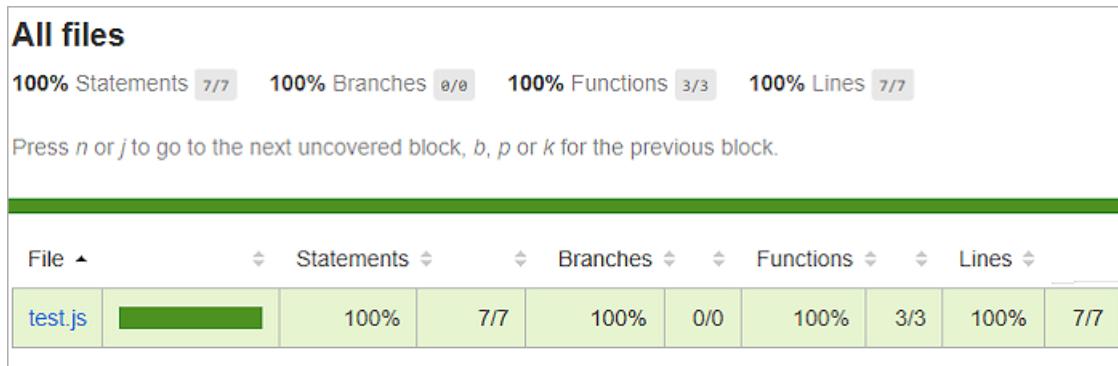
Code Coverage Report...
File container

drop
File container

- If you use the [Visual Studio Test](#) task to collect coverage for .NET and .NET Core apps, the artifact contains **.coverage** files that can be downloaded and used for further analysis in Visual Studio.

Code Coverage Results				
Hierarchy	Covered (Blocks)	Covered (% Blocks)	Not Covered (Blocks)	
20180831.5.release.any cpu.1249.coverage	3960	21.34%	14598	
mathlib.dll	6	100.00%	0	
mstestv2testproject1.dll	4	100.00%	0	
MSTestV2proj	4	100.00%	0	
AddTests	4	100.00%	0	
nunit.framework.dll	3176	19.19%	13372	
nunit3.testadapter.dll	766	38.45%	1226	
nunittestproject1.dll	4	100.00%	0	
xunittestproject1.dll	4	100.00%	0	

- If you publish code coverage using Cobertura or JaCoCo coverage formats, the code coverage artifact contains an HTML file that can be viewed offline for further analysis.



NOTE

For .NET and .NET Core, the link to download the artifact is available by choosing the code coverage milestone in the build summary.

Tasks

- [Publish Code Coverage Results](#) publishes code coverage results to Azure Pipelines or TFS, which were produced by a build in [Cobertura](#) or [JaCoCo](#) format.
- Built-in tasks such as [Visual Studio Test](#), [.NET Core](#), [Ant](#), [Maven](#), [Gulp](#), [Grunt](#), and [Gradle](#) provide the option to publish code coverage data to the pipeline.

Help and support

Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Trace test requirements

10/9/2018 • 5 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

Requirements traceability is the ability to relate and document two or more phases of a development process, which can then be traced both forward or backward from its origin. Requirements traceability help teams to get insights into indicators such as **quality of requirements** or **readiness to ship the requirement**. A fundamental aspect of requirements traceability is association of the requirements to test cases, bugs and code changes.

Read the [glossary](#) to understand test report terminology.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

Agile teams running automated tests

Agile teams have characteristics including, but not limited to the following

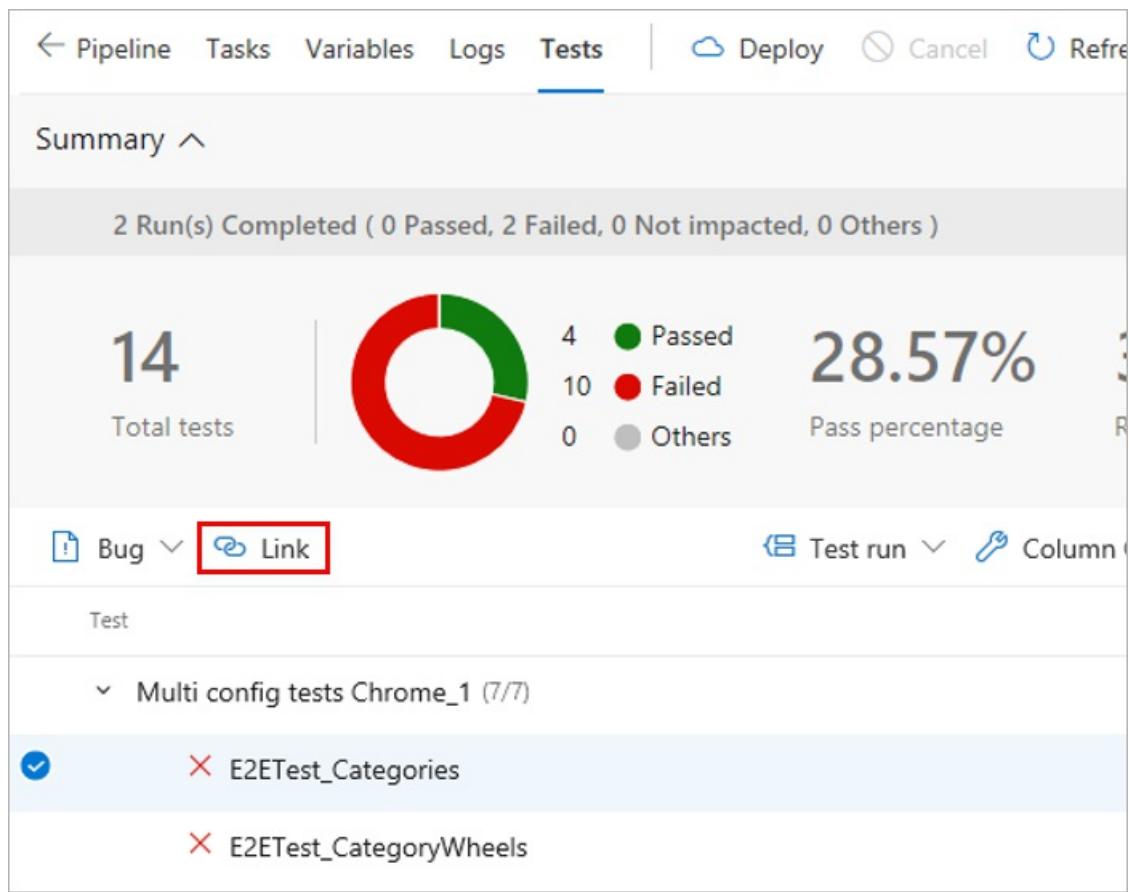
- Faster release cycles
- Continuous testing in a pipeline
- Negligible manual testing footprint; limited to exploratory testing
- High degree of automation

The following sections explore traceability from **Quality**, **Bug** and **Source** standpoints for Agile teams.

Quality traceability

To ensure user requirements meet the quality goals, the requirements in a project can be linked to test results, which can then be viewed on the team's dashboard. This enables end-to-end traceability with a simple way to monitor test results. To link automated tests with requirements, visit [test report](#) in build or release.

1. In the results section under **Tests** tab of a build or release summary, select the test(s) to be linked to requirements and choose **Link**.



2. Choose a work item to be linked to the selected test(s) in one of the specified way:

- Choose an applicable work item from the list of suggested work items. The list is based on the most recently viewed and updated work items.
- Specify a work item ID.
- Search for a work item based on the title text.

Associate tests to work item

Search for existing work item using work item id or keywords in title.

Showing 4 suggestions

ID ↑	Title	State	Assigned To
2626	As a user, I can reset a for...	Active	[redacted]
2627	As a user, I can let browser...	Active	[redacted]
2630	Escalate a support ticket	New	
2751	As a user, I can search the...	Resolved	[redacted]

Associate Close

The list shows only work items belonging to the Requirements category.

3. After the requirements have been linked to the test results you can view the test results grouped by requirement. Requirement is one of the many "Group by" options provided to make it easy to navigate the test results.

Test	Duration	Failing since
✓ Create and track support tickets (4/4)	0:01:08.959	
✗ E2ETest_CategoryWheels	0:00:41.750	8 months ago
✗ E2ETest_CategoryWheels	0:00:00.014	8 months ago
✗ E2ETest_Search	0:00:00.013	8 months ago
✗ E2ETest_Search	0:00:27.184	8 months ago
> User can add cash to wallet (3/4)	0:01:00.662	
> Not associated (5/8)	0:00:59.607	

4. Teams often want to pin the summarized view of requirements traceability to a dashboard. Use the [Requirements quality](#) widget for this.

5. Configure the **Requirements quality** widget with the required options and save it.

- **Requirements query:** Select a work item query that captures the requirements, such as the user stories in the current iteration.
- **Quality data:** Specify the stage of the pipeline for which the requirements quality should be traced.

Configuration

Title
Current iteration quality

Size
3 x 2

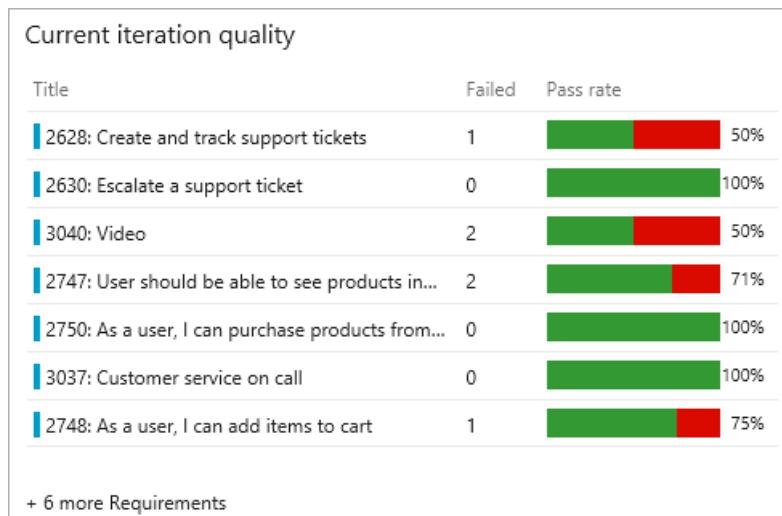
Requirements query
Open User Stories

Quality Data
 From Build From Release

Release pipeline
PartsUnlimited.CD

Save **Cancel**

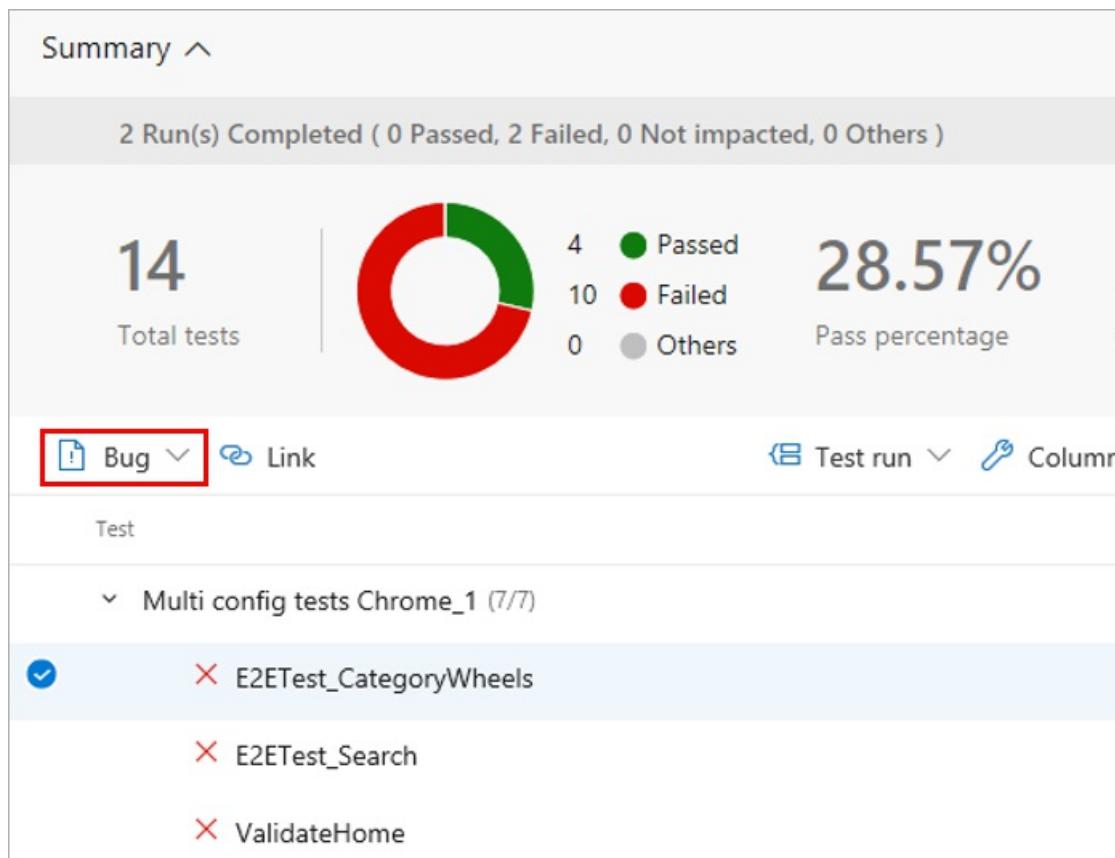
- View the widget in the team's dashboard. It lists all the **Requirements** in scope, along with the **Pass Rate** for the tests and count of Failed tests. Selecting a **Failed** test count opens the **Tests** tab for the selected build or release. The widget also helps to track the requirements without any associated test(s).



Bug traceability

Testing gives a measure of the confidence to ship a change to users. A test failure signals an issue with the change. Failures can happen for many reasons such as errors in the source under test, bad test code, environmental issues, [flaky tests](#), and more. Bugs provide a robust way to track test failures and drive accountability in the team to take the required remedial actions. To associate bugs with test results, visit [test report](#) in build or release.

- In the results section of the **Tests** tab select the tests against which the bug should be created and choose **Bug**. Multiple test results can be mapped to a single bug. This is typically done when the reason for the failures is attributable to a single cause such as the unavailability of a dependent service, a database connection failure, or similar issues.



- Open the work item to see the bug. It captures the complete context of the test results including key information such as the error message, stack trace, comments, and more.

NEW BUG *

E2ETest_CategoryWheels Failed in PartsUnlimited.CD_20180106.1

Unassigned 0 comments Add tag

State	New	Area	PartsUnlimited
Reason	New	Iteration	PartsUnlimited\Iteration 3

Repro Steps

B I U A ⌂ ⌂ ⌂ ⌂ ⌂ ⌂ ⌂ ⌂

Test: **FunctionalTests.FunctionalTests.E2ETest_CategoryWheels**
 Priority: not available
 Test file: portale2e-selenium.dll
 Machine: VINFTAGENTTWO
 Tested build: [PartsUnlimited.CI_20180106.1](#)
 Error message: **Test method FunctionalTests.FunctionalTests.E2ETest_CategoryWheels threw exception:**
OpenQA.Selenium.WebDriverException: Unexpected error. System.Net.WebException: Unable to connect to the remote server ---> System.Net.Sockets.SocketException: No connection could be made because the target machine actively refused it 127.0.0.1:50355 at
System.Net.Sockets.Socket.DoConnect(EndPoint endPointSnapshot, SocketAddress socketAddress) at

Planning

Resolved Re
Story Points
Priority
2
Severity
3 - Medium
Activity

- View the bug with the test result, directly in context, within the **Tests** tab. The **Work Items** tab also lists any linked requirements for the test result.

Summary ^

2 Run(s) Completed (0 Passed, 2 Failed, 0 Skipped)

14 Total tests

Category	Count
Passed	4
Failed	10
Skipped	0

Bug ▾ Link

Test

- Multi config tests Chrome_1 (7/7)
- E2ETest_CategoryWheels
- E2ETest_Search

Work items

Bugs (1)

ID	Title	State
4246	E2ETest_CategoryWheels Failed in PartsUnlimited.CD_20180106.1	New

Requirements (2)

ID	Title	State
3262	User can add cash to wallet	New
2628	Create and track support tickets	Resolved

4. From a work item, navigate directly to the associated test results. Both the [test case](#) and the specific [test result](#) are linked to the bug.

BUG 4246

4246 E2ETest_CategoryWheels Failed in PartsUnlimited.CD_20180106.1

Unassigned 0 comments Add tag

State	<input checked="" type="radio"/> New	Area	PartsUnlimited
Reason	<input checked="" type="radio"/> New	Iteration	PartsUnlimited\Iteration 3

Links

+ Add link ▾

Link ↑

Found in build

- Found in build PartsUnlimited.CI_PartsUnlimited.CI_20180106.1

Test

- E2ETest_CategoryWheels

Test Result

- E2ETest_CategoryWheels

5. In the work item, select **Test case** or **Test result** to go directly to the **Tests** page for the selected build or release. You can troubleshoot the failure, update your analysis in the bug, and make the changes required to fix the issue as applicable. While both the links take you to the **Tests tab**, the default section shown are **History** and **Debug** respectively.

PartsUnlimited.CD > PartsUnlimited.CD-141 > QA-Selenium

Pipeline Tasks Variables Logs Tests Deploy Cancel Refresh

Multi config tests Chrome_1 >

Debug Work items Attachments History Bug Link

Error message ^

```
Test method FunctionalTests.FunctionalTests.E2ETest_CategoryWheels
    at OpenQA.Selenium.WebDriverException: Unexpected error. System.Net.WebConnectionException: Failed to establish a network connection to the target machine because the target machine actively refused it.
        at System.Net.Sockets.Socket.DoConnect(EndPoint endPointSnapshot, SocketAddress socketAddress)
        at System.Net.ServicePoint.ConnectSocketInternal(Boolean connectFailure, Socket socket, Socket& socketToUse, IAsyncResult asyncResult, Exception& exception)
--- End of inner exception stack trace ---
    at System.Net.HttpWebRequest.GetRequestStream(TransportContext& context)
```

Source traceability

When troubleshooting test failures that occur consistently over a period of time, it is important to trace back to the initial set of changes - where the failure originated. This can help significantly to narrow down the scope for identifying the problematic test or source under test. To discover the first instance of test failures and trace it back to the associated code changes, visit [Tests tab](#) in build or release.

- In the **Tests** tab, select a test failure to be analyzed. Based on whether it's a build or release, choose the **Failing build** or **Failing release** column for the test.

Summary ^

5 Run(s) Completed (4 Passed, 1 Failed, 0 Not impacted, 0 Others)

89	Total tests		86 Passed 3 Failed 0 Others	96.62% Pass percentage	59m 20s Run duration
----	-------------	---	-----------------------------------	------------------------	----------------------

Bug Link

Filter by test name	Test file	Owner	
Test	Duration	Failing since	Failing release
Not associated (91/91)	0:59:02.646		
AgentBasedPipelineShould...	0:00:24.083		
AgentBasedPipelineShould...	0:02:02.373	13 hours ago	RM.CDP_VSO.RM.CI_master...
AgentRequestOwnerRefer...	0:00:17.187		

2. This opens another instance of the **Tests** tab in a new window, showing the first instance of consecutive failures for the test.

3. Based on the build or release pipeline, you can choose the timeline or pipeline view to see what code changes were committed. You can analyze the code changes to identify the potential root cause of the test failure.

Traditional teams using planned testing

Teams that are moving from manual testing to continuous (automated) testing, and have a subset of tests already automated, can execute them as part of the pipeline or on demand (see [test report](#)). Referred to as **Planned**

testing, automated tests can be [associated to the test cases](#) in a test plan and executed from [Azure Test Plans](#). Once associated, these tests contribute towards the quality metrics of the corresponding requirements.

Help and support

Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

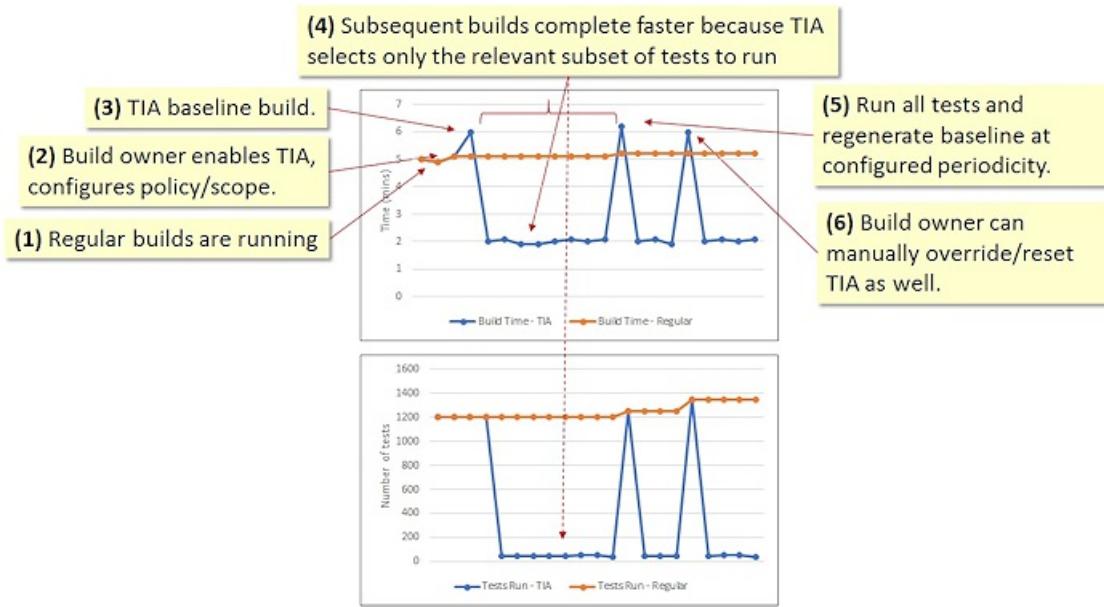
Speed up testing by using Test Impact Analysis (TIA)

10/9/2018 • 6 minutes to read • [Edit Online](#)

Visual Studio 2015.3 and later | TFS 2017.1 and later | Azure Pipelines

Continuous Integration (CI) is a key practice in the industry. Integrations are frequent, and verified with an automated build that runs regression tests to detect integration errors as soon as possible. However, as the codebase grows and matures, its regression test suite tends to grow as well - to the extent that running a full regression test might require hours. This slows down the frequency of integrations, and ultimately defeats the purpose of continuous integration. In order to have a CI pipeline that completes quickly, some teams defer the execution of their longer running tests to a separate stage in the pipeline. However, this only serves to further defeat continuous integration.

Instead, [enable Test Impact Analysis \(TIA\)](#) when using the [Visual Studio Test](#) task in a build pipeline. TIA performs incremental validation by automatic test selection. It will automatically select only the subset of tests required to validate the code being committed. For a given code commit entering the CI/CD pipeline, TIA will select and run only the relevant tests required to validate that commit. Therefore, that test run will complete more quickly, if there is a failure you will get to know about it sooner, and because it is all scoped by relevance, analysis will be faster as well.



Test Impact Analysis has:

- **A robust test selection mechanism.** It includes existing impacted tests, previously failing tests, and newly added tests.
- **Safe fallback.** For commits and scenarios that TIA cannot understand, it will fall back to running all tests. TIA is currently scoped to only managed code, and single machine topology. So, for example, if the code commit contains changes to HTML or CSS files, it cannot reason about them and will fall back to running all tests.
- **Configurable overrides.** You can run all tests at a configured periodicity.

However, be aware of the following caveats when using TIA with Visual Studio 2015:

- **Running tests in parallel.** In this case, tests will run serially.
- **Running tests with code coverage enabled.** In this case, code coverage data will not get collected.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Test Impact Analysis supported scenarios

At present, TIA is supported for:

- TFS 2017 Update 1 onwards, and Azure Pipelines
- Version 2.* of the [Visual Studio Test](#) task in the build pipeline
- Build vNext, with multiple VSTest Tasks
- VS2015 Update 3 onwards on the build agent
- Local and hosted build agents
- CI and in PR workflows
- Git, GitHub, External Git, TFVC repos (including partially mapped TFVC repositories with a [workaround](#))
- IIS interactions (over REST, SOAP APIs), using HTTP/HTTPS protocols
- Automated Tests
- Single machine topology. Tests and app (SUT) must be running on the same machine.
- Managed code (any .NET Framework app, any .NET service)

At present, TIA is **not** supported for:

- Multi-machine topology (where the test is exercising an app deployed to a different machine)
- Data driven tests
- Test Adapter-specific parallel test execution
- .NET Core
- UWP

[More information about TIA scope and applications](#)

Enable Test Impact Analysis

TIA is supported through Version 2.* of the [Visual Studio Test](#) task. If your app is a single tier application, all you need to do is to check **Run only impacted tests** in the task UI. The Test Impact data collector is automatically configured. No additional steps are required.

Visual Studio Test [?](#)

Version 2.* [▼](#)

Display name [*](#)
VsTest - testAssemblies

Test selection [^](#)

Select tests using [*](#) [?](#)
Test assemblies [▼](#)

Test assemblies [*](#) [?](#)
***test*.dll
!***TestAdapter.dll
!**\obj**

Search folder [*](#) [?](#)
\$(System.DefaultWorkingDirectory)

Test filter criteria [?](#)

Run only impacted tests [?](#)

Number of builds after which all tests should be run [?](#)
50

Test mix contains UI tests [?](#)

Execution options [^](#)

If your application interacts with a service in the context of IIS, you must also configure the Test Impact data collector to run in the context of IIS by using a **.runsettings** file. Here is a sample that creates this configuration:

```
<?xml version="1.0" encoding="utf-8"?>
<RunSettings>
  <DataCollectionRunSettings>
    <DataCollectors>
      <!-- This is the TestImpact data collector.-->
      <DataCollector uri="datacollector://microsoft/TestImpact/1.0"
        assemblyQualifiedName="Microsoft.VisualStudio.TraceCollector.TestImpactDataCollector,
        Microsoft.VisualStudio.TraceCollector, Version=15.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
        friendlyName="Test Impact">
        <Configuration>
          <!-- enable IIS data collection-->
          <InstrumentIIS>True</InstrumentIIS>
          <!-- file level data collection -->
          <ImpactLevel>file</ImpactLevel>
          <!-- any job agent related executable or any other service that the test is using needs to be
          profiled. -->
          <ServicesToInstrument>
            <Name>TeamFoundationSshService</Name>
          </ServicesToInstrument>
        </Configuration>
      </DataCollector>
    </DataCollectors>
  </DataCollectionRunSettings>
</RunSettings>
```

View Test Impact Analysis outcome

TIA is integrated into existing test reporting at both the summary and details levels, including notification emails.

The screenshot shows the Azure Pipelines build summary for the 'AspNetWebAppSample-Azure Web App for A' project. The build status is green with a checkmark icon. The pipeline step 'Build artifacts published' is shown with 1 artifact drop. The 'Tests succeeded' step is highlighted with a red border, indicating 100% passed. Below it, a summary table shows 1 Run(s) completed with 1 Passed, 0 Failed, 0 Not impacted, and 0 Others. A donut chart shows 100% Pass percentage. The total tests were 2, with 2 Passed, 0 Failed, and 0 Others. The run duration was 1s 6ms, which increased by +1s 6ms from the previous run.

Run	Status	Passed	Failed	Not Impacted	Others
1	Passed	1	0	0	0

2 Total tests
+2
100% Pass percentage
1s 6ms Run duration
↑ +1s 6ms

The screenshot shows the Azure Pipelines build summary for the 'AspNetWebAppSample-Azure Web App for A' project. The build status is green with a checkmark icon. The pipeline step 'Build artifacts published' is shown with 1 artifact drop. The 'Tests succeeded' step is highlighted with a red border, indicating 100% passed. Below it, a summary table shows 1 Run(s) completed with 1 Passed, 0 Failed, 0 Not impacted, and 0 Others. A donut chart shows 100% Pass percentage. The total tests were 2, with 2 Passed, 0 Failed, and 0 Others. The run duration was 1s 674ms, which increased by +1s 674ms from the previous run.

Run	Status	Passed	Failed	Not Impacted	Others
1	Passed	1	0	0	0

2 Total tests
+2
100% Pass percentage
1s 674ms Run duration
↑ +1s 674ms

[More information about TIA and Azure Pipelines integration](#)

Manage Test Impact Analysis behavior

You can influence the way that tests are either included or ignored during a test run:

- **Through the VSTest task UI.** TIA can be conditioned to run all tests at a configured periodicity. Setting this option is recommended, and is the means to regulate test selection.

- **By setting a build variable.** Even after TIA has been enabled in the VSTest task, it can be disabled for a specific build by setting the variable **DisableTestImpactAnalysis** to **true**. This override will force TIA to run all tests for that build. In subsequent builds, TIA will go back to optimized test selection.

When TIA opens a commit and sees an unknown file type, it falls back to running all tests. While this is good from a safety perspective, tuning this behavior might be useful in some cases. For example:

- Set the **TI_IncludePathFilters** variable to specific paths to include only these paths in a repository for which you want TIA to apply. This is useful when teams use a shared repository. Setting this variable disables TIA for all other paths not included in the setting.
- Set the **TIA_IncludePathFilters** variable to specify file types that do not influence the outcome of tests and for which changes should be ignored. For example, to ignore changes to .csproj files set the variable to the value **!***.csproj**.

Use the [minimatch pattern](#) when setting variables, and separate multiple items with a semicolon.

To evaluate whether TIA is selecting the appropriate tests:

- Manually validate the selection. A developer who knows how the SUT and tests are architected could manually validate the test selection using the [TIA reporting capabilities](#).
- Run TIA selected tests and then all tests in sequence. In a build pipeline, use two test tasks - one that runs only impacted Tests (T1) and one that runs all tests (T2). If T1 passes, check that T2 passes as well. If there was a failing test in T1, check that T2 reports the same set of failures.

[More information about TIA advanced configuration](#)

Provide custom dependency mappings

TIA uses dependency maps of the following form.

```
TestMethod1
dependency1
dependency2
TestMethod2
dependency1
dependency3
```

TIA can generate such a dependencies map for managed code execution. Where such dependencies reside in **.cs** and **.vb** files, TIA can automatically watch for commits into such files and then run tests that had these source files in their list of dependencies.

You can extend the scope of TIA by explicitly providing the dependencies map as an XML file. For example, you might want to support code in other languages such as JavaScript or C++, or support the scenario where tests and product code are running on different machines. The mapping can even be approximate, and the set of tests you want to run can be specified in terms of a test case filter such as you would typically provide in the VSTest task parameters.

The XML file should be checked into your repository, typically at the root level. Then set the build variable **TIA.UserMapFile** to point to it. For example, if the file is named **TIAmap.xml**, set the variable to **\$(System.DefaultWorkingDirectory)/TIAmap.xml**.

For an example of the XML file format, see [TIA custom dependency mapping](#).

See Also

- [TIA overview and VSTS integration](#)

- [TIA scope and applications](#)
- [TIA advanced configuration](#)
- [TIA custom dependency mapping](#)

Help and support

Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Run tests in parallel for any test runner

10/9/2018 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018.2 and later

Running tests to validate changes to code is key to maintaining quality. For continuous integration practice to be successful, it is essential you have a good test suite that is run with every build. However, as the codebase grows, the regression test suite tends to grow as well and running a full regression test can take a long time. Sometimes, tests themselves may be long running - this is typically the case if you write end-to-end tests. This reduces the speed with which customer value can be delivered as pipelines cannot process builds quickly enough.

Running tests in parallel is a great way to improve the efficiency of CI/CD pipelines. This can be done easily by employing the additional capacity offered by the cloud. This article discusses how you can parallelize tests by using multiple agents to process jobs.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called service *endpoints* in TFS 2018 and in older versions.

Pre-requisite

Familiarize yourself with the concepts of [agents](#) and [jobs](#). Each agent can run only one job at a time. To run multiple jobs in parallel, you must configure multiple agents. You also need sufficient [parallel jobs](#).

Setting up parallel jobs

Specify 'parallel' strategy in the YAML and indicate how many jobs should be dispatched. The variables

`System.JobPositionInPhase` and `System.TotalJobsInPhase` are added to each job.

```
jobs:  
- job: ParallelTesting  
  strategy:  
    parallel: 2
```

TIP

You can specify as many as 99 agents to scale up testing for large test suites.

Slicing the test suite

To run tests in parallel you must first slice (or partition) the test suite so that each slice can be run independently. For example, instead of running a large suite of 1000 tests on a single agent, you can use two agents and run 500 tests in parallel on each agent. Or you can reduce the amount of time taken to run the tests even further by using 8 agents and running 125 tests in parallel on each agent.

The step that runs the tests in a job needs to know which test slice should be run. The variables

`System.JobPositionInPhase` and `System.TotalJobsInPhase` can be used for this purpose:

- `System.TotalJobsInPhase` indicates the total number of slices (you can think of this as "totalSlices")

- `System.JobPositionInPhase` identifies a particular slice (you can think of this as "sliceNum")

If you represent all test files as a single dimensional array, each job can run a test file indexed at [sliceNum + totalSlices], until all the test files are run. For example, if you have six test files and two parallel jobs, the first job (slice0) will run test files numbered 0, 2, and 4, and second job (slice1) will run test files numbered 1, 3, and 5.



If you use three parallel jobs instead, the first job (slice0) will run test files numbered 0 and 3, the second job (slice1) will run test files numbered 1 and 4, and the third job (slice2) will run test files numbered 2 and 5.



Sample code

This Python sample uses a PowerShell script to slice the tests. The tests are run using pytest. JUnit-style test results created by pytest are then published to the server. Import (into Azure Repos or TFS) or fork (into GitHub) this repo:

```
https://github.com/PBoraMSFT/ParallelTestingSample-Python
```

This JavaScript sample uses a bash script to slice the tests. The tests are run using the mocha runner. JUnit-style test results created by mocha are then published to the server. Import (into Azure Repos or TFS) or fork (into GitHub) this repo:

```
https://github.com/PBoraMSFT/ParallelTestingSample-Mocha
```

The sample code includes a file `azure-pipelines.yml` at the root of the repository that you can use to create a pipeline. Follow all the instructions in [Create your first pipeline](#) to create a pipeline and see test slicing in action.

Combine parallelism for massively parallel testing

When parallel jobs are used in a pipeline, the pipeline employs multiple machines to run each job in parallel. Most test runners provide the capability to run tests in parallel on a single machine (typically by creating multiple processes or threads that are run in parallel). The two types of parallelism can be combined for massively parallel testing, which makes testing in pipelines extremely efficient.

Help and support

Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Run tests in parallel using the Visual Studio Test task

11/19/2018 • 8 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2017.1 and later

Running tests to validate changes to code is key to maintaining quality. For continuous integration practice to be successful, it is essential you have a good test suite that is run with every build. However, as the codebase grows, the regression test suite tends to grow as well and running a full regression test can take a long time. Sometimes, tests themselves may be long running - this is typically the case if you write end-to-end tests. This reduces the speed with which customer value can be delivered as pipelines cannot process builds quickly enough.

Running tests in parallel is a great way to improve the efficiency of CI/CD pipelines. This can be done easily by employing the additional capacity offered by the cloud. This article discusses how you can configure the [Visual Studio Test task](#) to run tests in parallel by using multiple agents.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

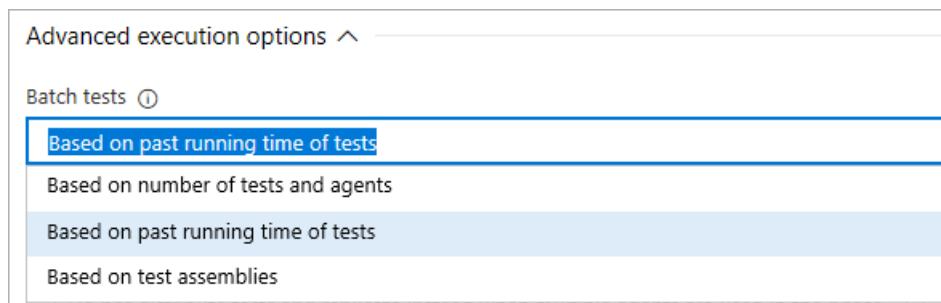
Pre-requisite

Familiarize yourself with the concepts of [agents](#) and [jobs](#). To run multiple jobs in parallel, you must configure multiple agents. You also need sufficient [parallel jobs](#).

Test slicing

The Visual Studio Test task (version 2) is designed to work seamlessly with parallel job settings. When a pipeline job that contains the Visual Studio Test task (referred to as the "VSTest task" for simplicity) is configured run on multiple agents in parallel, it automatically detects that multiple agents are involved and creates test slices that can be run in parallel across these agents.

The task can be configured to create test slices to suit different requirements such as batching based on the number of tests and agents, the previous test running times, or the location of tests in assemblies.



These options are explained in the following sections.

Simple slicing based on the number of tests and agents

This setting uses a simple slicing algorithm to divide up the number of tests 'T' across 'N' agents so that each agent runs T/N tests. For example, if your test suite contains 1000 tests, and you use two agents for parallel jobs, each agent will run 500 tests. Or you can reduce the amount of time taken to run the tests even further by using eight agents, in which case each agent runs 125 tests in parallel.

This option is typically used when all tests have similar running times. If test running times are not similar, agents may not be utilized effectively because some agents may receive slices with several long-running tests, while other agents may receive slices with short-running tests and finish much earlier than the rest of the agents.

Slicing based on the past running time of tests

This setting considers past running times to create slices of tests so that each slice has approximately the same running time. Short-running tests will be batched together, while long-running tests will be allocated to separate slices.

This option should be used when tests within an assembly do not have dependencies, and do not need to run on the same agent. This option results in the most efficient utilization of agents because every agent gets the same amount of 'work' and all finish at approximately the same time.

Slicing based on test assemblies

This setting uses a simple slicing algorithm that divides up the number of test assemblies (or files) 'A' across 'N' agents, so that each agent runs tests from A/N assemblies. The number of tests within an assembly is not taken into account when using this option. For example, if your test suite contains ten test assemblies and you use two agents for parallel jobs, each agent will receive five test assemblies to run. You can reduce the amount of time taken to run the tests even further by using five agents, in which case each agent gets two test assemblies to run.

This option should be used when tests within an assembly have dependencies or utilize `AssemblyInitialize` and `AssemblyCleanup`, or `ClassInitialize` and `ClassCleanup` methods, to manage state in your test code.

Run tests in parallel in build pipelines

If you have a large test suite or long-running integration tests to run in your build pipeline, use the following steps.

NOTE

To use the multi-agent capability in build pipelines with on-premises TFS server, you must use TFS 2018 Update 2 or a later version.

1. **Build job using a single agent.** Build Visual Studio projects and publish build artifacts using the tasks shown in the following image. This uses the default job settings (single agent, no parallel jobs).

Pipeline Build pipeline

Get sources Partsunlimited master

Build job uses 1 agent Run on agent

- Use NuGet 4.4.1 NuGet Tool Installer
- NuGet restore NuGet
- Build solution PartsUnlimited.sln Visual Studio Build
- Publish symbols path Index Sources & Publish Symbols
- Copy Files to: \$(build.artifactstagingdirectory) Copy Files
- Publish Artifact: drop Publish Build Artifacts

Agent job ①

Display name *

Build job uses 1 agent

Agent selection ^

Agent pool ① | Manage ↗

<inherit from pipeline>

Demands ①

Name	Count
msbuild	0
visualstudio	0

+ Add

Execution plan ^

Parallelism ①

None Multi-configuration Multi-agent

2. Run tests in parallel using multiple agents:

- Add an **agent job**

Tasks Variables Triggers Options Retention History | Save & queue

Pipeline Build pipeline

Get sources Partsunlimited master

Build job uses 1 agent Run on agent

Use NuGet 4.4.1 NuGet Tool Installer

... Add an agent job

Add an agentless job

Learn more about jobs ↗ Hosted VS2017

Parameters ①

- Configure the job to use **multiple agents in parallel**. The example here uses three agents.

Pipeline
Build pipeline

- Get sources** Partsunlimited master
- Build job uses 1 agent** Run on agent
- Use NuGet 4.4.1** NuGet Tool Installer
- NuGet restore** NuGet
- Build solution PartsUnlimited.sln** Visual Studio Build
- Publish symbols path** Index Sources & Publish Symbols
- Copy Files to: \$(build.artifactstagingdirectory)** Copy Files
- Publish Artifact: drop** Publish Build Artifacts
- Parallel test job uses multiple agents** Run on agent
- Download Build Artifacts** Download Build Artifacts
- VsTest - testAssemblies** Visual Studio Test

Agent job ①

Display name * Parallel test job uses multiple agents

Agent selection ^ Agent pool ① | Manage ↗ <inherit from pipeline>

Demands ①

Name	Condition
vstest	exists

+ Add

Execution plan ^

Parallelism ①

None Multi-configuration Multi-agent

Number of agents * ① 3

Continue on error

Timeout * ① 0

TIP

For massively parallel testing, you can specify as many as 99 agents.

- Add a **Download Build Artifacts** task to the job. This step is the link between the build job and the test job, and is necessary to ensure that the binaries generated in the build job are available on the agents used by the test job to run tests. Ensure that the task is set to download artifacts produced by the 'Current build' and the artifact name is the same as the artifact name used in the **Publish Build Artifacts** task in the build job.

The screenshot shows the Azure DevOps Pipeline editor. On the left, a list of tasks is displayed:

- Get sources (Partsunlimited, master)
- Build job uses 1 agent (Run on agent)
- Use NuGet 4.4.1 (NuGet Tool Installer)
- NuGet restore (NuGet)
- Build solution PartsUnlimited.sln (Visual Studio Build)
- Publish symbols path (Index Sources & Publish Symbols)
- Copy Files to: \$(build.artifactstagingdirectory) (Copy Files)
- Publish Artifact: drop** (Publish Build Artifacts)
- Parallel test job uses multiple agents (Run on agent)
- Download Build Artifacts** (Download Build Artifacts)
- VsTest - testAssemblies (Visual Studio Test)

On the right, configuration details for the "Download Build Artifacts" task are shown:

- Version: 0.*
- Display name: Download Build Artifacts
- Download artifacts produced by:
 - Current build
 - Specific build
- Download type:
 - Specific artifact
 - Specific files
- Artifact name: drop (highlighted with a red box)
- Matching pattern: **
- Destination directory: \$(System.ArtifactsDirectory)
- Advanced, Control Options, and Output Variables sections are also visible.

- Add the **Visual Studio Test** task and configure it to use the required [slicing strategy](#).

Run tests in parallel in release pipelines

Use the following steps if you have a large test suite or long-running functional tests to run after deploying your application. For example, you may want to deploy a web-application and run Selenium tests in a browser to validate the app functionality.

NOTE

To use the multi-agent capability in release pipelines with on-premises TFS server, you must use TFS 2017 Update 1 or a later version.

1. **Deploy app using a single agent.** Use the tasks shown in the image below to deploy a web app to Azure App Services. This uses the default job settings (single agent, no parallel jobs).

Pipeline Tasks Variables Retention Options History

QA
Deployment process

Deploy app job - 1 agent
Run on agent

Azure Deployment:Create Or Update Resource G...
Azure Resource Group Deployment

Azure App Service Deploy: \$(WebsiteName)
Azure App Service Deploy

Parallel test job using multiple agents - Selenium tests
Run on agent

Pre-test step using Powershell
PowerShell

VsTest - testAssemblies
Visual Studio Test

Agent job ⓘ

Display name *
Deploy app job - 1 agent

Agent selection ^

Agent pool ⓘ | Pool information | Manage
Hosted VS2017

Demands ⓘ

Name

+ Add

Execution plan ^

Parallelism ⓘ

None Multi-configuration N

Timeout * ⓘ
0

2. Run tests in parallel using multiple agents:

- Add an **agent job**

Pipeline Tasks Variables Retention Options History

QA
Deployment process

Deploy app job - 1 agent
Run on agent

Azure Deployment:Create Or Update Resou
Azure Resource Group Deployment

Azure App Service Deploy: \$(WebsiteName)
Azure App Service Deploy

...

Stage name

+ Add an agent job

Add a deployment group job

Add an agentless job

Learn more about jobs ↗

- Configure the job to use **multiple agents in parallel**. The example here uses three agents.

The screenshot shows the Azure DevOps Pipeline editor. On the left, a list of tasks is visible: QA (Deployment process), Deploy app job - 1 agent (Run on agent), Azure Deployment:Create Or Update Resource (Azure Resource Group Deployment), Azure App Service Deploy: \$(WebsiteName) (Azure App Service Deploy), Parallel test job using multiple agents - Selenium tests (Run on agent), Pre-test step using Powershell (PowerShell), and VsTest - testAssemblies (Visual Studio Test). The 'Parallel test job using multiple agents - Selenium tests' task is currently selected. On the right, the configuration pane shows the 'Agent job' settings. Under 'Parallelism', the 'Multi-agent' option is selected, and the 'Number of agents' field is set to 3.

TIP

For massively parallel testing, you can specify as many as 99 agents.

- Add any **additional tasks** that must run before the Visual Studio test task is run. For example, run a PowerShell script to set up any data required by your tests.

TIP

Jobs in release pipelines download all artifacts linked to the release pipeline by default. To save time, you can configure the job to download only the test artifacts required by the job. For example, web app binaries are not required to run Selenium tests and downloading these can be skipped if the app and test artifacts are published separately by your build pipeline.

- Add the **Visual Studio Test** task and configure it to use the required [slicing strategy](#).

TIP

If the test machines do not have Visual Studio installed, you can use the [Visual Studio Test Platform Installer task](#) to acquire the required version of the test platform.

Massively parallel testing by combining parallel pipeline jobs with parallel test execution

When parallel jobs are used in a pipeline, it employs multiple machines (agents) to run each job in parallel. Test frameworks and runners also provide the capability to run tests in parallel on a single machine, typically by creating multiple processes or threads that are run in parallel. Parallelism features can be combined in a layered fashion to achieve massively parallel testing. In the context of the [Visual Studio Test task](#), parallelism can be

combined in the following ways:

1. **Parallelism offered by test frameworks.** All modern test frameworks such as MSTest v2, NUnit, xUnit, and others provide the ability to run tests in parallel. Typically, tests in an assembly are run in parallel. These test frameworks interface with the Visual Studio Test platform using a test adapter and the test framework, together with the corresponding adapter, and work within a test host process that the Visual Studio Test Platform creates when tests are run. Therefore, parallelization at this layer is within a process for all frameworks and adapters.
2. **Parallelism offered by the Visual Studio Test Platform (`vstest.console.exe`).** Visual Studio Test Platform can run test assemblies in parallel. Users of `vstest.console.exe` will recognize this as the [/parallel switch](#). It does so by launching a test host process on each available core, and handing it tests in an assembly to execute. This works for any framework that has a test adapter for the Visual Studio test platform because the unit of parallelization is a test assembly or test file. This, when combined with the parallelism offered by test frameworks (described above), provides the maximum degree of parallelization when tests run on a single agent in the pipeline.
3. **Parallelism offered by the Visual Studio Test (VSTest) task.** The VSTest task supports running tests in parallel across multiple agents (or machines). Test slices are created, and each agent executes one slice at a time. The three different [slicing strategies](#), when combined with the parallelism offered by the test platform and test framework (as described above), result in the following:
 - Slicing based on the number of tests and agents. Simple slicing where tests are grouped in equally sized slices. A slice contains tests from one or more assemblies. Test execution on the agent then conforms to the parallelism described in **1** and **2** above.
 - Slicing based on past running time. Based on the previous timings for running tests, and the number of available agents, tests are grouped into slices such that each slice requires approximately equal execution time. A slice contains tests from one or more assemblies. Test execution on the agent then conforms to the parallelism described in **1** and **2** above.
 - Slicing based on assemblies. A slice is a test assembly, and so contains tests that all belong to the same assembly. Execution on the agent then conforms to the parallelism described in **1** and **2** above. However, **2** may not occur if an agent receives only one assembly to run.

Help and support

Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support page](#).

UI testing considerations

11/19/2018 • 5 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2017.1 and later

When running automated tests in the CI/CD pipeline, you may need a special configuration in order to run UI tests such as Selenium, Appium or Coded UI tests. This topic describes the typical considerations for running UI tests.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

Pre-requisites

Familiarize yourself with [agents](#) and [deploying an agent on Windows](#).

Headless mode or visible UI mode?

When running Selenium tests for a web app, you can launch the browser in two ways:

1. **Headless mode.** In this mode, the browser runs as normal but without any UI components being visible. While this mode is obviously not useful for browsing the web, it is useful for running automated tests in an unattended manner in a CI/CD pipeline. [Chrome](#) and [Firefox](#) browsers can be run in headless mode.

This mode generally consumes less resources on the machine because the UI is not rendered and tests run faster. As a result, potentially more tests can be run in parallel on the same machine to reduce the total test execution time.

Screenshots can be captured in this mode and used for troubleshooting failures.

NOTE

Microsoft Edge browser currently cannot be run in the headless mode. To follow developments in this space, see [this user voice item](#).

2. **Visible UI mode.** In this mode, the browser runs normally and the UI components are visible. When running tests in this mode on Windows, [special configuration of the agents](#) is required.

If you are running UI tests for a desktop application, such as [Appium tests using WinAppDriver](#) or Coded UI tests, a [special configuration of the agents](#) is required.

TIP

End-to-end UI tests generally tend to be long-running. When using the visible UI mode, depending on the test framework, you may not be able to run tests in parallel on the same machine because the app must be in focus to receive keyboard and mouse events. In this scenario, you can speed up testing cycles by running tests in parallel on *different* machines. See [run tests in parallel for any test runner](#) and [run tests in parallel using Visual Studio Test task](#).

UI testing in visible UI mode

A special configuration is required for agents to run UI tests in visible UI mode.

Visible UI testing using Microsoft-hosted agents

Microsoft-hosted agents are pre-configured for UI testing and UI tests for both web apps and desktop apps.

Microsoft-hosted agents are also pre-configured with [popular browsers and matching web-driver versions](#) that can be used for running Selenium tests. The browsers and corresponding web-drivers are updated on a periodic basis. To learn more about running Selenium tests, see [UI test with Selenium](#)

Visible UI testing using self-hosted Windows agents

Agents that are configured to run as service can run Selenium tests only with headless browsers. If you are not using a headless browser, or if you are running UI tests for desktop apps, Windows agents *must* be configured to run as an interactive process with auto-logon enabled.

When configuring agents, select 'No' when prompted to run as a service. Subsequent steps then allow you to configure the agent with auto-logon. When your UI tests run, applications and browsers are launched in the context of the user specified in the auto-logon settings.

If you use Remote Desktop to access the computer on which an agent is running with auto-logon, simply disconnecting the Remote Desktop causes the computer to be locked and any UI tests that run on this agent may fail. To avoid this, use the [tscon](#) command on the remote computer to disconnect from Remote Desktop. For example:

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

In this example, the number '1' is the ID of the remote desktop session. This number may change between remote sessions, but can be viewed in Task Manager. Alternatively, to automate finding the current session ID, create a batch file containing the following code:

```
for /f "skip=1 tokens=3" %%s in ('query user %USERNAME%') do (
    %windir%\System32\tscon.exe %%s /dest:console
)
```

Save the batch file and create a desktop shortcut to it, then change the shortcut properties to 'Run as administrator'. Running the batch file from this shortcut disconnects from the remote desktop but preserves the UI session and allows UI tests to run.

Provisioning agents in Azure VMs for UI testing

If you are provisioning virtual machines (VMs) on Azure, agent configuration for UI testing is available through the [Agent artifact for DevTest Labs](#).

The screenshot shows two windows side-by-side. The left window is titled 'Add artifacts' and lists several Microsoft artifacts. The right window is titled 'Add artifact' and is specifically for configuring a 'Build Agent'. It includes fields for 'Account Name', 'Personal Access Token', 'Agent Name', 'Agent Name Suffix', 'Agent Pool', and a dropdown for 'Enable Autologon' which is highlighted with a red box. The 'Enable Autologon' dropdown contains the following options:

- false
- true
- false
- INT AUTHORITY\NETWORKSERVICE

Setting screen resolution

Before running UI tests you may need to adjust the screen resolution so that apps render correctly. For this, a [screen resolution utility task](#) is available from Marketplace. Use this task in your pipeline to set the screen resolution to a value that is supported by the agent machine. By default, this utility sets the resolution to the optimal value supported by the agent machine.

If you encounter failures using the screen resolution task, ensure that the agent is configured to run with auto-logon enabled and that all remote desktop sessions are safely disconnected using the **tscon** command as described above.

NOTE

The screen resolution utility task runs on the unified build/release/test agent, and cannot be used with the deprecated [Run Functional Tests task](#).

Troubleshooting failures in UI tests

When you run UI tests in an unattended manner, capturing diagnostic data such as [screenshots](#) or [video](#) is useful for discovering the state of the application when the failure was encountered.

Capture screenshots

Most UI testing frameworks provide the ability to capture screenshots. The screenshots collected are available as an attachment to the test results when these results are published to the server.

If you use the [Visual Studio test task](#) to run tests, captured screenshots must be added as a result file in order to be available in the test report. For this, use the following code:

- [MSTest](#)
- [NUnit](#)

First, ensure that `TestContext` is defined in your test class. For example:

```
public TestContext TestContext { get; set; }
```

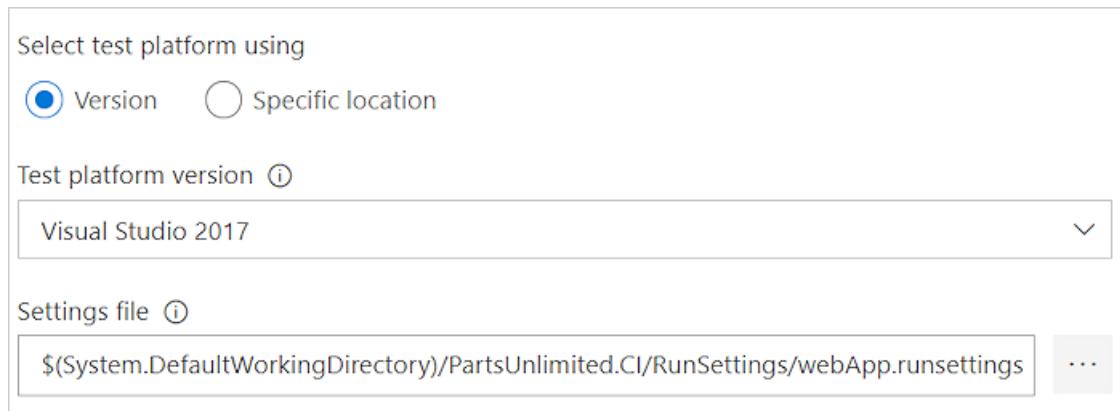
Add the screenshot file using `TestContext.AddResultFile(fileName);` //Where `fileName` is the name of the file.

NOTE

If you use the [Publish Test Results task](#) to publish results, test result attachments can only be published if you are using the VSTest (TRX) results format or the [NUnit 3.0 results](#) format. Result attachments cannot be published if you use JUnit or xUnit test results.

Capture video

If you use the [Visual Studio test task](#) to run tests, video of the test can be captured and is automatically available as an attachment to the test result. For this, you must configure the [video data collector in a .runsettings file](#) and this file must be specified in the task settings.



Help and support

Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

FAQs for continuous testing and test automation

10/16/2018 • 12 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

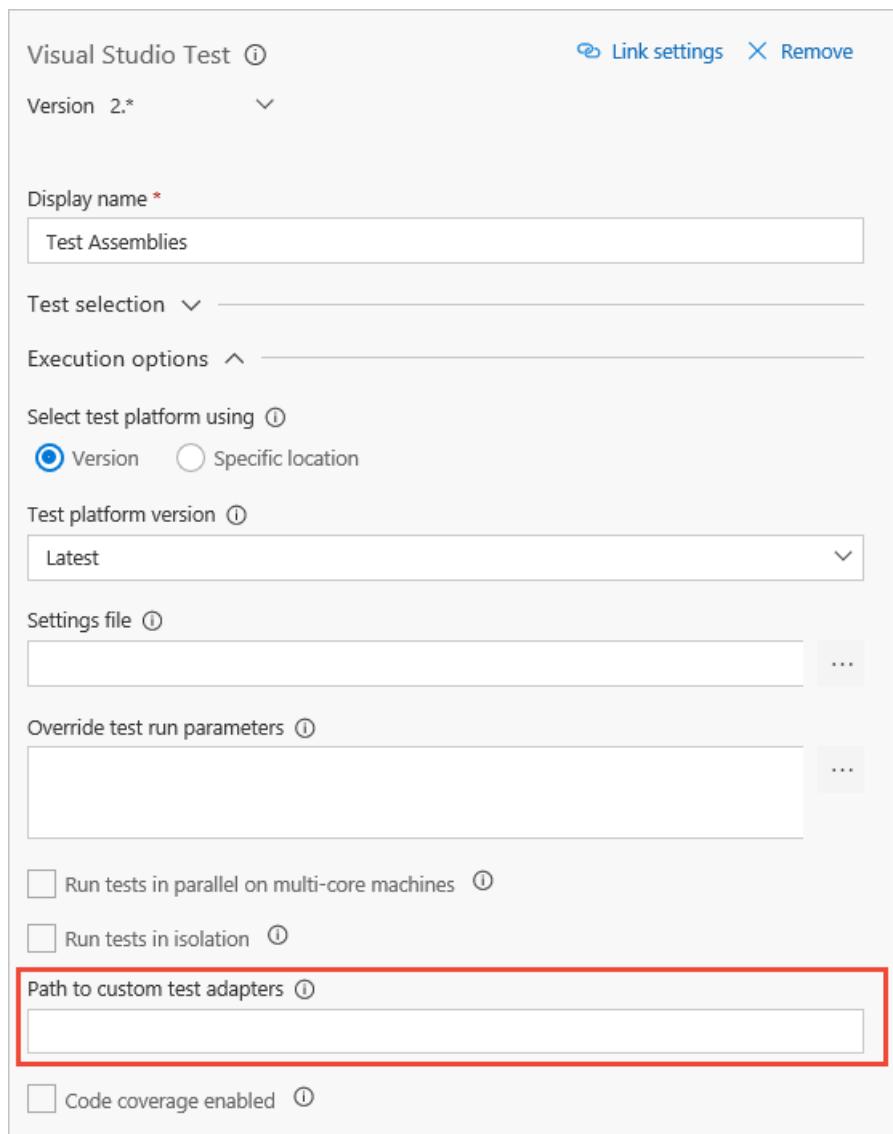
General topics

[Go to related topic >](#)

Q: How do I run tests from different unit test frameworks with my build?

A: First, set up your test frameworks in your development tool. For example, in Visual Studio:

1. [Set up the plug-in for your test framework](#), if you haven't already.
2. Create a folder that has the custom binaries for your plug-in. (The plug-in package for your framework is a .vsix file. Change the extension to .zip so that you can unzip it.)
3. Add these assemblies to version control and [let the build controller know where to get them](#).
4. In your build pipeline, provide the path to the test framework in the *Path to Custom Test Adapters** parameter:



Q: I'm having problems using xUnit with .NET Core apps. Where can I get more information?

A: See the blog post [Unit Tests with .NET Core and VSTS](#).

Q: I am not using the Visual Studio Test task to run my unit tests. Can I still collect and publish code coverage data?

A: Yes, use the [Publish Code Coverage Results task](#).

Q: I have multiple Publish Code Coverage Results tasks in my pipeline. Do I get a merged code coverage summary?

A: Code coverage is automatically merged for only Visual Studio coverage (.coverage) files. A merged summary is not currently available for coverage files published using multiple Publish Code Coverage Results tasks.

Q: What are the typical types of tests I can run to validate my app and deployment?

A: Testing in a continuous integration and continuous deployment (CI/CD) scenario typically consists of:

1. Run **unit tests** in the CI pipeline, as demonstrated in the example above. The **Visual Studio Test** task automatically runs tests included in the app assemblies, but there is a wide range of configuration options you can specify, such as running only specific tests. See [Visual Studio Test task](#).
2. Run **functional tests** in the early stages of the CD pipeline. These are typically [Selenium](#) (for web apps) and [Coded UI](#) tests. Use version 2.x or higher of the [Visual Studio Test task](#) together with [jobs](#) to run unit and functional tests on the universal agent. See [Run tests in parallel using the Visual Studio Test task](#).
3. Run **load tests** after the app is deployed to staging and production, after it passes all functional tests. The example shown above is just a simple test that accesses a single page in the web app to validate that

deployment succeeded and the app is running successfully. You can perform more comprehensive load testing to validate the entire app by running [cloud-based load tests](#) and [Apache JMeter load tests](#).

Q: Can I find a specific test run?

A: Yes, type the Run ID into the search box in the left column:

The screenshot shows a user interface for managing test runs. On the left, there's a sidebar with a search bar containing '102640' and a 'Go' button. Below the search bar are buttons for 'Recent test runs' and 'Test runs'. Under 'Test runs', a list item '102640: VSTest...' is highlighted. The main area is titled 'Recent test runs' and contains a table with columns: State, Run Id, Title, Completed Date, and Build Number. Two rows are shown, both marked as 'Completed':
1. Run Id 108980, Title 'VSTest Test Run', Completed Date 7/16/2015 9:55:21 PM, Build Number Fabrikam.CI_20150716.4
2. Run Id 108979, Title 'VSTest Test Run', Completed Date 7/16/2015 9:50:26 PM, Build Number Fabrikam.CI_20150716.3

Q: Can I find specific results from a test run?

A: Yes, after you find your test run, create a query in the **Filter** tab to find the test results you want:

The screenshot shows a detailed view of a specific test run. On the left, there's a sidebar with a search bar for 'Enter Run Id...', a 'Go' button, and a list of recent test runs. The 'Test runs' section is expanded, showing '102640: VSTest...'. The main area is titled 'Run 102640 - VSTest Test Run' and has tabs for 'Run summary', 'Test results', and 'Filter'. The 'Filter' tab is selected, showing a query builder with a single clause: 'Outcome = Failed'. Below the filter, a table lists test cases with their outcomes:
1. Failed: VerifyPackageShouldThrowArgumentNullExceptionIfPackageDescriptorIsNull
2. Failed: QueueDeleteProjectShouldSucceedForValidParameters
3. Failed: PublishPackageShouldThrowArgumentNullExceptionIfPackageDescriptorIsNull

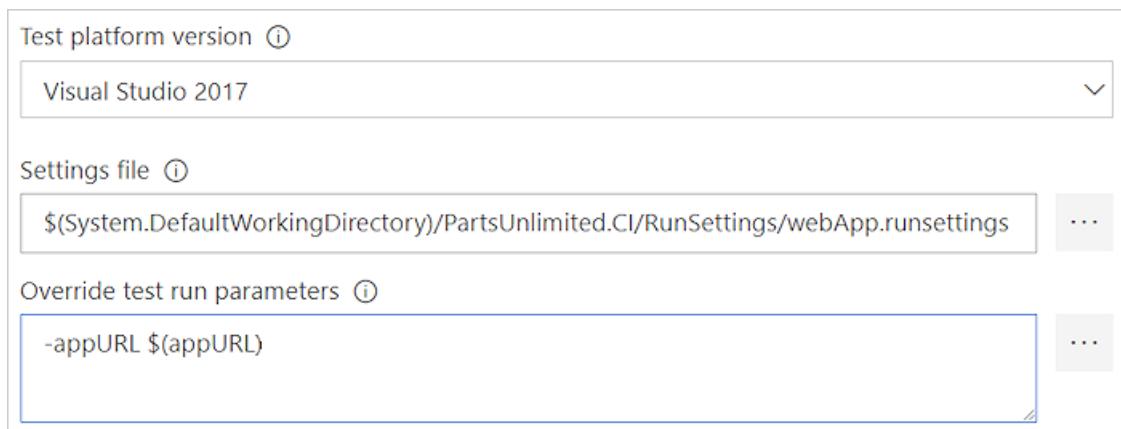
Q: Can I deploy to a staging slot first, and then to production?

A: Yes, you can create additional deployment slots in Azure Web Apps, and specify which slot to deploy your app to. If you do not specify a slot, the default **Production** slot is used. After you deploy, you can swap an app to a different slot using the [Azure App Service Manage](#) task. See [Azure Web Apps deployment](#).

You can use [task jobs](#) and the **Manual Intervention** task in your release pipeline to pause a deployment; for example, to examine test results after the load tests have run and before the app is swapped from staging to production.

Q: How do I pass parameters to my test code from a build or release pipeline?

A: Use a [runsettings file](#) to pass values as parameters to your test code. For example, in a release that contains several stages, you can pass the appropriate app URL to each the test tasks in each one. The runsettings file and matching parameters must be specified in the [Visual Studio Test task](#).



Q: Where can I find details about configuring test agents?

A: See [Install and configure test agents](#)

Q: What if I want to run debug builds of native (.cpp) unit tests on the machine with the test agent?

A: Make sure that you have debug versions of the Universal C Runtime (UCRT) on the machine with the test agent, specifically these libraries: ucrtbased.dll and vcruntime140d.dll. You can include these items with your deployment. If you're running release builds of .cpp unit tests, make sure that you have Windows Update KB2999226 on your test agent machine.

Q: Where can I learn more about integrating tests with my build?

A: Try these blog posts and videos:

- [Configuring Continuous Integration and Continuous Testing with Visual Studio](#)
- [Testing in Continuous Integration and Continuous Deployment Workflows](#)
- [Integrating Testing Efforts into the DevOps Process with Build vNext and Visual Studio Release Management](#)

Associating tests with test cases

[Go to related topic >](#)

Q: What are the differences if I am still using a XAML build?

A: If you are using a XAML build in Azure Pipelines or TFS, you can run tests that you have associated in a Build-Deploy-Test workflow using a [Lab environment](#). You can also run tests using Microsoft Test Manager and a [Lab environment](#).

Q: What types of tests are supported?

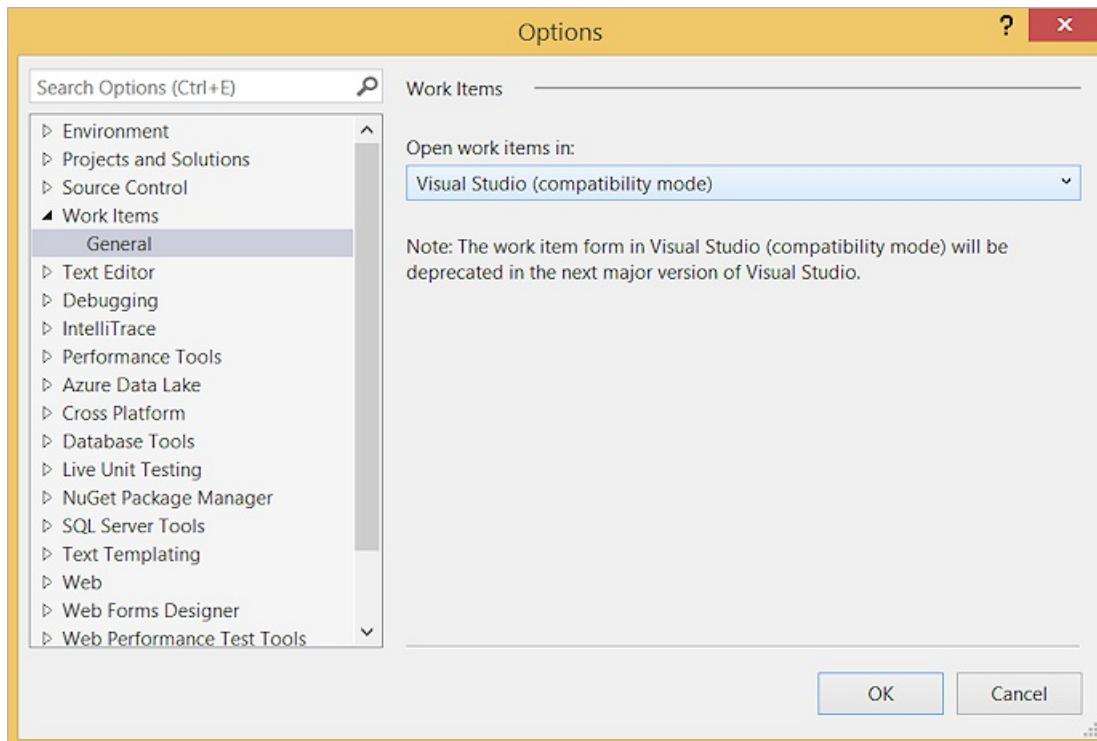
A: These are the limitations for each type of test:

- Coded UI test, Selenium tests, and unit tests written using Version 1 of the MSTest framework **can** be associated with a test case.
- Tests that use MSTest v2, NUnit, and xUnit frameworks **can** be associated with a test case workitem when using Visual Studio 15.9 Preview 2 or later.
- Tests that use the .NET core framework **can** be associated with a test case workitem when using Visual Studio 15.9 Preview 2 or later. To run the .NET core tests the appropriate target framework must be specified in a [runsettings file](#).
- Tests that use other test frameworks such as Chutzpah (for JavaScript tests such as Mocha or QUnit), or Jest **cannot** be associated with a test case.
- Associating generic tests **may** work, but running these tests is not supported.

Q: Can I configure work items to open in Visual Studio?

A: Yes, if you want test work items to open inside Visual Studio instead of the default Azure Pipelines or TFS UI in

your web browser, change the **Work Items | General** setting from the **Tools | Options** menu in Visual Studio.



Running automated tests from Azure Test Plans

[Go to related topic >](#)

Q: What permissions do I need to run automated tests from Azure Test Plans?

You must be a Project Contributor, or have the following permissions:

- Create releases
- Manage releases
- Edit release stage
- Manage deployment

For more information, see [Set permissions for release pipelines](#) and [Release permissions](#).

Q: Can I override the build or stage set at the test plan level for a specific instance of test run?

A: Yes, you can do this using the **Run with options** command. Open the shortcut menu for the test suite in the left column and choose **Run with options**.

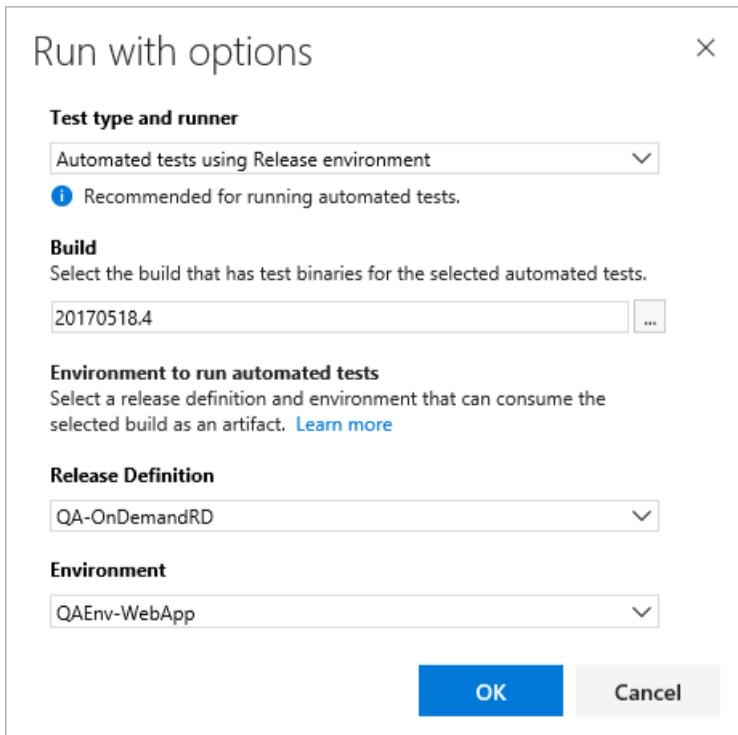
The screenshot shows the 'Test suite: Functional Tests (Suite ID: 3223)' interface. In the top navigation bar, 'Tests' is selected. The toolbar includes 'New', 'Add existing', 'Run', and other execution controls. A context menu is open over the second test row, with the 'Run with options' option highlighted by a red box. The table below lists three test cases: 'ValidateHomePage' (ID 3216), 'ValidateCar_PhantomJS' (ID 3217), and 'ValidateSearch_PhantomJS' (ID 3218). All three tests are marked as 'Automated'.

Outcome ↑	Order	ID	Title	Automation status
Active	...	3216	ValidateHomePage	Automated
Active	...	3217	ValidateCar_PhantomJS	Automated
Active	...	3218	ValidateSearch_PhantomJS	Automated

Enter the following values in the Run with options dialog and then choose **OK**:

- **Test type and runner:** Select **Automated tests using Release Stage**.

- **Build:** Select the build that has the test binaries. The test results will be associated this build.
- **Release Pipeline:** Select a pipeline from the list of release pipelines that can consume the selected build artifact.
- **Release Stage:** Select the name of the stage configured in your release pipeline.



Q: Why use release stages to run tests?

A: Azure Pipelines offers a compelling orchestration workflow to obtain test binaries as artifacts and run tests. This workflow shares the same concepts used in the scheduled testing workflow, meaning users running tests in scheduled workflow will find it easy to adapt; for example, by cloning an existing scheduled testing release pipeline.

Another major benefit is the availability of a rich set of tasks in the task catalog that enable a range of activates to be performed before and after running tests. Examples include preparing and cleaning test data, creating and cleaning configuration files, and more.

Q: How does selecting "Test run" in the Visual Studio Test task version 2 work?

A: The Test management sub-system uses the test run object to pass the list of tests selected for execution. The test task looks up the test run identifier, extracts the test execution information such as the container and test method names, runs the tests, updates the test run results, and sets the test points associated with the test results in the test run. From an auditing perspective, the Visual Studio task provides a trace from the historical releases and the test run identifiers to the tests that were submitted for on-demand test execution.

Q: Should the agent run in interactive mode or as a service?

A: If you are running UI tests such as [coded UI](#) or [Selenium](#) tests, the agent on the test machines must be running in interactive mode with auto-logon enabled, not as a service, to allow the agent to launch a web browser. If you are using a headless browser such as [PhantomJS](#), the agent can be run as a service or in interactive mode. See [Build and release agents](#), [Deploy an agent on Windows](#), and [Agent pools](#).

Q: Where can I find detailed documentation on how to run Selenium tests?

A: See [Get started with Selenium testing](#).

Q: What happens if I select multiple configurations for the same test?

A: Currently, the on-demand workflow is not configuration-aware. In future releases, we plan to pass configuration context to the test method and report the appropriate results.

Q: What if I need to download product binaries and test binaries from different builds? Or if I need to obtain artifacts from a source such as Jenkins?

A: The current capability is optimized for a single team build to be tested on-demand using an Azure Pipelines workflow. We will evaluate support for multi-artifact releases, including non-Azure Pipelines artifacts such as Jenkins, based on user feedback.

Q: I already have a scheduled testing release pipeline. Can I reuse the same pipeline to run test on-demand, or should I create a new pipeline as shown above?

A: We recommend you use a separate release pipeline and stage for on-demand automated testing from Azure Test Plans because:

- You may not want to deploy the app every time you want to run a few on-demand tests. Scheduled testing stages are typically set up to deploy the product and then run tests.
- New releases are triggered for every on-demand run. If you have many testers executing a few on-demand test runs every day, your scheduled testing release pipeline could be overloaded with releases for these runs, making it difficult to find releases that were triggered for the pipeline that contains scheduled testing and deployment to production.
- You may want to configure the Visual Studio Test task with a Test run identifier as an input so that you can trace what triggered the release. See [How does selecting "Test run \(for on-demand runs\)" in the Visual Studio Test task work?](#)

Q: Can I trigger these runs and view the results in Microsoft Test Manager?

A: No. Microsoft Test Manager will not support running automated tests against Team Foundation builds. It only works in the web-based interface for Azure Pipelines and TFS. All new manual and automated testing product development investments will be in the web-based interface. No further development is planned for Microsoft Test Manager. See [Guidance on Microsoft Test Manager usage](#).

Q: I have multiple testers in my team. Can they run tests from different test suites or test plans in parallel using the same release pipeline?

A: They can use the same release pipeline to trigger multiple test runs in parallel if:

- The agent pool associated with the stage has sufficient agents to cater for parallel requests. If sufficient agents are not available, runs can still be triggered but releases will be queued for processing until agents are available.
- You have sufficient jobs to enable parallel jobs. See [Parallel jobs in Azure Pipelines](#) or [Parallel jobs in TFS](#) for more information.
- Testers do not run the same tests in parallel. Doing so may cause results to be overwritten depending on the order of execution.

To enable multiple different test runs to execute in parallel, set the Azure Pipelines stage trigger option for [behavior when multiple releases are waiting to be deployed](#) as follows:

- If your application supports tests running in parallel from different sources, set this option to **Allow multiple releases to be deployed at the same time**.
- If your application does not support tests running in parallel from different sources, set this option to **Allow only one active deployment at a time**.

Q: What are the typical error scenarios or issues I should look out for if my tests don't run?

A: Check and resolve issues as follows:

- The release pipeline and stage in which I want to run tests are not shown after I select the build.
 - Make sure the build pipeline that is generating the build is linked as the primary artifact in the **Artifacts**

tab of the release pipeline.

- I get an error that I don't have sufficient permission to trigger a release.
 - Configure **Create releases** and **Manage deployments** permissions for the user in the **Security** menu of the release pipeline. See [Release permissions](#).
- I get an error that no automated tests were found.
 - Check the automation status of the selected tests. Do this in the work item for the test case, or use the **Column options** link in the **Test Plans** page of **Azure Pipelines** section in Azure DevOps or the **Build & Release** hub in TFS to add the **Automation status** column to the list of tests. See the [pre-requisites section](#) for information about automating manual tests.
- My tests didn't execute, and I suspect the release pipeline is incorrect.
 - Use the link in the **Run summary** page to access the release instance used to run the tests, and view the release logs.
- My tests go into the error state, or remain "in-progress" even after release to the stage is triggered.
 - Check if the release stage that you selected has the correct task and version selected. You must use version 2 or higher of the **Visual Studio Test** task. Version 1 of the task, and the **Run Functional Tests** task, are not supported.

Restore Package Management NuGet packages in Azure Pipelines

11/6/2018 • 4 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

This walkthrough will cover setting up an existing build to restore NuGet packages from Package Management feeds. It assumes that you've already:

- [Set up your solution](#) to consume packages from a Package Management feed
- [Created a build](#) for that solution
- [Added the correct build service identity](#) to your feed

To build a solution that relies on NuGet packages from Package Management feeds, add the **NuGet** task (if one is not already present).

First, click **Add build tasks...**, select the **Package** category, and add the **NuGet** task. Then drag to order the task above any build tasks that require your packages.

Next, configure these options:

- **Command:** restore
- **Path to solution, packages.config, or project.json:** The path to the file that specifies the packages you want to restore

Then, select feeds to use:

- If you've checked in a [NuGet.config](#), select **Feeds in my NuGet.config** and select the file from your repo.
- If you're using a single Azure Artifacts/TFS feed, select the **Feed(s) I select here** option and select your feed from the dropdown.

NuGet

Version 2.*

Display name *

NuGet restore

Command *

restore

Path to solution, packages.config, or project.json *

***.sln

Feeds and authentication

Feeds to use *

Feed(s) I select here (selected) Feeds in my NuGet.config

Use packages from this VSTS/TFS feed

FabrikamFiber

Use packages from NuGet.org

Finally, save your build.

Specifying sources in NuGet.config

The NuGet.config you check in should list all the package sources you want to consume. The example below demonstrates how that might look.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <packageSources>
    <!-- remove any machine-wide sources with <clear/> -->
    <clear />
    <!-- add an Azure Artifacts feed -->
    <add key="MyGreatFeed"
      value="https://fabrikam.pkgs.visualstudio.com/DefaultCollection/_packaging/MyGreatFeed/nuget/v3/index.json" />
    <!-- also get packages from the NuGet Gallery -->
    <add key="nuget.org" value="https://www.nuget.org/api/v2/" />
  </packageSources>
  <activePackageSource>
    <add key="All" value="(Aggregate source)" />
  </activePackageSource>
</configuration>
```

Restoring packages from feeds in a different organization

If your NuGet.config contains feeds in a different Azure DevOps organization (dev.azure.com/organization) than the organization running the build, you'll need to set up credentials for those feeds manually.

1. Select a login account (either a service account (recommended) or a user's account) that has access to the remote feed
2. Using your browser's InPrivate mode, Incognito mode, or similar, go to the Azure DevOps organization that contains the feed, sign in with the login account you selected in step 1, click the user profile circle in the top right, and select Security
3. Create a PAT with the **Packaging (read)** scope and keep it handy
4. In the Azure DevOps organization that contains the build, edit the build's NuGet step and ensure you're using version 2 or greater of the task, using the version selector

5. In the **Feeds and authentication** section, Ensure you've selected the **Feeds in my NuGet.config** radio button
6. Set the path to your NuGet.config in the **Path to NuGet.config**
7. In **Credentials for feeds outside this organization/collection**, click the +
8. In the service connection dialog that appears, enter the feed URL (make sure it matches what's in your NuGet.config) and the PAT you created in step 3
9. Save the service connection and the build, then queue a new build

Q & A

Why can't my build restore packages?

NuGet restore can fail due to a variety of issues. One of the most common issues is the introduction of a new project in your solution that requires a [target framework](#) that isn't understood by the version of NuGet your build is using. This issue generally doesn't present itself on a developer machine because Visual Studio updates the NuGet restore mechanism at the same time it adds new project types. We're looking into similar features for Azure Artifacts. In the meantime though, the first thing to try when you can't restore packages is to update to the latest version of NuGet.

How do I use the latest version of NuGet?

If you're using Azure Pipelines or TFS 2018, new template-based builds will work automatically thanks to a new "NuGet Tool Installer" task that's been added to the beginning of all build templates that use the NuGet task. We periodically update the default version that's selected for new builds around the same time we install Visual Studio updates on the Hosted build agents.

For existing builds, just add or update a NuGet Tool Installer task to select the version of NuGet for all the subsequent tasks. You can see all available versions of NuGet [on nuget.org](#).

The screenshot shows the Azure DevOps interface for a project named 'FabrikamFiber'. Under the 'Build and Release' tab, the 'FabrikamFiber-CI' pipeline is selected. The 'Tasks' tab is active, displaying a list of build steps. A specific task, 'NuGet Tool Installer', is highlighted. Its configuration pane is open, showing the following details:

- Version:** 0.*
- Display name:** Use NuGet 4.3.0
- Version of NuGet.exe to install:** 4.3.0
- Control Options:** Includes a checkbox for 'Always download the latest matching version'.

Other tasks listed in the pipeline include 'Get sources' and 'NuGet restore'.

TFS 2017 and earlier

Because the NuGet Tool Installer is not available in TFS versions prior to TFS 2018, there is a recommended workaround to use versions of NuGet > 4.0.0 in Azure Pipelines.

1. Add the task, if you haven't already. If you have a "NuGet Restore" task in the catalog (it may be in the Deprecated tasks section), insert it into your build. Otherwise, insert a "NuGet" task.
2. For your NuGet/NuGet Installer task, use the version selector under the task name to select version "0.*".

3. In the Advanced section, set the NuGet Version to "Custom" and the Path to NuGet.exe as
\$(Build.BinariesDirectory)\nuget.exe
4. Before your NuGet task, add a "PowerShell" task, select "Inline Script" as the Type, enter this PowerShell script as the Inline Script, and enter "4.3.0" (or any version of NuGet from this list) as the Arguments.

Our thanks to [GitHub user leftler](#) for creating the original version of the PowerShell script linked above.

Use Jenkins to restore and publish packages

10/9/2018 • 4 minutes to read • [Edit Online](#)

[Azure Artifacts](#) | [TFS 2018](#) | [TFS 2017](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

Azure Artifacts works with the continuous integration tools your team already uses. In this [Jenkins](#) walkthrough, you'll create a NuGet package and publish it to an Azure Artifacts feed. If you need help on Jenkins setup, you can learn more on [the Jenkins wiki](#).

Setup

This walkthrough uses Jenkins 1.635 running on Windows 10. The walkthrough is simple, so any recent Jenkins and Windows versions should work.

Ensure the following Jenkins plugins are enabled:

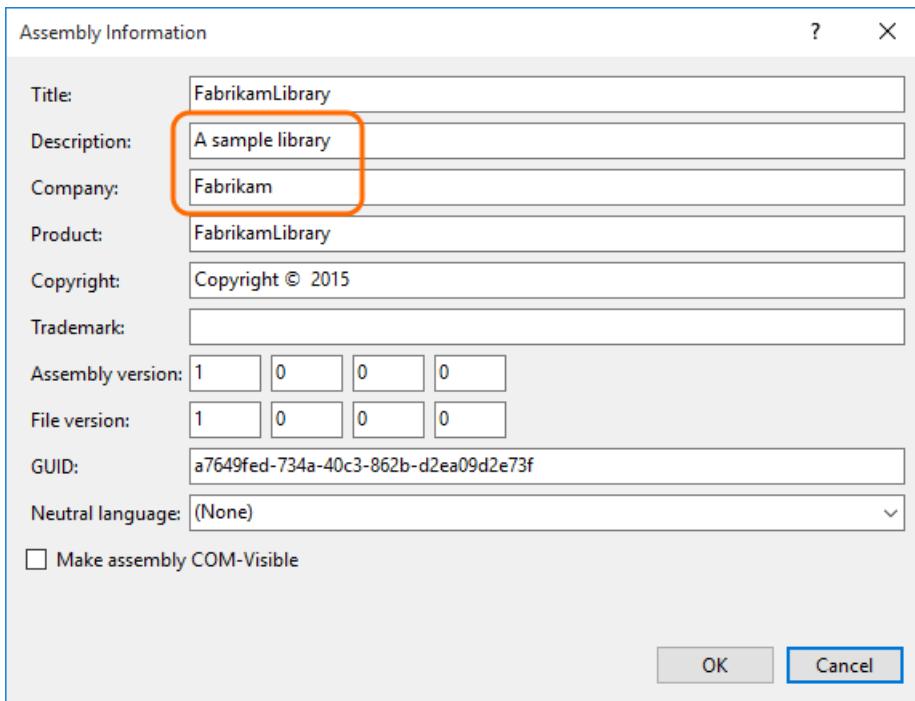
- [MSBuild 1.24](#)
- [Git 2.4.0](#)
- [Git Client 1.19.0](#)
- [Credentials Binding plugin 1.6](#)

Some of these plugins are enabled by default. Others you will need to install by using Jenkins's "Manage Plugins" feature.

The example project

The sample project is a simple shared library written in C#.

- To follow along with this walkthrough, create a new C# Class Library solution in Visual Studio 2015.
- Name the solution "FabrikamLibrary" and uncheck the **Create directory for solution** checkbox.
- On the FabrikamLibrary project's context menu, choose **Properties**, then choose **Assembly Information**.
- Edit the description and company fields. Now generating a NuGet package is easier.



- Check the new solution into a Git repo where your Jenkins server can access it later.

Add the Azure Artifacts NuGet tools to your repo

The easiest way to use the Azure Artifacts NuGet service is by adding the [Microsoft.VisualStudio.Services.NuGet.Bootstrap package](#) to your project.

Create a package from your project

Whenever you work from a command line, run `init.cmd` first. This sets up your environment to allow you to work with `nuget.exe` and the Azure Artifacts NuGet service.

- Change into the directory containing `FabrikamLibrary.csproj`.
- Run the command `nuget spec` to create the file `FabrikamLibrary.nuspec`, which defines how your NuGet package builds.
- Edit `FabrikamLibrary.nuspec` to remove the boilerplate tags `<licenseUrl>`, `<projectUrl>`, and `<iconUrl>`. Change the tags from `Tag1 Tag2` to `fabrikam`.
- Ensure that you can build the package using the command `nuget pack FabrikamLibrary.csproj`. Note, you should target the `.csproj` (project) file, not the NuSpec file.
- A file called `FabrikamLibrary.1.0.0.0.nupkg` will be produced.

Set up a feed in Azure Artifacts and add it to your project

- [Create a feed](#) in your Azure DevOps organization called `MyGreatFeed`. Since you're the owner of the feed, you will automatically be able to push packages to it.
- Add the URL for the feed you just generated to the `nuget.config` in the root of your repo.
 - Find the `<packageSources>` section of `nuget.config`.
 - Just before `</packageSources>`, add a line using this template:
`<add key="MyGreatFeed" value="{feed_url}" />`. Change `{feed_url}` to the URL of your feed.
 - Commit this change to your repo.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <configuration>
3      <packageSources>
4          <clear />
5          <add key="vss-package-management" value=
6              "https://www.myget.org/F/vss-package-management/api/v2" />
7          <add key="MyGreatFeed" value=
8              "https://fabrikam.pkgs.visualstudio.com/DefaultCollection/_apis/packaging/MyGr
9                  eatFeed/nuget/index.json" />
10     </packageSources>
11     <activePackageSource>
12         <add key="All" value="(Aggregate source)" />
13     </activePackageSource>
14 </configuration>

```

- Generate a [PAT \(personal access token\)](#) for your user account. This PAT will allow the Jenkins job to authenticate to Azure Artifacts as you, so be sure to protect your PAT like a password.
- Save your feed URL and PAT to a text file for use later in the walkthrough.

Create a build pipeline in Jenkins

- Ensure you have the [correct plugins installed in Jenkins](#).
- This will be a Freestyle project. Call it "Fabrikam.Walkthrough".

Item name

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

External Job
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system. See [the documentation for more details](#).

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Copy existing Item

OK

- Under Source Code Management, set the build to use **Git** and select your Git repo.
- Under Build Environment, select the **Use secret text(s) or file(s)** option.
 - Add a new **Username and password (separated)** binding.
 - Set the **Username Variable** to "FEEDUSER" and the **Password Variable** to "FEEDPASS". These are the environment variables Jenkins will fill in with your credentials when the build runs.

- Choose the **Add** button to create a new username and password credential in Jenkins.
- Set the **username** to "token" and the **password** to the PAT you generated earlier. Choose **Add** to save these credentials.

Add Credentials

Kind: Username with password

Scope: Global (Jenkins, nodes, items, all child items, etc)

Username: token

Password: (redacted)

Description: feedpat

Advanced...

Add Cancel

Build Environment

Use secret text(s) or file(s) ?

Bindings

Username and password (separated) ?

Username Variable: FEEDUSER ?

Password Variable: FEEDPASS ?

Credentials: Specific credentials Parameter expression ?

token/***** (feedpat) Add

Delete

- Under Build (see screenshot below), follow these steps:
 - Choose **Execute Windows batch command**. In the **Command** box, type `init.cmd`.
 - Choose **Build a Visual Studio project or solution using MSBuild**. This task should point to `msbuild.exe` and `FabrikamLibrary.sln`.
 - Choose **Execute Windows batch command** again, but this time, use this command:
`.tools\VSS.NuGet\nuget pack FabrikamLibrary\FabrikamLibrary.csproj`.

Build

1 Execute Windows batch command
Command: init.cmd
See the list of available environment variables Delete

2 Build a Visual Studio project or solution using MSBuild
MSBuild Version: msbuild.exe
MSBuild Build File: FabrikamLibrary.sln
Command Line Arguments: Delete

3 Pass build variables as properties Advanced... Delete

Execute Windows batch command
Command: .tools\VSS.NuGet\nuget pack FabrikamLibrary\FabrikamLibrary.csproj
See the list of available environment variables Delete

- Save this build pipeline and queue a build.
- The build's Workspace will now contain a .nupkg just like the one you built locally earlier.

Publish a package using Jenkins

These are the last walkthrough steps to publish the package to a feed:

- Edit the build pipeline in Jenkins.
- After the last build task (which runs `nuget pack`), add a new **Execute a Windows batch command** build task.
- In the new **Command** box, add these two lines:
 - The first line puts credentials where NuGet can find them:

```
.tools\VSS.NuGet\nuget sources update -Name "MyGreatFeed" -UserName "%FEEDUSER%" -Password "%FEEDPASS%"
```
 - The second line pushes your package using the credentials saved above:

```
.tools\VSS.NuGet\nuget push *.nupkg -Name "MyGreatFeed" -ApiKey VSS
```

Execute Windows batch command



Command

```
.tools\VSS.NuGet\nget sources update -Name "MyGreatFeed" -UserName "%FEEDUSER%"  
-Password "%FEEDPASS%"  
.tools\VSS.NuGet\nget push *.nupkg -Name "MyGreatFeed" -ApiKey VSS
```

See [the list of available environment variables](#)

[Delete](#)

- Queue another build. This time, the build machine will authenticate to Azure Artifacts and push the package to the feed you selected.

Migrate from XAML builds to new builds

11/19/2018 • 13 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [XAML builds](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

We introduced XAML build automation capabilities based on the Windows Workflow Foundation in Team Foundation Server (TFS) 2010. We released another version of [XAML builds](#) in TFS 2013.

After that we sought to expand beyond .NET and Windows and add support for other kinds of apps that are based on operating systems such as macOS and Linux. It became clear that we needed to switch to a more open, flexible, web-based foundation for our build automation engine. In early 2015 in Azure Pipelines, and then in TFS 2015, we introduced a simpler task- and script-driven cross-platform build system.

Because the systems are so different, there's no automated or general way to migrate a XAML build pipeline into a new build pipeline. The migration process is to manually create the new build pipelines that replicate what your XAML builds do.

If you're building standard .NET applications, you probably used our default templates as provided out-of-the-box. In this case the process should be reasonably easy.

If you have customized your XAML templates or added custom tasks, then you'll need to also take other steps including writing scripts, installing extensions, or creating custom tasks.

Overview of the migration effort

Here are the steps to migrate from XAML builds to newer builds:

1. If you're using a private TFS server, [set up agents](#) to run your builds.
2. To get familiar with the new build system, create a "[Hello world](#)" build pipeline.
3. Create a new build pipeline intended to replace one of your XAML build pipelines.
 - a. Create a new build pipeline.
 - b. Port your XAML settings.
4. On the [General tab](#), disable the XAML build pipeline.
5. Repeat the previous two steps for each of your XAML build pipelines.
6. Take advantage of new build features and learn more about the kinds of apps you can build.
7. Learn how to customize, and if necessary extend your system.
8. When you no longer need the history and artifacts from your XAML builds, delete the XAML builds, and then the XAML build pipelines.

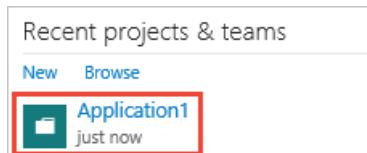
WARNING

After you delete the XAML builds and pipelines, you cannot get them back.

Create new build pipelines

If you're building a standard .NET app, you're probably using one of the out-of-the-box build templates such as TfvcTemplate.12.xaml or GitTemplate.12.xaml. In this case, it will probably just take you a few clicks to create build pipelines in the new build system.

1. Open your project in your web browser ▼

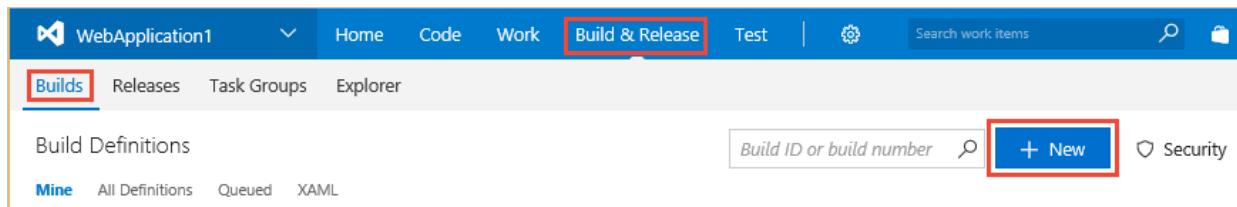


(If you don't see your project listed on the home page, select **Browse**.)

- On-premises TFS: `http://{your_server}:8080/tfs/DefaultCollection/{your_project}`
- Azure Pipelines: `https://dev.azure.com/{your_organization}/{your_project}`

The TFS URL doesn't work for me. How can I get the correct URL?

2. Create a build pipeline (Pipelines tab > Builds) ▼



3. Select a template to add commonly used tasks to your build pipeline.
4. Make any necessary changes to your build pipeline to replicate your XAML build pipeline. The tasks added by the template should simply work in many cases. But if you changed process parameters or other settings in your XAML build pipelines, below are some pointers to get you started replicating those changes.

Port your XAML settings

In each of the following sections we show the XAML user interface, and then provide a pointer to the place where you can port the setting into your new build pipeline.

General tab

Build definition name:
OurBuild

Description (optional):

Queue processing:

- Enabled**
Requests queued by users or triggered by the system will be added to the queue and be started in priority order.
- Paused**
Requests queued by users or triggered by the system will be added to the queue but will not start unless the build administrator forces them to start.
- Disabled**
No requests will be queued or started. This definition will also not participate in triggered builds like Continuous Integration or Gated.

XAML SETTING	TFS 2017 EQUIVALENT	AZURE PIPELINES AND TFS 2018 AND NEWER EQUIVALENT
Build pipeline name	You can change it whenever you save the pipeline.	When editing the pipeline: On the Tasks tab, in left pane click Pipeline , and the Name field appears in right pane. In the Builds hub (Mine or All pipelines tab), open the action menu and choose Rename .
Description (optional)	Not supported.	Not supported.
Queue processing	Not yet supported. As a partial alternative, disable the triggers.	Not yet supported. As an alternative, disable the triggers.

Source Settings tab

TFVC

Working folders:		
Status	Source Control Folder	Build Agent Folder
Active	\$/OurTFVCProject	\$(SourceDir)
Cloaked	\$/OurTFVCProject/Drops	
	Click here to enter a new working folder	

XAML SETTING	TFS 2017 AND NEWER EQUIVALENT	AZURE PIPELINES EQUIVALENT
Source Settings tab	On the Repository tab specify your mappings with Active paths as Map and Cloaked paths as Cloak .	On the Tasks tab, in left pane click Get sources . Specify your workspace mappings with Active paths as Map and Cloaked paths as Cloak .

The new build pipeline offers you some new options. The specific extra options you'll see depend on the version you're using of TFS or Azure Pipelines. If you're using Azure Pipelines, first make sure to display **Advanced settings**. See [Build TFVC repositories](#).

Git

XAML build* ➔ X

General	Choose the source of the build. You can choose a Team Foundation service.				
Trigger	<input checked="" type="checkbox"/> Get sources from a Team Foundation Git repository				
Source Settings	Repository name: TestGit Default branch for Manual and Scheduled builds: refs/heads/master Monitored branches for Continuous Integration and Rolling builds: <table border="1"><tr><td>Include/Exclude</td><td>Branch</td></tr><tr><td>Include</td><td>refs/heads/master</td></tr></table> Click here to add a new row	Include/Exclude	Branch	Include	refs/heads/master
Include/Exclude	Branch				
Include	refs/heads/master				
Build Defaults					
Process					
Retention Policy					

XAML SETTING	TFS 2017 AND NEWER EQUIVALENT	AZURE PIPELINES EQUIVALENT
Source Settings tab	On the Repository tab specify the repository and default branch.	On the Tasks tab, in left pane click Get sources . Specify the repository and default branch.

The new build pipeline offers you some new options. The specific extra options you'll see depend on the version you're using of TFS or Azure Pipelines. If you're using Azure Pipelines, first make sure to display **Advanced settings**. See [Pipeline options for Git repositories](#).

Trigger tab

XAMLBuild ➔ X Build OurBuild_20170516.3 ➔ X

General	Select one of the following triggers:
Trigger	<input type="radio"/> Manual - Check-ins do not trigger a new build <input checked="" type="radio"/> Continuous Integration - Build each check-in <input type="radio"/> Rolling builds - accumulate check-ins until the prior build finishes <input type="checkbox"/> Build no more often than <u>every</u> <input type="text"/> minutes. <input type="radio"/> Gated Check-in - accept check-ins only if the submitted changes merge and build successfully <input type="checkbox"/> Merge and build up to <input type="text"/> submissions. <input type="radio"/> Schedule - build <u>every week</u> on the following days <input checked="" type="checkbox"/> Monday <input checked="" type="checkbox"/> Tuesday <input checked="" type="checkbox"/> Wednesday <input checked="" type="checkbox"/> Thursday <input checked="" type="checkbox"/> Friday <input type="checkbox"/> Saturday <input type="checkbox"/> Sunday Queue the build on the build controller at: <input type="text"/> 03:00 Eastern Daylight Time (UTC -04:00) <input type="checkbox"/> Build <u>even</u> if nothing has changed since the previous build
Source Settings	
Build Defaults	
Process	
Retention Policy	

XAML SETTING	TFS 2017 AND NEWER, AZURE PIPELINES EQUIVALENT
Trigger tab	On the Triggers tab, select the trigger you want to use: CI, scheduled, or gated.

The new build pipeline offers you some new options. For example:

- You can potentially create fewer build pipelines to replace a larger number of XAML build pipelines. This is because you can use a single new build pipeline with multiple triggers. And if you're using Azure Pipelines, then you can add multiple scheduled times.
- The **Rolling builds** option is replaced by the **Batch changes** option. You can't specify minimum time between builds. But if you're using Azure Pipelines, you can specify the maximum number of parallel jobs

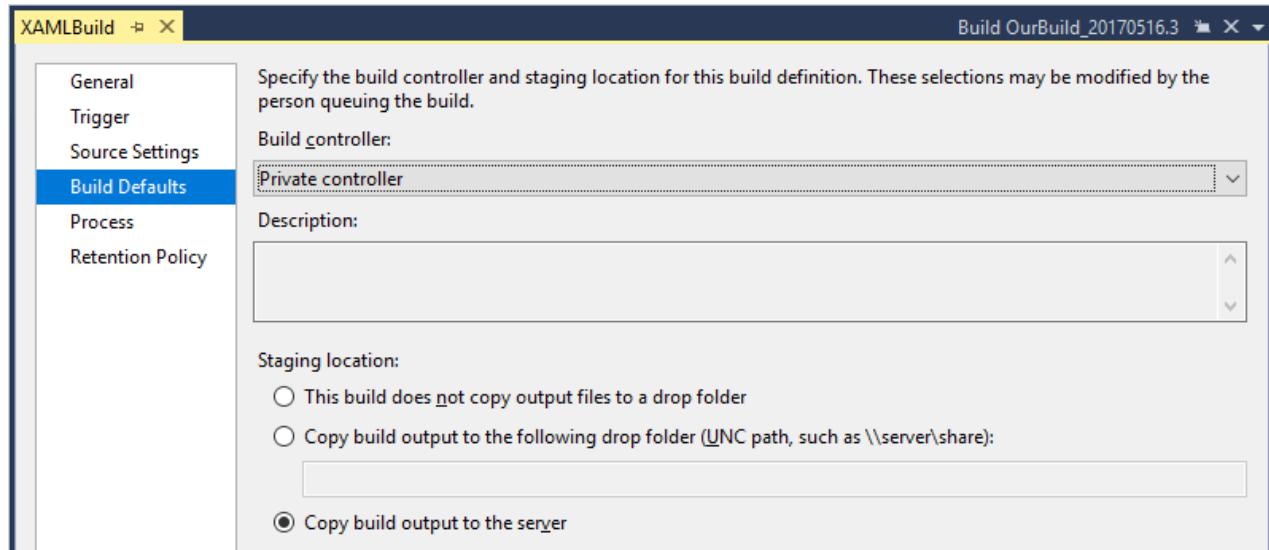
per branch.

- If your code is in TFVC, you can add folder path filters to include or exclude certain sets of files from triggering a CI build.
- If your code is in TFVC and you're using the gated check-in trigger, you've got the option to also run CI builds or not. You can also use the same workspace mappings as your repository settings, or specify different mappings.
- If your code is in Git, then you specify the branch filters directly on the **Triggers** tab. And you can add folder path filters to include or exclude certain sets of files from triggering a CI build.

The specific extra options you'll see depend on the version you're using of TFS or Azure Pipelines. See [Build pipeline triggers](#)

We don't yet support the **Build even if nothing has changed since the previous build** option.

Build Defaults tab



XAML PROCESS PARAMETER	TFS 2017 AND NEWER EQUIVALENT	AZURE PIPELINES EQUIVALENT
Build controller	On the General tab, select the default agent pool.	On the Options tab, select the default agent pool.
Staging location	On the Tasks tab, specify arguments to the Copy Files and Publish Build Artifacts tasks. See Build artifacts .	On the Tasks tab, specify arguments to the Copy Files and Publish Build Artifacts tasks. See Build artifacts .

The new build pipeline offers you some new options. For example:

- You don't need a controller, and the new agents are easier to set up and maintain. See [Build and release agents](#).
- You can exactly specify which sets of files you want to publish as build artifacts. See [Build artifacts](#).

Process tab

TF Version Control

Build process parameters:	
1. TF Version Control	
1. Clean workspace	True
2. Get version	
3. Label Sources	True
2. Build	
3. Test	
4. Publish Symbols	
5. Advanced	

XAML PROCESS PARAMETER	TFS 2017 AND NEWER EQUIVALENT	AZURE PIPELINES EQUIVALENT
Clean workspace	On the Repository tab, open the Clean menu, and then select true .	On the Tasks tab, in left pane click Get sources . Display Advanced settings , and then select Clean . (We plan to change move this option out of advanced settings.)
Get version	You can't specify a changeset in the build pipeline, but you can specify one when you manually queue a build.	You can't specify a changeset in the build pipeline, but you can specify one when you manually queue a build.
Label Sources	On the Repository tab, select an option from the Label sources menu.	Tasks tab, in left pane click Get sources . Select one of the Tag sources options. (We plan to change the name of this to Label sources .)

The new build pipeline offers you some new options. See [Build TFVC repositories](#).

Git

Build process parameters:	
1. Git	
1. Clean repository	True
2. Checkout override	
2. Build	
3. Test	
4. Publish Symbols	
5. Advanced	

XAML PROCESS PARAMETER	TFS 2017 AND NEWER EQUIVALENT	AZURE PIPELINES EQUIVALENT
Clean repository	Repository tab, open Clean menu, select true .	On the Tasks tab, in left pane click Get sources . Show Advanced settings , and then select Clean . (We plan to change move this option out of advanced settings.)
Checkout override	You can't specify a commit in the build pipeline, but you can specify one when you manually queue a build.	You can't specify a commit in the build pipeline, but you can specify one when you manually queue a build.

The new build pipeline offers you some new options. See [Pipeline options for Git repositories](#).

Build

Build process parameters:	
> 1. TF Version Control	
▽ 2. Build	
1. Projects	\$/TestTFVC/ConsoleApplication2/ConsoleApplication2.sln
2. Configurations	
3. Clean build	True
4. Output location	SingleFolder
▽ 5. Advanced	
MSBuild arguments	
MSBuild platform	Auto
Perform code analysis	AsConfigured
Post-build script arguments	
Post-build script path	
Pre-build script arguments	
Pre-build script path	
> 3. Test	
> 4. Publish Symbols	
> 5. Advanced	

On the **Build** tab (TFS 2017 and newer) or the **Tasks** tab (Azure Pipelines), after you select the Visual Studio Build task, you'll see the arguments that are equivalent to the XAML build parameters.

XAML PROCESS PARAMETER	TFS 2017 AND NEWER, AZURE PIPELINES EQUIVALENT ARGUMENT
Projects	Solution
Configurations	Platform, Configuration. See Visual Studio Build: How do I build multiple configurations for multiple platforms?
Clean build	Clean
Output location	The Visual Studio Build task builds and outputs files in the same way you do it on your dev machine, in the local workspace. We give you full control of publishing artifacts out of the local workspace on the agent. See Artifacts in Azure Pipelines .
Advanced, MSBuild arguments	MSBuild Arguments
Advanced, MSBuild platform	Advanced, MSBuild Architecture
Advanced, Perform code analysis	Use an MSBuild argument such as <code>/p:RunCodeAnalysis=true</code>
Advanced, post- and pre-build scripts	You can run one or more scripts at any point in your build pipeline by adding one or more instances of the PowerShell, Batch, and Command tasks. For example, see Use a PowerShell script to customize your build pipeline .

IMPORTANT

In the Visual Studio Build arguments, on the **Visual Studio Version** menu, make sure to select version of Visual Studio that you're using.

The new build pipeline offers you some new options. See [Visual Studio Build](#).

Learn more: [Visual Studio Build task](#) (for building solutions), [MSBuild task](#) (for building individual projects).

Test

Build process parameters:	
> 1. TF Version Control	
> 2. Build	
▽ 3. Test	
▽ 1. Automated tests	1 set(s) of tests specified. - Run tests in test sources matching ***test*.dll;***test*.appx, Target platform X86 Fail build on test failure Run settings Run settings file Type of run settings Target platform for test execution Test case filter Test run name Test sources spec ***test*.dll;***test*.appx
▽ 2. Advanced	Analyze test impact Disable tests Post-test script arguments Post-test script path Pre-test script arguments Pre-test script path
> 4. Publish Symbols	
> 5. Advanced	

See [Get started with continuous testing](#) and [Visual Studio Test task](#).

Publish Symbols

Build process parameters:	
> 1. TF Version Control	
> 2. Build	
> 3. Test	
▽ 4. Publish Symbols	Path to publish symbols
> 5. Advanced	

XAML PROCESS PARAMETER	TFS 2017 AND NEWER, AZURE PIPELINES EQUIVALENT
Path to publish symbols	Click the Publish Symbols task and then copy the path into the Path to publish symbols argument.

Advanced

Build process parameters:	
> 1. TF Version Control	
> 2. Build	
> 3. Test	
> 4. Publish Symbols	
▽ 5. Advanced	
▽ Agent settings	Use agent where Name=* and Tags=[] (MatchExactly) 00:00:00 Maximum agent execution time 04:00:00 Maximum agent reservation wait time Name filter * Tag comparison operator MatchExactly Tags filter
Build number format	\$(BuildDefinitionName)_\$(Date:yyyyMMdd)\$(Rev:.r)
Create work item on failure	True
Update work items with build number	True

XAML PROCESS PARAMETER	TFS 2017 AND NEWER EQUIVALENT	AZURE PIPELINES EQUIVALENT

XAML PROCESS PARAMETER	TFS 2017 AND NEWER EQUIVALENT	AZURE PIPELINES EQUIVALENT
Maximum agent execution time	None	On the Options tab you can specify Build job timeout in minutes .
Maximum agent reservation wait time	None	None
Name filter, Tag comparison operator, Tags filter	A build pipeline asserts demands that are matched with agent capabilities. See Agent capabilities .	A build pipeline asserts demands that are matched with agent capabilities. See Agent capabilities .
Build number format	On the General tab, copy your build number format into the Build number format field.	On the General tab, copy your build number format into the Build number format field.
Create work item on failure	On the Options tab, select this check box.	On the Options tab, enable this option.
Update work items with build number	None	On the Options tab you can enable Automatically link new work in this build .

The new build pipeline offers you some new options. See:

- [Agent capabilities](#)
- [Build number format](#)

Retention Policy tab

Specify how builds should be retained:			
Build Outcome	Retention Policy	What to Delete	
Triggered and Manual			
Stopped	Keep Latest Only	Details, Drop, Label, Symbols	
Failed	Keep 10 Latest	Details, Drop, Label, Symbols	
Partially Succeeded	Keep 10 Latest	Details, Drop, Label, Symbols	
Succeeded	Keep 10 Latest	Details, Drop, Label, Symbols	
Private			
Stopped	Keep Latest Only	Details, Drop, Label, Symbols	
Failed	Keep 10 Latest	Details, Drop, Label, Symbols	
Partially Succeeded	Keep 10 Latest	Details, Drop, Label, Symbols	
Succeeded	Keep 10 Latest	Details, Drop, Label, Symbols	

XAML PROCESS PARAMETER	TFS 2017 AND NEWER, AZURE PIPELINES EQUIVALENT
Retention Policy tab	On the Retention tab specify the policies you want to implement.

The new build pipeline offers you some new options. See [Build and release retention policies](#).

Build and release different kinds of apps

In XAML builds you had to create your own custom templates to build different types of apps. In the new build system you can pick from a set of pre-defined templates. The largest and most current set of templates are available on Azure Pipelines and in our newest version of TFS.

Build

Here are a few examples of the kinds of apps you can build:

- [Build your ASP.NET 4 app.](#)
- Build your ASP.NET Core app
- [Build your Universal Windows Platform app](#)
- [Build your Xamarin app](#)
- [C++ apps for Windows](#)

Release

The new build system is tightly integrated with Azure Pipelines. So it's easier than ever to automatically kick off a deployment after a successful build. Learn more:

- [CI/CD Hello world](#)
- [Release pipelines](#)
- [Triggers](#)

A few examples include:

- [Continuous deployment of your app to an Azure web site](#)
- [IIS using deployment groups](#)

Other apps and tasks

For more examples of apps you can build and deploy, see [Build and deploy your app](#).

For a complete list of our build, test, and deployment tasks, see [Build and release tasks](#).

Customize your tasks

In XAML builds you created custom XAML tasks. In the new builds, you've got a range of options that begin with easier and lighter-weight approaches.

Get tasks from the Marketplace

[Visual Studio Marketplace](#) offers hundreds of extensions that you can install to add tasks that extend your build and deployment capabilities.

Write a script

A major feature of the new build system is its emphasis on using scripts to customize your build pipeline. You can check your scripts into version control and customize your build using any of these methods:

- [PowerShell scripts \(Windows\)](#)
- [Batch scripts \(Windows\)](#)
- [Command prompt](#)
- [Shell scripts \(macOS and Linux\)](#)

TIP

If you're using TFS 2017 or newer, you can write a short PowerShell script directly inside your build pipeline.

The screenshot shows the TFS build pipeline editor. On the left, there's a sidebar with a green plus icon labeled "Add build step...". Below it are two items: "Build solution \$/TestTFVC/C Visual Studio Build" (with a VSTS icon) and "PowerShell Script PowerShell" (with a blue script icon). The "PowerShell Script" item is selected and highlighted in light blue. On the right, the task configuration pane is open for "PowerShell Script". It has three tabs: "Type" (selected), "Arguments", and "Inline Script". Under "Type", "Inline Script" is chosen. In the "Arguments" section, there is a single line of PowerShell code: "Write-Host \"Hello World from \$Env:AGENT_NAME.\"".

TFS 2017 or newer inline PowerShell script

For all these tasks we offer a set of built-in variables, and if necessary, you can define your own variables. See [Build variables](#).

Write a custom task

If necessary, you can write your own [custom extensions](#) to [custom tasks](#) for your builds and releases.

Reuse patterns

In XAML builds you created custom XAML templates. In the new builds, it's easier to create reusable patterns.

Create a template

If you don't see a template for the kind of app you can start from an empty pipeline and [add the tasks you need](#). After you've got a pattern that you like, you can clone it or save it as a template directly in your web browser. See [CI/CD Hello world](#).

Task groups (TFS 2017 or newer)

In XAML builds, if you change the template, then you also change the behavior of all pipelines based on it. In the new build system, templates don't work this way. Instead, a template behaves as a traditional template. After you create the build pipeline, subsequent changes to the template have no effect on build pipelines.

If you want to create a reusable and automatically updated piece of logic, then [create a task group](#). You can then later modify the task group in one place and cause all the pipelines that use it to automatically be changed.

Q & A

I don't see XAML builds. What do I do?

XAML builds are deprecated. We strongly recommend that you migrate to the new builds as explained above.

If you're not yet ready to migrate, then to enable XAML builds you must connect a XAML build controller to your organization. See [Configure and manage your build system](#).

If you're not yet ready to migrate, then to enable XAML builds:

1. Install [TFS 2018.2](#).
2. Connect your XAML build servers to your TFS instance. See [Configure and manage your build system](#).

How do I add conditional logic to my build pipeline?

Although the new build pipelines are essentially linear, we do give you control of the conditions under which a task runs.

On TFS 2015 and newer: You can select Enabled, Continue on error, or Always run.

On Azure Pipelines, you can specify one of four built-in choices to control when a task is run. If you need more control, you can specify custom conditions. For example:

```
and(failed(), in(variables['Build.Reason'], 'IndividualCI', 'BatchedCI'),  
startsWith(variables['Build.SourceBranch'], 'refs/heads/features/'))
```

See [Specify conditions for running a task](#).

Create a virtual network isolated environment for build-deploy-test scenarios

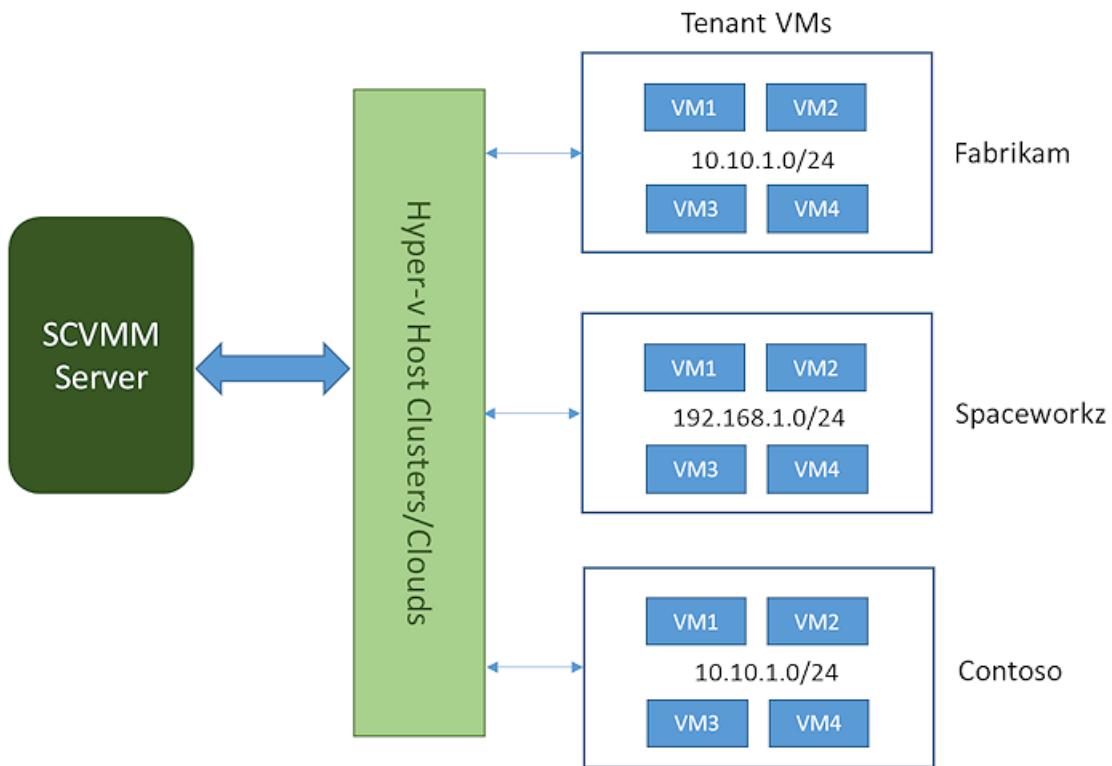
11/15/2018 • 9 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

Network Virtualization provides ability to create multiple virtual networks on a shared physical network. Isolated virtual networks can be created using SCVMM Network Virtualization concepts. VMM uses the concept of logical networks and corresponding VM networks to create isolated networks of virtual machines.



- You can create an isolated network of virtual machines that span across different hosts in a host-cluster or a private cloud.
- You can have VMs from different networks residing in the same host machine and still be isolated from each other.
- You can define IP address from the any IP pool of your choice for a VM Network.

See also: [Hyper-V Network Virtualization Overview](#).

To create a virtual network isolated environment:

- Ensure you meet the prerequisite conditions described in [this section](#).
- Set up Network Virtualization using SCVMM. This is a one-time setup task you do not need to repeat.

Follow [these steps](#).

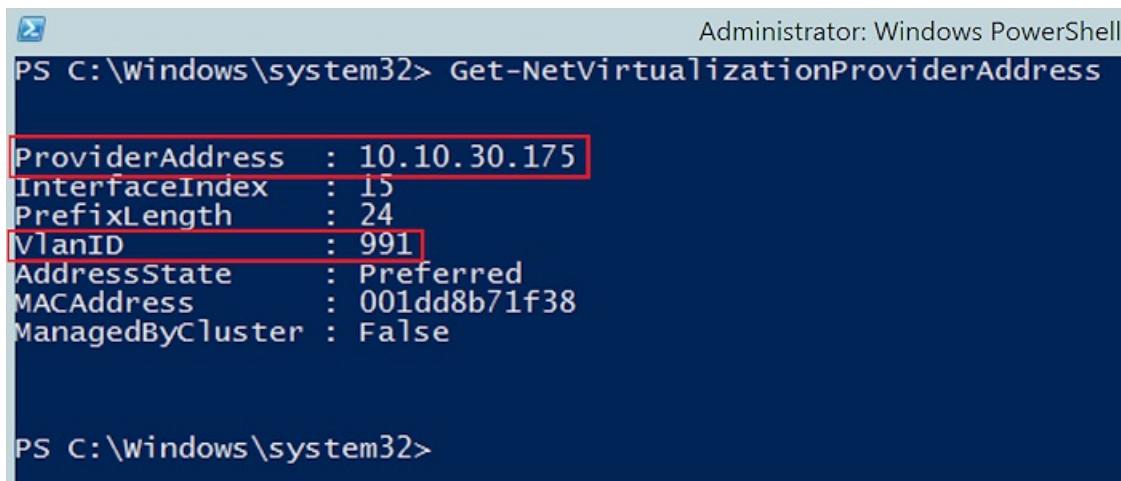
- Decide on the network topology you want to use. You'll specify this when you create the virtual network. The options and steps are described in [this section](#).
- Enable your build-deploy-test scenario as shown in [these steps](#).
- You can perform a range of operations to manage VMs using SCVMM. For examples, see [SCVMM deployment](#).

Prerequisites

- SCVMM Server 2012 R2 or later.
- Window 2012 R2 host machines with Hyper-V set up with at least two physical NICs attached.
- One NIC (perhaps external) with corporate network or Internet access.
- One NIC configured in Trunk Mode with a VLAN ID (such as 991) and routable IP subnets (such as 10.10.30.1/24). You network administrator can configure this.
- All Hyper-V hosts in the host group have the same VLAN ID. This host group will be used for your isolated networks.

Verify the setup is working correctly by following these steps:

- Open an RDP session to each of the host machines and open an administrator PowerShell session.
- Run the command `Get-NetVirtualizationProviderAddress`. This gets the provider address for the physical NIC configured in trunk mode with a VLAN ID.



```
Administrator: Windows PowerShell
PS C:\Windows\system32> Get-NetVirtualizationProviderAddress

ProviderAddress : 10.10.30.175
InterfaceIndex  : 15
PrefixLength    : 24
VlanID          : 991
AddressState    : Preferred
MACAddress      : 001dd8b71f38
ManagedByCluster : False

PS C:\Windows\system32>
```

- Go to another host and open an administrator PowerShell session. Ping other machines using the command `ping -p <Provider address>`. This confirms all host machines are connected to a physical NIC in trunk mode with IPs routable across the host machines. If this test fails, contact your network administrator.

Administrator: Windows PowerShell

```
PS C:\Windows\system32> ping -p 10.10.30.175

Pinging 10.10.30.175 with 32 bytes of data:
Reply from 10.10.30.175: bytes=32 time<1ms TTL=128

Ping statistics for 10.10.30.175:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
PS C:\Windows\system32>
```

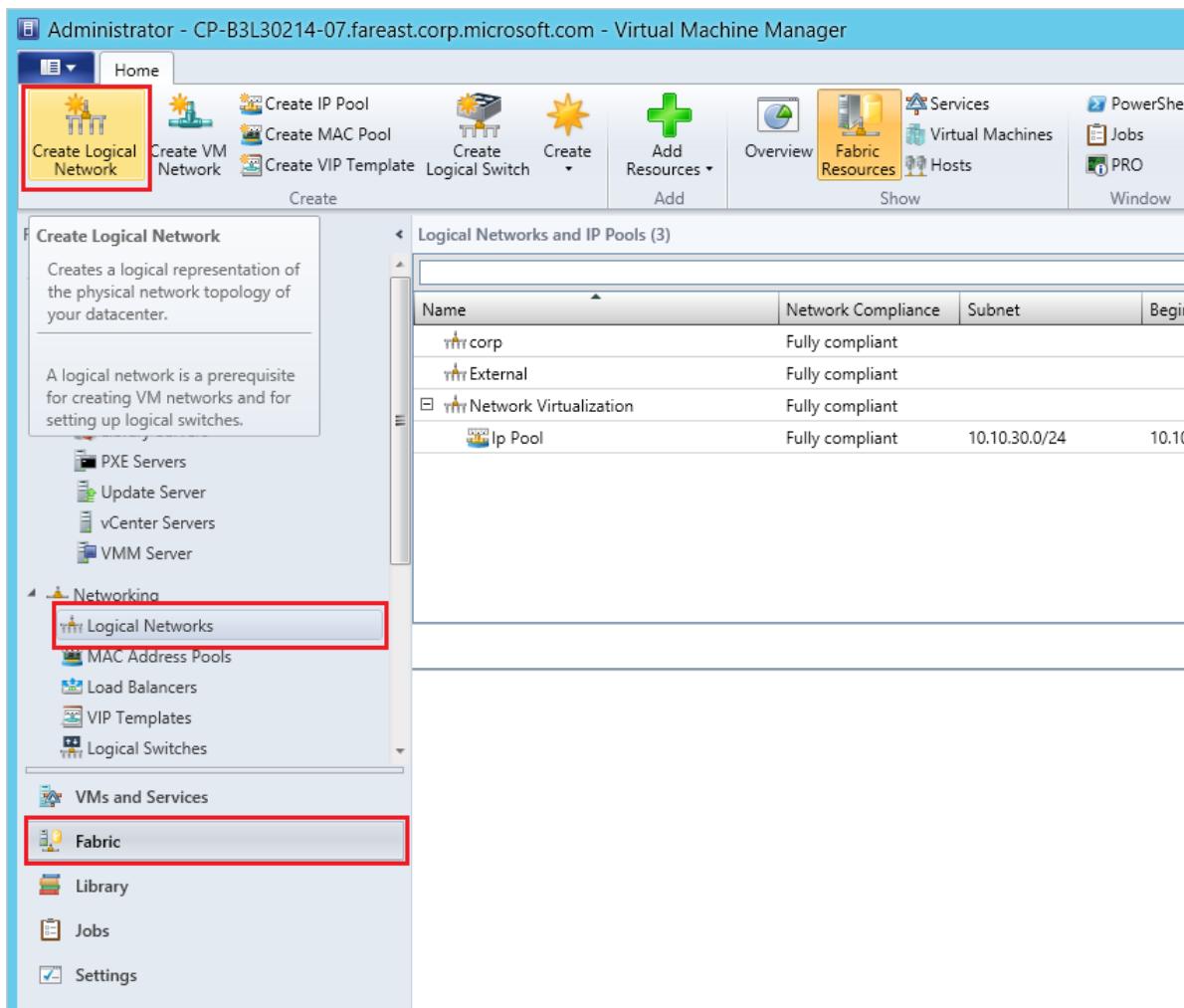
[Back to list of tasks](#)

Create a Network Virtualization layer in SCVMM

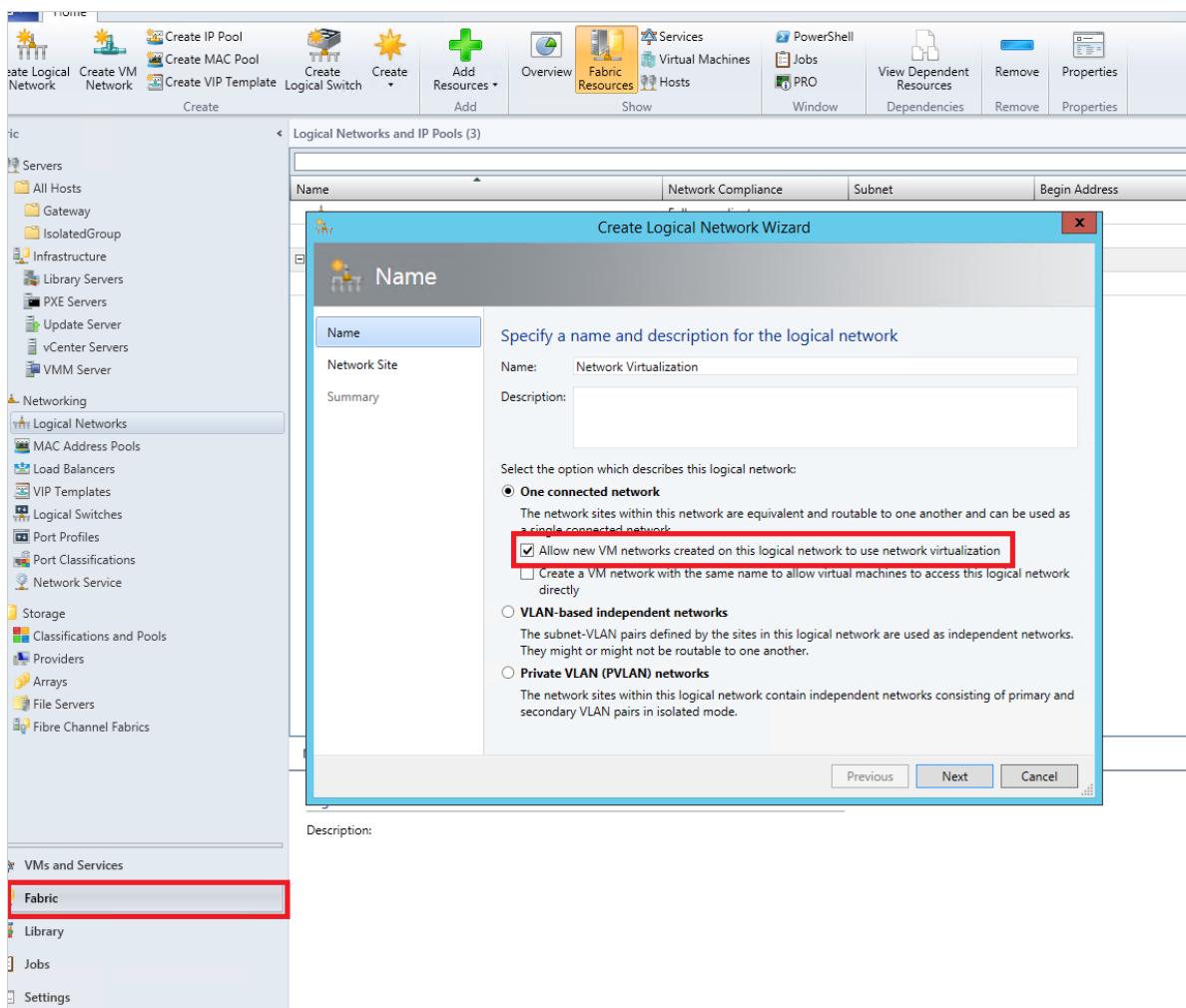
Setting up a network visualization layer in SCVMM includes creating logical networks, port profiles, logical switches, and adding the switches to the Hyper-V hosts.

Create logical networks

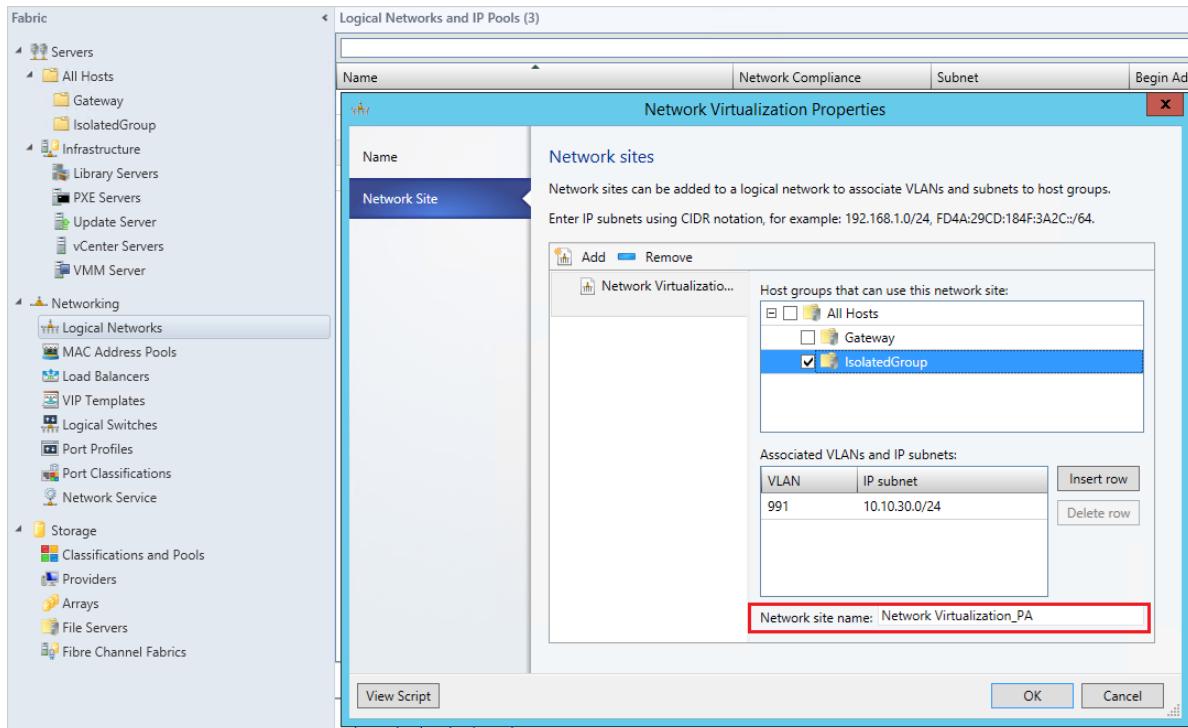
1. Log into the SCVMM admin console.
2. Go to **Fabric** -> **Networking** -> **Logical Networks** -> **Create new Logical Network**.



3. In the popup, enter an appropriate name and select **One Connected Network** -> **Allow new networks created on this logical network to use network virtualization**, then click **Next**.

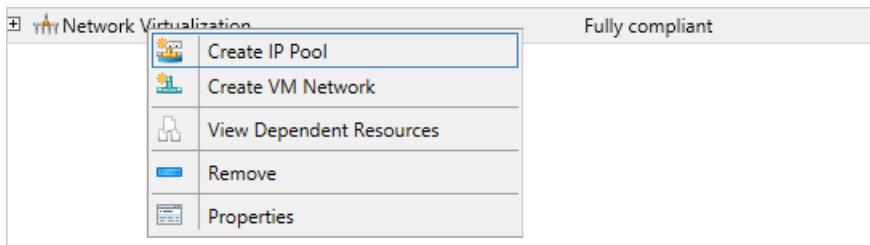


4. Add a new **Network Site** and select the host group to which the network site will be scoped. Enter the VLAN ID used to configure physical NIC in the Hyper-V host group and the corresponding routable IP subnet(s). To assist tracking, change the network site name to one that is memorable.

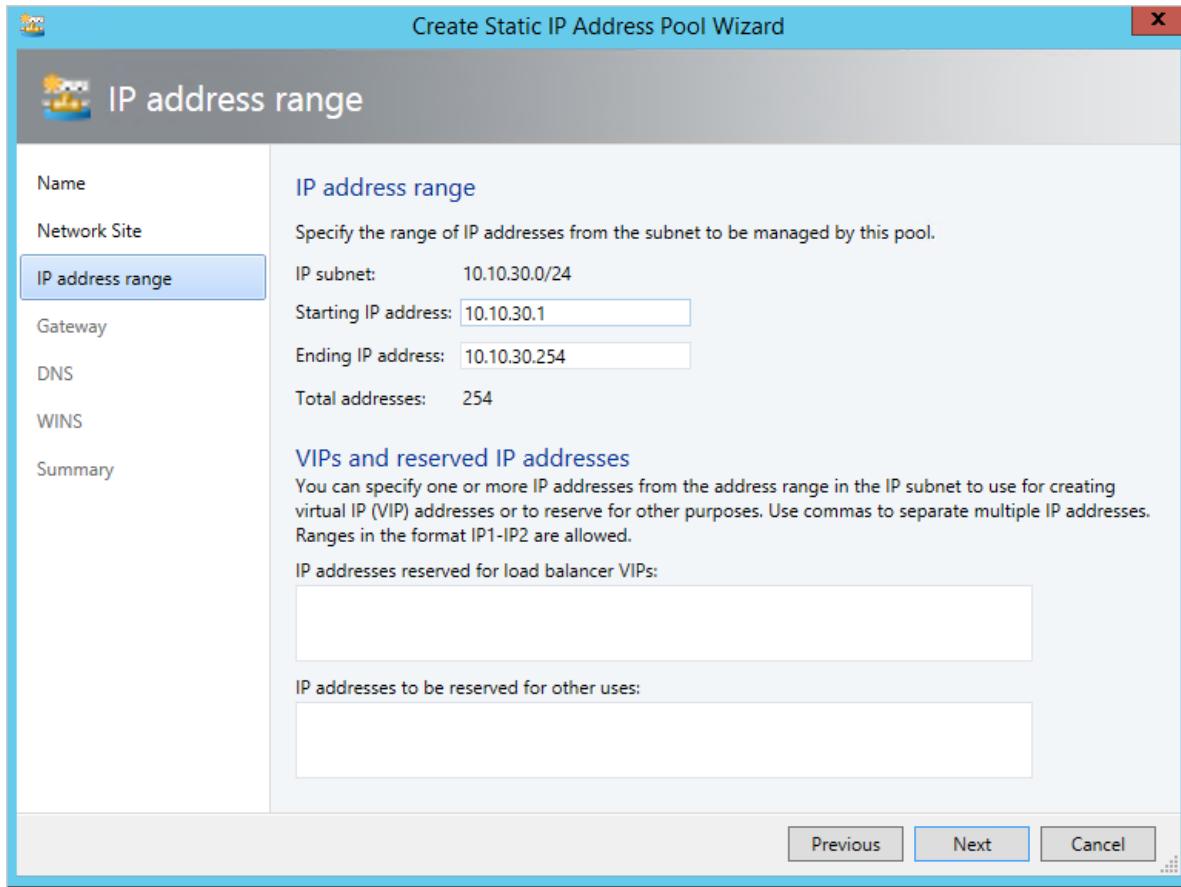


5. Click **Next** and **Save**.

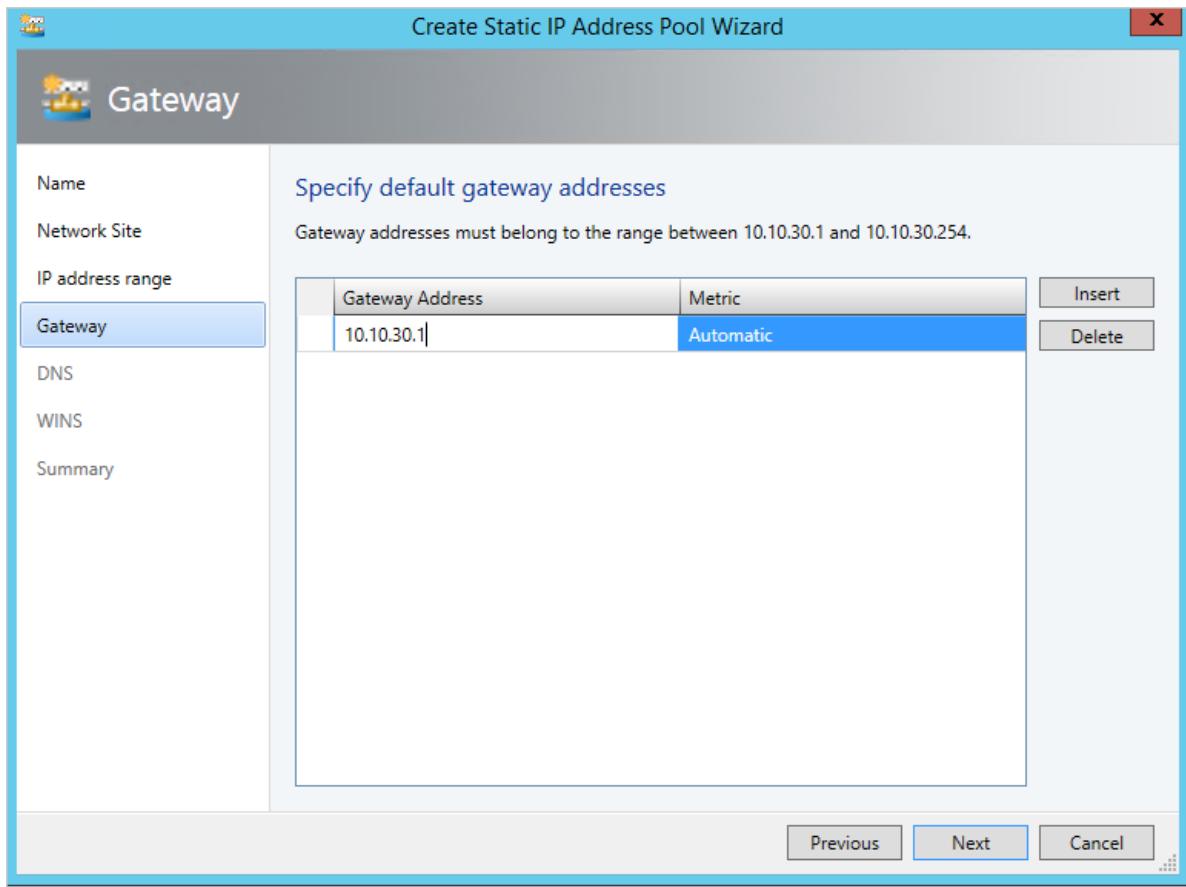
6. Create an IP pool for the new logical networks, enter a name, and click **Next**.



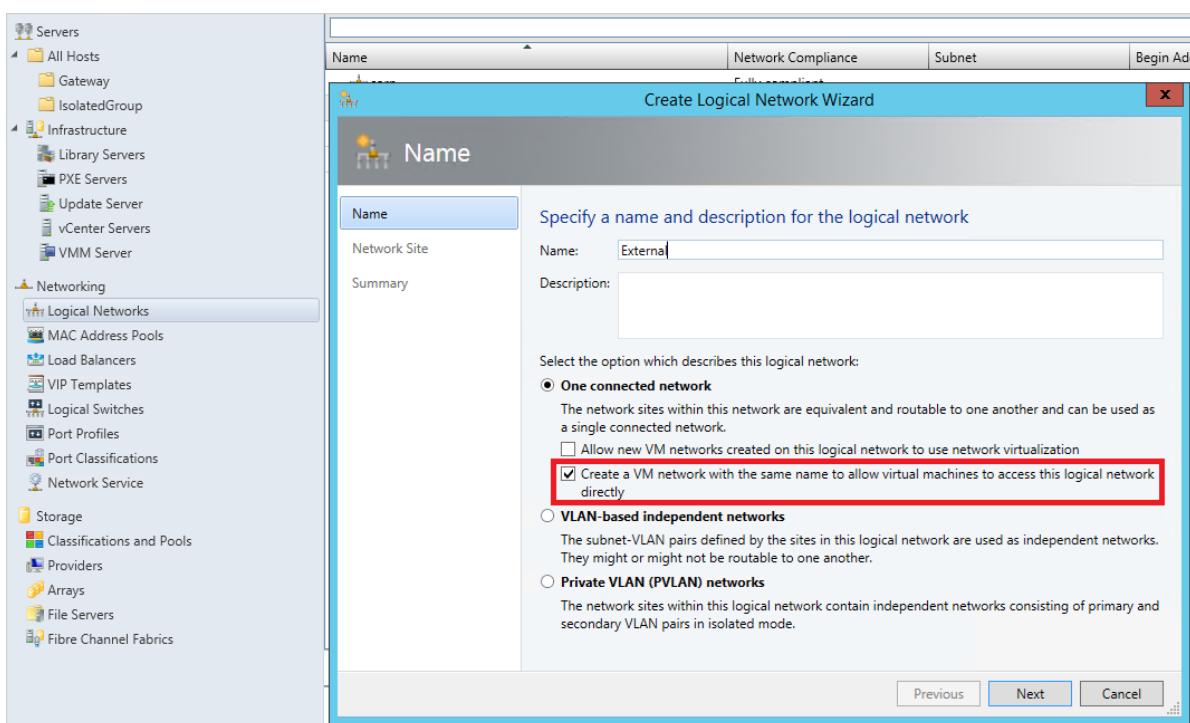
7. Select **Use and existing network site** and click **Next**. Enter the routable IP address range your network administrator configured for your VLAN and click **Next**. If you have multiple routable IP subnets associated with your VLAN, create an IP pool for each one.



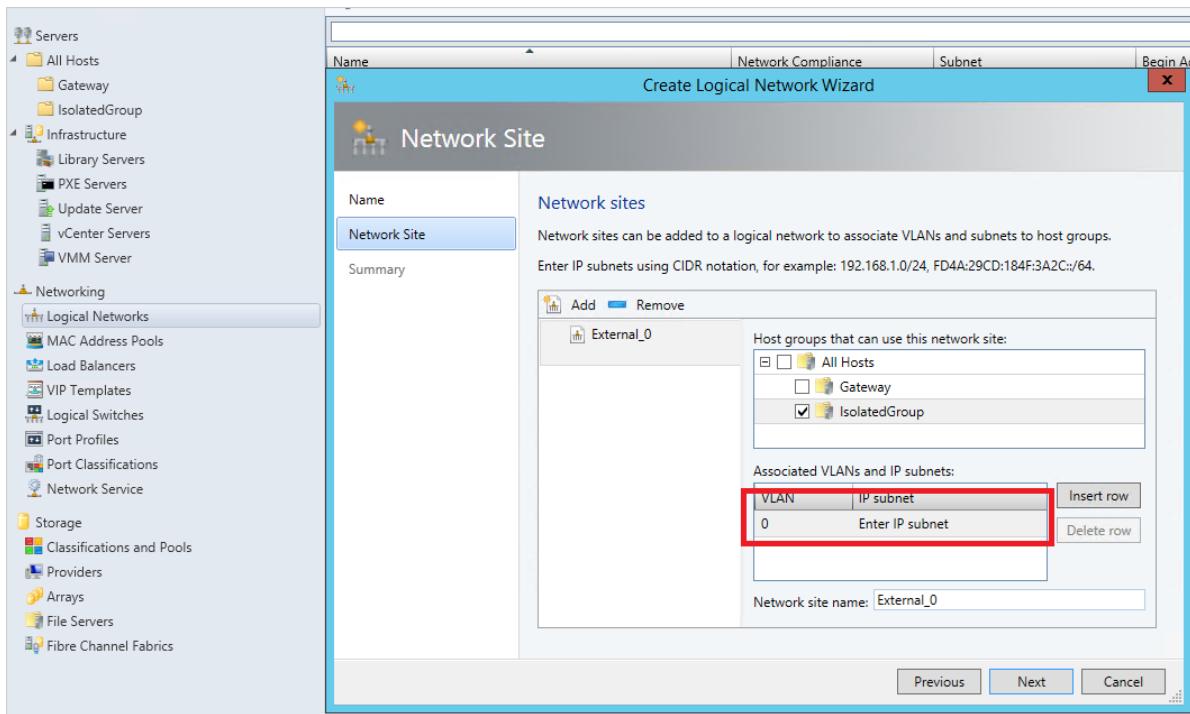
8. Provide the gateway address. By default, you can use the first IP address in your subnet.



9. Click **Next** and leave the existing DNS and WINS settings. Complete the creation of the network site.
10. Now create another **Logical Network** for external Internet access, but this time select **One Connected network** -> **Create a VM network with same name to allow virtual machines to access this logical network directly** and then click **Next**.



11. Add a network site and select the same host group, but this time add the VLAN as 0. This means the communication uses the default access mode NIC (Internet).



12. Click **Next** and **Save**.

13. The result should look like the following in your administrator console after creating the logical networks.

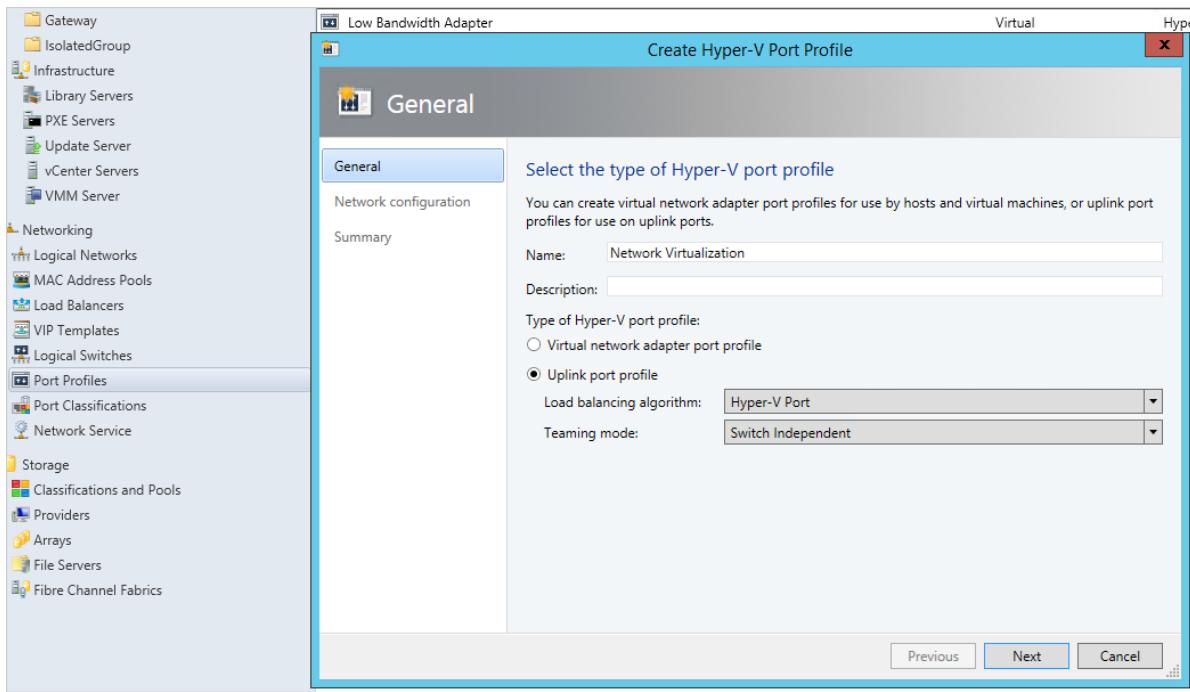
External	Fully compliant
Network Virtualization	Fully compliant
Ip Pool	Fully compliant 10.10.30.0/24 10.10.30.1 10.10.30.254 247 247

Create port profiles

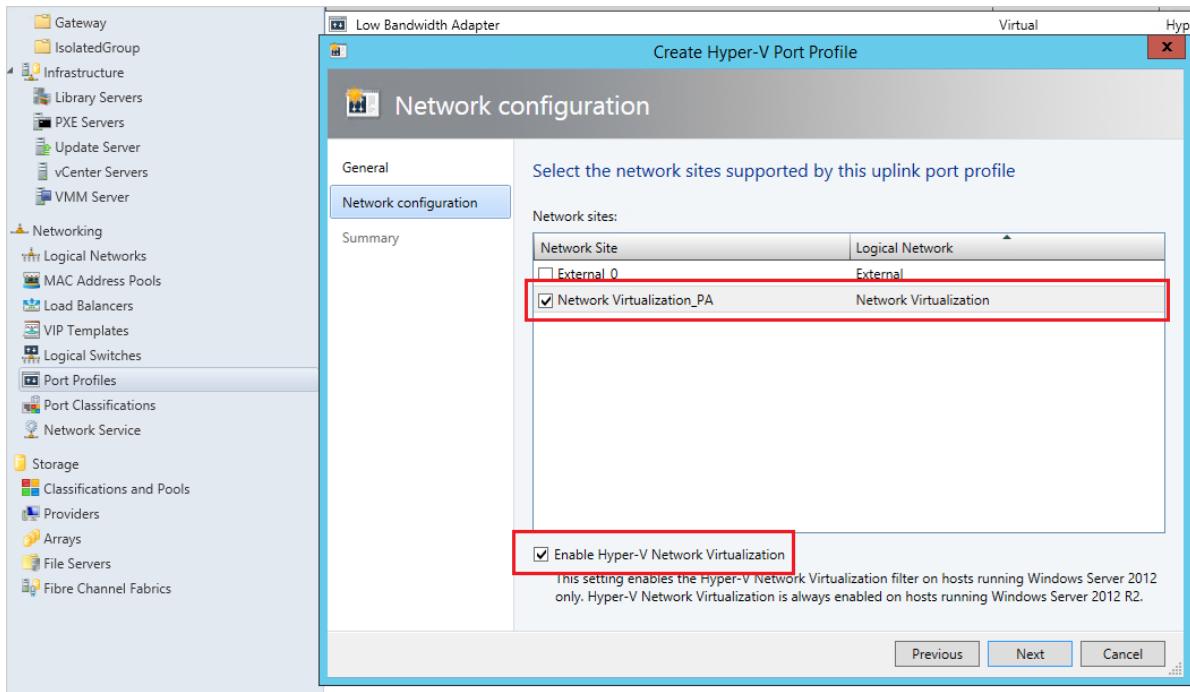
1. Go to **Fabric** -> **Networking** -> **Port profiles** and **Create Hyper-V port profile**.

Name:	Network Virtualization
Type:	Uplink
Applies to:	Hyper-V

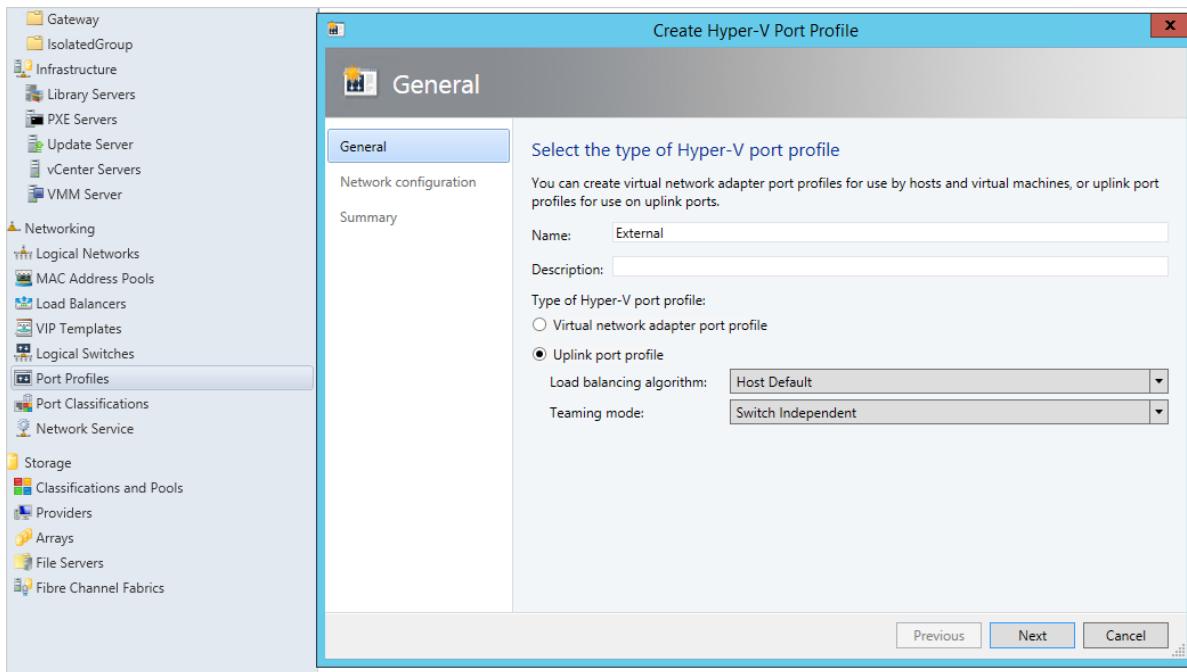
2. Select **Uplink port profile** and select **Hyper-V Port** as the load balancing algorithm, then click **Next**.



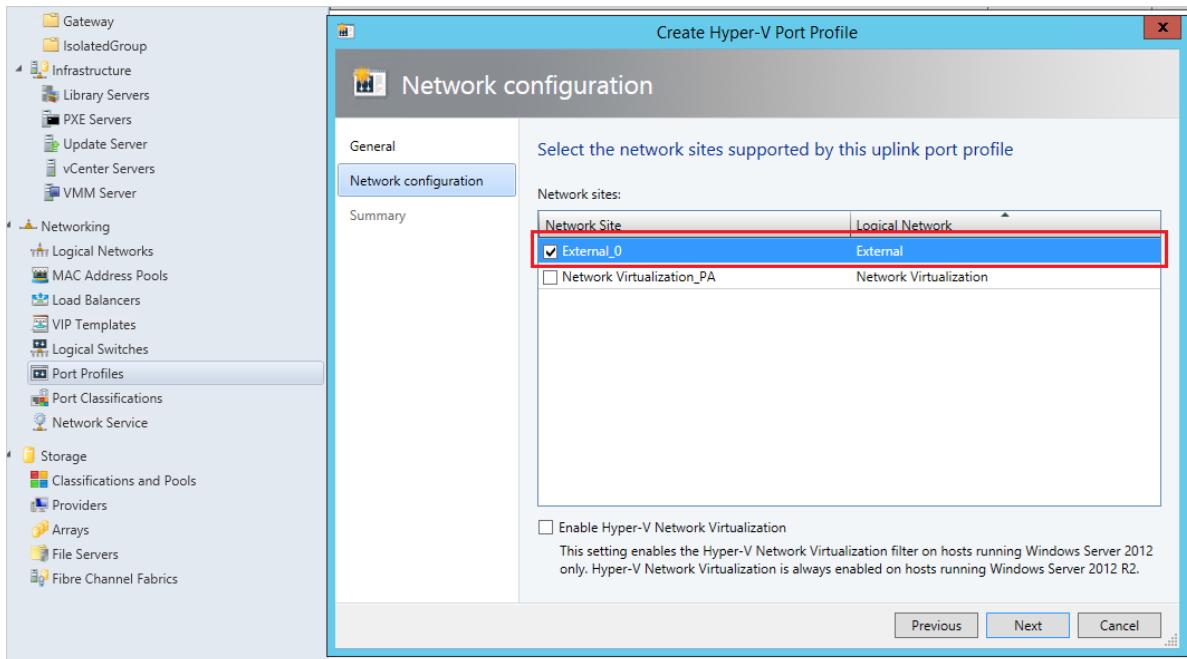
3. Select the Network Virtualization site created previously and choose the **Enable Hyper-V Network Virtualization** checkbox, then save the profile.



4. Now create another Hyper-V port profile for external logical network. Select **Uplink** mode and **Host default** as the load balancing algorithm, then click **Next**.

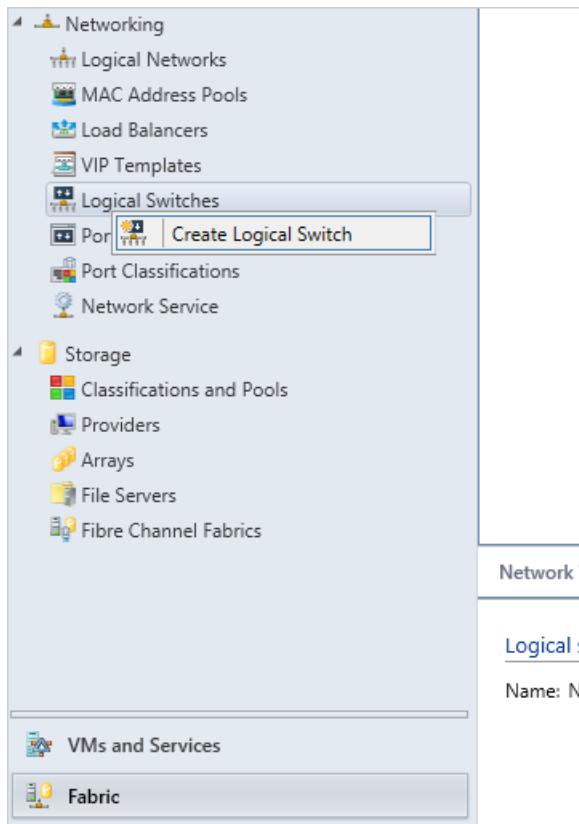


5. Select the other network site to be used for external communication, but and this time don't enable network virtualization. Then save the profile.

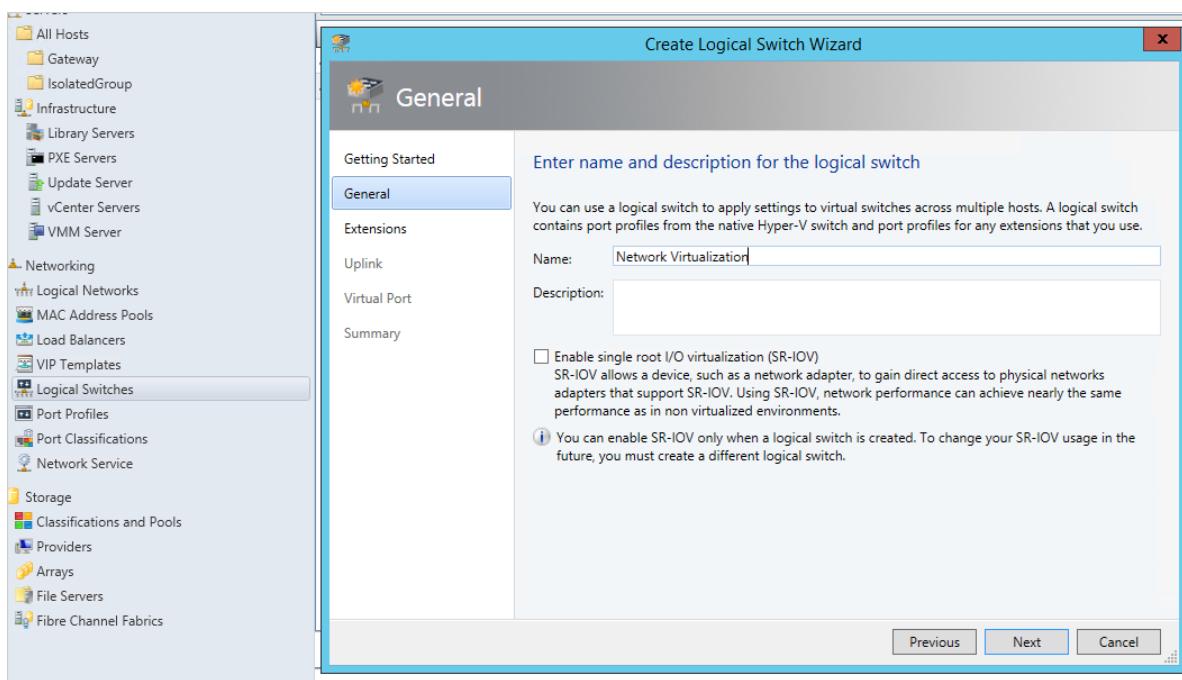


Create logical switches

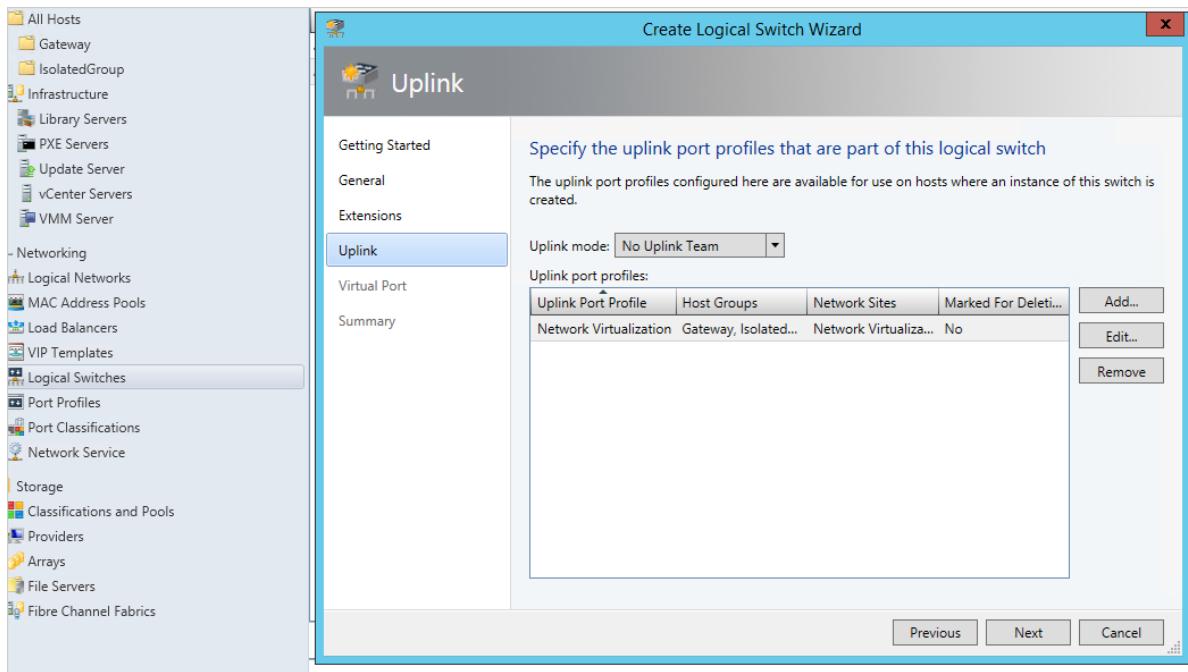
1. Go to **Fabric -> Networking -> Logical switches** and **Create Logical Switch**.



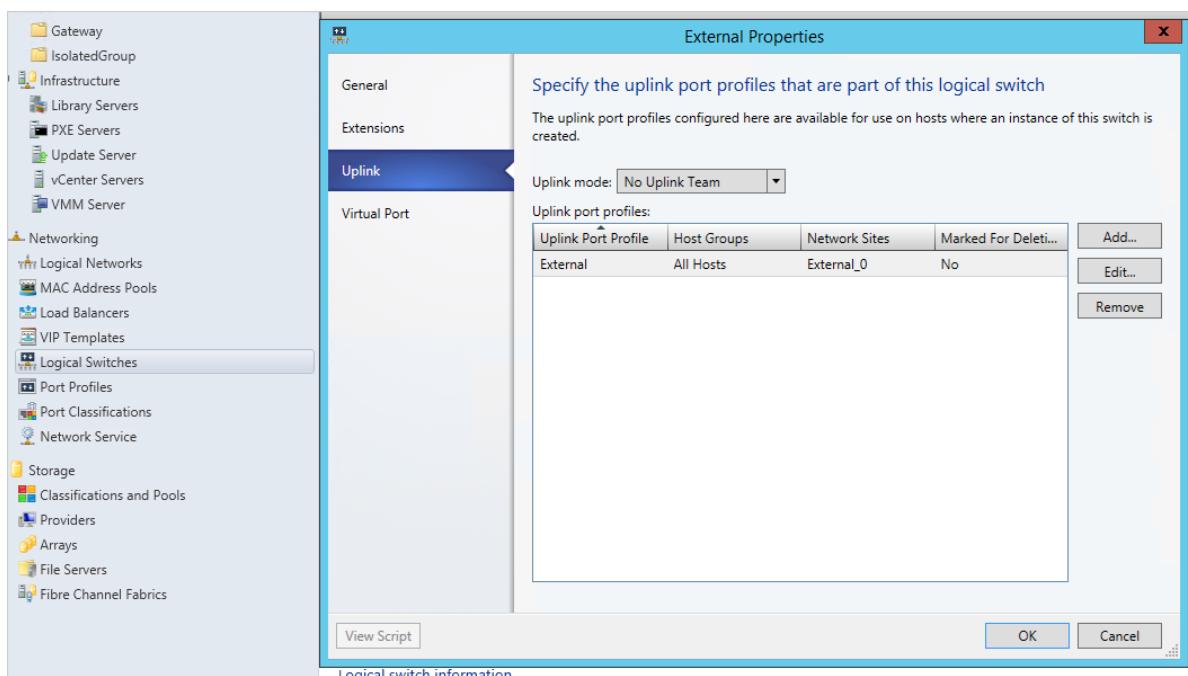
2. In the getting started wizard, click **Next** and enter a name for the switch, then click **Next**.



3. Click **Next** to open to **Uplink** tab. Click **Add uplink port profile** and add the network virtualization port profile you just created.

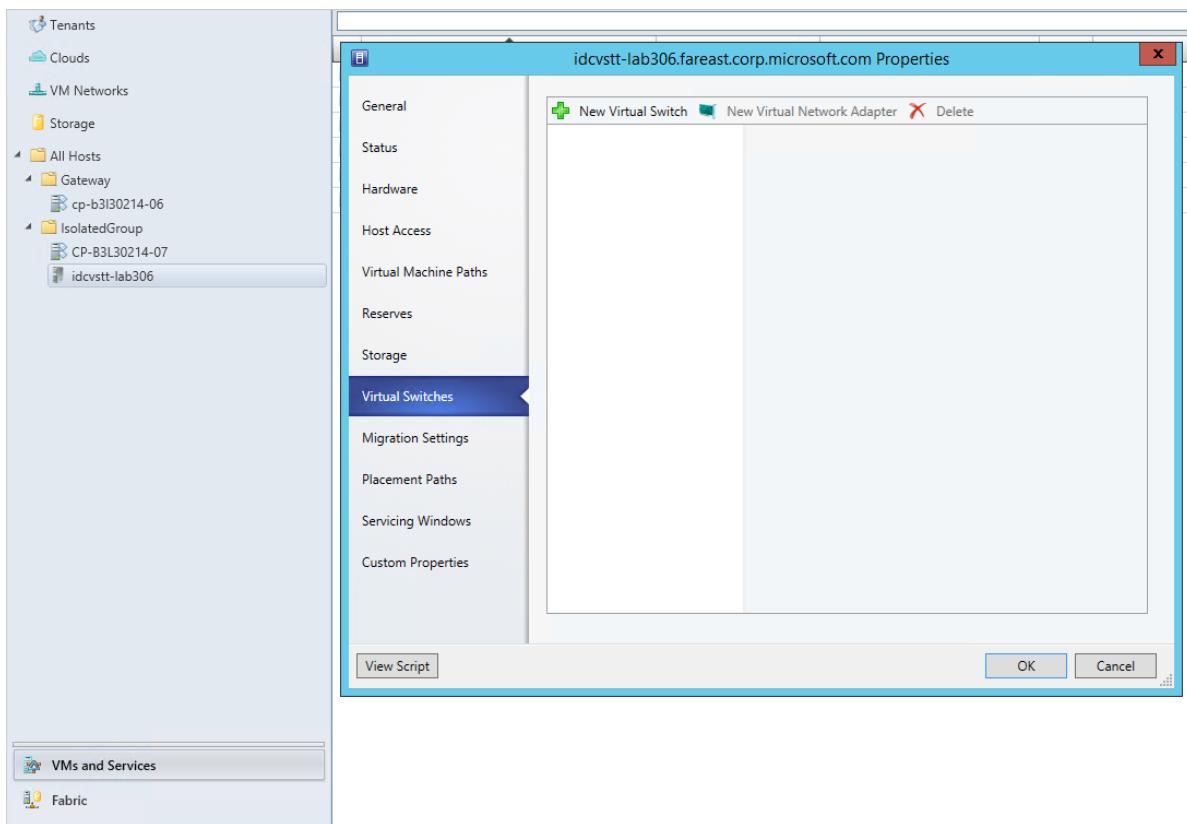


4. Click **Next** and save the logical switch.
5. Now create another logical switch for the external network for Internet communication. This time add the other uplink port profile you created for the external network.

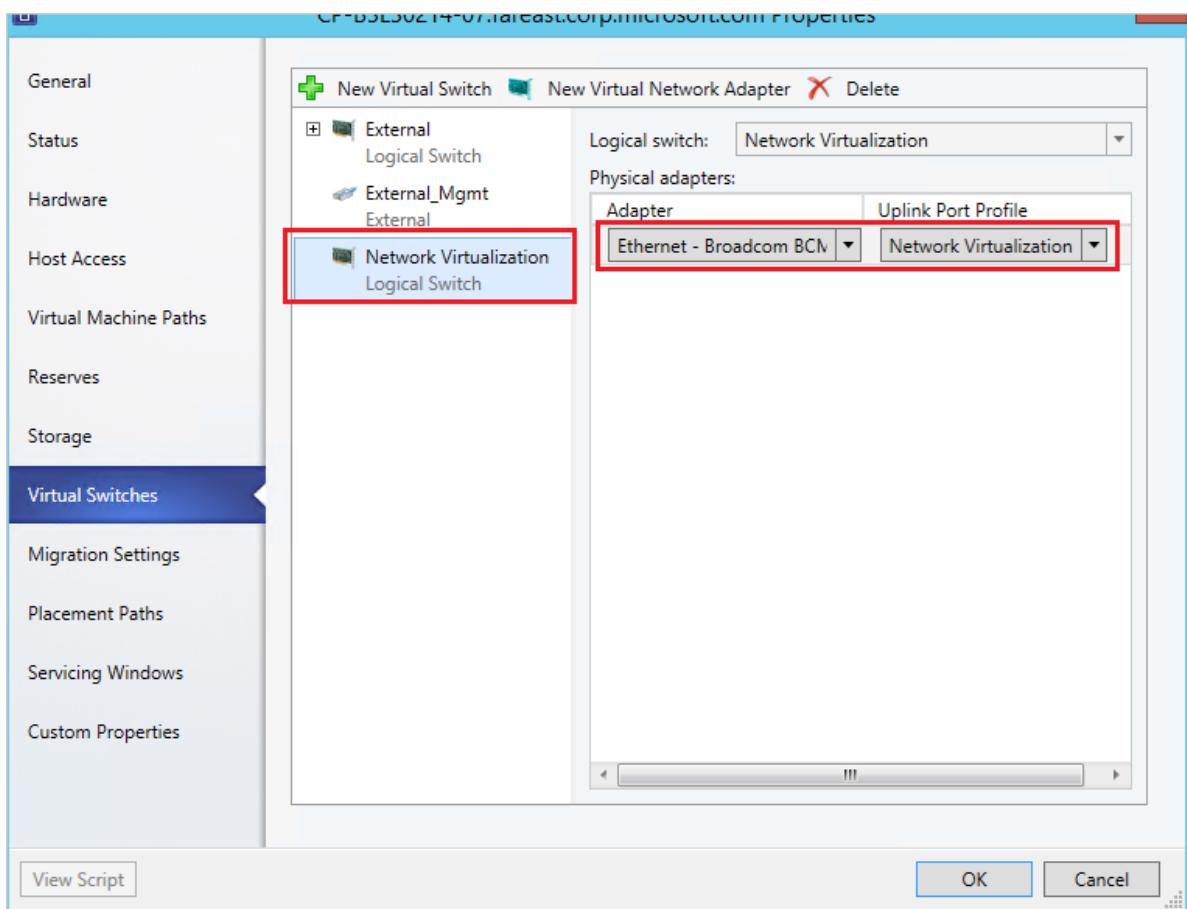


Add logical switches to Hyper-V hosts

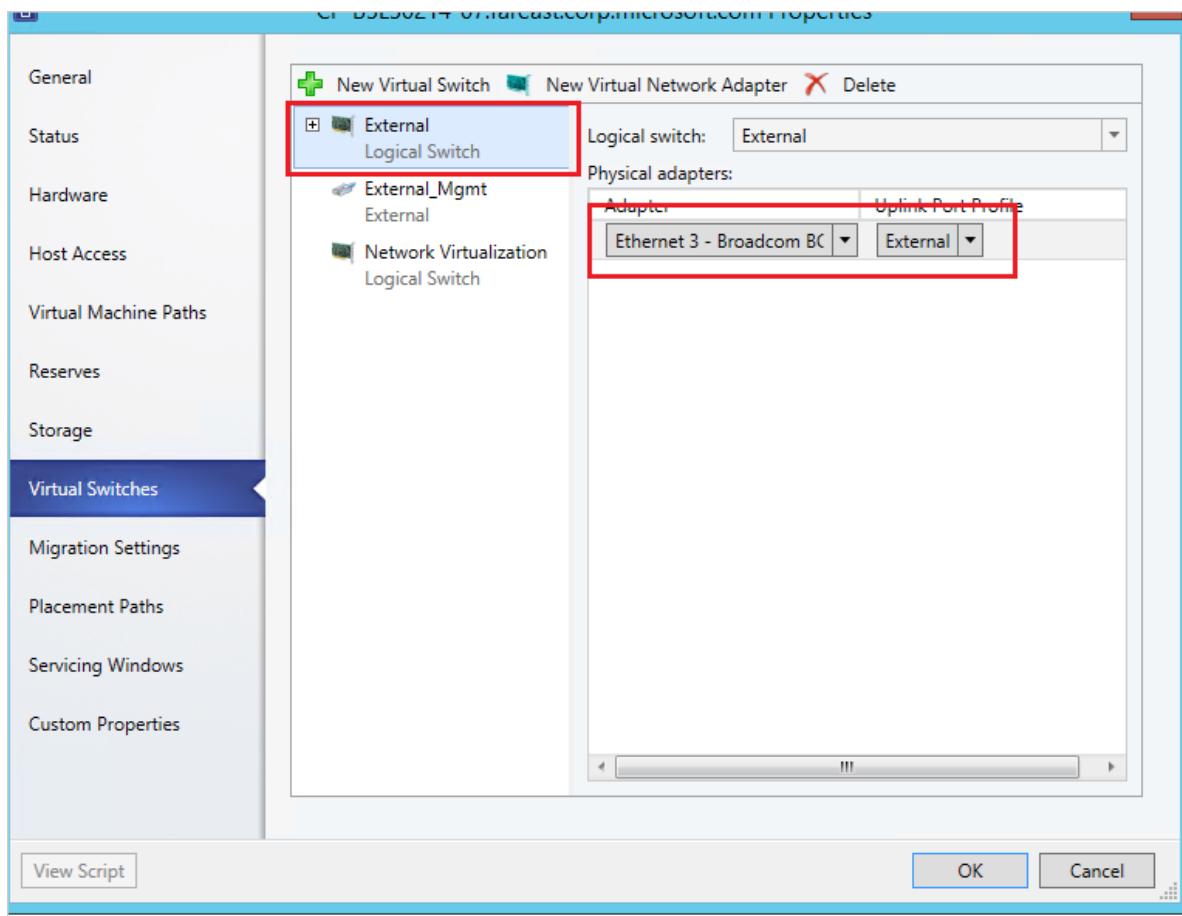
1. Go to **VM and Services** -> [Your host group] -> [each of the host machines in turn].
2. Right click and open the **Properties** -> **Virtual Switches** tab.



3. Click **New Virtual Switch** -> **New logical switch for network virtualization**. Assign the physical adapter you configured in trunk mode and select the network virtualization port profile.



4. Create another logical switch for external connectivity, assign the physical adapter used for external communication, and select the external port profile.



5. Do the same for all the Hyper-V hosts in the host group.

This is a one-time configuration for a specific host group of machines. After completing this setup, you can dynamically provision your isolated network of virtual machines using the **SCVMM extension** in TFS and Azure Pipelines builds and releases.

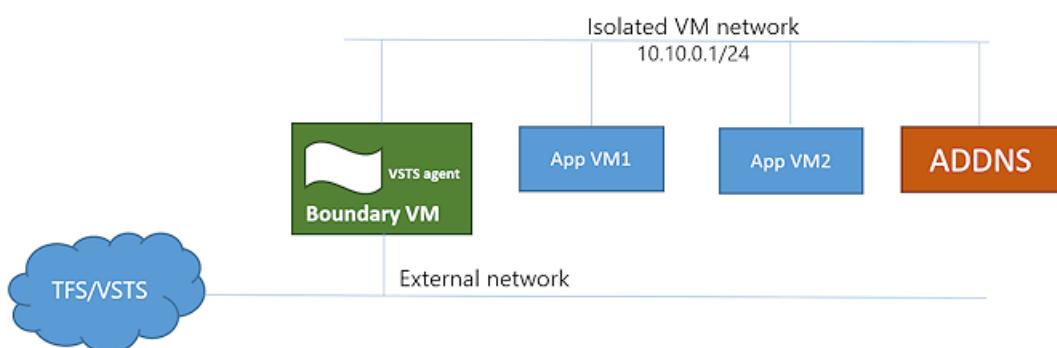
[Back to list of tasks](#)

Create the required virtual network topology

Isolated virtual networks can be broadly classified into three topologies.

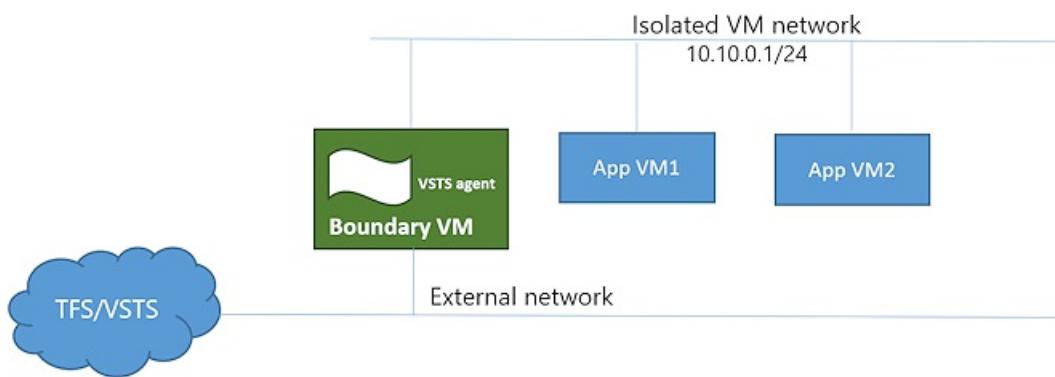
Topology 1: AD-backed Isolated VMs

- A boundary VM with Internet/TFS connectivity.
- An Azure Pipelines/TFS deployment group agent installed on the boundary VM.
- An AD-DNS VM if you want to use a local Active Directory domain.
- Isolated app VMs where you deploy and test your apps.



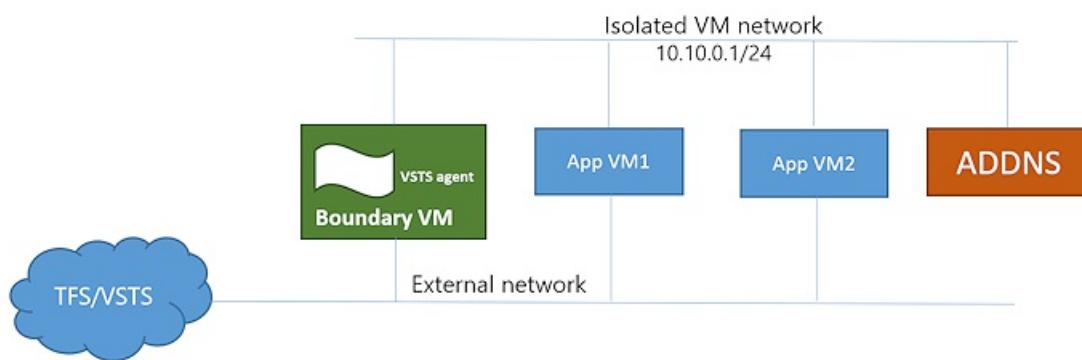
Topology 2: Non-AD backed isolated VMs

- A boundary VM with Internet/TFS connectivity.
- An Azure Pipelines/TFS deployment group agent installed on the boundary VM.
- Isolated app VMs where you deploy and test your apps.



Topology 3: AD-backed non-isolated VMs

- A boundary VM with Internet/TFS connectivity.
- An Azure Pipelines/TFS deployment group agent installed on the boundary VM.
- An AD-DNS VM if you want to use a local Active Directory domain.
- App VMs that are also connected to the external network where you deploy and test your apps.



You can create any of the above topologies using the SCVMM extension, as shown in the following steps.

1. Open your TFS or Azure Pipelines instance and install the **SCVMM extension** if not already installed. For more information, see [SCVMM deployment](#).

The **SCVMM task** provides a more efficient way capability to perform lab management operations using build and release pipelines. You can manage SCVMM environments, provision isolated virtual networks, and implement build-deploy-test scenarios.

2. In a build or release pipeline, add a new **SCVMM** task.
3. In the **SCVMM** task, select a service connection for the SCVMM server where you want to provision your virtual network and then select **New Virtual machines using Templates/Stored VMs and VHDS** to provision your VMs.

The screenshot shows the 'Tasks' tab of a pipeline definition named 'BDT_Scenario'. On the left, there's an 'Environment 1' section and an 'Agent phase' section containing a 'SCVMM :NewVM action' task. The task is configured with a 'Version' of '1.*', a 'Display name' of 'SCVMM :NewVM action', and a 'Service Connection' of 'scvmm'. The 'Action' dropdown is set to 'New Virtual Machine using Template/stored VM/VHD'.

4. You can create VMs from templates, stored VMs, and VHD/VHDx. Choose the appropriate option and enter the VM names and corresponding source information.

The screenshot shows the detailed configuration for the 'SCVMM :NewVM action' task. Under 'VM Template options', the 'Create virtual machines from VM Templates' checkbox is checked, and the 'Virtual machine names' field contains '\$(AppVM1)'. Under 'Stored VM options', the 'Create virtual machines from stored VMs' checkbox is checked, and the 'VM names' field contains '\$(AppVM2)'. Both fields have their respective labels underlined in red.

5. In case of topologies **1** and **2**, leave the **VM Network name** empty, which will clear all the old VM networks present in the created VMs (if any). For topology **3**, you must provide information about the external VM network here.

All definitions > BDT_Scenario

Pipeline Tasks Variables Retention Options History

Environment 1 Deployment process

Agent phase Run on agent

SCVMM :NewVM action SCVMM

VHD options

Create virtual machines from vhd

Override existing VM Network settings

Clear existing network adapters

VM network name

6. Enter the **Cloud Name** of the host where you want to provision your isolated network. In case of private cloud, ensure the host machines added to the cloud are connected to the same logical and external switches as explained above.

All definitions > BDT_Scenario

Pipeline Tasks Variables Retention Options History

Environment 1 Deployment process

Agent phase Run on agent

SCVMM :NewVM action SCVMM

Create VM options

Deploy the VMs to

Cloud

Cloud Name

`$(BDT Cloud)`

Additional Arguments

`-StartVM -StartAction NeverAutoTurnOnVM -StopAction SaveVM`

Wait Time

600

7. Select the **Network Virtualization** option to create the virtualization layer.

All definitions > BDT_Scenario

Pipeline Tasks Variables Retention Options History

Environment 1 Deployment process

Agent phase Run on agent

SCVMM :NewVM action SCVMM

Additional Arguments -StartVM -StartAction NeverAutoTurnOnVM -StopAction SaveVM

Wait Time 600

Network Virtualization options

Network Virtualization

8. Based on the topology you would like to create, decide if the network requires an Active Directory VM. For example, to create Topology 2 (AD-backed isolated network), you require an Active directory VM. Select the **Add Active Directory VM** checkbox, enter the AD VM name and the stored VM source. Also enter the static IP address configured in the AD VM source and the DNS suffix.

All definitions > BDT_Scenario

Pipeline Tasks Variables Retention Options History

Environment 1 Deployment process

Agent phase Run on agent

SCVMM :NewVM action SCVMM

Network Virtualization options

Network Virtualization

Add an Active Directory VM

Active Directory VM name * \$(AD_VM1)

Stored VM * \$(Stored_ADVM)

DNS IP 10.10.1.100

DNS Suffix * fabrikam.com

9. Enter the settings for the **VM Network** and subnet you want to create, and the backing logical network you created in the [previous section](#) (Logical Networks). Ensure the VM network name is unique. If possible, append the release name for easier tracking later.

All definitions > BDT_Scenario

Pipeline Tasks Variables Retention Options History

Environment 1 Deployment process ...

Agent phase Run on agent +

SCVMM :NewVM action SCVMM

VM Network

Create new Use existing

VM Network name * \$(VM_Network)_\$(Release.DefinitionName)_\$(Release.ReleaseName)

VM network subnet * 10.10.1.0/24

Logical Network name * NetworkVirtualization

- In the **Boundary Virtual Machine options** section, set **Create boundary VM for communication with Azure Pipelines/TFS**. This will be the entry point for external communication.
- Enter the boundary VM name and the source template (the boundary VM source should always be a VM template), and enter name of the existing external VM network you created for external communication.

All definitions > BDT_Scenario

Pipeline Tasks Variables Retention Options History

Environment 1 Deployment process ...

Agent phase Run on agent +

SCVMM :NewVM action SCVMM

Boundary Virtual Machine options

Create boundary VM for VSTS/TFS communication

TFS/VSTS boundary VM name * \$(Boundary_VM)

TFS/VSTS boundary VM template * \$(Boundary_Template)

External VM network for boundary VM * \$(External_Network)

Subnet name for external VM network

- Provide details for configuring the boundary VM agent to communicate with Azure Pipelines/TFS. You can configure a deployment agent or an automation agent. This agent will be used for app deployments.

The screenshot shows the 'Tasks' tab of a pipeline named 'BDT_Scenario'. On the left, there's a tree view with 'Environment 1' expanded, showing 'Agent phase' and 'SCVMM :NewVM action'. The 'Agent phase' node has a 'Run on agent' icon. On the right, under 'Boundary VM agent type', the 'Deployment Group agent' radio button is unselected (gray), while the 'Automation Pool agent' radio button is selected (blue). A red box highlights this selection. Below it, fields for 'Pool name' (set to '\$(AutomationPool)'), 'Personal Access Token' (set to '\${MyPAT}'), 'Agent name' (set to '\${MyAgent}_\${Release.ReleaseName}'), 'Agent installation path' (set to '\${AgentPath}'), 'Boundary VM administrator' (set to '\${UserName}'), and 'Boundary VM password' (set to '\${Password}') are shown.

13. Ensure the agent name you provide is unique. This will be used as demand in succeeding job properties so that the correct agent will be selected. If you selected the deployment group agent option, this parameter is replaced by the value of the tag, which must also be unique.
14. Ensure the boundary VM template has the agent configuration files downloaded and saved in the VHD before the template is created. Use this path as the agent installation path above.

Enable your build-deploy-test scenario

1. Create a new job in your pipeline, after your network virtualization job.
2. Based on the boundary VM agent (deployment group agent or automation agent) that is created as part of your boundary VM provisioning, choose **Deployment group job** or **Agent job**.
3. In the job properties, select the appropriate deployment group or automation agent pool.
4. In the case of an automation pool, add a new **Demand for Agent.Name** value. Enter the unique name used in the network virtualization job. In the case of deployment group job, you must set the tag in the properties of the group machines.

The screenshot shows the 'Tasks' tab for the 'BDT_Scenario' pipeline. In the 'Environment 1' section, there is a 'SCVMM :NewVM action' task. Below it, in the 'Agent phase' section, there are two tasks: 'SCVMM :NewVM action' and 'Run PowerShell on Target Machines'. The 'Run PowerShell on Target Machines' task is expanded, revealing its configuration. A red box highlights the 'Demands' table, which contains a single entry: 'Agent.Name equals \$(MyAgent)_\$(Release.ReleaseName)'. The 'Execution plan' section is also visible.

5. Inside the job, add the tasks you require for deployment and testing.

The screenshot shows the 'Tasks' tab for the 'BDT_Scenario' pipeline. The 'Run PowerShell on Target Machines' task is expanded, showing its configuration. A red box highlights the 'Machines' field, which contains the value '\$(AppVM1), \$(AppVM2)'. Other fields shown include 'Display name' (Run PowerShell on \$(AppVM1), \$(AppVM2)) and 'Admin Login' (\$(UserName)).

6. After testing is completed, you can destroy the VMs by using the **Delete VM** task option.

Now you can create release from this release pipeline. Each release will dynamically provision your isolated virtual network and run your deploy and test tasks in the environment. You can find the test results in the release summary. After your tests are completed, you can automatically decommission your environments. You can create as many environments as you need with just a click from **Azure Pipelines**.

[Back to list of tasks](#)

See also

- [SCVMM deployment](#)
- [Hyper-V Network Virtualization Overview](#)

Q&A

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Troubleshoot Build and Release

10/24/2018 • 11 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015

This topic provides general troubleshooting guidance. For specific troubleshooting about .NET Core, see [.NET Core troubleshooting](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Run commands locally at the command prompt

It is helpful to narrow whether a build or release failure is the result of an Azure Pipelines/TFS product issue (agent or tasks). Build and release failures may also result from external commands.

Check the logs for the exact command-line executed by the failing task. Attempting to run the command locally from the command line may reproduce the issue. It can be helpful to run the command locally from your own machine, and/or log-in to the machine and run the command as the service account.

For example, is the problem happening during the MSBuild part of your build pipeline (for example, are you using either the [MSBuild](#) or [Visual Studio Build](#) task)? If so, then try running the same [MSBuild command](#) on a local machine using the same arguments. If you can reproduce the problem on a local machine, then your next steps are to investigate the [MSBuild](#) problem.

Differences between local command prompt and agent

Keep in mind, some differences are in effect when executing a command on a local machine and when a build or release is running on an agent. If the agent is configured to run as a service on Linux, macOS, or Windows, then it is not running within an interactive logged-on session. Without an interactive logged-on session, UI interaction and other limitations exist.

Get logs to diagnose problems

Build and Release logs

Start by looking at the logs in your completed build or release. If they don't provide enough detail, you can make them more verbose:

1. On the **Variables** tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. Queue the build or release.
3. In the explorer tab, view your completed build or release and click the failing task to view its output.
4. If you need a copy of all the logs, click **Download all logs as zip**.

Diagnostic logs

1. On the build summary page, find the **Queue new build** button, next to it there is a drop down. Click the down arrow and choose **Queue new build with diagnostic logs**.
2. Queue the build.

3. On the build summary page, there will now be a **Diagnostic logs** section. You can download your diagnostic logs per job. If you would like to download everything you can also choose to **Download all logs as zip**.

Diagnostic logs are not yet available for releases.

Worker diagnostic logs

You can get the diagnostic log of the completed build that was generated by the worker process on the build agent. Look for the `worker` log file that has the date and time stamp of your completed build. For example, `worker_20160623-192022-utc_6172.log`.

Agent diagnostic logs

Agent diagnostic logs provide a record of how the agent was configured and what happened when it ran. Look for the `agent` log files. For example, `agent_20160624-144630-utc.log`. There are two kinds of agent log files:

- The log file generated when you ran `config.cmd`. This log:
 - Includes this line near the top: `Adding Command: configure`
 - Shows the configuration choices made.
- The log file generated when you ran `run.cmd`. This log:
 - Cannot be opened until the process is terminated.
 - Attempts to connect to your Azure DevOps organization or Team Foundation Server.
 - Shows when each job was run, and how it completed

Both logs show how the agent capabilities were detected and set.

Other logs

Inside the diagnostic logs you will find `environment.txt` and `capabilities.txt`.

The `environment.txt` file has various information about the environment within which your build ran. This includes information like what Tasks are run, whether or not the firewall is enabled, Powershell version info, and some other items. We continually add to this data to make it more useful.

The `capabilities` file provides a clean way to see all capabilities installed on the build machine that ran your build.

HTTP trace logs

Important: HTTP traces and trace files can contain passwords and other secrets. Do **not** post them on a public sites.

Use built-in HTTP tracing

If your agent is version 2.114.0 or newer, you can trace the HTTP traffic headers and write them into the diagnostic log. Set the `VSTS_AGENT_HTTPTRACE` environment variable before you launch the `agent.listener`.

```
Windows:  
set VSTS_AGENT_HTTPTRACE=true
```

```
macOS/Linux:  
export VSTS_AGENT_HTTPTRACE=true
```

Use full HTTP tracing

`Windows`

1. Start [Fiddler](#).
2. We recommend you listen only to agent traffic. File > Capture Traffic off (F12)
3. Enable decrypting HTTPS traffic. Tools > Fiddler Options > HTTPS tab. Decrypt HTTPS traffic
4. Let the agent know to use the proxy:

```
set VSTS_HTTP_PROXY=http://127.0.0.1:8888
```

5. Run the agent interactively. If you're running as a service, you can set as the environment variable in control panel for the account the service is running as.
6. Restart the agent.

macOS and Linux

Use Charles Proxy (similar to Fiddler on Windows) to capture the HTTP trace of the agent.

1. Start Charles Proxy.
2. Charles: Proxy > Proxy Settings > SSL Tab. Enable. Add URL.
3. Charles: Proxy > Mac OSX Proxy. Recommend disabling to only see agent traffic.

```
export VSTS_HTTP_PROXY=http://127.0.0.1:8888
```

4. Run the agent interactively. If it's running as a service, you can set in the .env file. See [nix service](#)
5. Restart the agent.

File- and folder-in-use errors

File or folder in use errors are often indicated by error messages such as:

Access to the path [...] is denied.
 The process cannot access the file [...] because it is being used by another process.
 Access is denied.
 Can't move [...] to [...]

Detect files and folders in use

On Windows, tools like [Process Monitor](#) can be used to capture a trace of file events under a specific directory. Or, for a snapshot in time, tools like [Process Explorer](#) or [Handle](#) can be used.

Anti-virus exclusion

Anti-virus software scanning your files can cause file or folder in use errors during a build or release. Adding an anti-virus exclusion for your agent directory and configured "work folder" may help to identify anti-virus software as the interfering process.

MSBuild and /nodeReuse:false

If you invoke MSBuild during your build, make sure to pass the argument `/nodeReuse:false` (short form `/nr:false`). Otherwise MSBuild process(es) will remain running after the build completes. The process(es) remain for some time in anticipation of a potential subsequent build.

This feature of MSBuild can interfere with attempts to delete or move a directory - due to a conflict with the working directory of the MSBuild process(es).

The MSBuild and Visual Studio Build tasks already add `/nr:false` to the arguments passed to MSBuild.

However, if you invoke MSBuild from your own script, then you would need to specify the argument.

MSBuild and /maxcpucount:[n]

By default the build tasks such as [MSBuild](#) and [Visual Studio Build](#) run MSBuild with the `/m` switch. In some cases this can cause problems such as multiple process file access issues.

Try adding the `/m:1` argument to your build tasks to force MSBuild to run only one process at a time.

File-in-use issues may result when leveraging the concurrent-process feature of MSBuild. Not specifying the argument `/maxcpucount:[n]` (short form `/m:[n]`) instructs MSBuild to use a single process only. If you are using the MSBuild or Visual Studio Build tasks, you may need to specify `/m:1` to override the `/m` argument that is added by default.

Intermittent or inconsistent MSBuild failures

If you are experiencing intermittent or inconsistent MSBuild failures, try instructing MSBuild to use a single-process only. Intermittent or inconsistent errors may indicate that your target configuration is incompatible with the concurrent-process feature of MSBuild. See [MSBuild and /maxcpucount:\[n\]](#)

Process hang

Waiting for Input

A process hang may indicate that a process is waiting for input.

Running the agent from the command line of an interactive logged on session may help to identify whether a process is prompting with a dialog for input.

Running the agent as a service may help to eliminate programs from prompting for input. For example in .Net, programs may rely on the `System.Environment.UserInteractive` Boolean to determine whether to prompt. When running as a Windows service, the value is false.

Process dump

Analyzing a dump of the process can help to identify what a deadlocked process is waiting on.

WiX project

Building a WiX project when custom MSBuild loggers are enabled, can cause WiX to deadlock waiting on the output stream. Adding the additional MSBuild argument `/p:RunWixToolsOutOfProc=true` will workaround the issue.

Line endings for multiple platforms

When you run pipelines on multiple platforms, you can sometimes encounter problems with different line endings. Historically, Linux and macOS used linefeed (LF) characters while Windows used a carriage return plus a linefeed (CRLF). Git tries to compensate for the difference by automatically making lines end in LF in the repo but CRLF in the working directory on Windows.

Most Windows tools are fine with LF-only endings, and this automatic behavior can cause more problems than it solves. If you encounter issues based on line endings, we recommend you configure Git to prefer LF everywhere. To do this, add a `.gitattributes` file to the root of your repository. In that file, add the following line:

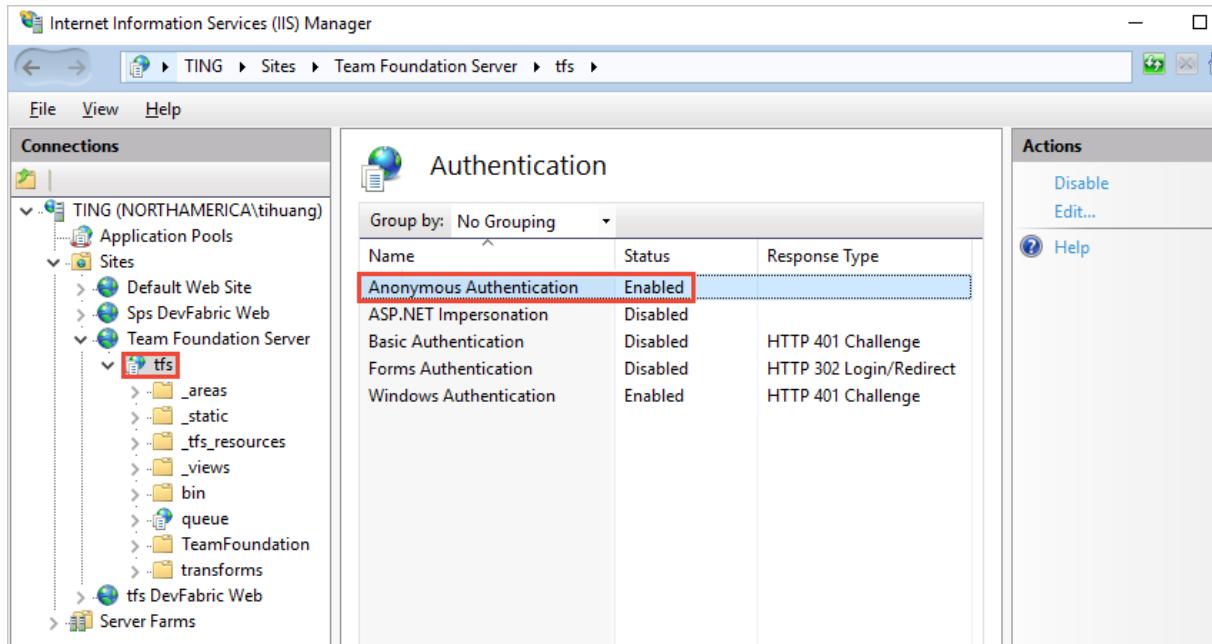
```
* text eol=lf
```

Agent connection issues

Config fails while testing agent connection (on-premises TFS only)

```
Testing agent connection.  
VS30063: You are not authorized to access http://<SERVER>:8080/tfs
```

If the above error is received while configuring the agent, log on to your TFS machine. Start the Internet Information Services (IIS) manager. Make sure **Anonymous Authentication** is enabled.



The screenshot shows the IIS Manager interface. The left pane displays the 'Connections' tree, which includes the 'TING (NORTHAMERICA\tihuang)' node, 'Application Pools', 'Sites' (with 'Default Web Site', 'Sps DevFabric Web', and 'Team Foundation Server' expanded), and 'tfs' (with subfolders '_areas', '_static', '_tfs_resources', '_views', 'bin', 'queue', 'TeamFoundation', and 'transforms'). The 'Team Foundation Server' node is selected. The right pane is titled 'Authentication' and lists the following configuration:

Name	Status	Response Type
Anonymous Authentication	Enabled	
ASP.NET Impersonation	Disabled	
Basic Authentication	Disabled	HTTP 401 Challenge
Forms Authentication	Disabled	HTTP 302 Login/Redirect
Windows Authentication	Enabled	HTTP 401 Challenge

The 'Anonymous Authentication' row is highlighted with a red box. The 'Actions' ribbon on the right side of the interface has buttons for 'Disable', 'Edit...', and 'Help'.

Agent lost communication

This issue is characterized by the error message:

```
The job has been abandoned because agent did not renew the lock. Ensure agent is running, not sleeping, and has not lost communication with the service.
```

This error may indicate the agent lost communication with the server for a span of several minutes. Check the following to rule out network or other interruptions on the agent machine:

- Verify automatic updates are turned off. A machine reboot from an update will cause a build or release to fail with the above error. Apply updates in a controlled fashion to avoid this type of interruption. Before rebooting the agent machine, the agent should first be marked disabled in the pool administration page and let any running build finish.
- Verify the sleep settings are turned off.
- If the agent is running on a virtual machine, avoid any live migration or other VM maintenance operation that may severely impact the health of the machine for multiple minutes.
- If the agent is running on a virtual machine, the same operating-system-update recommendations and sleep-setting recommendations apply to the host machine. And also any other maintenance operations that severely impact the host machine.
- Performance monitor logging or other health metric logging can help to correlate this type of error to constrained resource availability on the agent machine (disk, memory, page file, processor, network).
- Another way to correlate the error with network problems is to ping a server indefinitely and dump the output to a file, along with timestamps. Use a healthy interval, for example 20 or 30 seconds. If you are using Azure Pipelines, then you would want to ping an internet domain, for example bing.com. If you are using an on-premises TFS server, then you would want to ping a server on the same network.
- Verify the network throughput of the machine is adequate. You can perform an online speed test to check the throughput.

- If you use a proxy, verify the agent is configured to use your proxy. Refer to the agent deployment topic.

Builds or releases not starting

TFS Job Agent not started

This may be characterized by a message in the web console "Waiting for an agent to be requested". Verify the TFSJobAgent (display name: *Visual Studio Team Foundation Background Job Agent*) Windows service is started.

Misconfigured notification URL (1.x agent version)

This may be characterized by a message in the web console "Waiting for console output from an agent", and the process eventually times out.

A mismatching notification URL may cause the worker to process to fail to connect to the server. See *Team Foundation Administration Console, Application Tier*. The 1.x agent listens to the message queue using the URL that it was configured with. However, when a job message is pulled from the queue, the worker process uses the notification URL to communicate back to the server.

Team Foundation Version Control (TFVC)

Get sources not downloading some files

This may be characterized by a message in the log "All files up to date" from the *tf get* command. Verify the built-in service identity has permission to download the sources. Either the identity *Project Collection Build Service* or *Project Build Service* will need permission to download the sources, depending on the selected authorization scope on General tab of the build pipeline. In the version control web UI, you can browse the project files at any level of the folder hierarchy and check the security settings.

Get sources through Team Foundation Proxy

The easiest way to configure the agent to get sources through a Team Foundation Proxy is set environment variable `TFSPROXY` that point to the TFVC proxy server for the agent's run as user.

```
Windows:  
set TFSPROXY=http://tfvcproxy:8081  
setx TFSPROXY=http://tfvcproxy:8081 // If the agent service is running as NETWORKSERVICE or any  
service account you can't easily set user level environment variable  
macOS/Linux:  
export TFSPROXY=http://tfvcproxy:8081
```

I need more help. I found a bug. I've got a suggestion. Where do I go?

[Get subscription, billing, and technical support](#)

Report any problems on [Developer Community](#).

We welcome your suggestions:

- Send feedback and report problems through the [Developer Community](#).

Default and custom release variables and debugging

11/19/2018 • 12 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

As you compose the tasks for deploying your application into each stage in your DevOps CI/CD processes, variables will help you to:

- Define a more generic deployment pipeline once, and then customize it easily for each stage. For example, a variable can be used to represent the connection string for web deployment, and the value of this variable can be changed from one stage to another. These are **custom variables**.
- Use information about the context of the particular release, [stage](#), [artifacts](#), or [agent](#) in which the deployment pipeline is being run. For example, your script may need access to the location of the build to download it, or to the working directory on the agent to create temporary files. These are **default variables**.

You can view the [current values of all variables](#) for a release, and use a default variable to [run a release in debug mode](#).

Custom variables

Custom variables can be defined at various scopes.

- Share values across all of the definitions in a project by using [variable groups](#). Choose a variable group when you need to use the same values across all the definitions, stages, and tasks in a project, and you want to be able to change the values in a single place. You define and manage variable groups in the **Library** tab.
- Share values across all of the stages by using [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. You define and manage these variables in the **Variables** tab in a release pipeline.
- Share values across all of the tasks within one specific stage by using [stage variables](#). Use an stage-level variable for values that vary from stage to stage (and are the same for all the tasks in an stage). You define and manage these variables in the **Variables** tab of an stage in a release pipeline.

Using custom variables at project, release pipeline, and stage scope helps you to:

- Avoid duplication of values, making it easier to update all occurrences as one operation.
- Store sensitive values in a way that they cannot be seen or changed by users of the release pipelines. Designate a configuration property to be a secure (secret) variable by selecting the  (padlock) icon next to the variable.

The values of hidden (secret) variables are stored securely on the server and cannot be viewed by users after they are saved. During a deployment, the Azure Pipelines release service decrypts these values when referenced by the tasks and passes them to the agent over a secure HTTPS channel.

Using custom variables

To use custom variables in your build and release tasks, simply enclose the variable name in parentheses and precede it with a \$ character. For example, if you have a variable named **adminUserName**, you can insert the current value of that variable into a parameter of a task as `$(adminUserName)`.

Note: At present, variables in different groups that are linked to a pipeline in the same scope (such as a release or stage scope) will collide and the result may be unpredictable. Ensure that you use different names for variables across all your variable groups.

You can use custom variables to prompt for values during the execution of a release. For more details, see [Approvals](#).

Define and modify your variables in a script

To define or modify a variable from a script, use the `task.setvariable` logging command. Note that the updated variable value is scoped to the job being executed, and does not flow across jobs or stages. Variable names are transformed to uppercase, and the characters "." and " " are replaced by "_".

For example, `Agent.WorkFolder` becomes `AGENT_WORKFOLDER`. On Windows, you access this as `%AGENT_WORKFOLDER` or `$env:AGENT_WORKFOLDER`. On Linux and macOS, you use `$AGENT_WORKFOLDER`.

TIP

You can run a script on a:

- Windows agent using either a [Batch script task](#) or [PowerShell script task](#).
- macOS or Linux agent using a [Shell script task](#).

- [Batch](#)
- [PowerShell](#)
- [Shell](#)

Batch script

 Set the `sauce` and `secret.Sauce` variables

```
@echo ##vso[task.setvariable variable=sauce]crushed tomatoes
@echo ##vso[task.setvariable variable=secret.Sauce;issecret=true]crushed tomatoes with garlic
```

 Read the variables

Arguments

```
"$(sauce)" "$(secret.Sauce)"
```

Script

```

@echo off
set sauceArgument=%~1
set secretSauceArgument=%~2
@echo No problem reading %sauceArgument% or %SAUCE%
@echo But I cannot read %SECRET_SAUCE%
@echo But I can read %secretSauceArgument% (but the log is redacted so I do not spoil
the secret)

```

Console output from reading the variables:

```

No problem reading crushed tomatoes or crushed tomatoes
But I cannot read
But I can read ***** (but the log is redacted so I do not spoil the secret)

```

Default variables

Information about the execution context is made available to running tasks through default variables. Your tasks and scripts can use these variables to find information about the system, release, stage, or agent they are running in. With the exception of **System.Debug**, these variables are read-only and their values are automatically set by the system. Some of the most significant variables are described in the following tables.

System variables

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
System.TeamFoundationServerUri	The URL of the service connection in TFS or Azure Pipelines. Use this from your scripts or tasks to call Azure Pipelines REST APIs.	https://fabrikam.vsrn.visualstudio.com/	
System.TeamFoundationCollectionUri	The URL of the Team Foundation collection or Azure Pipelines. Use this from your scripts or tasks to call REST APIs on other services such as Build and Version control.	https://dev.azure.com/fabrikam/	
System.CollectionId	The ID of the collection to which this build or release belongs.	6c6f3423-1c84-4625-995a-f7f143a1e43d	TFS 2015
System.TeamProject	The name of the project to which this build or release belongs.	Fabrikam	
System.TeamProjectId	The ID of the project to which this build or release belongs.	79f5c12e-3337-4151-be41-a268d2c73344	TFS 2015

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
System.ArtifactsDirectory	The directory to which artifacts are downloaded during deployment of a release. The directory is cleared before every deployment if it requires artifacts to be downloaded to the agent. Same as Agent.ReleaseDirectory and System.DefaultWorkingDirectory.	C:\agent_work\r1\a	
System.DefaultWorkingDirectory	The directory to which artifacts are downloaded during deployment of a release. The directory is cleared before every deployment if it requires artifacts to be downloaded to the agent. Same as Agent.ReleaseDirectory and System.ArtifactsDirectory.	C:\agent_work\r1\a	
System.WorkFolder	The working directory for this agent, where subfolders are created for every build or release. Same as Agent.RootDirectory and Agent.WorkFolder.	C:\agent_work	
System.Debug	This is the only system variable that can be <i>set</i> by the users. Set this to true to run the release in debug mode to assist in fault-finding.	true	

Release variables

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
Release.DefinitionName	The name of the release pipeline to which the current release belongs.	fabrikam-cd	
Release.DefinitionId	The ID of the release pipeline to which the current release belongs.	1	TFS 2015
Release.ReleaseName	The name of the current release.	Release-47	
Release.ReleaseId	The identifier of the current release record.	118	
Release.ReleaseUri	The URI of current release.	vstfs:///ReleaseManagement/Release/118	

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
Release.ReleaseDescription	The text description provided at the time of the release.	Critical security patch	
Release.RequestedFor	The display name of identity that triggered the release.	Mateo Escobedo	
Release.RequestedForEmail	The email address of identity that triggered the release.	mateo@fabrikam.com	
Release.RequestedForId	The ID of identity that triggered the release.	2f435d07-769f-4e46-849d-10d1ab9ba6ab	
Release.EnvironmentName	The name of stage to which deployment is currently in progress.	Dev	
Release.EnvironmentId	The ID of the stage instance in a release to which the deployment is currently in progress.	276	
Release.EnvironmentUri	The URI of the stage instance in a release to which deployment is currently in progress.	vstfs:///ReleaseManagement/Environment/276	
Release.DefinitionEnvironmentId	The ID of the stage in the corresponding release pipeline.	1	TFS 2015
Release.AttemptNumber	The number of times this release is deployed in this stage.	1	TFS 2015
Release.Deployment.RequestedFor	The display name of the identity that triggered (started) the deployment currently in progress.	Mateo Escobedo	TFS 2015
Release.Deployment.RequestedForId	The ID of the identity that triggered (started) the deployment currently in progress.	2f435d07-769f-4e46-849d-10d1ab9ba6ab	TFS 2015
Release.DeploymentID	The ID of the deployment. Unique per job.	254	
Release.DeployPhaseID	The ID of the phase where deployment is running.	127	
Release.Environments.{stage-name}.status	The deployment status of the stage.	InProgress	

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
Release.ReleaseWebURL	The URL for this release.	https://dev.azure.com/fabrikam/f3325c6c/_release?releaseId=392&_a=release-summary	
Release.SkipArtifactDownload	Boolean value that specifies whether or not to skip downloading of artifacts to the agent.	FALSE	
Release.TriggeringArtifact.Alias	The alias of the artifact which triggered the release. This is empty when the release was scheduled or triggered manually.	fabrikam_app	

Release stage variables

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
Release.Environments.{stage name}.Status	The status of deployment of this release within a specified stage.	NotStarted	TFS 2015

Agent variables

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
Agent.Name	The name of the agent as registered with the agent pool . This is likely to be different from the computer name.	fabrikam-agent	
Agent.MachineName	The name of the computer on which the agent is configured.	fabrikam-agent	
Agent.Version	The version of the agent software.	2.109.1	
Agent.JobName	The name of the job that is running, such as Release or Build.	Release	
Agent.HomeDirectory	The folder where the agent is installed. This folder contains the code and resources for the agent.	C:\agent	

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
Agent.ReleaseDirectory	The directory to which artifacts are downloaded during deployment of a release. The directory is cleared before every deployment if it requires artifacts to be downloaded to the agent. Same as System.ArtifactsDirectory and System.DefaultWorkingDirectory.	C:\agent_work\r1\a	
Agent.RootDirectory	The working directory for this agent, where subfolders are created for every build or release. Same as Agent.WorkFolder and System.WorkFolder.	C:\agent_work	
Agent.WorkFolder	The working directory for this agent, where subfolders are created for every build or release. Same as Agent.RootDirectory and System.WorkFolder.	C:\agent_work	
Agent.DeploymentGroupId	The ID of the deployment group the agent is registered with. This is available only in deployment group jobs.	1	TFS 2018 U1

General artifact variables

For each artifact that is referenced in a release, you can use the following artifact variables. Not all variables are meaningful for each artifact type. The table below lists the default artifact variables and provides examples of the values that they have depending on the artifact type. If an example is empty, it implies that the variable is not populated for that artifact type.

Replace **{alias}** with the value you specified for the [artifact alias](#), or with the default value generated for the release pipeline.

VARIABLE NAME	DESCRIPTION	AZURE PIPELINES EXAMPLE	JENKINS/TEAMCITY EXAMPLE	TFVC/GIT EXAMPLE	GITHUB EXAMPLE
Release.Artifacts.{alias}.DefinitionId	The identifier of the build pipeline or repository.	1			fabrikam/asp
Release.Artifacts.{alias}.DefinitionName	The name of the build pipeline or repository.	fabrikam-ci		TFVC: \$/fabrikam, Git: fabrikam	fabrikam/asp (master)

VARIABLE NAME	DESCRIPTION	AZURE PIPELINES EXAMPLE	JENKINS/TEAMCITY EXAMPLE	TFVC/GIT EXAMPLE	GITHUB EXAMPLE
Release.Artifacts.{alias}.BuildNumber	The build number or the commit identifier.	20170112.1	20170112.1	TFVC: Changeset 3, Git: 38629c964	38629c964
Release.Artifacts.{alias}.BuildId	The build identifier.	130	130		38629c964d21fe 405ef830b7d02 20966b82c9e11
Release.Artifacts.{alias}.BuildURI	The URL for the build.	vstfs:///build-release /Build/130			
Release.Artifacts.{alias}.SourceBranch	The full path and name of the branch from which the source was built.	refs/heads/master			
Release.Artifacts.{alias}.SourceBranchName	The name only of the branch from which the source was built.	master			
Release.Artifacts.{alias}.SourceVersion	The commit that was built.	bc0044458ba1d 9298 cdc649cb5dcf01 3180706f7			
Release.Artifacts.{alias}.Repository.Provider	The type of repository from which the source was built	Git			
Release.Artifacts.{alias}.RequestedForID	The identifier of the account that triggered the build.	2f435d07-769f-4e46-849d-10d1ab9ba6ab			
Release.Artifacts.{alias}.RequestedFor	The name of the account that requested the build.	Mateo Escobedo			
Release.Artifacts.{alias}.Type	The type of artifact source, such as Build.	Build	Jenkins: Jenkins, TeamCity:TeamCity	TFVC: TFVC, Git: Git	GitHub

See also [Artifact source alias](#)

Primary artifact variables

You designate one of the artifacts as a primary artifact in a release pipeline. For the designated primary artifact, Azure Pipelines populates the following variables.

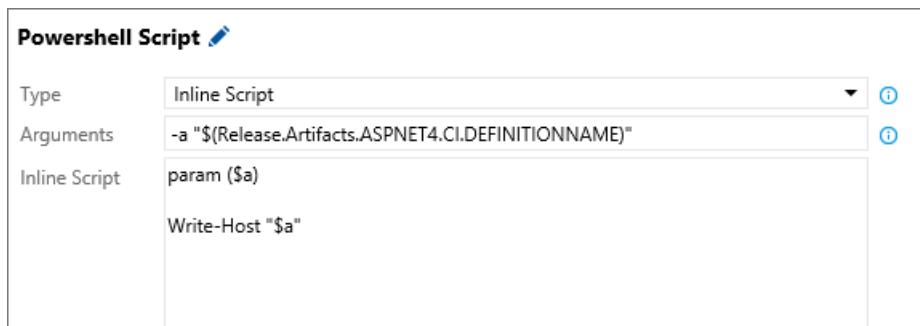
VARIABLE NAME	SAME AS
Build.DefinitionId	Release.Artifacts.{Primary artifact alias}.DefinitionId
Build.DefinitionName	Release.Artifacts.{Primary artifact alias}.DefinitionName
Build.BuildNumber	Release.Artifacts.{Primary artifact alias}.BuildNumber
Build.BuildId	Release.Artifacts.{Primary artifact alias}.BuildId
Build.BuildURI	Release.Artifacts.{Primary artifact alias}.BuildURI
Build.SourceBranch	Release.Artifacts.{Primary artifact alias}.SourceBranch
Build.SourceBranchName	Release.Artifacts.{Primary artifact alias}.SourceBranchName
Build.SourceVersion	Release.Artifacts.{Primary artifact alias}.SourceVersion
Build.Repository.Provider	Release.Artifacts.{Primary artifact alias}.Repository.Provider
Build.RequestedForID	Release.Artifacts.{Primary artifact alias}.RequestedForID
Build.RequestedFor	Release.Artifacts.{Primary artifact alias}.RequestedFor
Build.Type	Release.Artifacts.{Primary artifact alias}.Type

Using default variables

You can use the default variables in two ways - as parameters to tasks in a release pipeline or in your scripts.

You can directly use a default variable as an input to a task. For example, to pass

`Release.Artifacts.{Artifact alias}.DefinitionName` for the artifact source whose alias is **ASPNET4.CI** to a task, you would use `$(Release.Artifacts.ASPNET4.CI.DefinitionName)`.



To use a default variable in your script, you must first replace the `.` in the default variable names with `_`. For example, to print the value of artifact variable `Release.Artifacts.{Artifact alias}.DefinitionName` for the artifact source whose alias is **ASPNET4.CI** in a Powershell script, you would use

`$env:RELEASE_ARTIFACTS_ASPNET4_CI_DEFINITIONNAME`.

Powershell Script

Type	Inline Script
Arguments	
Inline Script	Write-Host "\$env:RELEASE_ARTIFACTS_ASPNET4_CI_DEFINITIONNAME"

Note that the original name of the artifact source alias, `ASPNET4.CI`, is replaced by `ASPNET4_CI`.

View the current values of all release variables

1. Open the pipelines view of the summary for the release, and choose the stage you are interested in. In the list of steps, choose **Initialize job**.

The screenshot shows the Azure DevOps pipeline summary for 'SampleApp - 1 > Release-2 > QA'. The 'Run on agent' step is highlighted with a red box around the 'Initialize job' sub-step, which is marked as 'succeeded'.

2. This opens the log for this step. Scroll down to see the values used by the agent for this job.

```

✓ Initialize job

1 2018-08-23T10:44:08.4457681Z ##[section]Starting: Initialize job
2 2018-08-23T10:44:08.4458248Z Current agent version: '2.139.0'
3 2018-08-23T10:44:08.4487538Z Prepare release directory.
4 2018-08-23T10:44:08.4501806Z ReleaseId=2, TeamProjectId=57633bf3-e6f1-4d73-8e33-89b718255fc
5 2018-08-23T10:44:08.4679318Z Release folder: D:\a\r1\a
6 2018-08-23T10:44:08.4837852Z Environment variables available are below. Note that these env
7   [AGENT_HOMEDIRECTORY] --> [C:\agents\2.139.0]
8   [AGENT_ID] --> [3]
9   [AGENT_JOBNAME] --> [Release]
10  [AGENT_MACHINENAME] --> [factoryvm-az24]
11  [AGENT_NAME] --> [Hosted Agent]
12  [AGENT_OS] --> [Windows_NT]
13  [AGENT_RELEASEDIRECTORY] --> [D:\a\r1\a]
14  [AGENT_ROOTDIRECTORY] --> [D:\a]
15  [AGENT_SERVEROMDIRECTORY] --> [C:\agents\2.139.0\externals\vstsom]
16  [AGENT_TEMPDIRECTORY] --> [D:\a\_temp]
17  [AGENT_TOOLS DIRECTORY] --> [C:/hostedtoolcache/windows]

```

Run a release in debug mode

Show additional information as a release executes and in the log files by running the entire release, or just the tasks in an individual release stage, in debug mode. This can help you resolve issues and failures.

- To initiate debug mode for an entire release, add a variable named `System.Debug` with the value `true` to the **Variables** tab of a release pipeline.
- To initiate debug mode for a single stage, open the **Configure stage** dialog from the shortcut menu of the stage and add a variable named `System.Debug` with the value `true` to the **Variables** tab.

- Alternatively, create a [variable group](#) containing a variable named `System.Debug` with the value `true` and link this variable group to a release pipeline.

If you get an error related to an Azure RM service connection, see [How to: Troubleshoot Azure Resource Manager service connections](#).

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

Troubleshoot Azure Resource Manager service connections

11/19/2018 • 4 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

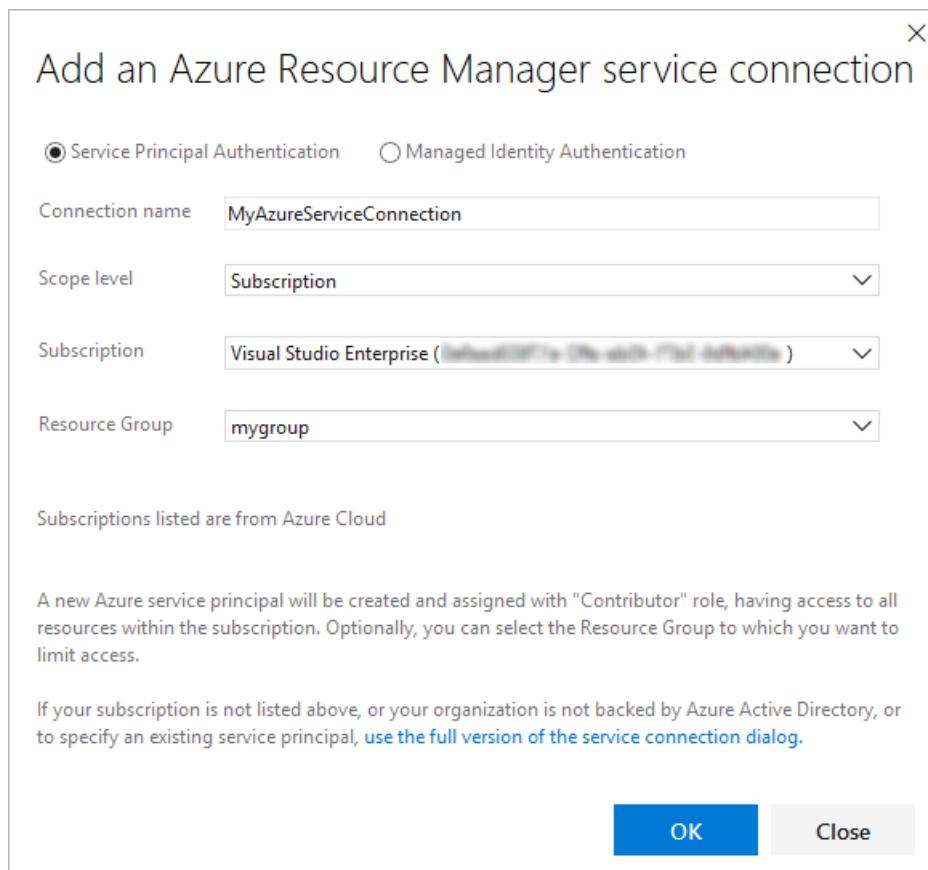
NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

This topic will help you resolve issues you may encounter when creating a connection to Microsoft Azure using an **Azure Resource Manager** service connection for your DevOps CI/CD processes.

What happens when you create a Resource Manager service connection?

You open the **Add Azure Resource Manager service connection** dialog, provide a connection name, and select a subscription from drop-down list of your subscriptions.



When you choose **OK**, the system:

1. Connects to the Azure Active Directory (Azure AD) tenant for the selected subscription
2. Creates an application in Azure AD on behalf of the user
3. After the application has been successfully created, assigns the application as a contributor to the selected subscription

- Creates an Azure Resource Manager service connection using this application's details

How to troubleshoot errors that may occur

Errors that may occur when the system attempts to create the service connection include:

- Insufficient privileges to complete the operation
- Failed to obtain an access token
- A valid refresh token was not found
- Failed to assign contributor role

Insufficient privileges to complete the operation

This typically occurs when the system attempts to create an application in Azure AD on your behalf.

This is a permission issue that may be due to the following causes:

- The user has only guest permission in the directory
- The user is not authorized to add applications in the directory

The user has only guest permission in the directory

The best approach to resolve this issue, while granting only the minimum additional permissions to the user, is to increase the Guest user permissions as follows.

- Sign into the Azure portal at <https://portal.azure.com> using an administrator account. The account should be an owner, global administrator, or user account administrator.
- Choose **Azure Active Directory** in the left navigation bar.
- Ensure you are editing the appropriate directory corresponding to the user subscription. If not, select **Switch directory** and log in using the appropriate credentials if required.
- In the **MANAGE** section choose **Users**.
- Choose **User settings**.
- In the **External users** section, choose **Manage external collaboration settings**.
- The **External collaboration settings** blade opens.
- Change **Guest user permissions are limited to No**.

Alternatively, if you are prepared to give the user additional (administrator-level) permissions, you can make the user a member of the **Global administrator** role as follows.

WARNING: Users with this role have access to all administrative features in Azure Active Directory, as well as services that use Azure Active Directory identities such as Exchange Online, SharePoint Online, and Skype for Business Online.

- Sign into the Azure portal at <https://portal.azure.com> using an administrator account. The account should be an owner, global administrator, or user account administrator.
- Choose **Azure Active Directory** in the left navigation bar.
- Ensure you are editing the appropriate directory corresponding to the user subscription. If not, select **Switch directory** and log in using the appropriate credentials if required.
- In the **MANAGE** section choose **Users**.
- Use the search box to filter the list and then choose the user you want to manage.

6. In the **MANAGE** section choose **Directory role** and change the role to **Global administrator**.

7. Save the change.

It typically takes 15 to 20 minutes to apply the changes globally. After this period has elapsed, the user can retry creating the service connection.

The user is not authorized to add applications in the directory

You must have permission to add integrated applications in the directory. The directory administrator has permission to change this setting, as follows:

1. Choose **Azure Active Directory** in the left navigation bar.
2. Ensure you are editing the appropriate directory corresponding to the user subscription. If not, select **Switch directory** and log in using the appropriate credentials if required.
3. In the **MANAGE** section choose **Users**.
4. Choose **User settings**.
5. In the **App registrations** section, change **Users can register applications** to **Yes**.

Failed to obtain an access token or A valid refresh token was not found

These errors typically occur when your session has expired.

To resolve these issues:

- Sign out of Azure Pipelines or TFS.
- Open an InPrivate or incognito browser window and navigate to <https://visualstudio.microsoft.com/team-services/>.
- If you are prompted to sign out, do so.
- Sign in using the appropriate credentials.
- Choose the organization you want to use from the list.
- Select the project you want to add the service connection to.
- Create the service connection you need by opening the **Settings** page, selecting the **Services** tab, choosing **New service connection**, and selecting **Azure Resource Manager**.

Failed to assign Contributor role

This error typically occurs when you do not have **Write** permission for the selected Azure subscription when the system attempts to assign the **Contributor** role.

To resolve this issue, ask the subscription administrator to [configure your identity in an Admin Access role](#).

Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

YAML schema reference

11/26/2018 • 14 minutes to read • [Edit Online](#)

Azure Pipelines

Here's a detailed reference guide to Azure Pipelines YAML pipelines, including a catalog of all supported YAML capabilities, and the available options.

The best way to get started with YAML pipelines is through the [quickstart guide](#). After that, to learn how to configure your YAML pipeline the way you need it to work, see conceptual topics such as [Build variables](#) and [Jobs](#).

Pipeline structure

Pipelines are made of one or more jobs and may include resources and variables. Jobs are made of one or more steps plus some job-specific data. Steps can be tasks, scripts, or references to external templates. This is reflected in the structure of the YAML file.

- Pipeline
 - Job 1
 - Step 1.1
 - Step 1.2
 - ...
 - Job 2
 - Step 2.1
 - Step 2.2
 - ...
 - ...

For simpler pipelines, not all of these levels are required. For example, in a single-job build, you can omit the container for "jobs" since there are only steps. Also, many options shown here are optional and have good defaults, so your YAML definitions are unlikely to include all of them.

Conventions

Conventions used in this topic:

- To the left of `:` are literal keywords used in pipeline definitions.
- To the right of `:` are data types. These can be primitives like **string** or references to rich structures defined elsewhere in this topic.
- `[datatype]` indicates an array of the mentioned data type. For instance, `[string]` is an array of strings.
- `{ datatype : datatype }` indicates a mapping of one data type to another. For instance, `{ string: string }` is a mapping of strings to strings.
- `|` indicates there are multiple data types available for the keyword. For instance, `job | templateReference` means either a job definition or a template reference are allowed.

YAML basics

This document covers the schema of an Azure Pipelines YAML file. To learn the basics of YAML, see [Learn YAML in Y Minutes](#). Note: Azure Pipelines doesn't support all features of YAML, such as complex keys and sets.

Pipeline

- [Schema](#)
- [Example](#)

```
name: string # build numbering format
resources:
  containers: [ container ]
  repositories: [ repository ]
variables: { string: string } | [ variable ]
trigger: trigger
pr: pr
jobs: [ job | templateReference ]
```

Learn more about [multi-job pipelines](#), using [containers](#) and [repositories](#) in pipelines, [triggers](#), [PR triggers](#), [variables](#), and [build number formats](#).

Variables

Hardcoded values can be added directly, or [variable groups](#) can be referenced.

- [Schema](#)
- [Example](#)

For a simple set of hardcoded variables:

```
variables: { string: string }
```

To include variable groups, switch to this list syntax:

```
variables:
  - name: string # name of a variable
    value: any # value of the variable
  - group: string # name of a variable group
```

`name` / `value` pairs and `group`s can be repeated.

Container

[Container jobs](#) let you isolate your tools and dependencies inside a container. The agent will launch an instance of your specified container, then run steps inside it. The `container` resource lets you specify your container images.

- [Schema](#)
- [Example](#)

```
resources:
  containers:
    - container: string # identifier (no spaces allowed)
      image: string # container image name
      options: string # arguments to pass to container at startup
      endpoint: string # endpoint for a private container registry
      env: { string: string } # list of environment variables to add
```

Repository

If your pipeline has [templates](#) in another repository, you must let the system know about that repository. The `repository` resource lets you specify an external repository.

- [Schema](#)

- [Example](#)

```
resources:  
  repositories:  
    - repository: string # identifier (no spaces allowed)  
      type: enum # see below  
      name: string # repository name (format depends on `type`)  
      ref: string # ref name to use, defaults to 'refs/heads/master'  
      endpoint: string # name of the service connection to use (for non-Azure Repos types)
```

Type

Pipelines support two types of repositories, `git` and `github`. `git` refers to Azure Repos Git repos. If you choose `git` as your type, then `name` refers to another repository in the same project. For example, `otherRepo`. To refer to a repo in another project within the same organization, prefix the name with that project's name. For example, `OtherProject/otherRepo`.

If you choose `github` as your type, then `name` is the full name of the GitHub repo including the user or organization. For example, `Microsoft/vscode`. Also, GitHub repos require a [service connection](#) for authorization.

Trigger

A trigger specifies what branches will cause a continuous integration build to run. If left unspecified, pushes to every branch will trigger a build. Learn more about [triggers](#) and how to specify them.

- [Schema](#)
- [Example](#)

List syntax:

```
trigger: [ string ] # list of branch names
```

Disable syntax:

```
trigger: none # will disable CI builds entirely
```

Full syntax:

```
trigger:  
  batch: boolean # batch changes if true, start a new build for every push if false  
  branches:  
    include: [ string ] # branch names which will trigger a build  
    exclude: [ string ] # branch names which will not  
  paths:  
    include: [ string ] # file paths which must match to trigger a build  
    exclude: [ string ] # file paths which will not trigger a build
```

PR trigger

A pull request trigger specifies what branches will cause a pull request build to run. If left unspecified, pull requests to every branch will trigger a build. Learn more about [pull request triggers](#) and how to specify them.

Note that `pr` is valid for GitHub, not any other Git provider.

- [Schema](#)
- [Example](#)

List syntax:

```
pr: [ string ] # list of branch names
```

Disable syntax:

```
pr: none # will disable PR builds entirely
```

Full syntax:

```
pr:  
branches:  
  include: [ string ] # branch names which will trigger a build  
  exclude: [ string ] # branch names which will not  
paths:  
  include: [ string ] # file paths which must match to trigger a build  
  exclude: [ string ] # file paths which will not trigger a build
```

Job

A [job](#) is a collection of steps to be run by an [agent](#) or, in some cases, on the server. Jobs can be run[conditionally](#), and they may [depend on earlier jobs](#).

- [Schema](#)
- [Example](#)

```
- job: string # name of the job, no spaces allowed  
  displayName: string # friendly name to display in the UI  
  dependsOn: string | [ string ]  
  condition: string  
  strategy:  
    matrix: # matrix strategy, see below  
    parallel: # parallel strategy, see below  
    maxParallel: number # maximum number of agents to simultaneously run copies of this job on  
    continueOnError: boolean # 'true' if future jobs should run even if this job fails; defaults to 'false'  
  pool: pool # see pool schema  
  workspace:  
    clean: outputs | resources | all # what to clean up after the job runs  
    container: string # container resource to run this job inside  
    timeoutInMinutes: number # how long to run the job before automatically cancelling  
    cancelTimeoutInMinutes: number # how much time to give 'run always even if cancelled tasks' before killing them  
  variables: { string: string } | [ variable ]  
  steps: [ script | bash | powershell | checkout | task | stepTemplate ]
```

Learn more about [variables](#). Also see the schema references for [pool](#), [server](#), [script](#), [bash](#), [powershell](#), [checkout](#), [task](#), and [step templates](#).

NOTE

If you have only one job, you can use [single-job syntax](#) which omits many of the keywords here.

Strategies

`matrix` and `parallel` are mutually-exclusive strategies for duplicating a job.

Matrix

Matrixing generates copies of a job with different inputs. This is useful for testing against different configurations or platform versions.

- [Schema](#)
- [Example](#)

```
strategy:  
  matrix: { string1: { string2: string3 } }
```

For each `string1` in the matrix, a copy of the job will be generated. `string1` is the copy's name and will be appended to the name of the job. For each `string2`, a variable called `string2` with the value `string3` will be available to the job.

Parallel

This specifies how many duplicates of the job should run. This is useful for slicing up a large test matrix. The [VS Test task](#) understands how to divide the test load across the number of jobs scheduled.

- [Schema](#)
- [Example](#)

```
strategy:  
  parallel: number
```

Maximum Parallelism

Regardless of which strategy is chosen and how many jobs are generated, this value specifies the maximum number of agents which will run at a time for this family of jobs. It defaults to unlimited if not specified.

- [Schema](#)
- [Example](#)

```
strategy:  
  maxParallel: number
```

Job templates

Jobs can also be specified in a job template. Job templates are separate files which you can reference in the main pipeline definition.

- [Schema](#)
- [Example](#)

In the main pipeline:

```
- template: string # name of template to include  
parameters: { string: any } # provided parameters
```

And in the included template:

```
parameters: { string: any } # expected parameters  
jobs: [ job ]
```

See [templates](#) for more about working with templates.

Pool

`pool` specifies which [pool](#) to use for a job of the pipeline. It also holds information about the job's strategy for running.

- [Schema](#)
- [Example](#)

Full syntax:

```
pool:  
  name: string # name of the pool to run this job in  
  demands: string | [ string ] ## see below  
  vmImage: string # name of the vm image you want to use, only valid in the Microsoft-hosted pool
```

If you're using a Microsoft-hosted pool, then choose an available `vmImage`.

If you're using a private pool and don't need to specify demands, this can be shortened to:

```
pool: string # name of the private pool to run this job in
```

Learn more about [conditions](#) and [timeouts](#).

Demands

`demands` is supported by private pools. You can check for existence of a capability or a specific string like this:

- [Schema](#)
- [Example](#)

```
pool:  
  demands: [ string ]
```

Server

`server` specifies a [server job](#).

- [Schema](#)
- [Example](#)

This will make the job run as a server job rather than an agent job.

```
pool: server
```

Script

`script` is a shortcut for the [command line task](#). It will run a script using cmd.exe on Windows and Bash on other platforms.

- [Schema](#)
- [Example](#)

```
- script: string # contents of the script to run
displayName: string # friendly name displayed in the UI
name: string # identifier for this step (no spaces allowed)
workingDirectory: string # initial working directory for the step
failOnStderr: boolean # if the script writes to stderr, should that be treated as the step failing?
condition: string
continueOnError: boolean # 'true' if future steps should run even if this step fails; defaults to 'false'
enabled: boolean # whether or not to run this step; defaults to 'true'
timeoutInMinutes: number
env: { string: string } # list of environment variables to add
```

Learn more about [conditions](#) and [timeouts](#).

Bash

`bash` is a shortcut for the [shell script task](#). It will run a script in Bash on Windows, macOS, or Linux.

- [Schema](#)
- [Example](#)

```
- bash: string # contents of the script to run
displayName: string # friendly name displayed in the UI
name: string # identifier for this step (no spaces allowed)
workingDirectory: string # initial working directory for the step
failOnStderr: boolean # if the script writes to stderr, should that be treated as the step failing?
condition: string
continueOnError: boolean # 'true' if future steps should run even if this step fails; defaults to 'false'
enabled: boolean # whether or not to run this step; defaults to 'true'
timeoutInMinutes: number
env: { string: string } # list of environment variables to add
```

Learn more about [conditions](#) and [timeouts](#).

PowerShell

`powershell` is a shortcut for the [PowerShell task](#). It will run a script in PowerShell on Windows.

- [Schema](#)
- [Example](#)

```
- powershell: string # contents of the script to run
displayName: string # friendly name displayed in the UI
name: string # identifier for this step (no spaces allowed)
errorActionPreference: enum # see below
ignoreLASTEXITCODE: boolean # see below
failOnStderr: boolean # if the script writes to stderr, should that be treated as the step failing?
workingDirectory: string # initial working directory for the step
condition: string
continueOnError: boolean # 'true' if future steps should run even if this step fails; defaults to 'false'
enabled: boolean # whether or not to run this step; defaults to 'true'
timeoutInMinutes: number
env: { string: string } # list of environment variables to add
```

Learn more about [conditions](#) and [timeouts](#).

Error action preference

Unless specified, the task defaults the error action preference to `stop`. The line `$ErrorActionPreference = 'stop'` is prepended to the top of your script.

When the error action preference is set to stop, errors will cause PowerShell to terminate and return a non-zero exit code. The task will also be marked as Failed.

- [Schema](#)
- [Example](#)

```
errorActionPreference: stop | continue | silentlyContinue
```

Ignore last exit code

By default, the last exit code returned from your script will be checked and, if non-zero, treated as a step failure. The system will prepend your script with:

```
if ((Test-Path -LiteralPath variable:\LASTEXITCODE)) { exit $LASTEXITCODE }
```

If you don't want this behavior, set `ignoreLASTEXITCODE` to `true`.

- [Schema](#)
- [Example](#)

```
ignoreLASTEXITCODE: boolean
```

Checkout

`checkout` informs the system how to handle checking out source code.

- [Schema](#)
- [Example](#)

```
- checkout: self # self represents the repo where the initial Pipelines YAML file was found
  clean: boolean # whether to fetch clean each time
  fetchDepth: number # the depth of commits to ask Git to fetch
  lfs: boolean # whether to download Git-LFS files
  submodules: true | recursive # set to 'true' for a single level of submodules or 'recursive' to get
    submodules of submodules
  persistCredentials: boolean # set to 'true' to leave the OAuth token in the Git config after the initial
    fetch
```

Or to avoid syncing sources at all:

```
- checkout: none
```

Task

[Tasks](#) are the building blocks of a pipeline. There is a [catalog of tasks](#) available to choose from.

- [Schema](#)
- [Example](#)

```
- task: string # reference to a task and version, e.g. "VSBUILD@1"
displayName: string # friendly name displayed in the UI
name: string # identifier for this step (no spaces allowed)
condition: string
continueOnError: boolean # 'true' if future steps should run even if this step fails; defaults to 'false'
enabled: boolean # whether or not to run this step; defaults to 'true'
timeoutInMinutes: number
inputs: { string: string } # task-specific inputs
env: { string: string } # list of environment variables to add
```

Step template

A set of steps can be defined in one file and used multiple places in another file.

- [Schema](#)
- [Example](#)

In the main pipeline:

```
steps:
- template: string # reference to template
parameters: { string: any } # provided parameters
```

And in the included template:

```
parameters: { string: any } # expected parameters
steps: [ script | bash | powershell | checkout | task ]
```

See [templates](#) for more about working with templates.

Build and release tasks

11/26/2018 • 11 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#) | [Previous versions \(XAML builds\)](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Build

TASK	VERSIONS
 .NET Core Standard projects and create .NET Core and .NET Standard NuGet packages using the <code>dotnet</code> command-line tool.	Azure Pipelines, TFS 2018
 .NET Core CLI dotnet application, or run a custom dotnet command.	Azure Pipelines, TFS 2018
 Android Build - deprecated. Use Gradle	Azure Pipelines, TFS 2015 RTM and newer
 Android Signing . Sign and align Android APK files	Azure Pipelines, TFS 2015 RTM and newer
 Ant . Build with Apache Ant	Azure Pipelines, TFS 2015 RTM and newer
 CMake . Build with the CMake cross-platform build system	Azure Pipelines, TFS 2015 RTM and newer
 Docker . Build, tag, push, or run Docker images, or run a Docker command.	Azure Pipelines
 Docker Compose . Build, push or run multi-container Docker applications.	Azure Pipelines
 Go . Get, build, test a go application, or run a custom go command.	Azure Pipelines
 Gradle . Build using a Gradle wrapper script	Azure Pipelines, TFS 2015 RTM and newer
 Grunt . The JavaScript Task Runner	Azure Pipelines, TFS 2015.3 and newer

TASK	VERSIONS
 gulp . Node.js streaming task based build system	Azure Pipelines, TFS 2015 RTM and newer
 Index Sources & Publish Symbols . Index your source code and publish symbols to a file share	Azure Pipelines, TFS 2015 RTM and newer
 Jenkins Queue Job . Queue a job on a Jenkins server	Azure Pipelines, TFS 2017 RTM and newer
 Maven . Build with Apache Maven	Azure Pipelines, TFS 2015 RTM and newer
 MSBuild . Build with MSBuild	Azure Pipelines, TFS 2015 RTM and newer
 Publish Build Artifacts . Publish Build artifacts to the server or a file share	TFS 2015 RTM. Deprecated on Azure Pipelines and newer versions of TFS.
 Publish Pipeline Artifact . Publish pipeline artifact	Azure Pipelines
SonarQube - Begin Analysis . Fetch the Quality Profile from SonarQube to configure the analysis	Azure Pipelines, TFS 2015.3 and newer
SonarQube - End Analysis . Finish the analysis and upload the results to SonarQube	Azure Pipelines, TFS 2015.3 and newer
 Visual Studio Build Visual Studio version property	Azure Pipelines, TFS 2015 RTM and newer
 Xamarin.Android . Build an Android app with Xamarin	Azure Pipelines, TFS 2015 RTM and newer
 Xamarin.iOS macOS	Azure Pipelines, TFS 2015 RTM and newer
 Xcode . Build an Xcode workspace on macOS	Azure Pipelines, TFS 2015 RTM and newer
 Xcode Package iOS build output	Azure Pipelines, TFS 2015 RTM and newer

Utility

TASK	VERSIONS
 Archive Files . Archive files using a variety of compression formats such as .7z, .rar, .tar.gz, and .zip.	Azure Pipelines, TFS 2017 and newer

TASK	VERSIONS
 Azure Network Load Balancer Connect/Disconnect an Azure virtual machine's network interface to a Load Balancer's backend address pool	Azure Pipelines
 Bash . Run a Bash script on macOS, Linux, or Windows	Azure Pipelines
 Batch Script . Run a windows cmd or bat script and optionally allow it to change the stage	Azure Pipelines, TFS 2015 RTM and newer
 Command Line . Run a command line with arguments	Azure Pipelines, TFS 2015 RTM and newer
 Copy and Publish Build Artifacts . Copy Build artifacts to staging folder then publish Build artifacts to the server or a file share	TFS 2015 RTM. Deprecated on Azure Pipelines and newer versions of TFS.
 Copy Files . Copy files from source folder to target folder using minimatch patterns (The minimatch patterns will only match file paths, not folder paths).	Azure Pipelines, TFS 2015.3 and newer
 cURL Upload Files . Use cURL to upload files with supported protocols. (FTP, FTPS, SFTP, HTTP, and more)	Azure Pipelines, TFS 2015 RTM and newer
 Decrypt File . A thin utility task for file decryption using OpenSSL.	Azure Pipelines
 Delay . Pause execution of the pipeline for a fixed delay time.	Azure Pipelines
 Delete Files . Delete files or folders.	Azure Pipelines, TFS 2015.3 and newer
 Download Build Artifacts . Download build artifacts.	Azure Pipelines
 Download Fileshare Artifacts . Download fileshare artifacts.	Azure Pipelines
 Download Pipeline Artifact . Download pipeline artifacts.	Azure Pipelines
 Download Package . Download a package from a Package Management feed in Azure Pipelines or TFS. Requires the Package Management extension.	Azure Pipelines

TASK	VERSIONS
 Download Secure File Download a secure file to a temporary location on the build or release agent.	Azure Pipelines
 Extract Files . Extract files from archives (.zip, .jar, .war, .ear, .tar, .7z, and others) to a target folder.	Azure Pipelines, TFS 2017 and newer
 FTP Upload . Upload files to a remote machine using the File Transfer Protocol (FTP), or securely with FTPS.	Azure Pipelines, TFS 2017 and newer
 GitHub Release . Create, edit, or discard a GitHub release.	Azure Pipelines
 Install Apple Certificate . Install an Apple certificate required to build on a macOS agent.	Azure Pipelines, TFS 2018
 Install Apple Provisioning Profile . Install an Apple provisioning profile required to build on a macOS agent.	Azure Pipelines, TFS 2018
 Install SSH Key . Install an SSH key prior to a build or release	Azure Pipelines
 Invoke Azure Function . Invoke a HTTP triggered function in an Azure function app and parse the response.	Azure Pipelines
 Invoke REST API . Invoke an HTTP API and parse the response.	Azure Pipelines
 Jenkins Download Artifacts . Download artifacts produced by a Jenkins job	Azure Pipelines
 Manual Intervention . Pause an active deployment within a stage, typically to perform some manual steps or actions, and then continue the automated deployment tasks.	Azure Pipelines
 PowerShell . Run a PowerShell script	Azure Pipelines, TFS 2015 RTM and newer
 Publish Build Artifacts . Publish Build artifacts to the server or a file share	Azure Pipelines, TFS 2015.3 and newer
 Publish Pipeline Artifact . Publish pipeline artifact	Azure Pipelines

TASK	VERSIONS
 Publish To Azure Service Bus . Send a message to an Azure Service Bus using a service connection and without using an agent.	Azure Pipelines
 Python Script . Run a Python script.	Azure Pipelines
 Query Azure Monitor Alerts . Observe the configured Azure monitor rules for active alerts.	Azure Pipelines
 Query Work Items . Ensure the number of matching items returned by a work item query is within the configured thresholds.	Azure Pipelines
 Service Fabric PowerShell . Runs any PowerShell command or script in a PowerShell session that has a Service Fabric cluster connection initialized.	Azure Pipelines
 Shell Script . Run a shell script using bash	Azure Pipelines, TFS 2015 RTM and newer
 Update Service Fabric App Versions . Automatically updates the versions of a packaged Service Fabric application	Azure Pipelines, TFS 2017 and newer

Test

TASK	VERSIONS
 App Center Test . Test mobile app packages with Visual Studio App Center	Azure Pipelines, TFS 2015.3 and newer
 Cloud-based Apache JMeter Load Test . Runs the Apache JMeter load test in cloud	Azure Pipelines, TFS 2015 RTM and newer
 Cloud-based Load Test . Runs the load test in cloud, with Azure Pipelines	Azure Pipelines, TFS 2015 RTM and newer
 Cloud-based Web Performance Test . Runs the quick web performance test in cloud, with Azure Pipelines	Azure Pipelines, TFS 2015 RTM and newer
 Publish Code Coverage Results . Publish code coverage results to Azure Pipelines/TFS	Azure Pipelines, TFS 2015.3 and newer
 Publish Test Results . Publish Test Results to Azure Pipelines/TFS	Azure Pipelines, TFS 2015 RTM and newer

TASK	VERSIONS
 Run Functional Tests UI/Selenium/Functional tests on a set of machines (using Test Agent)	Azure Pipelines, TFS 2015.3 and newer
 Visual Studio Test version 2 Visual Studio Test version 1 Run unit and functional tests (Selenium, Appium, Coded UI test, etc.) using the Visual Studio Test runner. Test frameworks that have a Visual Studio test adapter such as xUnit, NUnit, Chutzpah, etc. can also be run.	Azure Pipelines, TFS 2015 RTM and newer
 Visual Studio Test Agent Deployment configure Test Agent to run tests on a lab machine group	Azure Pipelines, TFS 2015 RTM and newer

Package

TASK	VERSIONS
 CocoaPods . CocoaPods is the dependency manager for Swift and Objective-C Cocoa projects. Runs pod install	Azure Pipelines, TFS 2015 RTM and newer
 Conda Environment . Create and activate a Conda environment	Azure Pipelines
 npm . Install npm packages	Azure Pipelines, TFS 2015 RTM and newer
 npm Authenticate . Don't use this task if you're also using the npm task. Provides npm credentials to an .npmrc file in your repository for the scope of the build.	Azure Pipelines, TFS 2015 RTM and newer
 NuGet Installer . Installs and updates missing NuGet packages	Azure Pipelines, TFS 2015 RTM and newer
 NuGet Packager . Creates nupkg outputs from csproj or nuspec files	Azure Pipelines, TFS 2015.3 and newer
 NuGet Publisher . Uploads nupkg files to a nuget server	Azure Pipelines, TFS 2015.3 and newer
 PyPI Publisher . Publish a Python package to PyPI	Azure Pipelines
 Xamarin Component Restore - deprecated. See Updating component references to NuGet	Azure Pipelines, TFS 2015 RTM and newer

Deploy

TASK	VERSIONS
 App Center Distribute . Distribute app builds to testers and users via App Center	Azure Pipelines, TFS 2017 and newer
 Azure App Service Deploy . Update Azure App Service using Web Deploy / Kudu REST APIs	Azure Pipelines, TFS 2017 and newer
 Azure App Service Manage . Start, Stop, Restart or Slot swap for an Azure App Service	Azure Pipelines, TFS 2017 and newer
 Azure CLI . Run a shell or batch script containing Azure CLI commands against an Azure subscription	Azure Pipelines, TFS 2017 and newer
 Azure Cloud PowerShell Deployment . Deploy an Azure Cloud Service	Azure Pipelines, TFS 2017 and newer
 Azure File Copy . Copy files to Azure blob or VM(s)	Azure Pipelines, TFS 2015.3 and newer
 Azure Key Vault . Incorporate secrets from an Azure Key Vault into a release pipeline	Azure Pipelines
 Azure Monitor Alerts . Configure alerts on available metrics for an Azure resource	Azure Pipelines
 Azure MySQL Deployment . Run your scripts and make changes to your Azure DB for MySQL.	Azure Pipelines
 Azure Policy Check Gate assessment with Azure policies on resources that belong to the resource group and Azure subscription.	Azure Pipelines
 Azure PowerShell . Run a PowerShell script within an Azure environment	Azure Pipelines, TFS 2015 RTM and newer
 Azure Resource Group Deployment . Deploy, start, stop, delete Azure Resource Groups	Azure Pipelines, TFS 2015.3 and newer
 Azure SQL Database Deployment . Deploy an Azure SQL database using DACPAC or run scripts using SQLCMD	Azure Pipelines, TFS 2015.3 and newer
 Azure VM Scale Set Deployment . Deploy a virtual machine scale set image.	Azure Pipelines

TASK	VERSIONS
 Build Machine Image (Packer) using Packer.	Azure Pipelines
 . Deploy to Chef environments by editing environment attributes	Azure Pipelines
 . Run Scripts with knife commands on your chef workstation	Azure Pipelines
 Copy Files Over SSH target folder on a remote machine over SSH	Azure Pipelines, TFS 2017 and newer
 . Build, tag, push, or run Docker images, or run a Docker command. Task can be used with Docker or Azure Container registry	Azure Pipelines, TFS 2017 and newer
 . Build, push or run multi-container Docker applications.	Azure Pipelines
 Helm Deploy Kubernetes cluster in Azure Container Service by running helm commands.	Azure Pipelines
 IIS Web App Deploy a machine group using WebDeploy	Azure Pipelines
 IIS Web App Manage web app, virtual directory, or application pool on a machine group	Azure Pipelines
 . Deploy, configure, update your Kubernetes cluster in Azure Container Service by running kubectl commands.	Azure Pipelines
 PowerShell on Target Machines scripts on remote machine(s)	Azure Pipelines, TFS 2015 RTM and newer
 Service Fabric Application Deployment Service Fabric application to a cluster	Azure Pipelines, TFS 2017 and newer
 Service Fabric Compose Deploy Fabric application to a cluster using a compose file	Azure Pipelines

TASK	VERSIONS
 SSH . Run shell commands or a script on a remote machine using SSH	Azure Pipelines, TFS 2017 and newer
 Windows Machine File Copy . Copy files to remote machine(s)	Azure Pipelines, TFS 2015 RTM and newer
 WinRM SQL Server DB Deployment . Deploy a SQL Server database using DACPAC or SQL scripts	Azure Pipelines

Tool

TASK	VERSIONS
 .NET Core Tool Installer . Acquires a specific version of .NET Core and adds it to the PATH. Use the task to change the Core version for subsequent tasks.	Azure Pipelines, TFS 2018
 Go Tool Installer . Finds or downloads a specific version of the Go tool into the tools cache and adds it to the PATH	Azure Pipelines
 Helm Tool Installer . Install Helm and Kubernetes on agent machine.	Azure Pipelines
 Java Tool Installer . Acquires a specific version of Java from a user supplied Azure blob, a location in the source or on the agent, or the tools cache and sets JAVA_HOME. Use this task to change the version of Java used in Java tasks.	Azure Pipelines
 Node.js Tool Installer . Finds or downloads and caches the specified version of Node.js and adds it to the PATH	Azure Pipelines
 NuGet Tool Installer . Finds or downloads and caches the specified version of NuGet and adds it to the PATH	Azure Pipelines
 Use Python Version . Selects a version of Python to run on an agent. Optionally adds it to PATH.	Azure Pipelines
 Use Ruby Version . Selects a version of Ruby to run on an agent. Optionally adds it to PATH.	Azure Pipelines
 Visual Studio Test Platform Installer . Acquires the Visual Studio Test Platform from nuget.org or the tools cache.	Azure Pipelines

To learn more about tool installer tasks, see [Tool installers](#).

Open source

These tasks are open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Where can I learn step-by-step how to build my app?

[Build your app](#)

Can I add my own build tasks?

Yes: [Add a build task](#)

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

.NET Core task

11/19/2018 • 6 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

Use this task in a build or release pipeline to build, test, and release .NET Core and .NET Standard projects and create .NET Core and .NET Standard NuGet packages using the `dotnet` command-line tool.

If your .NET Core or .NET Standard build depends on NuGet packages, make sure to add two copies of this step: one with the `restore` command and one with the `build` command.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

Restore NuGet packages

Demand

None

YAML snippet

```
# .NET Core
# Restore NuGet packages.
- task: DotNetCoreCLI@2
  inputs:
    command: 'restore'
    projects: '**/*.csproj'
    #verbosityRestore: 'detailed' # Options: quiet, minimal, normal, detailed, diagnostic
```

Arguments

ARGUMENT	DESCRIPTION
Command	(Required) The dotnet command to run. Select 'Custom' to add arguments or use a command not listed here.
Path to project(s)	(Optional) The path to the csproj file(s) to use. You can use wildcards (e.g. <code>./csproj</code> for all <code>.csproj</code> files in all subfolders).
Verbosity	(Required) Specifies the amount of detail displayed in the output.
CONTROL OPTIONS	

Pack NuGet packages

Demand

None

YAML snippet

```

# .NET Core
# Pack NuGet packages.
- task: DotNetCoreCLI@2
  inputs:
    command: 'pack'
    configuration: 'release'
    #packagesToPack: '**/*.csproj' # Required when command == pack
    #packDirectory: '$(build.artifactStagingDirectory)' # Optional
    #nobuild: false # Optional
    #versioningScheme: 'off' # Options: off, byPrereleaseNumber, byEnvVar, byBuildNumber
    #versionEnvVar: # Required when versioningScheme == byEnvVar
    #majorVersion: '1' # Required when versioningScheme == byPrereleaseNumber
    #minorVersion: '0' # Required when versioningScheme == byPrereleaseNumber
    #patchVersion: '0' # Required when versioningScheme == byPrereleaseNumber
    #buildProperties: # Optional
    #verbosityPack: 'detailed' # Options: quiet, minimal, normal, detailed, diagnostic

```

Arguments

ARGUMENT	DESCRIPTION
Command	(Required) The dotnet command to run. Select 'Custom' to add arguments or use a command not listed here.
Configuration to Package	(Optional) When using a csproj file this specifies the configuration to package
Package Folder	(Optional) Folder where packages will be created. If empty, packages will be created alongside the csproj file.
Do not build	(Optional) Don't build the project before packing. Corresponds to the --no-build command line parameter.
Automatic package versioning	(Required) Cannot be used with include referenced projects. If you choose 'Use the date and time', this will generate a SemVer -compliant version formatted as <code>X.Y.Z-ci-datetime</code> where you choose X, Y, and Z. If you choose 'Use an environment variable', you must select an environment variable and ensure it contains the version number you want to use. If you choose 'Use the build number', this will use the build number to version your package. Note: Under Options set the build number format to be <code>'\$(BuildDefinitionName)_\$(Year:yyyy).\$(Month).\$(DayOfMonth:h)\$(Rev:r)'</code> .
Environment variable	(Required) Enter the variable name without \$, \$env, or %.
Major	(Required) The 'X' in version X.Y.Z
Minor	(Required) The 'Y' in version X.Y.Z
Patch	(Required) The 'Z' in version X.Y.Z
Additional build properties	(Optional) Specifies a list of token = value pairs, separated by semicolons, where each occurrence of \${token\$} in the .nuspec file will be replaced with the given value. Values can be strings in quotation marks.

ARGUMENT	DESCRIPTION
Verbosity	(Required) Specifies the amount of detail displayed in the output.
CONTROL OPTIONS	

Push NuGet packages

Demands

None

YAML snippet

```
# .NET Core
# Push NuGet packages.
- task: DotNetCoreCLI@2
  inputs:
    command: 'push'
    #nuGetFeedType: 'internal' # Required when command == push. Options: internal, external
    #packagesToPush: '$(build.artifactStagingDirectory)/*.nupkg' # Required when command == push
    #publishVstsFeed: # Required when command == push && NuGetFeedType == internal
    #publishFeedCredentials: # Required when command == push && NuGetFeedType == external
```

Arguments

ARGUMENT	DESCRIPTION
Command	(Required) The dotnet command to run. Select 'Custom' to add arguments or use a command not listed here.
Target feed location	(Required) Use 'internal' for this organization/collection. Use 'external' for an external NuGet server (including other accounts/collections).
Path to NuGet package(s) to publish	(Required) The pattern to match or path to nupkg files to be uploaded. Multiple patterns can be separated by a semicolon, and you can make a pattern negative by prefixing it with '-'. Example: <code>***.nupkg;-:***.Tests.nupkg</code>
Target feed	(Required) Select a feed hosted in this organization. You must have Package Management installed and licensed to select a feed here.
NuGet server	(Required) The NuGet service connection that contains the external NuGet server's credentials.
CONTROL OPTIONS	

Custom NuGet command

YAML snippet

```

# .NET Core
# Custom NuGet command.
- task: DotNetCoreCLI@2
  inputs:
    command: custom
    projects: '**/*.csproj'
    custom: 'Enter your custom NuGet command here'
    arguments: '--configuration release --output $(build.artifactStagingDirectory)'

```

Arguments

ARGUMENT	DESCRIPTION
Command	(Required) The dotnet command to run. Select 'Custom' to add arguments or use a command not listed here.
Path to project(s)	(Optional) The path to the csproj file(s) to use. You can use wildcards (e.g. */.csproj for all .csproj files in all subfolders).
Custom command	(Required) The command to pass to dotnet.exe for execution.
Arguments	(Optional) Arguments to the selected command. For example, build configuration, output folder, runtime. The arguments depend on the command selected.

CONTROL OPTIONS

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Why is my build or publish step failing to restore packages?

Most `dotnet` commands, including `build` and `publish`, include an implicit `restore` step. This will fail against authenticated feeds, even if you ran a successful `dotnet restore` in an earlier step, because the earlier step will have cleaned up the credentials it used.

To fix this issue, add the `--no-restore` flag to the Arguments textbox.

Why should I check in a NuGet.Config?

Checking a NuGet.Config into source control ensures that a key piece of information needed to build your project—the location of its packages—is available to every developer that checks out your code.

However, for situations where a team of developers works on a large range of projects, it's also possible to add an Azure Artifacts feed to the global NuGet.Config on each developer's machine. In these situations, using the "Feeds I select here" option in the NuGet task replicates this configuration.

Where can I learn about Azure Artifacts?

[Package Management in Azure Artifacts and TFS](#)

Where can I learn more about NuGet?

[NuGet Docs](#) Overview

[NuGet Create](#) Packaging and publishing

[NuGet Consume](#) Seting up a solution to get dependencies

What other kinds of apps can I build?

[Build your app](#)

What other kinds of build tasks are available?

[Specify your build tasks](#)

How do we protect our codebase from build breaks?

- Git: [Improve code quality with branch policies](#) with an option to require that code builds before it can be merged to a branch. For GitHub repositories, similar policies are available in GitHub's repository settings under *Branches*.
- TFVC: [Use gated check-in](#).

How do I modify other parts of my build pipeline?

- [Specify your build tasks](#) to run tests, scripts, and a wide range of other processes.
- [Specify build options](#) such as specifying how completed builds are named, building multiple configurations, creating work items on failure.
- [Specify the repository](#) to pick the source of the build and modify options such as how the agent workspace is cleaned.
- [Set build triggers](#) to modify how your CI builds run and to specify scheduled builds.
- [Specify build retention policies](#) to automatically delete old builds.

I selected parallel multi-configuration, but only one build is running at a time.

If you're using Azure Pipelines, you might need more parallel jobs. See [Parallel jobs in Azure Pipelines](#).

How do I see what has changed in my build pipeline?

[View the change history of your build pipeline](#)

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

.NET Core CLI task

11/6/2018 • 5 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to build, test, package, or publish a dotnet application, or run a custom dotnet command. For package commands, supports NuGet.org and authenticated feeds like Package Management and MyGet.

YAML snippet

```
# .NET Core
# Build, test, package, or publish a dotnet application, or run a custom dotnet command. For package commands,
# supports NuGet.org and authenticated feeds like Package Management and MyGet.
- task: DotNetCoreCLI@2
  inputs:
    #command: 'build' # Options: build, push, pack, publish, restore, run, test, custom
    #publishWebProjects: true # Required when command == Publish
    #projects: # Optional
    #custom: # Required when command == Custom
    #arguments: # Optional
    #publishTestResults: true # Optional
    #zipAfterPublish: true # Optional
    #modifyOutputPath: true # Optional
    #feedsToUse: 'select' # Options: select, config
    #vstsFeed: # Required when feedsToUse == Select
    #includeNuGetOrg: true # Required when feedsToUse == Select
    #nugetConfigPath: # Required when feedsToUse == Config
    #externalFeedCredentials: # Optional
    #noCache: false
    restoreDirectory:
      #verbosityRestore: 'Detailed' # Options: -, quiet, minimal, normal, detailed, diagnostic
      #packagesToPush: '$(Build.ArtifactStagingDirectory)/*.nupkg' # Required when command == Push
      #nuGetFeedType: 'internal' # Required when command == Push# Options: internal, external
      #publishVstsFeed: # Required when command == Push && NuGetFeedType == Internal
      #publishFeedCredentials: # Required when command == Push && NuGetFeedType == External
      #packagesToPack: '**/*.csproj' # Required when command == Pack
      #configuration: '$(BuildConfiguration)' # Optional
      #packDirectory: '$(Build.ArtifactStagingDirectory)' # Optional
      #nobuild: false # Optional
      #versioningScheme: 'off' # Options: off, byPrereleaseNumber, byEnvVar, byBuildNumber
      #versionEnvVar: # Required when versioningScheme == ByEnvVar
      #majorVersion: '1' # Required when versioningScheme == ByPrereleaseNumber
      #minorVersion: '0' # Required when versioningScheme == ByPrereleaseNumber
      #patchVersion: '0' # Required when versioningScheme == ByPrereleaseNumber
      #buildProperties: # Optional
      #verbosityPack: 'Detailed' # Options: -, quiet, minimal, normal, detailed, diagnostic
    workingDirectory:
```

Arguments

ARGUMENT	DESCRIPTION
Command	(Required) The dotnet command to run. Select 'Custom' to add arguments or use a command not listed here.

ARGUMENT	DESCRIPTION
Publish Web Projects	(Required) If true, the task will try to find the web projects in the repository and run the publish command on them. Web projects are identified by presence of either a web.config file or wwwroot folder in the directory.
Path to project(s)	(Optional) The path to the csproj file(s) to use. You can use wildcards (e.g. **/*.csproj for all .csproj files in all subfolders).
Custom command	(Required) The command to pass to dotnet.exe for execution.
Arguments	(Optional) Arguments to the selected command. For example, build configuration, output folder, runtime. The arguments depend on the command selected.
Publish test results	(Optional) Enabling this option will generate a test results TRX file in <code>\$(Agent.TempDirectory)</code> and results will be published to the server. This option appends <code>--logger trx --results-directory \$(Agent.TempDirectory)</code> to the command line arguments.
Zip Published Projects	(Optional) If true, folder created by the publish command will be zipped.
Add project name to publish path	(Optional) If true, folders created by the publish command will have project file name prefixed to their folder names when output path is specified explicitly in arguments. This is useful if you want to publish multiple projects to the same folder.
Feeds to use	(Required) You can either select a feed from Azure Artifacts and/or NuGet.org here, or commit a nuget.config file to your source code repository and set its path here.
Use packages from this Azure Artifacts/TFS feed	(Required) Include the selected feed in the generated NuGet.config. You must have Package Management installed and licensed to select a feed here.
Use packages from NuGet.org	(Required) Include NuGet.org in the generated NuGet.config.
Path to NuGet.config	(Required) The NuGet.config in your repository that specifies the feeds from which to restore packages.
Credentials for feeds outside this organization/collection	(Optional) Credentials to use for external registries located in the selected NuGet.config. For feeds in this organization/collection, leave this blank; the build's credentials are used automatically.
Disable local cache	(Required) Prevents NuGet from using packages from local machine caches.
Destination directory	(Required) Specifies the folder in which packages are installed. If no folder is specified, packages are restored into the default NuGet package cache.

ARGUMENT	DESCRIPTION
Verbosity	(Required) Specifies the amount of detail displayed in the output.
Path to NuGet package(s) to publish	(Required) The pattern to match or path to nupkg files to be uploaded. Multiple patterns can be separated by a semicolon, and you can make a pattern negative by prefixing it with '-:'. Example: **/*.nupkg;:-**/*.Tests.nupkg
Target feed location	(Required) undefined
Target feed	(Required) Select a feed hosted in this organization. You must have Package Management installed and licensed to select a feed here.
NuGet server	(Required) The NuGet service connection that contains the external NuGet server's credentials.
Path to csproj or nuspec file(s) to pack	(Required) Pattern to search for csproj or nuspec files to pack. You can separate multiple patterns with a semicolon, and you can make a pattern negative by prefixing it with '-:'. Example: **/*.csproj;:-**/*.Tests.csproj
Configuration to Package	(Optional) When using a csproj file this specifies the configuration to package
Package Folder	(Optional) Folder where packages will be created. If empty, packages will be created alongside the csproj file.
Do not build	(Optional) Don't build the project before packing. Corresponds to the --no-build command line parameter.
Automatic package versioning	(Required) Cannot be used with include referenced projects. If you choose 'Use the date and time', this will generate a SemVer-compliant version formatted as X.Y.Z-ci-datetime where you choose X, Y, and Z. If you choose 'Use an environment variable', you must select an environment variable and ensure it contains the version number you want to use. If you choose 'Use the build number', this will use the build number to version your package. Note: Under Options set the build number format to be '\$(BuildDefinitionName)_\$(Year:yyyy).\$(Month).\$(DayOfMonth)\$(Rev:r)'.
Environment variable	(Required) Enter the variable name without \$, \$env, or %.
Major	(Required) The 'X' in version X.Y.Z
Minor	(Required) The 'Y' in version X.Y.Z
Patch	(Required) The 'Z' in version X.Y.Z
Additional build properties	(Optional) Specifies a list of token = value pairs, separated by semicolons, where each occurrence of \$token\$ in the .nuspec file will be replaced with the given value. Values can be strings in quotation marks.

ARGUMENT	DESCRIPTION
Verbosity	(Required) Specifies the amount of detail displayed in the output.
Working Directory	(Required) Current working directory where the script is run. Empty is the root of the repo (build) or artifacts (release), which is \$(System.DefaultWorkingDirectory)
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Android build task (deprecated; use Gradle)

12/3/2018 • 2 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

Use this task in a build or release pipeline to build an Android app using Gradle and optionally start the emulator for unit tests.

Deprecated

The **Android Build task** has been deprecated. Use the [Gradle](#) task instead.

Demands

The build agent must have the following capabilities:

- Android SDK (with the version number you will build against)
- Android Support Repository (if referenced by Gradle file)

Arguments

ARGUMENT	DESCRIPTION
Location of Gradle Wrapper	<p>The location in the repository of the gradlew wrapper used for the build. For agents on Windows (including Microsoft-hosted agents), you must use the <code>gradlew.bat</code> wrapper. Agents on Linux or macOS can use the <code>gradlew</code> shell script.</p> <p>See The Gradle Wrapper.</p>
Project Directory	<p>Relative path from the repo root to the root directory of the application (likely where your build.gradle file is).</p>
Gradle Arguments	<p>Provide any options to pass to the Gradle command line. The default value is <code>build</code></p> <p>See Gradle command line.</p>

ANDROID VIRTUAL DEVICE (AVD) OPTIONS

Name	<p>Name of the AVD to be started or created.</p> <p>Note: You must deploy your own agent to use this option. You cannot use a Microsoft-hosted pool if you want to create an AVD.</p>
Create AVD	<p>Select this check box if you would like the AVD to be created if it does not exist.</p>

ARGUMENT	DESCRIPTION
AVD Target SDK	Android SDK version the AVD should target. The default value is <code>android-19</code>
AVD Device	(Optional) Device pipeline to use. Can be a device index or id. The default value is <code>Nexus 5</code>
AVD ABI	The Application Binary Interface to use for the AVD. The default value is <code>default/armeabi-v7a</code> See ABI Management .
Overwrite Existing AVD	Select this check box if an existing AVD with the same name should be overwritten.
Create AVD Optional Arguments	Provide any options to pass to the <code>android create avd</code> command. See Android Command Line .
EMULATOR OPTIONS	
Start and Stop Android Emulator	Check if you want the emulator to be started and stopped when Android Build task finishes. Note: You must deploy your own agent to use this option. You cannot use a Microsoft-hosted pool if you want to use an emulator.
Timeout in Seconds	How long should the build wait for the emulator to start. The default value is <code>300</code> seconds.
Headless Display	Check if you want to start the emulator with no GUI (headless mode).
Emulator Optional Arguments	(Optional) Provide any options to pass to the <code>emulator</code> command. The default value is <code>-no-snapshot-load -no-snapshot-save</code>
Delete AVD	Check if you want the AVD to be deleted upon completion.
CONTROL OPTIONS	

Related tasks

[Android Signing](#)

Android signing task

10/29/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015

Use this task in a build or release pipeline to sign and align Android APK files.

Demands

The build agent must have the following capabilities:

- Java JDK

YAML snippet

```
# Android Signing
# Sign and align Android APK files
- task: AndroidSigning@3
  inputs:
    #apkFiles: '**/*.apk'
    #apksign: true # Optional
    #apksignerKeystoreFile: # Required when apksign == True
    #apksignerKeystorePassword: # Optional
    #apksignerKeystoreAlias: # Optional
    #apksignerKeyPassword: # Optional
    #apksignerArguments: '--verbose' # Optional
    #apksignerFile: # Optional
    #zipalign: true # Optional
    #zipalignFile: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
APK Files	<p>Relative path from the repo root to the APK(s) you want to sign. You can use wildcards to specify multiple files. For example:</p> <ul style="list-style-type: none"><code>outputs\apk*.apk</code> to sign all .APK files in the <code>outputs\apk\</code> subfolder<code>**\\bin*.apk</code> to sign all .APK files in all bin subfolders
SIGNING OPTIONS	<p>Sign the APK</p> <p>Select this option to sign the APK with a provided keystore file. Unsigned APKs can only run in an emulator. APKs must be signed to run on a device.</p>

ARGUMENT	DESCRIPTION
Keystore File	<p>Enter the file path to the keystore file that should be used to sign the APK. It can either be checked into source control or placed on the build machine directly by an administrator. It is recommended to encrypt the keystore file in source control and use the Decrypt File task to decrypt the file during the build.</p>
Keystore Password	<p>Enter the password for the provided keystore file.</p> <p>Important: We recommend that you put this value in a secret variable.</p>
Alias	<p>Enter the alias that identifies the public/private key pair to be used in the keystore file.</p>
Key Password	<p>Enter the key password for the alias and keystore file.</p> <p>Important: We recommend that you put this value in a secret variable.</p>
Jarsigner Arguments	<p>Provide any options to pass to the jarsigner command line. Default is</p> <pre>-verbose -sigalg MD5withRSA -digestalg SHA1</pre> <p>See jarsigner documentation.</p>
ZIPALIGN OPTIONS	
Zipalign	Select if you want to zipalign your package. This reduces the amount of RAM consumed by an app.
Zipalign Location	(Optional) The location of the zipalign executable used during signing. Defaults to the zipalign found in the Android SDK version folder your application builds against.
CONTROL OPTIONS	

Related tasks

[Android Build](#)

Ant task

11/6/2018 • 3 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

Use this task in a build or release pipeline to build with Apache Ant.

Demands

The build agent must have the following capability:

- Apache Ant

YAML snippet

```
# Ant
# Build with Apache Ant
- task: Ant@1
  inputs:
    #buildFile: 'build.xml'
    #options: # Optional
    #targets: # Optional
    #publishJUnitResults: true
    #testResultsFiles: '**/TEST-*.xml' # Required when publishJUnitResults == True
    #testRunTitle: # Optional
    #codeCoverageToolOptions: 'None' # Optional. Options: none, cobertura, jaCoCo
    #codeCoverageClassFilesDirectories: '.' # Required when codeCoverageToolOptions != None
    #codeCoverageClassFilter: # Optional
    #codeCoverageSourceDirectories: # Optional
    #codeCoverageFailIfEmpty: false # Optional
    #antHomeDirectory: # Optional
    #javaHomeOption: 'JDKVersion' # Options: jDKVersion, path
    #jdkVersionOption: 'default' # Optional. Options: default, 1.11, 1.10, 1.9, 1.8, 1.7, 1.6
    #jdkUserInputDirectory: # Required when javaHomeOption == Path
    #jdkArchitectureOption: 'x64' # Optional. Options: x86, x64
```

Arguments

ARGUMENT	DESCRIPTION
Ant Build File	Relative path from the repository root to the Ant build file. For more information about build files, see Using Apache Ant .

ARGUMENT	DESCRIPTION
Options	<p>Options that you want to pass to the Ant command line.</p> <p>You can provide your own properties (for example, <code>-DmyProperty=myPropertyValue</code>) and also use built-in variables (for example, <code>-DcollectionId=\$(system.collectionId)</code>). Alternatively, the built-in variables are already set as environment variables during the build and can be passed directly (for example, <code>-DcollectionIdAsEnvVar=%SYSTEM_COLLECTIONID%</code>).</p> <p>See Running Apache Ant.</p>
Target(s)	<p>Target(s) for Ant to execute for this build.</p> <p>See Using Apache Ant Targets.</p>
JUNIT TEST RESULTS	
Publish to Azure Pipelines/TFS	<p>Select this option to publish JUnit test results produced by the Ant build to Azure Pipelines or your on-premises Team Foundation Server. Each test result file that matches Test Results Files is published as a test run.</p>
Test Results Files	<p>Test results files path. Wildcards can be used. For example, <code>**/TEST-*.xml</code> for all xml files whose name starts with TEST-."</p>
Test Run Title	<p>Assign a title for the JUnit test case results for this build.</p>
CODE COVERAGE	
Code Coverage Tool	<p>Select the code coverage tool you want to use.</p> <p>If you are using the Microsoft-hosted agents, then the tools are set up for you. If you are using on-premises Windows agent, then if you select:</p> <ul style="list-style-type: none"> • JaCoCo, make sure jacocoant.jar is available in lib folder of Ant installation. See JaCoCo. • Cobertura, set up an environment variable COBERTURA_HOME pointing to the Cobertura .jar files location. See Cobertura. <p>After you select one of these tools, the following arguments appear:</p>
Class Files Directories	<p>Specify a comma-separated list of relative paths from the Ant build file to the directories that contain your .class files, archive files (such as .jar and .war). Code coverage is reported for class files present in the directories. Directories and archives are searched recursively for class files. For example: target/classes,target/testClasses.</p>

ARGUMENT	DESCRIPTION
Class Inclusion/Exclusion Filters	Specify a comma-separated list of filters to include or exclude classes from collecting code coverage. For example: <code>+:com.,+:org.,-:my.app.</code>
Source Files Directories	Specify a comma-separated list of relative paths from the Ant build file to your source directories. Code coverage reports will use these paths to highlight source code. For example: <code>src/java,src/Test.</code>
ADVANCED	
Set ANT_HOME Path	If set, overrides any existing ANT_HOME environment variable with the given path.
Set JAVA_HOME by JDK Version	Choose which JDK level will be used to run Ant. Will attempt to find JDK version and assign JAVA_HOME before running Ant.
Set JAVA_HOME by Path	Directory on build agent where the JDK is located.
JDK Architecture	Optionally supply the architecture (x86, x64) of the JDK.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

CMake task

11/6/2018 • 2 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

Use this task in a build or release pipeline to build with the CMake cross-platform build system.

Demands

cmake

YAML snippet

```
# CMake
# Build with the CMake cross-platform build system
- task: CMake@1
  inputs:
    #workingDirectory: 'build' # Optional
    #cmakeArgs: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Working Directory	Working directory when CMake is run. The default value is <code>build</code> . If you specify a relative path, then it is relative to your repo. For example, if you specify <code>build</code> , the result is the same as if you specified <code>\$(Build.SourcesDirectory)\build</code> . You can also specify a full path outside the repo, and you can use variables . For example: <code>\$(Build.ArtifactStagingDirectory)\build</code> If the path you specify does not exist, CMAke creates it.
Arguments	Arguments that you want to pass to CMake.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

How do I enable CMake for Microsoft-hosted agents?

The [Microsoft-hosted agents](#) have CMake installed, but you must manually add the [capability](#) to use the CMake build task.

1. Open the Agent Pools control panel tab:

- Azure Pipelines: https://dev.azure.com/{your_organization}/_admin/_AgentPool
- TFS 2018: https://{your_server}/DefaultCollection/_admin/_AgentPool
- TFS 2017: https://{your_server}/tfss/DefaultCollection/_admin/_AgentPool
- TFS 2015: http://{your_server}:8080/tfs/_admin/_AgentPool

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

2. In the left column, click the name of the Microsoft-hosted pool that you are using. In the right column click **Capabilities**.

3. Click **Add capability** and set the fields to `cmake` and `yes`.

4. Click **Save changes**

How do I enable CMake for my on-premises agent?

1. [Deploy an agent](#).

2. [Install CMake](#) and make sure to add it to the path of the user that the agent is running as on your agent machine.

3. In your web browser, navigate to the **Agent pools** control panel tab:

- Azure Pipelines: https://dev.azure.com/{your_organization}/_admin/_AgentPool
- TFS 2018: https://{your_server}/DefaultCollection/_admin/_AgentPool
- TFS 2017: https://{your_server}/tfss/DefaultCollection/_admin/_AgentPool
- TFS 2015: http://{your_server}:8080/tfs/_admin/_AgentPool

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

4. In the left column, click the name of your agent pool. In the right column click **Capabilities**.

5. Click **Add capability** and set the fields to `cmake` and `yes`.

6. Click **Save changes**

How does CMake work? What arguments can I use?

[About CMake](#)

[CMake Documentation](#)

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Docker task

11/19/2018 • 7 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018

Use this task in a build or release pipeline to build, tag, push, or run Docker images, or run a Docker command. This task can be used with Docker or Azure Container registry.

NOTE

For YAML pipelines, consider using script-based Docker commands as described in the [Docker guidance](#), and using this Docker task when working with container registries that require authentication.

The built-in Docker task enables you to build Docker images, push Docker images to an authenticated Docker registry, run Docker images, or execute other operations offered by the Docker CLI:

- **Use Docker best practices:** By writing minimal yaml you can build and push an image which is tagged with '\$(Build.BuildId)' and has rich metadata about the repository, commit, build information to the container image as Docker labels
- **Conform to Docker standards:** The task takes care of details like tagging image with the registry hostname and port image before pushing the image to a private registry like Azure Container Registry (ACR). It also helps you to follow Docker naming convention, for example, converting upper case character to lower case and removes spaces in image name which can happen if you are using \$(Build.Repository.Name) to name your images.
- **Manage secrets:** The task makes it easy to use either 'Docker registry service connection' for connecting to any private container registry or 'Azure Service Connection' For connecting to ACR. For example, in case of ACR you don't have to enable 'admin user' and manage username and password as secret. The task will use the Azure Service connection to login to ACR. Once you have used the Docker task to sign in, the session is maintained for the duration of the job thus allowing you to use follow-up tasks to execute any scripts by leveraging the login done by the Docker task. For example, You can use the Docker task to sign into ACR and then use a subsequent script to pull an image and scan the container image for vulnerabilities.

YAML snippet

Build Docker images

Build a Dockerfile into an image with a registry-qualified name and multiple tags such as the build ID, source branch name and Git tags:

```
- task: Docker@1
  displayName: 'Build an image'
  inputs:
    azureSubscriptionEndpoint: 'ContosoAzureSubscription'
    azureContainerRegistry: contoso.azurecr.io
```

For other private container registries

```
- task: Docker@1
displayName: 'Build an image'
inputs:
  containerregistrytype: 'Container Registry'
  dockerRegistryEndpoint: Contoso
```

'azureSubscriptionEndpoint' input is the name of Azure Service Connection. See [Azure Resource Manager service connection](#) to manually set up the connection. 'dockerRegistryEndpoint' input is the name of [Docker Registry service connection](#).

This will result in a docker login to the container registry by using the service connection and then a docker build command will be used to build and tag the image. For example, a simplified version of the command run is:

```
docker build -t contoso.azurecr.io/contoso-ci:11 .
```

By writing minimal yaml you can build and push an image which is tagged with '\$(Build.BuildId)' and has rich metadata about the repository, commit, build information to the container image as Docker labels. The task takes care of details like tagging image with the registry hostname and port image before pushing the image to a private registry like Azure Container Registry (ACR). It also helps you to follow Docker naming convention, for example, converting upper case character to lower case and removes spaces in image name which can happen if you are using \$(Build.Repository.Name) to name your images.

Push Docker images

Push Docker images with multiple tags to an authenticated Docker Registry and save the resulting repository image digest to a file:

```
- task: Docker@1
displayName: 'Push an image'
inputs:
  azureSubscriptionEndpoint: 'ContosoAzureSubscription'
  azureContainerRegistry: contoso.azurecr.io
  command: 'push'
```

This will result in a docker login to the container registry by using the service connection and then a docker push command will be used to push the image to the container registry. For example, a simplified version of the command run is:

```
docker push contoso.azurecr.io/contoso-ci:11
```

Build, tag and push container image

Here is an end to end sample yaml for building, tagging and pushing container image.

```

- task: Docker@1
  displayName: 'Build an image'
  inputs:
    imageName: 'contoso.azurecr.io/repositoryname:${Build.BuildId}'
- task: Docker@1
  displayName: Login
  inputs:
    azureSubscriptionEndpoint: 'ContosoAzureSubscription'
    azureContainerRegistry: contoso.azurecr.io
    command: login
- task: Docker@1
  displayName: 'Push an image'
  inputs:
    command: 'push'
    imageName: 'contoso.azurecr.io/$repositoryname:${Build.BuildId}'

```

Login to a container registry and run scripts

Task makes it easy to use either 'Docker registry service connection' for connecting to any private container registry or 'Azure Service Connection' For connecting to ACR. For example, in the case of ACR you don't have to enable 'admin user' and manage username and password as secret. The task will use the Azure Service connection to login to ACR. Once you have used the task to login, the session is maintained for the duration of the job thus allowing you to use follow-up tasks to execute any scripts by leveraging the login done by the Docker task. For example, You can use the Docker task to sign into ACR and then use a subsequent script to pull an image and scan the container image for vulnerabilities.

```

- task: Docker@1
  displayName: Login
  inputs:
    azureSubscriptionEndpoint: 'ContosoAzureSubscription'
    azureContainerRegistry: contoso.azurecr.io
    command: login
- bash: |
  # Write your commands here
  # Use the environment variables input below to pass secret variables to this script
  docker build -t contoso.azurecr.io/repositoryname:${Build.BuildId} . # include other options to meet your needs
  docker push contoso.azurecr.io/repositoryname:${Build.BuildId}
  displayName: 'Build, tag and push image'

```

Run Docker images

Perform isolated workloads inside a container by running a Docker image. A Docker image can also be run in the background with a specific restart policy.

```

- task: Docker@1
  displayName: 'Push an image'
  inputs:
    azureSubscriptionEndpoint: 'ContosoAzureSubscription'
    azureContainerRegistry: contoso.azurecr.io
    command: 'run'
    containerName: contosocontainer
    ports: 8084
    volumes: '$(System.DefaultWorkingDirectory):/src'
    workingDirectory: /src
    containerCommand: 'npm install'
    restartPolicy: onFailure

```

This will result in a docker run command. For example:

```
docker run -d --restart no atul-aks:1382
```

Arguments

ARGUMENT	DESCRIPTION
Container Registry Type	(Required) Select a Container Registry Type.
Docker Registry Connection	(Optional) Select a Docker registry connection. Required for commands that need to authenticate with a registry.
Azure subscription	(Optional) Select an Azure subscription
Azure Container Registry	(Optional) Select an Azure Container Registry
Action	(Required) Select a Docker action.
Docker File	(Required) Path to the Docker file to use. Must be within the Docker build context.
Build Arguments	(Optional) Build-time variables for the Docker file. Specify each name=value pair on a new line.
Use Default Build Context	(Optional) Set the build context to the directory that contains the Docker file.
Build Context	(Optional) Path to the build context.
Image Name	(Required) Name of the Docker image to build, push, or run.
Image Names Path	(Required) Path to a text file that contains the names of the Docker images to tag or push. Each image name is contained on its own line.
Qualify Image Name	(Optional) Qualify the image name with the Docker registry connection's hostname if not otherwise specified.
Additional Image Tags	(Optional) Additional tags for the Docker image being built or pushed.
Include Source Tags	(Optional) Include Git tags when building or pushing the Docker image.
Include Latest Tag	(Optional) Include the 'latest' tag when building or pushing the Docker image.
Image Digest File	(Optional) Path to a file that is created and populated with the full image repository digest of the Docker image that was pushed.
Container Name	(Optional) Name of the Docker container to run.
Ports	(Optional) Ports in the Docker container to publish to the host. Specify each host-port:container-port binding on a new line.
Volumes	(Optional) Volumes to mount from the host. Specify each host-dir:container-dir on a new line.

ARGUMENT	DESCRIPTION
Environment Variables	(Optional) Environment variables for the Docker container. Specify each name=value pair on a new line.
Working Directory	(Optional) The working directory for the Docker container.
Entrypoint Override	(Optional) Override the default entrypoint for the Docker container.
Command	(Optional) Command to run in the Docker container. For example, if the image contains a simple Python Flask web application you can specify 'python app.py' to launch the web application.
Run In Background	(Optional) Run the Docker container in the background.
Restart Policy	(Required) Select a restart policy.
Maximum Restart Retries	(Optional) The maximum number of restart retries the Docker daemon attempts.
Command	(Required) Docker command to execute, with arguments. For example, 'rmi -f image-name' to force remove an image.
Docker Host Connection	(Optional) Select a Docker host connection. Defaults to the agent's host.
Force image name to follow Docker naming convention	(Optional) If enabled docker image name will be modified to follow Docker naming convention. Converts upper case character to lower case and removes spaces in image name.
Working Directory	(Optional) Working directory for the Docker command.
Memory limit	(Optional) The maximum amount of memory available to the container as a integer with optional suffixes like '2GB'.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Docker Compose task

11/6/2018 • 3 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to build, push or run multi-container Docker applications. This task can be used with Docker or Azure Container registry.

YAML snippet

```
# Docker Compose
# Build, push or run multi-container Docker applications. Task can be used with Docker or Azure Container
registry.
- task: DockerCompose@0
  inputs:
    #containerregistrytype: 'Azure Container Registry' # Options: azure Container Registry, container Registry
    #dockerRegistryEndpoint: # Optional
    #azureSubscription: # Optional
    #azureContainerRegistry: # Optional
    #dockerComposeFile: '**/docker-compose.yml'
    #additionalDockerComposeFiles: # Optional
    #dockerComposeFileArgs: # Optional
    # projectName: '$(Build.Repository.Name)' # Optional
    #qualifyImageNames: true # Optional
    #action: 'Run a Docker Compose command' # Options: build Services, push Services, run Services, run A
    Specific Service, lock Services, write Service Image Digests, combine Configuration, run A Docker Compose
    Command
    #additionalImageTags: # Optional
    #includeSourceTags: false # Optional
    #includeLatestTag: false # Optional
    #buildImages: true # Optional
    #serviceName: # Required when action == Run A Specific Service
    #containerName: # Optional
    #ports: # Optional
    #workingDirectory: # Optional
    #entrypoint: # Optional
    #containerCommand: # Optional
    #detached: true # Optional
    #abortOnContainerExit: true # Optional
    #imageDigestComposeFile: '$(Build.StagingDirectory)/docker-compose.images.yml' # Required when action ==
    Write Service Image Digests
    #removeBuildOptions: false # Optional
    #baseResolveDirectory: # Optional
    #outputDockerComposeFile: '$(Build.StagingDirectory)/docker-compose.yml' # Required when action == Lock
    Services || Action == Combine Configuration
    #dockerComposeCommand: # Required when action == Run A Docker Compose Command
    #dockerHostEndpoint: # Optional
    #nopIfNoDockerComposeFile: false # Optional
    #requireAdditionalDockerComposeFiles: false # Optional
    #currentWorkingDirectory: '$(System.DefaultWorkingDirectory)' # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Container Registry Type	(Required) Select a Container Registry Type.

ARGUMENT	DESCRIPTION
Docker Registry Connection	(Optional) Select a Docker registry connection. Required for commands that need to authenticate with a registry.
Azure subscription	(Optional) Select an Azure subscription
Azure Container Registry	(Optional) Select an Azure Container Registry
Docker Compose File	(Required) Path to the primary Docker Compose file to use.
Additional Docker Compose Files	(Optional) Additional Docker Compose files to be combined with the primary Docker Compose file. Relative paths are resolved relative to the directory containing the primary Docker Compose file. If a specified file is not found, it is ignored. Specify each file path on a new line.
Environment Variables	(Optional) Environment variables to be set during the command. Specify each name=value pair on a new line.
Project Name	(Optional) Project name used for default naming of images and containers.
Qualify Image Names	(Optional) Qualify image names for built services with the Docker registry connection's hostname if not otherwise specified.
Action	(Required) Select a Docker Compose action.
Additional Image Tags	(Optional) Additional tags for the Docker images being built or pushed.
Include Source Tags	(Optional) Include Git tags when building or pushing Docker images.
Include Latest Tag	(Optional) Include the 'latest' tag when building or pushing Docker images.
Build Images	(Optional) Build images before starting service containers.
Service Name	(Required) Name of the specific service to run.
Container Name	(Optional) Name of the specific service container to run.
Ports	(Optional) Ports in the specific service container to publish to the host. Specify each host-port:container-port binding on a new line.
Working Directory	(Optional) The working directory for the specific service container.
Entrypoint Override	(Optional) Override the default entry point for the specific service container.

ARGUMENT	DESCRIPTION
Command	(Optional) Command to run in the specific service container. For example, if the image contains a simple Python Flask web application you can specify 'python app.py' to launch the web application.
Run In Background	(Optional) Run the service containers in the background.
Abort on Container Exit	(Optional) Stop all containers when any container exits.
Image Digest Compose File	(Required) Path to a Docker Compose file that is created and populated with the full image repository digests of each service's Docker image.
Remove Build Options	(Optional) Remove the build options from the output Docker Compose file.
Base Resolve Directory	(Optional) The base directory from which relative paths in the output Docker Compose file should be resolved.
Output Docker Compose File	(Required) Path to an output Docker Compose file.
Command	(Required) Docker Compose command to execute with arguments. For example, 'rm --all' to remove all stopped service containers.
Docker Host Connection	(Optional) Select a Docker host connection. Defaults to the agent's host.
No-op if no Docker Compose File	(Optional) If the Docker Compose file does not exist, skip this step. This is useful when the step offers optional behavior based on the existence of a Docker Compose file in the repository.
Require Additional Docker Compose Files	(Optional) Produces an error if the additional Docker Compose files do not exist. This overrides the default behavior which is to ignore a file if it does not exist.
Working Directory	(Optional) Working directory for the Docker Compose command.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Go task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to get, build, or test a go application, or run a custom go command.

YAML snippet

```
# Go
# Get, build, or test a Go application, or run a custom Go command.
- task: Go@0
  inputs:
    #command: 'get' # Options: get, build, test, custom
    #customCommand: # Required when command == Custom
    #arguments: # Optional
    workingDirectory:
```

Arguments

ARGUMENT	DESCRIPTION
Command	(Required) Select a Go command to run. Select 'Custom' to use a command not listed here.
Custom command	(Required) Custom Go command for execution. For example: to execute go version, enter version.
Arguments	(Optional) Arguments to the selected command. For example, build time arguments for go build command.
Working Directory	(Required) Current working directory where the script is run. Empty is the root of the repo (build) or artifacts (release), which is \$(System.DefaultWorkingDirectory)
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Gradle task

11/6/2018 • 4 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

Use this task in a build or release pipeline to build using a Gradle wrapper script.

YAML snippet

```
# Gradle
# Build using a Gradle wrapper script
- task: Gradle@2
  inputs:
    #gradleWrapperFile: 'gradlew'
    #workingDirectory: # Optional
    #options: # Optional
    #tasks: 'build'
    #publishJUnitResults: true
    #testResultsFiles: '**/TEST-*.xml' # Required when publishJUnitResults == True
    #testRunTitle: # Optional
    #codeCoverageToolOption: 'None' # Optional. Options: none, cobertura, jaCoCo
    #codeCoverageClassDirectories: 'build/classes/main/' # Required when codeCoverageToolOption == False
    #codeCoverageClassFilter: # Optional
    #codeCoverageFailIfEmpty: false # Optional
    #javaHomeOption: 'JDKVersion' # Options: jDKVersion, path
    #jdkVersionOption: 'default' # Optional. Options: default, 1.11, 1.10, 1.9, 1.8, 1.7, 1.6
    #jdkDirectory: # Required when javaHomeOption == Path
    #jdkArchitectureOption: 'x64' # Optional. Options: x86, x64
    #gradleOptions: '-Xmx1024m' # Optional
    #sonarQubeRunAnalysis: false
    #sqGradlePluginVersionChoice: 'specify' # Required when sonarQubeRunAnalysis == True# Options: specify,
  build
    #sonarQubeGradlePluginVersion: '2.6.1' # Required when sonarQubeRunAnalysis == True &&
    SqGradlePluginVersionChoice == Specify
    #checkStyleRunAnalysis: false # Optional
    #findBugsRunAnalysis: false # Optional
    #pmdRunAnalysis: false # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Gradle Wrapper	The location in the repository of the gradlew wrapper used for the build. For agents on Windows (including Microsoft-hosted agents), you must use the <code>gradlew.bat</code> wrapper. Agents on Linux or macOS can use the <code>gradlew</code> shell script. See The Gradle Wrapper .
Options	Specify any command line options you want to pass to the Gradle wrapper. See Gradle Command Line .

ARGUMENT	DESCRIPTION
Tasks	The task(s) for Gradle to execute. A list of tasks can be taken from <code>gradlew tasks</code> issued from a command prompt. See Gradle Build Script Basics .
JUNIT TEST RESULTS	
Publish to Azure Pipelines/TFS	Select this option to publish JUnit Test results produced by the Gradle build to Azure Pipelines/TFS.
Test Results Files	Test results files path. Wildcards can be used. For example, <code>**/TEST-*.xml</code> for all xml files whose name starts with TEST-."
Test Run Title	Assign a title for the JUnit test case results for this build.
CODE COVERAGE	
Code Coverage Tool	Choose a code coverage tool to determine the code that is covered by the test cases for the build.
ADVANCED	
Working Directory	Directory on the build agent where the Gradle wrapper will be invoked from. Defaults to the repository root.
Set JAVA_HOME by JDK Version	Choose which JDK level to run Gradle with. Will attempt to find JDK version and assign JAVA_HOME before running Gradle.
Set JAVA_HOME by Path	Directory on build agent where JDK is located.
JDK Architecture	Optionally supply the architecture (x86, x64) of JDK.
CODE ANALYSIS	
Run SonarQube Analysis	Select if you want to run a SonarQube analysis. See The Gradle build task now supports SonarQube analysis .
Run PMD Analysis	Select if you want to perform a PMD static analysis . A build result page for each project is shown on the Artifacts tab of the completed build. See Gradle build task now also supports PMD analysis .
Run Checkstyle Analysis	Select if you want to perform a Checkstyle static analysis . The build summary reports the number of issues found by Checkstyle. Detailed issue logs are available under the build Artifact tab of the build summary. If the Checkstyle analysis is customized, the task only attempts to find the reports and produce a summary.
CONTROL OPTIONS	

Example

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

How do I generate a wrapper from my Gradle project?

The Gradle wrapper allows the build agent to download and configure the exact Gradle environment that is checked into the repository without having any software configuration on the build agent itself other than the JVM.

1. Create the Gradle wrapper by issuing the following command from the root project directory where your `build.gradle` resides:

```
jamal@fabrikam> gradle wrapper
```

2. Upload your Gradle wrapper to your remote repository.

There is a binary artifact that is generated by the gradle wrapper (located at

`gradle/wrapper/gradle-wrapper.jar`). This binary file is small and doesn't require updating. If you need to change the Gradle configuration run on the build agent, you update the `gradle-wrapper.properties`.

The repository should look something like this:

```
|-- gradle/
|   '-- wrapper/
|       '-- gradle-wrapper.jar
|       '-- gradle-wrapper.properties
|-- src/
|-- .gitignore
|-- build.gradle
|-- gradlew
|-- gradlew.bat
```

How do I fix timeouts when downloading dependencies?

To fix errors such as `Read timed out` when downloading dependencies, users of Gradle 4.3+ can change the timeout by adding to `Options -Dhttp.socketTimeout=60000 -Dhttp.connectionTimeout=60000`. This increases the timeout from 10 seconds to 1 minute.

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Grunt task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015.3

Use this task in a build or release pipeline to run Grunt tasks using the JavaScript Task Runner.

Demands

The build agent must have the following capability:

- Grunt

YAML snippet

```
# Grunt
# The JavaScript Task Runner
- task: Grunt@0
  inputs:
    #gruntFile: 'gruntfile.js'
    #targets: # Optional
    #arguments: # Optional
    #workingDirectory: # Optional
    #gruntCli: 'node_modules/grunt-cli/bin/grunt'
    #publishJUnitResults: false # Optional
    #testResultsFiles: '**/TEST-*.xml' # Required when publishJUnitResults == True
    #testRunTitle: # Optional
    #enableCodeCoverage: false # Optional
    #testFramework: 'Mocha' # Optional. Options: mocha, jasmine
    #srcFiles: # Optional
    #testFiles: 'test/*.js' # Required when enableCodeCoverage == True
```

Arguments

ARGUMENT	DESCRIPTION
Grunt File Path	Relative path from the repo root to the Grunt script that you want to run. The default value is <code>gruntfile.js</code>
Grunt task(s)	(Optional) Space delimited list of tasks to run. If you leave it blank, the default task will run.
ADVANCED	
Arguments	Additional arguments passed to Grunt. See Using the CLI . Tip: <code>--gruntfile</code> is not needed. This argument is handled by the Grunt file path argument shown above.
Working directory	Current working directory when the script is run. If you leave it blank, the working directory is the folder where the script is located.

ARGUMENT	DESCRIPTION
CONTROL OPTIONS	

Example

See [Sample Gruntfile](#).

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Gulp task

11/19/2018 • 2 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

Use this task in a build or release pipeline to run gulp tasks using the Node.js streaming task based build system.

Demands

gulp

YAML snippet

```
# gulp
# Node.js streaming task based build system
- task: gulp@1
  inputs:
    #gulpFile: 'gulpfile.js'
    #targets: # Optional
    #arguments: # Optional
    #workingDirectory: # Optional
    #gulpjs: # Optional
    #publishJUnitResults: false # Optional
    #testResultsFiles: '**/TEST-*.xml' # Required when publishJUnitResults == True
    #testRunTitle: # Optional
    #enableCodeCoverage: false
    #testFramework: 'Mocha' # Optional. Options: mocha, jasmine
    #srcFiles: # Optional
    #testFiles: 'test/*.js' # Required when enableCodeCoverage == True
```

Arguments

ARGUMENT	DESCRIPTION
gulp file path	Relative path from the repo root to the gulp script that you want to run. The default value is <code>gulpfile.js</code>
gulp task(s)	(Optional) Space delimited list of tasks to run. If you leave it blank, the default task will run.

ADVANCED	
Arguments	Additional arguments passed to gulp. Tip: <code>--gulpfile</code> is not needed. This argument is handled by the gulp file path argument shown above.
Working directory	Current working directory when the script is run. If you leave it blank, the working directory is the folder where the script is located.
gulp.js location	gulp.js to run. The default value is <code>node_modules/gulp/bin/gulp.js</code>

ARGUMENT	DESCRIPTION
CONTROL OPTIONS	

Example

Run gulp.js

On the [Build](#) tab:

 Package: npm	Install npm. <ul style="list-style-type: none"> Command: <code>install</code>
 Build: gulp	Run your script. <ul style="list-style-type: none"> gulp file path: <code>gulpfile.js</code> Advanced, gulp.js location: <code>node_modules/gulp/bin/gulp.js</code>

Build a Node.js app

[Build your Node.js app with gulp](#)

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Index Sources & Publish Symbols task

11/28/2018 • 5 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

A symbol server is available with Package Management in [Azure Artifacts](#) and works best with **Visual Studio 2017.4 and newer**. **Team Foundation Server** users and users without the Package Management extension can publish symbols to a file share using this task.

Use this task in a build or release pipeline to index your source code and optionally publish symbols to the Package Management symbol server or a file share.

Indexing source code enables you to use your .pdb symbol files to debug an app on a machine other than the one you used to build the app. For example, you can debug an app built by a build agent from a dev machine that does not have the source code.

Symbol servers enables your debugger to automatically retrieve the correct symbol files without knowing product names, build numbers or package names. To learn more about symbols, read the [concept page](#); to publish symbols, use this task and see [the walkthrough](#).

NOTE

This build task works only:

- For code in Git or TFVC stored in Team Foundation Server (TFS) or Azure Repos. It does not work for any other type of repository.

Demands

None

YAML snippet

```
# Index Sources & Publish Symbols
# Index your source code and publish symbols to a file share or Azure Artifacts symbol server.
- task: PublishSymbols@2
  inputs:
    #symbolsFolder: '$(Build.SourcesDirectory)' # Optional
    #searchPattern: '**/bin/**/*.pdb'
    #indexSources: true # Optional
    #publishSymbols: true # Optional
    #symbolServerType: '' # Required when publishSymbols == True# Options: , teamServices, fileShare
    #symbolsPath: # Optional
    #compressSymbols: false # Required when symbolServerType == FileShare
    #detailedLog: true # Optional
    #treatNotIndexedAsWarning: false # Optional
    #symbolsMaximumWaitTime: # Optional
    #symbolsProduct: # Optional
    #symbolsVersion: # Optional
    #symbolsArtifactName: 'Symbols_${BuildConfiguration}' # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Path to symbols folder	The root path that is searched for symbol files using the search patterns supplied in the next input.
Search pattern	File matching pattern(s) (rooted at the path supplied in the previous input) used to discover <code>pdb</code> s that contain symbols.
Index sources	Adds information about the location of the source repository to the symbols. This enables users using these symbols to navigate to the relevant source code.
Publish symbols	Publishes symbols to the symbol server selected in the next inputs.
Symbol server type	<p>Package Management in Azure Artifacts:</p> <ul style="list-style-type: none"> Select this option to use the symbol server built into the Package Management extension. <p>File share:</p> <ul style="list-style-type: none"> Select this option to use the file share supplied in the next input.
Path to publish symbols	<p>The path to the SymStore file share.</p> <p>To prepare your SymStore symbol store:</p> <ol style="list-style-type: none"> Set up a folder on a file-sharing server to store the symbols. For example, set up <code>\fabrikam-share\symbols</code>. Grant full control permission to the build agent service account. <p>If you leave this argument blank, your symbols will be source indexed but not published. (You can also store your symbols with your drops. See Publish Build Artifacts).</p>
Compress symbols	Only available when File share is selected as the Symbol server type . Compresses your <code>pdb</code> s to save space.
ADVANCED	
Verbose logging	Enables additional log details.

ARGUMENT	DESCRIPTION
Warn if not indexed	<p>Enable this option if you want the build summary to show a warning when sources are not indexed for a PDB file. A common cause of sources to not be indexed are when your solution depends on binaries that it doesn't build.</p> <p>Even if you don't select this option, the messages are written in log.</p>
Max wait time (min)	If you want to set a time limit for this task, specify the number of minutes here. The build fails when the limit is reached. If you leave it blank, limit is 2 hours.
Product	If you are publishing your symbols, you can specify the product parameter that is passed to symstore.exe. If blank, <code>\$(Build.DefinitionName)</code> is passed.
Version	If you are publishing your symbols, you can specify the version parameter that is passed to symstore.exe. If blank, <code>\$(Build.BuildNumber)</code> is passed.
Artifact name	Specify the pattern used for the name of the link from the artifact tab in the build summary to the file share where you are publishing your symbols. For example, if you specify <code>Symbols_{BuildConfiguration}</code> , then the name of the link to your published release symbols would be <code>Symbols_release</code>

CONTROL OPTIONS

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

How does indexing work?

By choosing to index the sources, an extra section will be injected into the PDB files. PDB files normally contain references to the local source file paths only. For example, `C:\BuildAgent_work\1\src\MyApp\Program.cs`. The extra section injected into the PDB file contains mapping instructions for debuggers. The mapping information indicates how to retrieve the server item corresponding to each local path.

The Visual Studio debugger will use the mapping information to retrieve the source file from the server. An actual command to retrieve the source file is included in the mapping information. You may be prompted by Visual Studio whether to run the command. For example

```
tf.exe git view /collection:http://SERVER:8080/tfs/DefaultCollection /teamproject:"93fc2e4d-0f0f-4e40-9825-01326191395d" /repository:"647ed0e6-43d2-4e3d-b8bf-2885476e9c44"
/commitId:3a9910862e22f442cd56ff280b43dd544d1ee8c9 /path:"/MyApp/Program.cs"
/output:"C:\Users\username\AppData\Local\SOURCE~1\TFS_COMMIT\3a991086\MyApp\Program.cs" /applyfilters
```

Can I use source indexing on a portable PDB created from a .NET Core assembly?

No, source indexing is currently not enabled for Portable PDBs as SourceLink doesn't support authenticated source repositories. The workaround at the moment is to configure the build to generate full PDBs. Note that if

you are generating a .NET Standard 2.0 assembly and are generating full PDBs and consuming them in a .NET Framework (full CLR) application then you will be able to fetch sources from Azure Repos (provided you have embedded SourceLink information and enabled it in your IDE).

Where can I learn more about symbol stores and debugging?

[Symbol Server and Symbol Stores](#)

[SymStore](#)

[Use the Microsoft Symbol Server to obtain debug symbol files](#)

[The Srcsrv.ini File](#)

[Source Server](#)

[Source Indexing and Symbol Servers: A Guide to Easier Debugging](#)

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

How long are Symbols retained?

When symbols are published to Azure Pipelines they are associated with a build. When the build is deleted either manually or due to retention policy then the symbols are also deleted. If you want to retain the symbols indefinitely then you should mark the build as Retain Indefinately.

Jenkins Queue Job task

11/6/2018 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

Use this task in a build or release pipeline to queue a job on a Jenkins server.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions.
Service connections are called *service endpoints* in TFS 2018 and in older versions.

Demands

None

YAML snippet

```
# Jenkins Queue Job
# Queue a job on a Jenkins server
- task: JenkinsQueueJob@2
  inputs:
    serverEndpoint:
    jobName:
    #isMultibranchJob: # Optional
    #multibranchPipelineBranch: # Required when isMultibranchJob == True
    captureConsole:
    #capturePipeline: # Required when captureConsole == True
    isParameterizedJob:
    #jobParameters: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Jenkins service connection	Select the service connection for your Jenkins instance. To create one, click Manage and create a new Jenkins service connection.
Job name	The name of the Jenkins job to queue. This must exactly match the job name on the Jenkins server.
Capture console output and wait for completion	If selected, this task will capture the Jenkins build console output, wait for the Jenkins build to complete, and succeed/fail based on the Jenkins build result. Otherwise, once the Jenkins job is successfully queued, this task will successfully complete without waiting for the Jenkins build to run.

ARGUMENT	DESCRIPTION
Capture pipeline output and wait for pipeline completion	This option is similar to capture console output except it will capture the output for the entire Jenkins pipeline, wait for completion for the entire pipeline, and succeed/fail based on the pipeline result.
Parameterized job	Select this option if the Jenkins job requires parameters.
Job parameters	<p>This option is available for parameterized jobs. Specify job parameters, one per line, in the form parameterName=parameterValue</p> <p>To set a parameter to an empty value (useful for overriding a default value) leave off the parameter value, e.g. specify parameterName=</p> <p>Variables are supported, e.g. to define the commitId parameter to be the git commit ID for the build, use: commitId=\${Build.SourceVersion}.</p> <p>Supported Jenkins parameter types are:</p> <ul style="list-style-type: none"> • Boolean • String • Choice • Password
Trust server certificate	Selecting this option results in the Jenkins server's SSL certificate being trusted even if it is self-signed or cannot be validated by a Certificate Authority (CA).

Team Foundation Server Plug-in

You can use Team Foundation Server Plug-in (version 5.2.0 or newer) to automatically collect files from the Jenkins workspace and download them into the build.

To set it up:

1. Install the [Team Foundation Server Plug-in](#) on the Jenkins server.
2. On the Jenkins server, for each job you would like to collect results from, add the **Collect results for Azure Pipelines/TFS post-build action** and then configure it with one or more pairs of result type and include file pattern.
3. On the Jenkins Queue Job build task enable the **Capture console output and wait for completion** to collect results from the root level job, or the **Capture pipeline output and wait for pipeline completion** to collect results from all pipeline jobs.

Results will be downloaded to the **\$(Build.StagingDirectory)/jenkinsResults/<Job Name>/team-results.zip** and extracted to this location. Each set of result types collected by the plug-in, will be under the team-results directory, **\$(Build.StagingDirectory)/jenkinsResults/<Job Name>/team-results/<ResultType>/**. This is the directory where build results can be published by downstream tasks (e.g. Publish Test Results, and Publish Code Coverage Results).

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Q: I'm having problems. How can I troubleshoot them?

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

Q: How do variables work? What variables are available for me to use in the arguments?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See [Variables](#).

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Maven task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015

Use this task in a build or release pipeline to build your Java code.

Demands

The build agent must have the following capability:

- Maven

YAML snippet

```
# Maven
# Build with Apache Maven
- task: Maven@3
  inputs:
    #mavenPomFile: 'pom.xml'
    #goals: 'package' # Optional
    #options: # Optional
    #publishJUnitResults: true
    #testResultsFiles: '**/surefire-reports/TEST-*.xml' # Required when publishJUnitResults == True
    #testRunTitle: # Optional
    #codeCoverageToolOption: 'None' # Optional. Options: none, cobertura, jaCoCo
    #codeCoverageClassFilter: # Optional
    #codeCoverageClassFilesDirectories: # Optional
    #codeCoverageSourceDirectories: # Optional
    #codeCoverageFailIfEmpty: false # Optional
    #javaHomeOption: 'JDKVersion' # Options: jdkVersion, path
    #jdkVersionOption: 'default' # Optional. Options: default, 1.11, 1.10, 1.9, 1.8, 1.7, 1.6
    #jdkDirectory: # Required when javaHomeOption == Path
    #jdkArchitectureOption: 'x64' # Optional. Options: x86, x64
    #mavenVersionOption: 'Default' # Options: default, path
    #mavenDirectory: # Required when mavenVersionOption == Path
    #mavenSetM2Home: false # Required when mavenVersionOption == Path
    #mavenOptions: '-Xmx1024m' # Optional
    #mavenAuthenticateFeed: false
    #sonarQubeRunAnalysis: false
    #sqMavenPluginVersionChoice: 'latest' # Required when sonarQubeRunAnalysis == True# Options: latest, pom
    #checkStyleRunAnalysis: false # Optional
    #pmdRunAnalysis: false # Optional
    #findBugsRunAnalysis: false # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Maven POM file	Relative path from the repo root to the Maven POM .xml file. See Introduction to the POM .
Options	Specify any Maven options you want to use.

ARGUMENT	DESCRIPTION
Goal(s)	In most cases, set this to <code>package</code> to compile your code and package it into a .war file. If you leave this argument blank, the build will fail. See Introduction to the Maven build lifecycle .
JUNIT TEST RESULTS	
Publish to Azure Pipelines/TFS	Select this option to publish JUnit Test results produced by the Maven build to Azure Pipelines/TFS.
Test Results Files	Test results files path. Wildcards can be used. For example, <code>**/TEST-*.xml</code> for all xml files whose name starts with TEST-."
ADVANCED	
JDK Version	Will attempt to discover the path to the selected JDK version and set JAVA_HOME accordingly.
JDK Architecture	Optionally supply the architecture (x86, x64) of JDK.
CODE ANALYSIS	
Run SonarQube Analysis	Select if you want to run a SonarQube analysis. See The Maven build task now simplifies SonarQube analysis .
Run PMD Analysis	Select if you want to perform a PMD static analysis . A build result page for each .pom file is shown on the Artifacts tab of the completed build. See The Maven build task now supports PMD analysis out of the box .
CONTROL OPTIONS	

Example

[Build and Deploy your Java application to an Azure Web App](#)

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

MSBuild task

11/6/2018 • 4 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Use this task in a build or release pipeline to build with MSBuild.

Demands

msbuild

Azure Pipelines: If your team uses Visual Studio 2017 and you want to use the Microsoft-hosted agents, make sure you select as your default pool the **Hosted VS2017**. See [Microsoft-hosted agents](#).

YAML snippet

```
# MSBuild
# Build with MSBuild
- task: MSBuild@1
  inputs:
    #solution: '**/*.sln'
    #msbuildLocationMethod: 'version' # Optional. Options: version, location
    #msbuildVersion: 'latest' # Optional. Options: latest, 15.0, 14.0, 12.0, 4.0
    #msbuildArchitecture: 'x86' # Optional. Options: x86, x64
    #msbuildLocation: # Optional
    #platform: # Optional
    #configuration: # Optional
    #msbuildArguments: # Optional
    #clean: false # Optional
    #maximumCpuCount: false # Optional
    #restoreNugetPackages: false # Optional
    #logProjectEvents: false # Optional
    #createLogFile: false # Optional
```

Arguments

ARGUMENT	DESCRIPTION
----------	-------------

ARGUMENT	DESCRIPTION
Project	<p>If you want to build a single project, click the ... button and select the project.</p> <p>If you want to build multiple projects, specify search criteria. You can use a single-folder wildcard (<code>*</code>) and recursive wildcards (<code>**</code>). For example, <code>***.proj</code> searches for all MSBuild project (*.proj) files in all subdirectories.</p> <p>Make sure the projects you specify are downloaded by this build pipeline. On the Repository tab:</p> <ul style="list-style-type: none"> • If you use TFVC, make sure that the project is a child of one of the mappings on the Repository tab. • If you use Git, make sure that the project or project is in your Git repo, in a branch that you're building. <p>Tip: If you are building a solution, we recommend you use the Visual Studio build task instead of the MSBuild task.</p>
MSBuild Arguments	<p>You can pass additional arguments to MSBuild. For syntax, see MSBuild Command-Line Reference.</p>
Platform	<p>Specify the platform you want to build such as <code>Win32</code>, <code>x86</code>, <code>x64</code> or <code>any cpu</code>.</p> <p>Tips:</p> <ul style="list-style-type: none"> • If you are targeting an MSBuild project (*.proj) file instead of a solution, specify <code>AnyCPU</code> (no whitespace). • Declare a build variable such as <code>BuildPlatform</code> on the Variables tab (selecting Allow at Queue Time) and reference it here as <code>\$(BuildPlatform)</code>. This way you can modify the platform when you queue the build and enable building multiple configurations.
Configuration	<p>Specify the configuration you want to build such as <code>debug</code> or <code>release</code>.</p> <p>Tip: Declare a build variable such as <code>BuildConfiguration</code> on the Variables tab (selecting Allow at Queue Time) and reference it here as <code>\$(BuildConfiguration)</code>. This way you can modify the platform when you queue the build and enable building multiple configurations.</p>
Clean	<p>Set to False if you want to make this an incremental build. This setting might reduce your build time, especially if your codebase is large. This option has no practical effect unless you also set Clean repository to False.</p> <p>Set to True if you want to rebuild all the code in the code projects. This is equivalent to the MSBuild <code>/target:clean</code> argument.</p>

ARGUMENT	DESCRIPTION
Restore NuGet Packages	(Important) This option is deprecated. Make sure to clear this checkbox and instead use the NuGet Installer build task.
ADVANCED	
Record Project Details	Select this checkbox if you want details about how much time was needed to build each project. You can see these details when you select this build step in a completed build.
MSBuild	In some cases you might need more control over the version of MSBuild that you are running.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Should I use the Visual Studio Build task or the MSBuild task?

If you are building a solution, in most cases you should use the [Visual Studio Build task](#). This task automatically:

- Sets the `/p:VisualStudioVersion` property for you. This forces MSBuild to use a particular set of targets that increase the likelihood of a successful build.
- Specifies the MSBuild version argument.

In some cases you might need to use the [MSBuild task](#). For example, you should use it if you are building code projects apart from a solution.

Where can I learn more about MSBuild?

[MSBuild task](#)

[MSBuild reference](#)

[MSBuild command-line reference](#)

How do I build multiple configurations for multiple platforms?

1. On the Variables tab, make sure you've got variables defined for your configurations and platforms. To specify multiple values, separate them with commas.

For example, for a .NET app you could specify:

Name	Value
BuildConfiguration	debug, release
BuildPlatform	any cpu

For example, for a C++ app you could specify:

Name	Value
BuildConfiguration	debug, release
BuildPlatform	x86, x64

2. On the Options tab select **MultiConfiguration** and specify the Multipliers, separated by commas. For example: `BuildConfiguration, BuildPlatform`

Select Parallel if you want to distribute the jobs (one for each combination of values) to multiple agents in parallel if they are available.

3. On the Build tab, select this step and specify the Platform and Configuration arguments. For example:

- Platform: `$(BuildPlatform)`
- Configuration: `$(BuildConfiguration)`

Can I build TFSBuild.proj files?

You cannot build TFSBuild.proj files. These kinds of files are generated by TFS 2005 and 2008. These files contain tasks and targets are supported only using [XAML builds](#).

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Visual Studio Build task

11/6/2018 • 5 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Use this task in a build or release pipeline to build with MSBuild and set the Visual Studio version property.

Demands

msbuild, visualstudio

Azure Pipelines: If your team wants to use Visual Studio 2017 with the Microsoft-hosted agents, select **Hosted VS2017** as your default build pool. See [Microsoft-hosted agents](#).

YAML snippet

```
# Visual Studio Build
# Build with MSBuild and set the Visual Studio version property.
- task: VSBuild@1
  inputs:
    #solution: '**\*.sln'
    #vsVersion: 'latest' # Optional. Options: latest, 15.0, 14.0, 12.0, 11.0
    #msbuildArgs: # Optional
    #platform: # Optional
    #configuration: # Optional
    #clean: false # Optional
    #maximumCpuCount: false # Optional
    #restoreNugetPackages: false # Optional
    #msbuildArchitecture: 'x86' # Optional. Options: x86, x64
    #logProjectEvents: true # Optional
    #createLogFile: false # Optional
```

Arguments

ARGUMENT	DESCRIPTION
----------	-------------

ARGUMENT	DESCRIPTION
Solution	<p>If you want to build a single solution, click the ... button and select the solution.</p> <p>If you want to build multiple solutions, specify search criteria. You can use a single-folder wildcard (<code>*</code>) and recursive wildcards (<code>**</code>). For example, <code>***.sln</code> searches for all <code>.sln</code> files in all subdirectories.</p> <p>Make sure the solutions you specify are downloaded by this build pipeline. On the Repository tab:</p> <ul style="list-style-type: none"> • If you use TFVC, make sure that the solution is a child of one of the mappings on the Repository tab. • If you use Git, make sure that the project or solution is in your Git repo, and in a branch that you're building. <p>Tips:</p> <ul style="list-style-type: none"> • You can also build MSBuild project (<code>.*proj</code>) files. • If you are building a customized MSBuild project file, we recommend you use the MSBuild task instead of the Visual Studio Build task.
MSBuild Arguments	<p>You can pass additional arguments to MSBuild. For syntax, see MSBuild Command-Line Reference.</p>
Platform	<p>Specify the platform you want to build such as <code>Win32</code>, <code>x86</code>, <code>x64</code> or <code>any cpu</code>.</p> <p>Tips:</p> <ul style="list-style-type: none"> • If you are targeting an MSBuild project (<code>.*proj</code>) file instead of a solution, specify <code>AnyCPU</code> (no whitespace). • Declare a build variable such as <code>BuildPlatform</code> on the Variables tab (selecting Allow at Queue Time) and reference it here as <code>\$(BuildPlatform)</code>. This way you can modify the platform when you queue the build and enable building multiple configurations.
Configuration	<p>Specify the configuration you want to build such as <code>debug</code> or <code>release</code>.</p> <p>Tip: Declare a build variable such as <code>BuildConfiguration</code> on the Variables tab (selecting Allow at Queue Time) and reference it here as <code>\$(BuildConfiguration)</code>. This way you can modify the platform when you queue the build and enable building multiple configurations.</p>

ARGUMENT	DESCRIPTION
Clean	<p>Set to False if you want to make this an incremental build. This setting might reduce your build time, especially if your codebase is large. This option has no practical effect unless you also set Clean repository to False.</p> <p>Set to True if you want to rebuild all the code in the code projects. This is equivalent to the MSBuild <code>/target:clean</code> argument.</p>
Restore NuGet Packages	<p>(Important) This option is deprecated. Make sure to clear this checkbox and instead use the NuGet Installer build task.</p>
Visual Studio Version	<p>To avoid problems overall, you must make sure this value matches the version of Visual Studio used to create your solution.</p> <p>The value you select here adds the <code>/p:VisualStudioVersion={numeric_visual_studio_version}</code> argument to the MSBuild command run by the build. For example, if you select Visual Studio 2015, <code>/p:VisualStudioVersion=14.0</code> is added to the MSBuild command.</p> <p>Azure Pipelines: If your team wants to use Visual Studio 2017 with the Microsoft-hosted agents, select Hosted VS2017 as your default build pool. See Microsoft-hosted agents.</p>
ADVANCED	
MSBuild Architecture	<p>Select either MSBuild x86 or MSBuild x64.</p> <p>Tip: Because Visual Studio runs as a 32-bit application, you could experience problems when your build is processed by a build agent that is running the 64-bit version of Team Foundation Build Service. By selecting MSBuild x86, you might resolve these kinds of problems.</p>
Record Project Details	<p>Select this checkbox if you want details about how much time was needed to build each project. You can see these details when you select this build step in a completed build.</p>
CONTROL OPTIONS	

Open source

This task is open source on [GitHub](#). Feedback and contributions are welcome.

Q & A

Should I use the Visual Studio Build task or the MSBuild task?

If you are building a solution, in most cases you should use the [Visual Studio Build task](#). This task automatically:

- Sets the `/p:VisualStudioVersion` property for you. This forces MSBuild to use a particular set of targets

that increase the likelihood of a successful build.

- Specifies the MSBuild version argument.

In some cases you might need to use the [MSBuild task](#). For example, you should use it if you are building code projects apart from a solution.

Where can I learn more about MSBuild?

[MSBuild task](#)

[MSBuild reference](#)

[MSBuild command-line reference](#)

How do I build multiple configurations for multiple platforms?

1. On the Variables tab, make sure you've got variables defined for your configurations and platforms. To specify multiple values, separate them with commas.

For example, for a .NET app you could specify:

Name	Value
BuildConfiguration	debug, release
BuildPlatform	any cpu

For example, for a C++ app you could specify:

Name	Value
BuildConfiguration	debug, release
BuildPlatform	x86, x64

2. On the Options tab select **MultiConfiguration** and specify the Multipliers, separated by commas. For example: `BuildConfiguration, BuildPlatform`

Select Parallel if you want to distribute the jobs (one for each combination of values) to multiple agents in parallel if they are available.

3. On the Build tab, select this step and specify the Platform and Configuration arguments. For example:

- Platform: `$(BuildPlatform)`
- Configuration: `$(BuildConfiguration)`

Can I build TFSBuild.proj files?

You cannot build TFSBuild.proj files. These kinds of files are generated by TFS 2005 and 2008. These files contain tasks and targets are supported only using [XAML builds](#).

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features

are available on-premises if you have [upgraded to the latest version of TFS](#).

Xamarin.Android task

11/6/2018 • 2 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

Use this task in a build or release pipeline to build an Android app with .

Demands

AndroidSDK, MSBuild, Xamarin.Android

YAML snippet

```
# Xamarin.Android
# Build an Android app with Xamarin
- task: XamarinAndroid@1
  inputs:
    #projectFile: '**/*.csproj'
    #target: # Optional
    #outputDirectory: # Optional
    #configuration: # Optional
    #createAppPackage: true # Optional
    #clean: false # Optional
    #msbuildLocationOption: 'version' # Optional. Options: version, location
    #msbuildVersionOption: '15.0' # Optional. Options: latest, 15.0, 14.0, 12.0, 4.0
    #msbuildFile: # Required when msbuildLocationOption == Location
    #msbuildArchitectureOption: 'x86' # Optional. Options: x86, x64
    #msbuildArguments: # Optional
    #jdkOption: 'JDKVersion' # Options: jdkVersion, path
    #jdkVersionOption: 'default' # Optional. Options: default, 1.11, 1.10, 1.9, 1.8, 1.7, 1.6
    #jdkDirectory: # Required when jdkOption == Path
    #jdkArchitectureOption: 'x64' # Optional. Options: x86, x64
```

Arguments

ARGUMENT	DESCRIPTION
Project	If you want to build a single Xamarin.Android project, click the ... button and select the project. If you want to build multiple projects, specify search criteria. You can use a single-folder wildcard (<code>*</code>) and recursive wildcards (<code>**</code>). For example, <code>**/*.Android.csproj</code> searches for all Android.csproj files in all subdirectories in your repo. Note: The projects must have a PackageForAndroid target.
Target	(Optional) Specify the project targets you want to build. Use a semicolon to separate multiple targets.

ARGUMENT	DESCRIPTION
Output Directory	Use a variable to specify the folder where you want the output files to go. For example: <code>\$(build.binariesdirectory)/\$(BuildConfiguration)</code>
Configuration	Specify the configuration you want to build such as <code>debug</code> or <code>release</code> . Tip: Declare a build variable such as <code>BuildConfiguration</code> on the Variables tab (selecting Allow at Queue Time) and reference it here as <code>\$(BuildConfiguration)</code> . This way you can modify the platform when you queue the build and enable building multiple configurations.
MSBUILD OPTIONS	
MSBuild Location	(Optional) Path to MSBuild (on Windows) or xbuild (on macOS). Default behavior is to search for the latest version.
Additional Arguments	You can pass additional arguments to MSBuild (on Windows) or xbuild (on macOS). For syntax, see MSBuild Command-Line Reference .
JDK OPTIONS	
Select JDK to use for the build	Pick the JDK to be used during the build by selecting a JDK version that will be discovered during builds or by manually entering a JDK path. <ul style="list-style-type: none"> • JDK Version: Select the JDK version you want to use. • JDK Path: Specify the path to the JDK you want to use.
JDK Architecture	Select x86 or x64.
CONTROL OPTIONS	

Example

[Build your Xamarin app](#)

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Xamarin.iOS task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015

Use this task in a build or release pipeline to build an iOS app with Xamarin on macOS.

Demands

Xamarin.iOS

YAML snippet

```
# Xamarin.iOS
# Build an iOS app with Xamarin on macOS.
- task: XamariniOS@2
  inputs:
    #solutionFile: '**/*.sln'
    #configuration: 'Release'
    #clean: false # Optional
    packageApp:
    #buildForSimulator: false # Optional
    #runNugetRestore: false
    #args: # Optional
    #workingDirectory: # Optional
    #mdtoolFile: # Optional
    #signingIdentity: # Optional
    #signingProvisioningProfileID: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Solution	Click the ... button and select your solution.
Configuration	Specify a configuration such as <code>Ad-Hoc</code> , <code>AppStore</code> , <code>Debug</code> , or <code>Release</code>
Create App Package	Select if you want to create an .IPA app package file.
Build for iOS Simulator	Select if you want to build for the iOS Simulator instead of for physical iOS devices.
(OPTIONAL) SIGNING & PROVISIONING	

ARGUMENT	DESCRIPTION
Override Using (Optional)	<p>If the build should use a signing or provisioning method that is different than the default, choose that method here.</p> <p>Choose File Contents to use a P12 certificate and provisioning profile. Choose Identifiers to retrieve signing settings from the default Keychain and pre-installed profiles.</p> <p>Leave the corresponding fields blank if you do not wish to override default build settings.</p>
P12 Certificate File	Relative path to a PKCS12-formatted .p12 certificate file that contains a signing certificate to be used for this build.
P12 Password	Password to the .p12 file. <p>Important: Use a secret variable to avoid exposing this value.</p>
Provisioning Profile File	Relative path to .mobileprovision file that contains the provisioning profile override to be used for this build.
Remove Profile After Build	Select if you want the contents of the provisioning profile file to be removed from the build agent after the build is complete. <p>Important: Select only if you are running one agent per user.</p>
ADVANCED	
Arguments	(Optional) Specify additional command-line arguments for this build.
Working Directory	Working directory for the build. If you leave it blank, it is the root of the repo.
Xbuild Path	(Optional) Specify the path to xbuild . If you leave it blank, the default xbuild path is used.
CONTROL OPTIONS	

Example

[Build your Xamarin app](#)

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Xcode task

11/12/2018 • 6 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

Use this task in a build or release pipeline to build, test, or archive an Xcode workspace on macOS, and optionally package an app.

Demands

xcode

YAML snippet

```
# Xcode
# Build, test, or archive an Xcode workspace on macOS. Optionally package an app.
- task: Xcode@5
  inputs:
    #actions: 'build'
    #configuration: '$(Configuration)' # Optional
    #sdk: '$(SDK)' # Optional
    #xcWorkspacePath: '**/*.xcodeproj/project.xcworkspace' # Optional
    #scheme: # Optional
    #xcodeVersion: 'default' # Optional. Options: 8, 9, 10, default, specifyPath
    #xcodeDeveloperDir: # Optional
    packageApp:
      #archivePath: # Optional
      #exportPath: 'output/$(SDK)/$(Configuration)' # Optional
      #exportOptions: 'auto' # Optional. Options: auto, plist, specify
      #exportMethod: 'development' # Required when exportOptions == Specify
      #exportTeamId: # Optional
      #exportOptionsPlist: # Required when exportOptions == Plist
      #exportArgs: # Optional
      #signingOption: 'nosign' # Optional. Options: nosign, default, manual, auto
      #signingIdentity: # Optional
      #provisioningProfileUuid: # Optional
      #provisioningProfileName: # Optional
      #teamId: # Optional
      #destinationPlatformOption: 'default' # Optional. Options: default, iOS, tvOS, macOS, custom
      #destinationPlatform: # Optional
      #destinationTypeOption: 'simulators' # Optional. Options: simulators, devices
      #destinationSimulators: 'iPhone 7' # Optional
      #destinationDevices: # Optional
      #args: # Optional
      #workingDirectory: # Optional
      #useXcpretty: # Optional
      #publishJUnitResults: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Actions	Enter a space-delimited list of actions. Valid options are <code>build</code> , <code>clean</code> , <code>test</code> , <code>analyze</code> , and <code>archive</code> . For example, <code>clean build</code> will run a clean build. See Apple: Building from the command line with Xcode FAQ .
Configuration	Enter the Xcode project or workspace configuration to be built. The default value of this field is the variable <code>\$(Configuration)</code> . When using a variable, make sure to specify a value (for example, <code>Release</code>) on the Variables tab.

ARGUMENT	DESCRIPTION
SDK	Specify an SDK to use when building the Xcode project or workspace. From the macOS Terminal application, run <code>xcodebuild -showsdk</code> s to display the valid list of SDKs. The default value of this field is the variable <code>\$(SDK)</code> . When using a variable, make sure to specify a value (for example, <code>iphonesimulator</code>) on the Variables tab.
Workspace or project path	(Optional) Enter a relative path from the root of the repository to the Xcode workspace or project. For example, <code>MyApp/MyApp.xcworkspace</code> or <code>MyApp/MyApp.xcodeproj</code> .
Scheme	(Optional) Enter a scheme name defined in Xcode. It must be a shared scheme, with its Shared checkbox enabled under Managed Schemes in Xcode. If you specify a Workspace or project path above without specifying a scheme, and the workspace has a single shared scheme, it will be automatically used.
Xcode version	Specify the target version of Xcode. Select <code>Default</code> to use the default version of Xcode on the agent machine. Selecting a version number (e.g. <code>Xcode 10</code>) relies on environment variables being set on the agent machine for the version's location (e.g. <code>XCODE_10_DEVELOPER_DIR=/Applications/Xcode_10.0.0.app/Contents/Developer</code>). Select <code>Specify path</code> to provide a specific path to the Xcode developer directory.
Xcode developer path	(Optional) Enter a path to a specific Xcode developer directory (e.g. <code>/Applications/Xcode_10.0.0.app/Contents/Developer</code>). This is useful when multiple versions of Xcode are installed on the agent machine.
(OPTIONAL) SIGNING & PROVISIONING	
Signing style	Choose the method of signing the build. Select <code>Do not code sign</code> to disable signing. Select <code>Project defaults</code> to use only the project's signing configuration. Select <code>Manual signing</code> to force manual signing and optionally specify a signing identity and provisioning profile. Select <code>Automatic signing</code> to force automatic signing and optionally specify a development team ID. If your project requires signing, use the "Install Apple..." tasks to install certificates and provisioning profiles prior to the Xcode build.
Signing identity	(Optional) Enter a signing identity override with which to sign the build. This may require unlocking the default keychain on the agent machine. If no value is entered, the Xcode project's setting will be used.
Provisioning profile UUID	(Optional) Enter the UUID of an installed provisioning profile to be used for this build. Use separate build tasks with different schemes or targets to specify separate provisioning profiles by target in a single workspace (iOS, tvOS, watchOS).
Team ID	(Optional, unless you are a member of multiple development teams.) Specify the 10-character development team ID.
PACKAGE OPTIONS	
Create app package	Indicate whether an IPA app package file should be generated as a part of the build.
Archive path	(Optional) Specify a directory where created archives should be placed.

ARGUMENT	DESCRIPTION
Export path	(Optional) Specify the destination for the product exported from the archive.
Export options	Select a way of providing options for exporting the archive. When the default value of <code>Automatic</code> is selected, the export method is automatically detected from the archive. Select <code>Plist</code> to specify a plist file containing export options. Select <code>Specify</code> to provide a specific Export method and Team ID .
Export method	Enter the method that Xcode should use to export the archive. For example: <code>app-store</code> , <code>package</code> , <code>ad-hoc</code> , <code>enterprise</code> , or <code>development</code> .
Team ID	(Optional) Enter the 10-character team ID from the Apple Developer Portal to use during export.
Export options plist	Enter the path to the plist file that contains options to use during export.
Export arguments	(Optional) Enter additional command line arguments to be used during export.
DEVICES & SIMULATORS	
Destination platform	Select the destination device's platform to be used for UI testing when the generic build device isn't valid. Choose <code>Custom</code> to specify a platform not included in this list. When <code>Default</code> is selected, no simulators nor devices will be targeted.
Destination type	Choose the destination type to be used for UI testing. Devices must be connected to the Mac performing the build via a cable or network connection. See Devices and Simulators in Xcode.
Simulators	Enter an Xcode simulator name to be used for UI testing. For example, enter <code>iPhone X</code> (iOS and watchOS) or <code>Apple TV 4K</code> (tvOS). A target OS version is optional and can be specified in the format ' <code>OS=versionNumber</code> ', such as <code>iPhone X,OS=11.1</code> . A list of simulators installed on the Hosted macOS agent can be found here .
Devices	Enter the name of the device to be used for UI testing, such as <code>Raisa's iPad</code> . Only one device is currently supported. Note that Apple does not allow apostrophes (‘’) in device names. Instead, right single quotation marks (‘’) can be used.
ADVANCED	
Arguments	(Optional) Enter additional command line arguments with which to build. This is useful for specifying <code>-target</code> or <code>-project</code> arguments instead of specifying a workspace/project and scheme. See Apple: Building from the command line with Xcode FAQ .
Working directory	(Optional) Enter the working directory in which to run the build. If no value is entered, the root of the repository will be used.
Output directory	Enter a path relative to the working directory where build output (binaries) will be placed. The default value includes variables. When these are used, make sure to specify values on the Variables tab.

ARGUMENT	DESCRIPTION
Use xcpretty	Specify whether to use xcpretty to format xcodebuild output and generate JUnit test results. Enabling this requires xcpretty to be installed on the agent machine. It is preinstalled on Microsoft-hosted build agents. See xcpretty on GitHub.
Publish test results to Azure Pipelines/TFS	If xcpretty is enabled above, specify whether to publish JUnit test results to Azure Pipelines/TFS.
CONTROL OPTIONS	

Example

[Build your Xcode app](#)

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

Xcode Package iOS task

11/6/2018 • 2 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

Use this task in a build or release pipeline to generate an .ipa file from Xcode build output.

Deprecated

The Xcode Package iOS task has been deprecated. It is relevant only if you are using Xcode 6.4. Otherwise, use the latest version of the Xcode task.

Demands

xcode

YAML snippet

```
# Xcode Package iOS
# Generate an .ipa file from Xcode build output using xcrun (Xcode 7 or below)
- task: XcodePackageiOS@0
  inputs:
    #appName: 'name.app'
    #ipaName: 'name.ipa'
    provisioningProfile:
    #sdk: 'iphoneos'
    #appPath: '$(SDK)/$(Configuration)/build.sym/$(Configuration)-$(SDK)'
    #ipaPath: '$(SDK)/$(Configuration)/build.sym/$(Configuration)-$(SDK)/output'
```

Arguments

ARGUMENT	DESCRIPTION
Name of .app	Name of the .app file, which is sometimes different from the .ipa file.
Name of .ipa	Name of the .ipa file, which is sometimes different from the .app file.
Provisioning Profile Name	Name of the provisioning profile to use when signing.
SDK	The SDK you want to use. Run xcodetool -showsdks to see a list of valid SDK values.
ADVANCED	
Path to .app	Relative path to the built .app file. The default value is \$(SDK)/\$(Configuration)/build.sym/\$(Configuration)- \$(SDK) . Make sure to specify the variable values on the variables tab .

ARGUMENT	DESCRIPTION
Path to place .ipa	<p>Relative path where the .ipa will be placed. The directory will be created if it doesn't exist. The default value is <code>\$(SDK)/\$(Configuration)/build.sym/\$(Configuration)-\$(SDK)/output</code></p> <p>. Make sure to specify the variable values on the variables tab.</p>
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Archive Files task

11/6/2018 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

Use this task in a build or release pipeline to create an archive file from a source folder. A range of standard archive formats are supported including .zip, .jar, .war, .ear, .tar, .7z, and more.

Demands

None

YAML snippet

```
# Archive Files
# Archive files using compression formats such as .7z, .rar, .tar.gz, and .zip.
- task: ArchiveFiles@2
  inputs:
    #rootFolderOrFile: '$(Build.BinariesDirectory)'
    #includeRootFolder: true
    #archiveType: 'zip' # Options: zip, 7z, tar, wim
    #tarCompression: 'gz' # Optional. Options: gz, bz2, xz, none
    #archiveFile: '$(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip'
    #replaceExistingArchive: true
```

Arguments

ARGUMENT	DESCRIPTION
Root folder (or file) to archive	<p>The folder (or file) you wish to archive. The default file path is relative from the root folder of the repo (same as if you had specified <code>\$(Build.SourcesDirectory)</code>).</p> <p>If the specified path is a folder, recursively, all nested files and folders will be included in the archive.</p> <p>If the specified path is a file, only the single file will be included in the archive.</p>
Prefix root folder name to archive paths	<p>If selected, the root folder name will be prefixed to file paths within the archive. Otherwise, all file paths will start one level lower.</p> <p>For example, suppose the selected root folder is <code>/home/user/output/classes/</code>, and contains: <code>com/acme/Main.class</code>.</p> <ul style="list-style-type: none">• If selected, the resulting archive would contain: <code>classes/com/acme/Main.class</code>.• Otherwise, the resulting archive would contain: <code>com/acme/Main.class ..</code>

ARGUMENT	DESCRIPTION
Archive type	<p>Specify the compression scheme used. To create <code>foo.jar</code>, for example, choose <code>zip</code> for the compression, and specify <code>foo.jar</code> as the archive file to create. For all tar files (including compressed ones), choose <code>tar</code>.</p> <ul style="list-style-type: none"> • <code>zip</code> - default, zip format, choose this for all zip compatible types, (.zip, .jar, .war, .ear) • <code>7z</code> - 7-Zip format, (.7z) • <code>tar</code> - tar format, choose this for compressed tars, (.tar.gz, .tar.bz2, .tar.xz) • <code>wim</code> - wim format, (.wim)
Tar compression	<p>Only applicable if the <code>tar</code> archive type is selected.</p> <p>Optionally choose a compression scheme, or choose <code>None</code> to create an uncompressed tar file.</p> <ul style="list-style-type: none"> • <code>gz</code> - default, gzip compression (.tar.gz, .tar.tgz, .taz) • <code>bz2</code> - bzip2 compression (.tar.bz2, .tz2, .tbz2) • <code>xz</code> - xz compression (.tar.xz, .txz) • <code>None</code> - no compression, choose this to create a uncompressed tar file (.tar)
Archive file to create	<p>Specify the name of the archive file to create. The file extension should match the selected archive type. For example to create <code>foo.tgz</code>, select the <code>tar</code> archive type, <code>gz</code> for tar compression.</p>
Replace existing archive	<p>If an existing archive exists, specify whether to overwrite it. Otherwise, files will be added to it as long as it is not a compressed tar.</p> <p>If adding to an existing archive, these types are supported:</p> <ul style="list-style-type: none"> • <code>zip</code> • <code>7z</code> • <code>tar</code> - uncompressed only • <code>wim</code>
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Q: I'm having problems. How can I troubleshoot them?

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

Q: How do variables work? What variables are available for me to use in the arguments?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See [Variables](#).

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Azure Network Load Balancer task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to connect or disconnect an Azure virtual machine's network interface to a load balancer's address pool.

YAML snippet

```
# Azure Network Load Balancer
# Connect/Disconnect an Azure virtual machine's network interface to a Load Balancer's backend address pool
- task: AzureNLBManagement@1
  inputs:
    azureSubscription:
    resourceGroupName:
    loadBalancer:
    action: # Options: disconnect, connect
```

Arguments

ARGUMENT	DESCRIPTION
Azure Subscription	(Required) Select the Azure Resource Manager subscription for the deployment.
Resource Group	(Required) Select the resource group name.
Load Balancer Name	(Required) Select or enter the load balancer.
Action	(Required) Disconnect: Removes the virtual machine's primary network interface from the load balancer's backend pool. So that it stops receiving network traffic. Connect: Adds the virtual machine's primary network interface to load balancer backend pool. So that it starts receiving network traffic based on the load balancing rules for the load balancer resource.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Bash task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to run a Bash script on macOS, Linux, or Windows.

YAML snippet

```
# Bash
# Run a Bash script on macOS, Linux, or Windows
- task: Bash@3
  inputs:
    targetType: 'filePath' # Optional. Options: filePath, inline
    filePath: # Required when targetType == FilePath
    arguments: # Optional
    script: '# Write your commands here# Use the environment variables input below to pass secret variables to this script' # Required when targetType == Inline
    workingDirectory: # Optional
    failOnStderr: false # Optional
```

The Bash task also has a shortcut syntax in YAML:

```
- bash: # script path or inline
  workingDirectory: #
  displayName: #
  failOnStderr: #
  env: # mapping of environment variables to add
```

Arguments

ARGUMENT	DESCRIPTION
Type	Sets whether this is an inline script or a path to a .sh file
File path	Path of the script to execute. Must be a fully qualified path or relative to \$(System.DefaultWorkingDirectory). Required if Type is <code>filePath</code> .
Arguments	Arguments passed to the Bash script.
Script	Contents of the script. Required if Type is <code>inline</code>
Working Directory	Specify the working directory in which you want to run the command. If you leave it empty, the working directory is <code>\$(Build.SourcesDirectory)</code> .
Fail on standard error	If this is <code>true</code> , this task will fail if any errors are written to <code>stderr</code> .

ARGUMENT	DESCRIPTION
Env[ironment variables]	A list of additional items to map into the process's environment. For example, secret variables are not automatically mapped. If you have a secret variable called <code>Foo</code> , you can map it in like this: <pre>yaml - script: echo \$MYSECRET env: MySecret: \$(Foo)</pre>
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Batch Script task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015

Use this task in a build or release pipeline to run a Windows .bat or .cmd script and optionally allow it to change the stage.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

YAML snippet

```
# Batch Script
# Run a windows cmd or bat script and optionally allow it to change the environment
- task: BatchScript@1
  inputs:
    filename:
    #arguments: # Optional
    #modifyEnvironment: False # Optional
    #workingFolder: # Optional
    #failOnStandardError: false # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Path	Specify the path to the .bat or .cmd script you want to run. The path must be a fully qualified path or a valid path relative to the default working directory. In Team Foundation Build, this directory is \$(Build.SourcesDirectory) .
Arguments	Specify arguments to pass to the script.
Modify environment	Select this check box if you want stage variable modifications in the script to affect subsequent tasks.

ADVANCED

Working folder	Specify the working directory in which you want to run the script. If you leave it empty, the working directory is the folder where the script is located.
Fail on standard error	Select this check box if you want the build to fail if errors are written to the StandardError stream.

CONTROL OPTIONS

Example

Create `test.bat` at the root of your repo:

```
@echo off
echo Hello World from %AGENT_NAME%.
echo My ID is %AGENT_ID%.
echo AGENT_WORKFOLDER contents:
@dir %AGENT_WORKFOLDER%
echo AGENT_BUILDDIRECTORY contents:
@dir %AGENT_BUILDDIRECTORY%
echo BUILD_SOURCESDIRECTORY contents:
@dir %BUILD_SOURCESDIRECTORY%
echo Over and out.
```

On the Build tab of a build pipeline, add this task:



Utility: Batch Script

Run test.bat.

- Path: `test.bat`

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Where can I learn Windows commands?

[An A-Z Index of the Windows CMD command line](#)

How do I set a variable so that it can be read by subsequent scripts and tasks?

[Define and modify your build variables in a script](#)

[Define and modify your release variables in a script](#)

Q: I'm having problems. How can I troubleshoot them?

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

Q: How do variables work? What variables are available for me to use in the arguments?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See [Variables](#).

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Command Line task

11/6/2018 • 2 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

Use this task in a build or release pipeline to run a program from the command prompt.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Demands

None

YAML snippet

```
# Command Line
# Run a command line script using cmd.exe on Windows and bash on macOS and Linux.
- task: CmdLine@2
  inputs:
    #script: 'echo Write your commands here.'
    #workingDirectory: # Optional
    #failOnStderr: false # Optional
```

The CmdLine task also has a shortcut syntax in YAML:

```
- script: # script path or inline
  workingDirectory: #
  displayName: #
  failOnStderr: #
  env: # mapping of environment variables to add
```

Arguments

ARGUMENT	DESCRIPTION
Script	Contents of the script you want to run
OPTIONAL	
Working directory	Specify the working directory in which you want to run the command. If you leave it empty, the working directory is \$(Build.SourcesDirectory) .
Fail on standard error	If this is <code>true</code> , this task will fail if any errors are written to <code>stderr</code> .

ARGUMENT	DESCRIPTION
Env[ironment variables]	A list of additional items to map into the process's environment. For example, secret variables are not automatically mapped. If you have a secret variable called <code>Foo</code> , you can map it in like this:
CONTROL OPTIONS	

Example

- [YAML](#)
- [Designer](#)

```
steps:
- script: date /t
  displayName: Get the date
- script: dir
  workingDirectory: $(Agent.BuildDirectory)
  displayName: List contents of a folder
- script: |
  set MYVAR=foo
  set
  displayName: Set a variable and then display all
```

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Where can I learn Windows commands?

[An A-Z Index of the Windows CMD command line](#)

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Copy and Publish Build Artifacts task

11/19/2018 • 2 minutes to read • [Edit Online](#)

TFS 2015

Use this task in a build or release pipeline to copy build artifacts to a staging folder and then publish them to the server or a file share. Files are copied to the `$(Build.ArtifactStagingDirectory)` staging folder and then published.

IMPORTANT

If you're using Azure Pipelines, or Team Foundation Server (TFS) 2017 or newer, we recommend that you do NOT use this deprecated task. Instead, use the **Copy Files** and **Publish Build Artifacts** tasks. See [Artifacts in Azure Pipelines](#).

IMPORTANT

Are you using Team Foundation Server (TFS) 2015.4? If so, we recommend that you do NOT use this deprecated task. Instead, use the **Copy Files** and **Publish Build Artifacts** tasks. See [Artifacts in Azure Pipelines](#).

You should use this task only if you're using Team Foundation Server (TFS) 2015 RTM. In that version of TFS, this task is listed under the **Build** category and is named **Publish Build Artifacts**.

Demands

None

Arguments

ARGUMENT	DESCRIPTION
Copy Root	<p>Folder that contains the files you want to copy. If you leave it empty, the copying is done from the root folder of the repo (same as if you had specified <code>\$(Build.SourcesDirectory)</code>).</p> <p>If your build produces artifacts outside of the sources directory, specify <code>\$(Agent.BuildDirectory)</code> to copy files from the build agent working directory.</p>
Contents	<p>Specify pattern filters (one on each line) that you want to apply to the list of files to be copied. For example:</p> <ul style="list-style-type: none">• <code>**</code> copies all files in the root folder.• <code>***</code> copies all files in the root folder and all files in all sub-folders.• <code>**\bin</code> copies files in any sub-folder named bin.
Artifact Name	Specify the name of the artifact. For example: <code>drop</code>

ARGUMENT	DESCRIPTION
Artifact Type	Choose server to store the artifact on your Team Foundation Server. This is the best and simplest option in most cases. See Artifacts in Azure Pipelines .
CONTROL OPTIONS	

Q & A

Q: This step didn't produce the outcome I was expecting. How can I fix it?

This task has a couple of known issues:

- Some minimatch patterns don't work.
- It eliminates the most common root path for all paths matched.

You can avoid these issues by instead using the [Copy Files task](#) and the [Publish Build Artifacts task](#).

Q: I'm having problems. How can I troubleshoot them?

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

Q: How do variables work? What variables are available for me to use in the arguments?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See [Variables](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Copy Files task

11/19/2018 • 4 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015.3

Use this task in a build or release pipeline to copy files from a source folder to a target folder using match patterns.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Demands

None

YAML snippet

```
# Copy Files
# Copy files from source folder to target folder using match patterns (The match patterns will only match
# file paths, not folder paths)
- task: CopyFiles@2
  inputs:
    #sourceFolder: # Optional
    #contents: '**'
    targetFolder:
    #cleanTargetFolder: false # Optional
    #overWrite: false # Optional
    #flattenFolders: false # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Source Folder	<p>Folder that contains the files you want to copy. If you leave it empty, the copying is done from the root folder of the repo (same as if you had specified <code>\$(Build.SourcesDirectory)</code>).</p> <p>If your build produces artifacts outside of the sources directory, specify <code>\$(Agent.BuildDirectory)</code> to copy files from the directory created for the pipeline.</p>

ARGUMENT	DESCRIPTION
Contents	<p>Specify match pattern filters (one on each line) that you want to apply to the list of files to be copied. For example:</p> <ul style="list-style-type: none"> • <code>*</code> copies all files in the root folder. • <code>***</code> copies all files in the root folder and all files in all sub-folders. • <code>**\bin**</code> copies all files recursively from any <code>bin</code> folder. <p>The pattern is used to match only file paths, not folder paths. So you should specify patterns such as <code>**\bin**</code> instead of <code>**\bin</code>.</p> <p>More examples are shown below.</p>
Target Folder	<p>Folder where the files will be copied. In most cases you specify this folder using a variable. For example, specify <code>\$(Build.ArtifactStagingDirectory)</code> if you intend to publish the files as build artifacts.</p>
ADVANCED	
Clean Target Folder	Select this check box to delete all existing files in the target folder before beginning to copy.
Overwrite	Select this check box to replace existing files in the target folder.
CONTROL OPTIONS	

Notes

If no files are matched, the task will still report success. If a matched file already exists in the target, the task will report failure unless Overwrite is set to true.

Usage

A typical pattern for using this task is:

- Build something
- Copy build outputs to a staging directory
- Publish staged artifacts

For example:

```

steps:
- powershell: .\build.ps1
- task: CopyFiles@2
  inputs:
    contents: _buildOutput\**
    targetFolder: $(Build.ArtifactStagingDirectory)
- task: PublishBuildArtifacts@1
  inputs:
    pathToPublish: $(Build.ArtifactStagingDirectory)
    artifactName: MyBuildOutputs
  
```

Examples

Copy executables and a readme file

Goal

You want to copy just the readme and the files needed to run this C# console app:

```
-- ConsoleApplication1
|-- ConsoleApplication1.sln
|-- readme.txt
`-- ClassLibrary1
    |-- ClassLibrary1.csproj
`-- ClassLibrary2
    |-- ClassLibrary2.csproj
`-- ConsoleApplication1
    |-- ConsoleApplication1.csproj
```

Note: *ConsoleApplication1.sln* contains a *bin* folder with .dll and .exe files, see the Results below to see what gets moved!

On the Variables tab, `$(BuildConfiguration)` is set to `release`.

- [YAML](#)
- [Designer](#)

Example with multiple match patterns:

```
steps:
- task: CopyFiles@2
  displayName: 'Copy Files to: $(Build.ArtifactStagingDirectory)'
  inputs:
    Contents: |
      ConsoleApplication1\ConsoleApplication1\bin\**\*.exe
      ConsoleApplication1\ConsoleApplication1\bin\**\*.dll
      ConsoleApplication1\readme.txt
    TargetFolder: '$(Build.ArtifactStagingDirectory)'
```

Example with OR condition:

```
steps:
- task: CopyFiles@2
  displayName: 'Copy Files to: $(Build.ArtifactStagingDirectory)'
  inputs:
    Contents: |
      ConsoleApplication1\ConsoleApplication1\bin\**\?(*.exe|*.dll)
      ConsoleApplication1\readme.txt
    TargetFolder: '$(Build.ArtifactStagingDirectory)'
```

Example with NOT condition:

```

steps:
- task: CopyFiles@2
  displayName: 'Copy Files to: $(Build.ArtifactStagingDirectory)'
  inputs:
    Contents: |
      ConsoleApplication1\**\bin\**\!(*.pdb|*.config)
      !ConsoleApplication1\**\ClassLibrary\**
      ConsoleApplication1\readme.txt
    TargetFolder: '$(Build.ArtifactStagingDirectory)'
```

YAML builds are not yet available on TFS.

Results

These files are copied to the staging directory:

```

`-- ConsoleApplication1
  |-- readme.txt
  '-- ConsoleApplication1
    '-- bin
      '-- Release
        | -- ClassLibrary1.dll
        | -- ClassLibrary2.dll
        | -- ConsoleApplication1.exe
```

Copy everything from the source directory except the .git folder

- [YAML](#)
- [Designer](#)

Example with multiple match patterns:

```

steps:
- task: CopyFiles@2
  displayName: 'Copy Files to: $(Build.ArtifactStagingDirectory)'
  inputs:
    SourceFolder: '$(Build.SourcesDirectory)'
    Contents: |
      **/*
      !.git/**/*
    TargetFolder: '$(Build.ArtifactStagingDirectory)'
```

YAML builds are not yet available on TFS.

Open source

This task is open source on [GitHub](#). Feedback and contributions are welcome.

Q & A

Where can I learn more about file matching patterns?

[File matching patterns reference](#)

How do I use this task to publish artifacts?

See [Artifacts in Azure Pipelines](#).

Q: I'm having problems. How can I troubleshoot them?

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.

2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

Q: How do variables work? What variables are available for me to use in the arguments?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See [Variables](#).

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

cURL Upload Files task

11/6/2018 • 2 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

Use this task in a build or release pipeline to use [cURL](#) to upload files with supported protocols such as FTP, FTPS, SFTP, HTTP, and more.

Demands

curl

YAML snippet

```
# cURL Upload Files
# Use cURL to upload files.
- task: cURLUploader@2
  inputs:
    files:
      #authType: 'ServiceEndpoint' # Optional. Options: serviceEndpoint, userAndPass
      #serviceEndpoint: # Required when authType == ServiceEndpoint
      #username: # Optional
      #password: # Optional
      #url: # Required when authType == UserAndPass
      #remotePath: 'upload/${Build.BuildId}' # Optional
      #options: # Optional
      #redirectStderr: true # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Files	If you want to upload a single file, click the ... button and select the file. If you want to upload multiple files, specify a minimatch pattern filter. For example, specify <code>***.zip</code> to upload all ZIP files in all sub-folders.
Username	Specify the username for server authentication.
Password	Specify the password for server authentication. Important: Use a secret variable to avoid exposing this value.

ARGUMENT	DESCRIPTION
URL	<p>URL to the location where you want to upload the files. If you are uploading to a folder, make sure to end the argument with a trailing slash.</p> <p>Acceptable URL protocols include <code>DICT://</code>, <code>FILE://</code>, <code>FTP://</code>, <code>FTPS://</code>, <code>GOPHER://</code>, <code>HTTP://</code>, <code>HTTPS://</code>, <code>IMAP://</code>, <code>IMAPS://</code>, <code>LDAP://</code>, <code>LDAPS://</code>, <code>POP3://</code>, <code>POP3S://</code>, <code>RTMP://</code>, <code>RTSP://</code>, <code>SCP://</code>, <code>SFTP://</code>, <code>SMTP://</code>, <code>SMTPS://</code>, <code>TELNET://</code>, and <code>TFTP://</code>.</p>
Optional Arguments	Arguments to pass to cURL.
ADVANCED	
Redirect Standard Error to Standard Out	<p>In most cases you should leave this selected.</p> <p>Select if you want to add <code>--stderr</code> - as an argument to cURL. Otherwise, if you clear this check box, cURL will write its progress bar to stderr, which is interpreted by the build pipeline as error output, which could cause the build to fail.</p>
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Where can I learn more about file matching patterns?

[File matching patterns reference](#)

Where can I learn FTP commands?

[List of raw FTP commands](#)

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Decrypt File (OpenSSL) task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to decrypt files using OpenSSL.

YAML snippet

```
# Decrypt File (OpenSSL)
# A thin utility task for file decryption using OpenSSL.
- task: DecryptFile@1
  inputs:
    #cipher: 'des3'
    inFile:
    passphrase:
    #outFile: # Optional
    #workingDirectory: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Cypher	(Required) Encryption cypher to use. See cipher suite names for a complete list of possible values.
Encrypted file	(Required) Relative path of file to decrypt.
Passphrase	(Required) Passphrase to use for decryption. Use a Variable to encrypt the passphrase.
Decrypted file path	(Optional) Optional filename for decrypted file. Defaults to the Encrypted File with a ".out" extension
Working directory	(Optional) Working directory for decryption. Defaults to the root of the repository.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Delay task

11/15/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to pause execution of the pipeline for a fixed delay time.

Demands

Can be used in only an [agentless job](#) of a release pipeline.

YAML snippet

```
# Delay
# Delay further execution of the workflow by a fixed time.
- task: Delay@1
  inputs:
    #delayForMinutes: '0'
```

Arguments

PARAMETER	COMMENTS
Display name	Required. The name to display for this task.
Delay Time (minutes)	Required. The number of minutes to delay execution.
Control options	See Control options

Also see this task on [GitHub](#).

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

Delete Files task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015.3

Use this task in a build or release pipeline to delete files or folders from the agent working directory.

Demands

None

YAML snippet

```
# Delete Files
# Delete files or folders. (The minimatch patterns will only match file paths, not folder paths)
- task: DeleteFiles@1
  inputs:
    #sourceFolder: # Optional
    #contents: 'myFileShare'
```

Arguments

ARGUMENT	DESCRIPTION
Source Folder	<p>Folder that contains the files you want to delete. If you leave it empty, the deletions are done from the root folder of the repo (same as if you had specified <code>\$(Build.SourcesDirectory)</code>).</p> <p>If your build produces artifacts outside of the sources directory, specify <code>\$(Agent.BuildDirectory)</code> to delete files from the build agent working directory.</p>
Contents	<p>Specify minimatch pattern filters (one on each line) that you want to apply to the list of files to be deleted. For example:</p> <ul style="list-style-type: none">• <code>**</code> deletes all files and folders in the root folder.• <code>temp</code> deletes the temp folder in the root folder.• <code>temp*</code> deletes any file or folder in the root folder with a name that begins with temp.• <code>**\temp**</code> deletes all files in any sub-folder named temp.• <code>**\temp*</code> deletes any file or folder with a name that begins with temp.• <code>**\temp***</code> deletes files in any sub-folder that begins with the name temp.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Q: What's a minimatch pattern? How does it work?

A: See:

- <https://github.com/isaacs/minimatch>
- <https://realguess.net/tags/minimatch/>

Q: I'm having problems. How can I troubleshoot them?

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

Q: How do variables work? What variables are available for me to use in the arguments?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See [Variables](#).

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Download Build Artifacts task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to download build artifacts.

YAML snippet

```
# Download Build Artifacts
# Download Build Artifacts
- task: DownloadBuildArtifacts@0
  inputs:
    #buildType: 'current' # Options: current, specific
    #project: # Required when buildType == Specific
    #pipeline: # Required when buildType == Specific
    #specificBuildWithTriggering: false # Optional
    #buildVersionToDownload: 'latest' # Required when buildType == Specific# Options: latest,
    latestFromBranch, specific
    #branchName: 'refs/heads/master' # Required when buildType == Specific && BuildVersionToDownload ==
    LatestFromBranch
    #buildId: # Required when buildType == Specific && BuildVersionToDownload == Specific
    #tags: # Optional
    #downloadType: 'single' # Options: single, specific
    #artifactName: # Required when downloadType == Single
    #itemPattern: '**' # Optional
    #downloadPath: '$(System.ArtifactsDirectory)'
    #parallelizationLimit: '8' # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Download artifacts produced by	(Required) Download artifacts produced by the current build, or from a specific build.
Project	(Required) The project from which to download the build artifacts
Build pipeline	(Required) Select the build pipeline name
When appropriate, download artifacts from the triggering build.	(Optional) If checked, this build task will try to download artifacts from the triggering build. If there is no triggering build from the specified pipeline, it will download artifacts from the build specified in the options below.
Build version to download	(Required) undefined
Branch name	(Required) Specify to filter on branch/ref name, for example: <code>refs/heads/develop</code> .
Build	(Required) The build from which to download the artifacts

ARGUMENT	DESCRIPTION
Download type	(Required) Download a specific artifact or specific files from the build.
Artifact name	(Required) The name of the artifact to download
Matching pattern	(Optional) Specify files to be downloaded as multi line minimatch pattern. More Information The default pattern (**) will download all files across all artifacts in the build. To download all files within artifact drop use drop/**.
Destination directory	(Required) Path on the agent machine where the artifacts will be downloaded
Parallelization limit	(Optional) Number of files to download simultaneously

CONTROL OPTIONS

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Download Fileshare Artifacts task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to download fileshare artifacts.

YAML snippet

```
# Download Fileshare Artifacts
# Download artifacts from a file share e.g \\share\drop
- task: DownloadFileshareArtifacts@1
  inputs:
    filesharePath:
    artifactName:
    #itemPattern: '**' # Optional
    #downloadPath: '$(System.ArtifactsDirectory)'
    #parallelizationLimit: '8' # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Fileshare path	(Required) Example <code>\server\folder</code>
Artifact name	(Required) The name of the artifact to download.
Matching pattern	(Optional) Specify files to be downloaded as multiline minimatch patterns. More Information . The default pattern (<code>**</code>) will download all files within the artifact.
Download path	(Required) Path on the agent machine where the artifacts will be downloaded.
Parallelization limit	(Optional) Number of files to download simultaneously.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Download Package task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to download a package from a package management feed in Azure Artifacts or TFS. Requires the Package Management extension.

NOTE

This task currently only supports downloading NuGet packages.

YAML snippet

```
# Download Package
# Download a package from a Package Management feed in Azure Artifacts or TFS. Requires the Package Management extension.
- task: DownloadPackage@0
  inputs:
    feed:
    definition:
    version:
    #downloadPath: '$(System.ArtifactsDirectory)'
```

Arguments

ARGUMENT	DESCRIPTION
Feed	(Required) Select the package source
Package	(Required) Select the package to download, only NuGet packages are supported currently
Version	(Required) Version of the package
Destination directory	(Required) Path on the agent machine where the package will be downloaded

CONTROL OPTIONS

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Download Pipeline Artifact task

11/19/2018 • 2 minutes to read • [Edit Online](#)

Use this task in a build or release pipeline to download pipeline artifacts from earlier stages in this pipeline, or from another pipeline.

YAML snippet

```
# Download Pipeline Artifact
- task: DownloadPipelineArtifact@0
  inputs:
    #pipelineId: # Optional
    #artifactName: 'drop'
    targetPath:
```

Arguments

ARGUMENT	DESCRIPTION
targetPath	The target path where the contents of the pipeline artifact will be placed. See Artifacts in Azure Pipelines .
artifactName	The name of the artifact that you want to download.
pipelineId	The ID of the Pipeline to download from. For example: <code>1764</code>
Control options	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Download Secure File task

10/25/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to download a secure file to a temporary location on the build or release agent.

Use this task to download a [secure file](#) from the server during a build or release.

Once downloaded, the secure file is located in the `$(Agent.TempDirectory)` directory of the Azure Pipelines Agent.

The full path of the downloaded file is stored to the `$env:DOWNLOADSECUREFILE_SECUREFILEPATH` environment variable.

If you use multiple versions of the Download Secure File task in your pipeline, they can be referenced with the `$env:DOWNLOADSECUREFILE1_SECUREFILEPATH`, `$env:DOWNLOADSECUREFILE2_SECUREFILEPATH`, ... environment variables, where the number in the environment variable corresponds with the task version.

Note that if you use two Download Secure File tasks in the same pipeline with the same task version, the `$env:DOWNLOADSECUREFILE_SECUREFILEPATH` environment variable will not be populated, but both files will still be downloaded to `$(Agent.TempDirectory)`.

YAML snippet

```
# Download Secure File
# Download a secure file to a temporary location on the build or release agent
- task: DownloadSecureFile@1
  inputs:
    secureFile:
```

Arguments

ARGUMENT	DESCRIPTION
Secure File	Select the secure file to download to a temporary location on the agent machine. The file will be deleted after the build or release.

Extract Files task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

Use this task in a build or release pipeline to extract files from archives to a target folder using match patterns. A range of standard archive formats is supported, including .zip, .jar, .war, .ear, .tar, .7z, and more.

Demands

None

YAML snippet

```
# Extract Files
# Extract a variety of archive and compression files such as .7z, .rar, .tar.gz, and .zip.
- task: ExtractFiles@1
  inputs:
    #archiveFilePatterns: '*.zip'
    destinationFolder:
    #cleanDestinationFolder: true
```

Arguments

ARGUMENT	DESCRIPTION
Archive file patterns	<p>The archives you want to extract. The default file path is relative from the root folder of the repo (same as if you had specified <code>\$(Build.SourcesDirectory)</code>).</p> <p>Specify match pattern filters (one on each line) that you want to apply to identify the list of archives to extract. For example:</p> <ul style="list-style-type: none">• <code>test.zip</code> extracts the test.zip file to the root folder.• <code>test*.zip</code> extracts all .zip files in the test folder.• <code>***.tar</code> extracts all .tar files in the root folder and sub-folders.• <code>**\bin*.7z</code> extracts all ".7z" files in any sub-folder named bin. <p>The pattern is used to match only archive file paths, not folder paths, and not archive contents to be extracted. So you should specify patterns such as <code>**\bin**</code> instead of <code>**\bin</code>.</p>
Destination folder	Folder where the archives will be extracted. The default file path is relative to the root folder of the repo (same as if you had specified <code>\$(Build.SourcesDirectory)</code>).
Clean destination folder before extracting	Select this check box to delete all existing files in the destination folder before beginning to extract archives.

ARGUMENT	DESCRIPTION
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Where can I learn more about file matching patterns?

[File matching patterns reference](#)

Q: I'm having problems. How can I troubleshoot them?

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

Q: How do variables work? What variables are available for me to use in the arguments?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See [Variables](#).

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

FTP Upload task

11/15/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

Use this task in a build or release pipeline to upload files to a remote machine using the File Transfer Protocol (FTP), or securely with FTPS.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Demands

None

YAML snippet

```
# FTP Upload
# FTP Upload
- task: FtpUpload@1
  inputs:
    #credentialsOption: 'serviceEndpoint' # Options: serviceEndpoint, inputs
    #serverEndpoint: # Required when credentialsOption == ServiceEndpoint
    #serverUrl: # Required when credentialsOption == Inputs
    #username: # Required when credentialsOption == Inputs
    #password: # Required when credentialsOption == Inputs
    rootDirectory:
    #filePatterns: '**'
    #remoteDirectory: '/upload/${Build.BuildId}/'
    #clean: false
    #cleanContents: false # Required when clean == False
    #overwrite: true
    #preservePaths: false
    #trustSSL: false
```

Arguments

ARGUMENT	DESCRIPTION
----------	-------------

ARGUMENT	DESCRIPTION
FTP service connection	Select the service connection for your FTP server. To create one, click the Manage link and create a new Generic service connection, enter the FTP server URL for the server URL, e.g. <code>ftp://server.example.com</code> , and required credentials. Secure connections will always be made regardless of the specified protocol (<code>ftp://</code> or <code>ftps://</code>) if the target server supports FTPS. To allow only secure connections, use the <code>ftps://</code> protocol, e.g. <code>ftps://server.example.com</code> . Connections to servers not supporting FTPS will fail if <code>ftps://</code> is specified.
Source folder	The source folder to upload files from. The default file path is relative from the root folder of the repo (same as if you had specified <code>\$(Build.SourcesDirectory)</code>).
File patterns	File paths or patterns of the files to upload. Supports multiple lines of match patterns. To upload the entire folder content recursively, specify <code>**</code> .
Remote directory	Upload files to this directory on the remote FTP server.
Clean remote directory	Recursively delete all contents of the remote directory before uploading.
Overwrite	Overwrite existing files in the remote directory.
Trust server certificate	Selecting this option results in the FTP server's SSL certificate being trusted with <code>ftps://</code> , even if it is self-signed or cannot be validated by a Certificate Authority (CA).
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Where can I learn more about file matching patterns?

[File matching patterns reference](#)

Q: I'm having problems. How can I troubleshoot them?

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

Q: How do variables work? What variables are available for me to use in the arguments?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See [Variables](#).

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

GitHub Release

11/26/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in your pipeline to create, edit, or discard a [GitHub release](#).

Prerequisites

GitHub service connection

This task requires a [GitHub service connection](#) with **Write** permission to the GitHub repository. You can create a GitHub service connection in your Azure Pipelines project. Once created, use the name of the service connection in this task's settings.

YAML snippet

```
# GitHub Release
# Create, edit, or discard a GitHub release.
- task: GithubRelease@0
  inputs:
    gitHubConnection:
    repositoryName:
    #action: 'create' # Options: create, edit, discard
    #target: '${{build.sourceVersion}}' # Required when action == create || action == edit
    #tagSource: 'auto' # Required when action == create. Options: auto, manual
    #tag: # Required when action == edit || action == discard || tagSource == manual
    #title: # Optional
    #releaseNotesSource: 'file' # Optional. Options: file, input
    #releaseNotesFile: # Optional
    #releaseNotes: # Optional
    #assets: '$(build.artifactStagingDirectory)/*' # Optional
    #assetUploadMode: 'delete' # Optional. Options: delete, replace
    #isDraft: false # Optional
    #isPreRelease: false # Optional
    #addChangeLog: true # Optional
```

Arguments

ARGUMENT	DESCRIPTION
GitHub Connection	(Required) Enter the service connection name for your GitHub connection. Learn more about service connections here .
Repository	(Required) Select the name of GitHub repository in which GitHub releases will be created.
Action	(Required) Select the type of release operation you want perform. This task can create, edit, or discard a GitHub release.
Target	(Required) This is the commit SHA for which the GitHub release will be created. E.g. <code>48b11d8d6e92a22e3e9563a3f643699c16fd6e27</code> . You can also use variables here.

ARGUMENT	DESCRIPTION
Tag source	(Required) Configure the tag to be used for release creation. The 'Git tag' option automatically takes the tag which is associated with this commit. Use the 'User specified tag' option in case you want to manually provide a tag.
Tag	(Required) Specify the tag for which you want to create, edit, or discard a release. You can also use variables here. E.g. <code>\$(tagName)</code> .
Release title	(Optional) Specify the title of the GitHub release. If left empty, the tag will be used as the release title.
Release notes source	(Optional) Specify the description of the GitHub release. Use the 'Release notes file' option to use the contents of a file as release notes. Use the 'Inline release notes' option to manually enter the release notes.
Release notes file path	(Optional) Select the file which contains the release notes.
Release notes	(Optional) Type your release notes here. Markdown is supported.
Assets	(Optional) Specify the files to be uploaded as assets for the release. You can use wildcard characters to specify a set of files. E.g. <code>\$(Build.ArtifactStagingDirectory)/*.zip</code> . You can also specify multiple patterns - one per line. By default, all files in the <code>\$(Build.ArtifactStagingDirectory)</code> directory will be uploaded.
Asset upload mode	(Optional) Use the 'Delete existing assets' option to first delete any existing assets in the release and then upload all assets. Use the 'Replace existing assets' option to replace any assets that have the same name.
Draft release	(Optional) Indicate whether the release should be saved as a draft (unpublished). If <code>false</code> , the release will be published.
Pre-release	(Optional) Indicate whether the release should be marked as a pre-release.
Add changelog	(Optional) If set to <code>true</code> , a list of changes (commits and issues) between this and the last published release will be generated and appended to release notes.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q&A

Install Apple Certificate task

10/25/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018

Use this task in a build or release pipeline to install an Apple certificate that is required to build on a macOS agent. You can use this task to install an Apple certificate that is stored as a [secure file](#) on the server.

Demands

xcode

YAML snippet

```
# Install Apple Certificate
# Install an Apple certificate required to build on a macOS agent
- task: InstallAppleCertificate@2
  inputs:
    certSecureFile:
    #certPwd: # Optional
    #keychain: 'temp' # Options: default, temp, custom
    #keychainPassword: # Required when keychain == Custom || Keychain == Default
    #customKeychainPath: # Required when keychain == Custom
    #deleteCert: # Optional
    #deleteCustomKeychain: # Optional
    #signingIdentity: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Certificate (P12)	Select the certificate (.p12) that was uploaded to Secure Files to install on the macOS agent.
Certificate (P12) Password	Password to the Apple certificate (.p12). Use a new build variable with its lock enabled on the Variables tab to encrypt this value.
Advanced - Keychain	Select the keychain in which to install the Apple certificate. You can choose to install the certificate in a temporary keychain (default), the default keychain or a custom keychain. A temporary keychain will always be deleted after the build or release is complete.
Advanced - Keychain Password	Password to unlock the keychain. Use a new build variable with its lock enabled on the Variables tab to encrypt this value. A password is generated for the temporary keychain if not specified.
Advanced - Delete Certificate from Keychain	Select to delete the certificate from the keychain after the build or release is complete. This option is visible when custom keychain or default keychain are selected.

ARGUMENT	DESCRIPTION
Advanced - Custom Keychain Path	Full path to a custom keychain file. The keychain will be created if it does not exist. This option is visible when a custom keychain is selected.
Advanced - Delete Custom Keychain	Select to delete the custom keychain from the agent after the build or release is complete. This option is visible when a custom keychain is selected.

Install Apple Provisioning Profile task

10/25/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018

Use this task in a build or release pipeline to install an Apple provisioning profile that is required to build on a macOS agent. You can use this task to install provisioning profiles needed to build iOS Apps, Apple WatchKit Apps and App Extensions.

You can install an Apple provisioning profile that is:

- Stored as a [secure file](#) on the server.
- (**Azure Pipelines**) Committed to the source repository or copied to a local path on the macOS agent. We recommend encrypting the provisioning profiles if you are committing them to the source repository. The **Decrypt File** task can be used to decrypt them during a build or release.

Demands

xcode

YAML snippet

```
# Install Apple Provisioning Profile
# Install an Apple provisioning profile required to build on a macOS agent
- task: InstallAppleProvisioningProfile@1
  inputs:
    #provisioningProfileLocation: 'secureFiles' # Options: secureFiles, sourceRepository
    #provProfileSecureFile: # Required when provisioningProfileLocation == SecureFiles
    #provProfileSourceRepository: # Required when provisioningProfileLocation == SourceRepository
    #removeProfile: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Provisioning Profile Location (Azure Pipelines)	Select the location of the provisioning profile to install. The provisioning profile can be uploaded to Secure Files or stored in your source repository or a local path on the agent.
Provisioning Profile	Select the provisioning profile that was uploaded to Secure Files to install on the macOS agent (or) Select the provisioning profile from the source repository or specify the local path to a provisioning profile on the macOS agent.
Remove Profile After Build	Select to specify that the provisioning profile should be removed from the agent after the build or release is complete.

Install SSH Key task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to install an SSH key prior to a build or release.

YAML snippet

```
# Install SSH Key
# Install an SSH key prior to a build or release
- task: InstallSSHKey@0
  inputs:
    hostName:
    sshPublicKey:
    #sshPassphrase: # Optional
    sshKeySecureFile:
```

Arguments

ARGUMENT	DESCRIPTION
Known Hosts Entry	(Required) The entry for this SSH key for the known_hosts file.
SSH Public Key	(Required) The contents of the public SSH key.
SSH Passphrase	(Optional) The passphrase for the SSH key, if any.
SSH Key	(Required) Select the SSH key that was uploaded to Secure Files to install on the agent.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Invoke Azure Function task

11/15/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

Use this task in a build or release pipeline to invoke a HTTP triggered function in an Azure function app and parse the response.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Demands

Can be used in only an [agentless job](#) of a release pipeline.

YAML snippet

```
# Invoke Azure Function
# Invoke an Azure Function as a part of your pipeline.
- task: AzureFunction@1
  inputs:
    function:
      key:
        #method: 'POST' # Options: oPTIONS, gET, hEAD, pOST, pUT, dELETE, tRACE, pATCH
        #headers: '{Content-Type:application/json, PlanUrl: $(system.CollectionUri), ProjectId: $(system.TeamProjectId), HubName: $(system.HostType), PlanId: $(system.PlanId), JobId: $(system.JobId), TimelineId: $(system.TimelineId), TaskInstanceId: $(system.TaskInstanceId), AuthToken: $(system.AccessToken)}'
        #queryParameters: # Optional
        #body: # Required when method != GET && Method != HEAD
        #waitForCompletion: 'false' # Options: true, false
        #successCriteria: # Optional
```

Arguments

PARAMETER	COMMENTS
Azure function URL	Required. The URL of the Azure function to be invoked.
Function key	Required. The value of the available function or the host key for the function to be invoked. Should be secured by using a hidden variable.
Method	Required. The HTTP method with which the function will be invoked.
Headers	Optional. The header in JSON format to be attached to the request sent to the function.

PARAMETER	COMMENTS
Query parameters	Optional. Query parameters to append to the function URL. Must not start with "?" or "&".
Body	Optional. The request body for the Azure function call in JSON format.
Completion Event	Required. How the task reports completion. Can be API response (the default) - completion is when function returns success and success criteria evaluates to true, or Callback - the Azure function makes a callback to update the timeline record.
Success criteria	Optional. How to parse the response body for success.
Control options	See Control options

Succeeds if the function returns success and the response body parsing is successful, or when the function updates the timeline record with success.

For more information about using this task, see [Approvals and gates overview](#).

Also see this task on [GitHub](#).

Invoke REST API task

11/19/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018

Use this task in a build or release pipeline to invoke an HTTP API and parse the response.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

This task is available in both builds and releases in TFS 2018.2 In TFS 2018 RTM, this task is available only in releases.

Demands

This task can be used in only an [agentless job](#).

YAML snippet

```
# Invoke REST API
# Invoke a REST API as a part of your pipeline.
- task: InvokeRESTAPI@1
  inputs:
    #connectionType: 'connectedServiceName' # Options: connectedServiceName, connectedServiceNameARM
    #serviceConnection: # Required when connectionType == ConnectedServiceName
    #azureServiceConnection: # Required when connectionType == ConnectedServiceNameARM
    #method: 'POST' # Options: OPTIONS, gET, hHEAD, pOST, pUT, dELETE, tRACE, pATCH
    #headers: '{Content-Type:application/json, PlanUrl: $(system.CollectionUri), ProjectId: $(system.TeamProjectId), HubName: $(system.HostType), PlanId: $(system.PlanId), JobId: $(system.JobId), TimelineId: $(system.TimelineId), TaskInstanceId: $(system.TaskInstanceId), AuthToken: $(system.AccessToken)}'
    #body: # Required when method != GET && Method != HEAD
    #urlSuffix: # Optional
    #waitForCompletion: 'false' # Options: true, false
    #successCriteria: # Optional
```

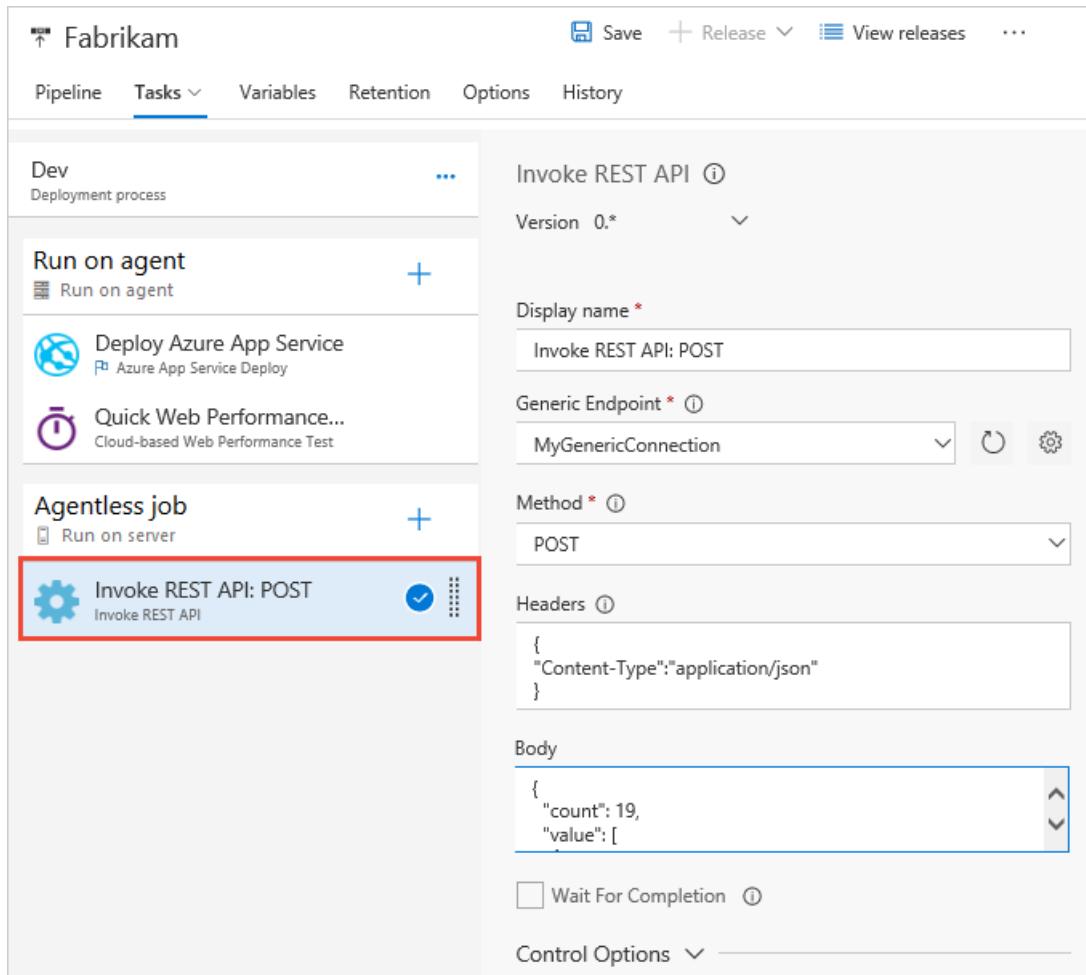
Arguments

PARAMETER	COMMENTS
Generic endpoint	Required. Select a Generic service connection. Provides the baseUrl for the call and the authorization to use.
Method	Required. The HTTP method with which the API will be invoked; for example, GET , PUT , or UPDATE
Headers	Optional. The header in JSON format to be attached to the request sent to the API.

PARAMETER	COMMENTS
Body	Optional. The request body for the function call in JSON format.
URL suffix and parameters	The string to append to the baseUrl from the Generic service connection while making the HTTP call
Completion event	Required. How the task reports completion. Can be API response (the default) - completion is when function returns success and success criteria evaluates to true, or Callback - the Azure function makes a callback to update the timeline record.
Success criteria	Optional. How to parse the response body for success. By default the task passes when 200 OK is returned from the call. Additionally, the success criteria - if specified - is evaluated.
Control options	See Control options

Succeeds if the API returns success and the response body parsing is successful, or when the API updates the timeline record with success.

The **Invoke REST API** task does not perform deployment actions directly. Instead, it allows you to invoke any generic HTTP REST API as part of the automated pipeline and, optionally, wait for it to be completed.



The screenshot shows the Azure DevOps Pipeline interface for a 'Dev Deployment process'. The 'Tasks' tab is selected. On the left, there are two sections: 'Run on agent' (containing 'Deploy Azure App Service' and 'Quick Web Performance...') and 'Agentless job' (containing 'Invoke REST API: POST'). The 'Invoke REST API: POST' task is highlighted with a red box. The task configuration pane on the right shows the following details:

- Display name:** Invoke REST API: POST
- Generic Endpoint:** MyGenericConnection
- Method:** POST
- Headers:**

```
{
  "Content-Type": "application/json"
}
```
- Body:**

```
{
  "count": 19,
  "value": [
    ...
  ]
}
```
- Wait For Completion:**
- Control Options:** [...](#)

For more information about using this task, see [Approvals and gates overview](#).

Open source

Also see this task on [GitHub](#).

Jenkins Download Artifacts task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

Use this task in a build or release pipeline to download artifacts produced by a Jenkins job.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions.
Service connections are called *service endpoints* in TFS 2018 and in older versions.

YAML snippet

```
# Jenkins Download Artifacts
# Download artifacts produced by a Jenkins job
- task: JenkinsDownloadArtifacts@1
  inputs:
    jenkinsServerConnection:
    jobName:
    #jenkinsJobType: # Optional
    #saveTo: 'jenkinsArtifacts'
    #jenkinsBuild: 'LastSuccessfulBuild' # Options: lastSuccessfulBuild, buildNumber
    #jenkinsBuildNumber: '1' # Required when jenkinsBuild == BuildNumber
    #itemPattern: '**' # Optional
    #downloadCommitsAndWorkItems: # Optional
    #startJenkinsBuildNumber: # Optional
    #artifactDetailsFileNameSuffix: # Optional
    #propagatedArtifacts: false # Optional
    #artifactProvider: 'azureStorage' # Required when propagatedArtifacts == NotValid# Options: azureStorage
    #connectedServiceNameARM: # Required when propagatedArtifacts == True
    #storageAccountName: # Required when propagatedArtifacts == True
    #containerName: # Required when propagatedArtifacts == True
    #commonVirtualPath: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Jenkins service connection	(Required) Select the service connection for your Jenkins instance. To create one, click the Manage link and create a new Jenkins service connection.
Job name	(Required) The name of the Jenkins job to download artifacts from. This must exactly match the job name on the Jenkins server.
Jenkins job type	(Optional) Jenkins job type, detected automatically.
Save to	(Required) Jenkins artifacts will be downloaded and saved to this directory. This directory will be created if it does not exist.

ARGUMENT	DESCRIPTION
Download artifacts produced by	(Required) Download artifacts produced by the last successful build, or from a specific build instance.
Jenkins build number	(Required) Download artifacts produced by this build.
Item Pattern	(Optional) Specify files to be downloaded as multi line minimatch pattern. More Information The default pattern (** will download all files across all artifacts produced by the Jenkins job. To download all files within artifact drop use drop/**.
Download Commits and WorkItems	(Optional) Enables downloading the commits and workitem details associated with the Jenkins Job
Download commits and workitems from	(Optional) Optional start build number for downloading commits and work items. If provided, all commits and work items between start build number and build number given as input to download artifacts will be downloaded.
Commit and WorkItem FileName	(Optional) Optional file name suffix for commits and workitem attachment. Attachment will be created with commits_{suffix}.json and workitem_{suffix}.json. If this input is not provided attachments will be create with the name commits.json and workitems.json
Artifacts are propagated to Azure	(Optional) Check this if Jenkins artifacts were propagated to Azure. To upload Jenkins artifacts to azure, refer to this Jenkins plugin
Artifact Provider	(Required) Choose the external storage provider used in Jenkins job to upload the artifacts.
Azure Subscription	(Required) Choose the Azure Resource Manager subscription for the artifacts.
Storage Account Name	(Required) Azure Classic and Resource Manager storage accounts are listed. Select the Storage account name in which the artifacts are propagated.
Container Name	(Required) Name of the container in the storage account to which artifacts are uploaded.
Common Virtual Path	(Optional) Path to the artifacts inside the Azure storage container.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Manual Intervention task

11/15/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a release pipeline to pause an active deployment within a stage, typically to perform some manual steps or actions, and then continue the automated deployment tasks.

Demands

Can be used in only an [agentless job](#) of a release pipeline.

YAML snippet

```
# Manual Intervention
# Pause deployment and wait for intervention
- task: ManualIntervention@8
  inputs:
    #instructions: # Optional
    #emailRecipients: # Optional
    #onTimeout: 'reject' # Optional. Options: reject, resume
```

Arguments

PARAMETER	COMMENTS
Display name	Required. The name to display for this task.
Instructions	Optional. The instruction text to display to the user when the task is activated.
Notify users	Optional. The list of users that will be notified that the task has been activated.
On timeout	Required. The action to take (reject or resume) if the task times out with no manual intervention. The default is to reject the deployment.
Control options	See Control options

The **Manual Intervention** task does not perform deployment actions directly. Instead, it allows you to pause an active deployment within a stage, typically to perform some manual steps or actions, and then continue the automated deployment tasks. For example, the user may need to edit the details of the current release before continuing; perhaps by entering the values for custom variables used by the tasks in the release.

The **Manual Intervention** task configuration includes an **Instructions** parameter that can be used to provide related information, or to specify the manual steps the user should execute during the agentless job. You can configure the task to send email notifications to users and user groups when it is awaiting intervention, and specify the automatic response (reject or resume the deployment) after a configurable timeout occurs.

All pipelines >  Fabrikam

Save + Release ...

Pipeline Tasks Variables Retention Options History

Dev Deployment process	...
Run on agent Run on agent	+
 Deploy Azure App Service Azure App Service Deploy	
Agentless job Run on server	+
 Manual Intervention Manual Intervention	...

Manual Intervention ⓘ

Version 8.*

Display name *

Manual Intervention

Instructions ⓘ

Ready to deploy to \$(Release.EnvironmentName) for customer \$(customerName). Please contact customer to confirm deployment requirements have been met.

You can use built-in and custom variables to generate portions of your instructions.

When the Manual Intervention task is activated during a deployment, it sets the deployment state to **IN PROGRESS** and displays a message bar containing a link that opens the Manual Intervention dialog containing the instructions. After carrying out the manual steps, the administrator or user can choose to resume the deployment, or reject it. Users with **Manage deployment** permission on the stage can resume or reject the manual intervention.

For more information about using this task, see [Approvals and gates overview](#).

PowerShell task

11/19/2018 • 4 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015

Use this task in a build or release pipeline to run a PowerShell script.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Demands

- DotNetFramework

YAML snippet

```
# PowerShell
# Run a PowerShell script on Windows, macOS, or Linux.
- task: PowerShell@2
  inputs:
    targetType: 'filePath' # Optional. Options: filePath, inline
    filePath: # Required when targetType == FilePath
    arguments: # Optional
    script: '# Write your powershell commands here.' # Required when targetType == Inline
    errorActionPreference: 'stop' # Optional. Options: stop, continue, silentlyContinue
    failOnStderr: false # Optional
    ignoreLASTEXITCODE: false # Optional
    pwsh: false # Optional
    workingDirectory: # Optional
```

The Powershell task also has a shortcut syntax in YAML:

```
- powershell: # inline script
  workingDirectory: #
  displayName: #
  failOnStderr: #
  errorActionPreference: #
  ignoreLASTEXITCODE: #
  env: # mapping of environment variables to add
```

Arguments

ARGUMENT	DESCRIPTION
Type	Sets whether this is an inline script or a path to a <code>.ps1</code> file.
File path	Path of the script to execute. Must be a fully qualified path or relative to <code>\$(System.DefaultWorkingDirectory)</code> . Required if Type is <code>filePath</code> .

ARGUMENT	DESCRIPTION
Arguments	Arguments passed to the Powershell script.
Script	Contents of the script. Required if Type is <code>inline</code> .
Working directory	Specify the working directory in which you want to run the command. If you leave it empty, the working directory is <code>\$(Build.SourcesDirectory)</code> .
Fail on standard error	If this is <code>true</code> , this task will fail if any errors are written to <code>stderr</code> .
errorActionPreference	Set PowerShell's error action preference. One of: <code>stop</code> , <code>continue</code> , <code>silentlyContinue</code> . Defaults to <code>stop</code> .
Ignore \$LASTEXITCODE	By default, the last exit code returned from your script will be checked and, if non-zero, treated as a step failure. If you don't want this behavior, set this to <code>true</code> .
Env[ironment variables]	A list of additional items to map into the process's environment. For example, secret variables are not automatically mapped. If you have a secret variable called <code>Foo</code> , you can map it in like this:
	<pre>yaml - script: echo \$env:MYSECRET env: MySecret: \$(Foo)</pre>

CONTROL OPTIONS

Examples

Hello World

Create `test.ps1` at the root of your repo:

```
Write-Host "Hello World from $Env:AGENT_NAME."
Write-Host "My ID is $Env:AGENT_ID."
Write-Host "AGENT_WORKFOLDER contents:"
gci $Env:AGENT_WORKFOLDER
Write-Host "AGENT_BUILDDIRECTORY contents:"
gci $Env:AGENT_BUILDDIRECTORY
Write-Host "BUILD_SOURCESDIRECTORY contents:"
gci $Env:BUILD_SOURCESDIRECTORY
Write-Host "Over and out."
```

On the Build tab of a build pipeline, add this task:

TASK	ARGUMENTS
 Utility: PowerShell	Run test.ps1. Script filename: <code>test.ps1</code>

Write a warning


Set warning message

- Arguments

```
"You've been warned by"
```

- Script

```
Write-Host "$('##vso[task.setvariable variable=WarningMessage]"') $($args[0])"
```

 Write warning using task.LogIssue

- Script

```
# Writes a warning to build summary and to log in yellow text
Write-Host "$('##vso[task.logissue type=warning;]") $($env:WarningMessage) $($("the task.LogIssue Azure Pipelines logging command."))"
```

 Write warning using PowerShell command

- Script

```
# Writes a warning to log preceded by "WARNING: "
Write-Warning $($env:WarningMessage) $($("the Write-Warning PowerShell command."))"
```

Write an error

 Set error message

- Arguments

```
"something went wrong."
```

- Script

```
Write-Host "$('##vso[task.setvariable variable=ErrorMessage]"') $($args[0])"
```

 Write error using task.LogIssue

- Script

```
# Writes an error to the build summary and to the log in red text
Write-Host "$('##vso[task.logissue type=error;]") $($("the task.LogIssue Azure Pipelines logging command reported that")) $($env:ErrorMessage)"
```

TIP

If you want this error to fail the build, then add this line:

```
exit 1
```



Write error using PowerShell command

- Script

```
# Writes an error to the build summary and the log with details about the error
Write-Error "$(the Write-Error PowerShell command reported that) $($env:ErrorMessage)"
```

TIP

If you don't want this error to fail the build, then clear the **Advanced: Fail on Standard Error** check box.

ApplyVersionToAssemblies.ps1

[Use a script to customize your build pipeline](#)

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Where can I learn about PowerShell scripts?

[Scripting with Windows PowerShell](#)

[Microsoft Script Center \(the Scripting Guys\)](#)

[Windows PowerShell Tutorial](#)

[PowerShell.org](#)

How do I set a variable so that it can be read by subsequent scripts and tasks?

[Define and modify your build variables in a script](#)

[Define and modify your release variables in a script](#)

Q: I'm having problems. How can I troubleshoot them?

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.

2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

Q: How do variables work? What variables are available for me to use in the arguments?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See [Variables](#).

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Publish Build Artifacts task

11/19/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015.3

Use this task in a build or release pipeline to publish build artifacts to Azure Pipelines, TFS, or a file share.

TIP

Looking to get started working with build artifacts? See [Artifacts in Azure Pipelines](#).

Demands

None

YAML snippet

```
# Publish Build Artifacts
# Publish build artifacts to Azure Pipelines/TFS or a file share
- task: PublishBuildArtifacts@1
  inputs:
    #pathToPublish: '$(Build.ArtifactStagingDirectory)'
    #artifactName: 'drop'
    #publishLocation: 'Container' # Options: container, filePath
    #targetPath: # Required when publishLocation == FilePath
    #parallel: false # Optional
    #parallelCount: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Path to publish	Path to the folder or file you want to publish. The path must be a fully-qualified path or a valid path relative to the root directory of your repository. Typically, you'll specify <code>\$(Build.ArtifactStagingDirectory)</code> . Subdirectories of the specified path will not be published. Wildcards in the path are not supported. See Artifacts in Azure Pipelines .
Artifact name	Specify the name of the artifact that you want to create. It can be whatever you want. For example: <code>drop</code>
Artifact publish location	In most cases, Azure Pipelines/TFS (<code>Server</code> on TFS 2018 RTM and older) is the best and simplest option. Otherwise, choose a file share and then specify a few more arguments (see below). To learn more, see Artifacts in Azure Pipelines .

ARGUMENT	DESCRIPTION
File share path	Specify the path to the file share where you want to copy the files. The path must be a fully-qualified path or a valid path relative to the root directory of your repository. Publishing artifacts from a Linux or macOS agent to a file share is not supported.
Parallel copy (Azure Pipelines , TFS 2018 , or newer)	Select whether to copy files in parallel using multiple threads for greater potential throughput. If this setting is not enabled, a single thread will be used.
Parallel count (Azure Pipelines , TFS 2018 , or newer)	Enter the degree of parallelism (the number of threads) used to perform the copy. The value must be at least 1 and not greater than 128. Choose a value based on CPU capabilities of the build agent. Typically, 8 is a good starting value.
Control options	

Usage

A typical pattern for using this task is:

- Build something
- Copy build outputs to a staging directory
- Publish staged artifacts

For example:

```
steps:
- powershell: .\build.ps1
- task: CopyFiles@2
  inputs:
    contents: _buildOutput\**
    targetFolder: $(Build.ArtifactStagingDirectory)
- task: PublishBuildArtifacts@1
  inputs:
    pathToPublish: $(Build.ArtifactStagingDirectory)
    artifactName: MyBuildOutputs
```

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Q: I'm having problems. How can I troubleshoot them?

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

Q: How do variables work? What variables are available for me to use in the arguments?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See

Variables.

Publish Pipeline Artifact task

11/19/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to publish pipeline artifacts to Azure Pipelines.

TIP

Looking to get started working with build artifacts? See [Artifacts in Azure Pipelines](#).

Demand

None

YAML snippet

```
# Publish Pipeline Artifact
- task: PublishPipelineArtifact@0
  inputs:
    #artifactName: 'drop'
    targetPath:
```

Arguments

ARGUMENT	DESCRIPTION
targetPath	Path to the folder or file you want to publish. The path must be a fully-qualified path or a valid path relative to the root directory of your repository. Typically, you'll specify <code>\$(Build.ArtifactStagingDirectory)</code> . See Artifacts in Azure Pipelines .
artifactName	Specify the name of the artifact that you want to create. It can be whatever you want. For example: <code>drop</code>
Control options	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Q: I'm having problems. How can I troubleshoot them?

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

Q: How do variables work? What variables are available for me to use in the arguments?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See [Variables](#).

Publish To Azure Service Bus task

11/15/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to send a message to an Azure Service Bus using a service connection and without using an agent.

Demands

Can be used in only an [agentless job](#) of a release pipeline.

YAML snippet

```
# Publish To Azure Service Bus
# Sends a message to azure service bus using a service connection (no agent required).
- task: PublishToAzureServiceBus@1
  inputs:
    azureSubscription:
    #messageBody: # Optional
    #signPayload: false
    #certificateString: # Required when signPayload == True
    #signatureKey: 'signature' # Optional
    #waitForCompletion: false
```

Arguments

PARAMETER	COMMENTS
Display name	Required. The name to display for this task.
Azure Service Bus Connection	Required. An existing service connection to an Azure Service Bus.
Message body	Required. The text of the message body to send to the Service Bus.
Wait for Task Completion	Optional. Set this option to force the task to halt until a response is received.
Control options	See Control options

Also see this task on [GitHub](#).

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

Python Script task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to run a Python script.

YAML snippet

```
# Python Script
# Run a Python script.
- task: PythonScript@0
  inputs:
    #scriptSource: 'filePath' # Options: filePath, inline
    #scriptPath: # Required when scriptSource == FilePath
    #script: # Required when scriptSource == Inline
    #arguments: # Optional
    #pythonInterpreter: # Optional
    #workingDirectory: # Optional
    #failOnStderr: false # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Type	(Required) Target script type: File path or Inline
Script Path	(Required when targetType == filePath) Path of the script to execute. Must be a fully qualified path or relative to \$(System.DefaultWorkingDirectory).
Script	(Required when targetType == inline) The Python script to run
Arguments	(Optional) Arguments passed to the script execution, available through <code>sys.argv</code> .
Python interpreter	(Optional) Absolute path to the Python interpreter to use. If not specified, the task will use the interpreter in PATH.
Working directory	(Optional) undefined
Fail on standard error	(Optional) If this is true, this task will fail if any text are written to the stderr stream.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Query Azure Monitor Alerts task

11/19/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

Use this task in a release pipeline to observe the configured Azure monitor rules for active alerts.

Can be used in only an [agentless job](#) of a release pipeline.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Demands

None

YAML snippet

```
# Query Azure Monitor Alerts
# Observe the configured Azure monitor rules for active alerts.
- task: AzureMonitor@0
  inputs:
    connectedServiceNameARM:
    resourceName:
      #resourceType: 'Microsoft.Insights/components' # Options: microsoft.Insights/Components,
      microsoft.Web/Sites, microsoft.Storage/StorageAccounts, microsoft.Compute/VirtualMachines
    resourceName:
    alertRules:
```

Arguments

PARAMETER	COMMENTS
Azure subscription	Required. Select an Azure Resource Manager service connection.
Resource group	Required. The resource group being monitored in the subscription.
Resource type	Required. Select the resource type in the selected group.
Resource name	Required. Select the resources of the chosen types in the selected group.
Alert rules	Required. Select from the currently configured alert rules to query for status.
Control options	See Control options

Succeeds if none of the alert rules are activated at the time of sampling.

For more information about using this task, see [Approvals and gates overview](#).

Also see this task on [GitHub](#).

Query Work Items task

11/15/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

Use this task in a build or release pipeline to ensure the number of matching items returned by a work item query is within the configured thresholds.

Can be used in only an [agentless job](#) of a release pipeline.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Demands

None

YAML snippet

```
# Query Work Items
# Executes a work item query and checks for the number of items returned.
- task: queryWorkItems@0
  inputs:
    queryId:
      #maxThreshold: '0'
      #minThreshold: '0'
```

Arguments

PARAMETER	COMMENTS
Query	Required. Select a work item query within the current project. Can be a built-in or custom query.
Upper threshold	Required. Maximum number of matching workitems for the query. Default value = 0
Lower threshold	Required. Minimum number of matching workitems for the query. Default value = 0
Control options	See Control options

Succeeds if *minimum-threshold* \leq *#-matching-workitems* \leq *maximum-threshold*

For more information about using this task, see [Approvals and gates overview](#).

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Service Fabric PowerShell task

11/15/2018 • 2 minutes to read [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to run a PowerShell script within the context of an Azure Service Fabric cluster connection. Runs any PowerShell command or script in a PowerShell session that has a Service Fabric cluster connection initialized.

Prerequisites

Service Fabric

- This task uses a Service Fabric installation to connect and deploy to a Service Fabric cluster.
- [Azure Service Fabric Core SDK](#) on the build agent.

YAML snippet

```
# Service Fabric PowerShell
# Run a PowerShell script within the context of an Azure Service Fabric cluster connection.
- task: ServiceFabricPowerShell@1
  inputs:
    clusterConnection:
      #scriptType: 'FilePath' # Options: filePath, inlineScript
      #scriptPath: # Optional
      #inline: '# You can write your PowerShell scripts inline here. # You can also pass predefined and custom variables to this script using arguments' # Optional
      #scriptArguments: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Cluster Connection	The Azure Service Fabric service connection to use to connect and authenticate to the cluster.
Script Type	Specify whether the script is provided as a file or inline in the task.
Script Path	Path to the PowerShell script to run. Can include wildcards and variables. Example: <code>\$(system.defaultworkingdirectory)/**/drop/projectartifacts/**/compose.yml</code> . Note: combining compose files is not supported as part of this task.
Script Arguments	Additional parameters to pass to the PowerShell script. Can be either ordinal or named parameters.
Inline Script	The PowerShell commands to run on the build agent. More information
Control options	See Control options

Also see: [Service Fabric Compose Deploy task](#)

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

Shell Script task

11/26/2018 • 2 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

Use this task in a build or release pipeline to run a shell script using bash.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Demands

sh

YAML snippet

```
- task: ShellScript@2
  inputs:
    scriptPath:
      #args: '' # Optional
      #disableAutoCwd: false # Optional
      #cwd: '' # Optional
      #failOnStandardError: false
```

Arguments

ARGUMENT	DESCRIPTION
Script Path	Relative path from the repo root to the shell script file that you want to run.
Arguments	Arguments that you want to pass to the script.

ADVANCED

Working Directory	Working directory in which you want to run the script. If you leave it empty it is folder where the script is located.
Fail on Standard Error	Select if you want this task to fail if any errors are written to the StandardError stream.

CONTROL OPTIONS

Example

Create `test.sh` at the root of your repo. We recommend creating this file from a Linux environment (such as a real Linux machine or Windows Subsystem for Linux) so that line endings are correct. Also, don't forget to `chmod +x test.sh` before you commit it.

```
#!/bin/bash
echo "Hello World"
echo "AGENT_WORKFOLDER is $AGENT_WORKFOLDER"
echo "AGENT_WORKFOLDER contents:"
ls -1 $AGENT_WORKFOLDER
echo "AGENT_BUILDDIRECTORY is $AGENT_BUILDDIRECTORY"
echo "AGENT_BUILDDIRECTORY contents:"
ls -1 $AGENT_BUILDDIRECTORY
echo "SYSTEM_HOSTTYPE is $SYSTEM_HOSTTYPE"
echo "Over and out."
```

On the [Build tab](#) of a build pipeline, add this task:

 Utility: Shell Script	Run test.bat. • Script Path: <code>test.sh</code>
---	--

This example also works with release pipelines.

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Where can I learn about Bash scripts?

[Beginners/BashScripting](#) to get started.

[Awesome Bash](#) to go deeper.

How do I set a variable so that it can be read by subsequent scripts and tasks?

[Define and modify your build variables in a script](#)

[Define and modify your release variables in a script](#)

Q: I'm having problems. How can I troubleshoot them?

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

Q: How do variables work? What variables are available for me to use in the arguments?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See [Variables](#).

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Update Service Fabric App Versions task

11/6/2018 • 2 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#)

Use this task in a build or release pipeline to automatically update the versions of a packaged Service Fabric app. This task appends a version suffix to all service and app versions, specified in the manifest files, in an Azure Service Fabric app package.

Demands

None

YAML snippet

```
# Update Service Fabric Manifests
# Automatically updates portions of the application and service manifests within a packaged Service Fabric
application.
- task: ServiceFabricUpdateManifests@2
  inputs:
    #updateType: 'Manifest versions' # Options: manifest Versions, docker Image Settings
    applicationPackagePath:
    #versionSuffix: '$(Build.BuildNumber)' # Required when updateType == Manifest Versions
    #versionBehavior: 'Append' # Optional. Options: append, replace
    #updateOnlyChanged: false # Required when updateType == Manifest Versions
    #pkgArtifactName: # Optional
    #logAllChanges: true # Optional
    #compareType: 'LastSuccessful' # Optional. Options: lastSuccessful, specific
    #buildNumber: # Optional
    #overwriteExistingPkgArtifact: true # Optional
    #imageNamesPath: # Optional
    #imageDigestsPath: # Required when updateType == Docker Image Settings
```

Arguments

ARGUMENT	DESCRIPTION
Application Package	<p>The location of the Service Fabric application package to be deployed to the cluster.</p> <ul style="list-style-type: none">• Example: \$(system.defaultworkingdirectory)/**/drop/applicationpackage• Can include wildcards and variables.

ARGUMENT	DESCRIPTION
Version Value	<p>The value appended to the versions in the manifest files. Default is <code>\$(Build.BuildNumber)</code>.</p> <p>Tip: You can modify the build number format directly or use a logging command to dynamically set a variable in any format. For example, you can use <code>\$(VersionSuffix)</code> defined in a PowerShell task:</p> <pre>\$versionSuffix = ".\${[DateTimeOffset]::UtcNow.ToString('yyyyMMdd.HHmmss')}"</pre> <pre>Write-Host "##vso[task.setvariable variable=VersionSuffix;]\$versionSuffix"</pre>
Version Behavior	Specify whether to append the version value to existing values in the manifest files, or replace them.
Update only if changed	<p>Select this check box if you want to append the new version suffix to only the packages that have changed from a previous build. If no changes are found, the version suffix from the previous build will be appended.</p> <p>Note: By default, the compiler will create different outputs even if you made no changes. Use the deterministic compiler flag to ensure builds with the same inputs produce the same outputs.</p>
Package Artifact Name	The name of the artifact containing the application package from the previous build.
Log all changes	Select this check box to compare all files in every package and log if the file was added, removed, or if its content changed. Otherwise, compare files in a package only until the first change is found for potentially faster performance.
Compare against	Specify whether to compare against the last completed, successful build or against a specific build.
Build Number	If comparing against a specific build, the build number to use.
CONTROL OPTIONS	

Also see: [Service Fabric Application Deployment task](#)

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

App Center Test task

11/6/2018 • 4 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

Use this task in a build or release pipeline to test app packages with Visual Studio App Center.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

YAML snippet

```
# App Center Test
# Test app packages with Visual Studio App Center.
- task: AppCenterTest@1
  inputs:
    appFile:
    #artifactsDirectory: '$(Build.ArtifactStagingDirectory)/AppCenterTest'
    #prepareTests: # Optional
    #frameworkOption: 'appium' # Required when prepareTests == True# Options: appium, espresso, calabash, uitest, xcuitest
    #appiumBuildDirectory: # Required when prepareTests == True && Framework == Appium
    #espressoBuildDirectory: # Optional
    #espressoTestApkFile: # Optional
    #calabashProjectDirectory: # Required when prepareTests == True && Framework == Calabash
    #calabashConfigFile: # Optional
    #calabashProfile: # Optional
    #calabashSkipConfigCheck: # Optional
    #uiTestBuildDirectory: # Required when prepareTests == True && Framework == Uitest
    #uitestStoreFile: # Optional
    #uiTestStorePassword: # Optional
    #uitestKeyAlias: # Optional
    #uiTestKeyPassword: # Optional
    #uiTestToolsDirectory: # Optional
    #signInfo: # Optional
    #xcUITestBuildDirectory: # Optional
    #xcUITestIpaFile: # Optional
    #prepareOptions: # Optional
    #runTests: # Optional
    #credentialsOption: 'serviceEndpoint' # Required when runTests == True# Options: serviceEndpoint, inputs
    #serverEndpoint: # Required when runTests == True && CredsType == ServiceEndpoint
    #username: # Required when runTests == True && CredsType == Inputs
    #password: # Required when runTests == True && CredsType == Inputs
    #appSlug: # Required when runTests == True
    #devices: # Required when runTests == True
    #series: 'master' # Optional
    #dsymDirectory: # Optional
    #localeOption: 'en_US' # Required when runTests == True# Options: da_DK, nl_NL, en_GB, en_US, fr_FR, de_DE, ja_JP, ru_RU, es_MX, es_ES, user
    #userDefinedLocale: # Optional
    #loginOptions: # Optional
    #runOptions: # Optional
    #skipWaitingForResults: # Optional
    #cliFile: # Optional
    #showDebugOutput: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Binary application file path	(Required) Relative path from the repo root to the APK or IPA file you want to test.
Artifacts directory	(Required) Where to place the artifacts produced by the prepare step and used by the run step. This directory will be created if it does not exist.
Prepare tests	(Optional) undefined
Test framework	(Required) undefined
Build directory	(Required) Path to directory with Appium tests.
Build directory	(Optional) Path to Espresso output directory.
Test APK path	(Optional) Path to APK file with Espresso tests. If not set, build-dir is used to discover it. Wildcard is allowed.
Project directory	(Required) Path to Calabash workspace directory.
Cucumber config file	(Optional) Path to Cucumber configuration file, usually cucumber.yml.
Profile to run	(Optional) Profile to run. This value must exists in the Cucumber configuration file.
Skip Configuration Check	(Optional) Force running without Cucumber profile.
Build directory	(Required) Path to directory with built test assemblies.
Store file	(Optional) undefined
Store password	(Optional) undefined
Key alias	(Optional) undefined
Key password	(Optional) undefined
Test tools directory	(Optional) Path to directory with Xamarin UI test tools that contains test-cloud.exe.
Signing information	(Optional) Use Signing Infor for signing the test server.
Build directory	(Optional) Path to the build output directory (usually \$(ProjectDir)/Build/Products/Debug-iphoneos).
Test IPA path	(Optional) Path to the *.ipa file with the XCUITest tests.
Additional options	(Optional) Additional arguments passed to the App Center test prepare step.

ARGUMENT	DESCRIPTION
Run tests	(Optional) undefined
Authentication method	(Required) Use App Center service connection or enter credentials to connect to Visual Studio App Center.
App Center connection	(Required) Select the service connection for your Visual Studio App Center connection. To create one, click the Manage link and create a new service connection.
App Center username	(Required) Visit https://appcenter.ms/settings/profile to get your username.
App Center password	(Required) Visit https://appcenter.ms/settings/profile to set your password. It can accept variable defined in Build/Release pipelines as '\$(passwordVariable)'. You may mark variable type as 'secret' to secure it.
App slug	(Required) The app slug is in the format of {username}/{app_identifier}. To locate {username} and {app_identifier} for an app, click on its name from https://appcenter.ms/apps , and the resulting url is in the format of https://appcenter.ms/users/{username}/apps/{app_identifier} .
Devices	(Required) String to identify what devices this test will run against. Copy and paste this string when you define a new test run from App Center Test beacon.
Test series	(Optional) The series name for organizing test runs (e.g. master, production, beta).
dSYM directory	(Optional) Path to iOS symbol files.
System language	(Required) If your language isn't displayed, select 'Other' and enter its locale below, such as en_US.
Other locale	(Optional) Enter any two-letter ISO-639 language code along with any two-letter ISO 3166 country code in the format [language]_[country], such as en_US.
Additional options for login	(Optional) Additional arguments passed to the App Center login step.
Additional options for run	(Optional) Additional arguments passed to the App Center test run.
Do not wait for test result	(Optional) Execute command asynchronously, exit when tests are uploaded, without waiting for test results.
App Center CLI location	(Optional) Path to the App Center CLI on the build or release agent.
Enable debug output	(Optional) Add --debug to the App Center CLI.

CONTROL OPTIONS

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Cloud-based Apache JMeter Load Test task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to run Apache JMeter load tests in the cloud.

Demands

The agent must have the following capability:

- Azure PowerShell

YAML snippet

```
# Cloud-based Apache JMeter Load Test
# Runs the Apache JMeter load test in cloud
- task: ApacheJMeterLoadTest@1
  inputs:
    #connectedServiceName: # Optional
    testDrop:
    #loadTest: 'jmeter.jmx'
    #agentCount: '1' # Options: 1, 2, 3, 4, 5
    #runDuration: '60' # Options: 60, 120, 180, 240, 300
    #geoLocation: 'Default' # Optional. Options: default, australia East, australia Southeast, brazil South, central India, central US, east Asia, east US 2, east US, japan East, japan West, north Central US, north Europe, south Central US, south India, southeast Asia, west Europe, west US
    #machineType: '0' # Optional. Options: 0, 2
```

Arguments

ARGUMENT	DESCRIPTION
VS Team Services Connection	(Optional) Select a previously registered service connection to talk to the cloud-based load test service. Choose 'Manage' to register a new connection.
Apache JMeter test files folder	(Required) Relative path from repo root where the load test files are available.
Apache JMeter file	(Required) The Apache JMeter test filename to be used under the load test folder specified above.
Agent Count	(Required) Number of test agents (dual-core) used in the run.
Run Duration (sec)	(Required) Load test run duration in seconds.
Run load test using	(Optional) undefined
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Cloud-based Load Test task

11/15/2018 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015

Use this task in a build or release pipeline to run a load test in the cloud, to understand, test, and validate your app's performance. The task uses the Cloud-based Load Test Service based in Microsoft Azure and can be used to test your app's performance by generating load on it.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

Demands

The agent must have the following capability:

- Azure PowerShell

YAML snippet

```
# Cloud-based Load Test
# Runs the load test in the cloud with Azure Pipelines.
- task: CloudLoadTest@1
  inputs:
    #connectedServiceName: # Optional
    #testDrop: '$(System.DefaultWorkingDirectory)'
    loadTest:
      #activeRunSettings: 'useFile' # Optional. Options: useFile, changeActive
      #runSettingName: # Required when activeRunSettings == ChangeActive
      #testContextParameters: # Optional
      #testSettings: # Optional
      #thresholdLimit: # Optional
      #machineType: '0' # Options: 0, 2
      #resourceGroupName: 'default' # Optional
      #numOfSelfProvisionedAgents: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Azure Pipelines connection	<p>The name of a Generic service connection that references the Azure DevOps organization you will be running the load test from and publishing the results to.</p> <ul style="list-style-type: none">- Required for builds and releases on TFS and must specify a connection to the Azure DevOps organization where the load test will run.- Optional for builds and releases on Azure Pipelines. In this case, if not provided, the current Azure Pipelines connection is used.- See Generic service connection.

ARGUMENT	DESCRIPTION
Test settings file	Required. The path relative to the repository root of the test settings file that specifies the files and data required for the load test such as the test settings, any deployment items, and setup/clean-up scripts. The task will search this path and any subfolders.
Load test files folder	Required. The path of the load test project. The task looks here for the files required for the load test, such as the load test file, any deployment items, and setup/clean-up scripts. The task will search this path and any subfolders.
Load test file	Required. The name of the load test file (such as myfile.loadtest) to be executed as part of this task. This allows you to have more than one load test file and choose the one to execute based on the deployment environment or other factors.
Number of permissible threshold violations	Optional. The number of critical violations that must occur for the load test to be deemed unsuccessful, aborted, and marked as failed.
Control options	See Control options

Examples

- [Load test your app in the cloud](#)
- [Scheduling Load Test Execution](#)

More Information

- [Cloud-based Load Testing](#)
- [Source code for this task](#)
- [Build your Visual Studio solution](#)
- [Cloud-based Load Testing Knowledge Base](#)

Related tasks

- [Cloud-based Web Performance Test](#)

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

How do I use a Test Settings file?

The **Test settings file** references any setup and cleanup scripts required to execute the load test. For more details see: [Using Setup and Cleanup Script in Cloud Load Test](#)

When should I specify the number of permissible threshold violations?

Use the **Number of permissible threshold violations** setting if your load test is not already configured with information about how many violations will cause a failure to be reported. For more details, see: [How to: Analyze Threshold Violations Using the Counters Panel in Load Test Analyzer](#).

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Help and support

- Report problems through the [Developer Community](#).

Cloud-based Web Performance Test task

11/15/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015

Use this task in a build or release pipeline to run the Quick Web Performance Test to easily verify your web application exists and is responsive. The task generates load against an application URL using the Azure Pipelines Cloud-based Load Test Service based in Microsoft Azure.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

Demands

The agent must have the following capability:

- Azure PowerShell

YAML snippet

```
# Cloud-based Web Performance Test
# Runs a quick web performance test in the cloud with Azure Pipelines.
- task: QuickPerfTest@1
  inputs:
    #connectedServiceName: # Optional
    websiteUrl:
    testName:
    #vuLoad: '25' # Options: 25, 50, 100, 250
    #runDuration: '60' # Options: 60, 120, 180, 240, 300
    #geoLocation: 'Default' # Optional. Options: default, australia East, australia Southeast, brazil South, central India, central US, east Asia, east US 2, east US, japan East, japan West, north Central US, north Europe, south Central US, south India, southeast Asia, west Europe, west US
    #machineType: '0' # Options: 0, 2
    #resourceGroupName: 'default' # Optional
    #numOfSelfProvisionedAgents: # Optional
    #avgResponseTimeThreshold: '0' # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Azure Pipelines connection	<p>The name of a Generic service connection that references the Azure DevOps organization you will be running the load test from and publishing the results to.</p> <ul style="list-style-type: none">- Required for builds and releases on TFS and must specify a connection to the Azure DevOps organization where the load test will run.- Optional for builds and releases on Azure Pipelines. In this case, if not provided, the current Azure Pipelines connection is used.- See Generic service connection.

ARGUMENT	DESCRIPTION
Website Url	Required. The URL of the app to test.
Test Name	Required. A name for this load test, used to identify it for reporting and for comparison with other test runs.
User Load	Required. The number of concurrent users to simulate in this test. Select a value from the drop-down list.
Run Duration (sec)	Required. The duration of this test in seconds. Select a value from the drop-down list.
Load Location	The location from which the load will be generated. Select a global Azure location, or Default to generate the load from the location associated with your Azure DevOps organization.
Run load test using	Select Automatically provisioned agents if you want the cloud-based load testing service to automatically provision agents for running the load tests. The application URL must be accessible from the Internet. Select Self-provisioned agents if you want to test apps behind the firewall. You must provision agents and register them against your Azure DevOps organization when using this option. See Testing private/intranet applications using Cloud-based load testing .
Fail test if Avg. Response Time (ms) exceeds	Specify a threshold for the average response time in milliseconds. If the observed response time during the load test exceeds this threshold, the task will fail.
Control options	See Control options

More Information

- [Cloud-based Load Testing](#)
- [Performance testing video and Q&A](#)

Related tasks

- [Cloud-based Load Test](#)
- [Cloud-based Apache JMeter Load Test](#)

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Help and support

- Report problems through the [Developer Community](#).

Publish Code Coverage Results task

11/19/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015

Use this task in a build pipeline to publish code coverage results to Azure Pipelines or TFS, which were produced by a build in [Cobertura](#) or [JaCoCo](#) format. In addition, there are built-in tasks such as [Visual Studio Test](#), [.NET Core](#), [Ant](#), [Maven](#), [Gulp](#), [Grunt](#) and [Gradle](#) that provide the option to publish code coverage data to the pipeline.

The example below shows the **Ant** task with the option to publish code coverage data in Cobertura or JaCoCo format.

The screenshot shows the Azure Pipelines task editor with the 'Tasks' tab selected. A pipeline named 'Pipeline' is displayed, containing a 'Get sources' step and an 'Agent job 1' step. Under 'Agent job 1', an 'Ant build.xml' task is selected. The task configuration pane on the right shows the following settings:

- Version:** 1.*
- Display name:** Ant build.xml
- Ant build file:** build.xml
- Options:** (empty)
- Target(s):** (empty)
- JUnit Test Results:** Publish to TFS/Team Services (checked), Test results files: **/TEST-*.xml, Test run title: (empty)
- Code Coverage:** Code coverage tool: Cobertura (selected from a dropdown menu also containing None and JaCoCo).
- Output Variables:** (empty)

Demands

[none]

YAML snippet

```
# Publish Code Coverage Results
# Publish Cobertura or JaCoCo code coverage results from a build
- task: PublishCodeCoverageResults@1
  inputs:
    #codeCoverageTool: 'JaCoCo' # Options: cobertura, jaCoCo
    summaryFileLocation:
    #reportDirectory: # Optional
    #additionalCodeCoverageFiles: # Optional
    #failIfCoverageEmpty: false # Optional
```

The **codeCoverageTool** and **summaryFileLocation** parameters are mandatory.

To publish code coverage results for Javascript with istanbul using YAML, see [JavaScript](#) in the Languages section of these topics, which also includes examples for other languages.

Arguments

ARGUMENT	DESCRIPTION
Code coverage tool	(Required) The tool with which code coverage results are generated. The supported formats include Cobertura and JaCoCo.
Summary file	(Required) Path of the summary file containing code coverage statistics, such as line, method, and class coverage. The value may contain minimatch patterns. For example: <code>\$(System.DefaultWorkingDirectory)/MyApp/**/site/cobertura/coverage.xml</code>
Report directory	(Optional) Path of the code coverage HTML report directory. The report directory is published for later viewing as an artifact of the build. The value may contain minimatch patterns. For example: <code>\$(System.DefaultWorkingDirectory)/MyApp/**/site/cobertura</code>
Additional files	(Optional) File path pattern specifying any additional code coverage files to be published as artifacts of the build. The value may contain minimatch patterns. For example: <code>\$(System.DefaultWorkingDirectory)/**/*.exec</code>
Fail when code coverage results are missing	(Optional) Available only on Azure Pipelines and TFS 2018 and later. Fail the task if code coverage did not produce any results to publish.

CONTROL OPTIONS

Docker

For apps using docker, build and tests may run inside the container, generating code coverage results within the container. In order to publish the results to the pipeline, the resulting artifacts should be made available to the [Publish Code Coverage Results](#) task. For reference you can see a similar example for publishing test results under [Build, test, and publish results with a Docker file](#) section for [Docker](#).

View results

In order to view the code coverage results in the pipeline, see [Review code coverage results](#)

Related tasks

- [Publish Test Results](#)

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Help and support

- Report problems through the [Developer Community](#).

Publish Test Results task

11/6/2018 • 11 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015

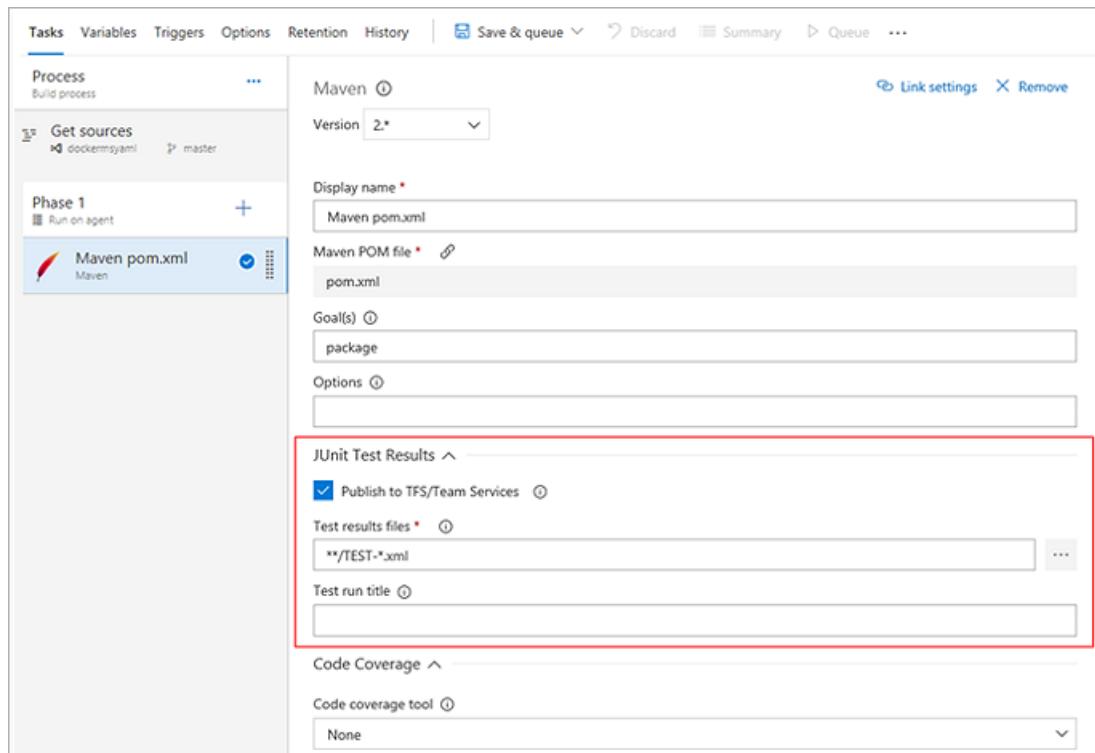
Use this task in a build or release pipeline to publish test results to Azure Pipelines or TFS when tests are executed using your choice of runner, and when results are available in any of the supported result formats.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

The test result formats supported by the task include [JUnit](#), [NUnit 2](#), [NUnit 3](#), Visual Studio Test (TRX), and [xUnit 2](#). Other built-in tasks such as [Visual Studio Test](#) automatically publish test results to the pipeline, while tasks such as [Ant](#), [Maven](#), [Gulp](#), [Grunt](#), and [Xcode](#) provide this as an option within the task.

The following example shows a Maven task configured to publish test results.



The published test results are displayed in the [Tests tab](#) in a build or release summary.

Demands

[none]

YAML snippet

```

# Publish Test Results
# Publish Test Results to Azure Pipelines/TFS
- task: PublishTestResults@2
  inputs:
    #testResultsFormat: 'JUnit' # Options: JUnit, NUnit, VSTest, xUnit
    #testResultsFiles: '**/TEST-*.xml'
    #searchFolder: '$(System.DefaultWorkingDirectory)' # Optional
    #mergeTestResults: false # Optional
    #testRunTitle: # Optional
    #buildPlatform: # Optional
    #buildConfiguration: # Optional
    #publishRunAttachments: true # Optional

```

The default option uses JUnit format to publish test results. When using VSTest as the **testRunner**, the **testResultsFiles** option should be changed to `**/TEST-*.trx`.

testResultsFormat is an alias for the **testRunner** input name. The results files can be produced by multiple runners, not just a specific runner. For example, jUnit results format is supported by many runners and not just jUnit.

To publish test results for Python using YAML, see [Python](#) in the **Languages** section of these topics, which also includes examples for other languages.

Arguments

NOTE

Options specified below are applicable to the latest version of the task.

ARGUMENT	DESCRIPTION
Test result formats	Specify the format of the results files you want to publish. The following formats are supported. - JUnit , NUnit 2 , NUnit 3 , Visual Studio Test (TRX) and xUnit 2
Test results files	Use this to specify one or more test results files. - You can use a single-folder wildcard (<code>*</code>) and recursive wildcards (<code>**</code>). For example, <code>**/TEST-*.xml</code> searches for all the XML files whose names start with <code>TEST-</code> in all subdirectories. If using VSTest as the test result format, the file type should be changed to <code>.trx</code> e.g. <code>**/TEST-*.trx</code> - Multiple paths can be specified, separated by a semicolon. - Additionally accepts minimatch patterns . For example, <code>!TEST[1-3].xml</code> excludes files named <code>TEST1.xml</code> , <code>TEST2.xml</code> , or <code>TEST3.xml</code> .
Search folder	Folder to search for the test result files. Default is <code>\$(System.DefaultWorkingDirectory)</code>
Merge test results	When this option is selected, test results from all the files will be reported against a single test run . If this option is not selected, a separate test run will be created for each test result file.

ARGUMENT	DESCRIPTION
Test run title	Use this option to provide a name for the test run against which the results will be reported. Variable names declared in the build or release pipeline can be used.
Advanced - Platform	Build platform against which the test run should be reported. For example, <code>x64</code> or <code>x86</code> . If you have defined a variable for the platform in your build task, use that here.
Advanced - Configuration	Build configuration against which the Test Run should be reported. For example, Debug or Release. If you have defined a variable for configuration in your build task, use that here.
Advanced - Upload test results files	When selected, the task will upload all the test result files as attachments to the test run.
Control options	See Control options

Result formats mapping

This table lists the fields reported in the [Tests tab](#) in a build or release summary, and the corresponding mapping with the attributes in the supported test result formats.

SCOPE	FIELD	VISUAL STUDIO TEST (TRX)	JUNIT	NUNIT 2	NUNIT 3	XUNIT 2
Test run	Title	Test run title specified in the task	Test run title specified in the task	Test run title specified in the task	Test run title specified in the task	Test run title specified in the task
	Date started	/TestRun/Times.Attributes[" start"].Value	/testsuites/testsuite.Attributes[" timeStamp"].Value	/test-results.Attributes[" date"].Value + /test-results.Attributes["time"].Value	/test-run/ start-time	/assemblies/assembly/ run-date + /assemblies/assembly/ run-time
	Date completed	/TestRun/Times.Attributes[" finish"].Value	/testsuites/testsuite.Attributes[" timeStamp"].Value + SUM(/testsuites/testsuite/testcase.Attributes["time"].Value) for all test cases in the test suite	Date started + /test-results/results/test-case.Attributes[" time"].Value for all test cases	/test-run/ end-time	Date started + /assemblies/assembly/ time
	Duration	Date completed - Date started	Date completed - Date started	Date completed - Date started	Date completed - Date started	Date completed - Date started

SCOPE	FIELD	VISUAL STUDIO TEST (TRX)	JUNIT	NUNIT 2	NUNIT 3	XUNIT 2
	Attachments	Refer to Attachments support section below	Results file, used to publish test results	Results file used to publish test results	Refer to Attachments support section below	Results file used to publish test results
Test result	Title	/TestRun/Results/UnitTestResult.Attributes[" testName"].Value Or /TestRun/Results/WebTestResult.Attributes["testName"].Value Or /TestRun/Results/TestResultAggregation.Attributes["testName"].Value	/testsuites/testsuite/testcase/Attributes[" name"].Value	/test-results/results/test-case.Attributes[" name"].Value	/test-suite[@type='Assembly']/test-case.Attributes[" name"].Value	/assemblies/assembly/collection/test.Attributes[" method"].Value
	Duration	/TestRun/Results/UnitTestResult.Attributes[" duration"].Value Or /TestRun/Results/WebTestResult.Attributes["duration"].Value Or /TestRun/Results/TestResultAggregation.Attributes["duration"].Value	/testsuites/testsuite/testcase.Attributes[" time"].Value	/test-results/results/test-case.Attributes[" time"].Value	/test-suite[@type='Assembly']/test-case.Attributes[" duration"].Value	/assemblies/assembly/collection/test.Attributes[" time"].Value
	Date started	/TestRun/Results/UnitTestResult.Attributes[" startTime"].Value Or /TestRun/Results/WebTestResult.Attributes["startTime"].Value Or /TestRun/Results/TestResultAggregation.Attributes["startTime"].Value	/testsuites/testsuite.Attributes[" timestamp"].Value	/test-results.Attributes[" date"].Value + /test-results.Attributes["time"].Value	/test-suite[@type='Assembly']/test-case.Attributes[" start-time"].Value	/assemblies/assembly/ run-date + /assemblies/assembly/ run-time

SCOPE	FIELD	VISUAL STUDIO TEST (TRX)	JUNIT	NUNIT 2	NUNIT 3	XUNIT 2
	Date completed	/TestRun/Results/UnitTestResult.Attributes[" startTime"].Value + /TestRun/Results/UnitTestResult.Attributes["duration"].Value Or /TestRun/Results/WebTestResult.Attributes["startTim e"].Value + /TestRun/Results/WebTestResult.Attributes["duration"].Value Or /TestRun/Results/TestResultAggregation.Attributes["startTime"].Val ue + /TestRun/Results/TestResultAggregation.Attributes["duration"].Val ue	/testsuites/testsuite.Attributes[" timesta mp"].Value + /testsuites/testsuite/testcase.Attributes["time"].Value	Date started + /test-results/results /test-case.Attributes[" time"].Value	/test-suite[@type='Assembly']/test-case.Attributes[" end-time"].Value	Date started + /assemblies/assembly/collection/test.Attributes[" time"].Value
	Owner	/TestRun/TestDefinitions/UnitTest/Owners/Owners/Owner.Attributes[" name"].Value	/testsuites/testsuite/testcase/Attributes[" owner"].Value	build or release requested for user	build or release requested for user	/assemblies/assembly/collection/test/trait[@name='owner'].Attributes[" valu e"].Value
	Outcome	/TestRun/Results/UnitTestResult.Attributes[" outcome"].Value Or /TestRun/Results/WebTestResult.Attributes["outcom e"].Value Or /TestRun/Results/TestResultAggregation.Attributes["outcome"].Val ue	Failed: if exists /Testsuites/testsuite/testcase/ failure Or /Testsuites/testsuite/testcase/ error Not Executed: if exists Testsuites/testsuite/testcase/ skipped Passed: for all other cases	Failed: if exists /test-results/results /test-case/ failure Not Executed: if exists /test-results/results /test-case.Attributes[" result"].Value == "Ignored" Passed: for all other cases	/test-results/test-suite/results/test-case.Attributes[" result"].Value	/assemblies/assembly/collection/test/failure.Attributes[" result"].Value

SCOPE	FIELD	VISUAL STUDIO TEST (TRX)	JUNIT	NUNIT 2	NUNIT 3	XUNIT 2
	Error message	/TestRun/Results/UnitTestResult/Output/ErrorInfo/ Message.InnerText Or /TestRun/Results/WebTestResultOutput/ErrorInfo/ Message.InnerText Or /TestRun/Results/TestResultAggregation/Output/ErrorInfo/ Message.InnerText	/Testsuites/testsuite/testcase/failure.Attributes[" message "].Value Or /Testsuites/testsuite/testcase/error.Attributes[" message "].Value Or /Testsuites/testsuite/testcase/skipped.Attributes[" message "].Value	/test-results/results /test-casefailure/ message.InnerText	/test-suite[@type='Assembly']/test-casefailure/ message	/assemblies/assembly/collection/test/failure/ message
	Stack trace	/TestRun/Results/UnitTestResult/Output/ErrorInfo/ StackTrace.InnerText Or /TestRun/Results/WebTestResultOutput/ErrorInfo/ StackTrace.InnerText Or /TestRun/Results/TestResultAggregation/Output/ErrorInfo/ StackTrace.InnerText	/Testsuites/testsuite/testcase/failure. innerText Or /Testsuites/testsuite/testcase/error. innerText	/test-results/results /test-casefailure/ stack-trace.InnerText	/test-suite[@type='Assembly']//test-casefailure/ stack-trace	/assemblies/assembly/collection/test/failure/ stack-trace
	Attachments	Refer to Attachments support section below	-	-	Refer to Attachments support section below	-

SCOPE	FIELD	VISUAL STUDIO TEST (TRX)	JUNIT	NUNIT 2	NUNIT 3	XUNIT 2
	Console log	/TestRun/Results/UnitTestResult/Output/ StdOut.InnerText Or /TestRun/Results/WebTestResultOutput/Output/ StdOut.InnerText Or /TestRun/Results/TestResultAggregation/Output/ StdOut.InnerText	/Testsuites/testsuite/testcase/ system-out	/test-results/results/test-case/failure/ message.InnerText	/test-suite[@type='Assembly']/test-case/failure/ output	/assemblies/assembly/collection/test/failure/ output
	Console error log	/TestRun/Results/UnitTestResult/Output/ StdErr.InnerText Or /TestRun/Results/WebTestResultOutput/Output/ StdErr.InnerText Or /TestRun/Results/TestResultAggregation/Output/ StdErr.InnerText	/Testsuites/testsuite/testcase/ system-err	/test-results/results/test-case/ output.InnerText	-	-
	Agent name	/TestRun/Results/UnitTestResult.Attributes[" computerName"].Value Or /TestRun/Results/WebTestResult.Attributes["computerName"].Value Or /TestRun/Results/TestResultAggregation.Attributes["computerName"].Value	/testsuites/testsuite.Attributes[" hostname"].Value	/test-results/environment.Attributes[" machine-name"].Value	/test-suite[@type='Assembly']/environment.Attributes[" machine-name"].Value	-

SCOPE	FIELD	VISUAL STUDIO TEST (TRX)	JUNIT	NUNIT 2	NUNIT 3	XUNIT 2
	Test file	/TestRun/Test Definitions/UnitTest.Attribute["storage"].Value	/testsuites/testsuite/testcase/Attributes["classname"].Value	/test-results/test-suite.Attributes["name"].Value	/test-suite[@type='Assembly'].Attributes["name"].Value	/assemblies/assembly.Attributes["name"].Value
	Priority	/TestRun/Test Definitions/UnitTest.Attribute["priority"].Value	-	-	-	/testcaseNode/traits/trait[@name='priority'].Attributes["value"].Value

Docker

For Docker based apps there are many ways to build your application and run tests:

- **Build and test in a build pipeline:** build and tests execute in the pipeline and test results are published using the **Publish Test Results** task.
- **Build and test with a multi-stage Docker file:** build and tests execute inside the container using a multi-stage Docker file, as such test results are not published back to the pipeline.
- **Build, test, and publish results with a Docker file:** build and tests execute inside the container and results are published back to the pipeline. See the example below.

Build, test, and publish results with a Docker file

In this approach, you build your code and run tests inside the container using a Docker file. The test results are then copied to the host to be published to the pipeline. To publish the test results to Azure Pipelines, you can use the **Publish Test Results** task. The final image will be published to Docker or Azure Container Registry

Get the code

1. Import into Azure DevOps or fork into GitHub the following repository. This sample code includes a `Dockerfile` file at the root of the repository along with `.vsts-ci.docker.yml` file.

```
https://github.com/MicrosoftDocs/pipelines-dotnet-core
```

2. Create a `Dockerfile.build` file at the root of the directory with the following:

```
# Build and run tests inside the docker container
FROM microsoft/dotnet:2.1-sdk
WORKDIR /app
# copy the contents of agent working directory on host to workdir in container
COPY . .
# dotnet commands to build, test, and publish
RUN dotnet restore
RUN dotnet build -c Release
RUN dotnet test dotnetcore-tests/dotnetcore-tests.csproj -c Release --logger
"trx;LogFileName=testresults.trx"
RUN dotnet publish -c Release -o out
ENTRYPOINT dotnet dotnetcore-sample/out/dotnetcore-sample.dll
```

This file contains the instructions to build code and run tests. The tests are then copied to a file `testresults.trx` inside the container.

3. To make the final image as small as possible, containing only the runtime and deployment artifacts, replace the contents of the existing `Dockerfile` with the following:

```
# This Dockerfile creates the final image to be published to Docker or
# Azure Container Registry
# Create a container with the compiled asp.net core app
FROM microsoft/aspnetcore:2.0
# Create app directory
WORKDIR /app
# Copy only the deployment artifacts
COPY /out .
ENTRYPOINT ["dotnet", "dotnetcore-sample.dll"]
```

Define the build pipeline

- [YAML](#)
- [Designer](#)

1. If you have a Docker Hub account, and want to push the image to your Docker registry, replace the contents of the `.vsts-ci.docker.yml` file with the following:

```
# Build Docker image for this app, to be published to Docker Registry
pool:
  vmImage: 'ubuntu-16.04'
variables:
  buildConfiguration: 'Release'
steps:
- script: |
    docker build -f Dockerfile.build -t $(dockerId)/dotnetcore-build:$BUILD_BUILDID .
    docker run --name dotnetcoreapp --rm -d $(dockerId)/dotnetcore-build:$BUILD_BUILDID
    docker cp dotnetcoreapp:/app/dotnetcore-tests/TestResults $(System.DefaultWorkingDirectory)
    docker cp dotnetcoreapp:/app/dotnetcore-sample/out $(System.DefaultWorkingDirectory)
    docker stop dotnetcoreapp

- task: PublishTestResults@2
  inputs:
    testRunner: VSTest
    testResultsFiles: '**/*.trx'

- script: |
    docker build -f Dockerfile -t $(dockerId)/dotnetcore-sample:$BUILD_BUILDID .
    docker login -u $(dockerId) -p $pswd
    docker push $(dockerId)/dotnetcore-sample:$BUILD_BUILDID
  env:
    pswd: $(dockerPassword)
```

Alternatively, if you configure an Azure Container Registry and want to push the image to that registry, replace the contents of the `.vsts-ci.yml` file with the following:

```

# Build Docker image for this app to be published to Azure Container Registry
pool:
  vmImage: 'ubuntu-16.04'
variables:
  buildConfiguration: 'Release'

steps:
- script: |
    docker build -f Dockerfile.build -t $(dockerId)/dotnetcore-build:$BUILD_BUILDID .
    docker run --name dotnetcoreapp --rm -d $(dockerId)/dotnetcore-build:$BUILD_BUILDID
    docker cp dotnetcoreapp:app/dotnetcore-tests/TestResults $(System.DefaultWorkingDirectory)
    docker cp dotnetcoreapp:app/dotnetcore-sample/out $(System.DefaultWorkingDirectory)
    docker stop dotnetcoreapp

- task: PublishTestResults@2
  inputs:
    testRunner: VSTest
    testResultsFiles: '**/*.trx'

- script: |
    docker build -f Dockerfile -t $(dockerId).azurecr.io/dotnetcore-sample:$BUILD_BUILDID .
    docker login -u $(dockerId) -p $pswd $(dockerId).azurecr.io
    docker push $(dockerId).azurecr.io/dotnetcore-sample:$BUILD_BUILDID
  env:
    pswd: $(dockerPassword)

```

2. Push the change to the master branch in your repository.
3. If you use Azure Container Registry, ensure you have [pre-created the registry](#) in the Azure portal. Copy the admin user name and password shown in the **Access keys** section of the registry settings in Azure portal.
4. Update your build pipeline with the following

- **Agent pool:** Hosted Ubuntu 1604
 - **dockerId:** Set the value to your Docker ID for DockerHub or the admin user name for Azure Container Registry.
 - **dockerPassword:** Set the value to your password for DockerHub or the admin password Azure Container Registry.
- **YAML file path:** ./vsts-ci.docker.yml

5. Queue a new build and watch it create and push a Docker image to your registry and the test results to Azure DevOps.

YAML builds are not yet available on TFS.

Attachments support

The Publish Test Results task provides support for attachments for both test run and test results for the following formats.

Visual Studio Test (TRX)

SCOPE	TYPE	PATH
Test run	Data Collector	/TestRun/ResultSummary/CollectorDataEntries/Collector/UriAttachments/UriAttachment/A.Attributes["href"].Value

SCOPE	TYPE	PATH
	Test Result	/TestRun/ResultSummary/ResultFiles/ResultFile.Attributes["path"].Value
	Code Coverage	/TestRun/TestSettings/Execution/AgentRule/DataCollectors/DataCollector/Configuration/CodeCoverage/Regular/CodeCoverageItem.Attributes["binaryFile"].Value And /TestRun/TestSettings/Execution/AgentRule/DataCollectors/DataCollector/Configuration/CodeCoverage/Regular/CodeCoverageItem.Attributes["pdbFile"].Value
Test result	Data Collectors	/TestRun/Results/UnitTestResult/CollectorDataEntries/Collector/UriAttachments/UriAttachment/A.Attributes["href"].Value Or /TestRun/Results/WebTestResult/CollectorDataEntries/Collector/UriAttachments/UriAttachment/A.Attributes["href"].Value Or /TestRun/Results/TestResultAggregation/CollectorDataEntries/Collector/UriAttachments/UriAttachment/A.Attributes["href"].Value
	Test Result	/TestRun/Results/UnitTestResult/ResultFiles/ResultFile.Attributes["path"].Value Or /TestRun/Results/WebTestResult/ResultFiles/ResultFile.Attributes["path"].Value Or /TestRun/Results/TestResultAggregation/ResultFiles/ResultFile.Attributes["path"].Value

NUnit 3

SCOPE	PATH
Test run	/test-suite/attachments/attachment/ filePath
Test run	/test-suite[@type='Assembly']/test-case/attachments/attachment/ filePath

NOTE

The option to upload the test results file as an attachment is a default option in the task, applicable to all formats.

Related tasks

- [Visual Studio Test](#)
- [Publish Code Coverage Results](#)

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Help and support

- Report problems through the [Developer Community](#).

Run Functional Tests task

11/15/2018 • 8 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

This task is deprecated in Azure Pipelines and TFS 2018 and later. Use version 2.x or higher of the [Visual Studio Test](#) task together with [jobs](#) to run unit and functional tests on the universal agent.

For more details, see [Testing with unified agents and jobs](#).

TFS 2017 and earlier

Use this task in a build or release pipeline to run Coded UI tests, Selenium tests, and functional tests on a set of machines using the test agent. Use this task when you want to run tests on remote machines, and you cannot run tests on the build machine.

Demands and prerequisites

This task must be preceded by a **Visual Studio Test Agent Deployment** task.

YAML snippet

```
# Run Functional Tests
# Deprecated: This task and it's companion task (Visual Studio Test Agent Deployment) are deprecated. Use the 'Visual Studio Test' task instead. The VSTest task can run unit as well as functional tests. Run tests on one or more agents using the multi-agent job setting. Use the 'Visual Studio Test Platform' task to run tests without needing Visual Studio on the agent. VSTest task also brings new capabilities such as automatically rerunning failed tests.
- task: RunVisualStudioTestsusingTestAgent@1
  inputs:
    testMachineGroup:
    dropLocation:
    #testSelection: 'testAssembly' # Options: testAssembly, testPlan
    #testPlan: # Required when testSelection == TestPlan
    #testSuite: # Required when testSelection == TestPlan
    #testConfiguration: # Required when testSelection == TestPlan
    #sourcefilters: '**\*test*.dll' # Required when testSelection == TestAssembly
    #testFilterCriteria: # Optional
    #runSettingsFile: # Optional
    #overrideRunParams: # Optional
    #codeCoverageEnabled: false # Optional
    #customSlicingEnabled: false # Optional
    #testRunTitle: # Optional
    #platform: # Optional
    #configuration: # Optional
    #testConfigurations: # Optional
    #autMachineGroup: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
----------	-------------

ARGUMENT	DESCRIPTION
Machines	<p>A comma-separated list of machine FQDNs or IP addresses, optionally including the port number. The maximum is 32 machines (or 32 agents). Can be:</p> <ul style="list-style-type: none"> - The name of an Azure Resource Group. - A comma-delimited list of machine names. Example: <code>dbserver.fabrikam.com, dbserver_int.fabrikam.com:5986, 192.168.34:5</code> - An output variable from a previous task.
Test Drop Location	<p>Required. The location on the test machine(s) where the test binaries have been copied by a Windows Machine File Copy or Azure File Copy task. System stage variables from the test agent machines can be used to specify the drop location. Examples: <code>c:\tests</code> and <code>%systemdrive%\Tests</code></p>
Test Selection	<p>Required. Whether the tests are to be selected from test assemblies or from a test plan.</p>
Test Assembly	<p>Required when Test Selection is set to Test Assembly. The test assemblies from which the tests should be executed. Paths are relative to the sources directory.</p> <ul style="list-style-type: none"> - Separate multiple paths with a semicolon. - Default is <code>***test*.dll</code> - For JavaScript tests, enter the path and name of the <code>.js</code> files containing the tests. - Wildcards can be used. Example: <code>**\commontests*test*.dll; **\frontendtests*test*.dll</code>
Test Filter criteria	<p>Optional when Test Selection is set to Test Assembly. A filter to specify the tests to execute within the test assembly files. Works the same way as the <code>/TestCaseFilter</code> option of <code>vstest.console.exe</code>. Example: <code>Priority=1 Name=MyTestMethod</code></p>
Test Plan	<p>Required if Test Suite is not specified when Test Selection is set to Test Plan. Select a test plan already configured for this organization.</p>
Test Suite	<p>Required if Test Plan is not specified when Test Selection is set to Test Plan. Select a test suite from the selected test plan.</p>
Test Configuration	<p>Optional when Test Selection is set to Test Plan. Select a test configuration from the selected test plan.</p>
Run Settings File	<p>Optional. The path to a <code>.runsettings</code> or <code>.testsettings</code> file on the build machine. Can be the path to a file in the repository or a file on disk. Use <code>\$(Build.SourcesDirectory)</code> to specify the project root folder.</p>
Override Test Run Parameters	<p>Optional. A string containing parameter overrides for parameters defined in the <code>TestRunParameters</code> section of the <code>.runsettings</code> file. Example: <code>Platform=\$(platform);Port=8080</code></p>
Code Coverage Enabled	<p>When set, the task will collect code coverage information during the run and upload the results to the server. Supported for .NET and C++ projects only.</p>
Distribute tests by number of machines	<p>When checked, distributes tests based on the number of machines, instead of distributing tests at the assembly level, irrespective of the container assemblies passed to the task.</p>

ARGUMENT	DESCRIPTION
Test Run Title	Optional. A name for this test run, used to identify it for reporting and in comparison with other test runs.
Platform	Optional. The build platform against which the test run should be reported. Used only for reporting. - If you are using the Build - Visual Studio template, this is automatically defined, such as <code>x64</code> or <code>x86</code> - If you have defined a variable for platform in your build task, use that here.
Configuration	Optional. The build configuration against which the test run should be reported. Used only for reporting. - If you are using the Build - Visual Studio template, this is automatically defined, such as <code>Debug</code> or <code>Release</code> - If you have defined a variable for configuration in your build task, use that here.
Test Configurations	Optional. A string that contains the filter(s) to report the configuration on which the test case was run. Used only for reporting with Microsoft Test Manager. - Syntax: {expression for test method name(s)} : {configuration ID from Microsoft Test Manager} - Example: <code>FullyQualifiedName~Chrome:12</code> to report all test methods that have Chrome in the Fully Qualified Name and map them to configuration ID 12 defined in Microsoft Test Manager. - Use <code>DefaultTestConfiguration:{Id}</code> as a catch-all.
Application Under Test Machines	A list of the machines on which the Application Under Test (AUT) is deployed, or on which a specific process such as W3WP.exe is running. Used to collect code coverage data from these machines. Use this in conjunction with the Code Coverage Enabled setting. The list can be a comma-delimited list of machine names or an output variable from an earlier task.
Control options	See Control options

The task supports a maximum of 32 machines/agents.

Scenarios

Typical scenarios include:

- Tests that require additional installations on the test machines, such as different browsers for Selenium tests
- Coded UI tests
- Tests that require a specific operating system configuration
- To execute a large number of unit tests more quickly by using multiple test machines

Use this task to:

- Run automated tests against on-premises standard environments
- Run automated tests against existing Azure environments
- Run automated tests against newly provisioned azure environments

You can run unit tests, integration tests, functional tests - in fact any test that you can execute using the Visual Studio test runner (`vstest`).

Using multiple machines in a Machine Group enables the task to run parallel distributed execution of tests. Parallelism is at the test assembly level, not at individual test level.

These scenarios are supported for:

- **TFS on-premises and Azure Pipelines**
- **Build agents**
 - [Hosted](#) and [on-premises](#) agents.
 - The build agent must be able to communicate with all test machines. If the test machines are on-premises behind a firewall, the hosted build agents cannot be used.
 - The build agent must have access to the Internet to download test agents. If this is not the case, the test agent must be manually downloaded and deployed to a network location that is accessible by the build agent, and a **Visual Studio Test Agent Deployment** task used with an appropriate path for the **Test Agent Location** parameter. Automatic checking for new test agent versions is not supported in this topology.
- **CI/CD workflow**
 - The Build-Deploy-Test (BDT) tasks are supported in both build and release pipelines.
- **Machine group configuration**
 - Only Windows machines are supported when using BDT tasks inside a Machine Group. Using Linux, macOS, or other platforms inside a Machine Group with BDT tasks is not supported.
 - Installing any version or release of Visual Studio on any of the test machines is not supported.
 - Installing an older version of the test agent on any of the test machines is not supported.
- **Test machine topologies**
 - Azure-based test machines are fully supported, both existing test machines and newly provisioned machines.
 - Domain-joined test machines are supported.
 - Workgroup-joined test machines must have HTTPS authentication enabled and configured during creation of the Machine Group.
 - Test agent machines must have network access to the Team Foundation Server instance. Test machines isolated on the network are not supported.
- **Usage Error Conditions**
 - Running tests across different Machine Groups, and running builds (with any BDT tasks) in parallel against these Machine Groups is not supported.
 - Cancelling an in-progress build or release with BDT tasks is not supported. If you do so, subsequent builds may not behave as expected.
 - Cancelling an in-progress test run queued through BDT tasks is not supported.
 - Configuring a test agent and running tests under a non-administrative account or under a service account is not supported.

More information

- [Using the Visual Studio Agent Deployment task on machines not connected to the internet](#)
- [Set up environments to run continuous test tasks with your builds](#)
- [Testing in Continuous Integration and Continuous Deployment Workflows](#)

Related tasks

- [Deploy Azure Resource Group](#)
- [Azure File Copy](#)
- [Windows Machine File Copy](#)
- [PowerShell on Target Machines](#)
- [Visual Studio Test Agent Deployment](#)

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

How do I create an Azure Resource Group for testing?

See [Using the Azure Portal to manage your Azure resources](#) and [Azure Resource Manager - Creating a Resource Group and a VNET](#).

Where can I get more information about the Run Settings file?

See [Configure unit tests by using a .runsettings file](#)

Where can I get more information about overriding settings in the Run Settings file?

See [Supplying Run Time Parameters to Tests](#)

How can I customize code coverage analysis and manage inclusions and exclusions

See [Customize Code Coverage Analysis](#)

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

::: moniker-end

Help and support

- Report problems through the [Developer Community](#).

Visual Studio Test task

11/19/2018 • 8 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to run unit and functional tests (Selenium, Appium, Coded UI test, and more) using the Visual Studio Test Runner. Other than MSTest-based tests, test frameworks that have a Visual Studio test adapter, such as xUnit, NUnit, Chutzpah, can also be executed.

Tests that target the .NET core framework can be executed by specifying the appropriate target framework value.

Tests can be distributed on multiple agents using version 2 of this task.

Demands

The agent must have the following capability:

vstest

The vstest demand can be satisfied in two ways:

1. Visual Studio is installed on the agent machine.
2. By using the [Visual Studio Test Platform Installer task](#) in the pipeline definition.

YAML snippet

```

# Visual Studio Test
# Run unit and functional tests (Selenium, Appium, Coded UI test, etc.) using the Visual Studio Test
(VsTest) runner. Test frameworks that have a Visual Studio test adapter such as MsTest, xUnit, NUnit,
Chutzpah (for JavaScript tests using QUnit, Mocha and Jasmine), etc. can be run. Tests can be
distributed on multiple agents using this task (version 2).
- task: VSTest@2
  inputs:
    #testSelector: 'testAssemblies' # Options: testAssemblies, testPlan, testRun
    #testAssemblyVer2: '**\*test*.dll!**\*TestAdapter.dll!**\obj\**' # Required when testSelector ==
TestAssemblies
    #testPlan: # Required when testSelector == TestPlan
    #testSuite: # Required when testSelector == TestPlan
    #testConfiguration: # Required when testSelector == TestPlan
    #tcmTestRun: '$(test.RunId)' # Optional
    #searchFolder: '$(System.DefaultWorkingDirectory)'
    #testFilterCriteria: # Optional
    #runOnlyImpactedTests: False # Optional
    #runAllTestsAfterXBuilds: '50' # Optional
    #uiTests: false # Optional
    #vstestLocationMethod: 'version' # Optional. Options: version, location
    #vsTestVersion: 'latest' # Optional. Options: latest, 15.0, 14.0, toolsInstaller
    #vstestLocation: # Optional
    #runSettingsFile: # Optional
    #overrideTestrunParameters: # Optional
    #pathToCustomTestAdapters: # Optional
    #runInParallel: False # Optional
    #runTestsInIsolation: False # Optional
    #codeCoverageEnabled: False # Optional
    #otherConsoleOptions: # Optional
    #distributionBatchType: 'basedOnTestCases' # Optional. Options: basedOnTestCases,
basedOnExecutionTime, basedOnAssembly
    #batchingBasedOnAgentsOption: 'autoBatchSize' # Optional. Options: autoBatchSize, customBatchSize
    #customBatchSizeValue: '10' # Required when distributionBatchType == BasedOnTestCases &&
BatchingBasedOnAgentsOption == CustomBatchSize
    #batchingBasedOnExecutionTimeOption: 'autoBatchSize' # Optional. Options: autoBatchSize,
customTimeBatchSize
    #customRunTimePerBatchValue: '60' # Required when distributionBatchType == BasedOnExecutionTime &&
BatchingBasedOnExecutionTimeOption == CustomTimeBatchSize
    #dontDistribute: False # Optional
    #testRunTitle: # Optional
    #platform: # Optional
    #configuration: # Optional
    #publishRunAttachments: true # Optional
    #diagnosticsEnabled: True # Optional
    #collectDumpOn: 'onAbortOnly' # Optional. Options: onAbortOnly, always, never
    #rerunFailedTests: False # Optional
    #rerunType: 'basedOnTestFailurePercentage' # Optional. Options: basedOnTestFailurePercentage,
basedOnTestFailureCount
    #rerunFailedThreshold: '30' # Optional
    #rerunFailedTestCasesMaxLimit: '5' # Optional
    #rerunMaxAttempts: '3' # Optional

```

Arguments

ARGUMENT	DESCRIPTION
----------	-------------

ARGUMENT	DESCRIPTION
testSelector Select tests using	<p>(Required)</p> <ul style="list-style-type: none"> • Test assembly: Use this option to specify one or more test assemblies that contain your tests. You can optionally specify a filter criteria to select only specific tests. • Test plan: Use this option to run tests from your test plan that have an automated test method associated with it. To learn more about how to associate tests with a test case work item, see Associate automated tests with test cases. • Test run: Use this option when you are setting up an environment to run tests from test plans. This option should not be used when running tests in a continuous integration /continuous deployment (CI/CD) pipeline.
testAssemblyVer2 Test assemblies	<p>(Required) Run tests from the specified files. Ordered tests and webtests can be run by specifying the .orderedtest and .webtest files respectively. To run .webtest, Visual Studio 2017 Update 4 or higher is needed.</p> <p>The file paths are relative to the search folder. Supports multiple lines of minimatch patterns. More Information</p>
testPlan Test plan	<p>(Required) Select a test plan containing test suites with automated test cases.</p>
testSuite Test suite	<p>(Required) Select one or more test suites containing automated test cases. Test case work items must be associated with an automated test method. Learn more.</p>
testConfiguration Test configuration	<p>(Required) Select Test Configuration.</p>
tcmTestRun Test Run	<p>(Optional) Test run based selection is used when triggering automated test runs from test plans. This option cannot be used for running tests in the CI/CD pipeline.</p>
searchFolder Search folder	<p>(Required) Folder to search for the test assemblies.</p>
testFiltercriteria Test filter criteria	<p>(Optional) Additional criteria to filter tests from Test assemblies. For example: <input type="text" value="Priority=1 Name=MyTestMethod"/> More information</p>
runOnlyImpactedTests Run only impacted tests	<p>(Optional) Automatically select, and run only the tests needed to validate the code change. More information</p>
runAllTestsAfterXBuilds Number of builds after which all tests should be run	<p>(Optional) Number of builds after which to automatically run all tests. Test Impact Analysis stores the mapping between test cases and source code. It is recommended to regenerate the mapping by running all tests, on a regular basis.</p>

Argument	Description
uiTests Test mix contains UI tests	(Optional) To run UI tests, ensure that the agent is set to run in interactive mode with autologon enabled . Setting up an agent to run interactively must be done before queueing the build / release. Checking this box does not configure the agent in interactive mode automatically. This option in the task is to only serve as a reminder to configure agent appropriately to avoid failures. Hosted Windows agents from the VS 2015 and 2017 pools can be used to run UI tests.
vstestLocationMethod Select test platform using	(Optional) Specify which test platform should be used.
vsTestVersion Test platform version	(Optional) The version of Visual Studio test to use. If latest is specified it chooses Visual Studio 2017 or Visual Studio 2015 depending on what is installed. Visual Studio 2013 is not supported. To run tests without needing Visual Studio on the agent, use the 'Installed by tools installer' option. Be sure to include the 'Visual Studio Test Platform Installer' task to acquire the test platform from NuGet.
vstestLocation Path to vstest.console.exe	(Optional) Specify the path to VSTest.
runSettingsFile Settings file	(Optional) Path to runsettings or testsettings file to use with the tests. Starting with Visual Studio 15.7, it is recommended to use runsettings for all types of tests. To learn more about converting a .testsettings file to a .runsettings file, see this topic .
overrideTestRunParameters Override test run parameters	(Optional) Override parameters defined in the <code>TestRunParameters</code> section of runsettings file or <code>Properties</code> section of testsettings file. For example: <code>-key1 value1 -key2 value2</code> . Note: Properties specified in testsettings file can be accessed via the <code>HttpContext</code> using Visual Studio 2017 Update 4 or higher
pathToCustomTestAdapters Path to custom test adapters	(Optional) Directory path to custom test adapters. Adapters residing in the same folder as the test assemblies are automatically discovered.
runInParallel Run tests in parallel on multi-core machines	(Optional) If set, tests will run in parallel leveraging available cores of the machine. This will override the <code>MaxCpuCount</code> if specified in your runsettings file. Click here to learn more about how tests are run in parallel.
runTestsInIsolation Run tests in isolation	(Optional) Runs the tests in an isolated process. This makes <code>vstest.console.exe</code> process less likely to be stopped on an error in the tests, but tests might run slower. This option currently cannot be used when running with the multi-agent job setting.
codeCoverageEnabled Code coverage enabled	(Optional) Collect code coverage information from the test run.

Argument	Description
otherConsoleOptions Other console options	<p>(Optional) Other console options that can be passed to vstest.console.exe, as documented here.</p> <p>These options are not supported and will be ignored when running tests using the 'Multi agent' parallel setting of an agent job or when running tests using 'Test plan' option. The options can be specified using a settings file instead.</p>
distributionBatchType Batch tests	<p>(Optional) A batch is a group of tests. A batch of tests runs at a time and results are published for that batch. If the job in which the task runs is set to use multiple agents, each agent picks up any available batches of tests to run in parallel.</p> <p>Based on number of tests and agents: Simple batching based on the number of tests and agents participating in the test run.</p> <p>Based on past running time of tests: This batching considers past running time to create batches of tests such that each batch has approximately equal running time.</p> <p>Based on test assemblies: Tests from an assembly are batched together.</p>
batchingBasedOnAgentsOption Batch options	<p>(Optional) Simple batching based on the number of tests and agents participating in the test run. When the batch size is automatically determined, each batch contains $\lceil \text{total number of tests} / \text{number of agents} \rceil$ tests. If a batch size is specified, each batch will contain the specified number of tests.</p>
customBatchSizeValue Number of tests per batch	(Required) Specify batch size
batchingBasedOnExecutionTimeOption Batch options	<p>(Optional) This batching considers past running time to create batches of tests such that each batch has approximately equal running time. Quick running tests will be batched together, while longer running tests may belong to a separate batch. When this option is used with the multi-agent job setting, total test time is reduced to a minimum.</p>
customRunTimePerBatchValue Running time (sec) per batch	(Required) Specify the running time (sec) per batch
dontDistribute Do not distribute tests and replicate instead when multiple agents are used in the job	<p>(Optional) Choosing this option will not distribute tests across agents when the task is running in a multi-agent job.</p> <p>Each of the selected test(s) will be repeated on each agent. The option is not applicable when the agent job is configured to run with no parallelism or with the multi-config option.</p>
testRunTitle Test run title	(Optional) Provide a name for the test run.

ARGUMENT	DESCRIPTION
platform Build platform	(Optional) Build platform against which the tests should be reported. If you have defined a variable for platform in your build task, use that here.
configuration Build configuration	(Optional) Build configuration against which the tests should be reported. If you have defined a variable for configuration in your build task, use that here.
publishRunAttachments Upload test attachments	(Optional) Opt in/out of publishing run level attachments.
rerunFailedTests Rerun failed tests	(Optional) Selecting this option will rerun any failed tests until they pass or the maximum # of attempts is reached.
rerunType Do not rerun if test failures exceed specified threshold	(Optional) Use this option to avoid rerunning tests when failure rate crosses the specified threshold. This is applicable if any environment issues leads to massive failures. You can specify % failures or # of failed tests as a threshold.
rerunFailedThreshold % failure	(Optional) Use this option to avoid rerunning tests when failure rate crosses the specified threshold. This is applicable if any environment issues leads to massive failures.
rerunFailedTestCasesMaxLimit # of failed tests	(Optional) Use this option to avoid rerunning tests when number of failed test cases crosses specified limit. This is applicable if any environment issues leads to massive failures.
rerunMaxAttempts Maximum # of attempts	(Optional) Specify the maximum # of times a failed test should be retried. If a test passes before the maximum # of attempts is reached, it will not be rerun further.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Visual Studio Test Agent Deployment task

11/19/2018 • 8 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

This task is deprecated in Azure Pipelines and TFS 2018 and later. Use version 2.x or higher of the [Visual Studio Test](#) task together with [jobs](#) to run unit and functional tests on the universal agent. For more details, see [Testing with unified agents and jobs](#).

TFS 2017 and earlier

Use this task in a build or release pipeline to deploy and configure the test agent to run tests on a set of machines. The test agent deployed by this task can collect data or run distributed tests using the [Visual Studio Test](#) task.

Demands and prerequisites

This task requires the target computer to have:

- Windows 7 Service Pack 1 or Windows 2008 R2 Service Pack 2 or higher
- .NET 4.5 or higher
- PSRemoting enabled by running the [Enable-PSRemoting](#) PowerShell script

Windows Remote Management (WinRM)

This task uses [Windows Remote Management](#) (WinRM) to access on-premises physical computers or virtual computers that are domain-joined or workgroup-joined.

To set up WinRM for on-premises physical computers or virtual machines

Follow the steps described in [domain-joined](#)

To set up WinRM for Microsoft Azure Virtual Machines

Azure Virtual Machines require WinRM to use the HTTPS protocol. You can use a self-signed Test Certificate. In this case, the automation agent will not validate the authenticity of the certificate as being issued by a trusted certification authority.

- **Azure Classic Virtual Machines.** When you create a [classic virtual machine](#) from the Azure portal, the virtual machine is already set up for WinRM over HTTPS, with the default port 5986 already opened in the firewall and a self-signed certificate installed on the machine. These virtual machines can be accessed with no further configuration required. Existing Classic virtual machines can be also selected by using the [Azure Resource Group Deployment](#) task.
- **Azure Resource Group.** If you have an [Azure Resource Group](#) already defined in the Azure portal, you must configure it to use the WinRM HTTPS protocol. You need to open port 5986 in the firewall, and install a self-signed certificate.

To dynamically deploy Azure Resource Groups that contain virtual machines, use the [Azure Resource Group Deployment](#) task. This task has a checkbox named **Enable Deployment Pre-requisites**. Select this to automatically set up the WinRM HTTPS protocol on the virtual machines, open port 5986 in the firewall, and install a test certificate. The virtual machines are then ready for use in the deployment task.

YAML snippet

```

# Visual Studio Test Agent Deployment
# Deprecated: This task and it's companion task (Run Functional Tests) are deprecated. Use the 'Visual Studio Test' task instead. The VSTest task can run unit as well as functional tests. Run tests on one or more agents using the multi-agent job setting. Use the 'Visual Studio Test Platform' task to run tests without needing Visual Studio on the agent. VSTest task also brings new capabilities such as automatically rerunning failed tests.
- task: DeployVisualStudioTestAgent@2
  inputs:
    testMachines:
    adminUserName:
    adminPassword:
    #winRmProtocol: 'Http' # Options: http, https
    #testCertificate: true # Optional
    machineUserName:
    machinePassword:
    #runAsProcess: false # Optional
    #isDataCollectionOnly: false # Optional
    #testPlatform: '14.0' # Optional. Options: 15.0, 14.0
    #agentLocation: # Optional
    #updateTestAgent: false # Optional

```

Arguments

ARGUMENT	DESCRIPTION
Machines	<p>A comma-separated list of machine FQDNs or IP addresses, optionally including the port number. The maximum is 32 machines (or 32 agents). Can be:</p> <ul style="list-style-type: none"> - The name of an Azure Resource Group. - A comma-delimited list of machine names. Example: dbserver.fabrikam.com, dbserver_int.fabrikam.com:5986, 192.168.34:5 - An output variable from a previous task.
Admin Login	<p>The username of either a domain or a local administrative account on the target host(s). This parameter is required when used with a list of machines. It is optional when specifying a machine group and, if specified, overrides the credential settings defined for the machine group.</p> <ul style="list-style-type: none"> - Formats such as username, domain\username, machinename\username, and .\\username are supported. - UPN formats such as username@domain.com and built-in system accounts such as NT Authority\System are not supported.
Password	<p>The password for the administrative account specified above. This parameter is required when used with a list of machines. It is optional when specifying a machine group and, if specified, overrides the credential settings defined for the machine group. Consider using a secret variable global to the build or release pipeline to hide the password. Example: <code>\$(passwordVariable)</code></p>
Protocol	<p>The protocol that will be used to connect to the target host, either HTTP or HTTPS.</p>
Agent Configuration - Username	<p>Required. The username that the test agent will use. Must be an account on the test machines that has administrative permissions.</p> <ul style="list-style-type: none"> - Formats such as username, domain\username, machinename\username, and .\\username are supported. - UPN formats such as username@domain.com and built-in system accounts such as NT Authority\System are not supported.
Agent Configuration - Password	<p>Required. The password for the Username for the test agent. To protect the password, create a variable and use the "padlock" icon to hide it.</p>

ARGUMENT	DESCRIPTION
Agent Configuration - Run UI tests	When set, the test agent will run as an interactive process. This is required when interacting with UI elements or starting applications during the tests. For example, Coded UI or Selenium tests that are running on full fidelity browsers will require this option to be set.
Agent Configuration - Enable data collection only	When set, the test agent will return previously collected data and not re-run the tests. At present this is only available for Code Coverage. Also see Q&A section below.
Advanced - Test agent version	The version of the test agent to use.
Advanced - Test agent location	Optional. The path to the test agent (vstf_testagent.exe) if different from the default path. - If you use a copy of the test agent located on your local computer or network, specify the path to that instance. - The location must be accessible by either the build agent (using the identity it is running under) or the test agent (using the identity configured above). - For Azure test machines, the web location can be used.
Advanced - Update test agent	If set, and the test agent is already installed on the test machines, the task will check if a new version of the test agent is available.
Control options	See Control options

The task supports a maximum of 32 machines/agents.

Supported scenarios

Use this task for:

- Running automated tests against on-premises standard environments
- Running automated tests against existing Azure environments
- Running automated tests against newly provisioned Azure environments

The supported options for these scenarios are:

- **TFS**
 - On-premises and Azure Pipelines
- **Build and release agents**
 - Hosted and on-premises agents are supported.
 - The agent must be able to communicate with all test machines. If the test machines are on-premises behind a firewall, an Azure Pipelines Microsoft-hosted agent cannot be used because it will not be able to communicate with the test machines.
 - The agent must have Internet access to download test agents. If this is not the case, the test agent must be manually downloaded, uploaded to a network location accessible to the agent, and the **Test Agent Location** parameter used to specify the location. The user must manually check for new versions of the agent and update the test machines.
- **Continuous integration/continuous deployment workflows**
 - Build/deploy/test tasks are supported in both build and release workflows.
- **Machine group configuration**
 - Only Windows-based machines are supported inside a machine group for build/deploy/test tasks. Linux, macOS, or other platforms are not supported inside a machine group.
 - Installing any version of Visual Studio on any of the test machines is not supported.

- Installing any older version of the test agent on any of the test machines is not supported.

● **Test machine topologies**

- Azure-based test machines are fully supported, both existing test machines and newly provisioned test machines.
- Machines with the test agent installed must have network access to the TFS instance in use. Network-isolated test machines are not supported.
- Domain-joined test machines are supported.
- Workgroup-joined test machines must use HTTPS authentication configured during machine group creation.

● **Usage Error Conditions**

- Using the same test machines across different machine groups, and running builds (with any build/deploy/test tasks) in parallel against those machine groups is not supported.
- Cancelling an in-progress build or release that contains any build/deploy/test tasks is not supported. If you do cancel, behavior of subsequent builds may be unpredictable.
- Cancelling an ongoing test run queued through build/deploy/test tasks is not supported.
- Configuring the test agent and running tests as a non-administrator, or by using a service account, is not supported.
- Running tests for Universal Windows Platform apps is not supported. Use the [Visual Studio Test task](#) to run these tests.

Example

- [Testing in Continuous Integration and Continuous Deployment Workflows](#)

More information

- [Using the Visual Studio Agent Deployment task on machines not connected to the internet](#)
- [Set up automated testing for your builds](#)
- [Source code for this task](#)

Related tasks

- [Visual Studio Test](#)
- [Azure File Copy](#)
- [Windows Machine File Copy](#)
- [PowerShell on Target Machines](#)

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

When would I use the Enable Data Collection Only option?

An example would be in a client-server application model, where you deploy the test agent on the servers and use another task to deploy the test agent to test machines. This enables you to collect data from both server and client machines without triggering the execution of tests on the server machines.

How do I create an Azure Resource Group for testing?

See [Using the Azure Portal to manage your Azure resources](#) and [Azure Resource Manager - Creating a Resource Group and a VNET](#).

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are

available on-premises if you have [upgraded to the latest version of TFS](#).

Help and support

- Report problems through the [Developer Community](#).

CocoaPods task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015

Use this task in a build or release pipeline to run CocoaPods `pod install`.

CocoaPods is the dependency manager for Swift and Objective-C Cocoa projects. This task optionally runs `pod repo update` and then runs `pod install`.

Demands

None

YAML snippet

```
# CocoaPods
# CocoaPods is a dependency manager for Swift and Objective-C Cocoa projects. This task runs 'pod install'.
- task: CocoaPods@0
  inputs:
    #workingDirectory: # Optional
    forceRepoUpdate:
    #projectDirectory: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Working directory	(Optional) Specify the working directory in which to execute this task. If left empty, the repository directory will be used.
Force repo update	(Required) Selecting this option will force running 'pod repo update' before install.
Project directory	(Optional) Optionally specify the path to the root of the project directory. If left empty, the project specified in the Podfile will be used. If no project is specified, then a search for an Xcode project will be made. If more than one Xcode project is found, an error will occur.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

What other kinds of apps can I build?

[Build your app](#)

What other kinds of build tasks are available?

[Specify your build tasks](#)

How do we protect our codebase from build breaks?

- Git: [Improve code quality with branch policies](#) with an option to require that code builds before it can be merged to a branch. For GitHub repositories, similar policies are available in GitHub's repository settings under *Branches*.
- TFVC: [Use gated check-in](#).

How do I modify other parts of my build pipeline?

- [Specify your build tasks](#) to run tests, scripts, and a wide range of other processes.
- [Specify build options](#) such as specifying how completed builds are named, building multiple configurations, creating work items on failure.
- [Specify the repository](#) to pick the source of the build and modify options such as how the agent workspace is cleaned.
- [Set build triggers](#) to modify how your CI builds run and to specify scheduled builds.
- [Specify build retention policies](#) to automatically delete old builds.

I selected parallel multi-configuration, but only one build is running at a time.

If you're using Azure Pipelines, you might need more parallel jobs. See [Parallel jobs in Azure Pipelines](#).

How do I see what has changed in my build pipeline?

[View the change history of your build pipeline](#)

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Conda Environment task

11/19/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to create and activate a Conda environment.

This task will create a Conda environment and activate it for subsequent build tasks.

If the task finds an existing environment with the same name, the task will simply reactivate it. This is possible on self-hosted agents. To recreate the environment and reinstall any of its packages, set the "Clean the environment" option.

Running with the "Update to the latest Conda" option will attempt to update Conda before creating or activating the environment. If you are running a self-hosted agent and have [configured a Conda installation to work with the task](#), this may result in your Conda installation being updated.

NOTE

Microsoft-hosted agents won't have Conda in their `PATH` by default. You will need to run this task in order to use Conda.

After running this task, `PATH` will contain the binary directory for the activated environment, followed by the binary directories for the Conda installation itself. You can run scripts as subsequent build tasks that run Python, Conda, or the command-line utilities from other packages you install. For example, you can run tests with `pytest` or upload a package to Anaconda Cloud with the [Anaconda client](#).

TIP

After running this task, the environment will be "activated," and packages you install by calling `conda install` will get installed to this environment.

Demands

None

Prerequisites

- A Microsoft-hosted agent, or a self-hosted agent with Anaconda or Miniconda installed.
- If using a self-hosted agent, you must either add the `conda` executable to `PATH` or set the `CONDA` environment variable to the root of the Conda installation.

YAML snippet

```

# Conda Environment
# Create and activate a Conda environment.
- task: CondaEnvironment@1
  inputs:
    #createCustomEnvironment: # Optional
    #environmentName: # Required when createCustomEnvironment == True
    #packageSpecs: 'python=3' # Optional
    #updateConda: # Optional
    #installOptions: # Optional
    #createOptions: # Optional
    #cleanEnvironment: # Optional

```

Arguments

ARGUMENT	DESCRIPTION
Create custom environment	Setting this to <code>true</code> creates or reactivates a Conda environment instead of using the <code>base</code> environment. This is recommended for self-hosted agents.
Environment name	Name of the Conda environment to create and activate.
Package specs	Space-delimited list of packages to install when creating the environment.
Update to the latest Conda	Update Conda to the latest version. This applies to the Conda installation found in <code>PATH</code> or at the path specified by the <code>CONDA</code> environment variable.

Advanced

ARGUMENT	DESCRIPTION
Install options	Space-delimited list of additional arguments to pass to the <code>conda install</code> command.
Environment creation options	Space-delimited list of other options to pass to the <code>conda create</code> command.
Clean the environment	Delete the environment and recreate it if it already exists. If not selected, the task will reactivate an existing environment.

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

How can I configure a self-hosted agent to use this task?

You can use this task either with a full Anaconda installation or a Miniconda installation. If using a self-hosted agent, you must add the `conda` executable to `PATH`. Alternatively, you can set the `CONDA` environment variable to the root of the Conda installation -- that is, the directory you specify as the "prefix" when installing Conda.

npm task

11/6/2018 • 3 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

Use this task in a build or release pipeline to install and publish npm packages.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

Install npm packages

Demands

npm

YAML snippet

```
# npm
# Install and publish npm packages, or run an npm command. Supports npmjs.com and authenticated registries
like Package Management.
- task: Npm@1
  inputs:
    #command: 'install' # Options: install, publish, custom
    #workingDir: # Optional
    #verbose: # Optional
    #customCommand: # Required when command == Custom
    #customRegistry: 'useNpmrc' # Optional. Options: useNpmrc, useFeed
    #customFeed: # Required when customRegistry == UseFeed
    #customEndpoint: # Optional
    #publishRegistry: 'useExternalRegistry' # Optional. Options: useExternalRegistry, useFeed
    #publishFeed: # Required when publishRegistry == UseFeed
    #publishEndpoint: # Required when publishRegistry == UseExternalRegistry
```

Arguments

ARGUMENT	DESCRIPTION
Command	npm command to run. Select <code>install</code> here.
Working folder with package.json	Path to the folder containing the target package.json and .npmrc files. Select the folder, not the file e.g. "/packages/mypackage".

CUSTOM REGISTRIES AND AUTHENTICATION

ARGUMENT	DESCRIPTION
Registries to use	<p>Leave this section blank to use packages from <code>npmjs</code> directly. Otherwise, select one of these options:</p> <p>Registries in my .npmrc:</p> <ul style="list-style-type: none"> Select this option to use feeds specified in a .npmrc file you've checked into source control. Credentials for registries outside this organization/collection can be used to inject credentials you've provided as an npm service connection into your <code>.npmrc</code> as the build runs. <p>Use packages from this Azure Artifacts/TFS registry:</p> <ul style="list-style-type: none"> Select this option to use one Package Management feed in the same organization/collection as the build.
ADVANCED	
Verbose logging	Enables verbose logging.
CONTROL OPTIONS	

Publish npm packages

Demands

[npm](#)

Arguments

ARGUMENT	DESCRIPTION
Command	npm command to run. Select publish here.
Working folder with package.json	Path to the folder containing the target package.json and <code>.npmrc</code> files. Select the folder, not the file e.g. <code>"/packages/mypackage"</code> .

DESTINATION REGISTRY AND AUTHENTICATION

Registry location	<ul style="list-style-type: none"> Registry I select here publishes to a Package Management registry in the same organization/collection as the build. After you select this option, select the target registry from the dropdown. External npm registry (including other organizations/collections) publishes to an external server such as npm, MyGet, or a Package Management feed in another Azure DevOps organization or TFS collection. After you select this option, create and select an npm service connection.
ADVANCED	

Verbose logging

Enables verbose logging.

CONTROL OPTIONS

Custom npm command

Demands

npm

Arguments

ARGUMENT	DESCRIPTION
Command	npm command to run. Select <code>custom</code> here.
Working folder with package.json	Path to the folder containing the target package.json and .npmrc files. Select the folder, not the file e.g. "/packages/mypackage".
Command and arguments	The custom command and arguments you wish to be executed. If your arguments contain double quotes ("), escape them with a slash (\), and surround the escaped string with double quotes ("). Example: to run <code>npm run myTask -- --users='{"foo":"bar"}'</code> , provide this input: <code>run myTask -- --users="{"\\"foo\\"":\\"bar\\""}"</code> .

CUSTOM REGISTRIES AND AUTHENTICATION

Registries to use	<p>Leave this section blank to use packages from npmjs directly. Otherwise, select one of these options:</p> <p>Registries in my .npmrc:</p> <ul style="list-style-type: none">Select this option to use feeds specified in a <code>.npmrc</code> file you've checked into source control.Credentials for registries outside this organization/collection can be used to inject credentials you've provided as an npm service connection into your <code>.npmrc</code> as the build runs. <p>Use packages from this Azure Artifacts/TFS registry:</p> <ul style="list-style-type: none">Select this option to use one Package Management feed in the same organization/collection as the build.
-------------------	--

CONTROL OPTIONS

Examples

Build: gulp

[Build your Node.js app with gulp](#)

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Where can I learn npm commands and arguments?

[npm docs](#)

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Package: npm Authenticate task (for task runners)

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to provide npm credentials to an .npmrc file in your repository for the scope of the build. This enables npm task runners like gulp and Grunt to authenticate with private registries.

WARNING

Don't use this task if you're also using the npm task.

YAML snippet

```
# npm Authenticate (for task runners)
# Don't use this task if you're also using the npm task. Provides npm credentials to an .npmrc file in your
repository for the scope of the build. This enables npm task runners like gulp and Grunt to authenticate with
private registries.
- task: npmAuthenticate@0
  inputs:
    #workingFile: # Optional
    #customEndpoint: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
.npmrc file to authenticate	(Optional) Path to the .npmrc file that specifies the registries you want to work with. Select the file, not the folder e.g. "/packages/mypackage.npmrc".
Credentials for registries outside this account/collection	(Optional) Credentials to use for external registries located in the project's .npmrc. For registries in this account/collection, leave this blank; the build's credentials are used automatically.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

NuGet task

11/6/2018 • 11 minutes to read [Edit Online](#)

Version 2.* | [Other versions](#)

Azure Pipelines | TFS 2018

Use this task in a build or release pipeline to install and update NuGet package dependencies, or package and publish NuGet packages.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

If your code depends on NuGet packages, make sure to add this step before your [Visual Studio Build step](#). Also make sure to clear the deprecated **Restore NuGet Packages** checkbox in that step.

TIP

Looking for help to get started? See the how-to's for [restoring](#) and [publishing](#) packages.

TIP

This version of the NuGet task uses NuGet 4.1.0 by default. To select a different version of NuGet, use the [Tool Installer](#).

NOTE

Using or creating .NET Core or .NET Standard packages? Use the [.NET Core](#) task, which has full support for all package scenarios currently supported by dotnet, including restore, pack, and nuget push.

YAML snippet

```

# NuGet
# Restore, pack, or push NuGet packages, or run a NuGet command. Supports NuGet.org and authenticated feeds like
# Package Management and MyGet. Uses NuGet.exe and works with .NET Framework apps. For .NET Core and .NET Standard
# apps, use the .NET Core task.
- task: NuGetCommand@2
  inputs:
    #command: 'restore' # Options: restore, pack, push, custom
    #restoreSolution: '**/*.sln' # Required when command == Restore
    #feedsToUse: 'select' # Options: select, config
    #vstsFeed: # Required when feedsToUse == Select
    #includeNuGetOrg: true # Required when feedsToUse == Select
    #nugetConfigPath: # Required when feedsToUse == Config
    #externalFeedCredentials: # Optional
    #noCache: false
    #disableParallelProcessing: false
    restoreDirectory:
      #verbosityRestore: 'Detailed' # Options: quiet, normal, detailed
      #packagesToPush:
        '$(Build.ArtifactStagingDirectory)/**/*.nupkg;!$(Build.ArtifactStagingDirectory)/**/*.symbols.nupkg' # Required when
        command == Push
      #nuGetFeedType: 'internal' # Required when command == Push# Options: internal, external
      #publishVstsFeed: # Required when command == Push && NuGetFeedType == Internal
      #allowPackageConflicts: # Optional
      #publishFeedCredentials: # Required when command == Push && NuGetFeedType == External
      #verbosityPush: 'Detailed' # Options: quiet, normal, detailed
      #packagesToPack: '**/*.csproj' # Required when command == Pack
      #configuration: '$(BuildConfiguration)' # Optional
      #packDestination: '$(Build.ArtifactStagingDirectory)' # Optional
      #versioningScheme: 'off' # Options: off, byPrereleaseNumber, byEnvVar, byBuildNumber
      #includeReferencedProjects: false # Optional
      #versionEnvVar: # Required when versioningScheme == ByEnvVar
      #majorVersion: '1' # Required when versioningScheme == ByPrereleaseNumber
      #minorVersion: '0' # Required when versioningScheme == ByPrereleaseNumber
      #patchVersion: '0' # Required when versioningScheme == ByPrereleaseNumber
      #packTimezone: 'utc' # Required when versioningScheme == ByPrereleaseNumber# Options: utc, local
      #includeSymbols: false # Optional
      #toolPackage: # Optional
      #buildProperties: # Optional
      #basePath: # Optional
      #verbosityPack: 'Detailed' # Options: quiet, normal, detailed
      #arguments: # Required when command == Custom

```

Versioning schemes

For **byPrereleaseNumber**, the version will be set to whatever you choose for major, minor, and patch, plus the date and time in the format `yyyymmdd-hhmmss`.

For **byEnvVar**, the version will be set as whatever environment variable, e.g. `$(MyVersion)`, you provide. Make sure the environment variable is set to a proper SemVer e.g. `1.2.3` or `1.2.3-beta1`.

For **byBuildNumber**, the version will be set to the build number, ensure that your build number is a proper SemVer e.g. `1.0.$(Rev:r)`.

Restore NuGet packages

Demand

None

Arguments

ARGUMENT	DESCRIPTION
Path to solution, packages.config, or project.json	Copy the Solution argument in your Visual Studio Build step and paste it here, or create a link using the Link button in the information panel.

ARGUMENT	DESCRIPTION
FEEDS AND AUTHENTICATION	
Feeds to use	<p>Feed(s) I select here:</p> <ul style="list-style-type: none"> Select this option to use NuGet.org and/or one Azure Artifacts/Package Management feed in the same organization/collection as the build. <p>Feeds in my NuGet.config:</p> <ul style="list-style-type: none"> Select this option to use feeds specified in a NuGet.config file you've checked into source control. Credentials for feeds outside this organization/collection can be used to inject credentials you've provided as a NuGet service connection into your NuGet.config as the build runs. Azure Artifacts users: see the walkthrough for help using packages from feeds in multiple Azure DevOps organizations.
ADVANCED	
Disable local cache	Prevents NuGet from using packages from local machine caches.
Destination directory	Specifies the folder in which packages are installed. If no folder is specified, packages are restored into a packages/ folder alongside the selected solution, packages.config, or project.json.
Verbosity	Specifies the amount of detail displayed in the output.
CONTROL OPTIONS	

Examples

Install NuGet dependencies

You're building a Visual Studio solution that depends on a NuGet feed.

```

`-- ConsoleApplication1
  |-- ConsoleApplication1.sln
  |-- NuGet.config
  '-- ConsoleApplication1
    |-- ConsoleApplication1.csproj
  
```

Build steps



Package: NuGet

Install your NuGet package dependencies.

- Path to solution, packages.config, or project.json:
`***.sln`
- Feeds to use: Feeds in my NuGet.config
- Path to NuGet.config:
`ConsoleApplication1/NuGet.config`



Build: Visual Studio Build

Build your solution.

- Solution: `***.sln`
- Restore NuGet Packages: **(Important)** Make sure this option is cleared.

Pack NuGet packages

Demands

None

Arguments

ARGUMENT	DESCRIPTION
Path to csproj or nuspec file(s) to pack	<p>Specify .csproj files (for example, <code>***.csproj</code>) for simple projects. In this case:</p> <ul style="list-style-type: none">• The packager compiles the .csproj files for packaging.• You must specify Configuration to Package (see below).• You do not have to check in a .nuspec file. If you do check one in, the packager honors its settings and replaces tokens such as <code>\$id\$</code> and <code>\$description\$</code>. <p>Specify .nuspec files (for example, <code>***.nuspec</code>) for more complex projects, such as multi-platform scenarios in which you need to compile and package in separate steps. In this case:</p> <ul style="list-style-type: none">• The packager does not compile the .csproj files for packaging.• Each project is packaged only if it has a .nuspec file checked in.• The packager does not replace tokens in the .nuspec file (except the <code><version/></code> element, see Use build number to version package, below). You must supply values for elements such as <code><id/></code> and <code><description/></code>. The most common way to do this is to hardcode the values in the .nuspec file. <p>To package a single file, click the ... button and select the file. To package multiple files, use file matching patterns. Note that these patterns were updated in version 2 of the NuGet task; if you have a pattern that contains <code>-:</code>, use <code>!</code> instead.</p>
Configuration to package	If you are packaging a .csproj file, you must specify a configuration that you are building and that you want to package. For example: <code>Release</code> or <code>\$(BuildConfiguration)</code>
Package folder	(Optional) Specify the folder where you want to put the packages. You can use a variable such as <code>\$(Build.ArtifactStagingDirectory)</code> . If you leave it empty, the package will be created in the root of your source tree.
PACK OPTIONS	
Automatic package versioning	This blog post provides an overview of the automatic package versioning available here.
ADVANCED	
Additional build properties	Semicolon delimited list of properties used to build the package. For example, you could replace <code><description>\$description\$</description></code> in the .nuspec file this way: <code>Description="This is a great package"</code> . Using this argument is equivalent to supplying properties from nuget pack with the <code>-Properties</code> option.
Verbosity	Specifies the amount of detail displayed in the output.
CONTROL OPTIONS	

Push NuGet packages

Demandes

None

Arguments

ARGUMENT	DESCRIPTION
Path to NuGet package(s) to publish	<p>Specify the packages you want to publish.</p> <ul style="list-style-type: none">• Default value: <code>\$(Build.ArtifactStagingDirectory)/*.nupkg</code>• To publish a single package, click the ... button and select the file.• Use file matching patterns to publish multiple packages. Note that these patterns were updated in version 2 of the NuGet task; if you have a pattern that contains <code>-:</code>, use <code>!</code> instead.• Use variables to specify directories. For example, if you specified <code>\$(Build.ArtifactStagingDirectory)\</code> as the package folder in the pack step above, you could specify <code>\$(Build.ArtifactStagingDirectory)***.nupkg</code> here.
Target feed location	<ul style="list-style-type: none">• This organization/collection publishes to an Azure Artifacts/Package Management feed in the same organization/collection as the build. After you select this option, select the target feed from the dropdown.<ul style="list-style-type: none">◦ "Allow duplicates to be skipped" allows you to continually publish a set of packages and only change the version number of the subset of packages that changed. It allows the task to report success even if some of your packages are rejected with 409 Conflict errors.• External NuGet server (including other organizations/collections) publishes to an external server such as NuGet, MyGet, or an Azure Artifacts/Package Management feed in another Azure DevOps organization or TFS collection. After you select this option, you create and select a NuGet service connection.
ADVANCED	
Verbosity	Specifies the amount of detail displayed in the output.
CONTROL OPTIONS	

Publishing symbols

When you push packages to an Azure Artifacts/Package Management feed, you can also publish symbols using the [Index Sources & Publish Symbols task](#).

Publishing packages to TFS with IIS Basic authentication enabled

This task is unable to publish NuGet packages to a TFS Package Management feed that is running on a TFS server with IIS Basic authentication enabled. [See here](#) for more details.

Custom NuGet command

YAML snippet

```

# NuGet Command
# Deprecated: use the "NuGet" task instead. It works with the new Tool Installer framework so you can easily use new
versions of NuGet without waiting for a task update, provides better support for authenticated feeds outside this
organization/collection, and uses NuGet 4 by default.
- task: NuGet@0
  inputs:
    command:
    arguments:

```

Arguments

ARGUMENT	DESCRIPTION
Command and arguments	NuGet command and arguments you want to pass as your custom command.

End-to-end example

You want to package and publish some projects in a C# class library to your Azure Artifacts feed.

You want to package and publish some projects in a C# class library to your TFS Package Management feed.

```

`-- Message
  |-- Message.sln
  '-- ShortGreeting
    |-- ShortGreeting.csproj
    |-- Class1.cs
    '-- Properties
      |-- AssemblyInfo.cs
  '-- LongGreeting
    |-- LongGreeting.csproj
    |-- Class1.cs
    '-- Properties
      |-- AssemblyInfo.cs

```

Prepare

AssemblyInfo.cs

Make sure your AssemblyInfo.cs files contain the information you want shown in your packages. For example,

`AssemblyCompanyAttribute` will be shown as the author, and `AssemblyDescriptionAttribute` will be shown as the description.

Variables tab

NAME	VALUE
<code>\$(BuildConfiguration)</code>	<code>release</code>
<code>\$(BuildPlatform)</code>	<code>any cpu</code>

Options

SETTING	VALUE
Build number format	<code>\$(BuildDefinitionName)_\$(Year:yyyy).\$(Month).\$(DayOfMonth)\$(Rev:.rrrr)</code>

Publish to Azure Artifacts

Publish to a TFS feed

1. Make sure you've prepared the build as described [above](#).
2. If you haven't already, [create a feed](#).

3. Add the following build steps:

 Package: NuGet	Install your NuGet package dependencies. <ul style="list-style-type: none">Path to solution, packages.config, or project.json: **/*.slnFeeds to use: Feeds I select hereUse packages from NuGet.org: Checked
 Build: Visual Studio Build	Build your solution. <ul style="list-style-type: none">Solution: **/*.slnPlatform: \$(BuildPlatform)Configuration: \$(BuildConfiguration)
 Package: NuGet	Package your projects. <ul style="list-style-type: none">Command: packPath to csproj or nuspec file(s) to pack: **/*.csprojConfiguration to Package: ReleasePackage Folder: \$(Build.ArtifactStagingDirectory)Pack options > Automatic package versioning: Use the build number
 Package: NuGet	Publish your packages. <ul style="list-style-type: none">Command: pushPath to NuGet package(s) to publish: \$(Build.ArtifactStagingDirectory)Target feed location: This organization/collectionTarget feed: Select your feed

Option 2: publish to NuGet.org

1. Make sure you've prepared the build as described [above](#).
2. If you haven't already, [register with NuGet.org](#).
3. Use the steps in the previous section, but substitute the final step for the step shown here.

 Package: NuGet	Publish your packages to NuGet.org. <ul style="list-style-type: none">Command: pushPath to NuGet package(s) to publish: \$(Build.ArtifactStagingDirectory)Target feed location: External NuGet serverNuGet server: Create a new NuGet service connection with your NuGet.org ApiKey and select it here
--	---

Task versions

Task: NuGet (formerly NuGet Restore at 1.* , NuGet Installer at 0.*)

TASK VERSION	AZURE PIPELINES	TFS
2.*	Available	Appeared in 2018
1.*	Deprecated but available	Appeared in 2017 Update 2, deprecated in 2018

TASK VERSION	AZURE PIPELINES	TFS
0.*	Deprecated but available	Appeared in 2017, deprecated in 2017 Update 2

Task: NuGet Packager

TASK VERSION	AZURE PIPELINES	TFS
0.*	Deprecated but available	Available in TFS < 2018, deprecated in TFS >= 2018

Task: NuGet Publisher

TASK VERSION	AZURE PIPELINES	TFS
0.*	Deprecated but available	Available in TFS < 2018, deprecated in TFS >= 2018

Task: NuGet Command

TASK VERSION	AZURE PIPELINES	TFS
0.*	Deprecated but available	Available in TFS < 2017 Update 2, deprecated in TFS >= 2018

Open source

These tasks are open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Why should I check in a NuGet.Config?

Checking a NuGet.Config into source control ensures that a key piece of information needed to build your project—the location of its packages—is available to every developer that checks out your code.

However, for situations where a team of developers works on a large range of projects, it's also possible to add an Azure Artifacts feed to the global NuGet.Config on each developer's machine. In these situations, using the "Feeds I select here" option in the NuGet task replicates this configuration.

Where can I learn about Azure Artifacts?

[Package Management in Azure Artifacts and TFS](#)

Where can I learn more about NuGet?

[NuGet Docs](#) Overview

[NuGet Create](#) Packaging and publishing

[NuGet Consume](#) Setting up a solution to get dependencies

What other kinds of apps can I build?

[Build your app](#)

What other kinds of build tasks are available?

[Specify your build tasks](#)

How do we protect our codebase from build breaks?

- Git: [Improve code quality with branch policies](#) with an option to require that code builds before it can be merged to

a branch. For GitHub repositories, similar policies are available in GitHub's repository settings under *Branches*.

- TFVC: [Use gated check-in](#).

How do I modify other parts of my build pipeline?

- [Specify your build tasks](#) to run tests, scripts, and a wide range of other processes.
- [Specify build options](#) such as specifying how completed builds are named, building multiple configurations, creating work items on failure.
- [Specify the repository](#) to pick the source of the build and modify options such as how the agent workspace is cleaned.
- [Set build triggers](#) to modify how your CI builds run and to specify scheduled builds.
- [Specify build retention policies](#) to automatically delete old builds.

I selected parallel multi-configuration, but only one build is running at a time.

If you're using Azure Pipelines, you might need more parallel jobs. See [Parallel jobs in Azure Pipelines](#).

How do I see what has changed in my build pipeline?

[View the change history of your build pipeline](#)

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Package: PyPI Publisher task (deprecated)

11/9/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to create and upload an sdist or wheel to a PyPI-compatible index using Twine.

This task builds an sdist package by running `python setup.py sdist` using the Python instance in `PATH`. It can optionally build a universal wheel in addition to the sdist. Then, it will upload the package to a PyPI index using `twine`. The task will install the `wheel` and `twine` packages with `python -m pip install --user`.

Deprecated

WARNING

The PyPI Publisher task has been deprecated. You can now [publish PyPI packages using twine authentication and custom scripts](#).

Demands

None

Prerequisites

A generic service connection for a PyPI index.

TIP

To configure a new generic service connection, go to Settings -> Services -> New service connection -> Generic.

- **Connection Name:** A friendly connection name of your choice
- **Server URL:** PyPI package server (for example: <https://upload.pypi.org/legacy/>)
- **User name:** username for your PyPI account
- **Password/Token Key:** password for your PyPI account

YAML snippet

```
# PyPI Publisher
# Create and upload an sdist or wheel to a PyPI-compatible index using Twine.
- task: PyPIPUBLISHER@0
  inputs:
    pypiConnection:
    packageDirectory:
    #alsoPublishWheel: false # Optional
```

Arguments

ARGUMENT	DESCRIPTION
PyPI connection	A generic service connection for connecting to the package index.
Python package directory	The directory of the Python package to be created and published, where setup.py is present.
Also publish a wheel	Select whether to create and publish a universal wheel package (platform independent) in addition to an sdist package.

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

Package: Python Pip Authenticate

11/19/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Provides authentication for the `pip` client that can be used to install Python distributions.

NOTE

The Python Pip Authenticate task in Azure Pipelines is currently in public preview.

YAML snippet

```
# Python Pip Authenticate
# Authentication task for the pip client used for installing Python distributions.
- task: PipAuthenticate@0
  inputs:
    artifactFeeds: # 'feed_name1, feed_name2'
    #externalFeeds: # Optional. 'feed_name1, feed_name2'
```

Arguments

ARGUMENT	DESCRIPTION
artifactFeeds	List of Azure Artifacts feeds to authenticate with pip.
externalFeeds	List of service endpoints from external organizations to authenticate with pip.

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Package: Python Twine Upload Authenticate

11/19/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Provides `twine` credentials to a `PYPIRC_PATH` environment variable for the scope of the build. This enables you to publish Python packages to feeds with `twine` from your build.

NOTE

The Python Twine Upload Authenticate task in Azure Pipelines is currently in public preview.

YAML snippet

```
# Python Twine Upload Authenticate
# Authentication for uploading Python distributions using twine. Please add '-r FeedName/EndpointName --config-file $(PYPIRC_PATH)' to your twine upload command. For feeds present in this organization, use the feed name as the repository (-r). Otherwise, use the endpoint name defined in the service connection.
- task: TwineAuthenticate@0
  inputs:
    artifactFeeds: # 'feed_name1, feed_name2'
    #externalFeeds: # Optional. 'feed_name1, feed_name2'
```

Arguments

ARGUMENT	DESCRIPTION
artifactFeeds	List of Azure Artifacts feeds to authenticate with <code>twine</code> .
externalFeeds	List of service endpoints from external organizations to authenticate with <code>twine</code> . The credentials stored in the endpoint must have package upload permissions.

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

NuGet Installer task version 0.*

10/29/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines (deprecated) | TFS 2017 (deprecated in 2017 Update 2)

Use this task in a build or release pipeline to install and update NuGet package dependencies.

Demands

If your code depends on NuGet packages, make sure to add this task before your [Visual Studio Build task](#). Also make sure to clear the deprecated **Restore NuGetPackages** checkbox in that task.

Arguments

ARGUMENT	DESCRIPTION
Path to Solution	Copy the value from the Solution argument in your Visual Studio Build task and paste it here.
Path to NuGet.config	If you are using a package source other than NuGet.org, you must check in a NuGet.config file and specify the path to it here.
Disable local cache	Equivalent to nuget restore with the <code>-NoCache</code> option.
NuGet Arguments	Additional arguments passed to nuget restore .
ADVANCED	
Path to NuGet.exe	(Optional) Path to your own instance of NuGet.exe. If you specify this argument, you must have your own strategy to handle authentication .
CONTROL OPTIONS	

Examples

Install NuGet dependencies

You're building a Visual Studio solution that depends on a NuGet feed.

```
'-- ConsoleApplication1
  |-- ConsoleApplication1.sln
  |-- NuGet.config
  '-- ConsoleApplication1
    |-- ConsoleApplication1.csproj
```

Build tasks



Package: NuGet Installer

Install your NuGet package dependencies.

- Path to Solution: `***.sln`

- Path to NuGet.config:

`ConsoleApplication1/NuGet.config`



Build: Visual Studio Build

Build your solution.

- Solution: `***.sln`

- Restore NuGet Packages: **(Important)** Make sure this option is cleared.

NuGet task

11/6/2018 • 11 minutes to read [Edit Online](#)

Version 2.* | [Other versions](#)

Azure Pipelines | TFS 2018

Use this task in a build or release pipeline to install and update NuGet package dependencies, or package and publish NuGet packages.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

If your code depends on NuGet packages, make sure to add this step before your [Visual Studio Build step](#). Also make sure to clear the deprecated **Restore NuGet Packages** checkbox in that step.

TIP

Looking for help to get started? See the how-to's for [restoring](#) and [publishing](#) packages.

TIP

This version of the NuGet task uses NuGet 4.1.0 by default. To select a different version of NuGet, use the [Tool Installer](#).

NOTE

Using or creating .NET Core or .NET Standard packages? Use the [.NET Core](#) task, which has full support for all package scenarios currently supported by dotnet, including restore, pack, and nuget push.

YAML snippet

```

# NuGet
# Restore, pack, or push NuGet packages, or run a NuGet command. Supports NuGet.org and authenticated feeds like
# Package Management and MyGet. Uses NuGet.exe and works with .NET Framework apps. For .NET Core and .NET Standard
# apps, use the .NET Core task.
- task: NuGetCommand@2
  inputs:
    #command: 'restore' # Options: restore, pack, push, custom
    #restoreSolution: '**/*.sln' # Required when command == Restore
    #feedsToUse: 'select' # Options: select, config
    #vstsFeed: # Required when feedsToUse == Select
    #includeNuGetOrg: true # Required when feedsToUse == Select
    #nugetConfigPath: # Required when feedsToUse == Config
    #externalFeedCredentials: # Optional
    #noCache: false
    #disableParallelProcessing: false
    restoreDirectory:
      #verbosityRestore: 'Detailed' # Options: quiet, normal, detailed
      #packagesToPush:
        '$(Build.ArtifactStagingDirectory)/**/*.nupkg;!$(Build.ArtifactStagingDirectory)/**/*.symbols.nupkg' # Required when
        command == Push
      #nuGetFeedType: 'internal' # Required when command == Push# Options: internal, external
      #publishVstsFeed: # Required when command == Push && NuGetFeedType == Internal
      #allowPackageConflicts: # Optional
      #publishFeedCredentials: # Required when command == Push && NuGetFeedType == External
      #verbosityPush: 'Detailed' # Options: quiet, normal, detailed
      #packagesToPack: '**/*.csproj' # Required when command == Pack
      #configuration: '$(BuildConfiguration)' # Optional
      #packDestination: '$(Build.ArtifactStagingDirectory)' # Optional
      #versioningScheme: 'off' # Options: off, byPrereleaseNumber, byEnvVar, byBuildNumber
      #includeReferencedProjects: false # Optional
      #versionEnvVar: # Required when versioningScheme == ByEnvVar
      #majorVersion: '1' # Required when versioningScheme == ByPrereleaseNumber
      #minorVersion: '0' # Required when versioningScheme == ByPrereleaseNumber
      #patchVersion: '0' # Required when versioningScheme == ByPrereleaseNumber
      #packTimezone: 'utc' # Required when versioningScheme == ByPrereleaseNumber# Options: utc, local
      #includeSymbols: false # Optional
      #toolPackage: # Optional
      #buildProperties: # Optional
      #basePath: # Optional
      #verbosityPack: 'Detailed' # Options: quiet, normal, detailed
      #arguments: # Required when command == Custom

```

Versioning schemes

For **byPrereleaseNumber**, the version will be set to whatever you choose for major, minor, and patch, plus the date and time in the format `yyyymmdd-hhmmss`.

For **byEnvVar**, the version will be set as whatever environment variable, e.g. `$(MyVersion)`, you provide. Make sure the environment variable is set to a proper SemVer e.g. `1.2.3` or `1.2.3-beta1`.

For **byBuildNumber**, the version will be set to the build number, ensure that your build number is a proper SemVer e.g. `1.0.$(Rev:r)`.

Restore NuGet packages

Demand

None

Arguments

ARGUMENT	DESCRIPTION
Path to solution, packages.config, or project.json	Copy the Solution argument in your Visual Studio Build step and paste it here, or create a link using the Link button in the information panel.

ARGUMENT	DESCRIPTION
FEEDS AND AUTHENTICATION	
Feeds to use	<p>Feed(s) I select here:</p> <ul style="list-style-type: none"> Select this option to use NuGet.org and/or one Azure Artifacts/Package Management feed in the same organization/collection as the build. <p>Feeds in my NuGet.config:</p> <ul style="list-style-type: none"> Select this option to use feeds specified in a NuGet.config file you've checked into source control. Credentials for feeds outside this organization/collection can be used to inject credentials you've provided as a NuGet service connection into your NuGet.config as the build runs. Azure Artifacts users: see the walkthrough for help using packages from feeds in multiple Azure DevOps organizations.
ADVANCED	
Disable local cache	Prevents NuGet from using packages from local machine caches.
Destination directory	Specifies the folder in which packages are installed. If no folder is specified, packages are restored into a packages/ folder alongside the selected solution, packages.config, or project.json.
Verbosity	Specifies the amount of detail displayed in the output.
CONTROL OPTIONS	

Examples

Install NuGet dependencies

You're building a Visual Studio solution that depends on a NuGet feed.

```

`-- ConsoleApplication1
  |-- ConsoleApplication1.sln
  |-- NuGet.config
  '-- ConsoleApplication1
    |-- ConsoleApplication1.csproj
  
```

Build steps



Package: NuGet

Install your NuGet package dependencies.

- Path to solution, packages.config, or project.json:
`***.sln`
- Feeds to use: Feeds in my NuGet.config
- Path to NuGet.config:
`ConsoleApplication1/NuGet.config`



Build: Visual Studio Build

Build your solution.

- Solution: `***.sln`
- Restore NuGet Packages: **(Important)** Make sure this option is cleared.

Pack NuGet packages

Demands

None

Arguments

ARGUMENT	DESCRIPTION
Path to csproj or nuspec file(s) to pack	<p>Specify .csproj files (for example, <code>***.csproj</code>) for simple projects. In this case:</p> <ul style="list-style-type: none">• The packager compiles the .csproj files for packaging.• You must specify Configuration to Package (see below).• You do not have to check in a .nuspec file. If you do check one in, the packager honors its settings and replaces tokens such as <code>\$id\$</code> and <code>\$description\$</code>. <p>Specify .nuspec files (for example, <code>***.nuspec</code>) for more complex projects, such as multi-platform scenarios in which you need to compile and package in separate steps. In this case:</p> <ul style="list-style-type: none">• The packager does not compile the .csproj files for packaging.• Each project is packaged only if it has a .nuspec file checked in.• The packager does not replace tokens in the .nuspec file (except the <code><version/></code> element, see Use build number to version package, below). You must supply values for elements such as <code><id/></code> and <code><description/></code>. The most common way to do this is to hardcode the values in the .nuspec file. <p>To package a single file, click the ... button and select the file. To package multiple files, use file matching patterns. Note that these patterns were updated in version 2 of the NuGet task; if you have a pattern that contains <code>-:</code>, use <code>!</code> instead.</p>
Configuration to package	If you are packaging a .csproj file, you must specify a configuration that you are building and that you want to package. For example: <code>Release</code> or <code>\$(BuildConfiguration)</code>
Package folder	(Optional) Specify the folder where you want to put the packages. You can use a variable such as <code>\$(Build.ArtifactStagingDirectory)</code> . If you leave it empty, the package will be created in the root of your source tree.
PACK OPTIONS	
Automatic package versioning	This blog post provides an overview of the automatic package versioning available here.
ADVANCED	
Additional build properties	Semicolon delimited list of properties used to build the package. For example, you could replace <code><description>\$description\$</description></code> in the .nuspec file this way: <code>Description="This is a great package"</code> . Using this argument is equivalent to supplying properties from nuget pack with the <code>-Properties</code> option.
Verbosity	Specifies the amount of detail displayed in the output.
CONTROL OPTIONS	

Push NuGet packages

Demandes

None

Arguments

ARGUMENT	DESCRIPTION
Path to NuGet package(s) to publish	<p>Specify the packages you want to publish.</p> <ul style="list-style-type: none">• Default value: <code>\$(Build.ArtifactStagingDirectory)/*.nupkg</code>• To publish a single package, click the ... button and select the file.• Use file matching patterns to publish multiple packages. Note that these patterns were updated in version 2 of the NuGet task; if you have a pattern that contains <code>-:</code>, use <code>!</code> instead.• Use variables to specify directories. For example, if you specified <code>\$(Build.ArtifactStagingDirectory)\</code> as the package folder in the pack step above, you could specify <code>\$(Build.ArtifactStagingDirectory)***.nupkg</code> here.
Target feed location	<ul style="list-style-type: none">• This organization/collection publishes to an Azure Artifacts/Package Management feed in the same organization/collection as the build. After you select this option, select the target feed from the dropdown.<ul style="list-style-type: none">◦ "Allow duplicates to be skipped" allows you to continually publish a set of packages and only change the version number of the subset of packages that changed. It allows the task to report success even if some of your packages are rejected with 409 Conflict errors.• External NuGet server (including other organizations/collections) publishes to an external server such as NuGet, MyGet, or an Azure Artifacts/Package Management feed in another Azure DevOps organization or TFS collection. After you select this option, you create and select a NuGet service connection.
ADVANCED	
Verbosity	Specifies the amount of detail displayed in the output.
CONTROL OPTIONS	

Publishing symbols

When you push packages to an Azure Artifacts/Package Management feed, you can also publish symbols using the [Index Sources & Publish Symbols task](#).

Publishing packages to TFS with IIS Basic authentication enabled

This task is unable to publish NuGet packages to a TFS Package Management feed that is running on a TFS server with IIS Basic authentication enabled. [See here](#) for more details.

Custom NuGet command

YAML snippet

```

# NuGet Command
# Deprecated: use the "NuGet" task instead. It works with the new Tool Installer framework so you can easily use new
versions of NuGet without waiting for a task update, provides better support for authenticated feeds outside this
organization/collection, and uses NuGet 4 by default.
- task: NuGet@0
  inputs:
    command:
    arguments:

```

Arguments

ARGUMENT	DESCRIPTION
Command and arguments	NuGet command and arguments you want to pass as your custom command.

End-to-end example

You want to package and publish some projects in a C# class library to your Azure Artifacts feed.

You want to package and publish some projects in a C# class library to your TFS Package Management feed.

```

`-- Message
  |-- Message.sln
  '-- ShortGreeting
    |-- ShortGreeting.csproj
    |-- Class1.cs
    '-- Properties
      |-- AssemblyInfo.cs
  '-- LongGreeting
    |-- LongGreeting.csproj
    |-- Class1.cs
    '-- Properties
      |-- AssemblyInfo.cs

```

Prepare

AssemblyInfo.cs

Make sure your AssemblyInfo.cs files contain the information you want shown in your packages. For example,

`AssemblyCompanyAttribute` will be shown as the author, and `AssemblyDescriptionAttribute` will be shown as the description.

Variables tab

NAME	VALUE
<code>\$(BuildConfiguration)</code>	<code>release</code>
<code>\$(BuildPlatform)</code>	<code>any cpu</code>

Options

SETTING	VALUE
Build number format	<code>\$(BuildDefinitionName)_\$(Year:yyyy).\$(Month).\$(DayOfMonth)\$(Rev:.rrrr)</code>

Publish to Azure Artifacts

Publish to a TFS feed

1. Make sure you've prepared the build as described [above](#).
2. If you haven't already, [create a feed](#).

3. Add the following build steps:

 Package: NuGet	Install your NuGet package dependencies. <ul style="list-style-type: none">Path to solution, packages.config, or project.json: <code>**/*.sln</code>Feeds to use: Feeds I select hereUse packages from NuGet.org: Checked
 Build: Visual Studio Build	Build your solution. <ul style="list-style-type: none">Solution: <code>***.sln</code>Platform: <code>\$(BuildPlatform)</code>Configuration: <code>\$(BuildConfiguration)</code>
 Package: NuGet	Package your projects. <ul style="list-style-type: none">Command: packPath to csproj or nuspec file(s) to pack: <code>**/*.csproj</code>Configuration to Package: <code>Release</code>Package Folder: <code>\$(Build.ArtifactStagingDirectory)</code>Pack options > Automatic package versioning: Use the build number
 Package: NuGet	Publish your packages. <ul style="list-style-type: none">Command: pushPath to NuGet package(s) to publish: <code>\$(Build.ArtifactStagingDirectory)</code>Target feed location: This organization/collectionTarget feed: Select your feed

Option 2: publish to NuGet.org

1. Make sure you've prepared the build as described [above](#).
2. If you haven't already, [register with NuGet.org](#).
3. Use the steps in the previous section, but substitute the final step for the step shown here.

 Package: NuGet	Publish your packages to NuGet.org. <ul style="list-style-type: none">Command: pushPath to NuGet package(s) to publish: <code>\$(Build.ArtifactStagingDirectory)</code>Target feed location: External NuGet serverNuGet server: Create a new NuGet service connection with your NuGet.org ApiKey and select it here
--	--

Task versions

Task: NuGet (formerly NuGet Restore at 1.* , NuGet Installer at 0.*)

TASK VERSION	AZURE PIPELINES	TFS
2.*	Available	Appeared in 2018
1.*	Deprecated but available	Appeared in 2017 Update 2, deprecated in 2018

TASK VERSION	AZURE PIPELINES	TFS
0.*	Deprecated but available	Appeared in 2017, deprecated in 2017 Update 2

Task: NuGet Packager

TASK VERSION	AZURE PIPELINES	TFS
0.*	Deprecated but available	Available in TFS < 2018, deprecated in TFS >= 2018

Task: NuGet Publisher

TASK VERSION	AZURE PIPELINES	TFS
0.*	Deprecated but available	Available in TFS < 2018, deprecated in TFS >= 2018

Task: NuGet Command

TASK VERSION	AZURE PIPELINES	TFS
0.*	Deprecated but available	Available in TFS < 2017 Update 2, deprecated in TFS >= 2018

Open source

These tasks are open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Why should I check in a NuGet.Config?

Checking a NuGet.Config into source control ensures that a key piece of information needed to build your project—the location of its packages—is available to every developer that checks out your code.

However, for situations where a team of developers works on a large range of projects, it's also possible to add an Azure Artifacts feed to the global NuGet.Config on each developer's machine. In these situations, using the "Feeds I select here" option in the NuGet task replicates this configuration.

Where can I learn about Azure Artifacts?

[Package Management in Azure Artifacts and TFS](#)

Where can I learn more about NuGet?

[NuGet Docs](#) Overview

[NuGet Create](#) Packaging and publishing

[NuGet Consume](#) Setting up a solution to get dependencies

What other kinds of apps can I build?

[Build your app](#)

What other kinds of build tasks are available?

[Specify your build tasks](#)

How do we protect our codebase from build breaks?

- Git: [Improve code quality with branch policies](#) with an option to require that code builds before it can be merged to

a branch. For GitHub repositories, similar policies are available in GitHub's repository settings under *Branches*.

- TFVC: [Use gated check-in](#).

How do I modify other parts of my build pipeline?

- [Specify your build tasks](#) to run tests, scripts, and a wide range of other processes.
- [Specify build options](#) such as specifying how completed builds are named, building multiple configurations, creating work items on failure.
- [Specify the repository](#) to pick the source of the build and modify options such as how the agent workspace is cleaned.
- [Set build triggers](#) to modify how your CI builds run and to specify scheduled builds.
- [Specify build retention policies](#) to automatically delete old builds.

I selected parallel multi-configuration, but only one build is running at a time.

If you're using Azure Pipelines, you might need more parallel jobs. See [Parallel jobs in Azure Pipelines](#).

How do I see what has changed in my build pipeline?

[View the change history of your build pipeline](#)

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

NuGet Packager task version 0.*

10/29/2018 • 5 minutes to read • [Edit Online](#)

Azure Pipelines (deprecated) | TFS 2017 Update 2 and below (deprecated in TFS 2018)

Use this task in a build or release pipeline to create a NuGet package from either a .csproj or .nuspec file.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Demands

None

YAML snippet

```
# NuGet Packager
# Deprecated: use the "NuGet" task instead. It works with the new Tool Installer framework so you can easily use new
# versions of NuGet without waiting for a task update, provides better support for authenticated feeds outside this
# account/collection, and uses NuGet 4 by default.
- task: NuGetPackager@0
  inputs:
    #searchPattern: '**\*.csproj'
    #outputdir: # Optional
    #includeReferencedProjects: false # Optional
    #versionByBuild: 'false' # Options: false, byPrereleaseNumber, byEnvVar, true
    #versionEnvVar: # Required when versionByBuild == ByEnvVar
    #requestedMajorVersion: '1' # Required when versionByBuild == ByPrereleaseNumber
    #requestedMinorVersion: '0' # Required when versionByBuild == ByPrereleaseNumber
    #requestedPatchVersion: '0' # Required when versionByBuild == ByPrereleaseNumber
    #configurationToPack: '$(BuildConfiguration)' # Optional
    #buildProperties: # Optional
    #nuGetAdditionalArgs: # Optional
    #nuGetPath: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
----------	-------------

ARGUMENT	DESCRIPTION
Path/Pattern to nuspec files	<p>Specify .csproj files (for example, <code>***.csproj</code>) for simple projects. In this case:</p> <ul style="list-style-type: none"> The packager compiles the .csproj files for packaging. You must specify Configuration to Package (see below). You do not have to check in a .nuspec file. If you do check one in, the packager honors its settings and replaces tokens such as <code>\$id\$</code> and <code>\$description\$</code>. <p>Specify .nuspec files (for example, <code>***.nuspec</code>) for more complex projects, such as multi-platform scenarios in which you need to compile and package in separate steps. In this case:</p> <ul style="list-style-type: none"> The packager does not compile the .csproj files for packaging. Each project is packaged only if it has a .nuspec file checked in. The packager does not replace tokens in the .nuspec file (except the <code><version/></code> element, see Use build number to version package, below). You must supply values for elements such as <code><id/></code> and <code><description/></code>. The most common way to do this is to hardcode the values in the .nuspec file. <p>To package a single file, click the ... button and select the file. To package multiple files, use single-folder wildcards (<code>*</code>) and recursive wildcards (<code>**</code>). For example, specify <code>***.csproj</code> to package all .csproj files in all subdirectories in the repo.</p> <p>You can use multiple patterns separated by a semicolon to create more complex queries. You can negate a pattern by prefixing it with <code>-:-</code>. For example, specify <code>***.csproj;-:***Tests.csproj</code> to package all .csproj files except those ending in 'Tests' in all subdirectories in the repo.</p>
Use build number to version package	<p>Select if you want to use the build number to version your package. If you select this option, for the pipeline options, set the build number format to something like <code>\$(BuildDefinitionName)_\$(Year:yyyy).\$(Month).\$(DayOfMonth)\$(Rev:.</code></p> <p>The build number format must be <code>{some_characters}_0.0.0.0</code>. The characters and the underscore character are omitted from the output. The version number at the end must be a unique number in a format such as <code>0.0.0.0</code> that is higher than the last published number.</p> <p>The version number is passed to nuget pack with the <code>-Version</code> option.</p> <p>Versions are shown prominently on NuGet servers. For example they are listed on the Azure Artifacts feeds page and on the NuGet.org package page.</p>
Package Folder	<p>(Optional) Specify the folder where you want to put the packages. You can use a variable such as <code>\$(Build.StagingDirectory)\packages</code></p> <p>If you leave it empty, the package will be created in the same directory that contains the .csproj or .nuspec file.</p>
ADVANCED	

ARGUMENT	DESCRIPTION
Configuration to Package	If you are packaging a .csproj file, you must specify a configuration that you are building and that you want to package. For example: <code>Release</code>
Additional build properties	Semicolon delimited list of properties used to build the package. For example, you could replace <code><description>\$description\$</description></code> in the <code>.nuspec</code> file this way: <code>Description="This is a great package"</code> Using this argument is equivalent to supplying properties from nuget pack with the <code>-Properties</code> option.
NuGet Arguments	(Optional) Additional arguments passed nuget pack .
Path to NuGet.exe	(Optional) Path to your own instance of NuGet.exe. If you specify this argument, you must have your own strategy to handle authentication .

CONTROL OPTIONS

Examples

You want to package and publish some projects in a C# class library to your Azure Artifacts feed.

```

`-- Message
  |-- Message.sln
  '-- ShortGreeting
    |-- ShortGreeting.csproj
    |-- Class1.cs
    '-- Properties
      |-- AssemblyInfo.cs
  '-- LongGreeting
    |-- LongGreeting.csproj
    |-- Class1.cs
    '-- Properties
      |-- AssemblyInfo.cs

```

Prepare

AssemblyInfo.cs

Make sure your AssemblyInfo.cs files contain the information you want shown in your packages. For example, `AssemblyCompanyAttribute` will be shown as the author, and `AssemblyDescriptionAttribute` will be shown as the description.

Variables tab

NAME	VALUE
<code>\$(BuildConfiguration)</code>	<code>release</code>
<code>\$(BuildPlatform)</code>	<code>any cpu</code>

Options

SETTING	VALUE
Build number format	<code>\$(BuildDefinitionName)_\$(Year:yyyy).\$(Month).\$(DayOfMonth)\$(Rev:.rrrr)</code>

Publish to Azure Artifacts

Make sure you've prepared the build as described [above](#).

Create the feed

See [Create a feed](#).

Build tasks

 Build: Visual Studio Build	Build your solution. <ul style="list-style-type: none">• Solution: <code>***.sln</code>• Platform: <code>\$(BuildPlatform)</code>• Configuration: <code>\$(BuildConfiguration)</code>
 Package: NuGet Packager	Package your projects. <ul style="list-style-type: none">• Path/Pattern to nuspec files: <code>***.csproj</code>• Use Build number to version package: Selected• Advanced, Configuration to Package: <code>Release</code>
 Package: NuGet Publisher	Publish your packages to Azure Artifacts. <ul style="list-style-type: none">• Path/Pattern to nupkg: <code>***.nupkg</code>• Feed type: Internal NuGet Feed• Internal feed URL: See Find your NuGet package source URL.

Publish to NuGet.org

Make sure you've prepared the build as described [above](#).

Register with NuGet.org

If you haven't already, [register with NuGet.org](#).

Build tasks

 Build: Visual Studio Build	Build your solution. <ul style="list-style-type: none">• Solution: <code>***.sln</code>• Platform: <code>\$(BuildPlatform)</code>• Configuration: <code>\$(BuildConfiguration)</code>
 Package: NuGet Packager	Package your projects. <ul style="list-style-type: none">• Path/Pattern to nuspec files: <code>***.csproj</code>• Use Build number to version package: Selected• Advanced, Configuration to Package: <code>Release</code>



Package: NuGet Publisher

Publish your packages to NuGet.org.

- Path/Pattern to nupkg: `***.nupkg`
 - Feed type: External NuGet Feed
 - NuGet Server Endpoint: **Manage**
1. Click "New service connection", and then click Generic.
 2. On the Add New Generic Connection dialog box:
 - Connection Name: `NuGet`
 - Server URL: `https://nuget.org/`
 - User name: `{your-name}`
 - Password TokenName Key: Paste API Key from your [NuGet account](#).

Package: NuGet Publisher task version 0.*

10/29/2018 • 3 minutes to read • [Edit Online](#)

Azure Pipelines (deprecated) | TFS 2017 Update 2 and below (deprecated in TFS 2018)

Use this task in a build or release pipeline to publish your NuGet package to a server and update your feed.

Demands

None

YAML snippet

```
# NuGet Publisher
# Deprecated: use the "NuGet" task instead. It works with the new Tool Installer framework so you can easily use new
# versions of NuGet without waiting for a task update, provides better support for authenticated feeds outside this
# organization/collection, and uses NuGet 4 by default.
- task: NuGetPublisher@0
  inputs:
    #searchPattern: '**/*.nupkg; -:**\packages\**\*.nupkg; -:**\*.symbols.nupkg'
    #nuGetFeedType: 'external' # Options: external, internal
    #connectedServiceName: # Required when nuGetFeedType == External
    #feedName: # Required when nuGetFeedType == Internal
    #nuGetAdditionalArgs: # Optional
    #verbosity: '-' # Options: -, quiet, normal, detailed
    #nuGetVersion: '3.3.0' # Options: 3.3.0, 3.5.0.1829, 4.0.0.2283, custom
    #nuGetPath: # Optional
    #continueOnEmptyNupkgMatch: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Path/Pattern to nupkg	<p>Specify the packages you want to publish.</p> <ul style="list-style-type: none">• Default value: <code>***.nupkg; -:**\packages***.nupkg</code>• To publish a single package, click the ... button and select the file.• Use single-folder wildcards (<code>*</code>) and recursive wildcards (<code>**</code>) to publish multiple packages.• Use variables to specify directories. For example, if you specified <code>\$(Build.StagingDirectory)\packages</code> as the package folder in the NuGet Packager task, you could specify <code>\$(Build.StagingDirectory)\packages***.nupkg</code> here.
Feed type	<ul style="list-style-type: none">• External NuGetFeed publishes to an external server such as NuGet or MyGet. After you select this option, you create and select a NuGet server endpoint.• Internal NuGet Feed publishes to an internal or Azure Artifacts feed. After you select this option, you specify the internal feed URL.
ADVANCED	

ARGUMENT	DESCRIPTION
NuGet Arguments	(Optional) Additional arguments passed to <code>nuget push</code> .
Path to NuGet.exe	(Optional) Path to your own instance of NuGet.exe. If you specify this argument, you must have your own strategy to handle authentication .
CONTROL OPTIONS	

Examples

You want to package and publish some projects in a C# class library to your Azure Artifacts feed.

```

`-- Message
  |-- Message.sln
  '-- ShortGreeting
    |-- ShortGreeting.csproj
    |-- Class1.cs
    '-- Properties
      |-- AssemblyInfo.cs
  '-- LongGreeting
    |-- LongGreeting.csproj
    |-- Class1.cs
    '-- Properties
      |-- AssemblyInfo.cs

```

Prepare

AssemblyInfo.cs

Make sure your AssemblyInfo.cs files contain the information you want shown in your packages. For example, `AssemblyCompanyAttribute` will be shown as the author, and `AssemblyDescriptionAttribute` will be shown as the description.

Variables tab

NAME	VALUE
<code>\$(BuildConfiguration)</code>	<code>release</code>
<code>\$(BuildPlatform)</code>	<code>any cpu</code>

Options

SETTING	VALUE
Build number format	<code>\$(BuildDefinitionName)_\$(Year:yyyy).\$(Month).\$(DayOfMonth)\$(Rev:.rrrr)</code>

Publish to Azure Artifacts

Make sure you've prepared the build as described [above](#).

Create the feed

See [Create a feed](#).

Build tasks

 <p>Build: Visual Studio Build</p>	<p>Build your solution.</p> <ul style="list-style-type: none"> • Solution: <code>***.sln</code> • Platform: <code>\$(BuildPlatform)</code> • Configuration: <code>\$(BuildConfiguration)</code>
 <p>Package: NuGet Packager</p>	<p>Package your projects.</p> <ul style="list-style-type: none"> • Path/Pattern to nuspec files: <code>***.csproj</code> • Use Build number to version package: Selected • Advanced, Configuration to Package: <code>Release</code>
 <p>Package: NuGet Publisher</p>	<p>Publish your packages to Azure Artifacts.</p> <ul style="list-style-type: none"> • Path/Pattern to nupkg: <code>***.nupkg</code> • Feed type: Internal NuGet Feed • Internal feed URL: See Find your NuGet package source URL.

Publish to NuGet.org

Make sure you've prepared the build as described [above](#).

Register with NuGet.org

If you haven't already, [register with NuGet.org](#).

Build tasks

 <p>Build: Visual Studio Build</p>	<p>Build your solution.</p> <ul style="list-style-type: none"> • Solution: <code>***.sln</code> • Platform: <code>\$(BuildPlatform)</code> • Configuration: <code>\$(BuildConfiguration)</code>
 <p>Package: NuGet Packager</p>	<p>Package your projects.</p> <ul style="list-style-type: none"> • Path/Pattern to nuspec files: <code>***.csproj</code> • Use Build number to version package: Selected • Advanced, Configuration to Package: <code>Release</code>
 <p>Package: NuGet Publisher</p>	<p>Publish your packages to NuGet.org.</p> <ul style="list-style-type: none"> • Path/Pattern to nupkg: <code>***.nupkg</code> • Feed type: External NuGet Feed • NuGet Server Endpoint: <input type="text"/>  <ol style="list-style-type: none"> 1. Click "New service connection", and then click Generic. 2. On the Add New Generic Connection dialog box: <ul style="list-style-type: none"> ○ Connection Name: <code>NuGet</code> ○ Server URL: <code>https://nuget.org/</code> ○ User name: <code>{your-name}</code> ○ Password/Token Key: Paste API Key from your NuGet account.

NuGet task

11/6/2018 • 11 minutes to read • [Edit Online](#)

Version 2.* | [Other versions](#)

Azure Pipelines | TFS 2018

Use this task in a build or release pipeline to install and update NuGet package dependencies, or package and publish NuGet packages.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

If your code depends on NuGet packages, make sure to add this step before your [Visual Studio Build step](#). Also make sure to clear the deprecated **Restore NuGet Packages** checkbox in that step.

TIP

Looking for help to get started? See the how-to's for [restoring](#) and [publishing](#) packages.

TIP

This version of the NuGet task uses NuGet 4.1.0 by default. To select a different version of NuGet, use the [Tool Installer](#).

NOTE

Using or creating .NET Core or .NET Standard packages? Use the [.NET Core](#) task, which has full support for all package scenarios currently supported by dotnet, including restore, pack, and nuget push.

YAML snippet

```

# NuGet
# Restore, pack, or push NuGet packages, or run a NuGet command. Supports NuGet.org and authenticated feeds like
# Package Management and MyGet. Uses NuGet.exe and works with .NET Framework apps. For .NET Core and .NET Standard
# apps, use the .NET Core task.
- task: NuGetCommand@2
  inputs:
    #command: 'restore' # Options: restore, pack, push, custom
    #restoreSolution: '**/*.sln' # Required when command == Restore
    #feedsToUse: 'select' # Options: select, config
    #vstsFeed: # Required when feedsToUse == Select
    #includeNuGetOrg: true # Required when feedsToUse == Select
    #nugetConfigPath: # Required when feedsToUse == Config
    #externalFeedCredentials: # Optional
    #noCache: false
    #disableParallelProcessing: false
    restoreDirectory:
      #verbosityRestore: 'Detailed' # Options: quiet, normal, detailed
    #packagesToPush:
      '$(Build.ArtifactStagingDirectory)/**/*.{nupkg,!$(Build.ArtifactStagingDirectory)/**/*.{symbols,nupkg}}' # Required
    when command == Push
      #nuGetFeedType: 'internal' # Required when command == Push# Options: internal, external
      #publishVstsFeed: # Required when command == Push & NuGetFeedType == Internal
      #allowPackageConflicts: # Optional
      #publishFeedCredentials: # Required when command == Push & NuGetFeedType == External
      #verbosityPush: 'Detailed' # Options: quiet, normal, detailed
      #packagesToPack: '**/*.csproj' # Required when command == Pack
      #configuration: '$(BuildConfiguration)' # Optional
      #packDestination: '$(Build.ArtifactStagingDirectory)' # Optional
      #versioningScheme: 'off' # Options: off, byPrereleaseNumber, byEnvVar, byBuildNumber
      #includeReferencedProjects: false # Optional
      #versionEnvVar: # Required when versioningScheme == ByEnvVar
      #majorVersion: '1' # Required when versioningScheme == ByPrereleaseNumber
      #minorVersion: '0' # Required when versioningScheme == ByPrereleaseNumber
      #patchVersion: '0' # Required when versioningScheme == ByPrereleaseNumber
      #packTimezone: 'utc' # Required when versioningScheme == ByPrereleaseNumber# Options: utc, local
      #includeSymbols: false # Optional
      #toolPackage: # Optional
      #buildProperties: # Optional
      #basePath: # Optional
      #verbosityPack: 'Detailed' # Options: quiet, normal, detailed
    #arguments: # Required when command == Custom

```

Versioning schemes

For **byPrereleaseNumber**, the version will be set to whatever you choose for major, minor, and patch, plus the date and time in the format `yyyymmdd-hhmmss`.

For **byEnvVar**, the version will be set as whatever environment variable, e.g. `$(MyVersion)`, you provide. Make sure the environment variable is set to a proper SemVer e.g. `1.2.3` or `1.2.3-beta1`.

For **byBuildNumber**, the version will be set to the build number, ensure that your build number is a proper SemVer e.g. `1.0.$(Rev:r)`.

Restore NuGet packages

Demand

None

Arguments

ARGUMENT	DESCRIPTION
Path to solution, packages.config, or project.json	Copy the Solution argument in your Visual Studio Build step and paste it here, or create a link using the Link button in the information panel.
FEEDS AND AUTHENTICATION	

ARGUMENT	DESCRIPTION
Feeds to use	<p>Feed(s) I select here:</p> <ul style="list-style-type: none"> Select this option to use NuGet.org and/or one Azure Artifacts/Package Management feed in the same organization/collection as the build. <p>Feeds in my NuGet.config:</p> <ul style="list-style-type: none"> Select this option to use feeds specified in a NuGet.config file you've checked into source control. Credentials for feeds outside this organization/collection can be used to inject credentials you've provided as a NuGet service connection into your NuGet.config as the build runs. Azure Artifacts users: see the walkthrough for help using packages from feeds in multiple Azure DevOps organizations.
ADVANCED	
Disable local cache	Prevents NuGet from using packages from local machine caches.
Destination directory	Specifies the folder in which packages are installed. If no folder is specified, packages are restored into a packages/ folder alongside the selected solution, packages.config, or project.json.
Verbosity	Specifies the amount of detail displayed in the output.
CONTROL OPTIONS	

Examples

Install NuGet dependencies

You're building a Visual Studio solution that depends on a NuGet feed.

```

`-- ConsoleApplication1
  |-- ConsoleApplication1.sln
  |-- NuGet.config
  '-- ConsoleApplication1
    |-- ConsoleApplication1.csproj
  
```

Build steps

 Package: NuGet	Install your NuGet package dependencies. <ul style="list-style-type: none"> Path to solution, packages.config, or project.json: <code>**/*.sln</code> Feeds to use: Feeds in my NuGet.config Path to NuGet.config: <code>ConsoleApplication1/NuGet.config</code>
 Build: Visual Studio Build	Build your solution. <ul style="list-style-type: none"> Solution: <code>**/*.sln</code> Restore NuGet Packages: (Important) Make sure this option is cleared.

Pack NuGet packages

Demand

None

Arguments

ARGUMENT	DESCRIPTION
Path to csproj or nuspec file(s) to pack	<p>Specify .csproj files (for example, <code>***.csproj</code>) for simple projects. In this case:</p> <ul style="list-style-type: none">• The packager compiles the .csproj files for packaging.• You must specify Configuration to Package (see below).• You do not have to check in a .nuspec file. If you do check one in, the packager honors its settings and replaces tokens such as <code>\$id\$</code> and <code>\$description\$</code>. <p>Specify .nuspec files (for example, <code>***.nuspec</code>) for more complex projects, such as multi-platform scenarios in which you need to compile and package in separate steps. In this case:</p> <ul style="list-style-type: none">• The packager does not compile the .csproj files for packaging.• Each project is packaged only if it has a .nuspec file checked in.• The packager does not replace tokens in the .nuspec file (except the <code><version/></code> element, see Use build number to version package, below). You must supply values for elements such as <code><id/></code> and <code><description/></code>. The most common way to do this is to hardcode the values in the .nuspec file. <p>To package a single file, click the ... button and select the file. To package multiple files, use file matching patterns. Note that these patterns were updated in version 2 of the NuGet task; if you have a pattern that contains <code>-:</code>, use <code>!</code> instead.</p>
Configuration to package	If you are packaging a .csproj file, you must specify a configuration that you are building and that you want to package. For example: <code>Release</code> or <code>\$(BuildConfiguration)</code>
Package folder	(Optional) Specify the folder where you want to put the packages. You can use a variable such as <code>\$(Build.ArtifactStagingDirectory)</code> . If you leave it empty, the package will be created in the root of your source tree.
PACK OPTIONS	
Automatic package versioning	This blog post provides an overview of the automatic package versioning available here.
ADVANCED	
Additional build properties	Semicolon delimited list of properties used to build the package. For example, you could replace <code><description>\$description\$</description></code> in the .nuspec file this way: <code>Description="This is a great package"</code> . Using this argument is equivalent to supplying properties from nuget pack with the <code>-Properties</code> option.
Verbosity	Specifies the amount of detail displayed in the output.
CONTROL OPTIONS	

Push NuGet packages

Demands

None

Arguments

ARGUMENT	DESCRIPTION
Path to NuGet package(s) to publish	<p>Specify the packages you want to publish.</p> <ul style="list-style-type: none"> Default value: \$(Build.ArtifactStagingDirectory)/*.nupkg To publish a single package, click the ... button and select the file. Use file matching patterns to publish multiple packages. Note that these patterns were updated in version 2 of the NuGet task; if you have a pattern that contains <code>-:</code>, use <code>!</code> instead. Use variables to specify directories. For example, if you specified <code>\$(Build.ArtifactStagingDirectory)\</code> as the package folder in the pack step above, you could specify \$(Build.ArtifactStagingDirectory)***.nupkg here.
Target feed location	<ul style="list-style-type: none"> This organization/collection publishes to an Azure Artifacts/Package Management feed in the same organization/collection as the build. After you select this option, select the target feed from the dropdown. <ul style="list-style-type: none"> "Allow duplicates to be skipped" allows you to continually publish a set of packages and only change the version number of the subset of packages that changed. It allows the task to report success even if some of your packages are rejected with 409 Conflict errors. External NuGet server (including other organizations/collections) publishes to an external server such as NuGet, MyGet, or an Azure Artifacts/Package Management feed in another Azure DevOps organization or TFS collection. After you select this option, you create and select a NuGet service connection.
ADVANCED	
Verbosity	Specifies the amount of detail displayed in the output.
CONTROL OPTIONS	

Publishing symbols

When you push packages to an Azure Artifacts/Package Management feed, you can also publish symbols using the [Index Sources & Publish Symbols task](#).

Publishing packages to TFS with IIS Basic authentication enabled

This task is unable to publish NuGet packages to a TFS Package Management feed that is running on a TFS server with IIS Basic authentication enabled. [See here](#) for more details.

Custom NuGet command

YAML snippet

```

# NuGet Command
# Deprecated: use the "NuGet" task instead. It works with the new Tool Installer framework so you can easily use
new versions of NuGet without waiting for a task update, provides better support for authenticated feeds outside
this organization/collection, and uses NuGet 4 by default.
- task: NuGet@0
  inputs:
    command:
    arguments:

```

Arguments

ARGUMENT	DESCRIPTION
Command and arguments	NuGet command and arguments you want to pass as your custom command.

End-to-end example

You want to package and publish some projects in a C# class library to your Azure Artifacts feed.

You want to package and publish some projects in a C# class library to your TFS Package Management feed.

```

`-- Message
  |-- Message.sln
  '-- ShortGreeting
    |-- ShortGreeting.csproj
    |-- Class1.cs
    '-- Properties
      |-- AssemblyInfo.cs
  '-- LongGreeting
    |-- LongGreeting.csproj
    |-- Class1.cs
    '-- Properties
      |-- AssemblyInfo.cs

```

Prepare

AssemblyInfo.cs

Make sure your AssemblyInfo.cs files contain the information you want shown in your packages. For example, `AssemblyCompanyAttribute` will be shown as the author, and `AssemblyDescriptionAttribute` will be shown as the description.

Variables tab

NAME	VALUE
<code>\$(BuildConfiguration)</code>	<code>release</code>
<code>\$(BuildPlatform)</code>	<code>any cpu</code>

Options

SETTING	VALUE
Build number format	<code>\$(BuildDefinitionName)_\$(Year:yyyy).\$(Month).\$(DayOfMonth)\$(Rev:###)</code>

Publish to Azure Artifacts

Publish to a TFS feed

1. Make sure you've prepared the build as described [above](#).
2. If you haven't already, [create a feed](#).
3. Add the following build steps:

 Package: NuGet	<p>Install your NuGet package dependencies.</p> <ul style="list-style-type: none"> Path to solution, packages.config, or project.json: <code>**/*.sln</code> Feeds to use: Feeds I select here Use packages from NuGet.org: Checked
 Build: Visual Studio Build	<p>Build your solution.</p> <ul style="list-style-type: none"> Solution: <code>***.sln</code> Platform: <code>\$(BuildPlatform)</code> Configuration: <code>\$(BuildConfiguration)</code>
 Package: NuGet	<p>Package your projects.</p> <ul style="list-style-type: none"> Command: pack Path to csproj or nuspec file(s) to pack: <code>**/*.csproj</code> Configuration to Package: <code>Release</code> Package Folder: <code>\$(Build.ArtifactStagingDirectory)</code> Pack options > Automatic package versioning: Use the build number
 Package: NuGet	<p>Publish your packages.</p> <ul style="list-style-type: none"> Command: push Path to NuGet package(s) to publish: <code>\$(Build.ArtifactStagingDirectory)</code> Target feed location: This organization/collection Target feed: Select your feed

Option 2: publish to NuGet.org

1. Make sure you've prepared the build as described [above](#).
2. If you haven't already, [register with NuGet.org](#).
3. Use the steps in the previous section, but substitute the final step for the step shown here.

 Package: NuGet	<p>Publish your packages to NuGet.org.</p> <ul style="list-style-type: none"> Command: push Path to NuGet package(s) to publish: <code>\$(Build.ArtifactStagingDirectory)</code> Target feed location: External NuGet server NuGet server: Create a new NuGet service connection with your NuGet.org ApiKey and select it here
---	--

Task versions

Task: NuGet (formerly NuGet Restore at 1.* , NuGet Installer at 0.*)

TASK VERSION	AZURE PIPELINES	TFS
2.*	Available	Appeared in 2018
1.*	Deprecated but available	Appeared in 2017 Update 2, deprecated in 2018

TASK VERSION	AZURE PIPELINES	TFS
0.*	Deprecated but available	Appeared in 2017, deprecated in 2017 Update 2

Task: NuGet Packager

TASK VERSION	AZURE PIPELINES	TFS
0.*	Deprecated but available	Available in TFS < 2018, deprecated in TFS >= 2018

Task: NuGet Publisher

TASK VERSION	AZURE PIPELINES	TFS
0.*	Deprecated but available	Available in TFS < 2018, deprecated in TFS >= 2018

Task: NuGet Command

TASK VERSION	AZURE PIPELINES	TFS
0.*	Deprecated but available	Available in TFS < 2017 Update 2, deprecated in TFS >= 2018

Open source

These tasks are open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Why should I check in a NuGet.Config?

Checking a NuGet.Config into source control ensures that a key piece of information needed to build your project—the location of its packages—is available to every developer that checks out your code.

However, for situations where a team of developers works on a large range of projects, it's also possible to add an Azure Artifacts feed to the global NuGet.Config on each developer's machine. In these situations, using the "Feeds I select here" option in the NuGet task replicates this configuration.

Where can I learn about Azure Artifacts?

[Package Management in Azure Artifacts and TFS](#)

Where can I learn more about NuGet?

[NuGet Docs](#) Overview

[NuGet Create](#) Packaging and publishing

[NuGet Consume](#) Setting up a solution to get dependencies

What other kinds of apps can I build?

[Build your app](#)

What other kinds of build tasks are available?

[Specify your build tasks](#)

How do we protect our codebase from build breaks?

- Git: [Improve code quality with branch policies](#) with an option to require that code builds before it can be merged to a branch. For GitHub repositories, similar policies are available in GitHub's repository settings under

Branches.

- TFVC: [Use gated check-in](#).

How do I modify other parts of my build pipeline?

- [Specify your build tasks](#) to run tests, scripts, and a wide range of other processes.
- [Specify build options](#) such as specifying how completed builds are named, building multiple configurations, creating work items on failure.
- [Specify the repository](#) to pick the source of the build and modify options such as how the agent workspace is cleaned.
- [Set build triggers](#) to modify how your CI builds run and to specify scheduled builds.
- [Specify build retention policies](#) to automatically delete old builds.

I selected parallel multi-configuration, but only one build is running at a time.

If you're using Azure Pipelines, you might need more parallel jobs. See [Parallel jobs in Azure Pipelines](#).

How do I see what has changed in my build pipeline?

[View the change history of your build pipeline](#)

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

App Center Distribute task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

Use this task in a build or release pipeline to distribute app builds to testers and users through App Center.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

YAML snippet

```
# App Center Distribute
# Distribute app builds to testers and users via App Center
- task: AppCenterDistribute@1
  inputs:
    serverEndpoint:
    appSlug:
    appFile:
    #symbolsOption: 'Apple' # Optional. Options: apple
    #symbolsPath: # Optional
    #symbolsPdbFiles: '**/*.pdb' # Optional
    #symbolsDsymFiles: # Optional
    #symbolsMappingTxtFile: # Optional
    #symbolsIncludeParentDirectory: # Optional
    #releaseNotesOption: 'input' # Options: input, file
    #releaseNotesInput: # Required when releaseNotesOption == Input
    #releaseNotesFile: # Required when releaseNotesOption == File
    #distributionGroupId: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
App Center connection	(Required) Select the service connection for your Visual Studio App Center connection. To create one, click the Manage link and create a new service connection.
App slug	(Required) The app slug is in the format of {username}/{app_identifier} . To locate {username} and app_identifier for an app, click on its name from https://appcenter.ms/apps , and the resulting URL is in the format of https://appcenter.ms/users/{username}/apps/{app_identifier} . If you are using orgs, the app slug is of the format {orgname}/{app_identifier} .
Binary file path	(Required) Relative path from the repo root to the APK or IPA file you want to publish
Symbols type	(Optional) undefined

ARGUMENT	DESCRIPTION
Symbols path	(Optional) Relative path from the repo root to the symbols folder.
Symbols path (*.pdb)	(Optional) Relative path from the repo root to PDB symbols files. Path may contain wildcards.
dSYM path	(Optional) Relative path from the repo root to dSYM folder. Path may contain wildcards.
Mapping file	(Optional) Relative path from the repo root to Android's mapping.txt file.
Include all items in parent folder	(Optional) Upload the selected symbols file or folder and all other items inside the same parent folder. This is required for React Native apps.
Create release notes	(Required) undefined
Release notes	(Required) Release notes for this version.
Release notes file	(Required) Select a UTF-8 encoded text file which contains the Release Notes for this version.
Destination ID	(Optional) ID of the distribution group or store the app will deploy to. Leave it empty to use the default group.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Azure App Service Deploy task

11/19/2018 • 14 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to deploy to a range of App Services on Azure. The task works on cross-platform agents running Windows, Linux, or Mac; and uses several different [underlying deployment technologies](#).

The task works for [ASP.NET](#), [ASP.NET Core](#), [PHP](#), [Java](#), [Python](#), [Go](#), and [Node.js](#) based web applications.

The task can be used to deploy to a range of Azure App Services such as:

- [Web Apps on both Windows and Linux](#)
- [Web Apps for Containers](#)
- [Function Apps](#)
- [WebJobs](#)
- Apps configured under [Azure App Service Environments](#)

Pre-requisites for the task

The following pre-requisites must be set up in the target machine(s) in order for the task to work correctly.

- **App Service instance.** The task is used to deploy a Web App project or Azure Function project to an existing Azure App Service instance, which must exist before the task runs. The App Service instance can be created from the [Azure portal](#) and [configured](#) there. Alternatively, the [Azure PowerShell task](#) can be used to run [AzureRM PowerShell scripts](#) to provision and configure the Web App.
- **Azure Subscription.** To deploy to Azure, an Azure subscription must be [linked to the pipeline](#). The task does not work with the Azure Classic service connection, and it will not list these connections in the settings of the task.

YAML snippet

```

# Azure App Service Deploy
# Update Azure App Services on Windows, Web App on Linux with built-in images or Docker containers, ASP.NET,
.NET Core, PHP, Python or Node.js based Web applications, Function Apps on Windows or Linux with Docker
Containers, Mobile Apps, API applications, Web Jobs using Web Deploy / Kudu REST APIs
- task: AzureRmWebAppDeployment@4
  inputs:
    #connectionType: 'AzureRM' # Options: azureRM, publishProfile
    #azureSubscription: # Required when connectionType == AzureRM
    #publishProfilePath: '$(System.DefaultWorkingDirectory)/**/*.*.pubxml' # Required when connectionType ==
PublishProfile
    #publishProfilePassword: # Required when connectionType == PublishProfile
    #appType: 'webApp' # Required when connectionType == AzureRM# Options: webApp, webAppLinux,
webAppContainer, functionApp, functionAppLinux, functionAppContainer, apiApp, mobileApp
    #webAppName: # Required when connectionType == AzureRM
    #deployToSlotOrASE: false # Optional
    #resourceGroupName: # Required when deployToSlotOrASE == True
    #slotName: 'production' # Required when deployToSlotOrASE == True
    #dockerNamespace: # Required when appType == WebAppContainer || WebAppkind == FunctionAppContainer
    #dockerRepository: # Required when appType == WebAppContainer || WebAppkind == FunctionAppContainer
    #dockerImageTag: # Optional
    #virtualApplication: # Optional
    #packageForLinux: '$(System.DefaultWorkingDirectory)/**/*.*.zip' # Required when connectionType ==
PublishProfile || WebAppKind == WebApp || WebAppKind == ApiApp || WebAppKind == FunctionApp || WebAppKind ==
MobileApp || WebAppKind == WebAppLinux || WebAppKind == FunctionAppLinux
    #runtimeStack: # Optional
    #runtimeStackFunction: # Optional. Options: DOCKER|Microsoft/Azure-Functions-Dotnet-Core2.0:2.0,
DOCKER|Microsoft/Azure-Functions-Node8:2.0
    #startupCommand: # Optional
    #scriptType: # Optional. Options: , inline Script, file Path
    #inlineScript: ':: You can provide your deployment commands here. One command per line.' # Required when
scriptType == Inline Script
    #scriptPath: # Required when scriptType == File Path
    #webConfigParameters: # Optional
    #appSettings: # Optional
    #configurationSettings: # Optional
    #enableCustomDeployment: false # Optional
    #deploymentType: 'webDeploy' # Required when enableCustomDeployment == True# Options: webDeploy,
zipDeploy, runFromZip
    #takeAppOfflineFlag: true # Optional
    #setParameterFile: # Optional
    #removeAdditionalFilesFlag: false # Optional
    #excludeFilesFromAppDataFlag: true # Optional
    #additionalArguments: '-retryAttempts:6 -retryInterval:10000' # Optional
    #renameFilesFlag: true # Optional
    #enableXmlTransform: # Optional
    #enableXmlVariableSubstitution: # Optional
    #JSONFiles: # Optional

```

Arguments

ARGUMENT	PARAMETER	DESCRIPTION
Connection type	connectionType	(Required) Select Azure Resource Manager .
Azure subscription	azureSubscription	(Required) Select your Azure Resource Manager subscription. If none exists, choose Manage to navigate to the Services page in the administration section. Choose New Service Endpoint and select Azure Resource Manager from the list, then enter the required details.

Argument	Parameter	Description
Publish profile path	publishProfilePath	(Required) The path to the file containing the publishing information.
Publish profile password	publishProfilePassword	(Required) The password for the profile file. Consider storing the password in a secret variable and using that variable here. Example: <code>\$(Password)</code> .
App Service type	appType	(Required) Select the Azure App Service type. The app types supported are Function App, Web App on Windows, Web App on Linux, Web App for Containers, and Azure App Service Environments.
App Service name	webAppName	(Required) Select an existing Azure App Service or enter the name of the Web App if it was provisioned dynamically using the Azure PowerShell task and AzureRM PowerShell scripts .
Deploy to Slot or App Service Environment	deployToSlotOrASE	(Optional) Select this option to deploy to an existing slot other than the Production slot. Do not select it if the Web project is being deployed to the Production slot. The Web App itself is the Production slot.
Resource group	resourceGroupName	(Required) Select the Azure Resource Group that contains the Azure App Service, or enter the name of the Azure Resource Group if has been dynamically provisioned using the Azure Resource Group Deployment task or Azure PowerShell task . This is a required parameter if the option Deploy to Slot has been selected.
Slot	slotName	(Required) Select the slot to deploy the Web project to, or enter the name of the slot if has been dynamically provisioned using Azure Resource Group Deployment task or Azure PowerShell task . This is a required parameter if the option Deploy to Slot has been selected.
Registry or Namespace	dockerNamespace	(Required) A globally unique top-level domain name for your specific registry or namespace. Note: the fully-qualified image name will be of the format: {registry or namespace}/{repository}:{tag} . For example, myregistry.azurecr.io/nginx:latest .

Argument	Parameter	Description
Image	dockerRepository	(Required) Name of the repository where the container images are stored. Note: the fully-qualified image name will be of the format: {registry or namespace}/{repository}:{tag} . For example, myregistry.azurecr.io/nginx:latest .
Tag	dockerImageTag	(Optional) Tags are optional, but are the mechanism that registries use to apply version information to Docker images. Note: the fully-qualified image name will be of the format: {registry or namespace}/{repository}:{tag} . For example, myregistry.azurecr.io/nginx:latest .
Virtual application	virtualApplication	(Optional) Specify the name of the Virtual Application that has been configured in the Azure portal. This option is not required for deployments to the website root. The Virtual Application must have been configured before deployment of the web project.
Package or folder	packageForLinux	(Required) Location of the Web App zip package or folder on the automation agent, or on a UNC path accessible to the automation agent such as \BudgetIT\Web\Deploy\Fabrikam.zip . Predefined system variables and wild cards such as \$(System.DefaultWorkingDirectory) *.zip can be also used here. Despite the name of the YAML property, this setting applies to both Linux and Windows apps.
Runtime Stack	runtimeStack	(Required) Web App on Linux offers two different options to publish your application: Custom image deployment (Web App for Containers) and App deployment with a built-in platform image (Web App on Linux). You will see this parameter only if you selected Linux Web App as the App type option.
Startup command	startupCommand	(Optional) The start up command for the container. For example, if you are using PM2 process manager for Nodejs, you can specify the PM2 file here.
Deployment script type	scriptType	(Optional) Customize the deployment by providing a script that runs on the Azure App Service after successful deployment. Choose inline deployment script or the path and name of a script file. Learn more .

Argument	Parameter	Description
Inline Script	inlineScript	(Required) The script to execute. See this example .
Deployment script path	scriptPath	(Required) The path and name of the script to execute.
Generate web.config parameters for Python, Node.js and Go apps	webConfigParameters	<p>(Optional) A standard Web.config will be generated and deployed to the Azure App Service if the app does not already have one. For example, for a Nodejs application, Web.config will have startup file and iis_node module values. Similarly, for Python (Bottle, Django, or Flask), Web.config will have details of the WSGI handler, Python path, and more. The task will generate a new Web.config only when the artifact package or folder does not contain an existing Web.config.</p> <p>Use this option to edit the values (such as the startup file) in the task-generated Web.config file. The default values populated by the task can be overridden by passing in Web.config parameters. This feature is for <i>only</i> the generated Web.config file. It is useful, for example, when the Azure App Service Manage task is used to install a specific Python version by using extensions, or when you want to provide a different startup file for Node.js. For Python, the path can be set as an output variable of the Azure App Service Manage task and then set as the Python path in the Web.config generated by this deploy task. You can try out this feature by selecting any Python, Nodejs, PHP release definition template. Learn more.</p>
App settings	appSettings	<p>(Optional) Edit web app application settings using the syntax <code>-key value</code>. Values containing spaces must be enclosed in double quotes. Examples:</p> <div style="border: 1px solid #ccc; padding: 2px;"><code>-Port 5000 -RequestTimeout 5000</code></div> <p>and</p> <div style="border: 1px solid #ccc; padding: 2px;"><code>-WEBSITE_TIME_ZONE "Eastern Standard Time"</code></div>
Configuration settings	configurationSettings	<p>(Optional) Edit web app configuration settings using the syntax <code>-key value</code>. Values containing spaces must be enclosed in double quotes. Example:</p> <div style="border: 1px solid #ccc; padding: 2px;"><code>-phpVersion 5.6 -linuxFxVersion: node 6.11</code></div>

Argument	Parameter	Description
Publish using Web Deploy	deploymentType	(Optional) Select Web Deploy, Container, Zip Deploy, RunFromZip, or Kudu REST APIs. By default, when this option is not selected, the task attempts to select the appropriate deployment technology based on the input package, app service type, and agent OS.
Take App Offline	takeAppOfflineFlag	(Optional) Select this option to take the Azure App Service offline by placing an app_offline.htm file in the root directory before the synchronization operation begins. The file will be removed after the synchronization completes successfully.
SetParameters file	setParametersFile	(Optional) The file is used to override the default settings in the Web Deploy zip package file, such as the IIS Web application name or the database connection string. This enables a single package to be deployed across multiple environments such as dev, test, staging, and production by using a specific parameter file for each environment.
Remove additional files at destination	removeAdditionalFilesFlag	(Optional) Select this option to delete the files in the Azure App Service that have no matching files in the Web App zip package. This ensures that, during deployment, any additional files in the Azure App Service are deleted, and the only files remaining are those in the Web App zip package. This will also remove all files related to any extension (for example, Application Insights) installed on this Azure App Service. To prevent this, enable Exclude files from the App_Data folder as well.
Exclude files from the App_Data folder	excludeFilesFromAppDataFlag	(Optional) Select this option to prevent files in the App_Data folder from being deployed to the Azure App Service. This is useful if a local database or a WebJob has previously been deployed to the Azure App Service and should not be deleted in subsequent deployments of the Web project.
Additional arguments	additionalArguments	(Optional) Additional Web Deploy arguments that will be appended to the MSDeploy command while deploying the Azure Web App such as <code>-disableLink:AppPoolExtension</code> and <code>-disableLink:ContentExtension</code> . This is useful for enabling and disabling rules, and for skipping synchronization of specific folders (more examples).

Argument	Parameter	Description
Rename locked files	renameFilesFlag	(Optional) Select this option to enable msdeploy flag MSDEPLOY_RENAME_LOCKED_FILES=1 in Azure App Service application settings. If set it enables msdeploy to rename locked files that are locked during app deployment
XML transformation	enableXmlTransform	(Optional) The configuration transformations will be run for *.Release.config and *.{EnvironmentName}.config on the *.config files. Configuration transformations run before variable substitution. XML transformations are supported only for Windows platform. Learn more .
XML variable substitution	enableXmlVariableSubstitution	(Optional) Variables defined in the build or release pipeline will be matched against the 'key' or 'name' entries in the appSettings , applicationSettings , and connectionStrings sections of any configuration file and parameters.xml file. Variable substitution runs after configuration transformations. Note: if the same variables are defined in the release pipeline and in the stage, the stage variables will supersede the release pipeline variables. Learn more .
JSON variable substitution	JSONFiles	(Optional) Provide a new line separated list of JSON files to substitute the variable values. File names must be relative to the root folder. To substitute JSON variables that are nested or hierarchical, specify them using JSONPath expressions. For example, to replace the value of ConnectionString in the sample below, define a variable named Data.DefaultConnection.Connection String in the build or release pipeline (or release pipelines stage). <pre>{ "Data": { "DefaultConnection": { "ConnectionString": "Server=(localdb)\SQLEXPRESS;Database=MyDB;Trusted_Connection=True" } } }</pre> Variable substitution runs after configuration transformations. Note: build and release pipeline variables are excluded from substitution. Learn more .

Output Variables

- **Web App Hosted URL:** Provide a name, such as `FabrikamWebAppURL` for the variable populated with the Azure App Service Hosted URL. The variable can be used as `$(variableName)`, for example `$(FabrikamWebAppURL)`, to refer to the hosted URL of the Azure App Service in subsequent tasks.

Usage notes

- The task works with the [Azure Resource Manager APIs](#) only.
- To ignore SSL errors, define a variable named `VSTS_ARM_REST_IGNORE_SSL_ERRORS` with value `true` in the release pipeline.
- For .NET apps targeting Web App on Windows, avoid deployment failure with the error `ERROR_FILE_IN_USE` by ensuring that **Rename locked files** and **Take App Offline** settings are enabled. For zero downtime deployment, use the slot swap option.
- When deploying to an App Service that has Application Insights configured, and you have enabled **Remove additional files at destination**, ensure you also enable **Exclude files from the App_Data folder** in order to maintain the Application insights extension in a safe state. This is required because the Application Insights continuous web job is installed into the App_Data folder.

Sample deployment script

The task provides an option to customize the deployment by providing a script that will run on the Azure App Service after the app's artifacts have been successfully copied to the App Service. You can choose to provide either an inline deployment script or the path and name of a script file in your artifact folder.

This is very useful when you want to restore your application dependencies directly on the App Service. Restoring packages for Node, PHP, and Python apps helps to avoid timeouts when the application dependency results in a large artifact being copied over from the agent to the Azure App Service.

An example of a deployment script is:

```
@echo off
if NOT exist requirements.txt (
    echo No Requirements.txt found.
    EXIT /b 0
)
if NOT exist "$(PYTHON_EXT)/python.exe" (
    echo Python extension not available >&2
    EXIT /b 1
)
echo Installing dependencies
call "$(PYTHON_EXT)/python.exe" -m pip install -U setuptools
if %errorlevel% NEQ 0 (
    echo Failed to install setuptools >&2
    EXIT /b 1
)
call "$(PYTHON_EXT)/python.exe" -m pip install -r requirements.txt
if %errorlevel% NEQ 0 (
    echo Failed to install dependencies>&2
    EXIT /b 1
)
```

Deployment methods

Several deployment methods are available in this task. Web Deploy (msdeploy.exe) is the default. To change the

deployment option, expand **Additional Deployment Options** and enable **Select deployment method** to choose from additional package-based deployment options.

Based on the type of Azure App Service and agent, the task chooses a suitable deployment technology. The different deployment technologies used by the task are:

- [Web Deploy](#)
- [Kudu REST APIs](#)
- [Container Registry](#)
- [Zip Deploy](#)
- [Run From Zip](#)
- Deploy .war files

By default, the task tries to select the appropriate deployment technology based on the input package type, App Service type, and agent operating system.

On Windows-based agents:

- For msdeploy (MSBuild-generated package) package, use Web Deploy
- When post-deployment script is provided, use Zip Deploy
- When the App Service type is Web App on Linux App, use Zip Deploy
- If a WAR file is provided, use War Deploy
- If a JAR file is provided, use Run From Zip
- For all others, use Run From Zip (through Zip Deploy)

On non-Windows agents (for any App Service type), the task relies on [Kudu REST APIs](#) to deploy the app.

Web Deploy

[Web Deploy](#) (msdeploy.exe) can be used to deploy a Web App on Windows or a Function App on a Windows agent. Web Deploy is feature-rich and offers options such as:

- **Rename locked files:** Rename any file that is still in use by the web server by enabling the msdeploy flag MSDEPLOY_RENAME_LOCKED_FILES=1 in the Azure App Service settings. This option, if set, enables msdeploy to rename files that are locked during app deployment.
- **Remove additional files at destination:** Deletes files in the Azure App Service that have no matching files in the App Service artifact package or folder being deployed.
- **Exclude files from the App_Data folder:** Prevent files in the App_Data folder (in the artifact package/folder being deployed) being deployed to the Azure App Service
- **Additional Web Deploy arguments:** Arguments that will be applied when deploying the Azure App Service. Example: `-disableLink:AppPoolExtension -disableLink:ContentExtension`. For more examples of Web Deploy operation settings, see [Web Deploy Operation Settings](#).

Install Web Deploy on the agent using the [Microsoft Web Platform Installer](#). Web Deploy 3.5 must be installed without the bundled SQL support. There is no need to choose any custom settings when installing Web Deploy. Web Deploy is installed at C:\Program Files (x86)\IIS\Microsoft Web Deploy V3.

Kudu REST APIs

[Kudu REST APIs](#) work on both Windows and Linux automation agents when the target is a Web App on Windows, Web App on Linux (built-in source), or Function App. The task uses Kudu to copy files to the Azure App service.

Container Registry

Works on both Windows and Linux automation agents when the target is a Web App for Containers. The task

updates the app by setting the appropriate container registry, repository, image name, and tag information. You can also use the task to pass a startup command for the container image.

Zip Deploy

Creates a .zip deployment package and deploys the file contents to the **wwwroot** folder of the App Service or Function App in Azure. This option overwrites all existing contents in the **wwwroot** folder. For more information, see [Zip deployment for Azure Functions](#).

RunFromZip

Creates the same deployment package as Zip Deploy. However, instead of deploying files to the **wwwroot** folder, the entire package is mounted by the Functions runtime and files in the **wwwroot** folder become read-only. For more information, see [Run your Azure Functions from a package file](#).

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Azure App Service Manage task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to start, stop, restart, slot swap, install site extensions, or enable continuous monitoring for an Azure App Service.

YAML snippet

```
# Azure App Service Manage
# Start, Stop, Restart, Slot swap, Install site extensions or Enable Continuous Monitoring for an Azure App Service
- task: AzureAppServiceManage@0
  inputs:
    azureSubscription:
      #action: 'Swap Slots' # Optional. Options: swap Slots, start Azure App Service, stop Azure App Service, restart Azure App Service, install Extensions, enable Continuous Monitoring, start All Continuous Webjobs, stop All Continuous Webjobs
      webAppName:
        #specifySlotOrASE: false # Optional
      #resourceGroupName: # Required when action == Swap Slots || SpecifySlot == True
      #sourceSlot: # Required when action == Swap Slots
      #swapWithProduction: true # Optional
      #targetSlot: # Required when action == Swap Slots && SwapWithProduction == False
      #preserveVnet: false # Optional
      #slot: 'production' # Required when action != Swap Slots && SpecifySlot == True
      #extensionsList: # Required when action == Install Extensions
      #outputVariable: # Optional
      #appInsightsResourceGroupName: # Required when action == Enable Continuous Monitoring
      #applicationInsightsResourceName: # Required when action == Enable Continuous Monitoring
      #applicationInsightsWebTestName: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Azure subscription	(Required) Select the Azure Resource Manager subscription
Action	(Optional) Action to be performed on the App Service. You can Start, Stop, Restart, Slot swap, Install site extensions or enable Continuous Monitoring for an Azure App Service
App Service name	(Required) Enter or select the name of an existing Azure App Service
Specify Slot or App Service Environment	(Optional) undefined
Resource group	(Required) Enter or Select the Azure Resource Group that contains the Azure App Service specified above
Source Slot	(Required) The swap action directs destination slot's traffic to the source slot

ARGUMENT	DESCRIPTION
Swap with Production	(Optional) Select the option to swap the traffic of source slot with production. If this option is not selected, then you will have to provide source and target slot names.
Target Slot	(Required) The swap action directs destination slot's traffic to the source slot
Preserve Vnet	(Optional) Preserve the Virtual network settings
Slot	(Required) undefined
Install Extensions	(Required) Site Extensions run on Microsoft Azure App Service. You can install set of tools as site extension and better manage your Azure App Service. The App Service will be restarted to make sure latest changes take effect.
Output variable	(Optional) Provide the variable name for the local installation path for the selected extension. This field is now deprecated and would be removed. Use LocalPathsForInstalledExtensions variable from Output Variables section in subsequent tasks.
Resource Group name for Application Insights	(Required) Enter or Select resource group where your application insights resource is available
Application Insights resource name	(Required) Select Application Insights resource where continuous monitoring data will be recorded. If your application insights resource is not listed here and you want to create a new resource, click on [+ New] button. Once the resource is created on Azure Portal, come back here and click on refresh button.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Azure CLI task

11/15/2018 • 3 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to run a shell or batch script containing Azure CLI commands against an Azure subscription.

This task is used to run Azure CLI commands on cross-platform agents running on Linux, macOS, or Windows operating systems.

This task is under development. If you encounter problems, or wish to share feedback about the task and features you would like to see, please [contact us](#).

What's new in Version 1.0

- Supports the new [AZ CLI 2.0](#), which is Python based
- Works with agents on Linux, macOS, and Windows
- To work with [Azure CLI 1.0](#), which is node based, switch to task version 0.0
- Both versions of Azure-CLI can coexist in the same system, but task V1.0 logs into the Python based AZ CLI using the user's subscription, whereas task V0.0 logs into the node based Azure CLI. Therefore, scripts should include only the appropriate corresponding commands.
- Limitations:
 - No support for Classic subscriptions. AZ CLI 2.0 supports only Azure Resource Manager (ARM) subscriptions
 - Currently, Microsoft-hosted agents do not have AZ CLI installed. You can either install using `npm install -g azure-cli` or use self-hosted agents with AZ CLI pre-installed

Demands

None

Prerequisites

- A Microsoft Azure subscription
- A service connection to your Azure account. You can use either:
 - [Azure Classic service connection](#)
 - [Azure Resource Manager service connection](#)
- Azure CLI installed on the computer(s) that run the build and release agent. See [Install the Azure CLI](#). If an agent is already running on the machine on which the Azure CLI is installed, restart the agent to ensure all the relevant stage variables are updated.

YAML snippet

```

# Azure CLI
# Run a Shell or Batch script with Azure CLI commands against an azure subscription
- task: AzureCLI@1
  inputs:
    azureSubscription:
      #scriptLocation: 'scriptPath' # Options: inlineScript, scriptPath
      #scriptPath: # Required when scriptLocation == ScriptPath
      #inlineScript: # Required when scriptLocation == InlineScript
      #arguments: # Optional
      #addSpnToEnvironment: false # Optional
      #workingDirectory: # Optional
      #failOnStandardError: false # Optional

```

Arguments

ARGUMENT	DESCRIPTION
Azure Connection Type	Required. Select the type of service connection used to define the connection to Azure. Choose Azure Classic or Azure Resource Manager . This parameter is shown only when the selected task version is 0.* as Azure CLI task v1.0 supports only Azure Resource Manager (ARM) subscriptions.
Azure Classic Subscription	Required if you select Azure Classic for the Azure Connection Type parameter. The name of an Azure Classic service connection configured for the subscription where the target Azure service, virtual machine, or storage account is located.
Azure RM Subscription	Required if you select Azure Resource Manager for the Azure Connection Type parameter. The name of an Azure Resource Manager service connection configured for the subscription where the target Azure service, virtual machine, or storage account is located. See Azure Resource Manager overview for more details.
Script Location	Required. The way that the script is provided. Choose Inline Script or Script Path (the default).
Inline Script	Required if you select Inline Script for the Script Location parameter. Type or copy the script code to execute here. You can include default variables , global variables, and stage variables.
Script Path	Required if you select Script Path for the Script Location parameter. The path to a linked artifact that is the .bat , .cmd , or .sh script you want to run. It can be a fully-qualified path, or a valid path relative to the default working directory.
Arguments	Optional. Any arguments you want to pass to the script.
Advanced - Working Directory	Optional. The working directory in which the script will execute. If not specified, this will be the folder containing the script file.
Advanced - Fail on Standard Error	Set this option if you want the build to fail if errors are written to the StandardError stream.

ARGUMENT	DESCRIPTION
Control options	See Control options

Related tasks

- [Azure Resource Group Deployment](#)
- [Azure Cloud Service Deployment](#)
- [Azure Web App Deployment](#)

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

Azure Cloud Service Deployment task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to deploy an Azure Cloud Service.

YAML snippet

```
# Azure Cloud Service Deployment
# Deploy an Azure Cloud Service
- task: AzureCloudPowerShellDeployment@1
  inputs:
    azureClassicSubscription:
    storageAccount:
    serviceName:
    serviceLocation:
    csPkg:
    csCfg:
    #slotName: 'Production'
    #deploymentLabel: '${Build.BuildNumber}' # Optional
    #appendDateTimeToLabel: false # Optional
    #allowUpgrade: true
    #simultaneousUpgrade: false # Optional
    #forceUpgrade: false # Optional
    #verifyRoleInstanceStatus: false # Optional
    #diagnosticStorageAccountKeys: # Optional
    #newServiceCustomCertificates: # Optional
    #newServiceAdditionalArguments: # Optional
    #newServiceAffinityGroup: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Azure subscription (Classic)	(Required) Azure Classic subscription to target for deployment.
Storage account	(Required) Storage account must exist prior to deployment.
Service name	(Required) Select or enter an existing cloud service name.
Service location	(Required) Select a region for new service deployment. Possible options are East US, East US 2, Central US, South Central US, West US, North Europe, West Europe and others.
CsPkg	(Required) Path of CsPkg under the default artifact directory.
CsCfg	(Required) Path of CsCfg under the default artifact directory.
Environment (Slot)	(Required) Production or Staging

ARGUMENT	DESCRIPTION
Deployment label	(Optional) Specifies the label name for the new deployment. If not specified, a Globally Unique Identifier (GUID) is used.
Append current date and time	(Optional) Appends current date and time to deployment label
Allow upgrade	(Required) When selected allows an upgrade to the Microsoft Azure deployment
Simultaneous upgrade	(Optional) Updates all instances at once. Your cloud service will be unavailable during update.
Force upgrade	(Optional) When selected sets the upgrade to a forced upgrade, which could potentially cause loss of local data.
Diagnostic storage account keys	<p>(Optional) Provide storage keys for diagnostics storage account in Role:Storagekey format. The diagnostics storage account name for each role will be obtained from diagnostics config file (.wadcfgx). If the .wadcfgx file for a role is not found, diagnostics extensions won't be set for the role. If the storage account name is missing in the .wadcfgx file, the default storage account will be used for storing diagnostics results and the storage key parameters from deployment task will be ignored. It's recommended to save <storage_account_key> as a secret variable unless there is no sensitive information in the diagnostics result for your stage.</p> <p>For example, WebRole: <WebRole_storage_account_key> WorkerRole: <WorkerRole_stoarge_account_key></p>
Custom certificates to import	<p>(Optional) Provide custom certificates in CertificatePfxBase64:CertificatePassword format. It's recommended to save <certificate_password> as a secret variable.</p> <p>For example, Certificate1: <Certificate1_password> Certificate2: <Certificate2_password></p>
Additional arguments	(Optional) Pass in additional arguments while creating a brand new service. These will be passed on to <code>New-AzureService</code> cmdlet. Eg: <code>-Label 'MyTestService'</code>
Affinity group	(Optional) While creating new service, this affinity group will be considered instead of using service location.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Azure File Copy task

11/15/2018 • 10 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015.3

Use this task in a build or release pipeline to copy files to Microsoft Azure storage blobs or virtual machines (VMs).

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions.

Service connections are called *service endpoints* in TFS 2018 and in older versions.

The task is used to copy application files and other artifacts that are required in order to install the app; such as PowerShell scripts, PowerShell-DSC modules, and more.

When the target is Azure VMs, the files are first copied to an automatically generated Azure blob container and then downloaded into the VMs. The container is deleted after the files have been successfully copied to the VMs.

The task uses **AzCopy**, the command-line utility built for fast copying of data from and into Azure storage accounts.

To dynamically deploy Azure Resource Groups that contain virtual machines, use the [Azure Resource Group Deployment](#) task. This task has a sample template that can perform the required operations to set up the WinRM HTTPS protocol on the virtual machines, open the 5986 port in the firewall, and install the test certificate.

NOTE

If you are deploying to Azure Static Websites as a container in blob storage, you must use **Version 2** or higher of the task in order to preserve the **\$web** container name.

Demands

None

YAML snippet

```

# Azure File Copy
# Copy files to Azure blob or VM(s)
- task: AzureFileCopy@2
  inputs:
    sourcePath:
      #azureConnectionType: 'ConnectedServiceNameARM' # Optional. Options: connectedServiceName,
      connectedServiceNameARM
      #azureClassicSubscription: # Required when azureConnectionType == ConnectedServiceName
      #azureSubscription: # Required when azureConnectionType == ConnectedServiceNameARM
      destination: # Options: azureBlob, azureVMs
      #classicStorage: # Required when azureConnectionType == ConnectedServiceName
      #storage: # Required when azureConnectionType == ConnectedServiceNameARM
      #containerName: # Required when destination == AzureBlob
      #blobPrefix: # Optional
      #cloudService: # Required when azureConnectionType == ConnectedServiceName && Destination == AzureVMs
      #resourceGroup: # Required when azureConnectionType == ConnectedServiceNameARM && Destination == AzureVMs
      #resourceFilteringMethod: 'machineNames' # Optional. Options: machineNames, tags
      #machineNames: # Optional
      #vmsAdminUserName: # Required when destination == AzureVMs
      #vmsAdminPassword: # Required when destination == AzureVMs
      #targetPath: # Required when destination == AzureVMs
      #additionalArgumentsForBlobCopy: # Optional
      #additionalArgumentsForVMCopy: # Optional
      #enableCopyPrerequisites: false # Optional
      #copyFilesInParallel: true # Optional
      #cleanTargetBeforeCopy: false # Optional
      #skipCACheck: true # Optional
      #outputStorageUri: # Optional
      #outputStorageContainerSasToken: # Optional

```

Arguments

ARGUMENT	DESCRIPTION
Source	Required. The source of the files to copy. Pre-defined system variables such as <code>\$(Build.Repository.LocalPath)</code> can be used. Names containing wildcards such as <code>*.zip</code> are not supported.
Azure Connection Type	Required. Select the type of service connection used to define the connection to Azure. Choose Azure Classic or Azure Resource Manager .
Azure Classic Subscription	Required if you select Azure Classic for the Azure Connection Type parameter. The name of an Azure Classic service connection configured for the subscription where the target Azure service, virtual machine, or storage account is located.
Azure RM Subscription	Required if you select Azure Resource Manager for the Azure Connection Type parameter. The name of an Azure Resource Manager service connection configured for the subscription where the target Azure service, virtual machine, or storage account is located. See Azure Resource Manager overview for more details.
Destination Type	Required. The type of target destination for the files. Choose Azure Blob or Azure VMs .

ARGUMENT	DESCRIPTION
Classic Storage Account	Required if you select Azure Classic for the Azure Connection Type parameter. The name of an existing storage account within the Azure subscription.
RM Storage Account	Required if you select Azure Resource Manager for the Azure Connection Type parameter. The name of an existing storage account within the Azure subscription.
Container Name	Required if you select Azure Blob for the Destination Type parameter. The name of the container to which the files will be copied. If a container with this name does not exist, a new one will be created.
Blob Prefix	Optional if you select Azure Blob for the Destination Type parameter. A prefix for the blob names, which can be used to filter the blobs. For example, using the build number enables easy filtering when downloading all blobs with the same build number.
Cloud Service	Required if you select Azure Classic for the Azure Connection Type parameter and Azure VMs for the Destination Type parameter. The name of the Azure Cloud Service in which the virtual machines run.
Resource Group	Required if you select Azure Resource Manager for the Azure Connection Type parameter and Azure VMs for the Destination Type parameter. The name of the Azure Resource Group in which the virtual machines run.
Select Machines By	Depending on how you want to specify the machines in the group when using the Filter Criteria parameter, choose Machine Names or Tags .
Filter Criteria	<p>Optional. A list of machine names or tag names that identifies the machines that the task will target. The filter criteria can be:</p> <ul style="list-style-type: none"> - The name of an Azure Resource Group. - An output variable from a previous task. - A comma-delimited list of tag names or machine names. <p>Format when using machine names is a comma-separated list of the machine FDQNs or IP addresses.</p> <p>Specify tag names for a filter as {TagName}:{Value} Example: <div style="border: 1px solid #ccc; padding: 2px;">Role:DB;OS:Win8.1</div></p>
Admin Login	<p>Required if you select Azure VMs for the Destination Type parameter. The user name of an account that has administrative permissions for all the target VMs.</p> <ul style="list-style-type: none"> - Formats such as username, domain\username, machine-name\username, and .\\username are supported. - UPN formats such as username@domain.com and built-in system accounts such as NT Authority\System are not supported.
Password	<p>Required if you select Azure VMs for the Destination Type parameter. The password for the account specified as the Admin Login parameter. Use the padlock icon for a variable defined in the Variables tab to protect the value, and insert the variable name here.</p>

Argument	Description
Destination Folder	<p>Required if you select Azure VMs for the Destination Type parameter. The folder in the Azure VMs to which the files will be copied. Environment variables such as <code>\$env:windir</code> and <code>\$env:systemroot</code> are supported. Examples: <code>\$env:windir\FabrikamFiber\Web</code> and <code>c:\FabrikamFiber</code></p>
Additional Arguments	<p>Optional. Any arguments you want to pass to the AzCopy.exe program for use when uploading to the blob and downloading to the VMs. See Transfer data with the AzCopy Command-Line Utility for more details. If you are using a Premium storage account, which supports only Azure page blobs, the pass <code>/BlobType:Page</code> as an additional argument.</p>
Enable Copy Prerequisites	<p>Available if you select Azure Resource Manager for the Azure Connection Type parameter and Azure VMs for the Destination Type parameter. Setting this option configures the Windows Remote Management (WinRM) listener over HTTPS protocol on port 5986, using a self-signed certificate. This configuration is required for performing copy operation on Azure virtual machines.</p> <ul style="list-style-type: none"> - If the target virtual machines are accessed through a load balancer, ensure an inbound NAT rule is configured to allow access on port 5986. - If the target virtual machines are associated with a Network Security Group (NSG), configure an inbound security rule to allow access on port 5986.
Copy in Parallel	<p>Available if you select Azure VMs for the Destination Type parameter. Setting this option causes the process to execute in parallel for the copied files. This can considerably reduce the overall time taken.</p>
Clean Target	<p>Available if you select Azure VMs for the Destination Type parameter. Setting this option causes all of the files in the destination folder to be deleted before the copy process starts.</p>
Test Certificate	<p>Available if you select Azure VMs for the Destination Type parameter. WinRM requires a certificate for the HTTPS transfer when copying files from the intermediate storage blob into the Azure VMs. If you set use a self-signed certificate, set this option to prevent the process from validating the certificate with a trusted certificate authority (CA).</p>
Output - Storage Container URI	<p>Optional. The name of a variable that will be updated with the URI of the storage container into which the files were copied. Use this variable as an input to subsequent tasks.</p>
Output - Storage Container SAS Token	<p>Optional. The name of a variable that will be updated with the Storage Access Security (SAS) token of the storage container into which the files were copied. Use this variable as an input to subsequent tasks. By default, the SAS token expires after 4 hours.</p>
Control options	<p>See Control options</p>

Related tasks

- [Azure Resource Group Deployment](#)
- [Azure Cloud Service Deployment](#)
- [Azure Web App Deployment](#)

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

What are the Azure PowerShell prerequisites for using this task?

The task requires Azure PowerShell to be installed on the machine running the automation agent. The recommended version is 1.0.2, but the task will work with version 0.9.8 and higher. You can use the [Azure PowerShell Installer v1.0.2](#) to obtain this.

What are the WinRM prerequisites for this task?

The task uses Windows Remote Management (WinRM) HTTPS protocol to copy the files from the storage blob container to the Azure VMs. This requires the WinRM HTTPS service to be configured on the VMs, and a suitable certificate installed.

If the VMs have been created without opening the WinRM HTTPS ports, follow these steps:

1. Configure an inbound access rule to allow HTTPS on port 5986 of each VM.
2. Disable [UAC remote restrictions](#).
3. Specify the credentials for the task to access the VMs using an administrator-level login in the simple form **username** without any domain part.
4. Install a certificate on the machine that runs the automation agent.
5. Set the **Test Certificate** parameter of the task if you are using a self-signed certificate.

For more details, see [this blog post](#).

What type of service connection should I choose?

The following table lists the storage accounts and the service connections that work with them. To identify whether a storage account is based on the classic APIs or the Resource Manager APIs, log into the [Azure portal](#) and browse for **Storage accounts (Classic)** or **Storage accounts**.

STORAGE ACCOUNT TYPE	AZURE SERVICE CONNECTIONS IN TFS/TS
Resource Manager	Azure Resource Manager service connection
Classic	Azure service connection with certificate-based or credentials-based authentication using a school or work account

- For Azure classic resources, use an **Azure** service connection type with certificate or credentials-based authentication. If you are using credentials-based authentication, ensure that the credentials are for a [school or work account](#). Microsoft accounts such as **joe@live.com** and **joe@hotmail.com** are not supported.
- For Azure Resource Manager VMs, use an **Azure Resource Manager** service connection type. See more details at [Automating Azure Resource Group deployment using a Service Principal](#).
- If you are using an **Azure Resource Manager** service connection type, or an **Azure** service connection type with certificate-based authentication, the task automatically filters appropriate classic storage accounts, the newer Azure Resource Manager storage accounts, and other fields. For example, the Resource Group or

cloud service, and the virtual machines.

- **Note:** Currently an **Azure** service connection type with credentials-based authentication does not filter the storage, Resource Group or cloud service, and virtual machine fields.

What happens if my Resource Group contains both Classic and Resource Manager VMs?

If the specified Resource Group contains both Azure Resource Manager and Classic VMs, the set of VMs that will be targeted depends on the connection type. For certificate-based connections and credentials-based connections, the copy operation will be performed only on Classic VMs. For Service Principal Name based connections, the copy operation will be performed on only Resource Manager VMs.

How do I create a school or work account for use with this task?

A suitable account can be easily created for use in a service connection:

1. Use the Azure portal to create a new user account in Azure Active Directory.
2. Add the Azure Active Directory user account to the co-administrators group in your Azure subscription.
3. Sign into the Azure portal with this user account and change the password.
4. Use the username and password of this account in the service connection. Deployments will be processed using this account.

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Azure Key Vault task

11/15/2018 • 3 minutes to read • [Edit Online](#)

Azure Pipelines

Overview

Use this task in a build or release pipeline to download secrets such as authentication keys, storage account keys, data encryption keys, .PFX files, and passwords from an [Azure Key Vault](#) instance. The task can be used to fetch the latest values of all or a subset of secrets from the vault, and set them as variables that can be used in subsequent tasks of a pipeline. The task is Node-based, and works with agents on Linux, macOS, and Windows.

Prerequisites

The task has the following pre-requisites:

- An Azure subscription linked to Azure Pipelines or Team Foundation Server using the [Azure Resource Manager service connection](#).
- An [Azure Key Vault](#) containing the secrets.

You can create a key vault:

- In the [Azure portal](#)
- By using [Azure PowerShell](#)
- By using the [Azure CLI](#)

Add secrets to a key vault:

- By using the PowerShell cmdlet [Set-AzureKeyVaultSecret](#). If the secret does not exist, this cmdlet creates it. If the secret already exists, this cmdlet creates a new version of that secret.
- By using the Azure CLI. To add a secret to a key vault, for example a secret named **SQLPassword** with the value **Pa\$\$w0rd**, type:

```
az keyvault secret set --vault-name 'ContosoKeyVault' --name 'SQLPassword' --value 'Pa$$w0rd'
```

When you want to access secrets:

- Ensure the Azure service connection has at least **Get** and **List** permissions on the vault. You can set these permissions in the [Azure portal](#):
 - Open the **Settings** blade for the vault, choose **Access policies**, then **Add new**.
 - In the **Add access policy** blade, choose **Select principal** and select the service principal for your client account.
 - In the **Add access policy** blade, choose **Secret permissions** and ensure that **Get** and **List** are checked (ticked).
 - Choose **OK** to save the changes.

YAML snippet

```

# Azure Key Vault
# Download Azure Key Vault Secrets
- task: AzureKeyVault@1
  inputs:
    azureSubscription:
    keyVaultName:
    #secretsFilter: '*' # Options: editableOptions

```

Arguments

PARAMETER	DESCRIPTION
Azure Subscription	Required. Select the service connection for the Azure subscription containing the Azure Key Vault instance, or create a new connection. Learn more
Key Vault	Required. Select the name of the Azure Key Vault from which the secrets will be downloaded.
Secrets filter	Required. A comma-separated list of secret names to be downloaded. Use the default value <code>*</code> to download all the secrets from the vault.

NOTE

Values are retrieved as strings. For example, if there is a secret named **connectionString**, a task variable `ConnectionString` is created with the latest value of the respective secret fetched from Azure key vault. This variable is then available in subsequent tasks.

If the value fetched from the vault is a certificate (for example, a PFX file), the task variable will contain the contents of the PFX in string format. You can use the following PowerShell code to retrieve the PFX file from the task variable:

```

$kvSecretBytes = [System.Convert]::FromBase64String($(PfxSecret))
$certCollection = New-Object System.Security.Cryptography.X509Certificates.X509Certificate2Collection
$certCollection.Import($kvSecretBytes,$null,
[System.Security.Cryptography.X509Certificates.X509KeyStorageFlags]::Exportable)

```

If the certificate file will be stored locally on the machine, it is good practice to encrypt it with a password:

```

#Get the file created
$password = 'your password'
$protectedCertificateBytes =
$certCollection.Export([System.Security.Cryptography.X509Certificates.X509ContentType]::Pkcs12, $password)
$pfxPath = [Environment]::GetFolderPath("Desktop") + "\MyCert.pfx"
[System.IO.File]::WriteAllBytes($pfxPath, $protectedCertificateBytes)

```

For more details, see [Get started with Azure Key Vault certificates](#).

Contact Information

Contact RM_Customer_Questions@microsoft.com if you discover issues using the task, to share feedback about the task, or to suggest new features that you would like to see.

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

Azure Monitor Alerts task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to configure alerts on available metrics for an Azure resource.

YAML snippet

```
# Azure Monitor Alerts
# Configure alerts on available metrics for an Azure resource
- task: AzureMonitorAlerts@0
  inputs:
    azureSubscription:
    resourceName:
    #resourceType: 'Microsoft.Insights/components' # Options: microsoft.Insights/Components, microsoft.Web/Sites, microsoft.Storage/StorageAccounts, microsoft.Compute/VirtualMachines
    resourceName:
    alertRules:
    #notifyServiceOwners: # Optional
    #notifyEmails: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Azure Subscription	(Required) Select the Azure Resource Manager subscription. Note: To configure new service connection, select the Azure subscription from the list and click 'Authorize'. If your subscription is not listed or if you want to use an existing Service Principal, you can setup an Azure service connection using 'Add' or 'Manage' button.
Resource Group	(Required) Select the Azure Resource Group that contains the Azure resource where you want to configure an alert.
Resource Type	(Required) Select the Azure resource type.
Resource name	(Required) Select name of Azure resource where you want to configure an alert.
Alert rules	(Required) List of Azure monitor alerts configured on selected Azure resource. To add or modify alerts, click on [...] button.
Subscription owners, contributors and readers	(Optional) Send email notification to everyone who has access to this resource group.
Additional administrator emails	(Optional) Add additional email addresses separated by semicolons(;) if you want to send email notification to additional people (whether or not you checked the "subscription owners..." box).
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Azure Database for Mysql Deployment task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to run your scripts and make changes to your Azure DB for Mysql. Note that this is an early preview version.

YAML snippet

```
# Azure Database for MySQL Deployment
# Run your scripts and make changes to your Azure Database for MySQL.
- task: AzureMysqlDeployment@1
  inputs:
    azureSubscription:
    serverName:
    #databaseName: # Optional
    sqlUsername:
    sqlPassword:
    #taskNameSelector: 'SqlTaskFile' # Optional. Options: sqlTaskFile, inlineSqlTask
    #sqlFile: # Required when taskNameSelector == SqlTaskFile
    #sqlInline: # Required when taskNameSelector == InlineSqlTask
    #sqlAdditionalArguments: # Optional
    #ipDetectionMethod: 'AutoDetect' # Options: autoDetect, ipAddressRange
    #startIpAddress: # Required when ipDetectionMethod == IPAddressRange
    #endIpAddress: # Required when ipDetectionMethod == IPAddressRange
    #deleteFirewallRule: true # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Azure Subscription	(Required) This is needed to connect to your Azure account. To configure new service connection, select the Azure subscription from the list and click 'Authorize'. If your subscription is not listed or if you want to use an existing Service Principal, you can setup an Azure service connection using 'Add' or 'Manage' button.
Host Name	(Required) Server name of 'Azure DB for Mysql'. Example: fabrikam.mysql.database.azure.com. When you connect using Mysql Workbench, this is the same value that is used for 'Hostname' in 'Parameters'
Database Name	(Optional) The name of database, if you already have one, on which the below script is needed to be run, else the script itself can be used to create the database.
Server Admin Login	(Required) Azure Database for MySQL server supports native MySQL authentication. You can connect and authenticate to a server with the server's admin login. Example: bbo1@fabrikam. When you connect using Mysql Workbench, this is the same value that is used for 'Username' in 'Parameters'.

ARGUMENT	DESCRIPTION
Password	(Required) Administrator password for Azure DB for Mysql. In case you don't recall the password you can change the password from Azure portal . It can be variable defined in the pipeline. Example : \$(password). Also, you may mark the variable type as 'secret' to secure it.
Type	(Optional) Select one of the options between Script File & Inline Script.
MySQL Script	(Required) Full path of the script file on the automation agent or on a UNC path accessible to the automation agent like, \\BudgetIT\DeployBuilds\script.sql. Also, predefined system variables like, \$(agent.releaseDirectory) can also be used here. A file containing SQL statements can be used here.?
Inline MySQL Script	(Required) Enter the MySQL script to execute on the Database selected above.
Additional Mysql Arguments	(Optional) Additional options supported by mysql simple SQL shell. These options will be applied when executing the given file on the Azure DB for MySQL.? Example: You can change to default tab separated output format to HTML or even XML format. Or if you have problems due to insufficient memory for large result sets, use the --quick option.?
Specify Firewall Rules Using	(Required) For successful execution of the task, we need to enable administrators to access the Azure Database for MySQL Server from the IP Address of the automation agent. By selecting auto-detect you can automatically add firewall exception for range of possible IP Address of automation agent ?or else you can specify the range explicitly.
Start IP Address	(Required) The starting IP Address of the automation agent machine pool like 196.21.30.50 .
End IP Address	(Required) The ending IP Address of the automation agent machine pool like 196.21.30.65 .
Delete Rule After Task Ends	(Optional) If selected, the added exception for IP addresses of the automation agent will be removed for corresponding Azure Database for MySQL.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Azure Policy Check Gate task

11/19/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Security and compliance assessment with Azure policies on resources that belong to the resource group and Azure subscription.

YAML snippet

```
# Azure Policy Check Gate
# Security and compliance assessment with Azure policies on resources that belong to the resource group and
# Azure subscription.
- task: AzurePolicyCheckGate@0
  inputs:
    azureSubscription:
    resourceGroupName:
    #resources: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Azure subscription	(Required) Select the Azure Resource Manager subscription to enforce the policies.
Resource group	(Required) Provide name of a resource group.
Resource name	(Optional) Select name of Azure resources for which you want to check the policy compliance.
CONTROL OPTIONS	

Q&A

Azure PowerShell task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to run a PowerShell script within an Azure environment.

YAML snippet

```
# Azure PowerShell
# Run a PowerShell script within an Azure environment
- task: AzurePowerShell@3
  inputs:
    #azureConnectionType: 'ConnectedServiceNameARM' # Optional. Options: connectedServiceName,
    connectedServiceNameARM
    #azureClassicSubscription: # Required when azureConnectionType == ConnectedServiceName
    #azureSubscription: # Required when azureConnectionType == ConnectedServiceNameARM
    #scriptType: 'FilePath' # Optional. Options: filePath, inlineScript
    #scriptPath: # Optional
    #inline: '# You can write your azure powershell scripts inline here. # You can also pass predefined and
    custom variables to this script using arguments' # Optional
    #scriptArguments: # Optional
    #errorActionPreference: 'stop' # Optional. Options: stop, continue, silentlyContinue
    #failOnStandardError: false # Optional
    #azurePowerShellVersion: 'OtherVersion' # Optional. Options: latestVersion, otherVersion
    #preferredAzurePowerShellVersion: # Required when azurePowerShellVersion == OtherVersion
```

Arguments

ARGUMENT	DESCRIPTION
Azure Connection Type	(Optional)
Azure Classic Subscription	(Required) Azure Classic subscription to configure before running PowerShell
Azure Subscription	(Required) Azure Resource Manager subscription to configure before running PowerShell
Script Type	(Optional) Type of the script: File Path or Inline Script
Script Path	(Optional) Path of the script. Should be fully qualified path or relative to the default working directory.
Inline Script	(Optional) Enter the script to execute.
Script Arguments	(Optional) Additional parameters to pass to PowerShell. Can be either ordinal or named parameters.
ErrorActionPreference	(Optional) Select the value of the ErrorActionPreference variable for executing the script.

ARGUMENT	DESCRIPTION
Fail on Standard Error	(Optional) If this is true, this task will fail if any errors are written to the error pipeline, or if any data is written to the Standard Error stream.
Azure PowerShell Version	(Optional) In case of Microsoft-hosted agents, the supported Azure PowerShell Versions are: 2.1.0, 3.8.0, 4.2.1 and 5.1.1 (Hosted VS2017 pool), 3.6.0 (Hosted pool). To pick the latest version available on the agent, select "Latest installed version". For self-hosted agents you can specify preferred version of Azure PowerShell using "Specify version"
Preferred Azure PowerShell Version	(Required) Preferred Azure PowerShell Version needs to be a proper semantic version eg. 1.2.3. Regex like 2.*,2.3.* is not supported. The Hosted VS2017 pool currently supports versions: 2.1.0, 3.8.0, 4.2.1, 5.1.1
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Azure Resource Group Deployment task

11/6/2018 • 5 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to deploy, start, stop, and delete Azure Resource Groups.

YAML snippet

```
# Azure Resource Group Deployment
# Deploy an Azure resource manager (ARM) template to a resource group. You can also start, stop, delete,
# deallocate all Virtual Machines (VM) in a resource group
- task: AzureResourceGroupDeployment@2
  inputs:
    azureSubscription:
      #action: 'Create Or Update Resource Group' # Options: create Or Update Resource Group, select Resource
      Group, start, stop, stopWithDeallocate, restart, delete, deleteRG
      resourceName:
        #location: # Required when action == Create Or Update Resource Group
        #templateLocation: 'Linked artifact' # Options: linked Artifact, uRL Of The File
        #csmFileLink: # Required when templateLocation == URL Of The File
        #csmParametersFileLink: # Optional
        #csmFile: # Required when TemplateLocation == Linked Artifact
        #csmParametersFile: # Optional
        #overrideParameters: # Optional
        #deploymentMode: 'Incremental' # Options: incremental, complete, validation
        #enableDeploymentPrerequisites: 'None' # Optional. Options: none, configureVMwithWinRM,
        configureVMWithDGAgent
        #teamServicesConnection: # Required when enableDeploymentPrerequisites == ConfigureVMWithDGAgent
        #teamProject: # Required when enableDeploymentPrerequisites == ConfigureVMWithDGAgent
        #deploymentGroupName: # Required when enableDeploymentPrerequisites == ConfigureVMWithDGAgent
        #copyAzureVMTags: # Optional
        #runAgentServiceAsUser: # Optional
        #userName: # Required when enableDeploymentPrerequisites == ConfigureVMWithDGAgent &&
        RunAgentServiceAsUser == True
        #password: # Optional
        #outputVariable: # Optional
        #deploymentOutputs: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Azure subscription	(Required) Select the Azure Resource Manager subscription for the deployment.
Action	(Required) Action to be performed on the Azure resources or resource group.
Resource group	(Required) Provide the name of a resource group.
Location	(Required) Location for deploying the resource group. If the resource group already exists in the subscription, then this value will be ignored.

ARGUMENT	DESCRIPTION
Template location	(Required) undefined
Template link	(Required) Specify the URL of the template file. Example: https://raw.githubusercontent.com/Azure/... To deploy a template stored in a private storage account, retrieve and include the shared access signature (SAS) token in the URL of the template. Example: <code><blob_storage_url>/template.json?<SASToken></code> To upload a template file (or a linked template) to a storage account and generate a SAS token, you could use Azure file copy task or follow the steps using PowerShell or Azure CLI . To view the template parameters in a grid, click on ... next to Override template parameters text box. This feature requires that CORS rules are enabled at the source. If templates are in Azure storage blob, refer to this to enable CORS.
Template parameters link	(Optional) Specify the URL of the parameters file. Example: https://raw.githubusercontent.com/Azure/... To use a file stored in a private storage account, retrieve and include the shared access signature (SAS) token in the URL of the template. Example: <code><blob_storage_url>/template.json?<SASToken></code> To upload a parameters file to a storage account and generate a SAS token, you could use Azure file copy task or follow the steps using PowerShell or Azure CLI . To view the template parameters in a grid, click on ... next to Override template parameters text box. This feature requires that CORS rules are enabled at the source. If templates are in Azure storage blob, refer to this to enable CORS.
Template	(Required) Specify the path or a pattern pointing to the Azure Resource Manager template. For more information about the templates see https://aka.ms/azuratemplates . To get started immediately use template https://aka.ms/sampletemplate .
Template parameters	(Optional) Specify the path or a pattern pointing for the parameters file for the Azure Resource Manager template.
Override template parameters	(Optional) To view the template parameters in a grid, click on "..." next to Override Parameters textbox. This feature requires that CORS rules are enabled at the source. If templates are in Azure storage blob, refer to this to enable CORS. Or type the template parameters to override in the textbox. Example, -storageName fabrikam –adminUsername \$(vmusername) -adminPassword \$(password) –azureKeyVaultName \$(fabrikamFibre). If the parameter value you're using has multiple words, enclose them in quotes, even if you're passing them using variables. For example, -name "parameter value" -name2 "\$(var)" To override object type parameters use stringified JSON objects. For example, -options ["option1"] -map {"key1": "value1" }.

ARGUMENT	DESCRIPTION
Deployment mode	(Required) Incremental mode handles deployments as incremental updates to the resource group . It leaves unchanged resources that exist in the resource group but are not specified in the template. Complete mode deletes resources that are not in your template. Validate mode enables you to find problems with the template before creating actual resources. By default, Incremental mode is used.
Enable prerequisites	<p>(Optional) These options would be applicable only when the Resource group contains virtual machines.</p> <p>Choosing Deployment Group option would configure Deployment Group agent on each of the virtual machines.</p> <p>Selecting WinRM option configures Windows Remote Management (WinRM) listener over HTTPS protocol on port 5986, using a self-signed certificate. This configuration is required for performing deployment operation on Azure machines. If the target Virtual Machines are backed by a Load balancer, ensure Inbound NAT rules are configured for target port (5986).</p>
Azure Pipelines/TFS endpoint	<p>(Required) Agent registration with a deployment group requires access to your Visual Studio project.?</p> <p>Click "Add" to create a service connection using a personal access token (PAT) with scope restricted to "Deployment Group" and a default expiration time of 90 days.</p> <p>?Click "Manage" to update service connection details.?</p>
Project	(Required) Specify the project which has the Deployment Group defined in it?
Deployment Group	(Required) Specify the Deployment Group against which the Agent(s) will be registered. For more guidance, refer to Deployment Groups
Copy Azure VM tags to agents	<p>(Optional) Choose if the tags configured on the Azure VM need to be copied to the corresponding Deployment Group agent.</p> <p>By default all Azure tags will be copied following the format "Key: Value". Example: An Azure Tag "Role : Web" would be copied as-is to the Agent machine.</p> <p>For more information on how tag Azure resources refer to link</p>
VM details for WinRM	(Optional) Provide a name for the variable for the resource group. The variable can be used as \${variableName} to refer to the resource group in subsequent tasks like in the PowerShell on Target Machines task for deploying applications. Valid only when the selected action is Create, Update or Select, and required when an existing resource group is selected.

ARGUMENT	DESCRIPTION
Deployment outputs	(Optional) Provide a name for the variable for the output variable which will contain the outputs section of the current deployment object in string format. You can use the "ConvertFrom-Json" PowerShell cmdlet to parse the JSON object and access the individual output values.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Azure SQL Database Deployment task

11/6/2018 • 3 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to deploy to Azure SQL DB using a DACPAC or run scripts using SQLCMD.

YAML snippet

```
# Azure SQL Database Deployment
# Deploy Azure SQL DB using DACPAC or run scripts using SQLCMD
- task: SqlAzureDacpacDeployment@1
  inputs:
    #azureConnectionType: 'ConnectedServiceNameARM' # Optional. Options: connectedServiceName,
    connectedServiceNameARM
    #azureClassicSubscription: # Required when azureConnectionType == ConnectedServiceName
    #azureSubscription: # Required when azureConnectionType == ConnectedServiceNameARM
    serverName:
    databaseName:
    sqlUsername:
    sqlPassword:
    #deploymentAction: 'Publish' # Options: publish, extract, export, import, script, driftReport,
    deployReport
    #taskNameSelector: 'DacpacTask' # Optional. Options: dacpacTask, sqlTask, inlineSqlTask
    #dacpacFile: # Required when taskNameSelector == DacpacTask || DeploymentAction == Script ||
    DeploymentAction == DeployReport
    #bacpacFile: # Required when deploymentAction == Import
    #sqlFile: # Required when taskNameSelector == SqlTask
    #sqlInline: # Required when taskNameSelector == InlineSqlTask
    #publishProfile: # Optional
    #additionalArguments: # Optional
    #sqlAdditionalArguments: # Optional
    #inlineAdditionalArguments: # Optional
    #ipDetectionMethod: 'AutoDetect' # Options: autoDetect, ipAddressRange
    #startIpAddress: # Required when ipDetectionMethod == IPAddressRange
    #endIpAddress: # Required when ipDetectionMethod == IPAddressRange
    #deleteFirewallRule: true # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Azure Connection Type	(Optional)
Azure Classic Subscription	(Required) Target Azure Classic subscription for deploying SQL files
Azure Subscription	(Required) Target Azure Resource Manager subscription for deploying SQL files
Azure SQL Server Name	(Required) Azure SQL Server name, like Fabrikam.database.windows.net,1433 or Fabrikam.database.windows.net.

Argument	Description
Database Name	(Required) Name of the Azure SQL Database, where the files will be deployed.
Server Admin Login	(Optional) Specify the Azure SQL Server administrator login.
Password	(Optional) Password for the Azure SQL Server administrator. It can accept variables defined in build/release pipelines as '\$(passwordVariable)'. You may mark the variable type as 'secret' to secure it.
Type	(Optional)
DACPAC File	(Required) Location of the DACPAC file on the automation agent or on a UNC path accessible to the automation agent like, \\BudgetIT\Web\Deploy\FabrikamDB.dacpac. Predefined system variables like, \$(agent.releaseDirectory) can also be used here.
SQL Script	(Required) Location of the SQL script file on the automation agent or on a UNC path accessible to the automation agent like, \\BudgetIT\Web\Deploy\FabrikamDB.sql. Predefined system variables like, \$(agent.releaseDirectory) can also be used here.
Inline SQL Script	(Required) Enter the SQL script to execute on the Database selected above.
Publish Profile	(Optional) Publish profile provides fine-grained control over Azure SQL Database creation or upgrades. Specify the path to the Publish profile XML file on the automation agent or on a UNC share. Predefined system variables like, \$(agent.buildDirectory) or \$(agent.releaseDirectory) can also be used here.
Additional SqlPackage.exe Arguments	(Optional) Additional SqlPackage.exe arguments that will be applied when deploying the Azure SQL Database, in case DACPAC option is selected like, /p:IgnoreAnsiNulls=True /p:IgnoreComments=True. These arguments will override the settings in the Publish profile XML file (if provided).
Additional Invoke-Sqlcmd Arguments	(Optional) Additional Invoke-Sqlcmd arguments that will be applied when executing the given SQL query on the Azure SQL Database like, -ConnectionTimeout 100 -OutputSqlErrors.
Additional Invoke-Sqlcmd Arguments	(Optional) Additional Invoke-Sqlcmd arguments that will be applied when executing the given SQL query on the Azure SQL Database like, -ConnectionTimeout 100 -OutputSqlErrors
Specify Firewall Rules Using	(Required) For the task to run, the IP Address of the automation agent has to be added to the 'Allowed IP Addresses' in the Azure SQL Server's Firewall. Select auto-detect to automatically add firewall exception for range of possible IP Address of automation agent or specify the range explicitly.

ARGUMENT	DESCRIPTION
Start IP Address	(Required) The starting IP Address of the automation agent machine pool like 196.21.30.50.
End IP Address	(Required) The ending IP Address of the automation agent machine pool like 196.21.30.65.
Delete Rule After Task Ends	(Optional) If selected, then after the task ends, the IP Addresses specified here are deleted from the 'Allowed IP Addresses' list of the Azure SQL Server's Firewall.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Azure VM Scale Set Deployment task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to deploy a virtual machine scale set image.

YAML snippet

```
# Azure VM scale set Deployment
# Deploy Virtual Machine scale set image
- task: AzureVmssDeployment@0
  inputs:
    azureSubscription:
      #action: 'Update image' # Options: update Image, configure Application Startup
    vmssName:
    vmssOsType: # Options: windows, linux
    imageUrl:
    #customScriptsDirectory: # Optional
    #customScript: # Optional
    #customScriptArguments: # Optional
    #customScriptsStorageAccount: # Optional
    #skipArchivingCustomScripts: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Azure subscription	(Required) Select the Azure Resource Manager subscription for the scale set.
Action	(Required) Choose between updating a VM scale set by using a VHD image and/or by running deployment/install scripts using Custom Script VM extension. The VHD image approach is better for scaling quickly and doing rollback. The extension approach is useful for post deployment configuration, software installation, or any other configuration / management task. You can use a VHD image to update a VM scale set only when it was created by using a custom image, the update will fail if the VM Scale set was created by using a platform/gallery image available in Azure. The Custom script VM extension approach can be used for VM scale set created by using either custom image or platform/gallery image.
Virtual Machine scale set name	(Required) Name of VM scale set which you want to update by using either a VHD image or by using Custom script VM extension.
OS type	(Required) Select the operating system type of VM scale set.

ARGUMENT	DESCRIPTION
Image url	(Required) Specify the URL of VHD image. If it is an Azure storage blob url, the storage account location should be same as scale set location.
Custom script directory	(Optional) Path to directory containing custom script(s) that will be run by using Custom Script VM extension. The extension approach is useful for post deployment configuration, application/software installation, or any other application configuration/management task. For example: the script can set a machine level stage variable which the application uses, like database connection string.
Command	(Optional) The script that will be run by using Custom Script VM extension. This script can invoke other scripts in the directory. The script will be invoked with arguments passed below. This script in conjugation with such arguments can be used to execute commands. For example: 1. Update-DatabaseConnectionStrings.ps1 -clusterType dev -user \$(dbUser) -password \$(dbUserPwd) will update connection string in web.config of web application. 2. install-secrets.sh --key-vault-type prod -key serviceprincipalkey will create an encrypted file containing service principal key.
Arguments	(Optional) The custom script will be invoked with arguments passed. Build/Release variables can be used which makes it easy to use secrets.
Azure storage account where custom scripts will be uploaded	(Optional) The Custom Script Extension downloads and executes scripts provided by you on each virtual machines in the VM scale set. These scripts will be stored in the storage account specified here. Specify a pre-existing ARM storage account.
Skip Archiving custom scripts	(Optional) By default, this task creates a compressed archive of directory containing custom scripts. This improves performance and reliability while uploading to azure storage. If not selected, archiving will not be done and all files will be individually uploaded.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Build Machine Image task

11/19/2018 • 3 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to build a machine image using Packer. This image can be used for Azure Virtual machine scale set deployment.

YAML snippet

```
# Build Machine Image
# Build machine image using Packer. This image can be used for Azure Virtual machine scale set deployment
- task: PackerBuild@1
  inputs:
    #templateType: 'builtin' # Options: builtin, custom
    #customTemplateLocation: # Required when templateType == Custom
    #customTemplateParameters: '{}' # Optional
    connectedServiceName:
    isManagedImage:
    #managedImageName: # Required when isManagedImage == True
    location:
    storageAccountName:
    azureResourceGroup:
    #baseImageSource: 'default' # Options: default, customVhd
    #baseImage: 'MicrosoftWindowsServer:WindowsServer:2012-R2-Datacenter:windows' # Required when
    baseImageSource == Default# Options: microsoftWindowsServer:WindowsServer:2012-R2-Datacenter:Windows,
    microsoftWindowsServer:WindowsServer:2016-Datacenter:Windows, microsoftWindowsServer:WindowsServer:2012-
    Datacenter:Windows, microsoftWindowsServer:WindowsServer:2008-R2-SP1:Windows, canonical:UbuntuServer:14.04.4-
    LTS:Linux, canonical:UbuntuServer:16.04-LTS:Linux, redHat:RHEL:7.2:Linux, redHat:RHEL:6.8:Linux,
    openLogic:CentOS:7.2:Linux, openLogic:CentOS:6.8:Linux, credativ:Debian:8:Linux, credativ:Debian:7:Linux,
    sUSE:OpenSUSE-Leap:42.2:Linux, sUSE:SLES:12-SP2:Linux, sUSE:SLES:11-SP4:Linux
    #customImageUrl: # Required when baseImageSource == CustomVhd
    #customImageOSType: 'windows' # Required when baseImageSource == CustomVhd# Options: windows, linux
    packagePath:
    deployScriptPath:
    #deployScriptArguments: # Optional
    #additionalBuilderParameters: '{}' # Optional
    #skipTempFileCleanupDuringVMDeprovision: # Optional
    #imageUri: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Packer template	(Required) Select whether you want the task to auto generate Packer template or use custom template provided by you.
Packer template location	(Required) Path to a custom user-provided template.

ARGUMENT	DESCRIPTION
Template parameters	(Optional) Specify parameters which will be passed to Packer for building custom template. This should map to "variables" section in your custom template. E.g. if the template has a variable named "drop-location", then add a parameter here with name "drop-location" and a value which you want to use. You can link the value to a release variable as well. To view/edit the additional parameters in a grid, click on "..." next to text box.
Azure subscription	(Required) Select the Azure Resource Manager subscription for baking and storing the machine image.
Storage location	(Required) Location for storing the built machine image. This location will also be used to create a temporary VM for the purpose of building image.
Storage account	(Required) Storage account for storing the built machine image. This storage account must be pre-existing in the location selected.
Resource group	(Required) Azure Resource group that contains the selected storage account.
Base image source	(Required) Select the source of base image. You can either choose from a curated gallery of OS images or provide url of your custom image.
Base image	(Required) Choose from curated list of OS images. This will be used for installing pre-requisite(s) and application(s) before capturing machine image.
Base image URL	(Required) Specify url of base image. This will be used for installing pre-requisite(s) and application(s) before capturing machine image.
Base image OS	(Required) undefined
Deployment Package	(Required) Specify the path for deployment package directory relative to \$(System.DefaultWorkingDirectory). Supports minimatch pattern. Example path: FrontendWebApp//GalleryApp
Deployment script	(Required) Specify the relative path to powershell script(for Windows) or shell script(for Linux) which deploys the package. This script should be contained in the package path selected above. Supports minimatch pattern. Example path: deploy//scripts/windows/deploy.ps1
Deployment script arguments	(Optional) Specify the arguments to be passed to deployment script.

ARGUMENT	DESCRIPTION
Additional Builder parameters	(Optional) In auto generated Packer template mode the task creates a Packer template with an Azure builder. This builder is used to generate a machine image. You can add keys to the Azure builder to customize the generated Packer template. For example setting ssh_tty=true in case you are using a CentOS base image and you need to have a tty to run sudo. To view/edit the additional parameters in a grid, click on "..." next to text box.
Skip temporary file cleanup during deprovision	(Optional) During deprovisioning of VM, skip clean-up of temporary files uploaded to VM. Refer here
Image URL	(Optional) Provide a name for the output variable which will store generated machine image url.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Chef task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to deploy to Chef environments by editing environment attributes.

YAML snippet

```
# Chef
# Deploy to Chef environments by editing environment attributes
- task: Chef@1
  inputs:
    connectedServiceName:
    environment:
    attributes:
    #chefWaitTime: '30'
```

Arguments

ARGUMENT	DESCRIPTION
Chef Connection	(Required) Name of the Chef subscription
Environment	(Required) Name of the Chef Environment to be used for Deployment. The attributes of that environment will be edited.
Environment Attributes	(Required) Specify the value of the leaf node attribute(s) to be updated. Example. { "default_attributes.connectionString": "\$(connectionString)", "override_attributes.buildLocation": " https://sample.blob.core.windows.net/build " }. Task fails if the leaf node does not exist.
Wait Time	(Required) The amount of time (in minutes) to wait for this task to complete. Default value: 30 minutes
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Chef Knife task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to run scripts with Knife commands on your Chef workstation.

YAML snippet

```
# Chef Knife
# Run Scripts with knife commands on your chef workstation
- task: ChefKnife@1
  inputs:
    connectedServiceName:
    scriptPath:
    #scriptArguments: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Chef Subscription	(Required) Chef subscription to configure before running knife commands
Script Path	(Required) Path of the script. Should be fully qualified path or relative to the default working directory.
Script Arguments	(Optional) Additional parameters to pass to Script. Can be either ordinal or named parameters.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Copy Files Over SSH task

11/15/2018 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

Use this task in a build or release pipeline to copy files from a source folder to a target folder on a remote machine over SSH.

This task allows you to connect to a remote machine using SSH and copy files matching a set of minimatch patterns from specified source folder to target folder on the remote machine. Supported protocols for file transfer are SFTP and SCP via SFTP. In addition to Linux, macOS is partially supported (see [Q&A](#)).

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

Prerequisites

- The task supports use of an SSH key pair to connect to the remote machine(s).
- The public key must be pre-installed or copied to the remote machine(s).

YAML snippet

```
# Copy Files Over SSH
# Copy files or build artifacts to a remote machine over SSH
- task: CopyFilesOverSSH@0
  inputs:
    sshEndpoint:
    #sourceFolder: # Optional
    #contents: '**'
    #targetFolder: # Optional
    #cleanTargetFolder: false # Optional
    #overwrite: true # Optional
    #failOnEmptySource: false # Optional
    #flattenFolders: false # Optional
```

Arguments

ARGUMENT	DESCRIPTION
SSH endpoint	The name of an SSH service connection containing connection details for the remote machine. - The hostname or IP address of the remote machine, the port number, and the user name are required to create an SSH service connection. - The private key and the passphrase must be specified for authentication.

ARGUMENT	DESCRIPTION
Source folder	The source folder for the files to copy to the remote machine. If omitted, the root of the repository is used. Names containing wildcards such as <code>*.zip</code> are not supported. Use variables if files are not in the repository. Example: <code>\$(Agent.BuildDirectory)</code>
Contents	File paths to include as part of the copy. Supports multiple lines of minimatch patterns . Default is <code>**</code> which includes all files (including sub folders) under the source folder. - Example: <code>**/*.jar \n **/*.war</code> includes all jar and war files (including sub folders) under the source folder. - Example: <code>** \n !**/*.xml</code> includes all files (including sub folders) under the source folder but excludes xml files.
Target folder	Target folder on the remote machine to where files will be copied. Example: <code>/home/user/MySite</code> . Preface with a tilde (~) to specify the user's home directory.
Advanced - Clean target folder	If this option is selected, all existing files in the target folder will be deleted before copying.
Advanced - Overwrite	If this option is selected (the default), existing files in the target folder will be replaced.
Advanced - Flatten folders	If this option is selected, the folder structure is not preserved and all the files will be copied into the specified target folder on the remote machine.
Control options	See Control options

See also

- [Install SSH Key task](#)
- [SSH task](#)
- Blog post [SSH build task](#)

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

Is this task supported for target machines running operating systems other than Linux?

This task is intended for target machines running Linux.

- For copying files to a macOS machine, this task may be used, but authenticating with a password is not

supported.

- For copying files to a Windows machine, consider using [Windows Machine File Copy](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Docker task

11/19/2018 • 7 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018

Use this task in a build or release pipeline to build, tag, push, or run Docker images, or run a Docker command. This task can be used with Docker or Azure Container registry.

NOTE

For YAML pipelines, consider using script-based Docker commands as described in the [Docker guidance](#), and using this Docker task when working with container registries that require authentication.

The built-in Docker task enables you to build Docker images, push Docker images to an authenticated Docker registry, run Docker images, or execute other operations offered by the Docker CLI:

- **Use Docker best practices:** By writing minimal yaml you can build and push an image which is tagged with '\$(Build.BuildId)' and has rich metadata about the repository, commit, build information to the container image as Docker labels
- **Conform to Docker standards:** The task takes care of details like tagging image with the registry hostname and port image before pushing the image to a private registry like Azure Container Registry (ACR). It also helps you to follow Docker naming convention, for example, converting upper case character to lower case and removes spaces in image name which can happen if you are using \$(Build.Repository.Name) to name your images.
- **Manage secrets:** The task makes it easy to use either 'Docker registry service connection' for connecting to any private container registry or 'Azure Service Connection'. For connecting to ACR. For example, in case of ACR you don't have to enable 'admin user' and manage username and password as secret. The task will use the Azure Service connection to login to ACR. Once you have used the Docker task to sign in, the session is maintained for the duration of the job thus allowing you to use follow-up tasks to execute any scripts by leveraging the login done by the Docker task. For example, You can use the Docker task to sign into ACR and then use a subsequent script to pull an image and scan the container image for vulnerabilities.

YAML snippet

Build Docker images

Build a Dockerfile into an image with a registry-qualified name and multiple tags such as the build ID, source branch name and Git tags:

```
- task: Docker@1
  displayName: 'Build an image'
  inputs:
    azureSubscriptionEndpoint: 'ContosoAzureSubscription'
    azureContainerRegistry: contoso.azurecr.io
```

For other private container registries

```
- task: Docker@1
displayName: 'Build an image'
inputs:
  containerregistrytype: 'Container Registry'
  dockerRegistryEndpoint: Contoso
```

'azureSubscriptionEndpoint' input is the name of Azure Service Connection. See [Azure Resource Manager service connection](#) to manually set up the connection. 'dockerRegistryEndpoint' input is the name of [Docker Registry service connection](#).

This will result in a docker login to the container registry by using the service connection and then a docker build command will be used to build and tag the image. For example, a simplified version of the command run is:

```
docker build -t contoso.azurecr.io/contoso-ci:11 .
```

By writing minimal yaml you can build and push an image which is tagged with '\$(Build.BuildId)' and has rich metadata about the repository, commit, build information to the container image as Docker labels. The task takes care of details like tagging image with the registry hostname and port image before pushing the image to a private registry like Azure Container Registry (ACR). It also helps you to follow Docker naming convention, for example, converting upper case character to lower case and removes spaces in image name which can happen if you are using \$(Build.Repository.Name) to name your images.

Push Docker images

Push Docker images with multiple tags to an authenticated Docker Registry and save the resulting repository image digest to a file:

```
- task: Docker@1
displayName: 'Push an image'
inputs:
  azureSubscriptionEndpoint: 'ContosoAzureSubscription'
  azureContainerRegistry: contoso.azurecr.io
  command: 'push'
```

This will result in a docker login to the container registry by using the service connection and then a docker push command will be used to push the image to the container registry. For example, a simplified version of the command run is:

```
docker push contoso.azurecr.io/contoso-ci:11
```

Build, tag and push container image

Here is an end to end sample yaml for building, tagging and pushing container image.

```

- task: Docker@1
  displayName: 'Build an image'
  inputs:
    imageName: 'contoso.azurecr.io/repositoryname:${Build.BuildId}'
- task: Docker@1
  displayName: Login
  inputs:
    azureSubscriptionEndpoint: 'ContosoAzureSubscription'
    azureContainerRegistry: contoso.azurecr.io
    command: login
- task: Docker@1
  displayName: 'Push an image'
  inputs:
    command: 'push'
    imageName: 'contoso.azurecr.io/$repositoryname:${Build.BuildId}'
```

Login to a container registry and run scripts

Task makes it easy to use either 'Docker registry service connection' for connecting to any private container registry or 'Azure Service Connection' For connecting to ACR. For example, in the case of ACR you don't have to enable 'admin user' and manage username and password as secret. The task will use the Azure Service connection to login to ACR. Once you have used the task to login, the session is maintained for the duration of the job thus allowing you to use follow-up tasks to execute any scripts by leveraging the login done by the Docker task. For example, You can use the Docker task to sign into ACR and then use a subsequent script to pull an image and scan the container image for vulnerabilities.

```

- task: Docker@1
  displayName: Login
  inputs:
    azureSubscriptionEndpoint: 'ContosoAzureSubscription'
    azureContainerRegistry: contoso.azurecr.io
    command: login
- bash: |
  # Write your commands here
  # Use the environment variables input below to pass secret variables to this script
  docker build -t contoso.azurecr.io/repositoryname:${Build.BuildId} . # include other options to meet
  your needs
  docker push contoso.azurecr.io/repositoryname:${Build.BuildId}
  displayName: 'Build, tag and push image'
```

Run Docker images

Perform isolated workloads inside a container by running a Docker image. A Docker image can also be run in the background with a specific restart policy.

```

- task: Docker@1
  displayName: 'Push an image'
  inputs:
    azureSubscriptionEndpoint: 'ContosoAzureSubscription'
    azureContainerRegistry: contoso.azurecr.io
    command: 'run'
    containerName: contosocontainer
    ports: 8084
    volumes: '$(System.DefaultWorkingDirectory):/src'
    workingDirectory: /src
    containerCommand: 'npm install'
    restartPolicy: onFailure
```

This will result in a docker run command. For example:

```
docker run -d --restart no atul-aks:1382
```

Arguments

ARGUMENT	DESCRIPTION
Container Registry Type	(Required) Select a Container Registry Type.
Docker Registry Connection	(Optional) Select a Docker registry connection. Required for commands that need to authenticate with a registry.
Azure subscription	(Optional) Select an Azure subscription
Azure Container Registry	(Optional) Select an Azure Container Registry
Action	(Required) Select a Docker action.
Docker File	(Required) Path to the Docker file to use. Must be within the Docker build context.
Build Arguments	(Optional) Build-time variables for the Docker file. Specify each name=value pair on a new line.
Use Default Build Context	(Optional) Set the build context to the directory that contains the Docker file.
Build Context	(Optional) Path to the build context.
Image Name	(Required) Name of the Docker image to build, push, or run.
Image Names Path	(Required) Path to a text file that contains the names of the Docker images to tag or push. Each image name is contained on its own line.
Qualify Image Name	(Optional) Qualify the image name with the Docker registry connection's hostname if not otherwise specified.
Additional Image Tags	(Optional) Additional tags for the Docker image being built or pushed.
Include Source Tags	(Optional) Include Git tags when building or pushing the Docker image.
Include Latest Tag	(Optional) Include the 'latest' tag when building or pushing the Docker image.
Image Digest File	(Optional) Path to a file that is created and populated with the full image repository digest of the Docker image that was pushed.
Container Name	(Optional) Name of the Docker container to run.
Ports	(Optional) Ports in the Docker container to publish to the host. Specify each host-port:container-port binding on a new line.

ARGUMENT	DESCRIPTION
Volumes	(Optional) Volumes to mount from the host. Specify each host-dir:container-dir on a new line.
Environment Variables	(Optional) Environment variables for the Docker container. Specify each name=value pair on a new line.
Working Directory	(Optional) The working directory for the Docker container.
Entrypoint Override	(Optional) Override the default entrypoint for the Docker container.
Command	(Optional) Command to run in the Docker container. For example, if the image contains a simple Python Flask web application you can specify 'python app.py' to launch the web application.
Run In Background	(Optional) Run the Docker container in the background.
Restart Policy	(Required) Select a restart policy.
Maximum Restart Retries	(Optional) The maximum number of restart retries the Docker daemon attempts.
Command	(Required) Docker command to execute, with arguments. For example, 'rmi -f image-name' to force remove an image.
Docker Host Connection	(Optional) Select a Docker host connection. Defaults to the agent's host.
Force image name to follow Docker naming convention	(Optional) If enabled docker image name will be modified to follow Docker naming convention. Converts upper case character to lower case and removes spaces in image name.
Working Directory	(Optional) Working directory for the Docker command.
Memory limit	(Optional) The maximum amount of memory available to the container as a integer with optional suffixes like '2GB'.
CONTROL OPTIONS	

Open source

This task is open source on [GitHub](#). Feedback and contributions are welcome.

Q & A

Docker Compose task

11/6/2018 • 3 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to build, push or run multi-container Docker applications. This task can be used with Docker or Azure Container registry.

YAML snippet

```
# Docker Compose
# Build, push or run multi-container Docker applications. Task can be used with Docker or Azure Container
registry.
- task: DockerCompose@0
  inputs:
    #containerregistrytype: 'Azure Container Registry' # Options: azure Container Registry, container Registry
    #dockerRegistryEndpoint: # Optional
    #azureSubscription: # Optional
    #azureContainerRegistry: # Optional
    #dockerComposeFile: '**/docker-compose.yml'
    #additionalDockerComposeFiles: # Optional
    #dockerComposeFileArgs: # Optional
    # projectName: '$(Build.Repository.Name)' # Optional
    #qualifyImageNames: true # Optional
    #action: 'Run a Docker Compose command' # Options: build Services, push Services, run Services, run A
Specific Service, lock Services, write Service Image Digests, combine Configuration, run A Docker Compose
Command
    #additionalImageTags: # Optional
    #includeSourceTags: false # Optional
    #includeLatestTag: false # Optional
    #buildImages: true # Optional
    #serviceName: # Required when action == Run A Specific Service
    #containerName: # Optional
    #ports: # Optional
    #workingDirectory: # Optional
    #entrypoint: # Optional
    #containerCommand: # Optional
    #detached: true # Optional
    #abortOnContainerExit: true # Optional
    #imageDigestComposeFile: '$(Build.StagingDirectory)/docker-compose.images.yml' # Required when action ==
Write Service Image Digests
    #removeBuildOptions: false # Optional
    #baseResolveDirectory: # Optional
    #outputDockerComposeFile: '$(Build.StagingDirectory)/docker-compose.yml' # Required when action == Lock
Services || Action == Combine Configuration
    #dockerComposeCommand: # Required when action == Run A Docker Compose Command
    #dockerHostEndpoint: # Optional
    #nopIfNoDockerComposeFile: false # Optional
    #requireAdditionalDockerComposeFiles: false # Optional
    #currentWorkingDirectory: '$(System.DefaultWorkingDirectory)' # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Container Registry Type	(Required) Select a Container Registry Type.

ARGUMENT	DESCRIPTION
Docker Registry Connection	(Optional) Select a Docker registry connection. Required for commands that need to authenticate with a registry.
Azure subscription	(Optional) Select an Azure subscription
Azure Container Registry	(Optional) Select an Azure Container Registry
Docker Compose File	(Required) Path to the primary Docker Compose file to use.
Additional Docker Compose Files	(Optional) Additional Docker Compose files to be combined with the primary Docker Compose file. Relative paths are resolved relative to the directory containing the primary Docker Compose file. If a specified file is not found, it is ignored. Specify each file path on a new line.
Environment Variables	(Optional) Environment variables to be set during the command. Specify each name=value pair on a new line.
Project Name	(Optional) Project name used for default naming of images and containers.
Qualify Image Names	(Optional) Qualify image names for built services with the Docker registry connection's hostname if not otherwise specified.
Action	(Required) Select a Docker Compose action.
Additional Image Tags	(Optional) Additional tags for the Docker images being built or pushed.
Include Source Tags	(Optional) Include Git tags when building or pushing Docker images.
Include Latest Tag	(Optional) Include the 'latest' tag when building or pushing Docker images.
Build Images	(Optional) Build images before starting service containers.
Service Name	(Required) Name of the specific service to run.
Container Name	(Optional) Name of the specific service container to run.
Ports	(Optional) Ports in the specific service container to publish to the host. Specify each host-port:container-port binding on a new line.
Working Directory	(Optional) The working directory for the specific service container.
Entrypoint Override	(Optional) Override the default entry point for the specific service container.

ARGUMENT	DESCRIPTION
Command	(Optional) Command to run in the specific service container. For example, if the image contains a simple Python Flask web application you can specify 'python app.py' to launch the web application.
Run In Background	(Optional) Run the service containers in the background.
Abort on Container Exit	(Optional) Stop all containers when any container exits.
Image Digest Compose File	(Required) Path to a Docker Compose file that is created and populated with the full image repository digests of each service's Docker image.
Remove Build Options	(Optional) Remove the build options from the output Docker Compose file.
Base Resolve Directory	(Optional) The base directory from which relative paths in the output Docker Compose file should be resolved.
Output Docker Compose File	(Required) Path to an output Docker Compose file.
Command	(Required) Docker Compose command to execute with arguments. For example, 'rm --all' to remove all stopped service containers.
Docker Host Connection	(Optional) Select a Docker host connection. Defaults to the agent's host.
No-op if no Docker Compose File	(Optional) If the Docker Compose file does not exist, skip this step. This is useful when the step offers optional behavior based on the existence of a Docker Compose file in the repository.
Require Additional Docker Compose Files	(Optional) Produces an error if the additional Docker Compose files do not exist. This overrides the default behavior which is to ignore a file if it does not exist.
Working Directory	(Optional) Working directory for the Docker Compose command.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Package and Deploy Helm Charts task

11/6/2018 • 5 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to deploy, configure, or update a Kubernetes cluster in Azure Container Service by running Helm commands. Helm is a tool that streamlines deploying and managing Kubernetes apps using a packaging format called [charts](#). You can define, version, share, install, and upgrade even the most complex Kubernetes app by using Helm.

- Helm helps you combine multiple Kubernetes manifests (yaml) such as service, deployments, configmaps, and more into a single unit called Helm Charts. You don't need to either invent or use a tokenization or a templating tool.
- Helm Charts also help you manage application dependencies and deploy as well as rollback as a unit. They are also easy to create, version, publish, and share with other partner teams.

Azure Pipelines has built-in support for Helm charts:

- The [Helm Tool installer task](#) can be used to get the correct version of Helm onto the agents.
- The Helm package and deploy task can be used to package the app and deploy it to a Kubernetes cluster. You can use the task to install or update Tiller to a Kubernetes namespace, to securely connect to Tiller over TLS for deploying charts, or to run any Helm command such as lint.
- The Helm task also supports connecting to an Azure Kubernetes Service by using an Azure service connection. You can connect to any Kubernetes cluster by using kubeconfig or a service account.
- Helm deployments can be supplemented by using the Kubectl task; for example, create/update, imagepullsecret.

YAML snippet

```

# Package and deploy Helm charts
# Deploy, configure, update your Kubernetes cluster in Azure Container Service by running helm commands.
- task: HelmDeploy@0
  inputs:
    #connectionType: 'Azure Resource Manager' # Options: azure Resource Manager, kubernetes Service Connection, none
    #azureSubscription: # Required when connectionType == Azure Resource Manager
    #azureResourceGroup: # Required when connectionType == Azure Resource Manager
    #kubernetesCluster: # Required when connectionType == Azure Resource Manager
    #kubernetesServiceConnection: # Required when connectionType == Kubernetes Service Connection
    #namespace: # Optional
    #command: 'ls' # Options: create, delete, expose, get, init, install, login, logout, ls, package, rollback, upgrade
    #chartType: 'Name' # Required when command == Install || Command == Upgrade# Options: name, filePath
    #chartName: # Required when chartType == Name
    #chartPath: # Required when chartType == FilePath || Command == Package
    #chartVersion: # Optional
    #releaseName: # Optional
    #overrideValues: # Optional
    #valueFile: # Optional
    #destination: '${Build.ArtifactStagingDirectory}' # Optional
    #canaryImage: false # Optional
    #upgradeTiller: true # Optional
    #updateDependency: false # Optional
    #save: true # Optional
    #install: true # Optional
    #recreate: false # Optional
    #resetValues: false # Optional
    #force: false # Optional
    #waitForExecution: true # Optional
    #arguments: # Optional
    #enableTls: false # Optional
    #caCert: # Required when enableTls == True
    #certificate: # Required when enableTls == True
    #privatekey: # Required when enableTls == True
    #tillerNamespace: # Optional

```

Arguments

ARGUMENT	DESCRIPTION
Connection Type	(Required) Select 'Azure Resource Manager' to connect to an Azure Kubernetes Service(AKS) cluster by using Azure Service Connection. Select 'Container registry' to connect to any Kubernetes cluster by using kubeconfig or Service Account.
Azure subscription	(Required) Select an Azure subscription, which has your Azure Container Registry.
Resource group	(Required) Enter or select the resource group of your AKS cluster.
Kubernetes cluster	(Required) Select an Azure Managed AKS Cluster.
Kubernetes Service Connection	(Required) Select a Kubernetes service connection.

ARGUMENT	DESCRIPTION
Namespace	(Optional) Specify the Kubernetes cluster namespace where you want to deploy your application. Kubernetes supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called namespaces. You can use Namespace to create different environments like dev/test/staging in the same cluster. Use Tiller namespace in advance section to specify tiller namespace.
Command	(Required) Select a helm command.
Chart Type	(Required) Select how you want to enter chart info. You can either provide name of the chart or folder/file path to the chart.
Chart Name	(Required) Name of the chart to install or upgrade.
Chart Path	(Required) Path to the chart to install or upgrade. Chart path can be a path to a packaged chart or a path to an unpacked chart directory. For example if './redis' is specified the task will run 'helm package ./redis'.
Version	(Optional) Specify the exact chart version to install. If this is not specified, the latest version is installed. Set the version on the chart to this semver version
Release Name	(Optional) Release name. If unspecified, it will autogenerate one for you. Cannot be used with the delete command.
Set Values	(Optional) Set values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2). In the Helm chart you can parameterize the container image details like name and tag because the same Helm chart can be used for deploying to different environments. These values can also be specified in the values.yaml file of the chart or be overridden by a user-supplied values file, which can in turn be overridden by --set parameters during helm install or helm upgrade.
Value File	(Optional) Specify values in a YAML file or a URL.
Destination	(Optional) Specify values in a YAML file or a URL.
Use canary image version.	(Optional) Use the canary Tiller image. Will install the latest pre-release version of Tiller.
Upgrade Tiller	(Optional) Upgrade if Tiller is already installed.
Update Dependency	(Optional) Run helm dependency update before installing the chart. Update dependencies from 'requirements.yaml' to dir 'charts/' before packaging
Save	(Optional) Save packaged chart to local chart repository (default true)

ARGUMENT	DESCRIPTION
Install if release not present.	(Optional) If a release by this name doesn't already exist, run an install.
Recreate Pods.	(Optional) Performs pods restart for the resource if applicable.
Reset Values.	(Optional) Reset the values to the ones built into the chart.
Force	(Optional) Force resource update through delete/recreate if you want to upgrade and rollback when there are any conflicts. This is useful in scenarios where applying patches can fail (e.g., for services, because clusterIP is immutable).
Wait	(Optional) Block till command execution completes.
Arguments	(Optional) Command arguments.
Enable TLS	(Optional) Enables using SSL between Helm and Tiller.
CA certificate	(Required) CA cert used to issue certificate for tiller and helm client.
Certificate	(Required) Specify Tiller certificate or Helm client certificate
Key	(Required) Specify Tiller Key or Helm client key
Tiller namespace	(Optional) Specify K8 namespace of tiller.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

IIS Web App Deploy task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to deploy a website or web app using WebDeploy.

YAML snippet

```
# IIS Web App Deploy
# Deploy a website or web application using Web Deploy
- task: IISWebAppDeploymentOnMachineGroup@0
  inputs:
    webSiteName:
    #virtualApplication: # Optional
    #package: '$(System.DefaultWorkingDirectory)\\**\\*.zip'
    #setParametersFile: # Optional
    #removeAdditionalFilesFlag: false # Optional
    #excludeFilesFromAppDataFlag: false # Optional
    #takeAppOfflineFlag: false # Optional
    #additionalArguments: # Optional
    #xmlTransformation: # Optional
    #xmlVariableSubstitution: # Optional
    #JSONFiles: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Website Name	(Required) Provide the name of an existing website on the machine group machines
Virtual Application	(Optional) Specify the name of an already existing Virtual Application on the target Machines
Package or Folder	(Required) File path to the package or a folder generated by MSBuild or a compressed archive file. Variables (Build Release), wild cards are supported. For example, \$(System.DefaultWorkingDirectory)***.zip.
SetParameters File	(Optional) Optional: location of the SetParameters.xml file to use.
Remove Additional Files at Destination	(Optional) Select the option to delete files on the Web App that have no matching files in the Web zip package.
Exclude Files from the App_Data Folder	(Optional) Select the option to prevent files in the App_Data folder from being deployed to the Web App.
Take App Offline	(Optional) Select the option to take the Web App offline by placing an app_offline.htm file in the root directory of the Web App before the sync operation begins. The file will be removed after the sync operation completes successfully.

ARGUMENT	DESCRIPTION
Additional Arguments	(Optional) Additional Web Deploy arguments that will be applied when deploying the Azure Web App like,- disableLink:AppPoolExtension -disableLink:ContentExtension.
XML transformation	(Optional) The config transforms will be run for <code>*.Release.config</code> and <code>*.<stageName>.config</code> on the <code>*.config file</code> . Config transforms will be run prior to the Variable Substitution. XML transformations are supported only for Windows platform.
XML variable substitution	(Optional) Variables defined in the Build or Release Pipeline will be matched against the 'key' or 'name' entries in the appSettings, applicationSettings, and connectionStrings sections of any config file and parameters.xml. Variable Substitution is run after config transforms. Note: If the same variables are defined in the release pipeline and in the stage, the stage variables will supersede the Release Pipeline variables.
JSON variable substitution	(Optional) Provide new line separated list of JSON files to substitute the variable values. File names are to be provided relative to the root folder. To substitute JSON variables that are nested or hierarchical, specify them using JSONPath expressions. For example, to replace the value of 'ConnectionString' in the sample below, you need to define a variable as 'Data.DefaultConnection.ConnectionString' in the build/release pipeline (or release pipeline's stage). <pre>{ "Data": { "DefaultConnection": { "ConnectionString": "Server=(localdb)\SQLEXPRESS;Database=MyDB;Trusted_Connection=True" } } }</pre> Variable Substitution is run after configuration transforms. Note: Build/release pipeline variables are excluded in substitution

CONTROL OPTIONS

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

IIS Web App Manage task

11/6/2018 • 9 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to create or update a Website, Web App, Virtual Directory, or Application Pool.

YAML snippet

```
# IIS Web App Manage
# Create or update a Website, Web App, Virtual Directories, and Application Pool
- task: IISWebAppManagementOnMachineGroup@0
  inputs:
    #enableIIS: false # Optional
    #iISDeploymentType: 'IISWebsite' # Options: iISWebsite, iISWebApplication, iISVirtualDirectory, iSApplicationPool
    #actionIISWebsite: 'CreateOrUpdateWebsite' # Required when iISDeploymentType == IISWebsite# Options: createOrUpdateWebsite, startWebsite, stopWebsite
    #actionIISApplicationPool: 'CreateOrUpdateAppPool' # Required when iISDeploymentType == IISApplicationPool# Options: createOrUpdateAppPool, startAppPool, stopAppPool, recycleAppPool
    #startStopWebsiteName: # Required when actionIISWebsite == StartWebsite || ActionIISWebsite == StopWebsite
    websiteName:
    #websitePhysicalPath: '%SystemDrive%\inetpub\wwwroot'
    #websitePhysicalPathAuth: 'WebsiteUserPassThrough' # Options: websiteUserPassThrough, websiteWindowsAuth
    #websiteAuthUserName: # Required when websitePhysicalPathAuth == WebsiteWindowsAuth
    #websiteAuthUserPassword: # Optional
    #addBinding: false # Optional
    #protocol: 'http' # Required when iISDeploymentType == RandomDeployment# Options: https, http
    #iPAddress: 'All Unassigned' # Required when iISDeploymentType == RandomDeployment
    #port: '80' # Required when iISDeploymentType == RandomDeployment
    #serverNameIndication: false # Optional
    #hostNameWithOutSNI: # Optional
    #hostNameWithHttp: # Optional
    #hostNameWithSNI: # Required when iISDeploymentType == RandomDeployment
    #sSLCertThumbPrint: # Required when iISDeploymentType == RandomDeployment
    bindings:
      #createOrUpdateAppPoolForWebsite: false # Optional
      #configureAuthenticationForWebsite: false # Optional
      appPoolNameForWebsite:
        #dotNetVersionForWebsite: 'v4.0' # Options: v4.0, v2.0, no Managed Code
        #pipeLineModeForWebsite: 'Integrated' # Options: integrated, classic
        #appPoolIdentityForWebsite: 'ApplicationPoolIdentity' # Options: applicationPoolIdentity, localService, localSystem, networkService, specificUser
        #appPoolUsernameForWebsite: # Required when appPoolIdentityForWebsite == SpecificUser
        #appPoolPasswordForWebsite: # Optional
        #anonymousAuthenticationForWebsite: false # Optional
        #basicAuthenticationForWebsite: false # Optional
        #windowsAuthenticationForWebsite: true # Optional
      parentWebsiteNameForVD:
      virtualPathForVD:
        #physicalPathForVD: '%SystemDrive%\inetpub\wwwroot'
        #vDPhysicalPathAuth: 'VDSUserPassThrough' # Optional. Options: vDUserPassThrough, vDWindowsAuth
        #vDAuthUserName: # Required when vDPhysicalPathAuth == VDWindowsAuth
        #vDAuthUserPassword: # Optional
      parentWebsiteNameForApplication:
      virtualPathForApplication:
        #physicalPathForApplication: '%SystemDrive%\inetpub\wwwroot'
        #applicationPhysicalPathAuth: 'ApplicationUserPassThrough' # Optional. Options: applicationUserPassThrough, applicationWindowsAuth
        #applicationAuthUserName: # Required when applicationPhysicalPathAuth == ApplicationWindowsAuth
```

```

#applicationAuthUserPassword: # Optional
#createOrUpdateAppPoolForApplication: false # Optional
appPoolNameForApplication:
#dotNetVersionForApplication: 'v4.0' # Options: v4.0, v2.0, no Managed Code
#pipeLineModeForApplication: 'Integrated' # Options: integrated, classic
#appPoolIdentityForApplication: 'ApplicationPoolIdentity' # Options: applicationPoolIdentity,
localService, localSystem, networkService, specificUser
#appPoolUsernameForApplication: # Required when appPoolIdentityForApplication == SpecificUser
#appPoolPasswordForApplication: # Optional
appPoolName:
#dotNetVersion: 'v4.0' # Options: v4.0, v2.0, no Managed Code
#pipeLineMode: 'Integrated' # Options: integrated, classic
#appPoolIdentity: 'ApplicationPoolIdentity' # Options: applicationPoolIdentity, localService, localSystem,
networkService, specificUser
#appPoolUsername: # Required when appPoolIdentity == SpecificUser
#appPoolPassword: # Optional
#startStopRecycleAppPoolName: # Required when actionIISApplicationPool == StartAppPool ||
ActionIISApplicationPool == StopAppPool || ActionIISApplicationPool == RecycleAppPool
#appCmdCommands: # Optional

```

Arguments

ARGUMENT	DESCRIPTION
Enable IIS	(Optional) Check this if you want to install IIS on the machine.
Configuration type	(Required) You can create or update sites, applications, virtual directories, and application pools.
Action	(Required) Select the appropriate action that you want to perform on an IIS website. "Create Or Update" will create a website or update an existing website. Start, Stop will start or stop the website respectively.
Action	(Required) Select the appropriate action that you want to perform on an IIS Application Pool. "Create Or Update" will create app-pool or update an existing one. Start, Stop, Recycle will start, stop or recycle the application pool respectively.
Website name	(Required) Provide the name of the IIS website.
Website name	(Required) Provide the name of the IIS website to create or update.
Physical path	(Required) Provide the physical path where the website content will be stored. The content can reside on the local Computer, or in a remote directory, or on a network share, like C:\Fabrikam or \\ContentShare\Fabrikam.
Physical path authentication	(Required) Select the authentication mechanism that will be used to access the physical path of the website.
Username	(Required) Provide the user name that will be used to access the website's physical path.

ARGUMENT	DESCRIPTION
Password	<p>(Optional) Provide the user's password that will be used to access the website's physical path.</p> <p>The best practice is to create a variable in the Build or Release pipeline, and mark it as 'Secret' to secure it, and then use it here, like '\$(userCredentials)'.</p> <p>Note: Special characters in password are interpreted as per command-line arguments</p>
Add binding	(Optional) Select the option to add port binding for the website.
Protocol	(Required) Select HTTP for the website to have an HTTP binding, or select HTTPS for the website to have a Secure Sockets Layer (SSL) binding.
IP address	<p>(Required) Provide an IP address that end-users can use to access this website.</p> <p>If 'All Unassigned' is selected, then the website will respond to requests for all IP addresses on the port and for the host name, unless another website on the server has a binding on the same port but with a specific IP address.</p>
Port	(Required) Provide the port, where the Hypertext Transfer Protocol Stack (HTTP.sys) will listen to the website requests.
Server Name Indication required	<p>(Optional) Select the option to set the Server Name Indication (SNI) for the website.</p> <p>SNI extends the SSL and TLS protocols to indicate the host name that the clients are attempting to connect to. It allows, multiple secure websites with different certificates, to use the same IP address.</p>
Host name	<p>(Optional) Enter a host name (or domain name) for the website.</p> <p>If a host name is specified, then the clients must use the host name instead of the IP address to access the website.</p>
Host name	<p>(Optional) Enter a host name (or domain name) for the website.</p> <p>If a host name is specified, then the clients must use the host name instead of the IP address to access the website.</p>
Host name	<p>(Required) Enter a host name (or domain name) for the website.</p> <p>If a host name is specified, then the clients must use the host name instead of the IP address to access the website.</p>
SSL certificate thumbprint	(Required) Provide the thumb-print of the Secure Socket Layer certificate that the website is going to use for the HTTPS communication as a 40 character long hexadecimal string. The SSL certificate should be already installed on the Computer, at Local Computer, Personal store.
Add bindings	(Required) Click on the extension [...] button to add bindings for the website.

ARGUMENT	DESCRIPTION
Create or update app pool	(Optional) Select the option to create or update an application pool. If checked, the website will be created in the specified app pool.
Configure authentication	(Optional) Select the option to configure authentication for website.
Name	(Required) Provide the name of the IIS application pool to create or update.
.NET version	(Required) Select the version of the .NET Framework that is loaded by the application pool. If the applications assigned to this application pool do not contain managed code, then select the 'No Managed Code' option from the list.
Managed pipeline mode	(Required) Select the managed pipeline mode that specifies how IIS processes requests for managed content. Use classic mode only when the applications in the application pool cannot run in the Integrated mode.
Identity	(Required) Configure the account under which an application pool's worker process runs. Select one of the predefined security accounts or configure a custom account.
Username	(Required) Provide the username of the custom account that you want to use.
Password	(Optional) Provide the password for custom account. The best practice is to create a variable in the Build or Release pipeline, and mark it as 'Secret' to secure it, and then use it here, like '\$(userCredentials)'. Note: Special characters in password are interpreted as per command-line arguments
Anonymous authentication	(Optional) Select the option to enable anonymous authentication for website.
Basic authentication	(Optional) Select the option to enable basic authentication for website.
Windows authentication	(Optional) Select the option to enable windows authentication for website.
Parent website name	(Required) Provide the name of the parent Website of the virtual directory.
Virtual path	(Required) Provide the virtual path of the virtual directory. Example: To create a virtual directory Site/Application/VDir enter /Application/Vdir. The parent website and application should be already existing.

ARGUMENT	DESCRIPTION
Physical path	(Required) Provide the physical path where the virtual directory's content will be stored. The content can reside on the local Computer, or in a remote directory, or on a network share, like C:\Fabrikam or \\ContentShare\Fabrikam.
Physical path authentication	(Optional) Select the authentication mechanism that will be used to access the physical path of the virtual directory.
Username	(Required) Provide the user name that will be used to access the virtual directory's physical path.
Password	(Optional) Provide the user's password that will be used to access the virtual directory's physical path. The best practice is to create a variable in the Build or Release pipeline, and mark it as 'Secret' to secure it, and then use it here, like '\$(userCredentials)'. Note: Special characters in password are interpreted as per command-line arguments
Parent website name	(Required) Provide the name of the parent Website under which the application will be created or updated.
Virtual path	(Required) Provide the virtual path of the application. Example: To create an application Site/Application enter /Application. The parent website should be already existing.
Physical path	(Required) Provide the physical path where the application's content will be stored. The content can reside on the local Computer, or in a remote directory, or on a network share, like C:\Fabrikam or \\ContentShare\Fabrikam.
Physical path authentication	(Optional) Select the authentication mechanism that will be used to access the physical path of the application.
Username	(Required) Provide the user name that will be used to access the application's physical path.
Password	(Optional) Provide the user's password that will be used to access the application's physical path. The best practice is to create a variable in the Build or Release pipeline, and mark it as 'Secret' to secure it, and then use it here, like '\$(userCredentials)'. Note: Special characters in password are interpreted as per command-line arguments
Create or update app pool	(Optional) Select the option to create or update an application pool. If checked, the application will be created in the specified app pool.
Name	(Required) Provide the name of the IIS application pool to create or update.

Argument	Description
.NET version	(Required) Select the version of the .NET Framework that is loaded by the application pool. If the applications assigned to this application pool do not contain managed code, then select the 'No Managed Code' option from the list.
Managed pipeline mode	(Required) Select the managed pipeline mode that specifies how IIS processes requests for managed content. Use classic mode only when the applications in the application pool cannot run in the Integrated mode.
Identity	(Required) Configure the account under which an application pool's worker process runs. Select one of the predefined security accounts or configure a custom account.
Username	(Required) Provide the username of the custom account that you want to use.
Password	(Optional) Provide the password for custom account. The best practice is to create a variable in the Build or Release pipeline, and mark it as 'Secret' to secure it, and then use it here, like '\$(userCredentials)'. Note: Special characters in password are interpreted as per command-line arguments
Name	(Required) Provide the name of the IIS application pool to create or update.
.NET version	(Required) Select the version of the .NET Framework that is loaded by the application pool. If the applications assigned to this application pool do not contain managed code, then select the 'No Managed Code' option from the list.
Managed pipeline mode	(Required) Select the managed pipeline mode that specifies how IIS processes requests for managed content. Use classic mode only when the applications in the application pool cannot run in the Integrated mode.
Identity	(Required) Configure the account under which an application pool's worker process runs. Select one of the predefined security accounts or configure a custom account.
Username	(Required) Provide the username of the custom account that you want to use.
Password	(Optional) Provide the password for custom account. The best practice is to create a variable in the Build or Release pipeline, and mark it as 'Secret' to secure it, and then use it here, like '\$(userCredentials)'. Note: Special characters in password are interpreted as per command-line arguments
Application pool name	(Required) Provide the name of the IIS application pool.

ARGUMENT	DESCRIPTION
Additional appcmd.exe commands	(Optional) Enter additional AppCmd.exe commands. For more than one command use a line separator, like list apppools list sites recycle apppool /apppool.name:ExampleAppPoolName
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Deploy to Kubernetes task

11/6/2018 • 3 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to deploy, configure, or update a Kubernetes cluster in Azure Container Service by running kubectl commands.

YAML snippet

```
# Deploy to Kubernetes
# Deploy, configure, update your Kubernetes cluster in Azure Container Service by running kubectl commands.
- task: Kubernetes@1
  inputs:
    #connectionType: 'Azure Resource Manager' # Options: azure Resource Manager, kubernetes Service Connection, none
    #kubernetesServiceEndpoint: # Required when connectionType == Kubernetes Service Connection
    #azureSubscriptionEndpoint: # Required when connectionType == Azure Resource Manager
    #azureResourceGroup: # Required when connectionType == Azure Resource Manager
    #kubernetesCluster: # Required when connectionType == Azure Resource Manager
    #namespace: # Optional
    #command: 'apply' # Options: apply, create, delete, exec, expose, get, login, logout, logs, run, set, top
    #useConfigurationFile: false # Optional
    #configuration: # Required when useConfigurationFile == True
    #arguments: # Optional
    #secretType: 'dockerRegistry' # Options: dockerRegistry, generic
    #secretArguments: # Optional
    #containerRegistryType: 'Azure Container Registry' # Required when secretType == DockerRegistry# Options: azure Container Registry, container Registry
    #dockerRegistryEndpoint: # Optional
    #azureSubscriptionEndpointForSecrets: # Optional
    #azureContainerRegistry: # Optional
    #secretName: # Optional
    #forceUpdate: true # Optional
    #configMapName: # Optional
    #forceUpdateConfigMap: false # Optional
    #useConfigMapFile: false # Optional
    #configMapFile: # Required when useConfigMapFile == True
    #configMapArguments: # Optional
    #versionOrLocation: 'version' # Optional. Options: version, location
    #versionSpec: '1.7.0' # Optional
    #checkLatest: false # Optional
    #specifyLocation: # Required when versionOrLocation == Location
    #workingDirectory: '$(System.DefaultWorkingDirectory)' # Optional
    #outputFormat: 'json' # Optional. Options: json, yaml
```

Arguments

ARGUMENT	DESCRIPTION
Kubernetes service connection	(Optional) Select a Kubernetes service connection.
Namespace	(Optional) Set the namespace for the kubectl command by using the –namespace flag. If the namespace is not provided, the commands will run in the default namespace.

ARGUMENT	DESCRIPTION
Command	(Required) Select or specify a kubectl command to run.
Use Configuration files	(Optional) Use Kubernetes configuration file with the kubectl command. Filename, directory, or URL to Kubernetes configuration files can also be provided.
Configuration file	(Required) Filename, directory, or URL to kubernetes configuration files that will be used with the commands.
Arguments	(Optional) Arguments to the specified kubectl command.
Type of secret	(Required) Create/update a generic or dockerimagepullsecret.
Arguments	(Optional) Specify keys and literal values to insert in secret. For example, --from-literal=key1=value1 --from-literal=key2="top secret".
Container Registry type	(Required) Select a Container registry type. The task can use Azure Subscription details to work with an Azure Container registry. Other standard Container registries are also supported.
Docker Registry connection	(Optional) Select a Docker registry connection. Required for commands that need to authenticate with a registry.
Azure subscription	(Optional) Select the Azure Resource Manager subscription, which contains Azure Container Registry. Note: To configure new service connection, select the Azure subscription from the list and click 'Authorize'. If your subscription is not listed or if you want to use an existing Service Principal, you can setup an Azure service connection using 'Add' or 'Manage' button.
Azure Container Registry	(Optional) Select an Azure Container Registry which will be used for pulling container images and deploying applications to the Kubernetes cluster. Required for commands that need to authenticate with a registry.
Secret name	(Optional) Name of the secret. You can use this secret name in the Kubernetes YAML configuration file.
Force update secret	(Optional) Delete the secret if it exists and create a new one with updated values.
ConfigMap name	(Optional) ConfigMaps allow you to decouple configuration artifacts from image content to keep containerized applications portable.
Force update configmap	(Optional) Delete the configmap if it exists and create a new one with updated values.
Arguments	(Optional) Specify keys and literal values to insert in configMap. For example, --from-literal=key1=value1 --from-literal=key2="top secret".
Kubectl	(Optional) undefined

ARGUMENT	DESCRIPTION
Version spec	(Optional) Version Spec of version to get. Examples: 1.7.0, 1.x.0, 4.x.0, 6.10.0, >=6.10.0
Check for latest version	(Optional) Always checks online for the latest available version (stable.txt) that satisfies the version spec. This is typically false unless you have a specific scenario to always get latest. This will cause it to incur download costs when potentially not necessary, especially with the hosted build pool.
Path to Kubectl	(Required) Full path to the kubectl.exe
Working directory	(Optional) Working directory for the Kubectl command.
Output format	(Optional) Output format.
Output variable name	(Optional) Name of the variable in which output of the command should be saved.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

PowerShell on Target Machines task

11/15/2018 • 5 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015

Use this task in a build or release pipeline to execute PowerShell scripts on remote machine(s).

This task can run both PowerShell scripts and PowerShell-DSC scripts:

- For PowerShell scripts, the computers must have PowerShell 2.0 or higher installed.
- For PowerShell-DSC scripts, the computers must have [Windows Management Framework 4.0](#) installed. This is installed by default on Windows 8.1, Windows Server 2012 R2, and later.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Prerequisites

This task uses [Windows Remote Management](#) (WinRM) to access on-premises physical computers or virtual computers that are domain-joined or workgroup-joined.

To set up WinRM for on-premises physical computers or virtual machines

Follow the steps described in [domain-joined](#)

To set up WinRM for Microsoft Azure Virtual Machines

Azure Virtual Machines require WinRM to use the HTTPS protocol. You can use a self-signed Test Certificate. In this case, the automation agent will not validate the authenticity of the certificate as being issued by a trusted certification authority.

- **Azure Classic Virtual Machines.** When you create a [classic virtual machine](#) from the Azure portal, the virtual machine is already set up for WinRM over HTTPS, with the default port 5986 already opened in the firewall and a self-signed certificate installed on the machine. These virtual machines can be accessed with no further configuration required. Existing Classic virtual machines can be also selected by using the [Azure Resource Group Deployment](#) task.
- **Azure Resource Group.** If you have an [Azure Resource Group](#) already defined in the Azure portal, you must configure it to use the WinRM HTTPS protocol. You need to open port 5986 in the firewall, and install a self-signed certificate.

To dynamically deploy Azure Resource Groups that contain virtual machines, use the [Azure Resource Group Deployment](#) task. This task has a checkbox named **Enable Deployment Pre-requisites**. Select this to automatically set up the WinRM HTTPS protocol on the virtual machines, open port 5986 in the firewall, and install a test certificate. The virtual machines are then ready for use in the deployment task.

YAML snippet

```

# PowerShell on Target Machines
# Execute PowerShell scripts on remote machine(s). This version of the task uses PSSession and Invoke-Command for
remoting.
- task: PowerShellOnTargetMachines@3
  inputs:
    machines:
      #userName: # Optional
      #userPassword: # Optional
      #scriptType: 'Inline' # Optional. Options: filePath, inline
      #scriptPath: # Required when scriptType == FilePath
      #inlineScript: '# Write your powershell commands here.Write-Output Hello World' # Required when scriptType ==
      Inline
      #scriptArguments: # Optional
      #initializationScript: # Optional
      #sessionVariables: # Optional
      #communicationProtocol: 'Https' # Optional. Options: http, https
      #authenticationMechanism: 'Default' # Optional. Options: default, credssp
      #newPsSessionOptionArguments: '-SkipCACheck -IdleTimeout 7200000 -OperationTimeout 0 -OutputBufferingMode Block'
    # Optional
      #errorActionPreference: 'stop' # Optional. Options: stop, continue, silentlyContinue
      #failOnStderr: false # Optional
      #ignoreLASTEXITCODE: false # Optional
      #workingDirectory: # Optional
      #runPowershellInParallel: true # Optional

```

Arguments

ARGUMENT	DESCRIPTION
Machines	A comma-separated list of machine FQDNs or IP addresses, optionally including the port number. Can be: <ul style="list-style-type: none"> - The name of an Azure Resource Group. - A comma-delimited list of machine names. Example: dbserver.fabrikam.com,dbserver_int.fabrikam.com:5986,192.168.34:5 - An output variable from a previous task. If you do not specify a port, the default WinRM port is used. This depends on the protocol you have configured: for WinRM 2.0, the default HTTP port is 5985 and the default HTTPS port is 5986.
Admin Login	The username of either a domain or a local administrative account on the target host(s). <ul style="list-style-type: none"> - Formats such as username, domain\username, machine-name\username, and .\\username are supported. - UPN formats such as username@domain.com and built-in system accounts such as NT Authority\System are not supported.
Password	The password for the administrative account specified above. Consider using a secret variable global to the build or release pipeline to hide the password. Example: <code>\$(passwordVariable)</code>
Protocol	The protocol that will be used to connect to the target host, either HTTP or HTTPS .
Test Certificate	If you choose the HTTPS option, set this checkbox to skip validating the authenticity of the machine's certificate by a trusted certification authority.
Deployment - PowerShell Script	The location of the PowerShell script on the target machine. Can include environment variables such as <code>\$env:windir</code> and <code>\$env:systemroot</code> Example: <code>C:\FabrikamFibre\Web\deploy.ps1</code>

ARGUMENT	DESCRIPTION
Deployment - Script Arguments	The arguments required by the script, if any. Example: -applicationPath \$(applicationPath) -username \$(vmusername) -password \$(vmpassword)
Deployment - Initialization Script	The location on the target machine(s) of the data script used by PowerShell-DSC. It is recommended to use arguments instead of an initialization script.
Deployment - Session Variables	Used to set up the session variables for the PowerShell scripts. A comma-separated list such as \$varx=valuex, \$vary=valuey Most commonly used for backward compatibility with earlier versions of the release service. It is recommended to use arguments instead of session variables.
Advanced - Run PowerShell in Parallel	Set this option to execute the PowerShell scripts in parallel on all the target machines
Advanced - Select Machines By	Depending on how you want to specify the machines in the group when using the Filter Criteria parameter, choose Machine Names or Tags .
Advanced - Filter Criteria	Optional. A list of machine names or tag names that identifies the machines that the task will target. The filter criteria can be: - The name of an Azure Resource Group . - An output variable from a previous task. - A comma-delimited list of tag names or machine names. Format when using machine names is a comma-separated list of the machine FDQNs or IP addresses. Specify tag names for a filter as {TagName}:{Value} Example: Role:DB;OS:Win8.1
Control options	See Control options

Version 3.x of the task includes the **Inline script** setting where you can enter your PowerShell script code.

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Service Fabric Application Deployment task

11/15/2018 • 2 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#)

Use this task in a build or release pipeline to deploy a Service Fabric application to a cluster. This task deploys an Azure Service Fabric application to a cluster according to the settings defined in the publish profile.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Prerequisites

Service Fabric

This task uses a Service Fabric installation to connect and deploy to a Service Fabric cluster.

[Download and install Service Fabric](#) on the build agent.

YAML snippet

```
# Service Fabric Application Deployment
# Deploy a Service Fabric application to a cluster.
- task: ServiceFabricDeploy@1
  inputs:
    applicationPackagePath:
    serviceConnectionName:
    #publishProfilePath: # Optional
    #applicationParameterPath: # Optional
    #overrideApplicationParameter: false # Optional
    #compressPackage: false # Optional
    #copyPackageTimeoutSec: # Optional
    #registerPackageTimeoutSec: # Optional
    #overwriteBehavior: 'SameAppTypeAndVersion' # Options: always, never, sameAppTypeAndVersion
    #skipUpgradeSameTypeAndVersion: false # Optional
    #skipPackageValidation: false # Optional
    #useDiffPackage: false # Optional
    #overridePublishProfileSettings: false # Optional
    #isUpgrade: true # Optional
    #unregisterUnusedVersions: true # Optional
    #upgradeMode: 'Monitored' # Required when overridePublishProfileSettings == True && IsUpgrade == True# Options: monitored, unmonitoredAuto, unmonitoredManual
    #failureAction: 'Rollback' # Required when overridePublishProfileSettings == True && IsUpgrade == True && UpgradeMode == Monitored#
Options: rollback, manual
    #upgradeReplicaSetCheckTimeoutSec: # Optional
    #timeoutSec: # Optional
    #forceRestart: false # Optional
    #healthCheckRetryTimeoutSec: # Optional
    #healthCheckWaitDurationSec: # Optional
    #healthCheckStableDurationSec: # Optional
    #upgradeDomainTimeoutSec: # Optional
    #considerWarningAsError: false # Optional
    #defaultServiceTypeHealthPolicy: # Optional
    #maxPercentUnhealthyDeployedApplications: # Optional
    #upgradeTimeoutSec: # Optional
    #serviceTypeHealthPolicyMap: # Optional
    #configureDockerSettings: false # Optional
    #registryCredentials: 'AzureResourceManagerEndpoint' # Required when configureDockerSettings == True# Options:
azureResourceManagerEndpoint, containerRegistryEndpoint, usernamePassword
    #dockerRegistryConnection: # Required when configureDockerSettings == True && RegistryCredentials == ContainerRegistryEndpoint
    #azureSubscription: # Required when configureDockerSettings == True && RegistryCredentials == AzureResourceManagerEndpoint
    #registryUserName: # Optional
    #registryPassword: # Optional
    #passwordEncrypted: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Publish Profile	The location of the publish profile that specifies the settings to use for deployment, including the location of the target Service Fabric cluster. Can include wildcards and variables. Example: <code>\$(system.defaultworkingdirectory)/**/drop/projectartifacts/**/PublishProfileName.publish.xml</code>
Application Package	The location of the Service Fabric application package to be deployed to the cluster. Can include wildcards and variables. Example: <code>\$(system.defaultworkingdirectory)/**/drop/applicationpackage</code>
Cluster Connection	The name of the Azure Service Fabric service connection defined in the TS/TFS project that describes the connection to the cluster.
Control options	See Control options

Also see: [Update Service Fabric App Versions task](#)

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Service Fabric Compose Deploy task

11/15/2018 • 2 minutes to read [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to deploy a Docker-compose application to a Service Fabric cluster. This task deploys an Azure Service Fabric application to a cluster according to the settings defined in the compose file.

Prerequisites

NOTE: This task is currently in preview and requires a preview version of Service Fabric that supports compose deploy. See [/azure/service-fabric/service-fabric-docker-compose](#).

Service Fabric

- This task uses a Service Fabric installation to connect and deploy to a Service Fabric cluster.
- [Azure Service Fabric Core SDK](#) on the build agent.

YAML snippet

```
# Service Fabric Compose Deploy
# Deploy a docker-compose application to a Service Fabric cluster.
- task: ServiceFabricComposeDeploy@0
  inputs:
    clusterConnection:
      #composeFilePath: '**/docker-compose.yml'
      #applicationName: 'fabric:/Application1'
      #registryCredentials: 'AzureResourceManagerEndpoint' # Options: azureResourceManagerEndpoint,
      containerRegistryEndpoint, usernamePassword, none
      #dockerRegistryConnection: # Optional
      #azureSubscription: # Required when registryCredentials == AzureResourceManagerEndpoint
      #registryUserName: # Optional
      #registryPassword: # Optional
      #passwordEncrypted: # Optional
      #upgrade: # Optional
      #deployTimeoutSec: # Optional
      #removeTimeoutSec: # Optional
      #getStatusTimeoutSec: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Cluster Connection	The Azure Service Fabric service connection to use to connect and authenticate to the cluster.
Compose File Path	Path to the compose file that is to be deployed. Can include wildcards and variables. Example: <code>\$(System.DefaultWorkingDirectory)/**/drop/projectartifacts/**/docker-compose.yml</code> • Note: combining compose files is not supported as part of this task.
Application Name	The Service Fabric Application Name of the application being deployed. Use <code>fabric:/</code> as a prefix. Application Names within a Service Fabric cluster must be unique.

ARGUMENT	DESCRIPTION
Registry Credentials Source	<p>Specifies how credentials for the Docker container registry will be provided to the deployment task:</p> <p>Azure Resource Manager Endpoint: An Azure Resource Manager service connection and Azure subscription to be used to obtain a service principal ID and key for an Azure Container Registry.</p> <p>Container Registry Endpoint: A Docker registry service connection. If a certificate matching the Server Certificate Thumbprint in the Cluster Connection is installed on the build agent, it will be used to encrypt the password; otherwise the password will not be encrypted and sent in clear text.</p> <p>Username and Password: Username and password to be used. We recommend you encrypt your password using Invoke-ServiceFabricEncryptText (Check Password Encrypted). If you do not, and a certificate matching the Server Certificate Thumbprint in the Cluster Connection is installed on the build agent, it will be used to encrypt the password; otherwise the password will not be encrypted and sent in clear text.</p> <p>None: No registry credentials are provided (used for accessing public container registries).</p>
Deploy Timeout (s)	Timeout in seconds for deploying the application.
Remove Timeout (s)	Timeout in seconds for removing an existing application.
Get Status Timeout (s)	Timeout in seconds for getting the status of an existing application.
Control options	See Control options

Also see: [Service Fabric PowerShell Utility](#)

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

SSH Deployment task

11/15/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

Use this task in a build or release pipeline to run shell commands or a script on a remote machine using SSH. This task enables you to connect to a remote machine using SSH and run commands or a script.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Prerequisites

- The task supports use of an SSH key pair to connect to the remote machine(s).
- The public key must be pre-installed or copied to the remote machine(s).

YAML snippet

```
# SSH
# Run shell commands or a script on a remote machine using SSH
- task: SSH@0
  inputs:
    sshEndpoint:
      #runOptions: 'commands' # Options: commands, script, inline
      #commands: # Required when runOptions == Commands
      #scriptPath: # Required when runOptions == Script
      #inline: # Required when runOptions == Inline
      #args: # Optional
      #failOnStdErr: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
SSH endpoint	The name of an SSH service connection containing connection details for the remote machine. The hostname or IP address of the remote machine, the port number, and the user name are required to create an SSH service connection. <ul style="list-style-type: none">- The private key and the passphrase must be specified for authentication.- A password can be used to authenticate to remote Linux machines, but this is not supported for macOS or Windows systems.
Run	Choose to run either shell commands or a shell script on the remote machine.

ARGUMENT	DESCRIPTION
Commands	The shell commands to run on the remote machine. This parameter is available only when Commands is selected for the Run option. Enter each command together with its arguments on a new line of the multi-line textbox. To run multiple commands together, enter them on the same line separated by semicolons. Example: <code>cd /home/user/myFolder;build</code>
Shell script path	Path to the shell script file to run on the remote machine. This parameter is available only when Shell script is selected for the Run option.
Arguments	The arguments to pass to the shell script. This parameter is available only when Shell script is selected for the Run option.
Advanced - Fail on STDERR	If this option is selected (the default), the build will fail if the remote commands or script write to STDERR .
Control options	See Control options

See also

- [Install SSH Key task](#)
- [Copy Files Over SSH](#)
- Blog post [SSH build task](#)

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

Windows Machine File Copy task

11/15/2018 • 3 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

Use this task in a build or release pipeline to copy application files and other artifacts such as PowerShell scripts and PowerShell-DSC modules that are required to install the application on Windows Machines. It uses RoboCopy, the command-line utility built for fast copying of data.

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

YAML snippet

```
# Windows Machine File Copy
# Copy files to remote machine(s)
- task: WindowsMachineFileCopy@2
  inputs:
    sourcePath:
      #machineNames: # Optional
      #adminUserName: # Optional
      #adminPassword: # Optional
    targetPath:
      #cleanTargetBeforeCopy: false # Optional
      #copyFilesInParallel: true # Optional
      #additionalArguments: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Source	The path to the files to copy. Can be a local physical path such as <code>c:\files</code> or a UNC path such as <code>\myserver\fileshare\files</code> . You can use pre-defined system variables such as <code>\$(Build.Repository.LocalPath)</code> (the working folder on the agent computer), which makes it easy to specify the location of the build artifacts on the computer that hosts the automation agent.
Machines	A comma-separated list of machine FQDNs or IP addresses, optionally including the port number. Can be: <ul style="list-style-type: none">- The name of an Azure Resource Group.- A comma-delimited list of machine names. Example: <code>dbserver.fabrikam.com,</code> <code>dbserver_int.fabrikam.com:5986,192.168.34:5986</code>- An output variable from a previous task.

ARGUMENT	DESCRIPTION
Admin Login	The username of either a domain or a local administrative account on the target host(s). - Formats such as domain\username , username , and machine-name\username are supported. - UPN formats such as username@domain.com and built-in system accounts such as NT Authority\System are not supported.
Password	The password for the administrative account specified above. Consider using a secret variable global to the build or release pipeline to hide the password. Example: <code>\$(passwordVariable)</code>
Destination Folder	The folder on the Windows machine(s) to which the files will be copied. Example: <code>C:\FabrikamFibre\Web</code>
Advanced - Clean Target	Set this option to delete all the files in the destination folder before copying the new files to it.
Advanced - Copy Files in Parallel	Set this option to copy files to all the target machines in parallel, which can speed up the copying process.
Advanced - Additional Arguments	Arguments to pass to the RoboCopy process. Example: <code>/min:33553332 /1</code>
Select Machines By	Depending on how you want to specify the machines in the group when using the Filter Criteria parameter, choose Machine Names or Tags .
Filter Criteria	Optional. A list of machine names or tag names that identifies the machines that the task will target. The filter criteria can be: - The name of an Azure Resource Group . - An output variable from a previous task. - A comma-delimited list of tag names or machine names. Format when using machine names is a comma-separated list of the machine FDQNs or IP addresses. Specify tag names for a filter as {TagName}:{Value} Example: <code>Role:DB;OS:Win8.1</code>
Control options	See Control options

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

I get a system error 53 when using this task. Why?

Typically this occurs when the specified path cannot be located. This may be due to a firewall blocking the necessary ports for file and printer sharing, or an invalid path specification. For more details, see [Error 53](#) on Technet.

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [Azure Pipelines](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

WinRM SQL Server DB Deployment task

11/6/2018 • 3 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to deploy to SQL Server Database using a DACPAC or SQL script.

YAML snippet

```
# SQL Server Database Deploy
# Deploy to SQL Server Database using DACPAC or SQL scripts
- task: SqlDacpacDeploymentOnMachineGroup@0
  inputs:
    #taskType: 'dacpac' # Options: dacpac, sqlQuery, sqlInline
    #dacpacFile: # Required when taskType == Dacpac
    #sqlFile: # Required when taskType == SqlCommand
    #executeInTransaction: false # Optional
    #exclusiveLock: false # Optional
    #appLockName: # Required when exclusiveLock == True
    #inlineSql: # Required when taskType == SqlCommand
    #targetMethod: 'server' # Required when taskType == Dacpac# Options: server, connectionString,
  publishProfile
    #serverName: 'localhost' # Required when targetMethod == Server || TaskType == SqlCommand || TaskType ==
  SqlInline
    #databaseName: # Required when targetMethod == Server || TaskType == SqlCommand || TaskType == SqlInline
    #authScheme: 'windowsAuthentication' # Required when targetMethod == Server || TaskType == SqlCommand || TaskType ==
  SqlInline# Options: windowsAuthentication, sqlServerAuthentication
    #sqlUsername: # Required when authScheme == SqlServerAuthentication
    #sqlPassword: # Required when authScheme == SqlServerAuthentication
    #connectionString: # Required when targetMethod == ConnectionString
    #publishProfile: # Optional
    #additionalArguments: # Optional
    #additionalArgumentsSql: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Deploy SQL Using	(Required) Specify the way in which you want to deploy DB, either by using Dacpac or by using Sql Script.
DACPAC File	(Required) Location of the DACPAC file on the target machines or on a UNC path like, \\BudgetIT\Web\Deploy\FabrikamDB.dacpac. The UNC path should be accessible to the machine's administrator account. Environment variables are also supported, such as \$env:windir, \$env:systemroot, \$env:windir\FabrikamFibre\DB. Wildcards can be used. For example, **/*.dacpac for DACPAC file present in all sub folders.

ARGUMENT	DESCRIPTION
Sql File	(Required) Location of the SQL file on the target. Provide semi-colon separated list of SQL script files to execute multiple files. The SQL scripts will be executed in the order given. Location can also be a UNC path like, \\BudgetIT\Web\Deploy\FabrikamDB.sql. The UNC path should be accessible to the machine's administrator account. Environment variables are also supported, such as \$env:windir, \$env:systemroot, \$env:windir\FabrikamFibre\DB. Wildcards can be used. For example, <code>**/*.sql</code> for sql file present in all sub folders.
Execute within a transaction	(Optional) Executes SQL script(s) within a transaction
Acquire an exclusive app lock while executing script(s)	(Optional) Acquires an exclusive app lock while executing script(s)
App lock name	(Required) App lock name
Inline Sql	(Required) Sql Queries inline
Specify SQL Using	(Required) Specify the option to connect to the target SQL Server Database. The options are either to provide the SQL Server Database details, or the SQL Server connection string, or the Publish profile XML file.
Server Name	(Required) Provide the SQL Server name like, machinename\FabrikamSQL,1433 or localhost or .\SQL2012R2. Specifying localhost will connect to the Default SQL Server instance on the machine.
Database Name	(Required) Provide the name of the SQL Server database.
Authentication	(Required) Select the authentication mode for connecting to the SQL Server. In Windows authentication mode, the administrator's account, as specified in the Machines section, is used to connect to the SQL Server. In SQL Server Authentication mode, the SQL login and Password have to be provided in the parameters below.
SQL User name	(Required) Provide the SQL login to connect to the SQL Server. The option is only available if SQL Server Authentication mode has been selected.
SQL Password	(Required) Provide the Password of the SQL login. The option is only available if SQL Server Authentication mode has been selected.
Connection String	(Required) Specify the SQL Server connection string like "Server=localhost;Database=Fabrikam;UserID=sqouser;Password=password;".
Publish Profile	(Optional) Publish profile provide fine-grained control over SQL Server database deployments. Specify the path to the Publish profile XML file on the target machine or on a UNC share that is accessible by the machine administrator's credentials.

ARGUMENT	DESCRIPTION
Additional Arguments	(Optional) Additional SqlPackage.exe arguments that will be applied when deploying the SQL Server database like, /p:IgnoreAnsiNulls=True /p:IgnoreComments=True. These arguments will override the settings in the Publish profile XML file (if provided).
Additional Arguments	(Optional) Additional Invoke-Sqlcmd arguments that will be applied when deploying the SQL Server database.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

.NET Core Tool Installer task

11/19/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to acquire a specific version of .NET Core from the internet or the tools cache and add it to the PATH. Use this task to change the version of .NET Core used in subsequent tasks.

YAML snippet

```
# .NET Core SDK Installer
# Acquires a specific version of the .NET Core SDK from the internet or the local cache and adds it to the PATH. Use this task to change the version of .NET Core used in subsequent tasks.
- task: DotNetCoreInstaller@0
  inputs:
    #packageType: 'sdk' # Options: runtime, sdk
    #version: '1.0.4'
```

Arguments

ARGUMENT	DESCRIPTION
Package to install	(Required) Please select whether to install only runtime or full SDK.
Version	(Required) Specify exact version of .NET Core SDK or runtime to install. Examples: 1. To install 1.0.4 SDK, use 1.0.4 2. To install 1.1.2 runtime, use 1.1.2 2. To install 2.0 preview 2 runtime, use 2.0.0-preview2-25407-01 For getting more details about exact version, refer here
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Go Tool Installer task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to find or download a specific version of the Go tool into the tools cache and add it to the PATH. Use the task to change the version of Go Lang used in subsequent tasks.

YAML snippet

```
# Go Tool Installer
# Finds or downloads a specific version of Go in the tools cache and adds it to the PATH. Use this to set the
# version of Go used in subsequent tasks.
- task: GoTool@0
  inputs:
    #version: '1.10'
    #goPath: # Optional
    #goBin: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Version	(Required) Go tool version to download and install. Example: 1.9.3
GOPATH	(Optional) Value for the GOPATH environment variable.
GOBIN	(Optional) Value for the GOBIN environment variable.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Helm Tool Installer task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to install Helm and Kubernetes on an agent machine.

YAML snippet

```
# Helm tool installer
# Install Helm and Kubernetes on agent machine.
- task: HelmInstaller@0
  inputs:
    #helmVersion: '2.9.1'
    #checkLatestHelmVersion: true # Optional
    #installKubectl: true
    #kubectlVersion: '1.8.9' # Optional
    #checkLatestKubectl: true # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Helm Version Spec	(Required) Specify the version of Helm to install
Check for latest version of Helm	(Optional) Check for latest version of Helm.
Install Kubectl	(Required) Install Kubectl.
Kubectl Version Spec	(Optional) Specify the version of Kubectl to install
Check for latest version of kubectl	(Optional) Check for latest version of kubectl.
CONTROL OPTIONS	

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Java Tool Installer task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to acquire a specific version of Java from a user supplied Azure blob, from a location in the source or on the agent, or from the tools cache. The task also sets the JAVA_HOME environment variable. Use this task to change the version of Java used in Java tasks.

Demands

None

YAML snippet

```
# Java Tool Installer
# Acquires a specific version of Java from a user supplied Azure blob or the tools cache and sets JAVA_HOME.
# Use this task to change the version of Java used in Java tasks.
- task: JavaToolInstaller@0
  inputs:
    #versionSpec: '8'
    jdkArchitectureOption: # Options: x64, x86
    jdkSourceOption: # Options: azureStorage, localDirectory
    #jdkFile: # Required when jdkSourceOption == LocalDirectory
    #azureResourceManagerEndpoint: # Required when jdkSourceOption == AzureStorage
    #azureStorageAccountName: # Required when jdkSourceOption == AzureStorage
    #azureContainerName: # Required when jdkSourceOption == AzureStorage
    #azureCommonVirtualFile: # Required when jdkSourceOption == AzureStorage
    jdkDestinationDirectory:
    #cleanDestinationDirectory: true
```

Arguments

ARGUMENT	DESCRIPTION
JDK Version	Specify which JDK version to download and use.
JDK Architecture	Specify the bit version of the JDK.
JDK source	Specify the source for the compressed JDK, either Azure blob storage or a local directory on the agent or source repository.
JDK file	Applicable when JDK is located in a local directory. Specify the path to the folder that contains the compressed JDK. The path could be in your source repository or a local path on the agent.
Azure Subscription	Applicable when the JDK is located in Azure Blob storage. Specify the Azure Resource Manager subscription for the JDK.

ARGUMENT	DESCRIPTION
Storage Account Name	Applicable when the JDK is located in Azure Blob storage. Specify the Storage account name in which the JDK is located. Azure Classic and Resource Manager storage accounts are listed.
Container Name	Applicable when the JDK is located in Azure Blob storage. Specify the name of the container in the storage account in which the JDK is located.
Common Virtual Path	Applicable when the JDK is located in Azure Blob storage. Specify the path to the JDK inside the Azure storage container.
Destination directory	Specify the destination directory into which the JDK should be extracted.
Clean destination directory	Select this option to clean the destination directory before the JDK is extracted into it.
Control options	See Control options .

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Where can I learn more about tool installers?

For an explanation of tool installers and examples, see [Tool installers](#).

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

Node.js Tool Installer task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Build

Use this task in a build or release pipeline to find, download, and cache a specified version of [Node.js](#) and add it to the PATH.

Demands

None

YAML snippet

```
# Node.js Tool Installer
# Finds or downloads and caches the specified version spec of Node.js and adds it to the PATH.
- task: NodeTool@0
  inputs:
    #versionSpec: '6.x'
    #checkLatest: false # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Version Spec	Specify which Node.js version you want to use. Examples: <code>7.x</code> , <code>6.x</code> , <code>6.10.0</code> , <code>>=6.10.0</code>
Check for Latest Version	Select if you want the agent to check for the latest available version that satisfies the version spec. For example, you select this option because you run this build on your self-hosted agent and you want to always use the latest <code>6.x</code> version. <small>TIP</small> If you're using the Microsoft-hosted agents , you should leave this check box cleared. We update the Microsoft-hosted agents on a regular basis, but they're often slightly behind the latest version. So selecting this box will result in your build spending a lot of time updating to a newer minor version.
Control options	See Control options .

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Where can I learn more about tool installers?

For an explanation of tool installers and examples, see [Tool installers](#).

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

NuGet Tool Installer task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Build

Use this task in a build or release pipeline to find, download, and cache a specified version of [NuGet](#) and add it to the PATH.

Demands

None

YAML snippet

```
# NuGet Tool Installer
# Acquires a specific version of NuGet from the internet or the tools cache and adds it to the PATH. Use this
# task to change the version of NuGet used in the NuGet tasks.
- task: NuGetToolInstaller@0
  inputs:
    #versionSpec: '4.3.0'
    #checkLatest: false # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Version Spec	Specify which NuGet version you want to use. Examples: <code>4.1.0</code> , <code>3.x</code> , <code>>2.x</code> , <code>>=3.5</code>
Check for Latest Version	Select if you want the agent to check for the latest available version that satisfies the version spec. For example, you select this option because you run this build on your self-hosted agent and you want to always use the latest <code>3.x</code> version. TIP If you're using the Microsoft-hosted agents , you should leave this check box cleared. We update the Microsoft-hosted agents on a regular basis, but they're often slightly behind the latest version. So selecting this box will result in your build spending a lot of time updating to a newer minor version.
Control options	See Control options .

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Where can I learn more about tool installers?

For an explanation of tool installers and examples, see [Tool installers](#).

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

Use Python Version task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to select a version of Python to run on an agent, and optionally add it to PATH.

Demands

None

Prerequisites

- A [Microsoft-hosted agent](#) with side-by-side versions of Python installed, or a self-hosted agent with Agent.ToolsDirectory configured (see [Q&A](#)).

This task will fail if no Python versions are found in Agent.ToolsDirectory. Available Python versions on Microsoft-hosted agents can be found [here](#).

YAML snippet

```
# Use Python Version
# Retrieves the specified version of Python from the tool cache. Optionally add it to PATH.
- task: UsePythonVersion@0
  inputs:
    #versionSpec: '3.x'
    #addToPath: true
    #architecture: 'x64' # Options: x86, x64
```

Arguments

ARGUMENT	DESCRIPTION
Version spec	Version range or exact version of a Python version to use.
Add to PATH	Whether to prepend the retrieved Python version to the PATH environment variable to make it available in subsequent tasks or scripts without using the output variable.
Advanced - Architecture	The target architecture (x86, x64) of the Python interpreter.

If the task completes successfully, the task's output variable will contain the directory of the Python installation:

Version ▼

Display name *

Version spec * ⓘ

 Add to PATH ⓘ

Control Options ▾

Output Variables ^

Reference name ⓘ

Variables list

 ⓘ

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Where can I learn more about tool installers?

For an explanation of tool installers and examples, see [Tool installers](#).

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

How can I configure a self-hosted agent to use this task?

You can run this task on a self-hosted agent with your own Python versions. To run this task on a self-hosted agent, set up Agent.ToolsDirectory by following the instructions [here](#). The tool name to use is "Python."

Use Ruby Version task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure Pipelines

Use this task in a build or release pipeline to select a version of Ruby to run on an agent, and optionally add it to PATH.

Demands

None

Prerequisites

- A [Microsoft-hosted agent](#) with side-by-side versions of Ruby installed, or a self-hosted agent with Agent.ToolsDirectory configured (see [Q&A](#)).

This task will fail if no Ruby versions are found in Agent.ToolsDirectory. Available Ruby versions on Microsoft-hosted agents can be found [here](#).

YAML snippet

```
# Use Ruby Version
# Retrieves the specified version of Ruby from the tool cache. Optionally add it to PATH.
- task: UseRubyVersion@0
  inputs:
    #versionSpec: '>= 2.4'
    #addToPath: true # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Version spec	Version range or exact version of a Ruby version to use.
Add to PATH	Whether to prepend the retrieved Ruby version to the PATH environment variable to make it available in subsequent tasks or scripts without using the output variable.

If the task completes successfully, the task's output variable will contain the directory of the Ruby installation.

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

Q & A

Where can I learn more about tool installers?

For an explanation of tool installers and examples, see [Tool installers](#).

Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#) for Linux, macOS, or Windows.

I can't select a default agent pool and I can't queue my build or release. How do I fix this?

See [Agent pools](#).

How can I configure a self-hosted agent to use this task?

You can run this task on a self-hosted agent with your own Ruby versions. To run this task on a self-hosted agent, set up Agent.ToolsDirectory by following the instructions [here](#). The tool name to use is "Ruby."

Visual Studio Test Platform Installer task

11/6/2018 • 2 minutes to read • [Edit Online](#)

Azure DevOps Services | TFS 2018 Update 1

Use this task in a build or release pipeline to acquire the [Microsoft test platform](#) from nuget.org or a specified feed, and add it to the tools cache. The installer task satisfies the 'vstest' demand and a subsequent [Visual Studio Test task](#) in a build or release pipeline can run without needing a full Visual Studio install on the agent machine.

Demands

[none]

YAML snippet

```
# Visual Studio Test Platform Installer
# Acquires the test platform from nuget.org or the tools cache. Satisfies the 'vstest' demand and can be used
# for running tests and collecting diagnostic data using the Visual Studio Test task.
- task: VisualStudioTestPlatformInstaller@1
  inputs:
    #packageFeedSelector: 'nugetOrg' # Options: nugetOrg, customFeed, netShare
    #versionSelector: 'latestPreRelease' # Required when packageFeedSelector == NugetOrg ||
    PackageFeedSelector == CustomFeed# Options: latestPreRelease, latestStable, specificVersion
    #testPlatformVersion: # Required when versionSelector == SpecificVersion
    #customFeed: # Required when packageFeedSelector == CustomFeed
    #username: # Optional
    #password: # Optional
    #netShare: # Required when packageFeedSelector == NetShare
```

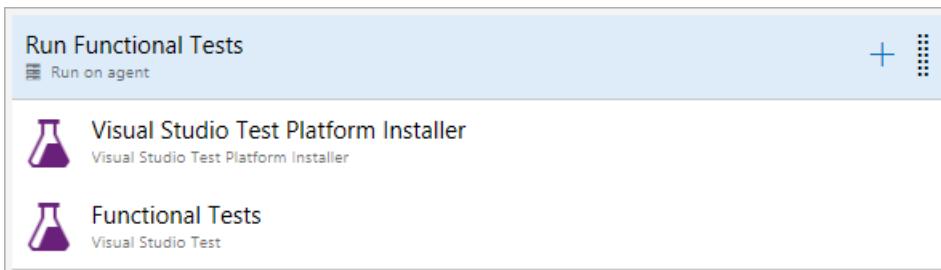
Arguments

ARGUMENT	DESCRIPTION
Package Feed	(Required) Can be: Official NuGet - Use this option to acquire the test platform package from NuGet . This option requires internet connectivity on the agent machine. Custom feed - Use this option to acquire the test platform package from a custom feed or a package management feed in Azure DevOps or TFS. Network path - Use this option to install the test platform from a network share. The desired version of Microsoft.TestPlatform.nupkg file must be downloaded from NuGet and placed on a network share that the build/release agent can access.
Version	(Required) Select whether to install the latest version (including any pre-release versions), the latest stable version, or a specific version of the Visual Studio Test Platform.
Test Platform Version	(Required) Specify the version of Visual Studio Test Platform to install on the agent. Available versions can be viewed on NuGet .

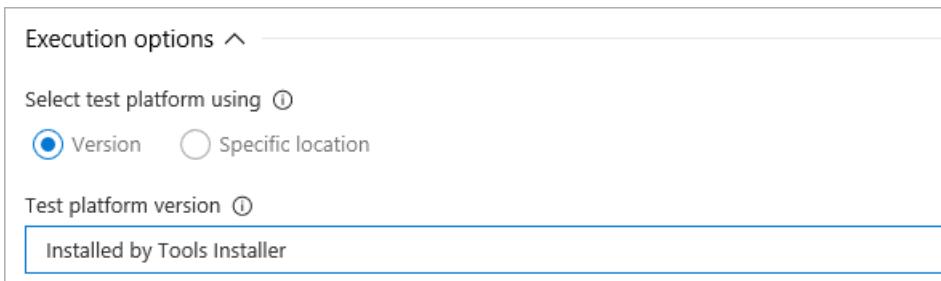
ARGUMENT	DESCRIPTION
Package Source	(Required) Specify the URL of a custom feed or a package management feed in Azure DevOps or TFS that contains the test platform package. Public as well as private feeds can be specified.
Username	Specify the user name to authenticate with the feed specified in the Package Source argument. If using a personal access token (PAT) in the password argument, this input is not required.
Password	Specify the password or personal access token (PAT) to authenticate with the feed specified in the Package Source argument.
UNC Path	(Required) Specify the full UNC path to the Microsoft.TestPlatform.nupkg file. The desired version of Microsoft.TestPlatform.nupkg must be downloaded from NuGet and placed on a network share that the build/release agent can access.

NOTE:

- The **Visual Studio Test Platform Installer** task must appear before the **Visual Studio Test** task in the build or release pipeline.



- The **Test platform version** option in the **Visual Studio Test** task must be set to **Installed by Tools Installer**.



See [Run automated tests from test plans](#)

Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

File matching patterns reference

11/6/2018 • 2 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

Pattern syntax

A pattern is a string or list of newline-delimited strings. File and directory names are compared to patterns to include (or sometimes exclude) them in a task. You can build up complex behavior by stacking multiple patterns.

Match characters

Most characters are used as exact matches. The following characters have special behavior.

- `*` matches zero or more characters within a file or directory name. See [examples](#).
- `?` matches any single character within a file or directory name. See [examples](#).
- `[]` matches a set or range of characters within a file or directory name. See [examples](#).
- `**` recursive wildcard. For example, `/hello/**/*` matches all descendants of `/hello`.

Extended globbing

- `?(hello|world)` - matches `hello` or `world` zero or one times
- `*(hello|world)` - zero or more occurrences
- `+(hello|world)` - one or more occurrences
- `@(hello|world)` - exactly once
- `!(hello|world)` - not `hello` or `world`

Note, extended globs cannot span directory separators. For example, `+(hello/world|other)` is not valid.

Comments

Patterns that begin with `#` are treated as comments.

Exclude patterns

Leading `!` changes the meaning of an include pattern to exclude. Interleaved exclude patterns are supported.

Multiple `!` flips the meaning. See [examples](#).

Escaping

Wrapping special characters in `[]` can be used to escape literal glob characters in a file name. For example the literal file name `hello[a-z]` can be escaped as `hello[[]a-z]`.

Slash

`/` is used as the path separator. Even on Windows, use `/` to ensure that the pattern works on any agent.

Examples

Basic pattern examples

Asterisk examples

Example 1: Given the pattern `*Website.sln` and files:

```
ConsoleHost.sln  
ContosoWebsite.sln  
FabrikamWebsite.sln  
Website.sln
```

The pattern would match:

```
ContosoWebsite.sln  
FabrikamWebsite.sln  
Website.sln
```

Example 2: Given the pattern `*Website/*.proj` and paths:

```
ContosoWebsite/index.html  
ContosoWebsite/ContosoWebsite.proj  
FabrikamWebsite/index.html  
FabrikamWebsite/FabrikamWebsite.proj
```

The pattern would match:

```
ContosoWebsite/ContosoWebsite.proj  
FabrikamWebsite/FabrikamWebsite.proj
```

Question mark examples

Example 1: Given the pattern `log?.log` and files:

```
log1.log  
log2.log  
log3.log  
script.sh
```

The pattern would match:

```
log1.log  
log2.log  
log3.log
```

Example 2: Given the pattern `image.???` and files:

```
image.tiff  
image.png  
image.ico
```

The pattern would match:

```
image.png  
image.ico
```

Character set examples

Example 1: Given the pattern `Sample[AC].dat` and files:

```
SampleA.dat  
SampleB.dat  
SampleC.dat  
SampleD.dat
```

The pattern would match:

```
SampleA.dat  
SampleC.dat
```

Example 2: Given the pattern `Sample[A-C].dat` and files:

```
SampleA.dat  
SampleB.dat  
SampleC.dat  
SampleD.dat
```

The pattern would match:

```
SampleA.dat  
SampleB.dat  
SampleC.dat
```

Example 3: Given the pattern `Sample[A-CEG].dat` and files:

```
SampleA.dat  
SampleB.dat  
SampleC.dat  
SampleD.dat  
SampleE.dat  
SampleF.dat  
SampleG.dat  
SampleH.dat
```

The pattern would match:

```
SampleA.dat  
SampleB.dat  
SampleC.dat  
SampleE.dat  
SampleG.dat
```

Exclude pattern examples

Given the pattern:

```
*
```

```
!* .xml
```

and files:

```
ConsoleHost.exe  
ConsoleHost.pdb  
ConsoleHost.xml  
Fabrikam.dll  
Fabrikam.pdb  
Fabrikam.xml
```

The pattern would match:

```
ConsoleHost.exe  
ConsoleHost.pdb  
Fabrikam.dll  
Fabrikam.pdb
```

Double exclude

Given the pattern:

```
*
```



```
!*.*xml
```



```
!!Fabrikam.xml
```

and files:

```
ConsoleHost.exe  
ConsoleHost.pdb  
ConsoleHost.xml  
Fabrikam.dll  
Fabrikam.pdb  
Fabrikam.xml
```

The pattern would match:

```
ConsoleHost.exe  
ConsoleHost.pdb  
Fabrikam.dll  
Fabrikam.pdb  
Fabrikam.xml
```

File transforms and variable substitution reference

11/19/2018 • 7 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017

NOTE

Build and release *pipelines* are called *definitions* in Microsoft Team Foundation Server (TFS) 2018 and in older versions. Service connections are called *service endpoints* in TFS 2018 and in older versions.

Some tasks, such as the [Azure App Service Deploy](#) task version 3 and later and the [IIS Web App Deploy](#) task, allow users to configure the package based on the environment specified. These tasks use **msdeploy.exe**, which supports the overriding of values in the **web.config** file with values from the **parameters.xml** file.

Configuration substitution is specified in the **File Transform and Variable Substitution Options** section of the settings for the tasks. The transformation and substitution options are:

- [XML transformation](#)
- [XML variable substitution](#)
- [JSON variable substitution](#)

When the task runs, it first performs XML transformation, XML variable substitution, and JSON variable substitution on configuration and parameters files. Next, it invokes **msdeploy.exe**, which uses the **parameters.xml** file to substitute values in the **web.config** file.

XML Transformation

XML transformation supports transforming the configuration files (`*.config` files) by following [Web.config Transformation Syntax](#) and is based on the environment to which the web package will be deployed. This option is useful when you want to add, remove or modify configurations for different environments. Transformation will be applied for other configuration files including Console or Windows service application configuration files (for example, **FabrikamService.exe.config**).

Configuration transform file naming conventions

XML transformation will be run on the `*.config` file for transformation configuration files named `*.Release.config` or `*.<stage>.config` and will be executed in the following order:

1. `*.Release.config` (for example, **fabrikam.Release.config**)
2. `*.<stage>.config` (for example, **fabrikam.Production.config**)

For example, if your package contains the following files:

- Web.config
- Web.Debug.config
- Web.Release.config
- Web.Production.config

and your stage name is **Production**, the transformation is applied for `Web.config` with `Web.Release.config` followed by `Web.Production.config`.

XML transformation example

1. Create a Web Application package with the necessary configuration and transform files. For example, use the following configuration files:

Configuration file

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <connectionStrings>
    <add name="DefaultConnection"
         connectionString="Data Source=(LocalDb)\MSDB;DbFilename=aspcore-local.mdf;" />
  </connectionStrings>
  <appSettings>
    <add key="webpages:Version" value="3.0.0.0" />
    <add key="webpages:Enabled" value="false" />
  </appSettings>
  <system.web>
    <authentication mode="None" />
    <compilation targetFramework="4.5" debug="true" />
  </system.web>
</configuration>
```

Transform file

```
<?xml version="1.0"?>
<configuration xmlns:xdt="http://schemas.microsoft.com/XML-Document-Transform">
  <connectionStrings>
    <add name="MyDB"
         connectionString="Data Source=ReleaseSQLServer;Initial Catalog=MyReleaseDB;Integrated
Security=True"
         xdt:Transform="Insert" />
  </connectionStrings>
  <appSettings>
    <add xdt:Transform="Replace" xdt:Locator="Match(key)" key="webpages:Enabled" value="true" />
  </appSettings>
  <system.web>
    <compilation xdt:Transform="RemoveAttributes(debug)" />
  </system.web>
</configuration>
```

This example transform configuration file does three things:

- It adds a new database connection string inside the `ConnectionString` element.
- It modifies value of `Webpages:Enabled` inside the `appSettings` element.
- It removes the `debug` attribute from the `compilation` element inside the `System.Web` element.

For more information, see [Web.config Transformation Syntax for Web Project Deployment Using Visual Studio](#)

2. Create a release pipeline with an stage named **Release**.
3. Add an **Azure App Service Deploy** task and set (tick) the **XML transformation** option.

4. Save the release pipeline and start a new release.
5. Open the `Web.config` file to see the transformations from `Web.Release.config`.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <connectionStrings>
    <add name="DefaultConnection"
      connectionString="Data Source=(LocalDb)\MSDB;DbFilename=aspcore-local.mdf;" />
    <add name="MyDB"
      connectionString="Data Source=ReleaseSQLServer;Initial Catalog=MyReleaseDB;Integrated
      Security=True" />
  </connectionStrings>
  <appSettings>
    <add key="webpages:Version" value="3.0.0.0" />
    <add key="webpages:Enabled" value="true" />
  </appSettings>
  <system.web>
    <authentication mode="None" />
    <compilation targetFramework="4.5" />
  </system.web>
</configuration>
```

Note:

- You can use this technique to create a default package and deploy it to multiple stages.
- XML transformation takes effect only when the configuration file and transform file are in the same folder within the specified package.
- Set the **Copy to Output Directory** property for the configuration transform files to **Copy If Newer**.
- By default, MSBuild applies the transformation as it generates the web package if the `<DependentUpon>` element is already present in the transform file in the `*.csproj` file. In such cases, the **Azure App Service Deploy** task will fail because there is no further transformation applied on the `Web.config` file. Therefore, it is recommended that the `<DependentUpon>` element is removed from all the transform files to disable any build-time configuration when using XML transformation.

```

...
<Content Include="Web.Debug.config">
    <DependentUpon>Web.config</DependentUpon>
</Content>
<Content Include="Web.Release.config">
    <DependentUpon>Web.config</DependentUpon>
</Content>
...

```

XML variable substitution

This feature enables you to modify configuration settings in configuration files (`*.config` files) inside web packages and XML parameters files (`parameters.xml`). In this way, the same package can be configured based on the environment to which it will be deployed.

Variable substitution takes effect only on the `applicationSettings`, `appSettings`, `connectionStrings`, and `configSections` elements of configuration files.

XML variable substitution example

As an example, consider the task of changing the following values in `Web.config`:

```

<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <configSection>
        <section name="entityFramework" />
    </configSection>
    <connectionStrings>
        <!-- Change connectionString in this line: -->
        <add name="DefaultConnection"
            connectionString="Data Source=(LocalDB)\LocalDB;FileName=Local.mdf" />
    </connectionStrings>
    <appSettings>
        <add key="ClientValidationEnabled" value="true" />
        <add key="UnobtrusiveJavascriptEnabled" value="true" />
        <!-- Change AdminUserName in this line: -->
        <add key="AdminUserName" value="__AdminUserName__" />
        <!-- Change AdminPassword in this line: -->
        <add key="AdminPassword" value="__AdminPasword__" />
    </appSettings>
    <entityFramework>
        <defaultConnectionFactory type="System.Data.Entity.LocalDbConnectionFactory">
            <parameters></parameters>
        </defaultConnectionFactory>
        <providers>
            <!-- Change invariantName in this line: -->
            <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer" />
        </providers>
    </entityFramework>
</configuration>

```

1. Create a release pipeline with a stage named **Release**.
2. Add an **Azure App Service Deploy** task and set (tick) the **XML variable substitution** option.

The screenshot shows the Azure DevOps interface for creating a new release definition. The top navigation bar includes 'Fabrikam' (selected), 'Dashboards', 'Code', 'Work', 'Build & Release', 'Test', and 'Wiki*'. Below the navigation is a secondary menu with 'Builds', 'Releases', 'Library', 'Task Groups', and 'Deployment Groups*'. The main area is titled 'New Release Definition' with a 'Pipeline' tab selected. Under the pipeline, there's a 'Release' section labeled 'Deployment process'. Below it is a 'Run on agent' section with a 'Run on agent' task. A specific 'Deploy Azure App Service' task is selected, indicated by a checkmark icon. To the right of the tasks, there's a configuration panel for 'File Transforms & Variable Substitution Options'. It contains three checkboxes: 'Generate Web.config', 'XML transformation', and 'XML variable substitution'. The 'XML variable substitution' checkbox is checked and has a red box drawn around it.

3. Define the required values in release pipeline variables:

NAME	VALUE	SECURE	SCOPE
DefaultConnection	Data Source=(ProdDB)\MSSQLProdDB;AttachFileName=Local.mdf	No	Release
AdminUserName	ProdAdminName	No	Release
AdminPassword	[your-password]	Yes	Release
invariantName	System.Data.SqlClientExtension	No	Release

4. Save the release pipeline and start a new release.

5. Open the `Web.config` file to see the variable substitutions.

```

<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <configSection>
        <section name="entityFramework" />
    </configSection>
    <connectionStrings>
        <add name="DefaultConnection"
            connectionString="Data Source=(ProdDB)\MSSQLProdDB;AttachFileName=Local.mdf" />
    </connectionStrings>
    <appSettings>
        <add key="ClientValidationEnabled" value="true" />
        <add key="UnobtrusiveJavascriptEnabled" value="true" />
        <add key="AdminUserName" value="ProdAdminName" />
        <add key="AdminPassword" value="*password_masked_for_display*" />
    </appSettings>
    <entityFramework>
        <defaultConnectionFactory type="System.Data.Entity.LocalDbConnectionFactory">
            <parameters></parameters>
        </defaultConnectionFactory>
        <providers>
            <provider invariantName="System.Data.SqlClientExtension"
                type="System.Data.Entity.SqlServer" />
        </providers>
    </entityFramework>
</configuration>

```

Note:

- By default, ASP.NET applications have a default parameterized connection attribute. These values are overridden only in the `parameters.xml` file inside the web package.
- Because substitution occurs before deployment, the user can override the values in `Web.config` using `parameters.xml` (inside the web package) or a `setparameters` file.

JSON variable substitution

This feature substitutes values in the JSON configuration files. It overrides the values in the specified JSON configuration files (for example, `appsettings.json`) with the values matching names of release pipeline and stage variables.

To substitute variables in specific JSON files, provide newline-separated list of JSON files. File names must be specified relative to the root folder. For example, if your package has this structure:

```

/WebPackage(.zip)
    ----- content
        ----- website
            ----- appsettings.json
            ----- web.config
            ----- [other folders]
    --- archive.xml
    --- systeminfo.xml

```

and you want to substitute values in `appsettings.json`, enter the relative path from the root folder; for example `content/website/appsettings.json`. Alternatively, use wildcard patterns to search for specific JSON files. For example, `**/appsettings.json` returns the relative path and name of files named `appsettings.json`.

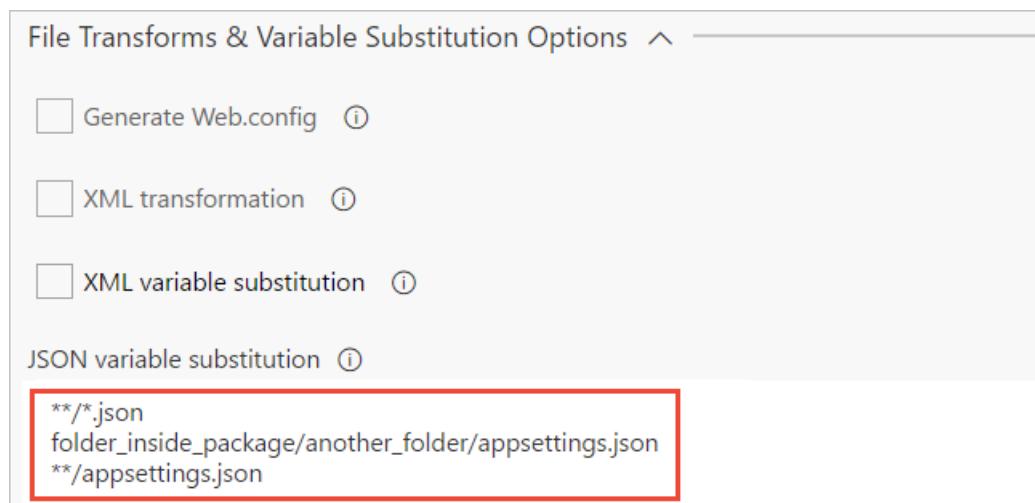
JSON variable substitution example

As an example, consider the task of overriding values in this JSON file:

```
{
  "Data": {
    "DefaultConnection": {
      "ConnectionString": "Data Source=(LocalDb)\MSDB;AttachDbFilename=aspcore-local.mdf;"
    },
    "DebugMode": "enabled",
    "DBAccess": {
      "Administrators": ["Admin-1", "Admin-2"],
      "Users": ["Vendor-1", "vendor-3"]
    },
    "FeatureFlags": {
      "Preview": [
        {
          "newUI": "AllAccounts"
        },
        {
          "NewWelcomeMessage": "Newusers"
        }
      ]
    }
  }
}
```

The task is to override the values of **ConnectionString**, **DebugMode**, the first of the **Users** values, and **NewWelcomeMessage** at the respective places within the JSON file hierarchy.

1. Create a release pipeline with a stage named **Release**.
2. Add an **Azure App Service Deploy** task and enter a newline-separated list of JSON files to substitute the variable values in the **JSON variable substitution** textbox. Files names must be relative to the root folder. You can use wildcards to search for JSON files. For example: `**/*.json` means substitute values in all the JSON files within the package.



3. Define the required substitution values in release pipeline or stage variables.

NAME	VALUE	SECURE	SCOPE
Data.DebugMode	disabled	No	Release
Data.DefaultConnection.ConnectionString	Data Source=(prodDB)\MSDB;AttachDbFilename=prod.mdf;	No	Release
Data.DBAccess.Users.0	Admin-3	Yes	Release

NAME	VALUE	SECURE	SCOPE
Data.FeatureFlags.Preview.1.NewWelcomeMessage	AllAccounts	No	Release

4. Save the release pipeline and start a new release.
5. After the transformation, the JSON will contain the following:

```
{
  "Data": {
    "DefaultConnection": {
      "ConnectionString": "Data Source=(prodDB)\MSDB;AttachDbFilename=prod.mdf;"
    },
    "DebugMode": "disabled",
    "DBAccess": {
      "Administrators": ["Admin-1", "Admin-2"],
      "Users": ["Admin-3", "vendor-3"]
    },
    "FeatureFlags": {
      "Preview": [
        {
          "newUI": "AllAccounts"
        },
        {
          "NewWelcomeMessage": "AllAccounts"
        }
      ]
    }
  }
  ...
}
```

Note:

- To substitute values in nested levels of the file, concatenate the names with a period (`.`) in hierarchical order.
- A JSON object may contain an array whose values can be referenced by their index. For example, to substitute the first value in the **Users** array shown above, use the variable name `DBAccess.Users.0`. To update the value in **NewWelcomeMessage**, use the variable name `FeatureFlags.Preview.1.NewWelcomeMessage`.
- Only **String** substitution is supported for JSON variable substitution.
- Substitution is supported for only UTF-8 and UTF-16 LE encoded files.
- If the file specification you enter does not match any file, the task will fail.
- Variable name matching is case-sensitive.