

Contents

[Azure Pipelines](#)

[Use Azure Pipelines](#)

[What is Azure Pipelines?](#)

[Get started](#)

[Sign up for Azure Pipelines](#)

[Create your first pipeline](#)

[Clone or import a pipeline](#)

[Customize your pipeline](#)

[Multi-stage pipelines user experience](#)

[DevOps Starter](#)

[Pipeline basics](#)

[Key concepts](#)

[YAML pipeline editor](#)

[Repositories](#)

[Supported repositories](#)

[Azure Repos Git](#)

[GitHub](#)

[GitHub Enterprise Server](#)

[Pipeline options for Git repositories](#)

[Bitbucket Cloud](#)

[Bitbucket Server](#)

[TFVC](#)

[Subversion](#)

[Multiple repositories](#)

[Build history](#)

[Triggers](#)

[Types of triggers](#)

[Scheduled triggers](#)

[Pipeline completion triggers](#)

[Build completion triggers \(classic\)](#)

[Release triggers \(classic\)](#)

[Tasks & templates](#)

[Task types & usage](#)

[Task groups](#)

[Template types & usage](#)

[Add a custom task extension](#)

[Jobs & stages](#)

[Specify jobs in your pipeline](#)

[Define container jobs](#)

[Add stages, dependencies & conditions](#)

[Deployment jobs](#)

[Author a custom pipeline decorator](#)

[Pipeline decorator context](#)

[Specify conditions](#)

[Specify demands](#)

[Library, variables & secure files](#)

[Library & shared resources](#)

[Define variables](#)

[Use variables in variable group](#)

[Use predefined variables](#)

[Use runtime parameters](#)

[Use classic process parameters](#)

[Use classic release and artifacts variables](#)

[Use secrets from Azure Key Vault](#)

[Use Azure Key Vault in your YAML Pipeline](#)

[Approvals, checks, & gates](#)

[Release approval and gates overview](#)

[Define approvals & checks](#)

[Define a gate](#)

[Use approvals and gates](#)

[Use approvals for release deployment control](#)

Pipeline runs

 Pipeline run sequence

 Job access tokens

Pipeline reports

 View pipeline reports

 Add widgets to a dashboard

 Test Results Trend (Advanced)

Ecosystems & integration

 Ecosystem support

 .NET Core

 .NET Framework

 JavaScript and Node.js apps

 Python

 Python to web app

 Anaconda

 C/C++

 Java

 Java apps

 Java to web App

 Java to web app with MicroProfile

 Java to Azure Function

 Android

 Go

 PHP

 PHP to web app

 Ruby

 Xamarin

 Xcode

 GitHub Actions

Build apps

 Build multiple branches

 Build on multiple platforms

[Publish Pipeline Artifacts](#)

[Use service containers](#)

[Cross-platform scripts](#)

[Run a PowerShell script](#)

[Run Git commands](#)

[Reduce build time using caching](#)

[Configure build run numbers](#)

[Classic Build options](#)

[Use tasks to increase run retention](#)

[Run pipeline tests](#)

[About pipeline tests](#)

[Set up parallel testing \(VSTest\)](#)

[Set up parallel testing \(Test Runner\)](#)

[Enable Test Impact Analysis \(TIA\)](#)

[Enable flaky test management](#)

[Run UI tests](#)

[Run UI tests with Selenium](#)

[Requirements traceability](#)

[Review test results](#)

[Review test results](#)

[Review test Analytics](#)

[Review code coverage](#)

[Review pull request code coverage](#)

[Deploy apps](#)

[Deploy apps to environments](#)

[Define and target environments](#)

[Kubernetes resource](#)

[Virtual machine resource](#)

[Deploy apps using VMs](#)

[Linux virtual machines](#)

[Manage a VM in Azure DevTest Labs](#)

[Deploy mobile apps](#)

- [Manage signing certificates](#)
- [Manage signing certificates \(TFS-2017\)](#)
- [Deploy apps \(Classic\)](#)
 - [Release pipelines overview](#)
 - [Deploy from multiple branches](#)
 - [Deploy pull request Artifacts](#)
 - [Classic release pipelines](#)
 - [Pipelines with PowerShell DSC](#)
 - [Stage templates in Azure Pipelines](#)
 - [View release progress and test results](#)
- [Deploy apps to VMs \(Classic\)](#)
 - [Linux VMs \(Classic\)](#)
 - [Windows VMs \(Classic\)](#)
 - [IIS servers \(WinRM\) \(Classic\)](#)
 - [Extend IIS Server deployments \(Classic\)](#)
 - [SCVMM \(Classic\)](#)
 - [VMware \(Classic\)](#)
- [Deploy to Azure](#)
 - [Deploy to Azure overview](#)
 - [Azure Resource Manager service connection](#)
 - [Azure App Service \(Web Apps\)](#)
 - [Web App on Linux](#)
 - [Web App on Linux Container](#)
 - [Web App on Windows](#)
 - [Visual Studio Code Deploy to Azure extension](#)
 - [Azure Web App \(Classic\)](#)
 - [Azure Web App for Containers \(Classic\)](#)
 - [Azure Functions](#)
 - [Function App on Container](#)
 - [Function App on Linux](#)
 - [Function App on Windows](#)
 - [Azure Resource Manager \(ARM\) templates](#)

- [Use a Resource Manager template to deploy Azure VM using Azure Resource Manager \(Classic\)](#)
- [Azure Databricks](#)
 - [Data pipeline overview](#)
 - [Build a data pipeline](#)
- [Azure Machine Learning](#)
- [Azure Databases](#)
 - [Azure SQL database](#)
 - [Azure Cosmos DB CI/CD \(Classic\)](#)
- [Azure Virtual Machine Scale Set](#)
 - [Deploy a virtual machine scale set](#)
- [Azure Government Cloud](#)
- [Azure Stack](#)
- [Azure Container Registry](#)
 - [Build and push to Azure Container Registry](#)
 - [Deploy ASP.NET Core app](#)
- [Azure Kubernetes Service](#)
 - [Build and deploy to Azure Kubernetes Service](#)
 - [Deploy a Docker container app](#)
 - [Deploy to AKS with Jenkins](#)
- [Azure IoT Edge \(Classic\)](#)
- [Azure Policy Compliance \(Classic\)](#)
- [Deploy apps using containers](#)
 - [Build images](#)
 - [Push images](#)
 - [Content trust](#)
 - [Kubernetes](#)
 - [Deploy manifests](#)
 - [Bake manifests](#)
 - [Multi-cloud deployments](#)
 - [Deployment strategies](#)
 - [Kubernetes canary deployments](#)

Consume & publish artifacts

[About artifacts](#)

[Publish & download artifacts](#)

[Build artifacts](#)

[Releases in Azure Pipelines](#)

[Release artifacts and artifact sources](#)

[Maven](#)

[npm](#)

[NuGet](#)

[Python](#)

[Symbols](#)

[Universal Packages](#)

[Restore NuGet packages](#)

[Restore & publish NuGet packages \(Jenkins\)](#)

Create & use resources

[About resources](#)

[Add resources to a pipeline](#)

[Add protection to a repository resource](#)

[Add & use variable groups](#)

[Use secure files](#)

[Manage service connections](#)

Manage agents & agent pools

[About agents & agent pools](#)

[Add & manage agent pools](#)

[Historical graph for agent pools](#)

[Microsoft-hosted agents](#)

[Self-hosted Linux agents](#)

[Self-hosted macOS agents](#)

[Self-hosted Windows agents](#)

[Windows agents \(TFS 2015\)](#)

[Scale set agents](#)

[Run an agent behind a web proxy](#)

- [Run an agent in Docker](#)
- [Use a self-signed certificate](#)
- [Create & use deployment groups](#)
 - [Provision deployment groups](#)
 - [Provision agents for deployment groups](#)
 - [Add a deployment group job to a release pipeline](#)
 - [Deploying to Azure VMs using deployment groups in Azure Pipelines](#)
- [Configure security & settings](#)
 - [Set retention policies](#)
 - [Configure and pay for parallel jobs](#)
 - [Set pipelines permissions](#)
 - [Add users to contribute to pipelines](#)
 - [Grant version control permissions to the build service](#)
- [Pipelines security walkthrough](#)
 - [Overview](#)
 - [Approach to securing YAML pipelines](#)
 - [Secure repositories](#)
 - [Pipeline resources](#)
 - [Project structure](#)
 - [Security through templates](#)
 - [Variables and parameters](#)
 - [Shared infrastructure](#)
 - [Other security considerations](#)
- [Integrate with 3rd party software](#)
 - [Microsoft Teams](#)
 - [Slack](#)
 - [Integrate with ServiceNow](#)
 - [Integrate with Jenkins \(Classic\)](#)
- [Migrate](#)
 - [Migrate from Classic pipelines](#)
 - [Migrate from Jenkins](#)
 - [Migrate from Travis](#)

[Migrate from XAML builds](#)

[Migrate from Lab Management](#)

[Pipeline tasks](#)

[Task index](#)

[Build tasks](#)

[.NET Core CLI](#)

[Android build](#)

[Android signing](#)

[Ant](#)

[Azure IoT Edge](#)

[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 \(TFS 2018 and lower\)](#)

[Xcode Package iOS](#)

[Utility tasks](#)

[Archive files](#)

[Azure Network Load Balancer](#)

[Bash](#)

[Batch script](#)

- [Cache](#)
- [Command line](#)
- [Copy and Publish Build Artifacts](#)
- [Copy Files](#)
- [cURL Upload Files](#)
- [Decrypt File](#)
- [Delay](#)
- [Delete Files](#)
- [Download Build Artifacts](#)
- [Download Fileshare Artifacts](#)
- [Download GitHub Release](#)
- [Download Package](#)
- [Download Pipeline Artifact](#)
- [Download Secure File](#)
- [Extract Files](#)
- [File Transform](#)
- [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](#)
- [Manual Validation](#)
- [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 Manifests](#)
- [Xamarin license \(deprecated\)](#)
- [Test tasks](#)
 - [App Center Test](#)
 - [Cloud-based Apache JMeter Load Test](#)
 - [Cloud-based Load Test](#)
 - [Cloud-based Web Performance Test](#)
 - [Container Structure Test Task](#)
 - [Publish Code Coverage Results](#)
 - [Publish Test Results](#)
 - [Run Functional Tests](#)
 - [Visual Studio Test](#)
 - [Visual Studio Test Agent Deployment](#)
 - [Xamarin Test Cloud task \(deprecated\)](#)
- [Package tasks](#)
 - [CocoaPods](#)
 - [Conda Environment](#)
 - [Maven Authenticate](#)
 - [npm](#)
 - [npm Authenticate](#)
 - [NuGet](#)
 - [NuGet Authenticate](#)
 - [PyPI Publisher](#)
 - [Python Pip Authenticate](#)
 - [Python Twine Upload Authenticate](#)
 - [Universal Packages](#)
 - [Xamarin Component Restore](#)
 - [Previous versions](#)
 - [NuGet Installer 0.*](#)

- [NuGet Restore 1.*](#)
- [NuGet Packager 0.*](#)
- [NuGet Publisher 0.*](#)
- [NuGet Command 0.*](#)
- [Pip Authenticate 0.*](#)
- [Twine Authenticate 0.*](#)

Deploy tasks

- [App Center Distribute](#)
- [Azure App Service Deploy](#)
- [Azure App Service Manage](#)
- [Azure App Service Settings](#)
- [Azure CLI](#)
- [Azure Cloud PowerShell Deployment](#)
- [Azure File Copy](#)
- [Azure File Copy \(previous version\)](#)
- [Azure Function App](#)
- [Azure Function App for Container](#)
- [Azure Key Vault](#)
- [Azure Monitor Alerts](#)
- [Azure MySQL Deployment](#)
- [Azure Policy](#)
- [Azure PowerShell](#)
- [Azure Resource Group Deployment](#)
- [Azure Spring Cloud](#)
- [Azure SQL Database Deployment](#)
- [Azure Web App](#)
- [Azure virtual machine scale set deployment](#)
- [Azure Web App for Container](#)
- [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 \(Machine Group\)](#)
- [Kubectl task](#)
- [Kubernetes manifest task](#)
- [PowerShell on Target Machines](#)
- [Service Fabric App Deployment](#)
- [Service Fabric Compose Deploy](#)
- [SSH](#)
- [Windows Machine File Copy](#)
- [WinRM SQL Server DB Deployment](#)
- [MySQL Database Deployment On Machine Group](#)
- [Tool tasks](#)
 - [Docker Installer](#)
 - [Go Tool Installer](#)
 - [Helm Installer](#)
 - [Java Tool Installer](#)
 - [Kubectl Installer](#)
 - [Node.js Tool Installer](#)
 - [NuGet Tool Installer](#)
 - [Use .NET Core](#)
 - [Use Python Version](#)
 - [Use Ruby Version](#)
 - [Visual Studio Test Platform Installer](#)
- [Troubleshooting](#)
 - [Troubleshoot pipeline runs](#)
 - [Review logs](#)
 - [Debug deployment issues](#)
 - [Troubleshoot Azure connections](#)
- [Reference](#)
- [YAML schema](#)

- [Expressions](#)
- [File matching patterns](#)
- [File and variable transform](#)
- [Logging commands](#)
- [Artifact policy checks](#)
- [Azure CLI](#)
- [Case studies & best practices](#)
 - [Add continuous security validation](#)
 - [Progressively expose releases using deployment rings](#)
 - [Progressively expose features in production](#)
- [Developer resources](#)
 - [REST API reference](#)
 - [Azure DevOps CLI](#)
 - [DevOps Starter to deploy to Azure](#)
 - [Azure Lab Services](#)
 - [Microsoft Learn](#)
 - [Create a build pipeline](#)
 - [Implement a code workflow in your build pipeline by using Git and GitHub](#)
 - [Run quality tests in your build pipeline](#)
 - [Manage build dependencies with Azure Artifacts](#)
 - [Automated testing](#)

Use Azure Pipelines

3/6/2021 • 6 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

Azure Pipelines supports continuous integration (CI) and continuous delivery (CD) to constantly and consistently test and build your code and ship it to any target. You accomplish this by defining a pipeline. You define pipelines using the YAML syntax or through the user interface (Classic).

Azure Pipelines supports continuous integration (CI) and continuous delivery (CD) to constantly and consistently test and build your code and ship it to any target. You accomplish this by defining a pipeline using the user interface, also referred to as *Classic*.

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Automate tests, builds, and delivery

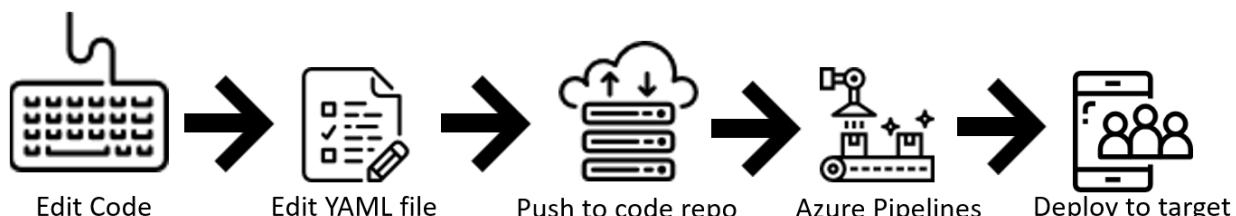
Continuous integration automates tests and builds for your project. CI helps to catch bugs or issues early in the development cycle, when they're easier and faster to fix. Items known as artifacts are produced from CI systems. They're used by the continuous delivery release pipelines to drive automatic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

Continuous delivery automatically deploys and tests code in multiple stages to help drive quality. Continuous integration systems produce deployable artifacts, which include 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)
<ul style="list-style-type: none">- Increase code coverage- Build faster by splitting test and build runs- Automatically ensure you don't ship broken code- Run tests continually.	<ul style="list-style-type: none">- Automatically deploy code to production- Ensure deployment targets have latest code- Use tested code from CI process.

Define pipelines using YAML syntax

You define your pipeline in a YAML file called `azure-pipelines.yml` with the rest of your app.



- The pipeline is versioned with your code. It 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.

Follow these basic steps:

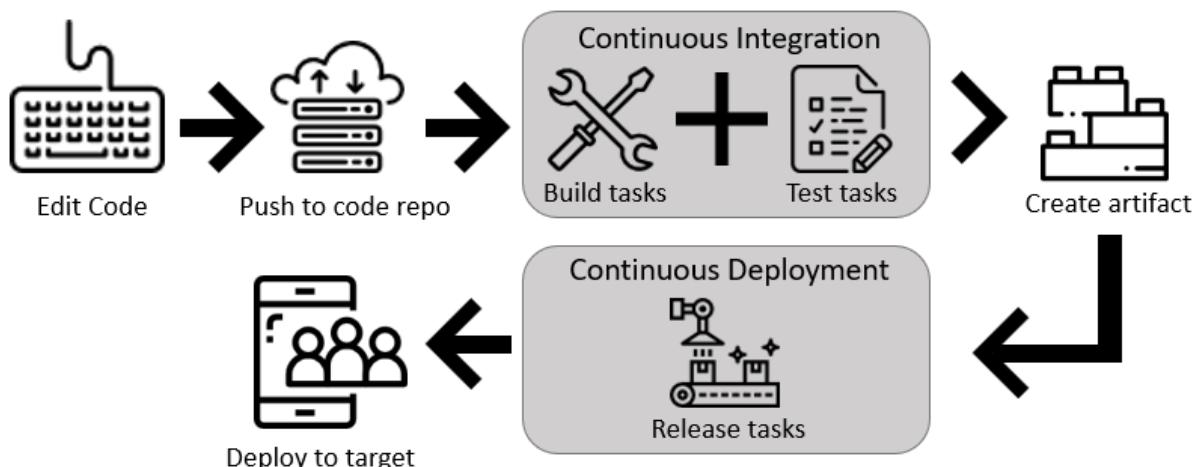
1. Configure Azure Pipelines to use your Git repo.
2. Edit your `azure-pipelines.yml` file to define your build.
3. Push your code to your version control repository. This action kicks off the default trigger to build and deploy and then monitor the results.

Your code is now updated, built, tested, and packaged. It can be deployed to any target.

YAML pipelines aren't available in TFS 2018 and earlier versions.

Define pipelines using the Classic interface

Create and configure pipelines in the Azure DevOps web portal with the Classic user interface editor. 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.



Follow these basic steps:

1. Configure Azure Pipelines to use your Git repo.
2. Use the Azure Pipelines classic editor 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.

The build creates an artifact that's used by the rest of your pipeline to run tasks such as deploying to staging or production.

Your code is now updated, built, tested, and packaged. It can be deployed to any target.

Feature availability

Certain pipeline features are only available when using YAML or when defining build or release pipelines with the Classic interface. The following table indicates which features are supported and for which tasks and methods.

FEATURE	YAML	CLASSIC BUILD	CLASSIC RELEASE	NOTES
---------	------	---------------	-----------------	-------

FEATURE	YAML	CLASSIC BUILD	CLASSIC RELEASE	NOTES
Agents	Yes	Yes	Yes	Specifies a required resource on which the pipeline runs.
Approvals	Yes	No	Yes	Defines a set of validations required prior to completing a deployment stage.
Artifacts	Yes	Yes	Yes	Supports publishing or consuming different package types.
Caching	Yes	Yes	No	Reduces build time by allowing outputs or downloaded dependencies from one run to be reused in later runs. In Preview, available with Azure Pipelines only.
Conditions	Yes	Yes	Yes	Specifies conditions to be met prior to running a job.
Container jobs	Yes	No	No	Specifies jobs to run in a container.
Demands	Yes	Yes	Yes	Ensures pipeline requirements are met before running a pipeline stage. Requires self-hosted agents.
Dependencies	Yes	Yes	Yes	Specifies a requirement that must be met in order to run the next job or stage.
Deployment groups	Yes	No	Yes	Defines a logical set of deployment target machines.
Deployment group jobs	No	No	Yes	Specifies a job to release to a deployment group.
Deployment jobs	Yes	No	No	Defines the deployment steps.

FEATURE	YAML	CLASSIC BUILD	CLASSIC RELEASE	NOTES
Environment	Yes	No	No	Represents a collection of resources targeted for deployment. Available with Azure Pipelines only.
Gates	No	No	Yes	Supports automatic collection and evaluation of external health signals prior to completing a release stage. Available with Classic Release only.
Jobs	Yes	Yes	Yes	Defines the execution sequence of a set of steps.
Service connections	Yes	Yes	Yes	Enables a connection to a remote service that is required to execute tasks in a job.
Service containers	Yes	No	No	Enables you to manage the lifecycle of a containerized service.
Stages	Yes	No	Yes	Organizes jobs within a pipeline.
Task groups	No	Yes	Yes	Encapsulates a sequence of tasks into a single reusable task. If using YAML, see templates.
Tasks	Yes	Yes	Yes	Defines the building blocks that make up a pipeline.
Templates	Yes	No	No	Defines reusable content, logic, and parameters.
Triggers	Yes	Yes	Yes	Defines the event that causes a pipeline to run.
Variables	Yes	Yes	Yes	Represents a value to be replaced by data to pass to the pipeline.

FEATURE	YAML	CLASSIC BUILD	CLASSIC RELEASE	NOTES
Variable groups	Yes	Yes	Yes	Use to store values that you want to control and make available across multiple pipelines.

TFS 2015 through TFS 2018 supports the Classic interface only. The following table indicates which pipeline features are available when defining build or release pipelines.

FEATURE	CLASSIC BUILD	CLASSIC RELEASE	NOTES
Agents	Yes	Yes	Specifies a required resource on which the pipeline runs.
Approvals	No	Yes	Defines a set of validations required prior to completing a deployment stage.
Artifacts	Yes	Yes	Supports publishing or consuming different package types.
Conditions	Yes	Yes	Specifies conditions to be met prior to running a job.
Demands	Yes	Yes	Ensures pipeline requirements are met before running a pipeline stage. Requires self-hosted agents.
Dependencies	Yes	Yes	Specifies a requirement that must be met in order to run the next job or stage.
Deployment groups	No	Yes	Defines a logical set of deployment target machines.
Deployment group jobs	No	Yes	Specifies a job to release to a deployment group.
Jobs	Yes	Yes	Defines the execution sequence of a set of steps.
Service connections	Yes	Yes	Enables a connection to a remote service that is required to execute tasks in a job.
Stages	No	Yes	Organizes jobs within a pipeline.

FEATURE	CLASSIC BUILD	CLASSIC RELEASE	NOTES
Task groups	Yes	Yes	Encapsulates a sequence of tasks into a single reusable task. If using YAML, see templates.
Tasks	Yes	Yes	Defines the building blocks that make up a pipeline.
Triggers	Yes	Yes	Defines the event that causes a pipeline to run.
Variables	Yes	Yes	Represents a value to be replaced by data to pass to the pipeline.
Variable groups	Yes	Yes	Use to store values that you want to control and make available across multiple pipelines.

Try this next

[Create your first pipeline](#)

Related articles

- [Key concepts for new Azure Pipelines users](#)

What is Azure Pipelines?

3/6/2021 • 4 minutes to read • [Edit Online](#)

Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

Azure Pipelines automatically builds and tests code projects to make them available to others. 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.

Continuous Integration (CI) is the practice used by development teams to automate the merging and testing of code. Implementing CI helps to catch bugs early in the development cycle, which makes them less expensive to fix. Automated tests execute as part of the CI process to ensure quality. Artifacts are produced from CI systems and fed to release processes to drive frequent deployments. The Build service in TFS helps you set up and manage CI for your applications.

Continuous Delivery (CD) is a process by which code is built, tested, and deployed to one or more test and production environments. Deploying and testing in multiple environments drives quality. CI systems produce the deployable artifacts including infrastructure and apps. Automated release processes consume these artifacts to release new versions and fixes to existing systems. Monitoring and alerting systems run continually to drive visibility into the entire CD process. The Release service in TFS helps you set up and manage CD for your applications.

Continuous Testing (CT) on-premises or in the cloud is the use of automated build-deploy-test workflows, with a choice of technologies and frameworks, that test your changes continuously in a fast, scalable, and efficient manner.

Version control systems

The starting point for configuring CI and CD for your applications is to have your source code in a version control system. Azure DevOps supports two forms of version control - Git and Team Foundation Version Control (TFVC). The Build service integrates with both of these version control systems. Once you have configured CI, any changes you push to your version control repository will be automatically built and validated.

You can also manage your source code in GitHub, GitHub Enterprise, Bitbucket Cloud, Subversion, or any other Git repository.

Languages

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

Application types

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

To configure CI, you create a pipeline definition. A pipeline definition is a representation of the automation process that you want to run to build and test your application. The automation process is defined as a collection of tasks. Azure DevOps has a number of tasks to build and test your application. For example, tasks exist to build .Net, Java, Node, Android, Xcode, and C++ applications. Similarly, there are tasks to run tests using a number of testing frameworks and services. You can also run command line, PowerShell, or Shell scripts in your automation.

Deployment targets

Use Azure Pipelines to deploy your code to multiple targets. Targets include virtual machines, containers, on-premises and cloud platforms, or PaaS services. You can also publish your mobile application to a store.

Once you have continuous integration in place, the next step is to create a release definition to automate the deployment of your application to one or more environments. This automation process is again defined as a collection of tasks.

Continuous testing

Whether your app is on-premises or in the cloud, you can automate build-deploy-test workflows and choose the technologies and frameworks, then [test your changes continuously](#) in a fast, scalable, and efficient manner.

- Maintain quality and find problems as you develop. Continuous testing with TFS ensures your app still works after every check-in and build, enabling you to find problems earlier by running tests automatically with each build.
- Any test type and any test framework. Choose the test technologies and frameworks you prefer to use.
- Rich analytics and reporting. When your build is done, review your test results to start resolving the problems you find. Rich and actionable build-on-build reports let you instantly see if your builds are getting healthier. But it's not just about speed - detailed and customizable test results measure the quality of your app.

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

Pricing

If you use public projects, Azure Pipelines is free. To learn more, see [What is a public project?](#) 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 Azure Pipelines?

Implementing CI and CD pipelines helps to ensure consistent and quality code that's readily available to users. And, Azure Pipelines provides a quick, easy, and safe way to automate building your projects and making them available to users.

Use Azure Pipelines because it supports the following scenarios:

- Works with any language or platform
- Deploys to different types of targets at the same time
- Integrates with Azure deployments
- Builds on Windows, Linux, or Mac machines
- Integrates with GitHub

- Works with open-source projects.

Next steps

[Use Azure Pipelines](#)

Sign up for Azure Pipelines

4/18/2021 • 4 minutes to read • [Edit Online](#)

Azure Pipelines

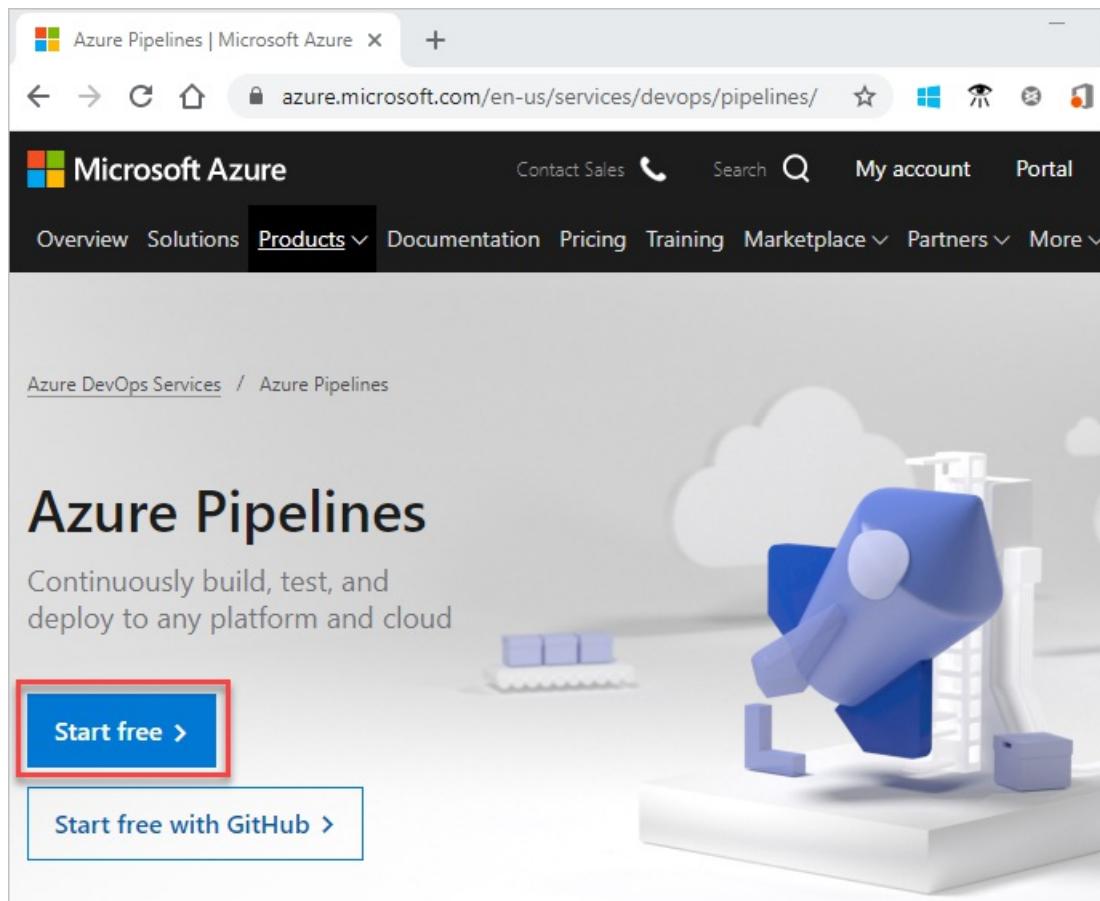
Sign up for an Azure DevOps organization and Azure Pipelines to begin managing CI/CD to deploy your code with high-performance pipelines.

For more information on Azure Pipelines, see [What is Azure Pipelines](#).

Sign up with a personal Microsoft account

If you have a Microsoft account, follow these steps to sign up for Azure Pipelines.

1. Open [Azure Pipelines](#) and choose **Start free**.



2. Enter your email address, phone number, or Skype ID for your Microsoft account. If you're a Visual Studio subscriber and you get Azure DevOps as a benefit, use the Microsoft account associated with your subscription. Select **Next**.



Sign in

Email, phone, or Skype

No account? [Create one!](#)

[Can't access your account?](#)

[Next](#)



[Sign in with GitHub](#)

3. Enter your password and select Sign in.



fabrikamfiber4@hotmail.com

Enter password

.....

Keep me signed in

[Forgot my password](#)

[Sign in with a different Microsoft account](#)

[Sign in](#)

4. To get started with Azure Pipelines, select Continue.



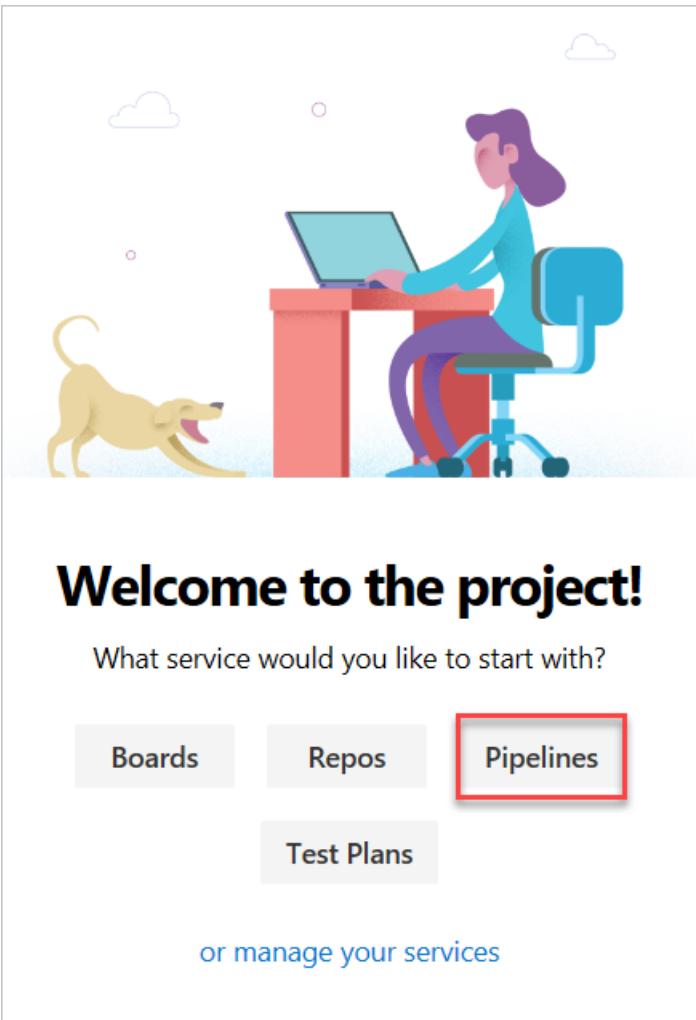
fabrikamfiber4@hotmail.com

Get started with Azure DevOps

Choosing **Continue** means that you agree to our [Terms of Service](#), [Privacy Statement](#), and [Code of Conduct](#).

[Continue](#)

5. Select the service you'd like to start your project with, **Pipelines**.



Welcome to the project!

What service would you like to start with?

Boards

Repos

Pipelines

Test Plans

or manage your services

An organization is created based on the account you used to sign in. Use the following URL to sign in to your organization at any time:

<https://dev.azure.com/{yourorganization}>

Your next step is to [create a project](#).

Sign up with a GitHub account

If you have a GitHub account, follow these steps to sign up for Azure Pipelines.

IMPORTANT

If your GitHub email address is associated with an Azure AD-backed organization in Azure DevOps, you can't sign in with your GitHub account, rather you must sign in with your Azure AD account.

1. Choose **Start free with GitHub**. If you're already part of an Azure DevOps organization, choose **Start free**.

Azure Pipelines | Microsoft Azure

Contact Sales Search My account Portal

Overview Solutions Products Documentation Pricing Training Marketplace Partners More

Azure DevOps Services / Azure Pipelines

Azure Pipelines

Continuously build, test, and deploy to any platform and cloud

[Start free >](#)

[Start free with GitHub >](#)

The screenshot shows the Azure Pipelines homepage. A red box highlights the 'Start free with GitHub >' button, which is located below the main 'Start free >' button. The background features a 3D illustration of a blue robotic arm working on a white surface, with a cloud and server racks in the background.

2. Enter your GitHub account credentials, and then select Sign in.

Sign in to GitHub
to continue to Microsoft-
Corporation

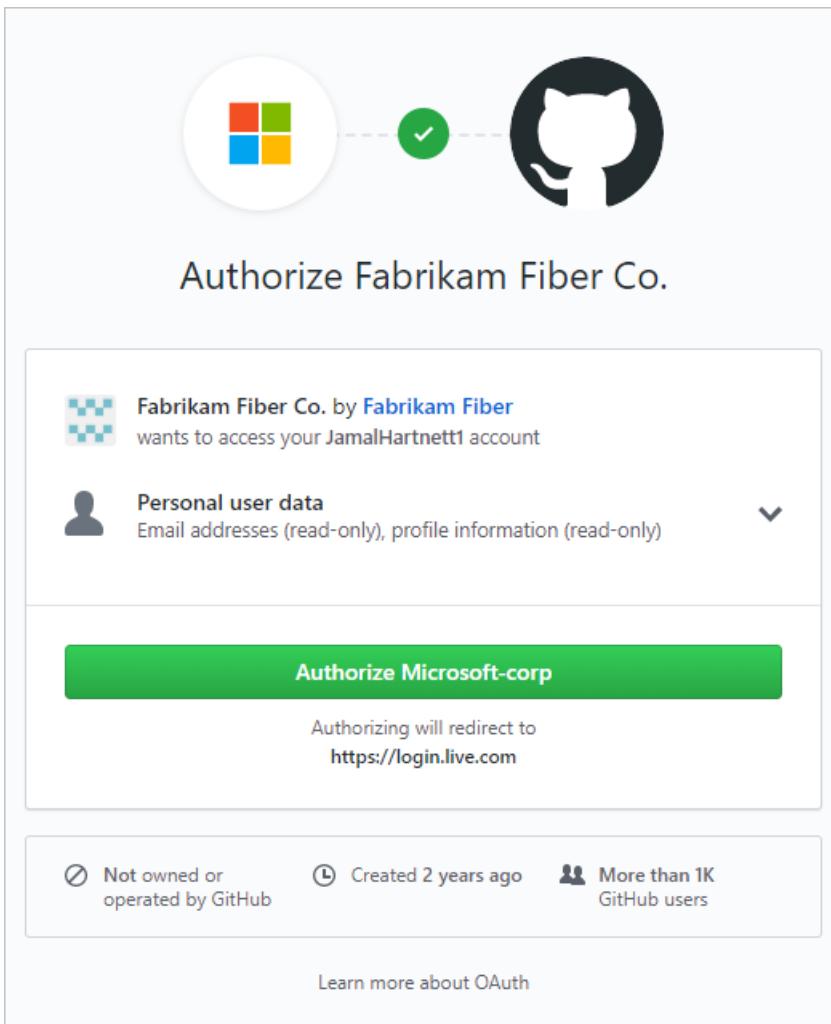
Username or email address

Password [Forgot password?](#)

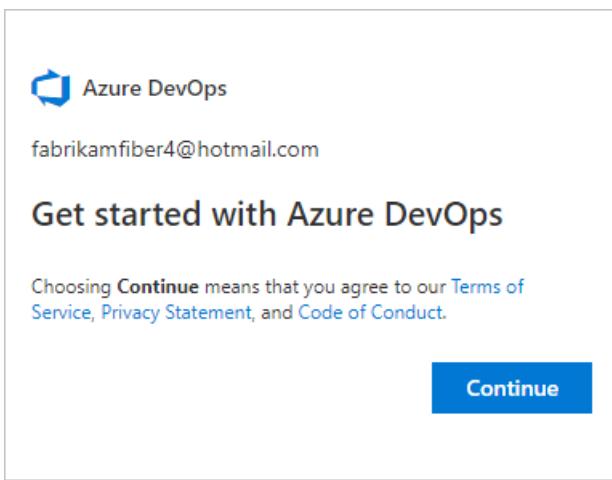
[Sign in](#)

The screenshot shows the GitHub sign-in page. It features a large circular Microsoft logo at the top. Below it, the text 'Sign in to GitHub to continue to Microsoft-Corporation'. There are two input fields: one for 'Username or email address' and one for 'Password', with a 'Forgot password?' link next to it. At the bottom is a green 'Sign in' button.

3. Select Authorize Microsoft-corp.



4. Choose **Continue**.



An organization gets created based on the account you used to sign in. Use the following URL to sign in to your organization at any time: <https://dev.azure.com/{yourorganization}>

For more information about GitHub authentication, see [FAQs](#).

Your next step is to [create a project](#).

Create a project

If you signed up for Azure DevOps with an existing MSA or GitHub identity, you're automatically prompted to create a project. Create either a public or private project. To learn more about public projects, see [What is a public project?](#).

1. Enter a name for your project, select the visibility, and optionally provide a description. Then choose **Create project**.

Create a project to get started

Project name *

Fabrikam



Description

Azure Boards to track issues

Visibility



Public

Anyone on the internet can view the project. Certain features like TFVC are not supported.



Private

Only people you give access to will be able to view this project.

+ Create project

Special characters aren't allowed in the project name (such as / : \ ~ & % ; @ ' " ? < > | # \$ * } { , + = []). The project name also can't begin with an underscore, can't begin or end with a period, and must be 64 characters or less. Set your project visibility to either public or private. Public visibility allows for anyone on the internet to view your project. Private visibility is for only people who you give access to your project.

2. When your project is created, the Kanban board automatically appears.

You're now set to [create your first pipeline](#), or [invite other users](#) to collaborate with your project.

Invite team members

You can add and invite others to work on your project by adding their email address to your organization and project.

1. From your project web portal, choose the Azure DevOps icon, and then select **Organization settings**.

The screenshot shows the Azure DevOps Home page. On the left sidebar, under 'My organizations', several projects are listed: fabrikam--fiber, fabrikamfabulous, fabrikamfib, FabrikamFiber (which is selected), and fabrikaminthecloud. Below this, there are links for 'Documentation', 'Get help', and '+ New organization'. At the bottom of the sidebar, the 'Organization settings' option is highlighted with a red box. The main content area displays the 'FabrikamFiber' organization with a project named 'FabrikamFiberTest' shown below it.

2. Select Users > Add users.

The screenshot shows the 'Organization Settings' page for the 'fabrikamfiber4' organization. The left sidebar lists various settings sections: General, Overview, Projects, **Users** (which is selected and highlighted with a red box), Billing, Auditing, Global notifications, Usage, Extensions, Azure Active Directory, Security, Policies, Permissions, and Boards. The main content area is titled 'Users' and shows a list of users. It includes a search bar, filter options, and buttons for 'All users' and 'Group rules'. A blue button labeled 'Add users' is highlighted with a red box. The user list shows one entry: 'Josh Hartnett' with the email 'FabrikamFiber4@hotmail.com' and an 'Access Level' of 'Basic'.

3. Complete the form by entering or selecting the following information:

- **Users:** Enter the email addresses (Microsoft accounts) or GitHub IDs for the users. You can add several email addresses by separating them with a semicolon (;). An email address appears in red when it's accepted.

- **Access level:** Assign one of the following access levels:
 - **Basic:** Assign to users who must have access to all Azure Pipelines features. You can grant up to five users Basic access for free.
 - **Stakeholder:** Assign to users for limited access to features to view, add, and modify work items. You can assign an unlimited amount of users Stakeholder access for free.
- **Add to project:** Select the project you named in the preceding procedure.
- **Azure DevOps groups:** Select one of the following security groups, which will determine the permissions the users have to do select tasks. To learn more, see [Azure Pipelines resources](#).
 - **Project Readers:** Assign to users who only require read-only access.
 - **Project Contributors:** Assign to users who will contribute fully to the project.
 - **Project Administrators:** Assign to users who will configure project resources.

NOTE

Add email addresses for [personal Microsoft accounts](#) and IDs for GitHub accounts unless you plan to use [Azure Active Directory \(Azure AD\)](#) to authenticate users and control organization access. If a user doesn't have a Microsoft or GitHub account, ask the user to [sign up](#) for a Microsoft account or a GitHub account.

4. When you're done, select **Add** to complete your invitation.

For more information, see [Add organization users for Azure DevOps Services](#).

Change organization or project settings

You can rename and delete your organization, or change the organization location. To learn more, see the following articles:

- [Manage organizations](#)
- [Rename an organization](#)
- [Change the location of your organization](#)

You can rename your project or change its visibility. To learn more about managing projects, see the following articles:

- [Manage projects](#)
- [Rename a project](#)
- [Change the project visibility, public or private](#)

Next steps

[Create your first pipeline](#)

Related articles

- [What is Azure Pipelines?](#)
- [Key concepts for new Azure Pipelines users](#)
- [Create your first pipeline](#)

Create your first pipeline

5/27/2021 • 32 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

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

Prerequisites - Azure DevOps

- A GitHub account, where you can create a repository. If you don't have one, you can [create one for free](#).
- An Azure DevOps organization. If you don't have one, you can [create one for free](#). (An Azure DevOps organization is different from your GitHub organization. Give them the same name if you want alignment between them.)

If your team already has one, then make sure you're an administrator of the Azure DevOps project that you want to use.

- An ability to run pipelines on Microsoft-hosted agents. You can either purchase a [parallel job](#) or you can request a free tier. To request a free tier, follow the instructions in [this article](#). Please note that it may take us 2-3 business days to grant the free tier.

Create your first pipeline

- [Java](#)
- [.NET](#)
- [Python](#)
- [JavaScript](#)
- [Azure CLI \(Java\)](#)

Get the Java sample code

To get started, fork the following repository into your GitHub account.

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

Create your first Java pipeline

1. Sign in to your Azure DevOps organization and navigate to your project.
2. In your project, navigate to the [Pipelines](#) page. Then choose the action to create a new pipeline.
3. Walk through the steps of the wizard by first selecting **GitHub** as the location of your source code.
4. You might be redirected to GitHub to sign in. If so, enter your GitHub credentials.
5. When the list of repositories appears, select your desired sample app repository.
6. Azure Pipelines will analyze your repository and recommend a Maven pipeline template. Select **Save and run**, then select **Commit directly to the main branch**, and then choose **Save and run again**.
7. A new run is started. Wait for the run to finish.

Learn more about [working with Java](#) in your pipeline.

Add a status badge to your repository

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



To copy the status badge to your clipboard:

1. In Azure Pipelines, go to the **Pipelines** page to view the list of pipelines. Select the pipeline you created in the previous section.
2. In the context menu for the pipeline, select **Status badge**.
3. Copy the sample Markdown from the status badge panel.

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

1. Go to the list of files and select `Readme.md`. Select the pencil icon to edit.
2. Paste the status badge Markdown at the beginning of the file.
3. Commit the change to the `master` branch.
4. Notice that the status badge appears in the description of your repository.

To configure anonymous access to badges for private projects:

1. Navigate to **Project Settings**
2. Open the **Settings** tab under **Pipelines**
3. Toggle the **Disable anonymous access to badges** slider under **General**

NOTE

Even in a private project, anonymous badge access is enabled by default. With anonymous badge access enabled, users outside your organization might be able to query information such as project names, branch names, job names, and build status through the badge status API.

Because you just changed the `Readme.md` file in this repository, Azure Pipelines automatically builds your code, according to the configuration in the `azure-pipelines.yml` file at the root of your repository. Back in Azure Pipelines, observe that a new run appears. Each time you make an edit, Azure Pipelines starts a new run.

Manage your pipeline with Azure CLI

You can manage the pipelines in your organization using these `az pipelines` commands:

- `az pipelines run`: Run an existing pipeline
- `az pipelines update`: Update an existing pipeline
- `az pipelines show`: Show the details of an existing pipeline

These commands require either the name or ID of the pipeline you want to manage. You can get the ID of a pipeline using the `az pipelines list` command.

Run a pipeline

You can queue (run) an existing pipeline with the `az pipelines run` command. To get started, see [Get started with Azure DevOps CLI](#).

```
az pipelines run [--branch]
    [--commit-id]
    [--folder-path]
    [--id]
    [--name]
    [--open]
    [--org]
    [--project]
    [--variables]
```

Parameters

- **branch**: Name of the branch on which the pipeline run is to be queued, for example, `refs/heads/main`.
- **commit-id**: Commit-id on which the pipeline run is to be queued.
- **folder-path**: Folder path of pipeline. Default is root level folder.
- **id**: Required if **name** is not supplied. ID of the pipeline to queue.
- **name**: Required if ID is not supplied, but ignored if ID is supplied. Name of the pipeline to queue.
- **open**: Open the pipeline results page in your web browser.
- **org**: Azure DevOps organization URL. You can configure the default organization using
`az devops configure -d organization=ORG_URL`. Required if not configured as default or picked up using
`git config`. Example: `--org https://dev.azure.com/MyOrganizationName/`.
- **project**: Name or ID of the project. You can configure the default project using
`az devops configure -d project=NAME_OR_ID`. Required if not configured as default or picked up using
`git config`.
- **variables**: Space separated "name=value" pairs for the variables you would like to set.

Example

The following command runs the pipeline named **myGithubname.pipelines-java** in the branch **pipeline** and shows the result in table format.

```
az pipelines run --name myGithubname.pipelines-java --branch pipeline --output table

Run ID      Number      Status      Result      Pipeline ID      Pipeline Name      Source Branch
Queued Time          Reason
-----  -----  -----  -----  -----  -----
-----  -----  -----  -----  -----  -----
123        20200123.2  notStarted      12      myGithubname.pipelines-java  pipeline
2020-01-23 11:55:56.633450  manual
```

Update a pipeline

You can update an existing pipeline with the [az pipelines update](#) command. To get started, see [Get started with Azure DevOps CLI](#).

```
az pipelines update [--branch]
    [--description]
    [--id]
    [--name]
    [--new-folder-path]
    [--new-name]
    [--org]
    [--project]
    [--queue-id]
    [--yaml-path]
```

Parameters

- **branch**: Name of the branch on which the pipeline run is to be configured, for example, `refs/heads/main`.

- **description:** New description for the pipeline.
- **id:** Required if **name** is not supplied. ID of the pipeline to update.
- **name:** Required if ID is not supplied. Name of the pipeline to update.
- **new-folder-path:** New full path of the folder to which the pipeline is moved, for example, *user1/production_pipelines*.
- **new-name:** New updated name of the pipeline.
- **org:** Azure DevOps organization URL. You can configure the default organization using
`az devops configure -d organization=ORG_URL`. Required if not configured as default or picked up using
`git config`. Example: `--org https://dev.azure.com/MyOrganizationName/`.
- **project:** Name or ID of the project. You can configure the default project using
`az devops configure -d project=NAME_OR_ID`. Required if not configured as default or picked up using
`git config`.
- **queue-id:** Queue ID of the agent pool where the pipeline needs to run.
- **yaml-path:** Path of the pipeline's yaml file in the repo.

Example

The following command updates the pipeline with the **ID** of 12 with a new name and description and shows the result in table format.

```
az pipelines update --id 12 --description "rename pipeline" --new-name updatedname.pipelines-java --output table

ID      Name           Status    Default Queue
----  -----
12     updatedname.pipelines-java  enabled   Hosted Ubuntu 1604
```

Show pipeline

You can view the details of an existing pipeline with the [az pipelines show](#) command. To get started, see [Get started with Azure DevOps CLI](#).

```
az pipelines show [--folder-path]
                  [--id]
                  [--name]
                  [--open]
                  [--org]
                  [--project]
```

Parameters

- **folder-path:** Folder path of pipeline. Default is root level folder.
- **id:** Required if **name** is not supplied. ID of the pipeline to show details.
- **name:** Required if **name** is not supplied, but ignored if ID is supplied. Name of the pipeline to show details.
- **open:** Open the pipeline summary page in your web browser.
- **org:** Azure DevOps organization URL. You can configure the default organization using
`az devops configure -d organization=ORG_URL`. Required if not configured as default or picked up using
`git config`. Example: `--org https://dev.azure.com/MyOrganizationName/`.
- **project:** Name or ID of the project. You can configure the default project using
`az devops configure -d project=NAME_OR_ID`. Required if not configured as default or picked up using
`git config`.

Example

The following command shows the details of the pipeline with the **ID** of 12 and returns the result in table format.

```
az pipelines show --id 12 --output table
```

ID	Name	Status	Default Queue
12	updatedname.pipelines-java	enabled	Hosted Ubuntu 1604

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

NOTE

This guidance applies to TFS version 2017.3 and newer.

We'll show you how to use the classic editor in Azure DevOps Server 2019 to create a build and release that prints "Hello world".

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

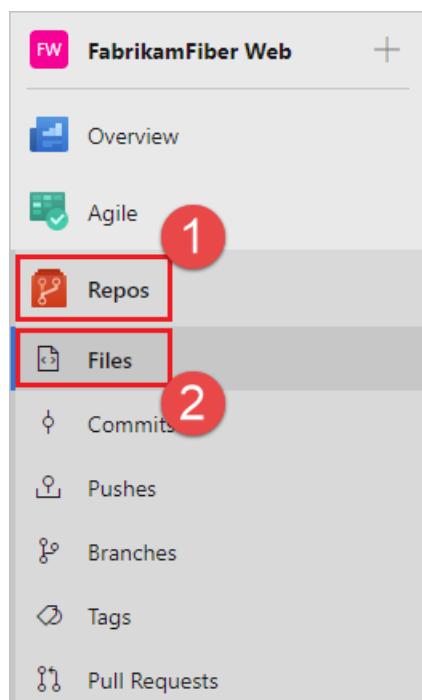
Prerequisites

- A [self-hosted Windows agent](#).

Initialize your repository

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

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

Add a .gitignore: None

Initialize

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

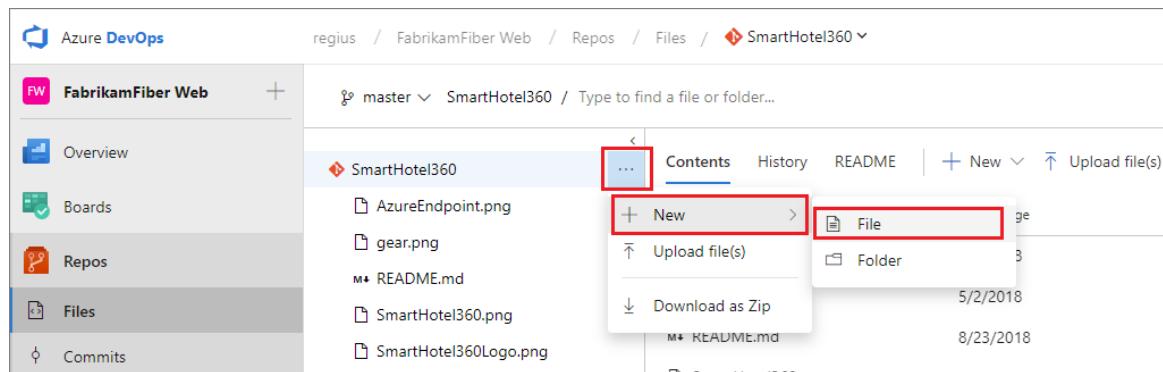
Initialize

Add a script to your repository

Create a PowerShell script that prints `Hello world`.

1. Go to Azure Repos.

2. Add a file.



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

```
HelloWorld.ps1
```

4. Copy and paste this script.

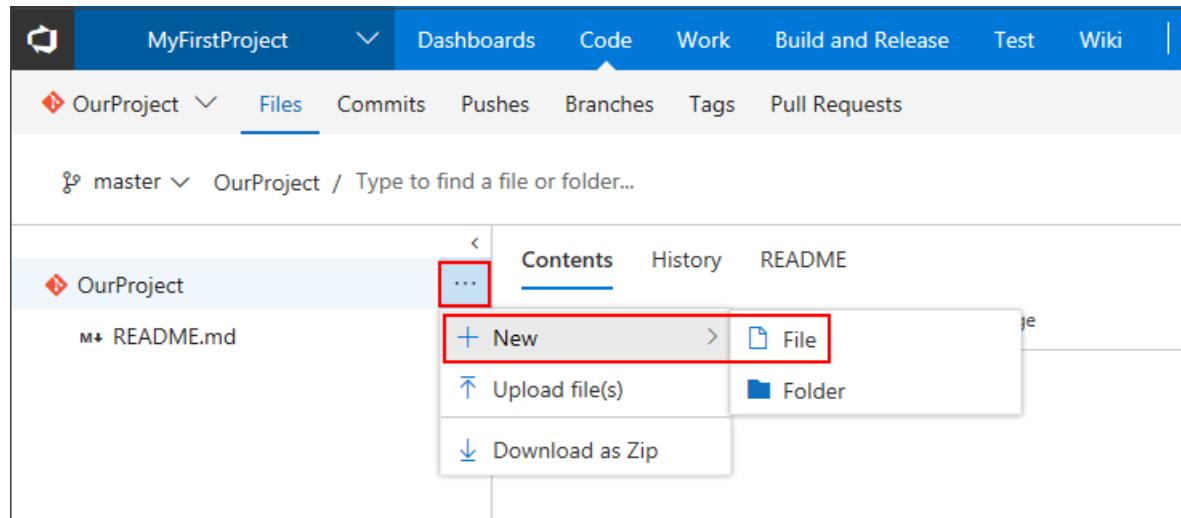
```
Write-Host "Hello world"
```

5. Commit (save) the file.

1. Go to the Code hub.

2. Add a file.

- [TFS 2018.2](#)
- [TFS 2018 RTM](#)



The screenshot shows the 'OurProject' repository in the 'Files' tab. A context menu is open over a file named 'README.md'. The 'Add file' option is highlighted with a red box.

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.

The screenshot shows the 'FabrikamFiber Web' project's 'Builds' page. The left sidebar has a 'Pipelines' link highlighted with a red box. On the right, there is a list of pipelines, with one named 'PublicWebCI' shown.

2. Create a new pipeline.

The screenshot shows the Azure DevOps interface for a project named 'FabrikamFiber Web'. On the left, there's a sidebar with links for Overview, Boards, Repos, Pipelines (which is selected), and Builds. The main area has a search bar at the top. Below it, there's a button labeled '+ New' with a dropdown menu. The menu item 'New build pipeline' is highlighted with a red box. Other items in the dropdown include 'PublicWe' (with a red X), 'master', 'Import a pipeline', and 'Com'.

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

The screenshot shows the user profile settings page. At the top right, there's a green circular icon with 'EB' in white. Below it is a list of settings: My profile, Security, Usage, Notification settings, Preview features (which is highlighted with a red box), Theme, Help, and Sign out.

New YAML pipeline creation experience Off
Enables 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**.

Pipeline
Build pipeline

Get sources SmartHotel360 master

Agent job 1 Run on agent

Add a task to Agent job 1

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.

Tasks Variables Triggers Options Retention History | Save & queue Discard Summary Queue ...

Pipeline Build pipeline

Get sources SmartHotel360 master

Agent job 1 Run on agent

PowerShell Script PowerShell

Display name * PowerShell Script

Type File Path (radio button selected)

Script Path * HelloWorld.ps1

9. Select **Save & queue**, and then select **Save**.
10. Select **Build and Release**, and then choose **Builds**.

OurProject Dashboards Code Work ... Search wiki in this project

OurProject Files Commits Pushes Branches Tags Build and Release Builds Test Wiki

master OurProject / Type to find a file or folder...

OurProject Contents History README

HelloWorld.ps1 Name ↑ Last change

HelloWorld.ps1 9 minutes ago

README.md 17 minutes ago

11. Create a new pipeline.

OurProject Dashboards Build and Release ... Search wiki in this project

Builds Releases Library Task Groups Deployment Groups

Build Definitions Build ID or build number + New + Import

Mine Definitions Queued

12. Start with an **empty pipeline**
13. Select **Pipeline** and specify whatever **Name** you want to use. For the **Agent pool**, select **Default**.
14. 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 'Process' build definition in Azure Pipelines. The 'Tasks' tab is selected. Under the 'Get sources' task, there is a 'PowerShell Script' task. A red box highlights the '+ Add Task' button at the bottom of the list.

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

The screenshot shows the 'Process' build definition with the 'PowerShell Script' task selected. The 'Type' is set to 'File Path' and the 'Script Path' is set to 'HelloWorld.ps1'. A red box highlights the 'HelloWorld.ps1' input field.

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

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

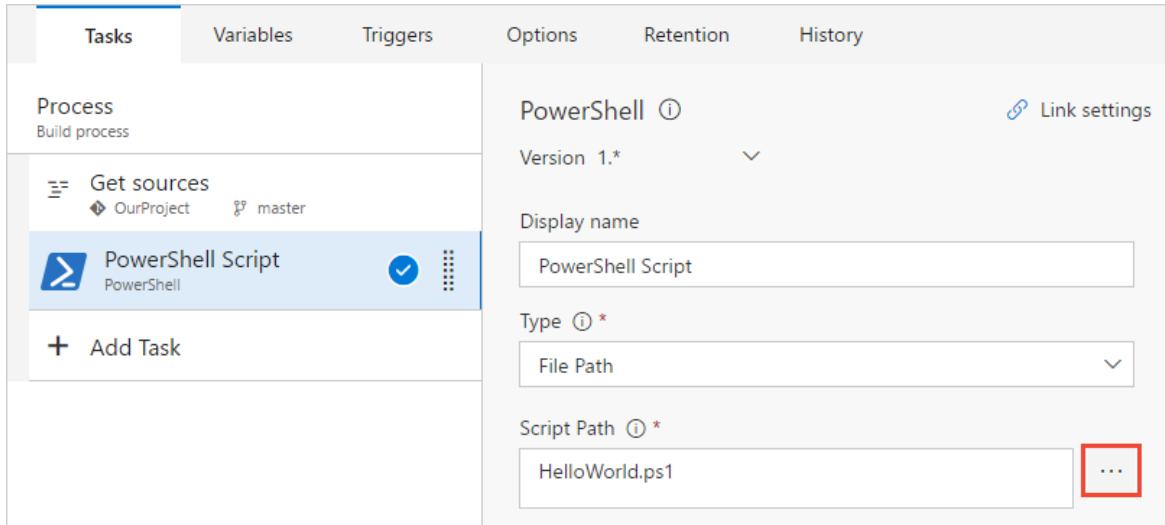
The screenshot shows the 'OurProject' dashboard. The 'Builds' tab is selected in the top navigation bar. A red box highlights the 'Builds' link in the dropdown menu under the 'Build & Release' section.

2. Create a new pipeline.

The screenshot shows the 'Build & Release' dashboard. The 'Build Definitions' section is visible. A red box highlights the '+ New' button at the bottom right of the section.

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.



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 Azure DevOps pipeline editor. On the left, there's a list of tasks: Process (Build process), Get sources (OurProject, master), Phase 1 (Run on agent), PowerShell Script (PowerShell), and Publish Artifact: drop (Publish Build Artifacts). The Publish Artifact task is selected and highlighted with a red box. On the right, the task configuration pane 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 DevOps pipeline editor. On the left, there's a list of tasks: Process (Build process), Get sources (OurProject, master), PowerShell Script (PowerShell), and Publish Artifact: drop (Publish Build Artifacts). The Publish Artifact task is selected and highlighted with a red box. On the right, the task configuration pane shows the following fields:

- Path to Publish**: HelloWorld.ps1
- Artifact Name**: drop
- Artifact Type**: Server

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

Artifact Name: Enter `drop`.

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.

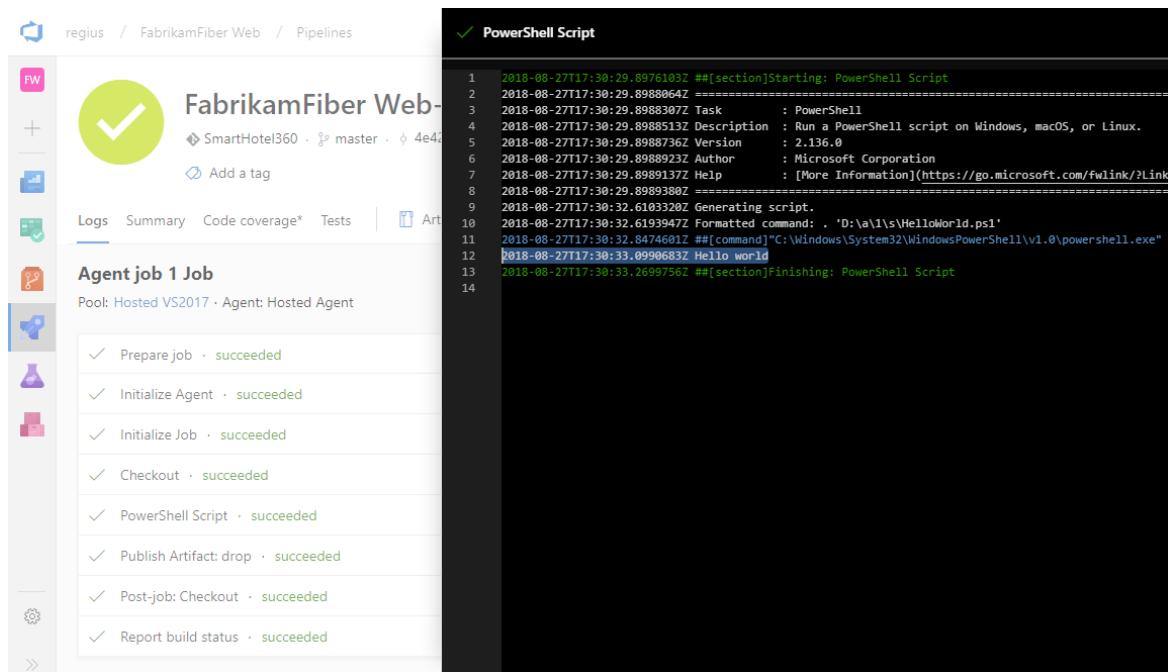
1. Select **Save & queue**, and then select **Save & queue**.
2. On the dialog box, select **Save & queue** once more.

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.



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.



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

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

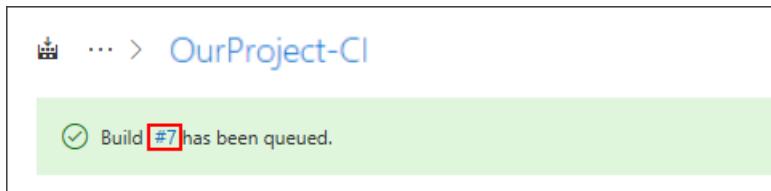
drop File container

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.



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.

- TFS 2018.2
- TFS 2018 RTM

Screenshot of the Azure Pipelines build summary page for Build 1722. The build status is "Build succeeded". The log output shows the PowerShell script running successfully.

```
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 build summary page for Build 1722. The build status is "Build succeeded".

Build 1722 / Build 1722 / Build

Build succeeded

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

Screenshot of the Azure Pipelines build summary page for Build 1722. The build status is "Build succeeded". The "Artifacts" tab is selected, showing a list of artifacts. The "Explore" button is highlighted with a red box.

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 status is 'Build succeeded'. The log output shows the PowerShell command 'git checkout -b <new-branch-name>' and the output 'Hello world'.

```
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
```

3. Go to the build summary.

The screenshot shows the Azure DevOps interface for a build named 'Build 1722'. The build status is 'Build succeeded'. The build ID 'Build 1722' is highlighted with a red box 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, there's a green bar indicating "Build succeeded". Below it, the build summary for "Build 1722" is shown, stating it ran for 31 seconds on a Hosted agent and completed 15.4 minutes ago. The "Artifacts" tab is selected, highlighted with a red box. Below the tabs, there's a list of artifacts with a "drop" folder expanded. A "Download" button and an "Explore" button are available. An "Explore" button is also highlighted with a red box. A modal window titled "Artifacts Explorer" is open, showing the contents of the "drop" folder, which includes a file named "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.

The screenshot shows the Azure DevOps Pipeline editor for a pipeline named "FabrikamFiber Web-Cl". The "Tasks" tab is selected. The pipeline consists of three tasks: "Get sources" (using SmartHotel360 repository and master branch), "Agent job 1" (which contains a "PowerShell Script" task), and "Publish Artifact: drop" (Publish Build Artifacts). The "PowerShell Script" task is currently selected. Its configuration pane shows the following details:

- PowerShell**: Version 2.*
- Display name**: PowerShell Script
- Type**: File Path (selected)
- Script Path**: HelloWorld.ps1
- Arguments**: `-greeter "$(Build.RequestedFor)" -trigger "$(Build.Reason)"`

- TFS 2018.2
- TFS 2018 RTM

The screenshot shows the Azure DevOps build pipeline editor. On the left, there's a list of tasks: Process (Build process), Get sources (OurProject master), Phase 1 (Run on agent), PowerShell Script (PowerShell), and Publish Artifact: drop (Publish Build Artifacts). The PowerShell Script task is selected. On the right, the task configuration pane is open for the PowerShell Script task. It shows the following settings:

- PowerShell**: Version 1.*
- Display name**: PowerShell Script
- Type**: File Path
- Script Path**: HelloWorld.ps1
- Arguments**: `-greeter "$(Build.RequestedFor)" -trigger "$(Build.Reason)"`

The Arguments field is highlighted with a red box.

This screenshot is similar to the one above, showing the build pipeline editor with the same tasks listed on the left. The PowerShell Script task is selected on the right, with its configuration pane open. The arguments are identical to the previous screenshot:

- PowerShell**: Version 1.*
- Display name**: PowerShell Script
- Type**: File Path
- Script Path**: HelloWorld.ps1
- Arguments**: `-greeter "$(Build.RequestedFor)" -trigger "$(Build.Reason)"`

The Arguments field is highlighted with a red box.

Arguments

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

Finally, save the build pipeline.

Next you'll add the arguments to your script.

1. Go to your **Files** in Azure Repos (the **Code** hub in the previous navigation and TFS).
2. Select the **HelloWorld.ps1** file, and then **Edit** the file.

3. Change the script as follows:

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

4. **Commit** (save) the script.

Now you can see the results of your changes. 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.

Now you can see the results of your changes. 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.

The screenshot shows the Azure DevOps interface. On the left, there's a sidebar with 'FabrikamFiber' selected. Under 'Pipelines', 'Agent job 1 Job' is listed. The main area shows a log titled 'PowerShell Script'. The log output is as follows:

```
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 2018-08-30T17:33:29.1730295Z Author : Microsoft Corporation  
7 2018-08-30T17:33:29.1730373Z Help : [More Information](https://go.microsoft.com/fwlink/?LinkID=)  
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 2018-08-30T17:33:31.3101400Z Hello world from Elijah Batkoski  
12 2018-08-30T17:33:31.3107687Z Trigger: Manual  
13 2018-08-30T17:33:31.3107687Z Trigger: Manual  
14 2018-08-30T17:33:31.4515161Z ##[section]Finishing: PowerShell Script  
15
```

The screenshot shows the Azure DevOps interface. On the left, 'Build 1723' is selected. In the center, it says 'Hello world / Build 1723 / Build / PowerShell Script' and 'Build succeeded'. Below that, it says 'PowerShell Script' and 'Ran for 1 seconds (Hosted Agent), completed 74 seconds'. The 'Logs' section shows the following output:

```
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://)  
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 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.

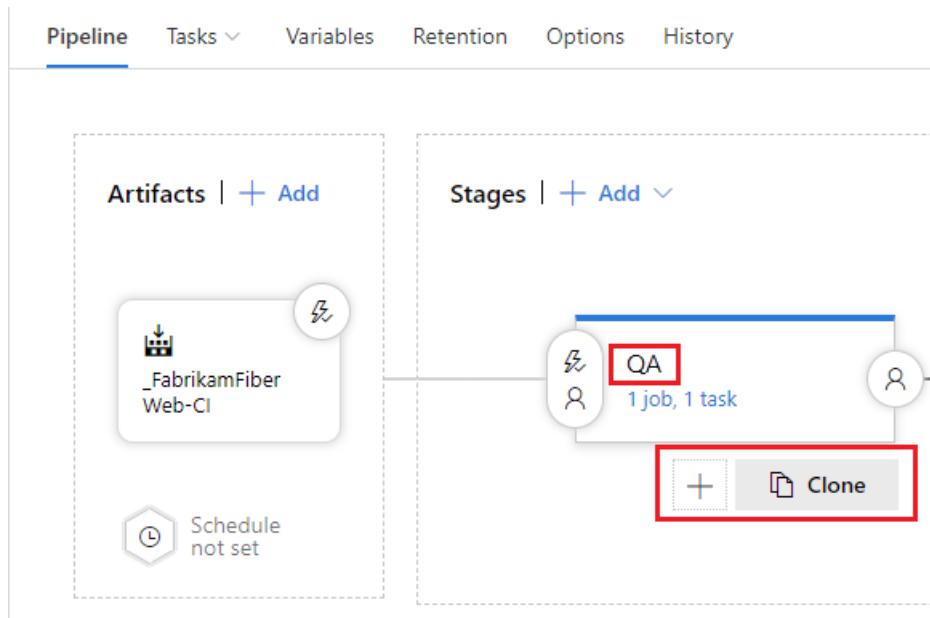
1. Go to the **Pipelines** tab, and then select **Releases**.
2. Select the action to create a **New pipeline**. If a release pipeline is already created, select the plus sign (+) and then select **Create a release pipeline**.
3. Select the action to start with an **Empty job**.
4. Name the stage **QA**.
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.

The screenshot shows the 'New release pipeline' configuration page. At the top, there are tabs for Pipeline, Tasks, Variables, Retention, Options, and History. The Pipeline tab is selected. On the left, under 'Artifacts', there is a single item: 'FabrikamFiber Web-Cl' with a 'Schedule not set' button below it. On the right, under 'Stages', there is one stage named 'QA' which contains '1 job, 0 task'. To the right of the stages, the 'Continuous deployment trigger' section is expanded, showing a toggle switch labeled 'Enabled' with the description 'Creates a release every time a new build is available.' A red box highlights this toggle switch. Below it, 'Build branch filters' are listed as 'No filters added.' with a '+ Add' button. Further down, the 'Pull request trigger' section is shown with a disabled toggle switch.

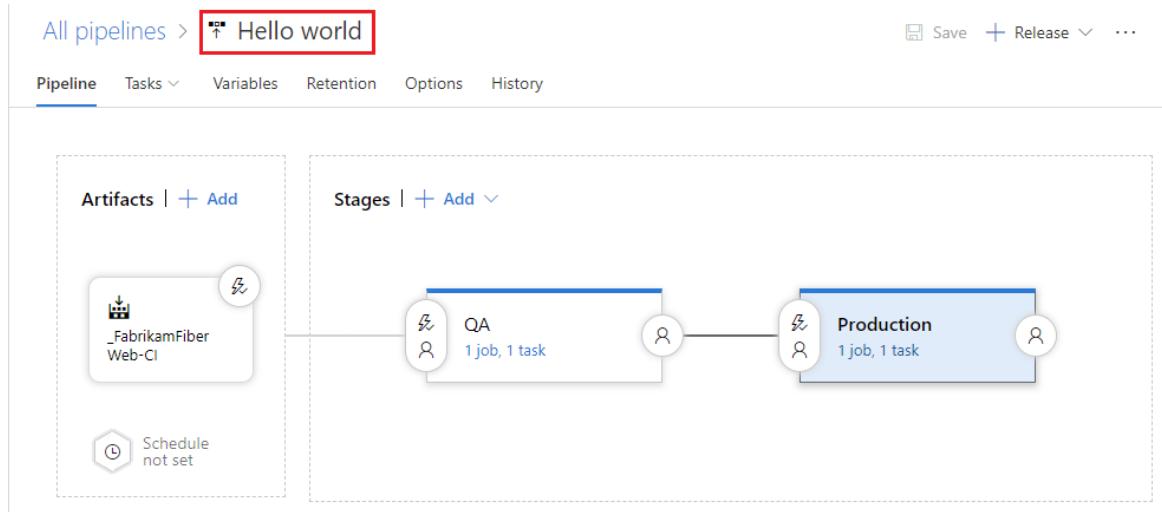
7. Select the **Tasks** tab and select your **QA** stage.
8. Select the plus sign (+) for the job to add a task to the job.
9. On the **Add tasks** dialog box, select **Utility**, locate the **PowerShell** task, and then select its **Add** button.
10. On the left side, select your new **PowerShell**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



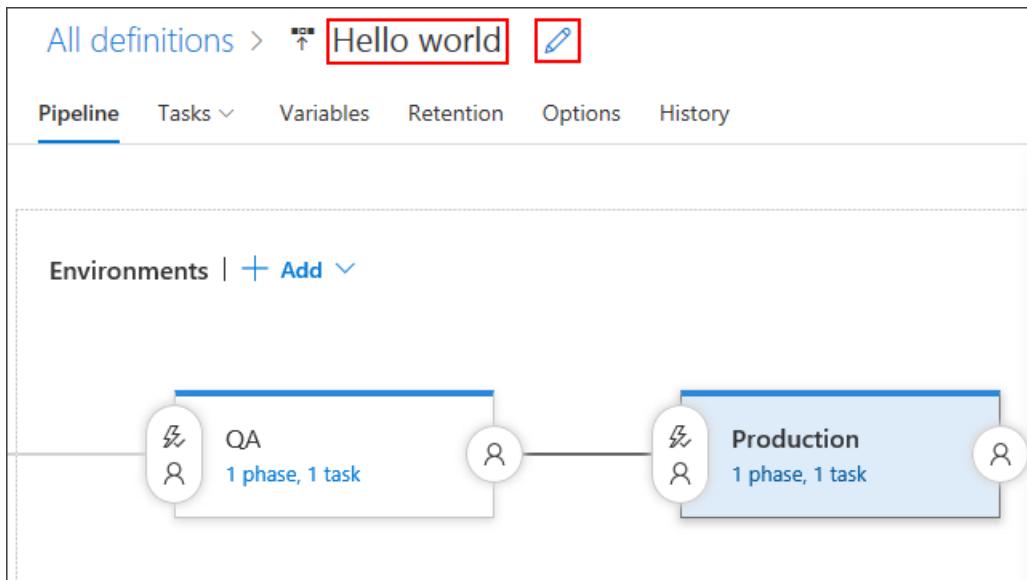
16. Save the release pipeline.
1. Go to the **Build and Release** tab, and then select **Releases**.
2. Select the action to create a **New pipeline**. If a release pipeline is already created, select the plus sign (+) and then select **Create a release definition**.
3. Select the action to start with 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.
 - TFS 2018.2
 - TFS 2018 RTM

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.

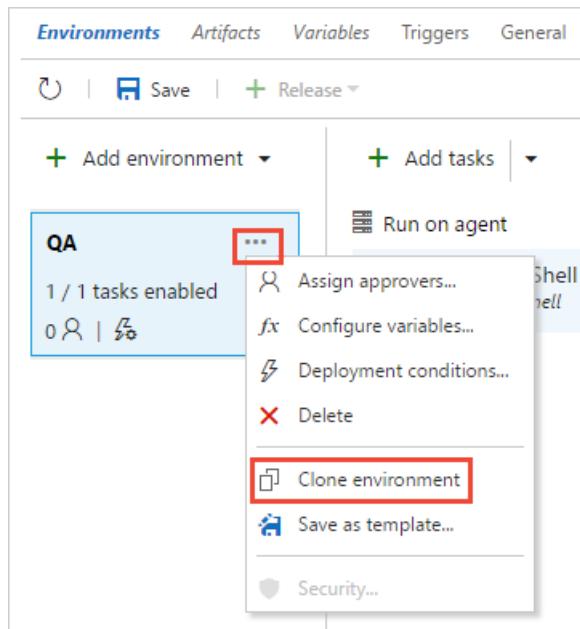
1. Go to **Azure Pipelines**, and then to the **Releases** tab.
2. Select the action to create a **New pipeline**.
3. On the dialog box, select the **Empty** template and select **Next**.
4. Make sure that your **Hello world** build pipeline that you created above is selected. Select **Continuous deployment**, 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 'New Empty Definition' release pipeline. The 'QA' stage has been renamed to 'New Empty Definition'. The stage contains a single PowerShell task named 'PowerShell Script'.

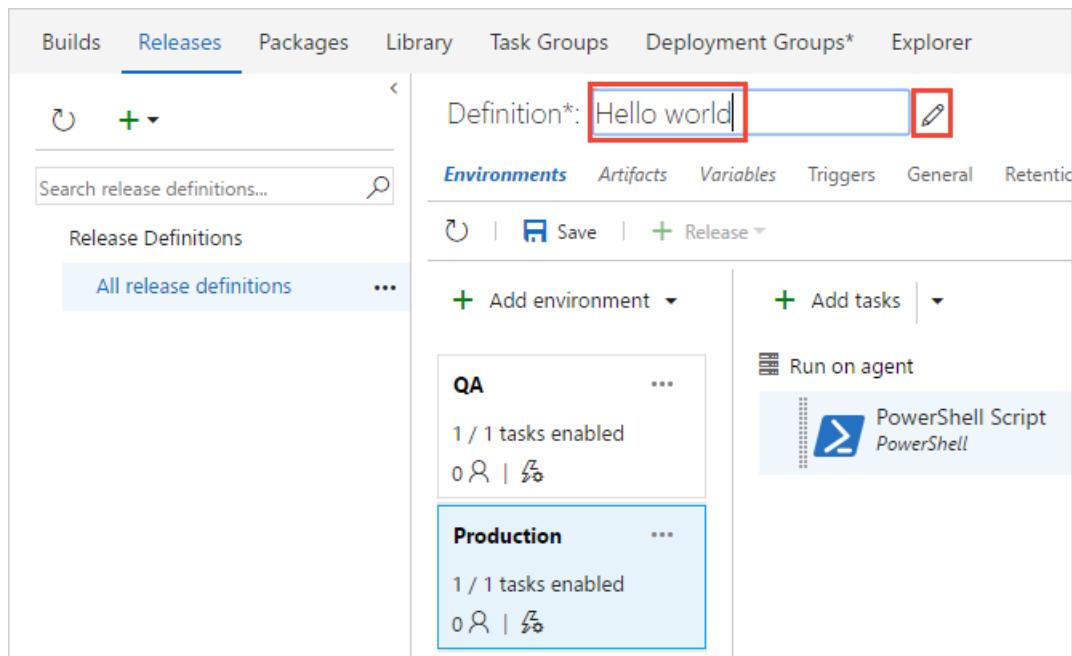
10. **Clone** the **QA** stage.



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

11. Rename the new stage **Production**.

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



13. Save the release pipeline.

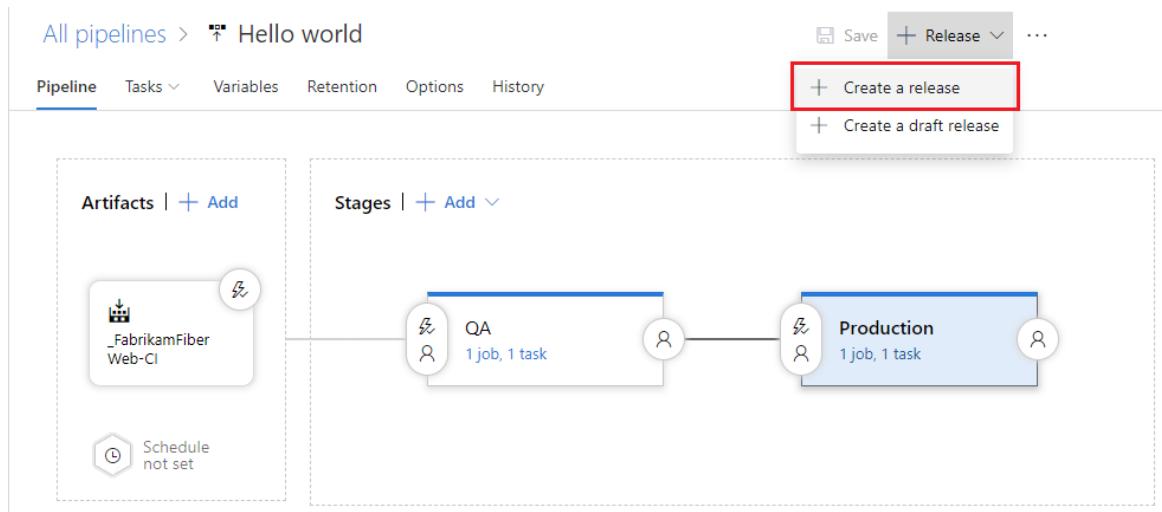
A release pipeline is a collection of stages to which the application build artifacts are deployed. It also defines the actual deployment pipeline for each stage, as well as how the artifacts are promoted from one stage to another.

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.



When **Create new release** appears, select **Create**.

2. Open the release that you created.

All pipelines > **Hello world**

Release **Release1** has been created

Pipeline Tasks Variables Retention Options History

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

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

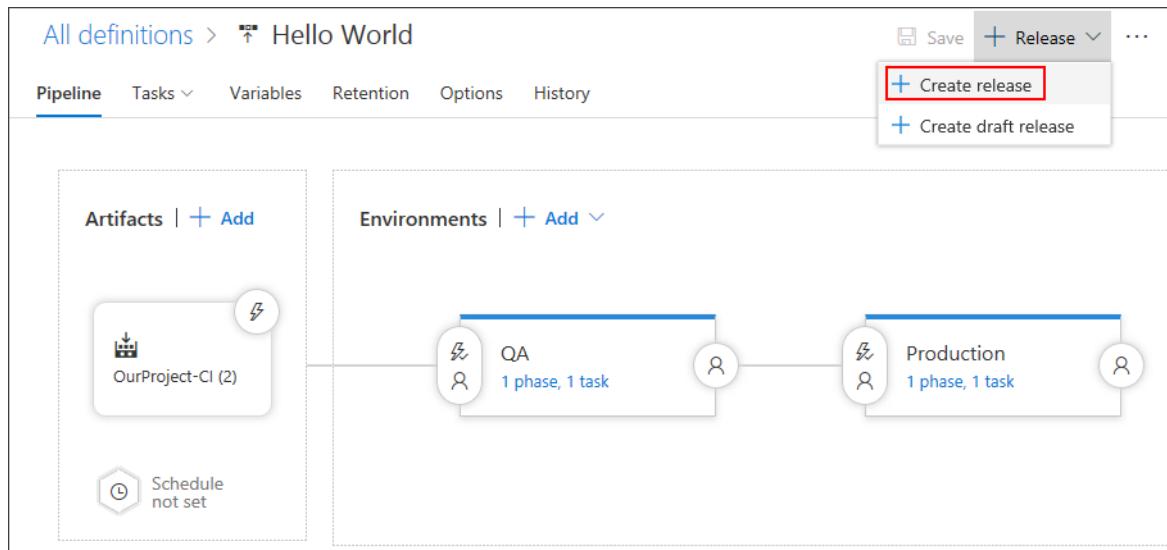
Stages

QA ✓ Succeeded on 8/27/2018 6:16 PM

Redeploy Logs

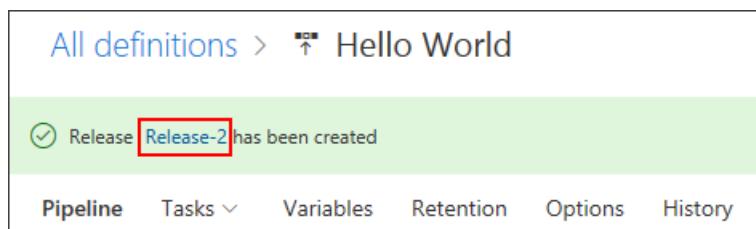
Production ⓘ Not deployed

4. Create a new release.

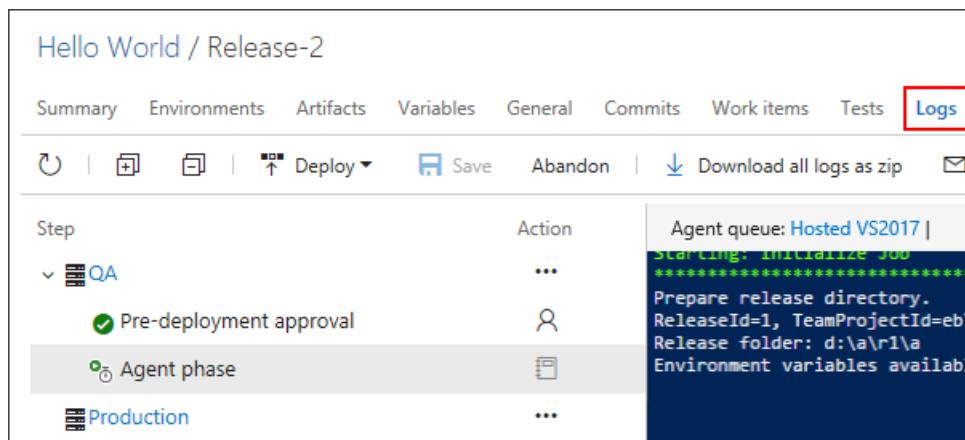


When **Create new release** appears, select **Create** (TFS 2018.2) or **Queue** (TFS 2018 RTM).

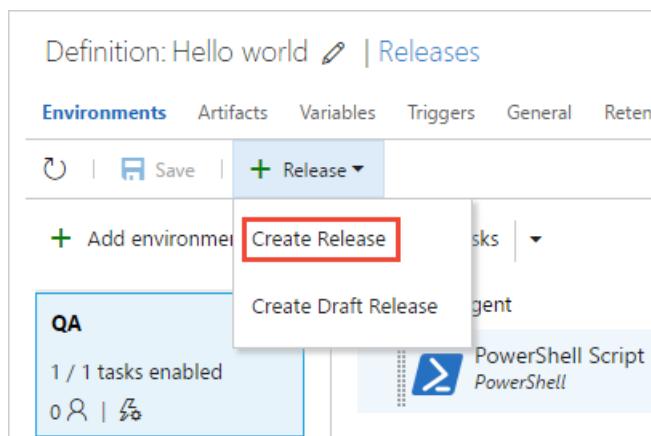
5. Open the release that you created.



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



7. Create a new release.



8. Open the release that you created.

The screenshot shows the 'Releases' tab for a project named 'Hello world'. A message at the top says 'Release **Release-2** has been created.' Below it, there's a summary card for 'Release-2' with fields for Title (locked), Environment (Environments), and Status (Running). The status bar indicates 'Release-2' is running.

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

The screenshot shows the 'Logs' tab for 'Release-1'. It lists two steps: 'QA' and 'Run on agent'. The 'QA' step is expanded, showing a green checkmark for 'Pre-deployment approval'. The 'Run on agent' step is collapsed. A modal window is open over the 'Run on agent' step, displaying deployment logs. The logs show the job starting, initializing, preparing the release directory, and noting 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.

✓ PowerShell Script

```

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\a\_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						Send Email																																															
<p>Download all logs as zip</p> <table border="1"> <thead> <tr> <th>Step</th> <th>Action</th> <th>Agent queue: Hosted VS2017 Agent: Hosted Agent</th> </tr> </thead> <tbody> <tr> <td>> QA</td> <td>...</td> <td>1 2018-04-25T14:53:54.9742188Z ##[section]Starting: PowerShell Script</td> </tr> <tr> <td> └ Production</td> <td>...</td> <td>2 2018-04-25T14:53:54.9746441Z =====</td> </tr> <tr> <td> Pre-deployment approval</td> <td></td> <td>3 2018-04-25T14:53:54.9746591Z Task : PowerShell</td> </tr> <tr> <td> └ Agent phase</td> <td></td> <td>4 2018-04-25T14:53:54.9746697Z Description : Run a PowerShell script</td> </tr> <tr> <td> Initialize Agent</td> <td></td> <td>5 2018-04-25T14:53:54.9746796Z Version : 1.2.3</td> </tr> <tr> <td> Initialize Job</td> <td></td> <td>6 2018-04-25T14:53:54.9746897Z Author : Microsoft Corporation</td> </tr> <tr> <td> Download artifact - OurProject-CI (2)</td> <td></td> <td>7 2018-04-25T14:53:54.9747808Z Help : [More Information](https://go.microsoft.com/fwlink/?LinkId=613736)</td> </tr> <tr> <td> PowerShell Script</td> <td></td> <td>8 2018-04-25T14:53:54.9747921Z =====</td> </tr> <tr> <td> Post-deployment approval</td> <td></td> <td>9 2018-04-25T14:53:55.0138848Z ##[command]. 'D:\a\r1\a\OurProject-CI\drop\HelloWorld.ps1' -greeter "Raisa Pokrovskaya"</td> </tr> <tr> <td></td> <td></td> <td>10 2018-04-25T14:53:57.2485315Z Hello world from Raisa Pokrovskaya</td> </tr> <tr> <td></td> <td></td> <td>11 2018-04-25T14:53:57.2486035Z Trigger: Hello world</td> </tr> <tr> <td></td> <td></td> <td>12 2018-04-25T14:53:57.2486230Z Now that you've got CI/CD, you can automatically deploy your app every time your team checks in code.</td> </tr> <tr> <td></td> <td></td> <td>13 2018-04-25T14:53:57.3058801Z ##[section]Finishing: PowerShell Script</td> </tr> <tr> <td></td> <td></td> <td>14</td> </tr> </tbody> </table>												Step	Action	Agent queue: Hosted VS2017 Agent: Hosted Agent	> QA	...	1 2018-04-25T14:53:54.9742188Z ##[section]Starting: PowerShell Script	└ Production	...	2 2018-04-25T14:53:54.9746441Z =====	Pre-deployment approval		3 2018-04-25T14:53:54.9746591Z Task : PowerShell	└ Agent phase		4 2018-04-25T14:53:54.9746697Z Description : Run a PowerShell script	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](https://go.microsoft.com/fwlink/?LinkId=613736)	PowerShell Script		8 2018-04-25T14:53:54.9747921Z =====	Post-deployment approval		9 2018-04-25T14:53:55.0138848Z ##[command]. 'D:\a\r1\a\OurProject-CI\drop\HelloWorld.ps1' -greeter "Raisa Pokrovskaya"			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 deploy your app every time your team checks in code.			13 2018-04-25T14:53:57.3058801Z ##[section]Finishing: PowerShell Script			14
Step	Action	Agent queue: Hosted VS2017 Agent: Hosted Agent																																																						
> QA	...	1 2018-04-25T14:53:54.9742188Z ##[section]Starting: PowerShell Script																																																						
└ Production	...	2 2018-04-25T14:53:54.9746441Z =====																																																						
Pre-deployment approval		3 2018-04-25T14:53:54.9746591Z Task : PowerShell																																																						
└ Agent phase		4 2018-04-25T14:53:54.9746697Z Description : Run a PowerShell script																																																						
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](https://go.microsoft.com/fwlink/?LinkId=613736)																																																						
PowerShell Script		8 2018-04-25T14:53:54.9747921Z =====																																																						
Post-deployment approval		9 2018-04-25T14:53:55.0138848Z ##[command]. 'D:\a\r1\a\OurProject-CI\drop\HelloWorld.ps1' -greeter "Raisa Pokrovskaya"																																																						
		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 deploy your app every time your team checks in code.																																																						
		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																																													
<p>Agent: Hosted Agent</p> <table border="1"> <thead> <tr> <th>Step</th> <th>Action</th> <th>Agent: Hosted Agent</th> </tr> </thead> <tbody> <tr> <td>> QA</td> <td>...</td> <td>1 2017-04-11T12:54:58.5891186Z ##[section]Starting: PowerShell Script</td> </tr> <tr> <td> └ Production</td> <td>...</td> <td>2 2017-04-11T12:54:58.6047463Z =====</td> </tr> <tr> <td> Pre-deployment approval</td> <td></td> <td>3 2017-04-11T12:54:58.6047463Z Task : PowerShell</td> </tr> <tr> <td> └ Run on agent</td> <td></td> <td>4 2017-04-11T12:54:58.6047463Z Description : Run a PowerShell script</td> </tr> <tr> <td> Initialize Agent</td> <td></td> <td>5 2017-04-11T12:54:58.6047463Z Version : 1.2.3</td> </tr> <tr> <td> Initialize Job</td> <td></td> <td>6 2017-04-11T12:54:58.6047463Z Author : Microsoft Corporation</td> </tr> <tr> <td> Download Artifacts</td> <td></td> <td>7 2017-04-11T12:54:58.6047463Z Help : [More Information](https://go.microsoft.com/fwlink/?LinkId=613736)</td> </tr> <tr> <td> PowerShell Script</td> <td></td> <td>8 2017-04-11T12:54:58.6047463Z =====</td> </tr> <tr> <td> Post-deployment approval</td> <td></td> <td>9 2017-04-11T12:54:58.6672458Z ##[command]. 'D:\a\r1\a\Hello world\drop\HelloWorld.ps1' -greeter "Raisa Pokrovskaya"</td> </tr> <tr> <td></td> <td></td> <td>10 2017-04-11T12:54:59.3703946Z Hello world from Raisa Pokrovskaya</td> </tr> <tr> <td></td> <td></td> <td>11 2017-04-11T12:54:59.3703946Z Trigger: Hello world</td> </tr> <tr> <td></td> <td></td> <td>12 2017-04-11T12:54:59.3703946Z Now that you've got CI/CD, you can automatically deploy your app every time your team checks in code.</td> </tr> <tr> <td></td> <td></td> <td>13 2017-04-11T12:54:59.4641245Z ##[section]Finishing: PowerShell Script</td> </tr> <tr> <td></td> <td></td> <td>14</td> </tr> </tbody> </table>												Step	Action	Agent: Hosted Agent	> QA	...	1 2017-04-11T12:54:58.5891186Z ##[section]Starting: PowerShell Script	└ Production	...	2 2017-04-11T12:54:58.6047463Z =====	Pre-deployment approval		3 2017-04-11T12:54:58.6047463Z Task : PowerShell	└ 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.microsoft.com/fwlink/?LinkId=613736)	PowerShell Script		8 2017-04-11T12:54:58.6047463Z =====	Post-deployment approval		9 2017-04-11T12:54:58.6672458Z ##[command]. 'D:\a\r1\a\Hello world\drop\HelloWorld.ps1' -greeter "Raisa Pokrovskaya"			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 deploy your app every time your team checks in code.			13 2017-04-11T12:54:59.4641245Z ##[section]Finishing: PowerShell Script			14
Step	Action	Agent: Hosted Agent																																																						
> QA	...	1 2017-04-11T12:54:58.5891186Z ##[section]Starting: PowerShell Script																																																						
└ Production	...	2 2017-04-11T12:54:58.6047463Z =====																																																						
Pre-deployment approval		3 2017-04-11T12:54:58.6047463Z Task : PowerShell																																																						
└ 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.microsoft.com/fwlink/?LinkId=613736)																																																						
PowerShell Script		8 2017-04-11T12:54:58.6047463Z =====																																																						
Post-deployment approval		9 2017-04-11T12:54:58.6672458Z ##[command]. 'D:\a\r1\a\Hello world\drop\HelloWorld.ps1' -greeter "Raisa Pokrovskaya"																																																						
		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 deploy your app every time your team checks in code.																																																						
		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 how to create your first pipeline in Azure. Learn more about configuring pipelines in the language of your choice:

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

Or, you can proceed to [customize the pipeline](#) you just created.

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

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

To learn how to publish your Pipeline Artifacts, see [Publish Pipeline Artifacts](#).

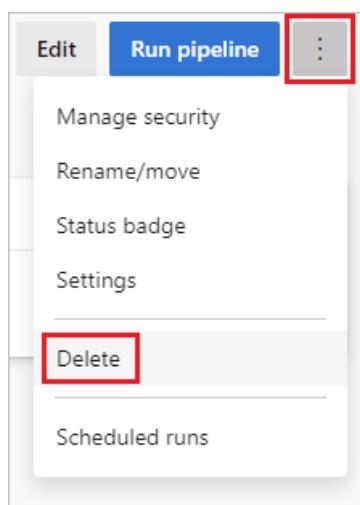
To find out what else you can do in YAML pipelines, see [YAML schema reference](#).

Clean up

If you created any test pipelines, they are easy to delete when you are done with them.

- Browser
- Azure DevOps CLI

To delete a pipeline, navigate to the summary page for that pipeline, and choose **Delete** from the ... menu at the top-right of the page. Type the name of the pipeline to confirm, and choose **Delete**.



You've learned the basics of creating and running 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
Java	Gradle

JavaScript	Node.js
Xcode	Xcode

FAQ

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?

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](#)
 - [Eclipse](#)
 - [Xcode](#)
 - [IntelliJ](#)
 - [Command line](#)
- Services
 - [Azure Pipelines](#)
 - Git service providers such as GitHub and Bitbucket Cloud
 - Subversion
- 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 Cloud
 - 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.

Azure DevOps Pipelines interface. On the left, the sidebar shows 'FabrikamFiber Web' selected. In the center, the 'FabrikamFiber Web-Cl' pipeline is listed under the 'master' branch. A context menu is open on the right, with the '...' button highlighted by a red box. Other options in the menu include: Security, Rename/move, Pause builds, Add to my favorites, Clone (highlighted), Save as a template (highlighted), Export (highlighted), Status badge, and Delete.

Azure DevOps Build Definitions interface. The 'Build Definitions' tab is selected. Under the 'All Definitions' tab, the 'HelloWorld-Cl' pipeline is selected. A context menu is open on the right, with the '...' button highlighted by a red box. Other options in the menu include: Queue new build..., Move definition, View definition summary, Edit..., Add to my favorites, Add to team favorites, Clone... (highlighted), Export (highlighted), Rename..., Save as a template... (highlighted), 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.

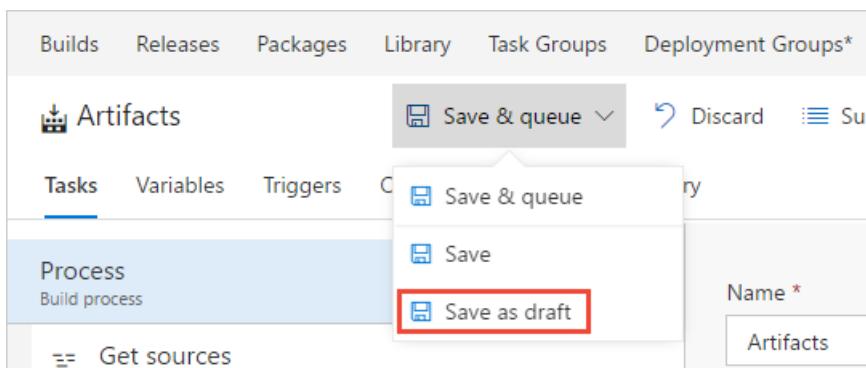
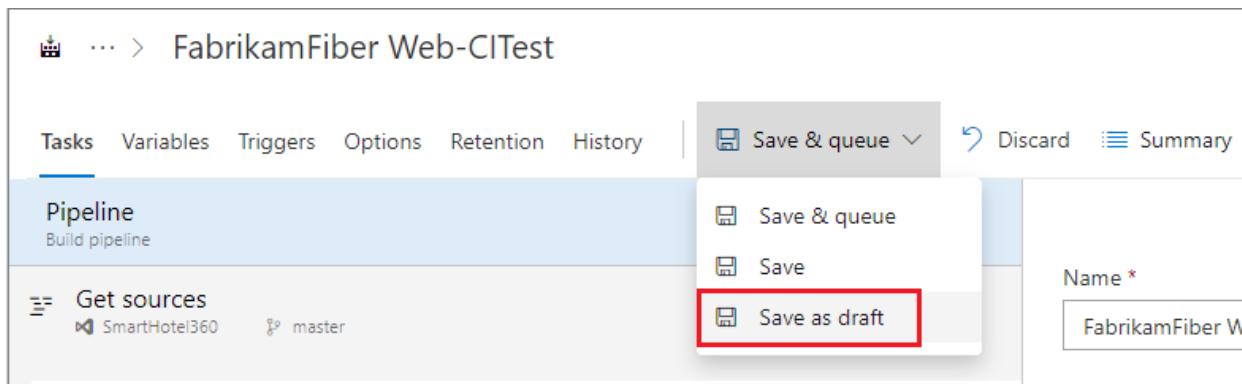
After you create a template, your team members can use it to follow the pattern in new pipelines.

TIP

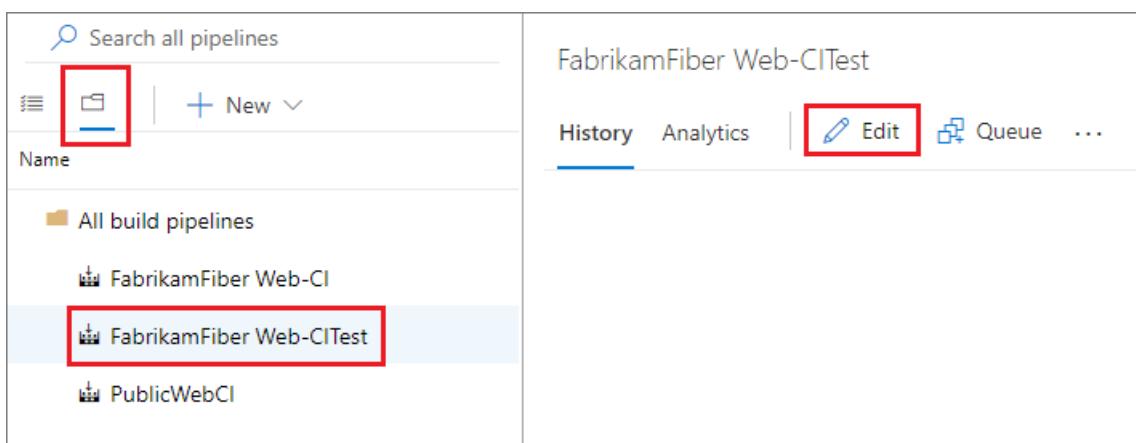
If you're using the New Build Editor, then your custom templates are shown at the bottom of the list.

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.



You can edit and test your draft as needed.



The screenshot shows the 'Build Definitions' page in the Azure DevOps interface. At the top, there are tabs for 'Builds', 'Releases', 'Packages', 'Library', 'Task Groups', 'Deployment Groups*', and 'Explorer'. Below these, a search bar says 'Search all definitions' and a '+ New' button are visible. Under the search bar, there are tabs for 'Mine', 'All Definitions' (which is underlined), 'Queued', and 'XAML'. A 'Default branch summary' is shown above a tree view. The tree view has a checked 'Artifacts' node, and under it are several items: 'Build.ArtifactStagingDirectory test', 'Build.BinariesDirectory test', 'Build.BuildNumber_Test', and 'Build.DefinitionsTest'. A context menu is open over the 'Artifacts' node, with options: 'Queue new build...', 'Move definition', 'View definition summary', and 'Edit...'. The 'Edit...' option is highlighted with a red box.

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

The first screenshot shows the pipeline configuration for 'FabrikamFiber Web-CITest'. It includes tabs for 'Tasks', 'Variables', 'Options', and 'History'. Below these are sections for 'Pipeline' (Build pipeline) and 'Get sources' (SmartHotel360, master branch). On the right, there's a 'Name' field set to 'FabrikamFiber Web-CITest'. The 'Publish draft' button is highlighted with a red box. The second screenshot shows the 'Artifacts' tab in the main navigation. It has a 'Save draft & queue' button and a 'Publish draft' button, which is also highlighted with a red box.

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

How can I delete a pipeline?

To delete a pipeline, navigate to the summary page for that pipeline, and choose **Delete** from the ... menu in the top-right of the page. Type the name of the pipeline to confirm, and choose **Delete**.

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](#).)

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](#).

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](#)

To learn more about build pipeline settings, see:

- [Getting sources](#)
- [Tasks](#)
- [Variables](#)
- [Triggers](#)
- [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](#).

Can I use a single command at the command line to run multiple pipelines in Azure DevOps Services?

Currently, the Azure CLI and Azure APIs don't offer commands that run multiple pipelines from the command line. You can use [Azure CLI commands](#) to list all pipelines and definitions and provide a *single* release or build ID as a parameter. All commands are designed to work for independent runs of independent pipelines, and they require unique ID requests that allow only one, unique value. To learn about pipeline triggers, see [Specify events that trigger pipelines](#).

Clone or import a pipeline

11/2/2020 • 3 minutes to read • [Edit Online](#)

One approach to creating a pipeline is to copy an existing pipeline and use it as a starting point. For YAML pipelines, the process is as easy as copying the YAML from one pipeline to another. For pipelines created in the classic editor, the procedure depends on whether the pipeline to copy is in the same project as the new pipeline. If the pipeline to copy is in the same project, you can clone it, and if it is in a different project you can export it from that project and import it into your project.

Clone a pipeline

- [YAML](#)
- [Classic](#)

For YAML pipelines, the process for cloning is to copy the YAML from the source pipeline and use it as the basis for the new pipeline.

1. Navigate to your pipeline, and choose **Edit**.

The screenshot shows the Azure DevOps interface for managing pipelines. On the left, there's a sidebar with icons for Overview, Boards, Repos, Pipelines (which is selected and highlighted with a red box and a red number 1), and Builds. The main area displays a list of pipelines under the project 'Fabrikam Fiber'. One pipeline, 'JobPoolTest', is expanded to show its details. Another pipeline, 'Fabrikam Fiber', is selected and highlighted with a red box and a red number 2. At the top right, there are three buttons: 'Edit' (highlighted with a red box and a red number 3), 'Queue', and a more options menu. Below the pipeline list, there's a commit history section.

2. Copy the pipeline YAML from the editor, and paste it into the YAML editor for your new pipeline.

3. To customize your newly cloned pipeline, see [Customize your pipeline](#).

1. Navigate to the [pipeline details](#) for your pipeline, and choose **Edit**.

The screenshot shows the 'FabrikamFiber' pipeline details page. On the left, there's a sidebar with icons for Overview, Boards, Repos, Pipelines, and Releases. The main area shows the pipeline name 'FabrikamFiber' and a 'Runs' tab. Under the 'Runs' tab, there's a table listing three recent runs. Each run row has a 'Description' column (e.g., '#20191209.3 Set up CI with Azure Pipelines'), a 'Stages' column (with status indicators), and a timestamp column. At the top right of the table, there are three buttons: 'Edit' (highlighted with a red box), 'Run pipeline', and a more options menu. The table also includes 'Branches' and 'Analytics' tabs at the bottom.

2. Copy the pipeline YAML from the editor, and paste it into the YAML editor for your new pipeline.

3. To customize your newly cloned pipeline, see [Customize your pipeline](#).

This version of TFS doesn't support YAML pipelines.

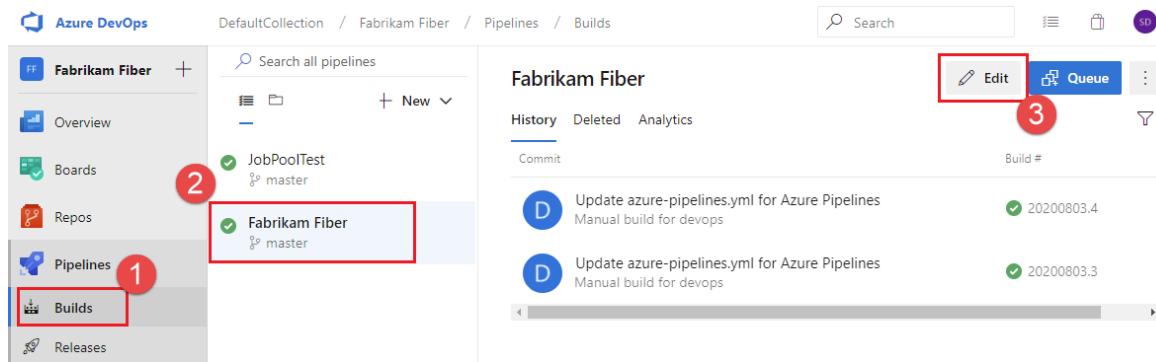
Export and Import a pipeline

You can create a new classic pipeline by exporting an existing one and then importing it. This is useful in cases where the new pipeline has to be created in a separate project.

- [YAML](#)
- [Classic](#)

In a YAML pipeline, exporting from one project and importing into another is the same process as cloning. You can simply copy the pipeline YAML from the editor and paste it into the YAML editor for your new pipeline.

1. Navigate to your pipeline, and choose **Edit**.

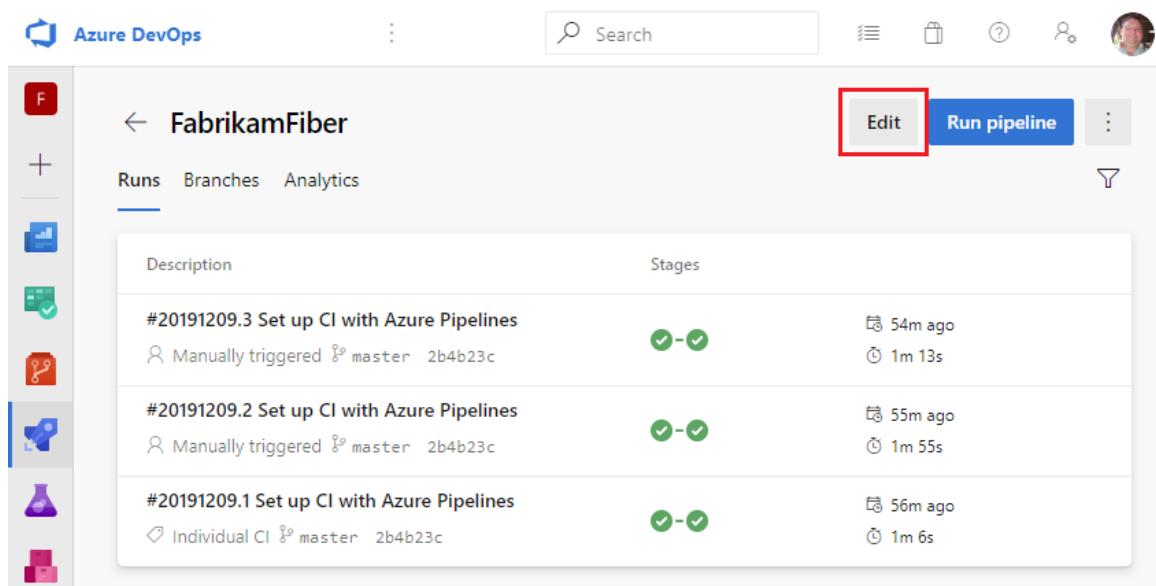


The screenshot shows the Azure DevOps interface for the 'Fabrikam Fiber' project. On the left sidebar, the 'Builds' tab is selected (Step 1). In the main area, the 'Fabrikam Fiber' pipeline is listed under 'Search all pipelines' (Step 2). On the right, the pipeline details are shown with two recent builds (Step 3). The 'Edit' button is highlighted with a red box.

2. Copy the pipeline YAML from the editor, and paste it into the YAML editor for your new pipeline.

3. To customize your newly cloned pipeline, see [Customize your pipeline](#).

1. [Navigate](#) to the [pipeline details](#) for your pipeline, and choose **Edit**.



The screenshot shows the 'FabrikamFiber' pipeline details page. The 'Edit' button is highlighted with a red box. The page displays three recent build runs: #20191209.3, #20191209.2, and #20191209.1, each with its status and timestamp.

Description	Stages	Time Ago	Duration
#20191209.3 Set up CI with Azure Pipelines Manually triggered master 2b4b23c	✓ - ✓	54m ago	1m 13s
#20191209.2 Set up CI with Azure Pipelines Manually triggered master 2b4b23c	✓ - ✓	55m ago	1m 55s
#20191209.1 Set up CI with Azure Pipelines Individual CI master 2b4b23c	✓ - ✓	56m ago	1m 6s

2. Copy the pipeline YAML from the editor, and paste it into the YAML editor for your new pipeline.

3. To customize your newly cloned pipeline, see [Customize your pipeline](#).

This version of TFS doesn't support YAML pipelines.

Next steps

Learn to [customize the pipeline](#) you just cloned or imported.

Customize your pipeline

6/24/2021 • 6 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019

This is a step-by-step guide on common ways to customize your pipeline.

Prerequisite

Follow instructions in [Create your first pipeline](#) to create a working pipeline.

Understand the `azure-pipelines.yml` file

A pipeline is defined using a YAML file in your repo. Usually, this file is named `azure-pipelines.yml` and is located at the root of your repo.

- Navigate to the **Pipelines** page in Azure Pipelines and select the pipeline you created.
- Select **Edit** in the context menu of the pipeline to open the YAML editor for the pipeline. Examine the contents of the YAML file.

```
trigger:
- main

pool:
  vmImage: 'Ubuntu-16.04'

steps:
- task: Maven@3
  inputs:
    mavenPomFile: 'pom.xml'
    mavenOptions: '-Xmx3072m'
    javaHomeOption: 'JDKVersion'
    jdkVersionOption: '1.8'
    jdkArchitectureOption: 'x64'
    publishJUnitResults: false
    testResultsFiles: '**/surefire-reports/TEST-*.xml'
    goals: 'package'
```

NOTE

The contents of your YAML file may be different depending on the sample repo you started with, or upgrades made in Azure Pipelines.

This pipeline runs whenever your team pushes a change to the main branch of your repo. It runs on a Microsoft-hosted Linux machine. The pipeline process has a single step, which is to run the Maven task.

Change the platform to build on

You can build your project on [Microsoft-hosted agents](#) that already include SDKs and tools for various development languages. Or, you can use [self-hosted agents](#) with specific tools that you need.

- Navigate to the editor for your pipeline by selecting **Edit pipeline** action on the build, or by selecting

Edit from the pipeline's main page.

- Currently the pipeline runs on a Linux agent:

```
pool:  
  vmImage: "ubuntu-16.04"
```

- To choose a different platform like Windows or Mac, change the `vmImage` value:

```
pool:  
  vmImage: "vs2017-win2016"
```

```
pool:  
  vmImage: "macos-latest"
```

- Select **Save** and then confirm the changes to see your pipeline run on a different platform.

Add steps

You can add additional **scripts** or **tasks** as steps to your pipeline. A task is a pre-packaged script. You can use tasks for building, testing, publishing, or deploying your app. For Java, the Maven task we used handles testing and publishing results, however, you can use a task to publish code coverage results too.

- Open the YAML editor for your pipeline.
- Add the following snippet to the end of your YAML file.

```
- task: PublishCodeCoverageResults@1  
  inputs:  
    codeCoverageTool: "JaCoCo"  
    summaryFileLocation: "$(System.DefaultWorkingDirectory)/**/site/jacoco/jacoco.xml"  
    reportDirectory: "$(System.DefaultWorkingDirectory)/**/site/jacoco"  
    failIfCoverageEmpty: true
```

- Select **Save** and then confirm the changes.
- You can view your test and code coverage results by selecting your build and going to the **Test** and **Coverage** tabs.

Build across multiple platforms

You can build and test your project on multiple platforms. One way to do it is with `strategy` and `matrix`. You can use variables to conveniently put data into various parts of a pipeline. For this example, we'll use a variable to pass in the name of the image we want to use.

- In your `azure-pipelines.yml` file, replace this content:

```
pool:  
  vmImage: "ubuntu-16.04"
```

with the following content:

```

strategy:
  matrix:
    linux:
      imageName: "ubuntu-16.04"
    mac:
      imageName: "macos-10.14"
    windows:
      imageName: "vs2017-win2016"
  maxParallel: 3

pool:
  vmImage: $(imageName)

```

- Select **Save** and then confirm the changes to see your build run up to three jobs on three different platforms.

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](#).

Build using multiple versions

To build a project using different versions of that language, you can use a `matrix` of versions and a variable. In this step you can either build the Java project with two different versions of Java on a single platform or run different versions of Java on different platforms.

- If you want to build on a single platform and multiple versions, add the following matrix to your `azure-pipelines.yml` file before the Maven task and after the `vmlImage`.

```

strategy:
  matrix:
    jdk10:
      jdk_version: "1.10"
    jdk11:
      jdk_version: "1.11"
  maxParallel: 2

```

- Then replace this line in your maven task:

```
jdkVersionOption: "1.11"
```

with this line:

```
jdkVersionOption: $(jdk_version)
```

- Make sure to change the `$(imageName)` variable back to the platform of your choice.
- If you want to build on multiple platforms and versions, replace the entire content in your `azure-pipelines.yml` file before the publishing task with the following snippet:

```

trigger:
- main

strategy:
matrix:
  jdk10_linux:
    imageName: "ubuntu-16.04"
    jdk_version: "1.10"
  jdk11_windows:
    imageName: "vs2017-win2016"
    jdk_version: "1.11"
  maxParallel: 2

pool:
  vmImage: $(imageName)

steps:
- task: Maven@3
  inputs:
    mavenPomFile: "pom.xml"
    mavenOptions: "-Xmx3072m"
    javaHomeOption: "JDKVersion"
    jdkVersionOption: "$(jdk_version)"
    jdkArchitectureOption: "x64"
    publishJUnitResults: true
    testResultsFiles: "**/TEST-*.xml"
    goals: "package"

```

- Select **Save** and then confirm the changes to see your build run two jobs on two different platforms and SDKs.

Customize CI triggers

You can use a `trigger:` to specify the events when you want to run the pipeline. YAML pipelines are configured by default with a CI trigger on your default branch (which is usually `main`). You can set up triggers for specific branches or for pull request validation. For a pull request validation trigger just replace the `trigger:` step with `pr:` as shown in the two examples below.

- If you'd like to set up triggers, add either of the following snippets at the beginning of your `azure-pipelines.yml` file.

```

trigger:
- main
- releases/*

```

```

pr:
- main
- releases/*

```

You can specify the full name of the branch (for example, `main`) or a prefix-matching wildcard (for example, `releases/*`).

Pipeline settings

There are some pipeline settings that you don't manage in your YAML file, such as the YAML file path and enabled status of your pipeline. To configure these settings, navigate to the [pipeline details page](#) and choose **More actions, settings**. For more information on navigating and browsing your pipelines, see [Navigating pipelines](#).

Pipeline settings

X

Processing of new run requests

- Enabled
- Paused
- Disabled

YAML file path

azure-pipelines.yml



- Automatically link work items included in this run

Cancel

Save

From the **Pipeline settings** pane you can configure the following settings.

- **Processing of new run requests** - Sometimes you'll want to prevent new runs from starting on your pipeline.
 - By default, the processing of new run requests is **Enabled**. This setting allows standard processing of all trigger types, including manual runs.
 - **Paused** pipelines allow run requests to be processed, but those requests are queued without actually starting. When new request processing is enabled, run processing resumes starting with the first request in the queue.
 - **Disabled** pipelines prevent users from starting new runs. All triggers are also disabled while this setting is applied.
- **YAML file path** - If you ever need to direct your pipeline to use a different YAML file, you can specify the path to that file. This setting can also be useful if you need to move/rename your YAML file.
- **Automatically link work items included in this run** - The changes associated with a given pipeline run may have work items associated with them. Select this option to link those work items to the run. When this option is selected, you'll need to specify a specific branch. Work items will only be associated with runs of that branch.
 - To get notifications when your runs fail, see how to [Manage notifications for a team](#)

You've just learned the basics of customizing your pipeline. Next we recommend that you learn more about customizing a pipeline for the language you use:

- [.NET Core](#)
- [Containers](#)
- [Go](#)
- [Java](#)
- [Node.js](#)
- [Python](#)

Or, to grow your CI pipeline to a CI/CD pipeline, include a [deployment job](#) with steps to deploy your app to an environment.

To learn more about the topics in this guide see [Jobs](#), [Tasks](#), [Catalog of Tasks](#), [Variables](#), [Triggers](#), or [Troubleshooting](#).

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

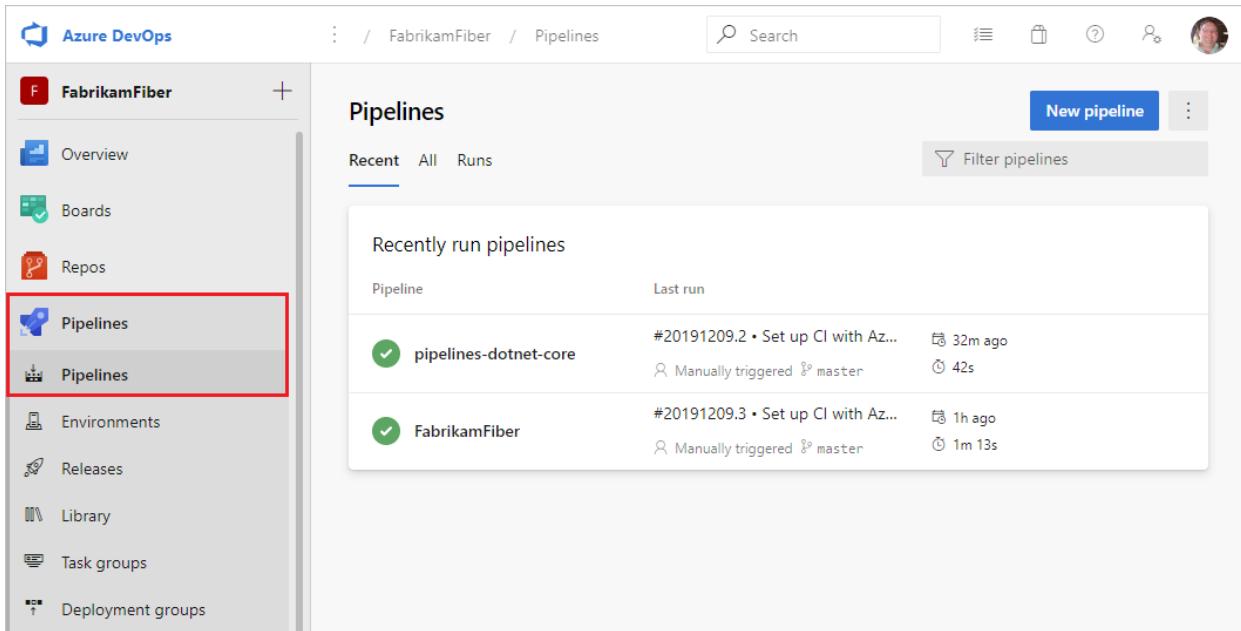
Multi-stage pipelines user experience

6/24/2021 • 4 minutes to read • [Edit Online](#)

The multi-stage pipelines experience brings improvements and ease of use to the Pipelines portal UI. This article shows you how to view and manage your pipelines using this new experience.

Navigating pipelines

You can view and manage your pipelines by choosing **Pipelines** from the left-hand menu.



The screenshot shows the Azure DevOps Pipelines portal for the project 'FabrikamFiber'. The left sidebar has a red box around the 'Pipelines' section, which is currently selected. The main area displays a table of 'Recently run pipelines' with two entries:

Pipeline	Last run
pipelines-dotnet-core	#20191209.2 • Set up CI with Az... Manually triggered 32m ago
FabrikamFiber	#20191209.3 • Set up CI with Az... Manually triggered 1h ago

You can drill down and view pipeline details, run details, pipeline analytics, job details, logs, and more.

At the top of each page is a breadcrumb navigation bar. Select the different areas of the bar to navigate to different areas of the portal. The breadcrumb navigation is a convenient way to go back one or more steps.

The screenshot shows the Azure Pipelines run summary page for run number #20191209.3. The breadcrumb navigation at the top is highlighted with red boxes and numbered 1 through 5 from right to left:

- fabrikam-tailspin (5)
- / FabrikamFiber (4)
- / Pipelines (3)
- / FabrikamFiber (2)
- / 20191209.3 (1)

Below the breadcrumb, the run summary for #20191209.3 is displayed. The pipeline was manually run by Steve Danielson. It ran on the FabrikamFiber project's master branch (commit 2b4b23c) and completed yesterday at 2:29 PM. The duration was 1m 13s and there were no tests. A "Get started" button is present.

The "Stages" tab is selected, showing two stages: "Build" and "Deploy".

1. This area of the breadcrumb navigation shows you what page you're currently viewing. In this example, the page is the run summary for run number 20191209.3.
2. One level up is a link to the [pipeline details](#) for that run.
3. The next level up is the [pipelines landing page](#).
4. This link is to the **FabrikamFiber** project, which contains the pipeline for this run.
5. The root breadcrumb link is to the Azure DevOps **fabrikam-tailspin** organization, which contains the project that contains the pipeline.

Many pages also contain a back button that takes you to the previous page.

The screenshot shows the Azure Pipelines pipelines landing page. A single pipeline entry for "#20191210.3 Update azure-pipelines.yml for Azure Pipelines" is listed. The pipeline was manually triggered on the master branch (commit b2f795e). The pass rate is 83.33% (▼ 33.3%) over the last 14 days. The pipeline has three stages: Build, Deploy, and Release, all of which are marked as successful (green checkmarks).

Pipelines landing page

From the pipelines landing page you can view pipelines and pipeline runs, create and import pipelines, manage security, and drill down into pipeline and run details.

Choose **Recent** to view recently run pipelines (the default view), or choose **All** to view all pipelines.

Pipelines

[New pipeline](#)[Recent](#) [All](#) [Runs](#)[Filter pipelines](#)

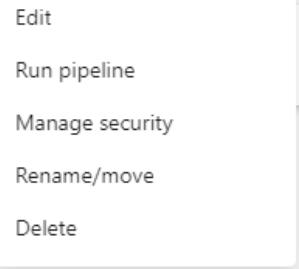
Recently run pipelines

Pipeline	Last run	
pipelines-dotnet-core	#20191209.2 • Set up CI with Az... Manually triggered ⌂ master	36m ago 42s
FabrikamFiber	#20191209.3 • Set up CI with Az... Manually triggered ⌂ master	1h ago 1m 13s

Select a pipeline to manage that pipeline and view its runs. Select the build number for the last run to view the results of that build, select the branch name to view the branch for that run, or select the context menu to run the pipeline and perform other management actions.

Recently run pipelines

Pipeline	Last run	
pipelines-dotnet-core	#20191209.2 • Set up CI with Azure Pip... Manually triggered ⌂ master	1h ago 42s
FabrikamFiber	#20191209.3 • Set up CI with Azure Pip... Manually triggered ⌂ master	2h ago 1m



Select **Runs** to view all pipeline runs. You can optionally filter the displayed runs.

Pipelines

Recent All **Runs**

New pipeline :

Filter by keywords State Repository Requested for Tags X

All pipeline runs

Description	Stages	Created	Last Updated
Set up CI with Azure Pipelines 🔗 #20191209.2 on pipelines-dotnet-core ⚡ master d4964...	✓	44m ago	42s
Set up CI with Azure Pipelines 🔗 #20191209.1 on pipelines-dotnet-core ⚡ master d4964...	✓	53m ago	46s
Set up CI with Azure Pipelines 🔗 #20191209.3 on FabrikamFiber ⚡ master 2b4b23c	✓ - ✓	1h ago	1m 13s
Set up CI with Azure Pipelines 🔗 #20191209.2 on FabrikamFiber ⚡ master 2b4b23c	✓ - ✓	1h ago	1m 55s
Set up CI with Azure Pipelines 🔗 #20191209.1 on FabrikamFiber ⚡ master 2b4b23c	✓ - ✓	1h ago	1m 6s

Select a pipeline run to view information about that run.

View pipeline details

The details page for a pipeline allows you to view and manage that pipeline.

Azure DevOps : Search ☰ ? User

FabrikamFiber Edit Run pipeline ⋮

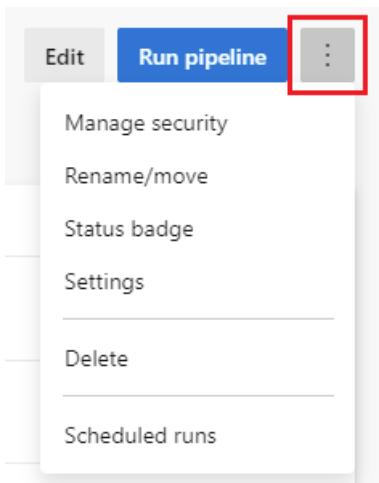
Runs Branches Analytics Filter

Description	Stages	Created	Last Updated
#20191209.3 Set up CI with Azure Pipelines 🔗 Manually triggered ⚡ master 2b4b23c	✓ - ✓	54m ago	1m 13s
#20191209.2 Set up CI with Azure Pipelines 🔗 Manually triggered ⚡ master 2b4b23c	✓ - ✓	55m ago	1m 55s
#20191209.1 Set up CI with Azure Pipelines 🔗 Individual CI ⚡ master 2b4b23c	✓ - ✓	56m ago	1m 6s

Choose **Edit** to edit your pipeline. For more information, see [YAML pipeline editor](#).

Pipeline settings

You can view and configure pipeline settings from the **More actions** ⋮ menu on the pipeline details page.



- **Manage security** - [Manage security](#)
- **Rename/move** - Edit your pipeline name and folder location.

Rename/move pipeline

Name

Select folder

[Cancel](#) [Save](#)

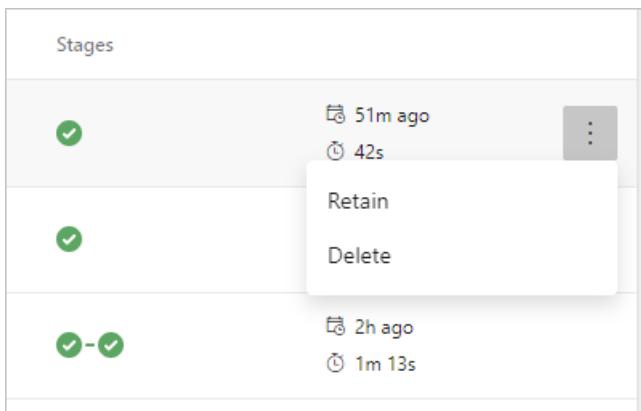
- **Status badge** - [Add a status badge to your repository](#)
- **Settings** - [Pipeline settings](#)
- **Delete** - Deletes the pipeline including all builds and associated artifacts.
- **Scheduled runs** - [Scheduled runs view](#)

Runs

Select Runs to view the runs for that pipeline. You can optionally filter the displayed runs.

← FabrikamFiber		Edit	Run pipeline	⋮	
Runs	Branches	Analytics	Filter by keywords		
		State ▾ Repository ▾ Requested for ▾ Tags ▾			
Description	Stages				
#20191209.3 Set up CI with Azure Pipelines Manually triggered ⏲ master 2b4b23c	✓ - ✓	1h ago	1m 13s		
#20191209.2 Set up CI with Azure Pipelines Manually triggered ⏲ master 2b4b23c	✓ - ✓	1h ago	1m 55s		
#20191209.1 Set up CI with Azure Pipelines Individual CI ⏲ master 2b4b23c	✓ - ✓	1h ago	1m 6s	⋮	

You can choose to **Retain** or **Delete** a run from the context menu. For more information on run retention, see [Build and release retention policies](#).



Branches

Select **Branches** to view the history or run for that branch. Hover over the **History** to view a summary for each run, and select a run to navigate to the details page for that run.

A screenshot of a software interface titled 'FabrikamFiber'. It shows three tabs: 'Runs' (disabled), 'Branches' (selected and highlighted with a red box), and 'Analytics'. The 'Branches' tab displays a table with three columns: 'Branch', 'History', and 'Last run'. Under 'Branch', it lists 'master' and 'default'. In the 'History' column, there is a timeline of vertical bars representing run histories. The 'Last run' column shows a green checkmark next to the text '#201912'. The interface includes standard navigation elements like back, edit, run pipeline, and more options buttons.

Analytics

Select **Analytics** to view pipeline metrics such as pass rate and run duration. Choose **View full report** for more information on each metric.

[Runs](#) [Branches](#) [Analytics](#)

Pipeline pass rate

100%

Runs:
3[View full report >](#)

Test pass rate



No test runs completed in the last 14 days

[View full report >](#)

Pipeline duration

80th percentile

2m 4s

Succeeded runs:
3

💡 74.44% of total run time is taken by step -
Component Detection (auto-injected by policy)

[View full report >](#)

View pipeline run details

From the pipeline run summary you can view the status of your run, both while it is running and when it is complete.

#20191210.2 Update azure-pipelines.yml for Azure Pipe... Run new

on FabrikamFiber

[Summary](#) [Environments](#)

Triggered by Steve Danielson

FabrikamFiber master b2f795e

Today at 12:56 PM

Duration: 1m 9s

Tests: Get started

Changes: 2 commits

Work items: 1 linked

Artifacts: 1 published

[Stages](#) [Jobs](#)

Build

1 job completed

41s

Deploy

1 job completed

13s

1 artifact

From the summary pane you can download artifacts, and navigate to linked commits, test results, and work items.

Cancel and re-run a pipeline

If the pipeline is running, you can cancel it by choosing **Cancel**. If the run has completed, you can re-run the pipeline by choosing **Run new**.

#20191210.3 Update azure-pipelines.yml for Azure Pip... Cancel

on FabrikamFiber

[Stages](#) [Jobs](#)

Build

0/1 completed

48s

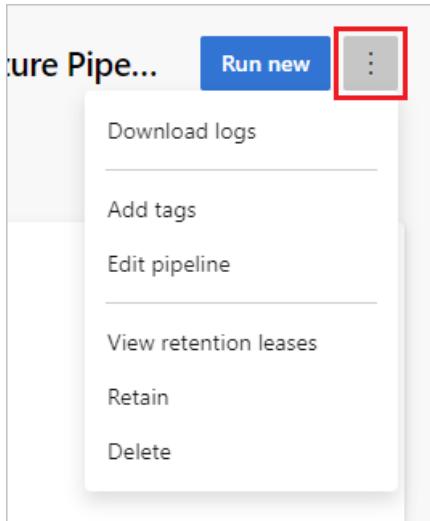
Deploy

Not started

Cancel

Pipeline run more actions menu

From the **More actions**  menu you can download logs, add tags, edit the pipeline, delete the run, and configure [retention](#) for the run.



NOTE

You can't delete a run if the run is retained. If you don't see **Delete**, choose **Stop retaining run**, and then delete the run. If you see both **Delete** and **View retention releases**, one or more configured retention policies still apply to your run. Choose **View retention releases**, delete the policies (only the policies for the selected run are removed), and then delete the run.

Jobs and stages

The jobs pane displays an overview of the status of your stages and jobs. This pane may have multiple tabs depending on whether your pipeline has stages and jobs, or just jobs. In this example the pipeline has two stages named **Build** and **Deploy**. You can drill down into the pipeline steps by choosing the job from either the **Stages** or **Jobs** pane.

Jobs			
Name	Status	Stage	Duration
Build	Success	Build	40s
DeployWeb	Success	Deploy	10s

Choose a job to see the steps for that job.

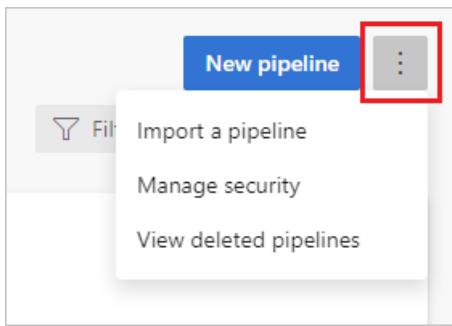
The screenshot shows the Azure Pipelines interface. On the left, there's a sidebar with a back arrow and the text "Jobs in run #20191...". Below that is a section titled "Build" which contains a tree view of pipeline steps. The steps are categorized into sections: "Build", "Deploy", and "Finalize build". Under "Build", there are seven steps: "Initialize job" (1s), "Checkout" (3s), "CmdLine" (2s), "Component Detect" (32s), "Post-job: Checkout" (<1s), "Finalize Job" (<1s), and "Report build status" (<1s). The "Deploy" section contains one step: "DeployWeb" (10s). The "Finalize build" section contains one step: "Report build status" (<1s). On the right side of the interface, there's a dark panel titled "Build" with detailed information about the pipeline run, including the pool, image, agent, start time, duration, and a link to "Job preparation parameters".

From the steps view, you can review the status and details of each step. From the **More actions** you can toggle timestamps or view a raw log of all steps in the pipeline.

This screenshot shows the "Build" pipeline details page. The "More actions" button is highlighted with a red box. A dropdown menu is open, showing two options: "View job raw log" and "Toggle timestamps". The background shows the pipeline summary with basic details like pool, image, and duration.

Manage security

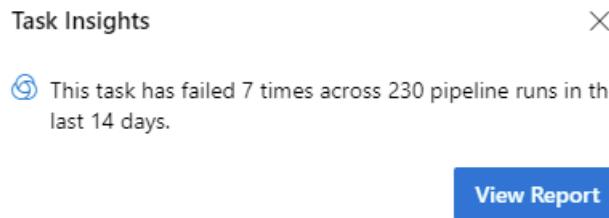
You can configure pipelines security on a project level from the **More actions** on the pipelines landing page, and on a pipeline level on the pipeline details page.



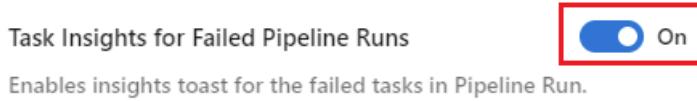
To support security of your pipeline operations, you can add users to a built-in security group, set individual permissions for a user or group, or add users to pre-defined roles. You can manage security for Azure Pipelines in the web portal, either from the user or admin context. For more information on configuring pipelines security, see [Pipeline permissions and security roles](#).

Task insights for failed pipeline runs

Azure DevOps provides a **Task Insights for Failed Pipeline Runs** setting, that when enabled, provides pop-up notifications of build failures with a link to view a report.



To configure this setting, navigate to [Preview features](#), find **Task Insights for Failed Pipeline Runs**, and choose the desired setting.



Next steps

Learn more about configuring pipelines in the language of your choice:

- [.NET Core](#)
- [Go](#)
- [Java](#)
- [Node.js](#)
- [Python](#)
- [Containers](#) and [Container jobs](#)

Learn more about building [Azure Repos](#) and [GitHub](#) repositories.

To learn what else you can do in YAML pipelines, see [Customize your pipeline](#), and for a complete reference see [YAML schema reference](#).

Key concepts for new Azure Pipelines users

4/7/2021 • 5 minutes to read • [Edit Online](#)

Learn about the key concepts and components that make up a pipeline. Understanding the basic terms and parts of a pipeline can help you deliver better code more efficiently and reliably.

Key concepts overview

- A [trigger](#) tells a Pipeline to run.
- A [pipeline](#) is made up of one or more [stages](#). A pipeline can deploy to one or more [environments](#).
- A [stage](#) is a way of organizing [jobs](#) in a pipeline and each stage can have one or more jobs.
- Each [job](#) runs on one [agent](#). A job can also be agentless.
- Each [agent](#) runs a job that contains one or more [steps](#).
- A [step](#) can be a [task](#) or [script](#) and is the smallest building block of a pipeline.
- A [task](#) is a pre-packaged script that performs an action, such as invoking a REST API or publishing a build artifact.
- An [artifact](#) is a collection of files or packages published by a [run](#).

Azure Pipelines terms

Agent

When your build or deployment runs, the system begins one or more jobs. An agent is computing infrastructure with installed agent software that runs one job at a time. For example, your job could run on a Microsoft-hosted Ubuntu agent.

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

Approvals

[Approvals](#) define a set of validations required before a deployment runs. Manual approval is a common check performed to control deployments to production environments. When checks are configured on an environment, pipelines will stop before starting a stage that deploys to the environment until all the checks are completed successfully.

Artifact

An artifact is a collection of files or packages published by a run. Artifacts are made available to subsequent tasks, such as distribution or deployment. For more information, see [Artifacts in Azure Pipelines](#).

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 include 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 group

A deployment group is a set of deployment target machines that have agents installed. A deployment group is just another grouping of agents, like an [agent pool](#). You can set the deployment targets in a pipeline for a job using a deployment group. Learn more about provisioning agents for [deployment groups](#).

Environment

An [environment](#) is a collection of resources, where you deploy your application. It can contain one or more virtual machines, containers, web apps, or any service that's used to host the application being developed. A pipeline might deploy the app to one or more environments after build is completed and tests are run.

Job

A stage 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. Jobs are most useful when you want to run a series of steps in different environments. For example, you might want to build two configurations - x86 and x64. In this case, you have one stage and two jobs. One job would be for x86 and the other job would be for x64.

Pipeline

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

Run

A run represents one execution of a pipeline. It collects the logs associated with running the steps and the results of running tests. During a run, Azure Pipelines will first process the pipeline and then send the run to one or more agents. Each agent will run jobs. Learn more about the [pipeline run sequence](#).

Script

A script runs code as a step in your pipeline using command line, PowerShell, or Bash. You can write [cross-platform scripts](#) for macOS, Linux, and Windows. Unlike a [task](#), a script is custom code that is specific to your pipeline.

Stage

A [stage](#) is a logical boundary in the pipeline. It can be used to mark separation of concerns (for example, Build, QA, and production). Each stage contains one or more jobs. When you define multiple stages in a pipeline, by default, they run one after the other. You can specify the conditions for when a stage runs. When you are thinking about whether you need a stage, ask yourself:

- Do separate groups manage different parts of this pipeline? For example, you could have a test manager that manages the jobs that relate to testing and a different manager that manages jobs related to production deployment. In this case, it makes sense to have separate stages for testing and production.

- Is there a set of [approvals](#) that are connected to a specific job or set of jobs? If so, you can use stages to break your jobs into logical groups that require approvals.
- Are there jobs that need to run a long time? If you have part of your pipeline that will have an extended run time, it makes sense to divide them into their own stage.

Step

A step is the smallest building block of a pipeline. For example, a pipeline might consist of build and test steps. A step can either be a script or a task. A task is simply a pre-created script offered as a convenience to you. To view the available tasks, see the [Build and release tasks](#) reference. For information on creating custom tasks, see [Create a custom task](#).

Task

A [task](#) is the building block for defining automation in a pipeline. A task is packaged script or procedure that has been abstracted with a set of inputs.

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. For more information, see [build triggers](#) and [release triggers](#).

About the authors

- [Dave Jarvis](#) contributed to the key concepts overview graphic.

YAML pipeline editor

6/24/2021 • 4 minutes to read • [Edit Online](#)

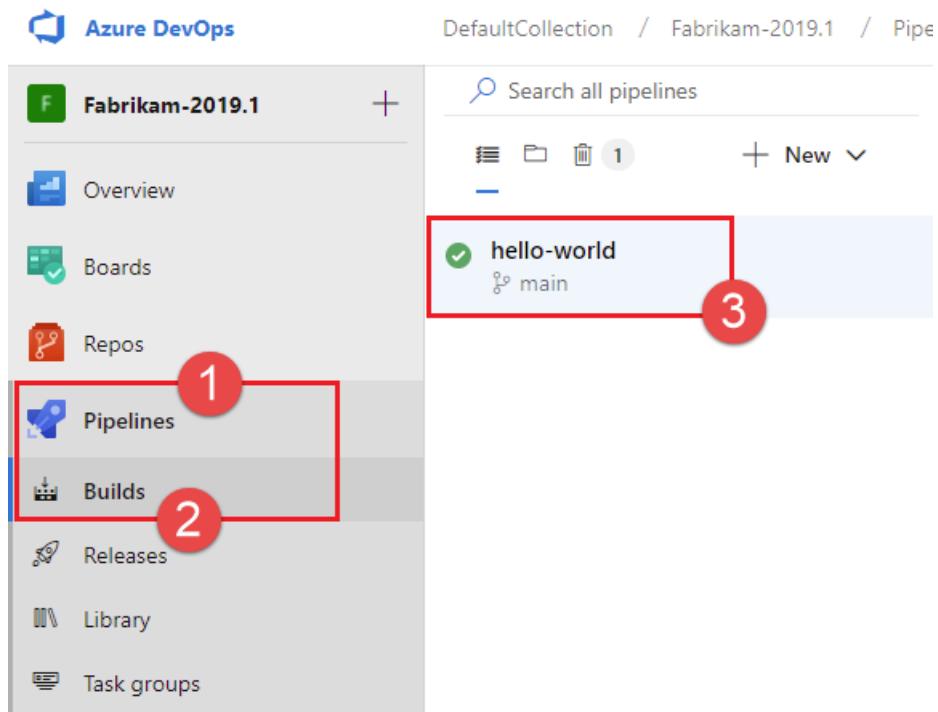
Azure Pipelines provides a YAML pipeline editor based on the [Monaco Editor](#) that you can use to author and edit your pipelines from the Azure DevOps portal. The editor provides tools like Intellisense support and a task assistant to provide guidance when editing a pipeline.

IMPORTANT

The YAML pipeline editor was introduced in Azure DevOps Server 2019 Update 1. If you are using Azure DevOps Server 2019 RTW, the YAML pipeline editor described in this article is not available, but you can edit your YAML using the text editor of your choice and check it into your pipeline's repository.

Edit a YAML pipeline

To access the YAML pipeline editor, choose **Pipelines**, **Builds**, and select the pipeline to edit.



Choose **Edit** to edit your pipeline in the YAML pipeline editor.

hello-world

Edit**Queue**

History Analytics

Commit

Build #

Merged PR 2: Set up CI with Azure Pipelines
CI build for Jamal Hartnett

✓ 20210615.2

Set up CI with Azure Pipelines
Pull request build for devops

✗ 20210615.1

Edit your pipeline using [Intellisense](#) keyboard shortcuts and the [task assistant](#) for guidance.

← JobPoolTest

Save

main ▾ ◆ JobPoolTest / azure-pipelines.yml *

```

1 trigger: none
2
3 # Implicit job
4 #pool: default
5 #
6 #steps:
7 #- script: Echo hello
8
9 # Explicit job
10 jobs:
11 - job: A2
12   pool: default
13   steps:
14     - script: echo "Stage A Job B1"
15
16     - cancelTimeoutInMinutes
17     - condition
18     - container
19     - continueOnError
20     - dependsOn
21     - displayName
22     - services
23     - strategy
24     - timeoutInMinutes
25     - variables
26     - workspace
27     - script: echo "Stage A Job B1"
28

```

Tasks

Search tasks

- .NET Core
Build, test, package, or publish a dotnet applicatio...
- Android signing
Sign and align Android APK files
- Ant
Build with Apache Ant
- App Center distribute
Distribute app builds to testers and users via Visu...
- App Center test
Test app packages with Visual Studio App Center
- Archive files
Compress files into .7z, .tar.gz, or .zip
- Azure App Service deploy
Deploy to Azure App Service a web, mobile, or AP...
- Azure App Service manage
Start, stop, restart, slot swap, install site extension...

To access the YAML pipeline editor, choose [Pipelines](#) and select the pipeline to edit. You can browse pipelines by [Recent](#), [All](#), and [Runs](#). For more information, see [Pipelines landing page](#).

The screenshot shows the Azure DevOps Pipelines interface. On the left, there's a sidebar with icons for Overview, Boards, Repos, Pipelines (highlighted with a red box and numbered 1), Pipelines (highlighted with a red box and numbered 2), and Pipelines. The main area is titled 'Pipelines' with tabs for 'Recent', 'All', and 'Runs'. Below this is a section titled 'Recently run pipelines' with a single entry: 'Pipeline' (FabrikamFiber) with a green checkmark icon and a red box around it, numbered 3.

Choose Edit to edit your pipeline in the YAML pipeline editor.

The screenshot shows the 'Runs' tab of the Pipelines page. It includes a back arrow, the project name 'FabrikamFiber', and buttons for 'Edit' (highlighted with a red box), 'Run pipeline', and a three-dot menu. Below this is a table with columns 'Description' and 'Stages'. The table lists four recent pipeline runs:

Description	Stages
#20210609.3 Update azure-pipelines.yml for Azure Pipelines Manually triggered for main 61bba8c	✓ Wednesday 21s
#20210609.2 Update azure-pipelines.yml for Azure Pipelines Manually triggered for main bc6c105	✓ Wednesday 30s
#20210609.1 Update azure-pipelines.yml for Azure Pipelines Manually triggered for main 53eb92b	✓ Wednesday 45s
#20210520.1 Updated README.md Individual CI for new-branch 1ada387	✓ May 20 13s

Edit your pipeline using [Intellisense](#) and the [task assistant](#) for guidance.

← azure-cli-example

Variables Run ...

main ◇ azure-cl... / azure-pipelines.yml

```
30 pool: default
31
32 steps:
33 # Specify python version and install if needed
34 - task: UsePythonVersion@0
35   condition: false
36   inputs:
37     versionSpec: '3.x'
38     architecture: 'x64'
39
40 # Update pip to latest
41 - bash: python -m pip install --upgrade pip
42   condition: false
43   displayName: 'Upgrade pip'
44
45 < container
46 < continueOnError
47 < name
48 < parameters
49 < pr
50 < resources
51 < schedules
52 < services
53 < strategy
54 < variables
55 < workspace
```

Tasks

Search tasks

- .NET Core
- Android signing
- Ant
- App Center distribute
- App Center test
- Archive files
- ARM template deployment
- Azure App Service deploy

Choose **Save** to save your pipeline. You can commit directly to your branch, or create a new branch and optionally start a pull request.

Save



Saving will commit /azure-pipelines.yml to the repository.

Commit message

Update azure-pipelines.yml for Azure Pipelines

Optional extended description

Add an optional description...

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

Cancel

Save

Keyboard shortcuts

The YAML pipeline editor provides several types of keyboard shortcuts.

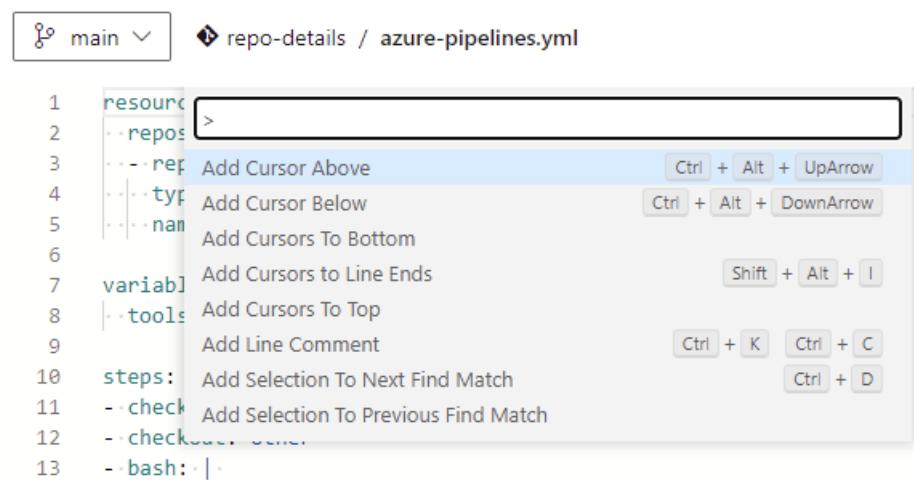
Press **Ctrl+Space** for Intellisense support while you are editing the YAML pipeline.

```

10   jobs:
11     - job: A2
12       pool: default
13       steps:
14         - script: echo "Stage A Job B1"
15
16         #< cancelTimeoutInMinutes
17         #< condition
18         #< container
19         #< continueOnError
20         #< dependsOn
21         #< displayName
22         #< services
23         #< strategy
24         #< timeoutInMinutes
25         #< variables
26         #< workspace
27         #< - script: echo "Stage A Job B1"

```

Press F1 (Fn+F1 on Mac) to display the command palette and view the available keyboard shortcuts.



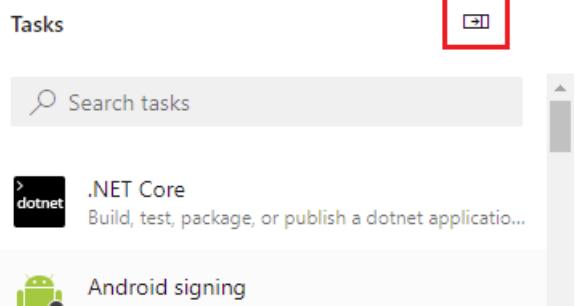
Task assistant

The task assistant provides a method for adding tasks to your YAML pipeline.

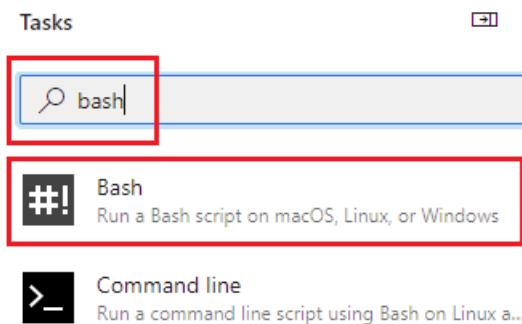
To display the task assistant, edit your YAML pipeline and choose **Show assistant**.

The screenshot shows the Azure DevOps interface with a code editor containing a YAML pipeline. The task assistant is open on the right side of the screen, showing a list of available tasks. A red box highlights the 'Show assistant' button at the top of the assistant window.

To hide the task assistant, choose **Hide assistant**.



To use the task assistant, browse or search for tasks in the **Tasks** pane.



Select the desired task and configure its inputs.

← Bash ⓘ

Type ⓘ

File Path

Inline

Script *

```
# Write your commands here
echo 'Hello world from the task assistant!'
```

Advanced ▾

About this task

Add

A screenshot of the 'Bash' task configuration page. At the top left is a back arrow and the word 'Bash' with an info icon. Below this is a 'Type' section with a radio button for 'File Path' and one for 'Inline' (which is selected). Underneath is a 'Script' field with a red border, containing the code '# Write your commands here' and 'echo 'Hello world from the task assistant!''. At the bottom left is an 'About this task' link. In the bottom right corner is a large blue 'Add' button with a red border.

Choose **Add** to insert the task YAML into your pipeline.

```
main ✘ FabrikamFiber / azure-pipelines.yml *
```

```
1 pool:
2   - vmImage: ubuntu-latest
3
4 steps:
5   - script:
6     - echo "Hello"
7     - echo "World!"
8
9   - task: Bash@3
10    inputs:
11      targetType: 'inline'
12      script:
13        # Write your commands here
14
15        echo 'Hello world from the task assistant!'
```

Edit the inserted YAML to make additional configuration changes to the task.

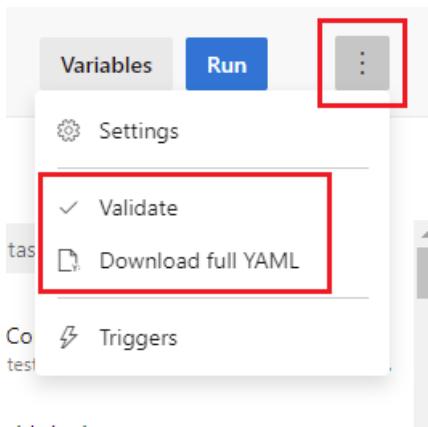
```
main ✘ FabrikamFiber / azure-pipelines.yml *
```

```
1 pool:
2   - vmImage: ubuntu-latest
3
4 steps:
5   - script:
6     - echo "Hello"
7     - echo "World!"
8
9   - task: Bash@3
10    inputs:
11      targetType: 'inline'
12      script:
13        # Write your commands here
14
15        echo 'Hello world from the task assistant!'
```

You can edit the YAML to make additional configuration changes to the task, or you can choose **Settings** above the task in the YAML pipeline editor to configure the inserted task in the task assistant.

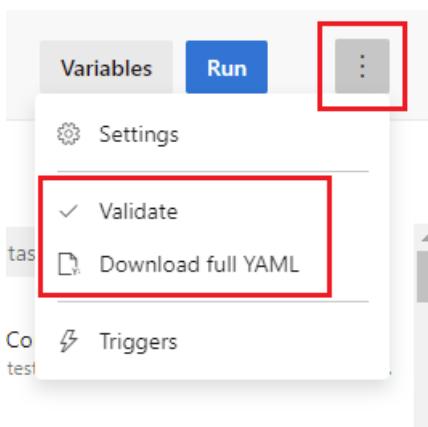
Validate

When editing a YAML pipeline, you can validate your changes by choosing **More actions, Validate**. You can use this feature to catch syntax errors in your pipeline that will prevent it from starting.



Download full YAML

You can [preview the fully parsed YAML document without committing or running the pipeline](#) by choosing More actions, Download full YAML.

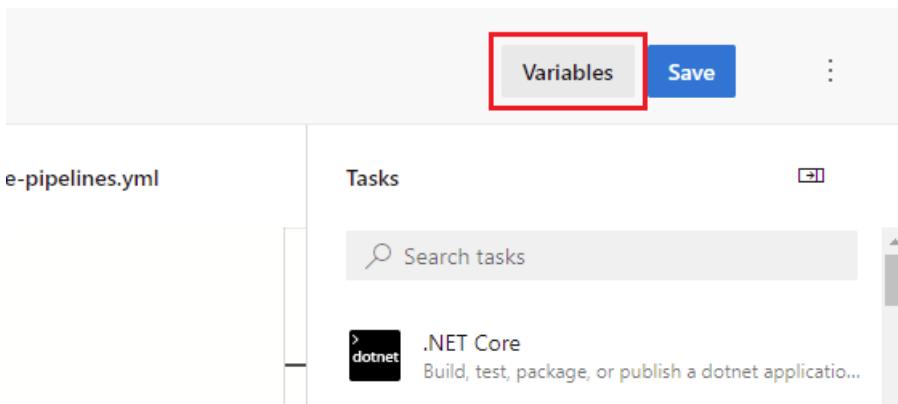


Download full YAML calls the [Runs](#) Azure DevOps REST API for Azure Pipelines, and initiates a download of the rendered YAML from the editor.

Manage pipeline variables

Use variables to store values or encrypted secrets separately from your repository.

To manage pipeline variables, edit your YAML pipeline and choose **Variables**.



Choose **New variable** to add your first variable and **Add** to add subsequent variables. Choose the variable name to edit the variable, and hover the mouse over the variable and choose **Delete** to delete a variable.

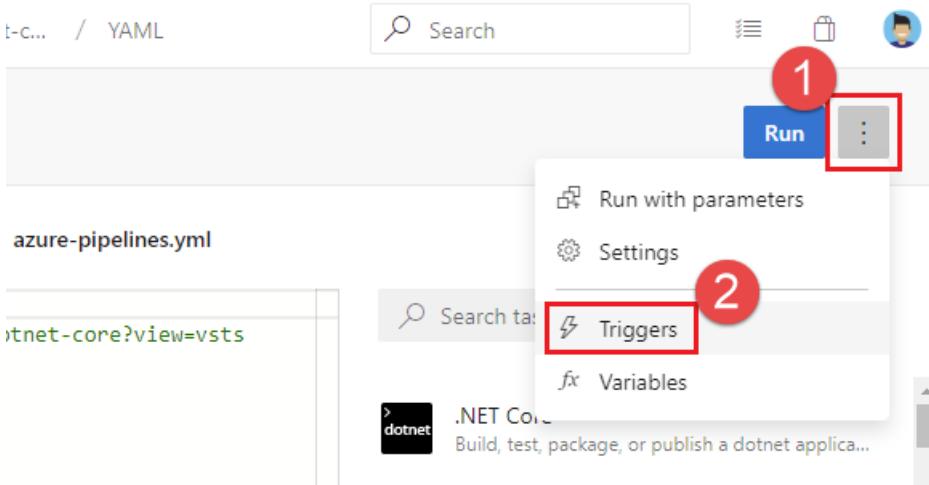
Variables

The screenshot shows the 'Variables' editor interface. At the top is a search bar labeled 'Search variables' and a '+' button. Below the search bar are two variable entries: 'myVariable' with the value '= myValue' and 'mySecret' with the value '*****'. There are icons for edit and delete next to each entry. At the bottom are 'Learn about variables', 'Cancel', and 'Save' buttons.

In addition to managing your variables directly from the YAML pipeline editor, you can also manage them in the pipeline settings UI.

When using Azure DevOps 2019 Update 1, you can manage your pipeline variables using the pipeline settings UI.

To manage YAML pipeline variables in the pipeline settings UI, edit the pipeline and choose **More actions**, **Triggers**.



Choose **Variables** to manage your pipeline variables.

The screenshot shows the 'Variables' tab in the pipeline settings. The top navigation bar includes 'YAML', 'Variables' (which is highlighted with a red box), 'Triggers', 'History', 'Save & queue', 'Discard', 'Summary', and a back arrow. The main area is divided into sections: 'Pipeline variables', 'Variable groups', and 'Predefined variables'. The 'Pipeline variables' section lists variables with their names and values:

Name ↑	Value
mySecret	*****
myVariable	myValue
system.collectionId	123abceb-be37-
system.definitionId	10
system.teamProject	FabrikamFiber

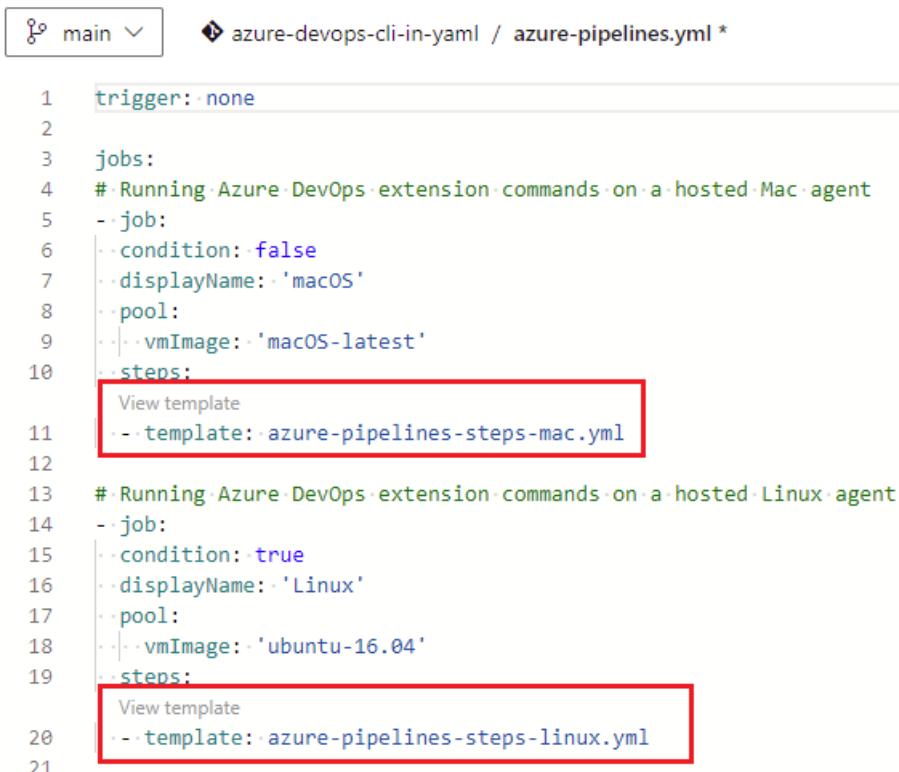
For more information on working with pipeline variables, see [Define variables](#).

View and edit templates

[Templates](#) are a commonly used feature in YAML pipelines. They are an easy way to share pipeline snippets. They are also a powerful mechanism in verifying or enforcing [security and governance](#) through your pipeline.

Azure Pipelines supports a YAML editor which can be handy when editing your pipeline. Previously, the editor did not support templates. Authors of YAML pipelines could not get intellisense assistance when using a template. With this release, we are previewing support for templates in the YAML editor. To enable this preview, [navigate to preview features](#) in your Azure DevOps organization, and enable [YAML templates editor](#).

As you edit your main Azure Pipelines YAML file, you can either *include* or *extend* a template. As you type in the name of your template, you may be prompted to validate your template. Once validated, the YAML editor understands the schema of the template including the input parameters.



```
1 trigger: none
2
3 jobs:
4 # Running Azure DevOps extension commands on a hosted Mac agent
5 - job:
6   condition: false
7   displayName: 'macOS'
8   pool:
9     vmImage: 'macOS-latest'
10  steps:
11    View template
12    --template: azure-pipelines-steps-mac.yml
13
14 # Running Azure DevOps extension commands on a hosted Linux agent
15 - job:
16   condition: true
17   displayName: 'Linux'
18   pool:
19     vmImage: 'ubuntu-16.04'
20   steps:
21     View template
22     --template: azure-pipelines-steps-linux.yml
```

Post validation, you can navigate into the template by choosing **View template**, which opens the template in a new browser tab. You will be able to make changes to the template using all the features of the YAML editor.

IMPORTANT

This feature is in preview. There are known limitations. If the template has required parameters that are not provided as inputs in the main YAML file, then the validation fails and prompts you to provide those inputs. In addition, you cannot create a new template from the editor. You can only use or edit existing templates.

Next steps

- [Learn how to navigate and view your pipelines](#)
- [Create your first pipeline](#)
- [Customize your pipeline](#)

Supported source repositories

11/2/2020 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Azure Pipelines, Azure DevOps Server, and TFS integrate with a number of version control systems. When you use any of these version control systems, you can configure a pipeline to build, test, and deploy your application.

YAML pipelines are a new form of pipelines that have been introduced in Azure DevOps Server 2019 and in Azure Pipelines. YAML pipelines only work with certain version control systems. The following table shows all the supported version control systems and the ones that support YAML pipelines.

REPOSITORY TYPE	AZURE PIPELINES (YAML)	AZURE PIPELINES (CLASSIC EDITOR)	AZURE DEVOPS SERVER 2019, TFS 2018, TFS 2017, TFS 2015.4	TFS 2015 RTM
Azure Repos Git	Yes	Yes	Yes	Yes
Azure Repos TFVC	No	Yes	Yes	Yes
GitHub	Yes	Yes	No	No
GitHub Enterprise Server	Yes	Yes	TFS 2018.2 and higher	No
Bitbucket Cloud	Yes	Yes	No	No
Bitbucket Server	No	Yes	Yes	Yes
Subversion	No	Yes	Yes	No

Build Azure Repos Git or TFS Git repositories

3/25/2021 • 33 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Azure Pipelines can automatically build and validate every pull request and commit to your Azure Repos Git repository.

Choose a repository to build

- [YAML](#)
- [Classic](#)

You create a new pipeline by first selecting a repository and then a YAML file in that repository. The repository in which the YAML file is present is called `self` repository. By default, this is the repository that your pipeline builds.

You can later configure your pipeline to check out a different repository or multiple repositories. To learn how to do this, see [multi-repo checkout](#).

YAML pipelines are not available in TFS.

Azure Pipelines must be granted access to your repositories to trigger their builds and fetch their code during builds. Normally, a pipeline has access to repositories in the same project. But, if you wish to access repositories in a different project, then you need to update the permissions granted to [job access tokens](#).

CI triggers

Continuous integration (CI) triggers cause a pipeline to run whenever you push an update to the specified branches or you push specified tags.

- [YAML](#)
- [Classic](#)

YAML pipelines are configured by default with a CI trigger on all branches.

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 wildcard (for example, `releases/*`). See [Wildcards](#) for information on the wildcard syntax.

NOTE

You cannot use `variables` in triggers, as variables are evaluated at runtime (after the trigger has fired).

NOTE

If you use `templates` to author YAML files, then you can only specify triggers in the main YAML file for the pipeline. You cannot specify triggers in the template files.

For more complex triggers that use `exclude` or `batch`, you must use the full syntax as shown in the following example.

```
# specific branch build
trigger:
  branches:
    include:
      - master
      - releases/*
    exclude:
      - releases/old*
```

In the above example, the pipeline will be triggered if a change is pushed to master or to any releases branch. However, it won't be triggered if a change is made to a releases branch that starts with `old`.

If you specify an `exclude` clause without an `include` clause, then it is equivalent to specifying `*` in the `include` clause.

In addition to specifying branch names in the `branches` lists, you can also configure triggers based on tags by using the following format:

```
trigger:
  branches:
    include:
      - refs/tags/{tagname}
    exclude:
      - refs/tags/{othertagname}
```

If you don't specify any triggers, the default is as if you wrote:

```
trigger:
  branches:
    include:
      - '*' # must quote since "*" is a YAML reserved character; we want a string
```

IMPORTANT

When you specify a trigger, it replaces the default implicit trigger, and only pushes to branches that are explicitly configured to be included will trigger a pipeline. Includes are processed first, and then excludes are removed from that list.

Batching CI runs

If you have many team members uploading changes often, you may want to reduce the number of runs you start. If you set `batch` to `true`, when a pipeline is running, the system waits until the run is completed, then starts another run with all changes that have not yet been built.

```
# specific branch build with batching
trigger:
  batch: true
  branches:
    include:
      - master
```

To clarify this example, let us say that a push A to master caused the above pipeline to run. While that pipeline is running, additional pushes B and C occur into the repository. These updates do not start new independent runs immediately. But after the first run is completed, all pushes until that point of time are batched together and a new run is started.

NOTE

If the pipeline has multiple jobs and stages, then the first run should still reach a terminal state by completing or skipping all its jobs and stages before the second run can start. For this reason, you must exercise caution when using this feature in a pipeline with multiple stages or approvals. If you wish to batch your builds in such cases, it is recommended that you split your CI/CD process into two pipelines - one for build (with batching) and one for deployments.

Paths

You can specify file paths to include or exclude.

```
# specific path build
trigger:
  branches:
    include:
      - master
      - releases/*
paths:
  include:
    - docs
  exclude:
    - docs/README.md
```

When you specify paths, you must explicitly specify branches to trigger on. You can't trigger a pipeline with only a path filter; you must also have a branch filter, and the changed files that match the path filter must be from a branch that matches the branch filter.

Tips:

- Wild cards are not supported with path filters.
- Paths are always specified relative to the root of the repository.
- If you don't set path filters, then the root folder of the repo is implicitly included by default.
- If you exclude a path, you cannot also include it unless you qualify it to a deeper folder. For example if you exclude `/tools` then you could include `/tools/trigger-runs-on-these`
- The order of path filters doesn't matter.
- Paths in Git *are case-sensitive*. Be sure to use the same case as the real folders.
- You cannot use [variables](#) in paths, as variables are evaluated at runtime (after the trigger has fired).

Tags

In addition to specifying tags in the `branches` lists as covered in the previous section, you can directly specify tags to include or exclude:

```
# specific tag
trigger:
  tags:
    include:
      - v2.0
    exclude:
      - v2.0
```

If you don't specify any tag triggers, then by default, tags will not trigger pipelines.

IMPORTANT

If you specify tags in combination with branch filters, the trigger will fire if either the branch filter is satisfied or the tag filter is satisfied. For example, if a pushed tag satisfies the branch filter, the pipeline triggers even if the tag is excluded by the tag filter, because the push satisfied the branch filter.

Opting out of CI

Disabling the CI trigger

You can opt out of CI triggers entirely by specifying `trigger: none`.

```
# A pipeline with no CI trigger
trigger: none
```

IMPORTANT

When you push a change to a branch, the YAML file in that branch is evaluated to determine if a CI run should be started.

For more information, see [Triggers](#) in the [YAML schema](#).

YAML pipelines are not available in TFS.

Skipping CI for individual commits

You can also tell Azure Pipelines to skip running a pipeline that a commit would normally trigger. Just include `***NO_CI***` in the commit message of the HEAD commit and Azure Pipelines will skip running CI.

You can also tell Azure Pipelines to skip running a pipeline that a commit would normally trigger. Just include `[skip ci]` in the commit message or description of the HEAD commit and Azure Pipelines will skip running CI.

You can also use any of the variations below.

- `[skip ci]` or `[ci skip]`
- `skip-checks: true` or `skip-checks:true`
- `[skip azurepipelines]` or `[azurepipelines skip]`
- `[skip azpipelines]` or `[azpipelines skip]`
- `[skip azp]` or `[azp skip]`
- `***NO_CI***`

Using the trigger type in conditions

It is a common scenario to run different steps, jobs, or stages in your pipeline depending on the type of trigger that started the run. You can do this using the system variable `Build.Reason`. For example, add the following condition to your step, job, or stage to exclude it from PR validations.

```
condition: and(succeeded(), ne(variables['Build.Reason'], 'PullRequest'))
```

Behavior of triggers when new branches are created

It is common to configure multiple pipelines for the same repository. For instance, you may have one pipeline to build the docs for your app and another to build the source code. You may configure CI triggers with appropriate branch filters and path filters in each of these pipelines. For instance, you may want one pipeline to trigger when you push an update to the `docs` folder, and another one to trigger when you push an update to your application code. In these cases, you need to understand how the pipelines are triggered when a new branch is created.

Here is the behavior when you push a new branch (that matches the branch filters) to your repository:

- If your pipeline has path filters, it will be triggered only if the new branch has changes to files that match that path filter.
- If your pipeline does not have path filters, it will be triggered even if there are no changes in the new branch.

Wildcards

When specifying a branch or tag, you may use an exact name or a wildcard. Wildcards patterns allow `*` to match zero or more characters and `?` to match a single character.

- If you start your pattern with `*` in a YAML pipeline, you must wrap the pattern in quotes, like `"*-releases"`.
- For branches and tags:
 - A wildcard may appear anywhere in the pattern.
- For paths:
 - You may include `*` as the final character, but it doesn't do anything differently from specifying the directory name by itself.
 - You may **not** include `*` in the middle of a path filter, and you may not use `?`.

```
trigger:
  branches:
    include:
      - master
      - releases/*
      - feature/*
    exclude:
      - releases/old*
      - feature/*-working
  paths:
    include:
      - '*' # same as '/' for the repository root
    exclude:
      - 'docs/*' # same as 'docs/'
```

PR triggers

Pull request (PR) triggers cause a pipeline to run whenever a pull request is opened with one of the specified target branches, or when changes are pushed to such a pull request. In Azure Repos Git, this functionality is implemented using branch policies. To enable pull request validation in Azure Git Repos, navigate to the branch policies for the desired branch, and configure the [Build validation policy](#) for that branch. For more information, see [Configure branch policies](#).

NOTE

To configure validation builds for an Azure Repos Git repository, you must be a project administrator of its project.

NOTE

Draft pull requests do not trigger a pipeline even if you configure a branch policy.

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 organization that your project is part of.

Limit job authorization scope

Azure Pipelines provides several security settings to configure the job authorization scope that your pipelines run with.

- [Limit job authorization scope to current project](#)
- [Limit job authorization scope to referenced Azure DevOps repositories](#)

Limit job authorization scope to current project

Azure Pipelines provides two **Limit job authorization scope to current project** settings:

- **Limit job authorization scope to current project for non-release pipelines** - This setting applies to YAML pipelines and classic build pipelines. This setting does not apply to classic release pipelines.
- **Limit job authorization scope to current project for release pipelines** - This setting applies to [classic release pipelines](#) only.

Pipelines run with collection scoped access tokens unless the relevant setting for the pipeline type is enabled. The **Limit job authorization scope** settings allow you to reduce the scope of access for all pipelines to the current project. This can impact your pipeline if you are accessing an Azure Repos Git repository in a different project in your organization.

If your Azure Repos Git repository is in a different project than your pipeline, and the **Limit job authorization scope** setting for your pipeline type is enabled, you must grant permission to the build service identity for your pipeline to the second project. For more information, see [Manage build service account permissions](#).

Azure Pipelines provides a security setting to configure the job authorization scope that your pipelines run with.

- **Limit job authorization scope to current project** - This setting applies to YAML pipelines and classic build pipelines. This setting does not apply to [classic release pipelines](#).

Pipelines run with collection scoped access tokens unless **Limit job authorization scope to current project** is enabled. This setting allows you to reduce the scope of access for all pipelines to the current project. This can impact your pipeline if you are accessing an Azure Repos Git repository in a different project in your organization.

If your Azure Repos Git repository is in a different project than your pipeline, and the **Limit job authorization scope** setting is enabled, you must grant permission to the build service identity for your pipeline to the second project. For more information, see [Job authorization scope](#).

For more information on **Limit job authorization scope**, see [Understand job access tokens](#).

Limit job authorization scope to referenced Azure DevOps repositories

Pipelines can access any Azure DevOps repositories in authorized projects, as described in the previous [Limit job authorization scope to current project](#) section, unless **Limit job authorization scope to referenced Azure DevOps repositories** is enabled. With this option enabled, you can reduce the scope of access for all pipelines to only Azure DevOps repositories explicitly referenced by a `checkout` step or a `uses` statement in the pipeline job that uses that repository.

To configure this setting, navigate to **Pipelines**, **Settings** at either **Organization settings** or **Project settings**. If enabled at the organization level, the setting is grayed out and unavailable at the project settings level.

IMPORTANT

Limit job authorization scope to referenced Azure DevOps repositories is enabled by default for new organizations and projects created after May 2020.

When **Limit job authorization scope to referenced Azure DevOps repositories** is enabled, your YAML pipelines must explicitly reference any Azure Repos Git repositories you want to use in the pipeline as a [checkout step](#) in the job that uses the repository. You won't be able to fetch code using scripting tasks and git commands for an Azure Repos Git repository unless that repo is first explicitly referenced.

There are a few exceptions where you don't need to explicitly reference an Azure Repos Git repository before using it in your pipeline when **Limit job authorization scope to referenced Azure DevOps repositories** is enabled.

- If you do not have an explicit checkout step in your pipeline, it is as if you have a `checkout: self` step, and the `self` repository is checked out.
- If you are using a script to perform read-only operations on a repository in a public project, you don't need to reference the public project repository in a `checkout` step.
- If you are using a script that provides its own authentication to the repo, such as a PAT, you don't need to reference that repository in a `checkout` step.

For example, when **Limit job authorization scope to referenced Azure DevOps repositories** is enabled, if your pipeline is in the `FabrikamProject/Fabrikam` repo in your organization, and you want to use a script to check out the `FabrikamProject/FabrikamTools` repo, you must either reference this repository in a `checkout` step or with a `uses` statement.

If you are already checking out the `FabrikamTools` repository in your pipeline using a checkout step, you may subsequently use scripts to interact with that repository, such as checking out different branches.

```
steps:  
- checkout: git://FabrikamFiber/FabrikamTools # Azure Repos Git repository in the same organization  
- script: Do something with that repo  
  
# Or you can reference it with a uses statement in the job  
uses:  
- FabrikamTools # Repository reference to FabrikamTools  
  
steps:  
- script: Do something with that repo like clone it
```

NOTE

For many scenarios, multi-repo checkout can be leveraged, removing the need to use scripts to check out additional repositories in your pipeline. For more information, see [Check out multiple repositories in your pipeline](#).

Checkout

When a pipeline is triggered, Azure Pipelines pulls your source code from the Azure Repos Git repository. You can control various aspects of how this happens.

Preferred version of Git

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.

Checkout path

- [YAML](#)
- [Classic](#)

If you are checking out a single repository, by default, your source code will be checked out into a directory called `s`. For YAML pipelines, you can change this by specifying `checkout` with a `path`. The specified path is relative to `$(Agent.BuildDirectory)`. For example: if the checkout path value is `mycustompath` and `$(Agent.BuildDirectory)` is `C:\agent_work\1`, then the source code will be checked out into `C:\agent_work\1\mycustompath`.

If you are using multiple `checkout` steps and checking out multiple repositories, and not explicitly specifying the folder using `path`, each repository is placed in a subfolder of `s` named after the repository. For example if you check out two repositories named `tools` and `code`, the source code will be checked out into `C:\agent_work\1\s\tools` and `C:\agent_work\1\s\code`.

Please note that the checkout path value cannot be set to go up any directory levels above

`$(Agent.BuildDirectory)`, so `path..\anotherpath` will result in a valid checkout path (i.e. `C:\agent_work\1\anotherpath`), but a value like `..\invalidpath` will not (i.e. `C:\agent_work\invalidpath`).

You can configure the `path` setting in the [Checkout](#) step of your pipeline.

```
steps:  
- 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  
  path: string # path to check out source code, relative to the agent's build directory (e.g. \_work\1)  
  persistCredentials: boolean # set to 'true' to leave the OAuth token in the Git config after the initial  
    fetch
```

Submodules

- [YAML](#)
- [Classic](#)

You can configure the `submodules` setting in the [Checkout](#) step of your pipeline if you want to download files from [submodules](#).

```
steps:  
- 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  
  path: string # path to check out source code, relative to the agent's build directory (e.g. \_work\1)  
  persistCredentials: boolean # set to 'true' to leave the OAuth token in the Git config after the initial  
    fetch
```

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 as the Azure Repos Git repo specified above. 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.
- Added by using a URL relative to the main repository. For example

- This one would be checked out:

```
git submodule add ../../../../FabrikamFiberProject/_git/FabrikamFiber FabrikamFiber
```

In this example the submodule refers to a repo (FabrikamFiber) in the same Azure DevOps organization, but in a different project (FabrikamFiberProject). 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. This requires that the job access token has access to the repository in the second project. If you restricted the job access token as explained in the section above, then you won't be able to do this. You can allow the job access token to access the repo in the second project by either (a) explicitly granting access to the project build service account in the second project or (b) using collection-scoped access tokens instead of project-scoped tokens for the entire organization. For more information about these options and their security implications, see [Access repositories, artifacts, and other resources](#).

- This one would not be checked out:

```
git submodule add https://fabrikam-fiber@dev.azure.com/fabrikam-fiber/FabrikamFiberProject/_git/FabrikamFiber FabrikamFiber
```

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 if your job access token does not have access to the repository in a different project.

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 prefixed 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 <BASE64_ENCODED_STRING>"  
submodule update --init --recursive
```

Be sure to replace "`<BASE64_ENCODED_STRING>`" with your Base64-encoded "pat:token" string.

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.

Shallow fetch

You may 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.

- [YAML](#)
- [Classic](#)

You can configure the `fetchDepth` setting in the [Checkout](#) step of your pipeline.

```
steps:  
- 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  
  path: string # path to check out source code, relative to the agent's build directory (e.g. \_work\1)  
  persistCredentials: boolean # set to 'true' to leave the OAuth token in the Git config after the initial  
  fetch
```

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 pipeline is started, 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.

Don't sync sources

You may 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.
-
- [YAML](#)
 - [Classic](#)

You can configure the **Don't sync sources** setting in the [Checkout](#) step of your pipeline, by setting

```
checkout: none .
```

```
steps:  
- checkout: none # Don't sync sources
```

NOTE

When you use this option, the agent also skips running Git commands that clean the repo.

Clean build

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.

- [YAML](#)
- [Classic](#)

You can configure the `clean` setting in the [Checkout](#) step of your pipeline.

```
steps:  
- 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  
  path: string # path to check out source code, relative to the agent's build directory (e.g. \_work\1)  
  persistCredentials: boolean # set to 'true' to leave the OAuth token in the Git config after the initial  
  fetch
```

When `clean` is set to `true` 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 -ffdx  
git reset --hard HEAD
```

For more options, you can configure the `workspace` setting of a [Job](#).

```
jobs:  
- job: string # name of the job, A-Z, a-z, 0-9, and underscore  
  ...  
  workspace:  
    clean: outputs | resources | all # what to clean up before the job runs
```

This gives the following clean options.

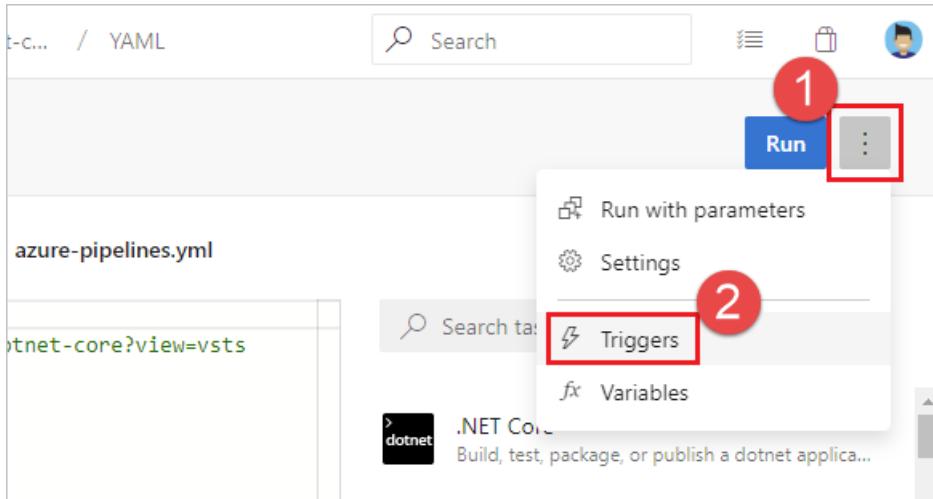
- **outputs:** Same operation as the clean setting described in the previous the checkout task, plus: Deletes and recreates `$(Build.BinariesDirectory)`. Note that the `$(Build.ArtifactStagingDirectory)` and `$(Common.TestResultsDirectory)` are always deleted and recreated prior to every build regardless of any of these settings.
- **resources:** Deletes and recreates `$(Build.SourcesDirectory)`. This results in initializing a new, local Git repository for every build.
- **all:** Deletes and recreates `$(Agent.BuildDirectory)`. This results in initializing a new, local Git repository for every build.

Label sources

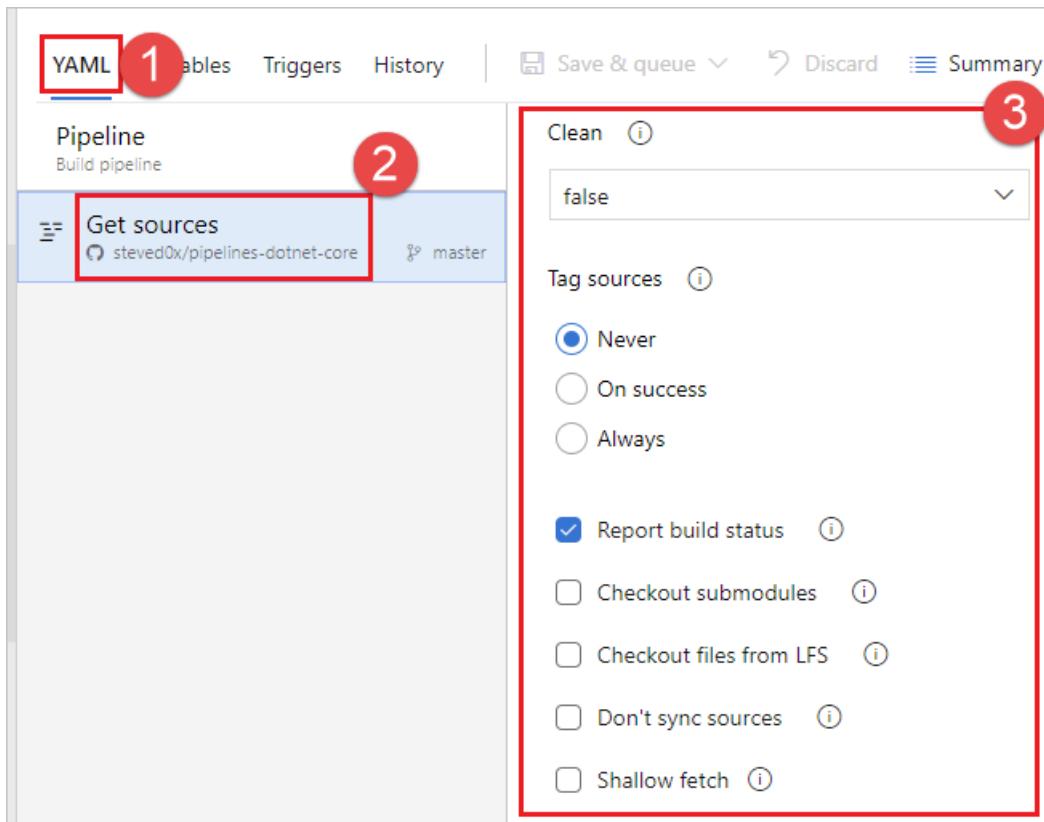
You may want to label your source code files to enable your team to easily identify which version of each file is included in the completed build. You also have the option to specify whether the source code should be labeled for all builds or only for successful builds.

- [YAML](#)
- [Classic](#)

You can't currently configure this setting in YAML but you can in the classic editor. When editing a YAML pipeline, you can access the classic editor by choosing either **Triggers** from the YAML editor menu.



From the classic editor, choose **YAML**, choose the **Get sources** task, and then configure the desired properties there.



In the **Tag 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. The tag is considered a build artifact since it is produced by the build. When the build is deleted either manually or through a retention policy, the tag is also deleted.

FAQ

Problems related to Azure Repos integration fall into three categories:

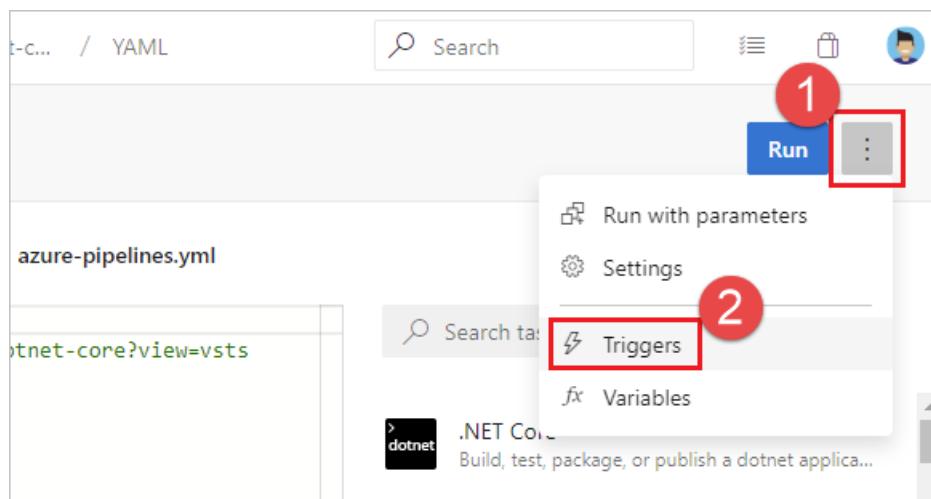
- **Failing triggers:** My pipeline is not being triggered when I push an update to the repo.
- **Failing checkout:** My pipeline is being triggered, but it fails in the checkout step.
- **Wrong version:** My pipeline runs, but it is using an unexpected version of the source/YAML.

Failing triggers

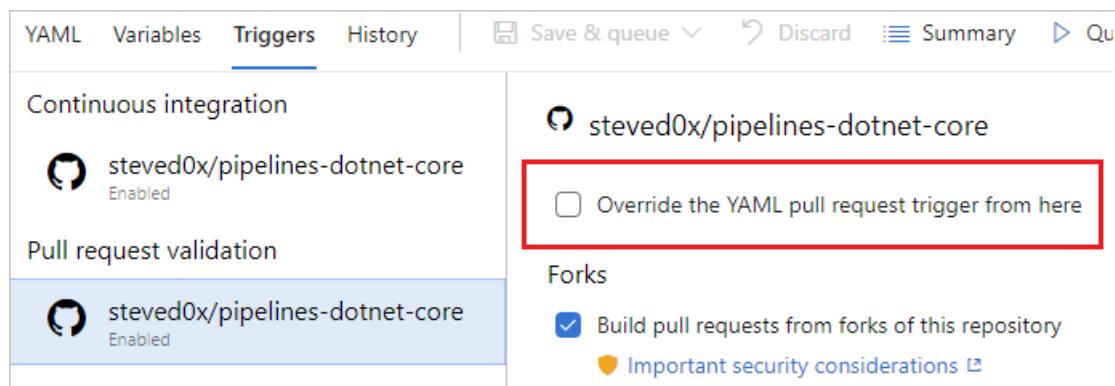
I just created a new YAML pipeline with CI/PR triggers, but the pipeline is not being triggered.

Follow each of these steps to troubleshoot your failing triggers:

- Are your YAML CI or PR triggers being [overridden by pipeline settings in the UI](#)? While editing your pipeline, choose ... and then Triggers.



Check the **Override the YAML trigger from here** setting for the types of trigger (Continuous integration or Pull request validation) available for your repo.



- Are you configuring the PR trigger in the YAML file or in branch policies for the repo? For an Azure Repos Git repo, you cannot configure a PR trigger in the YAML file. You need to use [branch policies](#).
- Is your pipeline paused or disabled? Open the editor for the pipeline, and then select **Settings** to check. If your pipeline is paused or disabled, then triggers do not work.

- Have you updated the YAML file in the correct branch? If you push an update to a branch, then the YAML file in that same branch governs the CI behavior. If you push an update to a source branch, then the YAML file resulting from merging the source branch with the target branch governs the PR behavior. Make sure that the YAML file in the correct branch has the necessary CI or PR configuration.
- Have you configured the trigger correctly? When you define a YAML trigger, you can specify both include and exclude clauses for branches, tags, and paths. Ensure that the include clause matches the details of your commit and that the exclude clause doesn't exclude them. Check the syntax for the triggers and make sure that it is accurate.
- Have you used variables in defining the trigger or the paths? That is not supported.
- Did you use templates for your YAML file? If so, make sure that your triggers are defined in the main YAML file. Triggers defined inside template files are not supported.
- Have you excluded the branches or paths to which you pushed your changes? Test by pushing a change to an included path in an included branch. Note that paths in triggers are case-sensitive. Make sure that you use the same case as those of real folders when specifying the paths in triggers.
- Do you have wildcards in your path filters? Understand the limitations of wildcards in your paths as described in this article.
- Did you just push a new branch? If so, the new branch may not start a new run. See the section "Behavior of triggers when new branches are created".

My CI or PR triggers have been working fine. But, they stopped working now.

First go through the troubleshooting steps in the previous question. Then, follow these additional steps:

- Do you have merge conflicts in your PR? For a PR that did not trigger a pipeline, open it and check whether it has a merge conflict. Resolve the merge conflict.
- Are you experiencing a delay in the processing of push or PR events? You can usually verify this by seeing if the issue is specific to a single pipeline or is common to all pipelines or repos in your project. If a push or a PR update to any of the repos exhibits this symptom, we might be experiencing delays in processing the update events. Check if we are experiencing a service outage on our [status page](#). If the status page shows an issue, then our team must have already started working on it. Check the page frequently for updates on the issue.

I do not want users to override the list of branches for triggers when they update the YAML file. How can I do this?

Users with permissions to contribute code can update the YAML file and include/exclude additional branches. As a result, users can include their own feature or user branch in their YAML file and push that update to a feature or user branch. This may cause the pipeline to be triggered for all updates to that branch. If you want to prevent this behavior, then you can:

1. Edit the pipeline in the Azure Pipelines UI.
2. Navigate to the **Triggers** menu.
3. Select **Override the YAML continuous Integration trigger from here**.
4. Specify the branches to include or exclude for the trigger.

When you follow these steps, any CI triggers specified in the YAML file are ignored.

I have multiple repositories in my YAML pipeline. How do I set up triggers for each repository?

See triggers in [Using multiple repositories](#).

Failing checkout

I see the following error in the log file during checkout step. How do I fix it?

```
remote: TF401019: The Git repository with name or identifier XYZ does not exist or you do not have  
permissions for the operation you are attempting.  
fatal: repository 'XYZ' not found  
##[error] Git fetch failed with exit code: 128
```

Follow each of these steps to troubleshoot your failing triggers:

- Does the repository still exist? First, make sure it does by opening it in the [Repos](#) page.
- Are you accessing the repository using a script? If so, check the [Limit job authorization scope to referenced Azure DevOps repositories](#) setting. When **Limit job authorization scope to referenced Azure DevOps repositories** is enabled, you won't be able to check out Azure Repos Git repositories using a script unless they are explicitly referenced first in the pipeline.
- What is the [job authorization scope](#) of the pipeline?
 - If the scope is **collection**:
 - This may be an intermittent error. Re-run the pipeline.
 - Someone may have removed the access to **Project Collection Build Service account**.
 - Go to the **project settings** of the project in which the repository exists. Select **Repos -> Repositories -> specific repository**.
 - Check if **Project Collection Build Service (your-collection-name)** exists in the list of users.
 - Check if that account has **Create tag** and **Read** access.
 - If the scope is **project**:
 - Is the repo in the same project as the pipeline?
 - Yes:
 - This may be an intermittent error. Re-run the pipeline.
 - Someone may have removed the access to **Project Build Service account**.
 - Go to the **project settings** of the project in which the repository exists. Select **Repos -> Repositories -> specific repository**.
 - Check if **your-project-name Build Service (your-collection-name)** exists in the list of users.
 - Check if that account has **Create tag** and **Read** access.
 - No:
 - Is your pipeline in a public project?
 - Yes: You cannot access resources outside of your public project. Make the project private.
 - No: You need to take additional steps to grant access. Let us say that your pipeline exists in project A and that your repository exists in project B.
 - Go to the project settings of the project in which the repository exists (B). Select **Repos -> Repositories -> specific repository**.
 - Add **your-project-name Build Service (your-collection-name)** to the list of users, where **your-project-name** is the name of the project in which your pipeline exists (A).
 - Give **Create tag** and **Read** access to the account.

Wrong version

A wrong version of the YAML file is being used in the pipeline. Why is that?

- For CI triggers, the YAML file that is in the branch you are pushing is evaluated to see if a CI build should be

run.

- For PR triggers, the YAML file resulting from merging the source and target branches of the PR is evaluated to see if a PR build should be run.

Related articles

- [Scheduled triggers](#)
- [Pipeline completion triggers](#)

Build GitHub repositories

5/13/2021 • 54 minutes to read • [Edit Online](#)

Azure Pipelines

Azure Pipelines can automatically build and validate every pull request and commit to your GitHub repository. This article describes how to configure the integration between GitHub and Azure Pipelines.

If you're new to Azure Pipelines integration with GitHub, follow the steps in [Create your first pipeline](#) to get your first pipeline working with a GitHub repository, and then come back to this article to learn more about configuring and customizing the integration between GitHub and Azure Pipelines.

Organizations and users

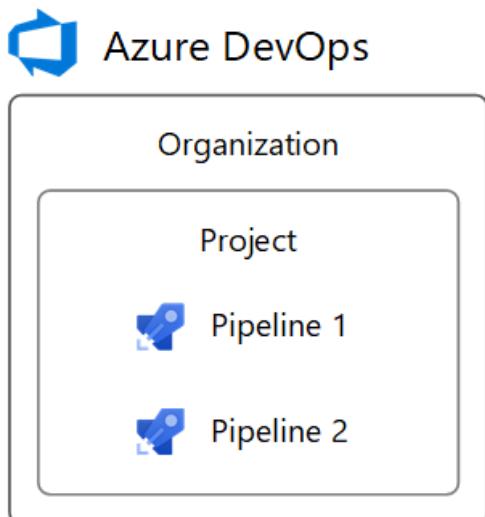
GitHub and Azure Pipelines are two independent services that integrate well together. Each of them have their own organization and user management. This section makes a recommendation on how to replicate the organization and users from GitHub to Azure Pipelines.

Organizations

GitHub's structure consists of **organizations and user accounts** that contain **repositories**. See [GitHub's documentation](#).

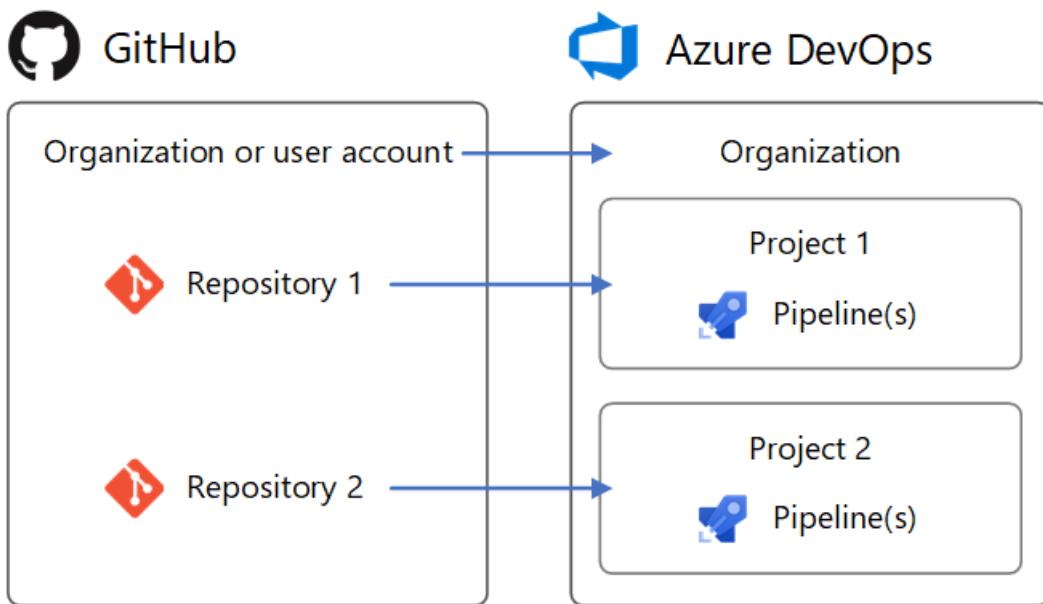


Azure DevOps' structure consists of **organizations** that contain **projects**. See [Plan your organizational structure](#).



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 an identical structure in 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 Project, create pipelines named after the GitHub organization and repository they build, such as [your-organization.your-repository](https://dev.azure.com/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:

SERVICE	URL
GitHub	https://github.com/python/cpython
Azure DevOps	https://dev.azure.com/python/cpython

Users

Your GitHub users do not automatically get access to Azure Pipelines. Azure Pipelines is unaware of GitHub identities. For this reason, there is no way to configure Azure Pipelines to automatically notify users of a build failure or a PR validation failure using their GitHub identity and email address. You must explicitly create new users in Azure Pipelines to replicate GitHub users. Once you create new users, you can configure their permissions in Azure DevOps to reflect their permissions in GitHub. You can also configure notifications in Azure DevOps using their Azure DevOps identity.

GitHub organization roles

GitHub organization member roles are found at [\(replace your-organization\)](https://github.com/orgs/your-organization/people).

Azure DevOps organization member permissions are found at

[\(replace your-organization\).](https://dev.azure.com/your-organization/_settings/security)

Roles in a GitHub organization and equivalent roles in an Azure DevOps organization are shown below.

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.

GitHub user account roles

A GitHub user account has one 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>

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`).

Equivalent permissions between GitHub repositories and Azure DevOps Projects are 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 Pipelines 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.

Access to GitHub repositories

- [YAML](#)
- [Classic](#)

You create a new pipeline by first selecting a GitHub repository and then a YAML file in that repository. The repository in which the YAML file is present is called **self** repository. By default, this is the repository that your pipeline builds.

You can later configure your pipeline to check out a different repository or multiple repositories. To learn how to do this, see [multi-repo checkout](#).

Azure Pipelines must be granted access to your repositories to 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	PIPELINES RUN USING	WORKS WITH GITHUB CHECKS
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

GitHub app authentication

The Azure Pipelines GitHub App is the **recommended** authentication type for continuous integration pipelines. By installing the GitHub App 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 app works with [GitHub Checks](#) to display build, test, and code coverage results in GitHub.

To use the GitHub App, install it in your GitHub organization or user account for some or all repositories. The GitHub App can be installed and uninstalled from the app's [homepage](#).

After installation, the GitHub App will become Azure Pipelines' default method of authentication to GitHub (instead of OAuth) when pipelines are created for the repositories.

If you install the GitHub 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, repository admins can install it one at a time for individual repositories. This requires more work for admins, but has no advantage nor disadvantage.

Permissions needed in GitHub

Installation of Azure Pipelines GitHub app requires you to be a GitHub organization owner or repository admin. In addition, to create a pipeline for a GitHub 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 appropriate access is configured.

- If the repo is in your personal GitHub account, install the Azure Pipelines GitHub App in your personal GitHub account. You will be able to list this repository when create the pipeline in Azure Pipelines.
- If the repo is in someone else's personal GitHub account, the other person must install the Azure Pipelines GitHub App in their personal GitHub account. 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. Once you have done so, you can create a pipeline for that repository.

- If the repo is in a GitHub organization that you own, install the Azure Pipelines GitHub App in the GitHub organization. You must also be added as a collaborator, or your team must be added, in the repository's settings under "Collaborators and teams".
- If the repo is in a GitHub organization that someone else owns, a GitHub organization owner or repository admin must install the Azure Pipelines GitHub App in the organization. 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.

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.

Troubleshooting GitHub App installation

GitHub may display an error such as:

You do not have permission to modify this app on your-organization. Please contact an Organization Owner.

This means that the GitHub App is likely already installed for your organization. When you create a pipeline for a repository in the organization, the GitHub App will automatically be used to connect to GitHub.

Create pipelines in multiple Azure DevOps organizations and projects

Once the GitHub 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.

OAuth authentication

[OAuth](#) is the simplest authentication type to get started with for repositories in your personal GitHub account. GitHub status updates will be performed on behalf of your personal GitHub identity. For pipelines to keep working, your repository access must remain active. Some GitHub features, like Checks, are unavailable with OAuth and require the GitHub App.

To use OAuth, click **Choose a different connection** below the list of repositories while creating a pipeline. Then, click **Authorize** to sign into GitHub and authorize with OAuth. An OAuth connection will be saved in your Azure DevOps project for later use, as well as used in the pipeline being created.

Permissions needed in GitHub

To create a pipeline for a GitHub 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 appropriate access is configured.

- If the repo is in your personal GitHub account, 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. Grant Azure Pipelines access to your repositories under "Permissions" [here](#).
- If the repo is in someone else's personal GitHub account, 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. The other person must grant Azure Pipelines access to their repositories under "Permissions" [here](#). 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.
- If the repo is in a GitHub organization that you own, 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. Grant Azure Pipelines access to your organization under "Organization access" [here](#). You must be added as a collaborator, or your team must be added, in the repository's settings under "Collaborators and teams".
- If the repo is in a GitHub organization that someone else owns, 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. The organization owner must grant Azure Pipelines access to the organization under "Organization access" [here](#). 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.

Revoke 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.

Personal access token (PAT) authentication

[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 active.

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 when creating pipelines.

Permissions needed in GitHub

To create a pipeline for a GitHub 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.

- If the repo is in your personal GitHub account, the PAT must have the required access scopes under [Personal access tokens](#): `repo`, `admin:repo_hook`, `read:user`, and `user:email`.

- If the repo is in someone else's personal GitHub account, the PAT must have the required access scopes under [Personal access tokens](#): `repo`, `admin:repo_hook`, `read:user`, and `user:email`. 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.
- If the repo is in a GitHub organization that you own, the PAT must have the required access scopes under [Personal access tokens](#): `repo`, `admin:repo_hook`, `read:user`, and `user:email`. You must be added as a collaborator, or your team must be added, in the repository's settings under "Collaborators and teams".
- If the repo is in a GitHub organization that someone else owns, the PAT must have the required access scopes under [Personal access tokens](#): `repo`, `admin:repo_hook`, `read:user`, and `user:email`. 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.

Revoke 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.

CI triggers

Continuous integration (CI) triggers cause a pipeline to run whenever you push an update to the specified branches or you push specified tags.

- [YAML](#)
- [Classic](#)

YAML pipelines are configured by default with a CI trigger on all branches.

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 wildcard (for example, `releases/*`). See [Wildcards](#) for information on the wildcard syntax.

NOTE

You cannot use [variables](#) in triggers, as variables are evaluated at runtime (after the trigger has fired).

NOTE

If you use [templates](#) to author YAML files, then you can only specify triggers in the main YAML file for the pipeline. You cannot specify triggers in the template files.

For more complex triggers that use `exclude` or `batch`, you must use the full syntax as shown in the following example.

```
# specific branch build
trigger:
  branches:
    include:
      - master
      - releases/*
    exclude:
      - releases/old*
```

In the above example, the pipeline will be triggered if a change is pushed to master or to any releases branch. However, it won't be triggered if a change is made to a releases branch that starts with `old`.

If you specify an `exclude` clause without an `include` clause, then it is equivalent to specifying `*` in the `include` clause.

In addition to specifying branch names in the `branches` lists, you can also configure triggers based on tags by using the following format:

```
trigger:
  branches:
    include:
      - refs/tags/{tagname}
    exclude:
      - refs/tags/{othertagname}
```

If you don't specify any triggers, the default is as if you wrote:

```
trigger:
  branches:
    include:
      - '*' # must quote since "*" is a YAML reserved character; we want a string
```

IMPORTANT

When you specify a trigger, it replaces the default implicit trigger, and only pushes to branches that are explicitly configured to be included will trigger a pipeline. Includes are processed first, and then excludes are removed from that list.

Batching CI runs

If you have many team members uploading changes often, you may want to reduce the number of runs you start. If you set `batch` to `true`, when a pipeline is running, the system waits until the run is completed, then starts another run with all changes that have not yet been built.

```
# specific branch build with batching
trigger:
  batch: true
  branches:
    include:
      - master
```

To clarify this example, let us say that a push `A` to master caused the above pipeline to run. While that pipeline is running, additional pushes `B` and `C` occur into the repository. These updates do not start new independent runs immediately. But after the first run is completed, all pushes until that point of time are batched together and a new run is started.

NOTE

If the pipeline has multiple jobs and stages, then the first run should still reach a terminal state by completing or skipping all its jobs and stages before the second run can start. For this reason, you must exercise caution when using this feature in a pipeline with multiple stages or approvals. If you wish to batch your builds in such cases, it is recommended that you split your CI/CD process into two pipelines - one for build (with batching) and one for deployments.

Paths

You can specify file paths to include or exclude.

```
# specific path build
trigger:
  branches:
    include:
      - master
      - releases/*
paths:
  include:
    - docs
  exclude:
    - docs/README.md
```

When you specify paths, you must explicitly specify branches to trigger on. You can't trigger a pipeline with only a path filter; you must also have a branch filter, and the changed files that match the path filter must be from a branch that matches the branch filter.

Tips:

- Wild cards are not supported with path filters.
- Paths are always specified relative to the root of the repository.
- If you don't set path filters, then the root folder of the repo is implicitly included by default.
- If you exclude a path, you cannot also include it unless you qualify it to a deeper folder. For example if you exclude `/tools` then you could include `/tools/trigger-runs-on-these`
- The order of path filters doesn't matter.
- Paths in Git *are case-sensitive*. Be sure to use the same case as the real folders.
- You cannot use [variables](#) in paths, as variables are evaluated at runtime (after the trigger has fired).

Tags

In addition to specifying tags in the `branches` lists as covered in the previous section, you can directly specify tags to include or exclude:

```
# specific tag
trigger:
  tags:
    include:
      - v2.*
    exclude:
      - v2.0
```

If you don't specify any tag triggers, then by default, tags will not trigger pipelines.

IMPORTANT

If you specify tags in combination with branch filters, the trigger will fire if either the branch filter is satisfied or the tag filter is satisfied. For example, if a pushed tag satisfies the branch filter, the pipeline triggers even if the tag is excluded by the tag filter, because the push satisfied the branch filter.

Opting out of CI

Disabling the CI trigger

You can opt out of CI triggers entirely by specifying `trigger: none`.

```
# A pipeline with no CI trigger
trigger: none
```

IMPORTANT

When you push a change to a branch, the YAML file in that branch is evaluated to determine if a CI run should be started.

For more information, see [Triggers](#) in the [YAML schema](#).

Skiping CI for individual commits

You can also tell Azure Pipelines to skip running a pipeline that a commit would normally trigger. Just include `[skip ci]` in the commit message or description of the HEAD commit and Azure Pipelines will skip running CI. You can also use any of the variations below.

- `[skip ci]` or `[ci skip]`
- `skip-checks: true` or `skip-checks:true`
- `[skip azurepipelines]` or `[azurepipelines skip]`
- `[skip azpipelines]` or `[azpipelines skip]`
- `[skip azp]` or `[azp skip]`
- `***NO_CI***`

Using the trigger type in conditions

It is a common scenario to run different steps, jobs, or stages in your pipeline depending on the type of trigger that started the run. You can do this using the system variable `Build.Reason`. For example, add the following condition to your step, job, or stage to exclude it from PR validations.

```
condition: and(succeeded(), ne(variables['Build.Reason'], 'PullRequest'))
```

Behavior of triggers when new branches are created

It is common to configure multiple pipelines for the same repository. For instance, you may have one pipeline to build the docs for your app and another to build the source code. You may configure CI triggers with appropriate branch filters and path filters in each of these pipelines. For instance, you may want one pipeline to trigger when you push an update to the `docs` folder, and another one to trigger when you push an update to your application code. In these cases, you need to understand how the pipelines are triggered when a new branch is created.

Here is the behavior when you push a new branch (that matches the branch filters) to your repository:

- If your pipeline has path filters, it will be triggered only if the new branch has changes to files that match that path filter.
- If your pipeline does not have path filters, it will be triggered even if there are no changes in the new branch.

Wildcards

When specifying a branch or tag, you may use an exact name or a wildcard. Wildcards patterns allow `*` to match zero or more characters and `?` to match a single character.

- If you start your pattern with `*` in a YAML pipeline, you must wrap the pattern in quotes, like `"*-releases"`.
- For branches and tags:
 - A wildcard may appear anywhere in the pattern.
- For paths:
 - You may include `*` as the final character, but it doesn't do anything differently from specifying the directory name by itself.
 - You may **not** include `*` in the middle of a path filter, and you may not use `?`.

```
trigger:
  branches:
    include:
      - master
      - releases/*
      - feature/*
    exclude:
      - releases/old*
      - feature/*-working
  paths:
    include:
      - '*' # same as '/' for the repository root
    exclude:
      - 'docs/*' # same as 'docs/'
```

PR triggers

Pull request (PR) triggers cause a pipeline to run whenever a pull request is opened with one of the specified target branches, or when updates are made to such a pull request.

- [YAML](#)
- [Classic](#)

Branches

You can specify the target branches when validating your pull requests. For example, to validate pull requests that target `master` and `releases/*`, you can use the following `pr` trigger.

```
pr:
  - master
  - releases/*
```

This configuration starts a new run the first time a new pull request is created, and after every update made to the pull request.

You can specify the full name of the branch (for example, `master`) or a wildcard (for example, `releases/*`).

NOTE

You cannot use [variables](#) in triggers, as variables are evaluated at runtime (after the trigger has fired).

NOTE

If you use [templates](#) to author YAML files, then you can only specify triggers in the main YAML file for the pipeline. You cannot specify triggers in the template files.

GitHub creates a new *ref* when a pull request is created. The ref points to a *merge commit*, which is the merged code between the source and target branches of the pull request. The PR validation pipeline builds the commit this ref points to. This means that the YAML file that is used to run the pipeline is also a merge between the source and the target branch. As a result, the changes you make to the YAML file in source branch of the pull request can override the behavior defined by the YAML file in target branch.

If no `pr` triggers appear in your YAML file, pull request validations are automatically enabled for all branches, as if you wrote the following `pr` trigger. This configuration triggers a build when any pull request is created, and when commits come into the source branch of any active pull request.

```
pr:  
  branches:  
    include:  
      - '*' # must quote since "*" is a YAML reserved character; we want a string
```

IMPORTANT

When you specify a `pr` trigger, it replaces the default implicit `pr` trigger, and only pushes to branches that are explicitly configured to be included will trigger a pipeline.

For more complex triggers that need to exclude certain branches, you must use the full syntax as shown in the following example.

```
# specific branch  
pr:  
  branches:  
    include:  
      - master  
      - releases/*  
    exclude:  
      - releases/old*
```

Paths

You can specify file paths to include or exclude. For example:

```
# specific path  
pr:  
  branches:  
    include:  
      - master  
      - releases/*  
  paths:  
    include:  
      - docs  
    exclude:  
      - docs/README.md
```

Tips:

- Wild cards are not supported with path filters.

- Paths are always specified relative to the root of the repository.
- If you don't set path filters, then the root folder of the repo is implicitly included by default.
- If you exclude a path, you cannot also include it unless you qualify it to a deeper folder. For example if you exclude `/tools` then you could include `/tools/trigger-runs-on-these`
- The order of path filters doesn't matter.
- Paths in Git *are case-sensitive*. Be sure to use the same case as the real folders.
- You cannot use [variables](#) in paths, as variables are evaluated at runtime (after the trigger has fired).

Multiple PR updates

You can specify whether additional updates to a PR should cancel in-progress validation runs for the same PR. The default is `true`.

```
# auto cancel false
pr:
  autoCancel: false
  branches:
    include:
      - master
```

Draft PR validation

By default, pull request triggers fire on draft pull requests as well as pull requests that are ready for review. To disable pull request triggers for draft pull requests, set the `drafts` property to `false`.

```
pr:
  autoCancel: boolean # indicates whether additional pushes to a PR should cancel in-progress runs for the
  same PR. Defaults to true
  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
  drafts: boolean # whether to build draft PRs, defaults to true
```

Opting out of PR validation

You can opt out of pull request validation entirely by specifying `pr: none`.

```
# no PR triggers
pr: none
```

For more information, see [PR trigger](#) in the [YAML schema](#).

NOTE

If your `pr` trigger isn't firing, follow the troubleshooting steps in the [FAQ](#).

NOTE

[Draft pull requests](#) do not trigger a pipeline.

Protected branches

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 mandatory validation builds for a GitHub repository, you must be its owner, a collaborator with the Admin role, or a GitHub organization member with the Write role.

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.

For the status check, select the name of your pipeline in the **Status checks** list.

Require status checks to pass before merging
Choose which [status checks](#) must pass before branches can be merged into a branch that matches this rule. When enabled, commits must first be pushed to another branch, then merged or pushed directly to a branch that matches this rule after status checks have passed.

Require branches to be up to date before merging
This ensures pull requests targeting a matching branch have been tested with the latest code. This setting will not take effect unless at least one status check is enabled (see below).

Status checks found in the last week for this repository

steved0x.pipelines-dotnet-core Required

IMPORTANT

If your pipeline doesn't show up in this list, please ensure the following:

- You are using [GitHub app authentication](#)
- Your pipeline has run at least once in the last week

Contributions from external sources

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.

You should keep in mind the following considerations when using Azure Pipelines in a public project when accepting contributions from external sources.

- [Access restrictions](#)
- [Validate contributions from forks](#)
- [Important security considerations](#)

Access restrictions

Be aware of the following access restrictions when you're running pipelines in Azure DevOps public projects:

- **Secrets:** By default, secrets associated with your pipeline are not made available to pull request validations of forks. See [Validate contributions from forks](#).
- **Cross-project access:** All pipelines in an Azure DevOps public project run with an access token restricted to the project. Pipelines in a public project can access resources such as build artifacts or test results only within the project and not in other projects of the Azure DevOps organization.
- **Azure Artifacts packages:** If your pipelines need access to packages from Azure Artifacts, you must explicitly grant permission to the **Project Build Service** account to access the package feeds.

Contributions from forks

IMPORTANT

These settings affect the security of your pipeline.

When you create a pipeline, it 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**, locate your 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 Server pipelines. Secrets include:

- A security token with access to your GitHub repository.
- These items, if your pipeline 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. Such code can cause harm in the following ways:

- Leak secrets from your pipeline. To mitigate 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. This option is disabled by default.
- Compromise the machine running the agent to steal code or secrets from other pipelines. To mitigate this:
 - 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.

Comment triggers

Repository collaborators can comment on a pull request to manually run a pipeline. Here are a few common reasons for why you might want to do this:

- You may not want to automatically build pull requests from unknown users until their changes can be reviewed. You want one of your team members to first review their code and then run the pipeline. This is commonly used as a security measure when building contributed code from forked repositories.
- You may want to run an optional test suite or an additional validation build.

To enable comment triggers you must follow the following two steps:

1. Enable pull request triggers for your pipeline, and make sure that you did not exclude the target branch.
2. In Azure Pipelines web interface, select the **Triggers** tab in your pipeline's settings. Then, under **Pull request validation**, enable **Only trigger builds for collaborators' pull request comments** and save the pipeline.

With these two changes, the pull request validation build will not be triggered automatically. Only repository owners and collaborators with 'Write' permission can trigger the build by commenting on the pull request with `/AzurePipelines run` or `/AzurePipelines run <pipeline-name>`.

The following commands can be issued to Azure Pipelines in comments:

COMMAND	RESULT
<code>/AzurePipelines help</code>	Display help for all supported commands.
<code>/AzurePipelines help <command-name></code>	Display help for the specified command.
<code>/AzurePipelines run</code>	Run all pipelines that are associated with this repository and whose triggers do not exclude this pull request.
<code>/AzurePipelines run <pipeline-name></code>	Run the specified pipeline unless its triggers exclude this pull request.

NOTE

For brevity, you can comment using `/azp` instead of `/AzurePipelines`.

IMPORTANT

Responses to these commands will appear in the pull request discussion only if your pipeline uses the [Azure Pipelines GitHub App](#).

Troubleshoot pull request comment triggers

If you have the necessary repository permissions, but pipelines aren't getting triggered by your comments, make sure that your membership is **public** in the repository's organization, or directly add yourself as a repository collaborator. Azure Pipelines cannot see private organization members unless they are direct collaborators or belong to a team that is a direct collaborator. You can change your GitHub organization membership from private to public here (replace `Your-Organization` with your organization name):

<https://github.com/orgs/Your-Organization/people>.

Checkout

When a pipeline is triggered, Azure Pipelines pulls your source code from the Azure Repos Git repository. You can control various aspects of how this happens.

Preferred version of Git

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.

Checkout path

- [YAML](#)
- [Classic](#)

If you are checking out a single repository, by default, your source code will be checked out into a directory called `s`. For YAML pipelines, you can change this by specifying `checkout` with a `path`. The specified path is relative to `$(Agent.BuildDirectory)`. For example: if the checkout path value is `mycustompath` and `$(Agent.BuildDirectory)` is `C:\agent_work\1`, then the source code will be checked out into `C:\agent_work\1\mycustompath`.

If you are using multiple `checkout` steps and checking out multiple repositories, and not explicitly specifying the folder using `path`, each repository is placed in a subfolder of `s` named after the repository. For example if you check out two repositories named `tools` and `code`, the source code will be checked out into `C:\agent_work\1\s\tools` and `C:\agent_work\1\s\code`.

Please note that the checkout path value cannot be set to go up any directory levels above `$(Agent.BuildDirectory)`, so `path..\anotherpath` will result in a valid checkout path (i.e. `C:\agent_work\1\anotherpath`), but a value like `..\invalidpath` will not (i.e. `C:\agent_work\invalidpath`).

You can configure the `path` setting in the [Checkout](#) step of your pipeline.

```
steps:  
- 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  
  path: string # path to check out source code, relative to the agent's build directory (e.g. \_work\1)  
  persistCredentials: boolean # set to 'true' to leave the OAuth token in the Git config after the initial  
    fetch
```

Submodules

- [YAML](#)
- [Classic](#)

You can configure the `submodules` setting in the [Checkout](#) step of your pipeline if you want to download files from [submodules](#).

```
steps:  
- 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  
  path: string # path to check out source code, relative to the agent's build directory (e.g. \_work\1)  
  persistCredentials: boolean # set to 'true' to leave the OAuth token in the Git config after the initial  
    fetch
```

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 as the Azure Repos Git repo specified above. 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.
 - Added by using a URL relative to the main repository. For example

- This one would be checked out:

```
git submodule add ../../../../FabrikamFiberProject/_git/FabrikamFiber FabrikamFiber
```

In this example the submodule refers to a repo (FabrikamFiber) in the same Azure DevOps organization, but in a different project (FabrikamFiberProject). 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. This requires that the job access token has access to the repository in the second project. If you restricted the job access token as explained in the section above, then you won't be able to do this. You can allow the job access token to access the repo in the second project by either (a) explicitly granting access to the project build service account in the second project or (b) using collection-scoped access tokens instead of project-scoped tokens for the entire organization. For more information about these options and their security implications, see [Access repositories, artifacts, and other resources](#).

- This one would not be checked out:

```
git submodule add https://fabrikam-fiber@dev.azure.com/fabrikam-fiber/FabrikamFiberProject/_git/FabrikamFiber FabrikamFiber
```

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 if your job access token does not have access to the repository in a different project.

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 prefixed 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 <BASE64_ENCODED_STRING>"  
submodule update --init --recursive
```

Be sure to replace "`<BASE64_ENCODED_STRING>`" with your Base64-encoded "pat:token" string.

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.

Shallow fetch

You may 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.

- [YAML](#)
- [Classic](#)

You can configure the `fetchDepth` setting in the [Checkout](#) step of your pipeline.

```
steps:  
- 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  
path: string # path to check out source code, relative to the agent's build directory (e.g. \_work\1)  
persistCredentials: boolean # set to 'true' to leave the OAuth token in the Git config after the initial  
fetch
```

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 pipeline is started, 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.

Don't sync sources

You may 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.
- [YAML](#)
- [Classic](#)

You can configure the **Don't sync sources** setting in the [Checkout](#) step of your pipeline, by setting

```
checkout: none .
```

```
steps:  
- checkout: none # Don't sync sources
```

NOTE

When you use this option, the agent also skips running Git commands that clean the repo.

Clean build

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.

- [YAML](#)
- [Classic](#)

You can configure the `clean` setting in the [Checkout](#) step of your pipeline.

```
steps:  
- 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  
  path: string # path to check out source code, relative to the agent's build directory (e.g. \_work\1)  
  persistCredentials: boolean # set to 'true' to leave the OAuth token in the Git config after the initial  
  fetch
```

When `clean` is set to `true` 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 -ffdx  
git reset --hard HEAD
```

For more options, you can configure the `workspace` setting of a [Job](#).

```
jobs:  
- job: string # name of the job, A-Z, a-z, 0-9, and underscore  
  ...  
  workspace:  
    clean: outputs | resources | all # what to clean up before the job runs
```

This gives the following clean options.

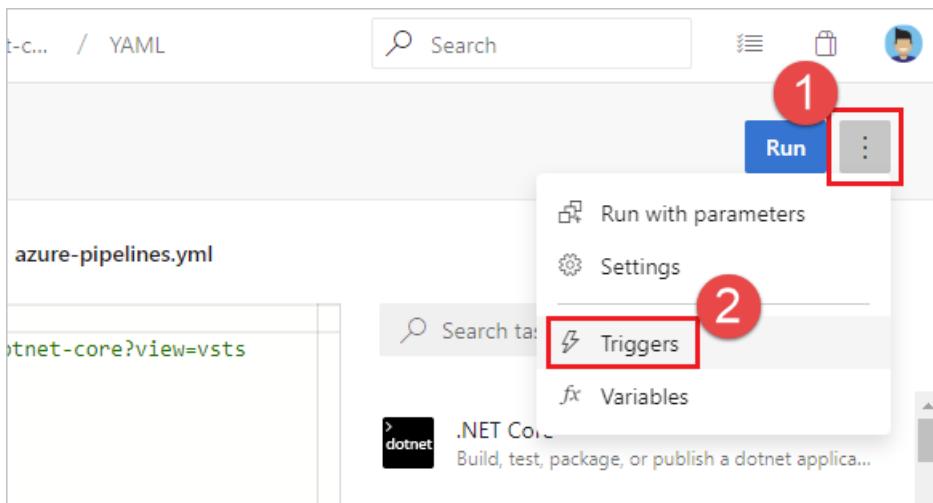
- **outputs:** Same operation as the clean setting described in the previous the checkout task, plus: Deletes and recreates `$(Build.BinariesDirectory)`. Note that the `$(Build.ArtifactStagingDirectory)` and `$(Common.TestResultsDirectory)` are always deleted and recreated prior to every build regardless of any of these settings.
- **resources:** Deletes and recreates `$(Build.SourcesDirectory)`. This results in initializing a new, local Git repository for every build.
- **all:** Deletes and recreates `$(Agent.BuildDirectory)`. This results in initializing a new, local Git repository for every build.

Label sources

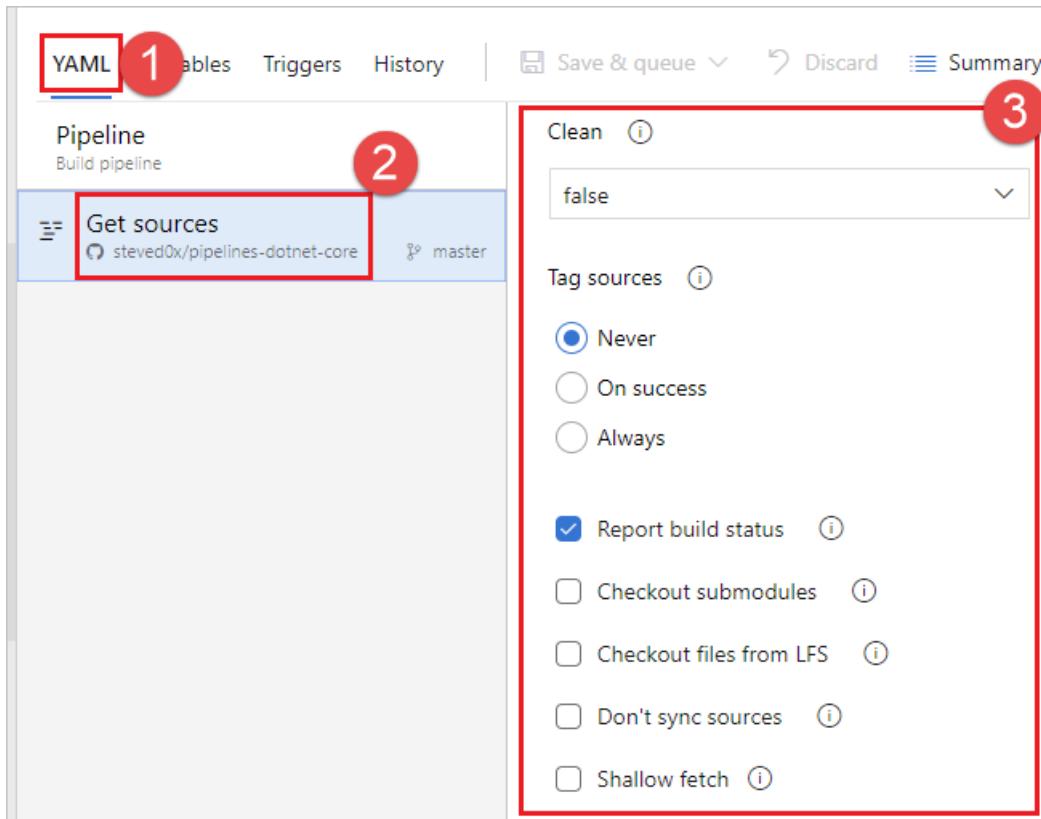
You may want to label your source code files to enable your team to easily identify which version of each file is included in the completed build. You also have the option to specify whether the source code should be labeled for all builds or only for successful builds.

- [YAML](#)
- [Classic](#)

You can't currently configure this setting in YAML but you can in the classic editor. When editing a YAML pipeline, you can access the classic editor by choosing either **Triggers** from the YAML editor menu.



From the classic editor, choose **YAML**, choose the **Get sources** task, and then configure the desired properties there.



In the **Tag 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. The tag is considered a build artifact since it is produced by the build. When the build is deleted either manually or through a retention policy, the tag is also deleted.

Pre-defined variables

When you build a GitHub repository, most of the [pre-defined variables](#) are available to your jobs. However, since Azure Pipelines does not recognize the identity of a user making an update in GitHub, the following variables are set to system identity instead of user's identity:

- `Build.RequestedFor`
- `Build.RequestedForId`
- `Build.RequestedForEmail`

Status updates

There are two types of statuses that Azure Pipelines posts back to GitHub - basic statuses and GitHub Check Runs. GitHub Checks functionality is only available with GitHub Apps.

Pipeline statuses show up in various places in the GitHub UI.

- For PRs, they are displayed on the PR conversations tab.
- For individual commits, they are displayed when hovering over the status mark after the commit time on the repo's commits tab.

PAT or OAuth GitHub connections

For pipelines using [PAT](#) or [OAuth](#) GitHub connections, statuses are posted back to the commit/PR that triggered the run. The [GitHub status API](#) is used to post such updates. These statuses contain limited information: pipeline status (failed, success), URL to link back to the build pipeline, and a brief description of the status.

Statuses for PAT or OAuth GitHub connections are only sent at the run level. In other words, you can have a single status updated for an entire run. If you have multiple jobs in a run, you cannot post a separate status for each job. However, multiple pipelines can post separate statuses to the same commit.

GitHub Checks

For pipelines set up using the Azure Pipelines [GitHub app](#), the status is posted back in the form of GitHub Checks. GitHub Checks allow for sending detailed information about the pipeline status as well as test, code coverage, and errors. The GitHub Checks API can be found [here](#).

For every pipeline using the GitHub App, Checks are posted back for the overall run as well as each job in that run.

GitHub allows three options when one or more Check Runs fail for a PR/commit. You can choose to "re-run" the individual Check, re-run all the failing Checks on that PR/commit, or re-run all the Checks, whether they succeeded initially or not.

Update README.md #32

Edit

The screenshot shows a GitHub pull request for "Update README.md". The "Checks" tab is selected, displaying an Azure Pipelines check run for the commit b8dc359. The run is labeled "Azure Pipelines / keljos.deep-space" and is failing 26 minutes ago in 1m 4s. It lists three failing checks: "keljos.deep-space (firsttestj...)" (Re-run), "keljos.deep-space (secondte...)" (Re-run), and "keljos.deep-space (thirdtestj...)" (Re-run). In the top right corner of the check run details, there are two yellow-highlighted buttons: "Re-run failing checks" and "Re-run all checks".

Clicking on the "Re-run" link next to the Check Run name will result in Azure Pipelines retrying the run that generated the Check Run. The resultant run will have the same run number and will use the same version of the source code, configuration, and YAML file as the initial build. Only those jobs that failed in the initial run and any dependent downstream jobs will be run again. Clicking on the "Re-run all failing checks" link will have the same effect. This is the same behavior as clicking "Re-try run" in the Azure Pipelines UI. Clicking on "Re-run all checks" will result in a new run, with a new run number and will pick up changes in the configuration or YAML file.

FAQ

Problems related to GitHub integration fall into the following categories:

- **Connection types:** I am not sure what connection type I am using to connect my pipeline to GitHub.
- **Failing triggers:** My pipeline is not being triggered when I push an update to the repo.
- **Failing checkout:** My pipeline is being triggered, but it fails in the checkout step.
- **Wrong version:** My pipeline runs, but it is using an unexpected version of the source/YAML.
- **Missing status updates:** My GitHub PRs are blocked because Azure Pipelines did not report a status update.

Connection types

To troubleshoot triggers, how do I know the type of GitHub connection I'm using for my pipeline?

Troubleshooting problems with triggers very much depends on the type of GitHub connection you use in your pipeline. There are two ways to determine the type of connection - from GitHub and from Azure Pipelines.

- From GitHub: If a repo is set up to use the GitHub app, then the statuses on PRs and commits will be Check Runs. If the repo has Azure Pipelines set up with OAuth or PAT connections, the statuses will be the "old" style of statuses. A quick way to determine if the statuses are Check Runs or simple statuses is to look at the "conversation" tab on a GitHub PR.
 - If the "Details" link redirects to the Checks tab, it is a Check Run and the repo is using the app.
 - If the "Details" link redirects to the Azure DevOps pipeline, then the status is an "old style" status and the repo is not using the app.
- From Azure Pipelines: You can also determine the type of connection by inspecting the pipeline in Azure Pipelines UI. Open the editor for the pipeline. Select **Triggers** to open the classic editor for the pipeline. Then, select **YAML** tab and then the **Get sources** step. You'll notice a banner **Authorized using connection:** indicating the service connection that was used to integrate the pipeline with GitHub. The name of the service connection is a hyperlink. Select it to navigate to the service connection properties. The properties of the service connection will indicate the type of connection being used:

- **Azure Pipelines app** indicates GitHub app connection
- **oauth** indicates OAuth connection
- **personalaccesstoken** indicates PAT authentication

How do I switch my pipeline to use GitHub app instead of OAuth?

Using a GitHub app instead of OAuth or PAT connection is the recommended integration between GitHub and Azure Pipelines. To switch to GitHub app, follow these steps:

1. Navigate [here](#) and install the app in the GitHub organization of your repository.
2. During installation, you'll be redirected to Azure DevOps to choose an Azure DevOps organization and project. Choose the organization and project that contain the classic build pipeline you want to use the app for. This choice associates the GitHub App installation with your Azure DevOps organization. If you choose incorrectly, you can visit [this page](#) to uninstall the GitHub app from your GitHub org and start over.
3. In the next page that appears, you do not need to proceed creating a new pipeline.
4. Edit your pipeline by visiting the Pipelines page (e.g., https://dev.azure.com/YOUR_ORG_NAME/YOUR_PROJECT_NAME/_build), selecting your pipeline, and clicking Edit.
5. If this is a YAML pipeline, select the Triggers menu to open the classic editor.
6. Select the "Get sources" step in the pipeline.
7. On the green bar with text "Authorized using connection", click "Change" and select the GitHub App connection with the same name as the GitHub organization in which you installed the app.
8. On the toolbar, select "Save and queue" and then "Save and queue". Click the link to the pipeline run that was queued to make sure it succeeds.
9. Create (or close and reopen) a pull request in your GitHub repository to verify that a build is successfully queued in its "Checks" section.

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.

- If you're using the GitHub App, see [GitHub App authentication](#).
- If you're using OAuth, see [OAuth authentication](#).
- If you're using PATs, see [Personal access token \(PAT\) authentication](#).

When I select a repository during pipeline creation, I get an error "The repository {repo-name} is in use with the Azure Pipelines GitHub App in another Azure DevOps organization."

This means that your repository is already associated with a pipeline in a different organization. CI and PR events from this repository won't work as they will be delivered to the other organization. Here are the steps you should take to remove the mapping to the other organization before proceeding to create a pipeline.

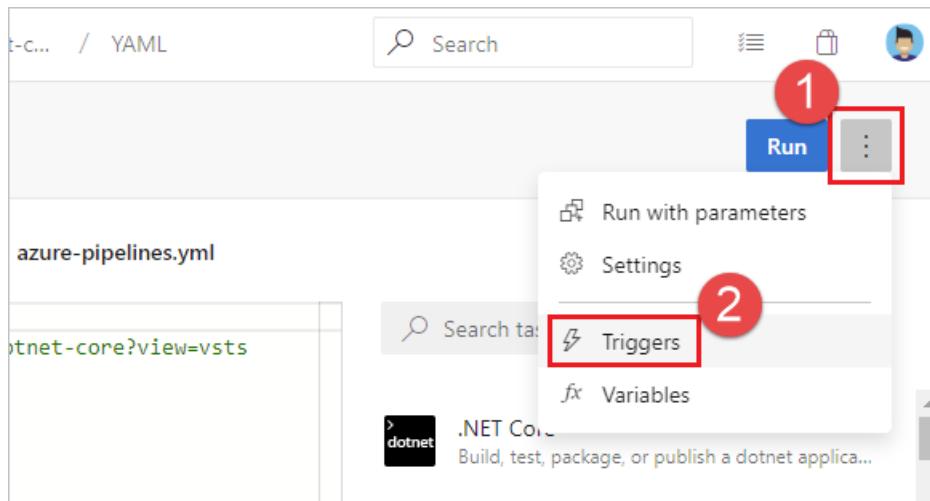
1. Open a pull request in your GitHub repository, and make the comment `/azp where`. This reports back the Azure DevOps organization that the repository is mapped to.
2. To change the mapping, uninstall the app from the GitHub organization, and re-install it. As you re-install it, make sure to select the correct organization when you are redirected to Azure DevOps.

Failing triggers

I just created a new YAML pipeline with CI/PR triggers, but the pipeline is not being triggered.

Follow each of these steps to troubleshoot your failing triggers:

- Are your YAML CI or PR triggers being [overridden by pipeline settings in the UI](#)? While editing your pipeline, choose ... and then Triggers.



Check the **Override the YAML trigger from here** setting for the types of trigger (**Continuous integration** or **Pull request validation**) available for your repo.

The screenshot shows the 'Triggers' tab in the pipeline settings. It has two sections: 'Continuous integration' and 'Pull request validation'. Under 'Continuous integration', there is one entry for 'steved0x/pipelines-dotnet-core' which is enabled. Under 'Pull request validation', there is also one entry for 'steved0x/pipelines-dotnet-core' which is enabled. On the right side, there is a section for the repository 'steved0x/pipelines-dotnet-core' with a checkbox labeled 'Override the YAML pull request trigger from here' which is unchecked. Below it are 'Forks' settings: 'Build pull requests from forks of this repository' (checked) and 'Important security considerations'.

- Are you using the GitHub app connection to connect the pipeline to GitHub? See [Connection types](#) to determine the type of connection you have. If you are using a GitHub app connection, follow these steps:
 - Is the mapping set up properly between GitHub and Azure DevOps? Open a pull request in your GitHub repository, and make the comment `/azp where`. This reports back the Azure DevOps organization that the repository is mapped to.
 - If no organizations are set up to build this repository using the app, go to https://github.com/<org_name>/<repo_name>/settings/installations and complete the configuration of the app.
 - If a different Azure DevOps organization is reported, then someone has already established a pipeline for this repo in a different organization. We currently have the limitation that we can only map a GitHub repo to a single DevOps org. Only the pipelines in the first Azure DevOps org can be automatically triggered. To change the mapping, uninstall the app from the GitHub organization, and re-install it. As you re-install it, make sure to select the correct organization when you are redirected to Azure DevOps.
- Are you using OAuth or PAT to connect the pipeline to GitHub? See [Connection types](#) to determine the type of connection you have. If you are using a GitHub connection, follow these steps:
 - OAuth and PAT connections rely on webhooks to communicate updates to Azure Pipelines. In GitHub, navigate to the settings for your repository, then to Webhooks. Verify that the webhooks exist. Usually you should see three webhooks - push, pull_request, and issue_comment. If you don't, then you must re-create the service connection and update the pipeline to use the new service connection.
 - Select each of the webhooks in GitHub and verify that the payload that corresponds to the user's

commit exists and was sent successfully to Azure DevOps. You may see an error here if the event could not be communicated to Azure DevOps.

- The traffic from Azure DevOps could be throttled by GitHub. When Azure Pipelines receives a notification from GitHub, it tries to contact GitHub and fetch more information about the repo and YAML file. If you have a repo with a large number of updates and pull requests, this call may fail due to such throttling. In this case, see if you can reduce the frequency of builds by using batching or stricter path/branch filters.
- Is your pipeline paused or disabled? Open the editor for the pipeline, and then select **Settings** to check. If your pipeline is paused or disabled, then triggers do not work.
- Have you updated the YAML file in the correct branch? If you push an update to a branch, then the YAML file in that same branch governs the CI behavior. If you push an update to a source branch, then the YAML file resulting from merging the source branch with the target branch governs the PR behavior. Make sure that the YAML file in the correct branch has the necessary CI or PR configuration.
- Have you configured the trigger correctly? When you define a YAML trigger, you can specify both include and exclude clauses for branches, tags, and paths. Ensure that the include clause matches the details of your commit and that the exclude clause doesn't exclude them. Check the syntax for the triggers and make sure that it is accurate.
- Have you used variables in defining the trigger or the paths? That is not supported.
- Did you use templates for your YAML file? If so, make sure that your triggers are defined in the main YAML file. Triggers defined inside template files are not supported.
- Have you excluded the branches or paths to which you pushed your changes? Test by pushing a change to an included path in an included branch. Note that paths in triggers are case-sensitive. Make sure that you use the same case as those of real folders when specifying the paths in triggers.
- Do you have wildcards in your path filters? Understand the limitations of wildcards in your paths as described in this article.
- Did you just push a new branch? If so, the new branch may not start a new run. See the section "Behavior of triggers when new branches are created".

My CI or PR triggers have been working fine. But, they stopped working now.

First go through the troubleshooting steps in the previous question. Then, follow these additional steps:

- Do you have merge conflicts in your PR? For a PR that did not trigger a pipeline, open it and check whether it has a merge conflict. Resolve the merge conflict.
- Are you experiencing a delay in the processing of push or PR events? You can usually verify this by seeing if the issue is specific to a single pipeline or is common to all pipelines or repos in your project. If a push or a PR update to any of the repos exhibits this symptom, we might be experiencing delays in processing the update events. Check if we are experiencing a service outage on our [status page](#). If the status page shows an issue, then our team must have already started working on it. Check the page frequently for updates on the issue.

I do not want users to override the list of branches for triggers when they update the YAML file. How can I do this?

Users with permissions to contribute code can update the YAML file and include/exclude additional branches. As a result, users can include their own feature or user branch in their YAML file and push that update to a feature or user branch. This may cause the pipeline to be triggered for all updates to that branch. If you want to prevent this behavior, then you can:

1. Edit the pipeline in the Azure Pipelines UI.
2. Navigate to the **Triggers** menu.
3. Select **Override the YAML continuous Integration trigger from here**.

4. Specify the branches to include or exclude for the trigger.

When you follow these steps, any CI triggers specified in the YAML file are ignored.

Failing checkout

I see the following error in the log file during checkout step. How do I fix it?

```
remote: Repository not found.  
fatal: repository <repo> not found
```

This could be caused by an outage of GitHub. Try to access the repository in GitHub and make sure that you are able to.

Wrong version

A wrong version of the YAML file is being used in the pipeline. Why is that?

- For CI triggers, the YAML file that is in the branch you are pushing is evaluated to see if a CI build should be run.
- For PR triggers, the YAML file resulting from merging the source and target branches of the PR is evaluated to see if a PR build should be run.

Missing status updates

My PR in GitHub is blocked since Azure Pipelines did not update the status.

This could be a transient error that resulted in Azure DevOps not being able to communicate with GitHub. Retry the check in GitHub if you use the GitHub app. Or, make a trivial update to the PR to see if the problem can be resolved.

Related articles

- [Scheduled triggers](#)
- [Pipeline completion triggers](#)

Build GitHub Enterprise Server repositories

5/6/2021 • 10 minutes to read • [Edit Online](#)

You can integrate your on-premises GitHub Enterprise Server with Azure Pipelines. Your on-premises server may be exposed to the Internet or it may not be.

If your GitHub Enterprise Server is reachable from the servers that run Azure Pipelines service, then:

- you can set up classic build and YAML pipelines
- you can configure CI, PR, and scheduled triggers

If your GitHub Enterprise Server is not reachable from the servers that run Azure Pipelines service, then:

- you can only set up classic build pipelines
- you can only start manual or scheduled builds
- you cannot set up YAML pipelines
- you cannot configure CI or PR triggers for your classic build pipelines

If your on-premises server is reachable from Microsoft-hosted agents, then you can use them to run your pipelines. Otherwise, you must set up self-hosted agents that can access your on-premises server and fetch the code.

Reachable from Azure Pipelines

The first thing to check is whether your GitHub Enterprise Server is reachable from Azure Pipelines service.

1. In your Azure DevOps UI, navigate to your project settings, and select **Service Connections** under **Pipelines**.
2. Select **New service connection** and choose **GitHub Enterprise Server** as the connection type.
3. Enter the required information to create a connection to your GitHub Enterprise Server.
4. Select **Verify** in the service connection panel.

If the verification passes, then the servers that run Azure Pipelines service are able to reach your on-premises GitHub Enterprise Server. You can proceed and set up the connection. Then, you can use this service connection when creating a classic build or YAML pipeline. You can also configure CI and PR triggers for the pipeline. A majority of features in Azure Pipelines that work with GitHub also work with GitHub Enterprise Server. Review the documentation for [GitHub](#) to understand these features. Here are some differences:

- The integration between GitHub and Azure Pipelines is made easier through an Azure Pipelines app in GitHub marketplace. This app allows you to set up an integration without having to rely on a particular user's OAuth token. We do not have a similar app that works with GitHub Enterprise Server. So, you must use a PAT, username and password, or OAuth to set up the connection between Azure Pipelines and GitHub Enterprise server.
- When working with GitHub, Azure Pipelines supports a number of security features to validate contributions from external forks. For instance, secrets stored in a pipeline are not made available to a running job. These protections are not available when working with GitHub Enterprise server.
- Comment triggers are not available with GitHub Enterprise server. You cannot use comments in a GitHub Enterprise server repo pull request to trigger a pipeline.
- GitHub Checks are not available in GitHub Enterprise server. All status updates are through basic statuses.

Not reachable from Azure Pipelines

When the verification of a GitHub Enterprise Server connection as explained in the above section fails, then Azure Pipelines cannot communicate with your server. This is likely caused by how your enterprise network is set up. For instance, a firewall in your network may prevent external traffic from reaching your servers. You have two options in this case:

- Work with your IT department to open a network path between Azure Pipelines and GitHub Enterprise Server. For example, you can add exceptions to your firewall rules to allow traffic from Azure Pipelines to flow through. See the section on [Azure DevOps IPs](#) to see which IP addresses you need to allow. Furthermore, you need to have a public DNS entry for the GitHub Enterprise Server so that Azure Pipelines can resolve the FQDN of your server to an IP address. With all of these changes, attempt to create and verify a GitHub Enterprise Server connection in Azure Pipelines.
- Instead of using a GitHub Enterprise Server connection, you can use a [Other Git](#) connection. Make sure to uncheck the option to **Attempt accessing this Git server from Azure Pipelines**. With this connection type, you can only configure a classic build pipeline. CI and PR triggers will not work in this configuration. You can only start manual or scheduled pipeline runs.

Reachable from Microsoft-hosted agents

Another decision you possibly have to make is whether to use Microsoft-hosted agents or self-hosted agents to run your pipelines. This often comes down to whether Microsoft-hosted agents can reach your server. To check whether they can, create a simple pipeline to use Microsoft-hosted agents and make sure to add a step to checkout source code from your server. If this passes, then you can continue using Microsoft-hosted agents.

Not reachable from Microsoft-hosted agents

If the simple test pipeline mentioned in the above section fails with the error

TF401019: The Git repository with name or identifier <your repo name> does not exist or you do not have permissions for the operation you are attempting

, then the GitHub Enterprise Server is not reachable from Microsoft-hosted agents. This is again probably caused by a firewall blocking traffic from these servers. You have two options in this case:

- Work with your IT department to open a network path between Microsoft-hosted agents and GitHub Enterprise Server. See the section on [networking](#) in Microsoft-hosted agents.
- Switch to using [self-hosted agents](#) or [scale-set agents](#). These agents can be set up within your network and hence will have access to the GitHub Enterprise Server. These agents only require outbound connections to Azure Pipelines. There is no need to open a firewall for inbound connections. Make sure that the name of the server you specified when creating the GitHub Enterprise Server connection is resolvable from the self-hosted agents.

Azure DevOps IP addresses

Azure Pipelines sends requests to GitHub Enterprise Server to:

- Query for a list of repositories during pipeline creation (classic and YAML pipelines)
- Look for existing YAML files during pipeline creation (YAML pipelines)
- Check-in YAML files (YAML pipelines)
- Register a webhook during pipeline creation (classic and YAML pipelines)
- Present an editor for YAML files (YAML pipelines)
- Resolve templates and expand YAML files prior to execution (YAML pipelines)
- Check if there are any changes since the last scheduled run (classic and YAML pipelines)

- Fetch details about latest commit and display that in the user interface (classic and YAML pipelines)

You can observe that YAML pipelines fundamentally require communication between Azure Pipelines and GitHub Enterprise Server. Hence, it is not possible to set up a YAML pipeline if the GitHub Enterprise Server is not visible to Azure Pipelines.

When you use **Other Git** connection to set up a classic pipeline, disable communication between Azure Pipelines service and GitHub Enterprise Server, and use self-hosted agents to build code, you will get a degraded experience:

- You will have to type in the name of the repository manually during pipeline creation
- You cannot use CI or PR triggers as Azure Pipelines cannot register a webhook in GitHub Enterprise Server
- You cannot use scheduled triggers with the option to build only when there are changes
- You cannot view information about the latest commit in the user interface

If you want to set up YAML pipelines or if you want to enhance the experience with classic pipelines, it is important that you enable communication from Azure Pipelines to GitHub Enterprise Server.

To allow traffic from Azure DevOps to reach your GitHub Enterprise Server, add the IP addresses or service tags specified in [Inbound connections](#) to your firewall's allow-list. If you use ExpressRoute, make sure to also include [ExpressRoute IP ranges](#) to your firewall's allow-list.

FAQ

Problems related to GitHub Enterprise integration fall into the following categories:

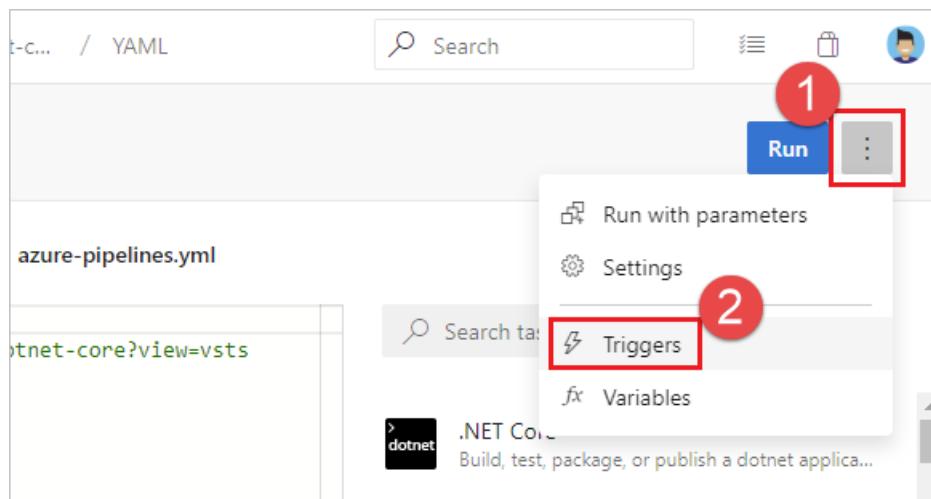
- Failing triggers:** My pipeline is not being triggered when I push an update to the repo.
- Failing checkout:** My pipeline is being triggered, but it fails in the checkout step.
- Wrong version:** My pipeline runs, but it is using an unexpected version of the source/YAML.

Failing triggers

I just created a new YAML pipeline with CI/PR triggers, but the pipeline is not being triggered.

Follow each of these steps to troubleshoot your failing triggers:

- Are your YAML CI or PR triggers being [overridden by pipeline settings in the UI?](#) While editing your pipeline, choose ... and then Triggers.



Check the **Override the YAML trigger from here** setting for the types of trigger (**Continuous integration** or **Pull request validation**) available for your repo.

The screenshot shows the Azure Pipelines YAML editor interface. At the top, there are tabs for 'YAML', 'Variables', 'Triggers', 'History', and 'Save & queue'. Below the tabs, there's a 'Continuous integration' section with a 'steved0x/pipelines-dotnet-core' trigger, which is enabled. In the 'Pull request validation' section, another 'steved0x/pipelines-dotnet-core' trigger is also enabled. On the right side, under 'steved0x/pipelines-dotnet-core', there's a red-bordered box containing a checkbox labeled 'Override the YAML pull request trigger from here'. Below this, there's a 'Forks' section with a checked checkbox for 'Build pull requests from forks of this repository' and a link for 'Important security considerations'.

- Webhooks are used to communicate updates from GitHub Enterprise to Azure Pipelines. In GitHub Enterprise, navigate to the settings for your repository, then to Webhooks. Verify that the webhooks exist. Usually you should see two webhooks - push, pull_request. If you don't, then you must re-create the service connection and update the pipeline to use the new service connection.
- Select each of the webhooks in GitHub Enterprise and verify that the payload that corresponds to the user's commit exists and was sent successfully to Azure DevOps. You may see an error here if the event could not be communicated to Azure DevOps.
- When Azure Pipelines receives a notification from GitHub, it tries to contact GitHub and fetch more information about the repo and YAML file. If the GitHub Enterprise Server is behind a firewall, this traffic may not reach your server. See [Azure DevOps IP Addresses](#) and verify that you have granted exceptions to all the required IP addresses. These IP addresses may have changed since you have originally set up the exception rules.
- Is your pipeline paused or disabled? Open the editor for the pipeline, and then select **Settings** to check. If your pipeline is paused or disabled, then triggers do not work.
- Have you updated the YAML file in the correct branch? If you push an update to a branch, then the YAML file in that same branch governs the CI behavior. If you push an update to a source branch, then the YAML file resulting from merging the source branch with the target branch governs the PR behavior. Make sure that the YAML file in the correct branch has the necessary CI or PR configuration.
- Have you configured the trigger correctly? When you define a YAML trigger, you can specify both include and exclude clauses for branches, tags, and paths. Ensure that the include clause matches the details of your commit and that the exclude clause doesn't exclude them. Check the syntax for the triggers and make sure that it is accurate.
- Have you used variables in defining the trigger or the paths? That is not supported.
- Did you use templates for your YAML file? If so, make sure that your triggers are defined in the main YAML file. Triggers defined inside template files are not supported.
- Have you excluded the branches or paths to which you pushed your changes? Test by pushing a change to an included path in an included branch. Note that paths in triggers are case-sensitive. Make sure that you use the same case as those of real folders when specifying the paths in triggers.
- Do you have wildcards in your path filters? Understand the limitations of wildcards in your paths as described in this article.
- Did you just push a new branch? If so, the new branch may not start a new run. See the section "Behavior of triggers when new branches are created".

My CI or PR triggers have been working fine. But, they stopped working now.

First go through the troubleshooting steps in the previous question. Then, follow these additional steps:

- Do you have merge conflicts in your PR? For a PR that did not trigger a pipeline, open it and check

whether it has a merge conflict. Resolve the merge conflict.

- Are you experiencing a delay in the processing of push or PR events? You can usually verify this by seeing if the issue is specific to a single pipeline or is common to all pipelines or repos in your project. If a push or a PR update to any of the repos exhibits this symptom, we might be experiencing delays in processing the update events. Check if we are experiencing a service outage on our [status page](#). If the status page shows an issue, then our team must have already started working on it. Check the page frequently for updates on the issue.

Failing checkout

Do you use Microsoft-hosted agents? If so, these agents may not be able to reach your GitHub Enterprise Server. See [Not reachable from Microsoft-hosted agents](#) for more information.

Wrong version

A wrong version of the YAML file is being used in the pipeline. Why is that?

- For CI triggers, the YAML file that is in the branch you are pushing is evaluated to see if a CI build should be run.
- For PR triggers, the YAML file resulting from merging the source and target branches of the PR is evaluated to see if a PR build should be run.

Pipeline options for Git repositories

3/6/2021 • 17 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

While editing a pipeline that uses a Git repo—in an Azure DevOps project, GitHub, GitHub Enterprise Server, Bitbucket Cloud, or another Git repo—you have the following options.

FEATURE	AZURE PIPELINES	TFS 2017.2 AND HIGHER	TFS 2017 RTM	TFS 2015.4	TFS 2015 RTM
Branch	Yes	Yes	Yes	Yes	Yes
Clean	Yes	Yes	Yes	Yes	Yes
Tag or label sources	Project; Classic only	Team project	Team project	Team project	No
Report build status	Yes	Yes	Yes	No	No
Checkout submodules	Yes	Yes	Yes	Yes	Yes
Checkout files from LFS	Yes	Yes	Linux and macOS agents	Linux and macOS agents	Linux and macOS agents
Clone a second repo	Yes	Yes	Yes	Yes	Yes
Don't sync sources	Yes	Yes	No	No	No
Shallow fetch	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. When using self-hosted agents, depending on how your agents pools are configured, you may get a new agent for subsequent pipeline runs (or stages or jobs in the same pipeline), so **not** cleaning is not a guarantee that subsequent runs, jobs, or stages will be able to access outputs from previous runs, jobs, or stages.

NOTE

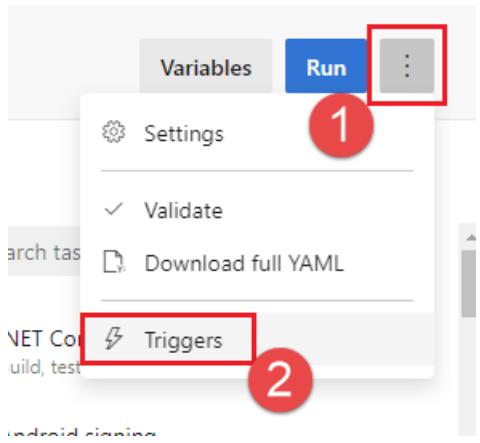
When using self-hosted agents, depending on how your agents pools are configured, you may get a new agent for subsequent pipeline runs (or stages or jobs in the same pipeline), so **not** cleaning is not a guarantee that subsequent runs, jobs, or stages will be able to access outputs from previous runs, jobs, or stages. You can use [Build artifacts](#) to share outputs of a pipeline run, stage, or job with subsequent runs, stages, or jobs.

- [YAML](#)
- [Classic](#)

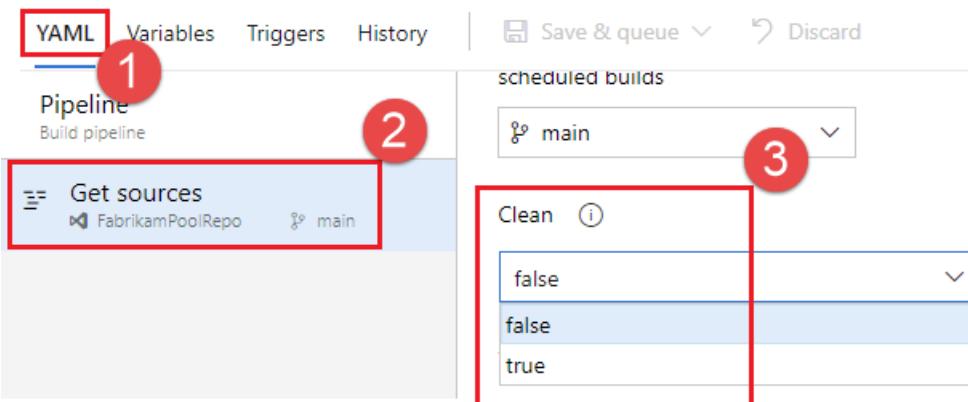
Azure Pipelines, Azure DevOps Server 2019 and newer

There are several different clean options available for YAML pipelines.

- The `checkout` step has a `clean` option. When set to `true`, the pipeline runs `execute git clean -ffdx && git reset --hard HEAD` before fetching the repo. For more information, see [Checkout](#).
- The `workspace` setting for `job` has multiple clean options (outputs, resources, all). For more information, see [Workspace](#).
- The pipeline settings UI has a **Clean** setting, that when set to true is equivalent of specifying `clean: true` for every `checkout` step in your pipeline. To configure the **Clean** setting:
 1. Edit your pipeline, choose ..., and select **Triggers**.



2. Select **YAML**, **Get sources**, and configure your desired **Clean** setting. The default is **false**.



To override clean settings when manually running a pipeline, you can use [runtime parameters](#). In the following example, a runtime parameter is used to configure the checkout clean setting.

```

parameters:
- name: clean
  displayName: Checkout clean
  type: boolean
  default: false
  values:
  - false
  - true

trigger:
- main

pool: FabrikamPool
# vmImage: 'ubuntu-latest'

steps:
- checkout: self
  clean: ${{ parameters.clean }}

```

By default, `clean` is set to `false` but can be overridden when manually running the pipeline by checking the **Checkout clean** checkbox that is added for the runtime parameter.

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 GitHub repository, or 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. The tag is considered a build artifact since it is produced by the build. When the build is deleted either manually or through a retention policy, the tag is also deleted.

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 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 path

If you are checking out a single repository, by default, your source code will be checked out into a directory called `s`. For YAML pipelines, you can change this by specifying `checkout` with a `path`. The specified path is relative to `$(Agent.BuildDirectory)`. For example: if the checkout path value is `mycustompath` and `$(Agent.BuildDirectory)` is `C:\agent_work\1`, then the source code will be checked out into `C:\agent_work\1\mycustompath`.

If you are using multiple `checkout` steps and checking out multiple repositories, and not explicitly specifying the folder using `path`, each repository is placed in a subfolder of `s` named after the repository. For example if you check out two repositories named `tools` and `code`, the source code will be checked out into `C:\agent_work\1\s\tools` and `C:\agent_work\1\s\code`.

Please note that the checkout path value cannot be set to go up any directory levels above `$(Agent.BuildDirectory)`, so `path..\anotherpath` will result in a valid checkout path (i.e. `C:\agent_work\1\anotherpath`), but a value like `..\invalidpath` will not (i.e. `C:\agent_work\invalidpath`).

NOTE

The checkout path can only be specified for YAML pipelines. For more information, see [Checkout](#) in the [YAML schema](#).

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. If you want to use LFS with submodules, be sure to see the [note about using LFS with submodules](#).

NOTE

For more information about the YAML syntax for checking out submodules, see [Checkout in the YAML schema](#).

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 Cloud 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 prefixed 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<BASE64_ENCODED_TOKEN_DESCRIBED_ABOVE>" 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 \(LFS\)](#).

In the classic editor, select the check box to enable this option.

In a YAML build, add a checkout step with `lfs` set to `true`:

```
steps:  
- checkout: self  
  lfs: true
```

- **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. If your hosted agents use Windows, consider using the `System.PreferGitFromPath` variable to ensure that pipelines use the versions of git and git-lfs you installed on the machine.

Using Git LFS with submodules

If a submodule contains LFS files, Git LFS must be configured prior to checking out submodules. The Microsoft-hosted macOS and Linux agents come preconfigured this way. Windows agents and self-hosted macOS / Linux agents may not.

As a workaround, if you're using YAML, you can add the following step before your `checkout`:

```
steps:
- script: |
  git config --global --add filter.lfs.required true
  git config --global --add filter.lfs.smudge "git-lfs smudge -- %f"
  git config --global --add filter.lfs.process "git-lfs filter-process"
  git config --global --add filter.lfs.clean "git-lfs clean -- %f"
  displayName: Configure LFS for use with submodules
- checkout: self
  lfs: true
  submodules: true
# ... rest of steps ...
```

Clone a second repo

By default, your pipeline is associated with one repo from Azure Repos or an external provider. This is the repo that can trigger builds on commits and pull requests.

You may want to include sources from a second repo in your pipeline. You can do this by writing a script.

```
git clone https://github.com/Microsoft/TypeScript.git
```

If the repo is not public, you will need to pass authentication to the Git command.

Azure Repos

Your pipeline will already have access to other repos in its project, and you can clone them in your pipeline using a script command, as shown in the following example.

```
- script: |
  git clone -c http.extraheader="AUTHORIZATION: bearer $(System.AccessToken)"
  https://organization@dev.azure.com/project/FabrikamFiber/_git/reponame
```

You can clone multiple repositories in the same project as your pipeline by using [multi-repo checkout](#).

If you need to clone a repo from another project that is not public, you will need to authenticate as a user who has access to that project.

NOTE

Use a [secret variable](#) to store credentials securely.

Secret variables are not automatically made available to scripts as environment variables. See [Secret variables](#) on how to map them in.

For Azure Repos, you can use a personal access token with the **Code (Read)** permission. Send this as the password field in a "Basic" authorization header without a username. (In other words, base64-encode the value of `:<PAT>`, including the colon.)

```
AUTH=$(echo -n "$REPO_PAT" | openssl base64 | tr -d '\n')
git -c http.<repo URL>.extraheader="AUTHORIZATION: basic $AUTH" clone <repo URL> --no-checkout --branch
master
```

Don't sync sources (TFS 2017 and newer only)

Non-deployment jobs automatically fetch sources. Use this option if you want to skip that behavior. 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 Other Git

When using an Other/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.

FAQ

What protocols can the build agent use with Git?

The agent supports HTTPS.

The agent does not yet support SSH. See [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](#).

Build Bitbucket Cloud repositories

3/26/2021 • 17 minutes to read • [Edit Online](#)

Azure Pipelines

Azure Pipelines can automatically build and validate every pull request and commit to your Bitbucket Cloud repository. This article describes how to configure the integration between Bitbucket Cloud and Azure Pipelines.

Bitbucket and Azure Pipelines are two independent services that integrate well together. Your Bitbucket Cloud users do not automatically get access to Azure Pipelines. You must add them explicitly to Azure Pipelines.

Access to Bitbucket repositories

- [YAML](#)
- [Classic](#)

You create a new pipeline by first selecting a Bitbucket Cloud repository and then a YAML file in that repository. The repository in which the YAML file is present is called `self` repository. By default, this is the repository that your pipeline builds.

You can later configure your pipeline to check out a different repository or multiple repositories. To learn how to do this, see [multi-repo checkout](#).

Azure Pipelines must be granted access to your repositories to fetch the code during builds. In addition, the user setting up the pipeline must have admin access to Bitbucket, since that identity is used to register a webhook in Bitbucket.

There are 2 authentication types for granting Azure Pipelines access to your Bitbucket Cloud repositories while creating a pipeline.

AUTHENTICATION TYPE	PIPELINES RUN USING
1. OAuth	Your personal Bitbucket identity
2. Username and password	Your personal Bitbucket identity

OAuth authentication

OAuth is the simplest authentication type to get started with for repositories in your Bitbucket account. Bitbucket status updates will be performed on behalf of your personal Bitbucket identity. For pipelines to keep working, your repository access must remain active.

To use OAuth, login to Bitbucket when prompted during pipeline creation. Then, click **Authorize** to authorize with OAuth. An OAuth connection will be saved in your Azure DevOps project for later use, as well as used in the pipeline being created.

NOTE

The maximum number of BitBucket repositories that the Azure DevOps Services user interface can load is 2,000.

Password authentication

Builds and Bitbucket status updates will be performed on behalf of your personal identity. For builds to keep

working, your repository access must remain active.

To create a password connection, visit [Service connections](#) in your Azure DevOps project settings. Create a new Bitbucket service connection and provide the user name and password to connect to your Bitbucket Cloud repository.

CI triggers

Continuous integration (CI) triggers cause a pipeline to run whenever you push an update to the specified branches or you push specified tags.

- [YAML](#)
- [Classic](#)

YAML pipelines are configured by default with a CI trigger on all branches.

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 wildcard (for example, `releases/*`). See [Wildcards](#) for information on the wildcard syntax.

NOTE

You cannot use [variables](#) in triggers, as variables are evaluated at runtime (after the trigger has fired).

NOTE

If you use [templates](#) to author YAML files, then you can only specify triggers in the main YAML file for the pipeline. You cannot specify triggers in the template files.

For more complex triggers that use `exclude` or `batch`, you must use the full syntax as shown in the following example.

```
# specific branch build  
trigger:  
  branches:  
    include:  
    - master  
    - releases/*  
    exclude:  
    - releases/old*
```

In the above example, the pipeline will be triggered if a change is pushed to master or to any releases branch. However, it won't be triggered if a change is made to a releases branch that starts with `old`.

If you specify an `exclude` clause without an `include` clause, then it is equivalent to specifying `*` in the `include` clause.

In addition to specifying branch names in the `branches` lists, you can also configure triggers based on tags by using the following format:

```
trigger:  
  branches:  
    include:  
      - refs/tags/{tagname}  
    exclude:  
      - refs/tags/{othertagname}
```

If you don't specify any triggers, the default is as if you wrote:

```
trigger:  
  branches:  
    include:  
      - '*' # must quote since "*" is a YAML reserved character; we want a string
```

IMPORTANT

When you specify a trigger, it replaces the default implicit trigger, and only pushes to branches that are explicitly configured to be included will trigger a pipeline. Includes are processed first, and then excludes are removed from that list.

Batching CI runs

If you have many team members uploading changes often, you may want to reduce the number of runs you start. If you set `batch` to `true`, when a pipeline is running, the system waits until the run is completed, then starts another run with all changes that have not yet been built.

```
# specific branch build with batching  
trigger:  
  batch: true  
  branches:  
    include:  
      - master
```

To clarify this example, let us say that a push A to master caused the above pipeline to run. While that pipeline is running, additional pushes B and C occur into the repository. These updates do not start new independent runs immediately. But after the first run is completed, all pushes until that point of time are batched together and a new run is started.

NOTE

If the pipeline has multiple jobs and stages, then the first run should still reach a terminal state by completing or skipping all its jobs and stages before the second run can start. For this reason, you must exercise caution when using this feature in a pipeline with multiple stages or approvals. If you wish to batch your builds in such cases, it is recommended that you split your CI/CD process into two pipelines - one for build (with batching) and one for deployments.

Paths

You can specify file paths to include or exclude.

```
# specific path build
trigger:
  branches:
    include:
      - master
      - releases/*
  paths:
    include:
      - docs
    exclude:
      - docs/README.md
```

When you specify paths, you must explicitly specify branches to trigger on. You can't trigger a pipeline with only a path filter; you must also have a branch filter, and the changed files that match the path filter must be from a branch that matches the branch filter.

Tips:

- Wild cards are not supported with path filters.
- Paths are always specified relative to the root of the repository.
- If you don't set path filters, then the root folder of the repo is implicitly included by default.
- If you exclude a path, you cannot also include it unless you qualify it to a deeper folder. For example if you exclude `/tools` then you could include `/tools/trigger-runs-on-these`
- The order of path filters doesn't matter.
- Paths in Git *are case-sensitive*. Be sure to use the same case as the real folders.
- You cannot use [variables](#) in paths, as variables are evaluated at runtime (after the trigger has fired).

NOTE

For Bitbucket Cloud repos, using `branches` syntax is the only way to specify tag triggers. The `tags:` syntax is not supported for Bitbucket.

Opting out of CI

Disabling the CI trigger

You can opt out of CI triggers entirely by specifying `trigger: none`.

```
# A pipeline with no CI trigger
trigger: none
```

IMPORTANT

When you push a change to a branch, the YAML file in that branch is evaluated to determine if a CI run should be started.

For more information, see [Triggers](#) in the [YAML schema](#).

Skipping CI for individual commits

You can also tell Azure Pipelines to skip running a pipeline that a commit would normally trigger. Just include `[skip ci]` in the commit message or description of the HEAD commit and Azure Pipelines will skip running CI.

You can also use any of the variations below.

- `[skip ci]` or `[ci skip]`

- `skip-checks: true` or `skip-checks:true`
- `[skip azurepipelines]` or `[azurepipelines skip]`
- `[skip azpipelines]` or `[azpipelines skip]`
- `[skip azp]` or `[azp skip]`
- `***NO_CI***`

Using the trigger type in conditions

It is a common scenario to run different steps, jobs, or stages in your pipeline depending on the type of trigger that started the run. You can do this using the system variable `Build.Reason`. For example, add the following condition to your step, job, or stage to exclude it from PR validations.

```
condition: and(succeeded(), ne(variables['Build.Reason'], 'PullRequest'))
```

Behavior of triggers when new branches are created

It is common to configure multiple pipelines for the same repository. For instance, you may have one pipeline to build the docs for your app and another to build the source code. You may configure CI triggers with appropriate branch filters and path filters in each of these pipelines. For instance, you may want one pipeline to trigger when you push an update to the `docs` folder, and another one to trigger when you push an update to your application code. In these cases, you need to understand how the pipelines are triggered when a new branch is created.

Here is the behavior when you push a new branch (that matches the branch filters) to your repository:

- If your pipeline has path filters, it will be triggered only if the new branch has changes to files that match that path filter.
- If your pipeline does not have path filters, it will be triggered even if there are no changes in the new branch.

Wildcards

When specifying a branch or tag, you may use an exact name or a wildcard. Wildcards patterns allow `*` to match zero or more characters and `?` to match a single character.

- If you start your pattern with `*` in a YAML pipeline, you must wrap the pattern in quotes, like `"*-releases"`.
- For branches and tags:
 - A wildcard may appear anywhere in the pattern.
- For paths:
 - You may include `*` as the final character, but it doesn't do anything differently from specifying the directory name by itself.
 - You may **not** include `*` in the middle of a path filter, and you may not use `?`.

```
trigger:
  branches:
    include:
      - master
      - releases/*
      - feature/*
    exclude:
      - releases/old*
      - feature/*-working
  paths:
    include:
      - '*' # same as '/' for the repository root
    exclude:
      - 'docs/*' # same as 'docs/'
```

PR triggers

Pull request (PR) triggers cause a pipeline to run whenever a pull request is opened with one of the specified target branches, or when updates are made to such a pull request.

- [YAML](#)
- [Classic](#)

Branches

You can specify the target branches when validating your pull requests. For example, to validate pull requests that target `master` and `releases/*`, you can use the following `pr` trigger.

```
pr:  
- master  
- releases/*
```

This configuration starts a new run the first time a new pull request is created, and after every update made to the pull request.

You can specify the full name of the branch (for example, `master`) or a wildcard (for example, `releases/*`).

NOTE

You cannot use [variables](#) in triggers, as variables are evaluated at runtime (after the trigger has fired).

NOTE

If you use [templates](#) to author YAML files, then you can only specify triggers in the main YAML file for the pipeline. You cannot specify triggers in the template files.

Each new run builds the latest commit from the source branch of the pull request. This is different from how Azure Pipelines builds pull requests in other repositories (e.g., Azure Repos or GitHub), where it builds the merge commit. Unfortunately, Bitbucket does not expose information about the merge commit, which contains the merged code between the source and target branches of the pull request.

If no `pr` triggers appear in your YAML file, pull request validations are automatically enabled for all branches, as if you wrote the following `pr` trigger. This configuration triggers a build when any pull request is created, and when commits come into the source branch of any active pull request.

```
pr:  
branches:  
  include:  
    - '*' # must quote since "*" is a YAML reserved character; we want a string
```

IMPORTANT

When you specify a `pr` trigger, it replaces the default implicit `pr` trigger, and only pushes to branches that are explicitly configured to be included will trigger a pipeline.

For more complex triggers that need to exclude certain branches, you must use the full syntax as shown in the following example.

```
# specific branch
pr:
  branches:
    include:
      - master
      - releases/*
    exclude:
      - releases/old*
```

Paths

You can specify file paths to include or exclude. For example:

```
# specific path
pr:
  branches:
    include:
      - master
      - releases/*
  paths:
    include:
      - docs
    exclude:
      - docs/README.md
```

Tips:

- Wild cards are not supported with path filters.
- Paths are always specified relative to the root of the repository.
- If you don't set path filters, then the root folder of the repo is implicitly included by default.
- If you exclude a path, you cannot also include it unless you qualify it to a deeper folder. For example if you exclude `/tools` then you could include `/tools/trigger-runs-on-these`
- The order of path filters doesn't matter.
- Paths in Git *are case-sensitive*. Be sure to use the same case as the real folders.
- You cannot use [variables](#) in paths, as variables are evaluated at runtime (after the trigger has fired).

Multiple PR updates

You can specify whether additional updates to a PR should cancel in-progress validation runs for the same PR. The default is `true`.

```
# auto cancel false
pr:
  autoCancel: false
  branches:
    include:
      - master
```

Opting out of PR validation

You can opt out of pull request validation entirely by specifying `pr: none`.

```
# no PR triggers
pr: none
```

For more information, see [PR trigger](#) in the [YAML schema](#).

NOTE

If your `pr` trigger isn't firing, ensure that you have not [overridden YAML PR triggers in the UI](#).

FAQ

Problems related to Bitbucket integration fall into the following categories:

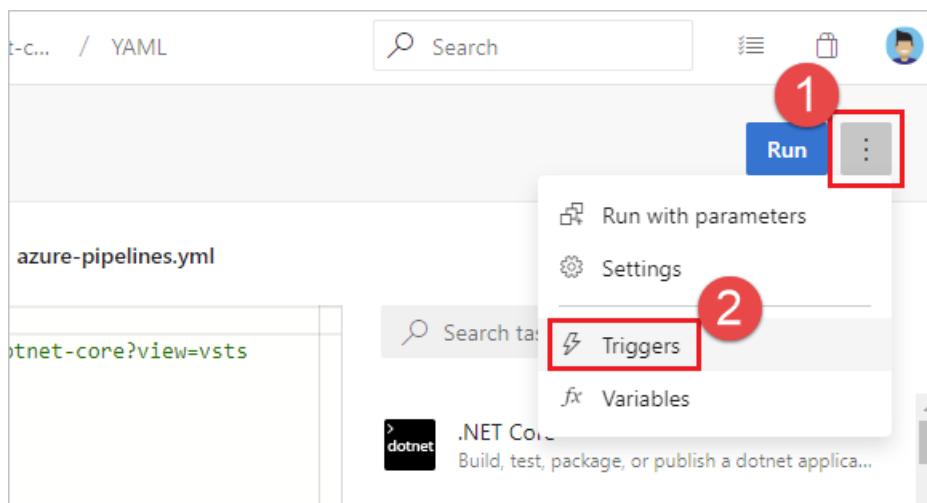
- **Failing triggers:** My pipeline is not being triggered when I push an update to the repo.
- **Wrong version:** My pipeline runs, but it is using an unexpected version of the source/YAML.

Failing triggers

I just created a new YAML pipeline with CI/PR triggers, but the pipeline is not being triggered.

Follow each of these steps to troubleshoot your failing triggers:

- Are your YAML CI or PR triggers being [overridden by pipeline settings in the UI](#)? While editing your pipeline, choose ... and then Triggers.



Check the **Override the YAML trigger from here** setting for the types of trigger (**Continuous integration** or **Pull request validation**) available for your repo.

A screenshot of the 'Triggers' tab in the Azure Pipelines editor. The tab bar includes 'YAML', 'Variables', 'Triggers' (which is selected and underlined), 'History', 'Save & queue', 'Discard', 'Summary', and 'Qu...'. The main area is divided into two sections: 'Continuous integration' and 'Pull request validation'. Under 'Continuous integration', there's a single entry for 'steved0x/pipelines-dotnet-core' with the status 'Enabled'. Under 'Pull request validation', there's also a single entry for 'steved0x/pipelines-dotnet-core' with the status 'Enabled'. To the right, there's a sidebar for the repository 'steved0x/pipelines-dotnet-core'. It shows a checkbox for 'Override the YAML pull request trigger from here' which is unchecked. Below that is a 'Forks' section with a checked checkbox for 'Build pull requests from forks of this repository' and a link 'Important security considerations'.

- Webhooks are used to communicate updates from Bitbucket to Azure Pipelines. In Bitbucket, navigate to the settings for your repository, then to Webhooks. Verify that the webhooks exist.
- Is your pipeline paused or disabled? Open the editor for the pipeline, and then select **Settings** to check. If your pipeline is paused or disabled, then triggers do not work.
- Have you updated the YAML file in the correct branch? If you push an update to a branch, then the YAML file in that same branch governs the CI behavior. If you push an update to a source branch, then the YAML file resulting from merging the source branch with the target branch governs the PR behavior. Make sure

that the YAML file in the correct branch has the necessary CI or PR configuration.

- Have you configured the trigger correctly? When you define a YAML trigger, you can specify both include and exclude clauses for branches, tags, and paths. Ensure that the include clause matches the details of your commit and that the exclude clause doesn't exclude them. Check the syntax for the triggers and make sure that it is accurate.
- Have you used variables in defining the trigger or the paths? That is not supported.
- Did you use templates for your YAML file? If so, make sure that your triggers are defined in the main YAML file. Triggers defined inside template files are not supported.
- Have you excluded the branches or paths to which you pushed your changes? Test by pushing a change to an included path in an included branch. Note that paths in triggers are case-sensitive. Make sure that you use the same case as those of real folders when specifying the paths in triggers.
- Do you have wildcards in your path filters? Understand the limitations of wildcards in your paths as described in this article.
- Did you just push a new branch? If so, the new branch may not start a new run. See the section "Behavior of triggers when new branches are created".

My CI or PR triggers have been working fine. But, they stopped working now.

First go through the troubleshooting steps in the previous question. Then, follow these additional steps:

- Do you have merge conflicts in your PR? For a PR that did not trigger a pipeline, open it and check whether it has a merge conflict. Resolve the merge conflict.
- Are you experiencing a delay in the processing of push or PR events? You can usually verify this by seeing if the issue is specific to a single pipeline or is common to all pipelines or repos in your project. If a push or a PR update to any of the repos exhibits this symptom, we might be experiencing delays in processing the update events. Check if we are experiencing a service outage on our [status page](#). If the status page shows an issue, then our team must have already started working on it. Check the page frequently for updates on the issue.

I do not want users to override the list of branches for triggers when they update the YAML file. How can I do this?

Users with permissions to contribute code can update the YAML file and include/exclude additional branches. As a result, users can include their own feature or user branch in their YAML file and push that update to a feature or user branch. This may cause the pipeline to be triggered for all updates to that branch. If you want to prevent this behavior, then you can:

1. Edit the pipeline in the Azure Pipelines UI.
2. Navigate to the **Triggers** menu.
3. Select **Override the YAML continuous Integration trigger from here**.
4. Specify the branches to include or exclude for the trigger.

When you follow these steps, any CI triggers specified in the YAML file are ignored.

Wrong version

A wrong version of the YAML file is being used in the pipeline. Why is that?

- For CI triggers, the YAML file that is in the branch you are pushing is evaluated to see if a CI build should be run.
- For PR triggers, the YAML file resulting from merging the source and target branches of the PR is evaluated to see if a PR build should be run.

Build on-premises Bitbucket repositories

3/6/2021 • 6 minutes to read • [Edit Online](#)

NOTE

To integrate Bitbucket Cloud with Azure Pipelines, see [Bitbucket Cloud](#).

You can integrate your on-premises Bitbucket server or another Git server with Azure Pipelines. Your on-premises server may be exposed to the Internet or it may not be.

If your on-premises server is reachable from the servers that run Azure Pipelines service, then:

- you can set up classic build and configure CI triggers

If your on-premises server is not reachable from the servers that run Azure Pipelines service, then:

- you can set up classic build pipelines and start manual builds
- you cannot configure CI triggers

NOTE

YAML pipelines do not work with on-premises Bitbucket repositories.

NOTE

PR triggers are not available with on-premises Bitbucket repositories.

If your on-premises server is reachable from the hosted agents, then you can use the hosted agents to run manual, scheduled, or CI builds. Otherwise, you must set up self-hosted agents that can access your on-premises server and fetch the code.

Reachable from Azure Pipelines

If your on-premises Bitbucket server is reachable from Azure Pipelines service, create a [Other Git](#) service connection and use that to create a pipeline. Check the option to **Attempt accessing this Git server from Azure Pipelines**.

CI triggers work through polling and not through webhooks. In other words, Azure Pipelines periodically checks the Bitbucket server if there are any updates to code. If there are, then Azure Pipelines will start a new run.

Not reachable from Azure Pipelines

If the Bitbucket server cannot be reached from Azure Pipelines, you have two options:

- Work with your IT department to open a network path between Azure Pipelines and on-premises Git server. For example, you can add exceptions to your firewall rules to allow traffic from Azure Pipelines to flow through. See the section on [Azure DevOps IPs](#) to see which IP addresses you need to allow. Furthermore, you need to have a public DNS entry for the Bitbucket server so that Azure Pipelines can resolve the FQDN of your server to an IP address.
- You can use a [Other Git](#) connection but tell Azure Pipelines not to **attempt accessing this Git server**

from Azure Pipelines. CI and PR triggers will not work in this configuration. You can only start manual or scheduled pipeline runs.

Reachable from Microsoft-hosted agents

Another decision you possibly have to make is whether to use Microsoft-hosted agents or self-hosted agents to run your pipelines. This often comes down to whether Microsoft-hosted agents can reach your server. To check whether they can, create a simple pipeline to use Microsoft-hosted agents and make sure to add a step to checkout source code from your server. If this passes, then you can continue using Microsoft-hosted agents.

Not reachable from Microsoft-hosted agents

If the simple test pipeline mentioned in the above section fails with the error

TF401019: The Git repository with name or identifier <your repo name> does not exist or you do not have permissions for the operation you are attempting

, then the Bitbucket server is not reachable from Microsoft-hosted agents. This is again probably caused by a firewall blocking traffic from these servers. You have two options in this case:

- Work with your IT department to open a network path between Microsoft-hosted agents and Bitbucket server. See the section on [networking](#) in Microsoft-hosted agents.
- Switch to using [self-hosted agents](#) or [scale-set agents](#). These agents can be set up within your network and hence will have access to the Bitbucket server. These agents only require outbound connections to Azure Pipelines. There is no need to open a firewall for inbound connections. Make sure that the name of the server you specified when creating the service connection is resolvable from the self-hosted agents.

Azure DevOps IP addresses

When you use [Other Git](#) connection to set up a classic pipeline, disable communication between Azure Pipelines service and Bitbucket server, and use self-hosted agents to build code, you will get a degraded experience:

- You will have to type in the name of the repository manually during pipeline creation
- You cannot use CI triggers as Azure Pipelines won't be able to poll for changes to the code
- You cannot use scheduled triggers with the option to build only when there are changes
- You cannot view information about the latest commit in the user interface

If you want to enhance this experience, it is important that you enable communication from Azure Pipelines to Bitbucket Server.

To allow traffic from Azure DevOps to reach your Bitbucket Server, add the IP addresses or service tags specified in [Inbound connections](#) to your firewall's allow-list. If you use ExpressRoute, make sure to also include [ExpressRoute IP ranges](#) to your firewall's allow-list.

Allow Azure Pipelines to attempt accessing the Git server in the [Other Git](#) service connection.

FAQ

Problems related to Bitbucket Server integration fall into the following categories:

- [Failing triggers](#): My pipeline is not being triggered when I push an update to the repo.
- [Failing checkout](#): My pipeline is being triggered, but it fails in the checkout step.

Failing triggers

I pushed a change to my server, but the pipeline is not being triggered.

Follow each of these steps to troubleshoot your failing triggers:

- Is your Bitbucket server accessible from Azure Pipelines? Azure Pipelines periodically polls Bitbucket server for changes. If the Bitbucket server is behind a firewall, this traffic may not reach your server. See

[Azure DevOps IP Addresses](#) and verify that you have granted exceptions to all the required IP addresses.

These IP addresses may have changed since you have originally set up the exception rules. You can only start manual runs if you used an external Git connection and if your server is not accessible from Azure Pipelines.

- Is your pipeline paused or disabled? Open the editor for the pipeline, and then select **Settings** to check. If your pipeline is paused or disabled, then triggers do not work.
- Have you excluded the branches or paths to which you pushed your changes? Test by pushing a change to an included path in an included branch. Note that paths in triggers are case-sensitive. Make sure that you use the same case as those of real folders when specifying the paths in triggers.

I did not push any updates to my code, however the pipeline is still being triggered.

- The continuous integration trigger for Bitbucket works through polling. After each polling interval, Azure Pipelines attempts to contact the Bitbucket server to check if there have been any updates to the code. If Azure Pipelines is unable to reach the Bitbucket server (possibly due to a network issue), then we start a new run anyway assuming that there might have been code changes. In a few cases, Azure Pipelines may also create a dummy failed build with an error message to indicate that it was unable to reach the server.

Failing checkout

When I attempt to start a new run manually, there is a delay of 4-8 minutes before it starts.

- Your Bitbucket server is not reachable from Azure Pipelines. Make sure that you have not selected the option **to attempt accessing this Git server from Azure Pipelines** in the Bitbucket service connection. If that option is selected, Azure Pipelines will attempt to contact to your server and since your server is unreachable, it eventually times out and starts the run anyway. Unchecking that option speeds up your manual runs.

The checkout step fails with the error that the server cannot be resolved.

Do you use Microsoft-hosted agents? If so, these agents may not be able to reach your Bitbucket server. See [Not reachable from Microsoft-hosted agents](#) for more information.

Build TFVC repositories

11/2/2020 • 6 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Choose the repository to build

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 GitHub repository, or 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](#).

CI triggers

Select **Enable continuous integration** on the Triggers tab to enable this trigger if you want the build to run whenever someone checks in code.

The screenshot shows the 'Triggers' tab of a build pipeline configuration. The 'Continuous integration' section contains a single trigger named 'FabrikamTFVC' which is currently 'Enabled'. In the 'Path filters' table, there is one entry: 'Type' set to 'Include' and 'Path specification' set to '\$/FabrikamTFVC'. Other sections like 'Gated check-in' and 'Scheduled' are listed but have their respective checkboxes unchecked.

Batch changes

Select this check box if you have many team members uploading changes often and you want to reduce the number of builds you are running. If you select this option, when a build is running, the system waits until the build is completed and then queues another build of all changes that have not yet been built.

You can batch changes and build them together.

Path filters

Select the version control paths you want to include and exclude. In most cases, you should make sure that these filters are consistent with your TFVC mappings. You can use path filters to reduce the set of files that you want to trigger a build.

Tips:

- Paths are always specified relative to the root of the workspace.
- If you don't set path filters, then the root folder of the workspace is implicitly included by default.
- If you exclude a path, you cannot also include it unless you qualify it to a deeper folder. For example if you exclude `/tools` then you could include `/tools/trigger-runs-on-these`
- The order of path filters doesn't matter.

Gated check-in

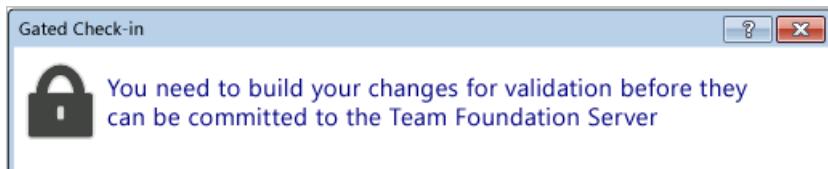
You can use gated check-in to protect against breaking changes.

By default **Use workspace mappings for filters** is selected. Builds are triggered whenever a change is checked in under a path specified in your source mappings.

Otherwise, you can clear this check box and specify the paths in the trigger.

How it affects your developers

When developers try 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 workspace mappings.
- You can run gated builds on either a [Microsoft-hosted agent](#) or a [self-hosted agent](#).

FAQ

I get the following error when running a pipeline:

```
The shelveset <xyz> could not be found for check-in
```

- Is your [job authorization scope](#) set to **collection**? TFVC repositories are usually spread across the projects in your collection. You may be reading or writing to a folder that can only be accessed when the scope is the entire collection. You can set this in organization settings or in project setting under the **Pipelines** tab.

I get the following error when running a pipeline:

```
The underlying connection was closed: An unexpected error occurred on a receive. ##[error]Exit code 100 returned from process: file name 'tf', arguments 'vc workspace /new /location:local /permission:Public'
```

- This is usually an intermittent error caused when the service is experiencing technical issues. Please re-run the pipeline.

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](#).

Build Subversion repositories

3/6/2021 • 3 minutes to read • [Edit Online](#)

You can integrate your on-premises Subversion server with Azure Pipelines. The Subversion server must be accessible to Azure Pipelines.

NOTE

YAML pipelines do not work with Subversion repositories.

If your server is reachable from the hosted agents, then you can use the hosted agents to run manual, scheduled, or CI builds. Otherwise, you must set up self-hosted agents that can access your on-premises server and fetch the code.

To integrate with Subversion, create a **Subversion** service connection and use that to create a pipeline. CI triggers work through polling. In other words, Azure Pipelines periodically checks the Subversion server if there are any updates to code. If there are, then Azure Pipelines will start a new run.

If the Subversion server cannot be reached from Azure Pipelines, work with your IT department to open a network path between Azure Pipelines and your server. For example, you can add exceptions to your firewall rules to allow traffic from Azure Pipelines to flow through. See the section on [Azure DevOps IPs](#) to see which IP addresses you need to allow. Furthermore, you need to have a public DNS entry for the Subversion server so that Azure Pipelines can resolve the FQDN of your server to an IP address.

Reachable from Microsoft-hosted agents

A decision you have to make is whether to use Microsoft-hosted agents or self-hosted agents to run your pipelines. This often comes down to whether Microsoft-hosted agents can reach your server. To check whether they can, create a simple pipeline to use Microsoft-hosted agents and make sure to add a step to checkout source code from your server. If this passes, then you can continue using Microsoft-hosted agents.

Not reachable from Microsoft-hosted agents

If the simple test pipeline mentioned in the above section fails with an error, then the Subversion server is probably not reachable from Microsoft-hosted agents. This is probably caused by a firewall blocking traffic from these servers. You have two options in this case:

- Work with your IT department to open a network path between Microsoft-hosted agents and Subversion server. See the section on [networking](#) in Microsoft-hosted agents.
- Switch to using [self-hosted agents](#) or [scale-set agents](#). These agents can be set up within your network and hence will have access to the Subversion server. These agents only require outbound connections to Azure Pipelines. There is no need to open a firewall for inbound connections. Make sure that the name of the server you specified when creating the service connection is resolvable from the self-hosted agents.

Azure DevOps IP addresses

To allow traffic from Azure DevOps to reach your Subversion Server, add the IP addresses or service tags specified in [Inbound connections](#) to your firewall's allow-list. If you use ExpressRoute, make sure to also include [ExpressRoute IP ranges](#) to your firewall's allow-list.

FAQ

Problems related to Subversion server integration fall into the following categories:

- **Failing triggers:** My pipeline is not being triggered when I push an update to the repo.
- **Failing checkout:** My pipeline is being triggered, but it fails in the checkout step.

Failing triggers

I pushed a change to my server, but the pipeline is not being triggered.

Follow each of these steps to troubleshoot your failing triggers:

- Is your Subversion server accessible from Azure Pipelines? Azure Pipelines periodically polls Subversion server for changes. If the Subversion server is behind a firewall, this traffic may not reach your server. See [Azure DevOps IP Addresses](#) and verify that you have granted exceptions to all the required IP addresses. These IP addresses may have changed since you have originally set up the exception rules.
- Is your pipeline paused or disabled? Open the editor for the pipeline, and then select **Settings** to check. If your pipeline is paused or disabled, then triggers do not work.

I did not push any updates to my code, however the pipeline is still being triggered.

- The continuous integration trigger for Subversion works through polling. After each polling interval, Azure Pipelines attempts to contact the Subversion server to check if there have been any updates to the code. If Azure Pipelines is unable to reach the server (possibly due to a network issue), then we start a new run anyway assuming that there might have been code changes. In a few cases, Azure Pipelines may also create a dummy failed build with an error message to indicate that it was unable to reach the server.

Failing checkout

The checkout step fails with the error that the server cannot be resolved.

Do you use Microsoft-hosted agents? If so, these agents may not be able to reach your Bitbucket server. See [Not reachable from Microsoft-hosted agents](#) for more information.

Check out multiple repositories in your pipeline

4/20/2021 • 10 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020

Pipelines often rely on multiple repositories that contain source, tools, scripts, or other items that you need to build your code. By using multiple `checkout` steps in your pipeline, you can fetch and check out other repositories in addition to the one you use to store your YAML pipeline.

Specify multiple repositories

Repositories can be specified as a [repository resource](#), or inline with the `checkout` step.

The following repository types are supported.

Azure Repos Git (`git`)

- Azure DevOps Server 2020 (limited to repositories in the same organization)
- Azure DevOps Services

GitHub (`github`)

- Azure DevOps Services

GitHubEnterprise (`githubenterprise`)

- Azure DevOps Services

Bitbucket Cloud (`bitbucket`)

- Azure DevOps Services

IMPORTANT

Only [Azure Repos Git \(`git`\)](#) repositories in the same organization as the pipeline are supported for multi-repo checkout in Azure DevOps Server 2020.

NOTE

Azure Pipelines provides **Limit job scope** settings for Azure Repos Git repositories. To check out Azure Repos Git repositories hosted in another project, **Limit job scope** must be configured to allow access. For more information, see [Limit job authorization scope](#).

The following combinations of `checkout` steps are supported.

No `checkout` steps

The default behavior is as if `checkout: self` were the first step, and the current repository is checked out.

A single `checkout: none` step

No repositories are synced or checked out.

A single `checkout: self` step

The current repository is checked out.

A single `checkout` step that isn't `self` or `none`

The designated repository is checked out instead of `self`.

Multiple `checkout` steps

Each designated repository is checked out to a folder named after the repository, unless a different `path` is specified in the `checkout` step. To check out `self` as one of the repositories, use `checkout: self` as one of the `checkout` steps.

NOTE

When you check out Azure Repos Git repositories other than the one containing the pipeline, you may be prompted to authorize access to that resource before the pipeline runs for the first time. For more information, see [Why am I prompted to authorize resources the first time I try to check out a different repository?](#) in the [FAQ](#) section.

Repository resource definition

You must use a [repository resource](#) if your repository type requires a service connection or other extended resources field. The following repository types require a service connection.

REPOSITORY TYPE	SERVICE CONNECTION
Bitbucket Cloud	Bitbucket Cloud
GitHub	GitHub
GitHub Enterprise Server	GitHub Enterprise Server
Azure Repos Git repositories in a different organization than your pipeline	Azure Repos/Team Foundation Server

You may use a repository resource even if your repository type doesn't require a service connection, for example if you have a repository resource defined already for templates in a different repository.

In the following example, three repositories are declared as repository resources. The [Azure Repos Git repository in another organization](#), [GitHub](#), and [Bitbucket Cloud](#) repository resources require [service connections](#), which are specified as the `endpoint` for those repository resources. This example has four `checkout` steps, which checks out the three repositories declared as repository resources along with the current `self` repository that contains the pipeline YAML.

```

resources:
  repositories:
    - repository: MyGitHubRepo # The name used to reference this repository in the checkout step
      type: github
      endpoint: MyGitHubServiceConnection
      name: MyGitHubOrgOrUser/MyGitHubRepo
    - repository: MyBitbucketRepo
      type: bitbucket
      endpoint: MyBitbucketServiceConnection
      name: MyBitbucketOrgOrUser/MyBitbucketRepo
    - repository: MyAzureReposGitRepository # In a different organization
      endpoint: MyAzureReposGitServiceConnection
      type: git
      name: OtherProject/MyAzureReposGitRepo

trigger:
  - main

pool:
  vmImage: 'ubuntu-latest'

steps:
  - checkout: self
  - checkout: MyGitHubRepo
  - checkout: MyBitbucketRepo
  - checkout: MyAzureReposGitRepository

  - script: dir $(Build.SourcesDirectory)

```

If the `self` repository is named `CurrentRepo`, the `script` command produces the following output:

`CurrentRepo MyAzureReposGitRepo MyBitbucketRepo MyGitHubRepo`. In this example, the names of the repositories are used for the folders, because no `path` is specified in the checkout step. For more information on repository folder names and locations, see the following [Checkout path](#) section.

Inline syntax checkout

If your repository doesn't require a service connection, you can declare it inline with your `checkout` step.

NOTE

Only Azure Repos Git repositories in the same organization can use the inline syntax. Azure Repos Git repositories in a different organization, and other supported repository types require a [service connection](#) and must be declared as a [repository resource](#).

```

steps:
  - checkout: git://MyProject/MyRepo # Azure Repos Git repository in the same organization

```

NOTE

In the previous example, the `self` repository is not checked out. If you specify any `checkout` steps, you must include `checkout: self` in order for `self` to be checked out.

Checkout path

Unless a `path` is specified in the `checkout` step, source code is placed in a default directory. This directory is different depending on whether you are checking out a single repository or multiple repositories.

- **Single repository:** If you have a single `checkout` step in your job, or you have no checkout step which is equivalent to `checkout: self`, your source code is checked out into a directory called `s` located as a subfolder of `(Agent.BuildDirectory)`. If `(Agent.BuildDirectory)` is `C:\agent_work\1`, your code is checked out to `C:\agent_work\1\s`.
- **Multiple repositories:** If you have multiple `checkout` steps in your job, your source code is checked out into directories named after the repositories as a subfolder of `s` in `(Agent.BuildDirectory)`. If `(Agent.BuildDirectory)` is `C:\agent_work\1` and your repositories are named `tools` and `code`, your code is checked out to `C:\agent_work\1\s\tools` and `C:\agent_work\1\s\code`.

NOTE

If no `path` is specified in the `checkout` step, the name of the repository is used for the folder, not the `repository` value which is used to reference the repository in the `checkout` step.

If a `path` is specified for a `checkout` step, that path is used, relative to `(Agent.BuildDirectory)`.

NOTE

If you are using default paths, adding a second repository `checkout` step changes the default path of the code for the first repository. For example, the code for a repository named `tools` would be checked out to `C:\agent_work\1\s` when `tools` is the only repository, but if a second repository is added, `tools` would then be checked out to `C:\agent_work\1\s\tools`. If you have any steps that depend on the source code being in the original location, those steps must be updated.

Checking out a specific ref

The default branch is checked out unless you designate a specific ref.

If you are using inline syntax, designate the ref by appending `@<ref>`. For example:

```
- checkout: git://MyProject/MyRepo@features/tools # checks out the features/tools branch
- checkout: git://MyProject/MyRepo@refs/heads/features/tools # also checks out the features/tools branch
- checkout: git://MyProject/MyRepo@refs/tags/MyTag # checks out the commit referenced by MyTag.
```

When using a repository resource, specify the ref using the `ref` property. The following example checks out the `features/tools/` branch of the designated repository.

```
resources:
  repositories:
    - repository: MyGitHubRepo
      type: github
      endpoint: MyGitHubServiceConnection
      name: MyGitHubOrgOrUser/MyGitHubRepo
      ref: features/tools

steps:
  - checkout: MyGitHubRepo
```

Triggers

You can trigger a pipeline when an update is pushed to the `self` repository or to any of the repositories declared as resources. This is useful, for instance, in the following scenarios:

- You consume a tool or a library from a different repository. You want to run tests for your application whenever the tool or library is updated.
- You keep your YAML file in a separate repository from the application code. You want to trigger the pipeline every time an update is pushed to the application repository.

IMPORTANT

Repository resource triggers only work for Azure Repos Git repositories in the same organization at present. They do not work for GitHub or Bitbucket repository resources.

If you do not specify a `trigger` section in a repository resource, then the pipeline won't be triggered by changes to that repository. If you specify a `trigger` section, then the behavior for triggering is similar to how CI triggers work for the self repository.

If you specify a `trigger` section for multiple repository resources, then a change to any of them will start a new run.

The trigger for `self` repository can be defined in a `trigger` section at the root of the YAML file, or in a repository resource for `self`. For example, the following two are equivalent.

```
trigger:
- main

steps:
...
```

```
resources:
repositories:
- repository: self
  type: git
  name: MyProject/MyGitRepo
trigger:
- main

steps:
...
```

NOTE

It is an error to define the trigger for `self` repository twice. Do not define it both at the root of the YAML file and in the `resources` section.

When a pipeline is triggered, Azure Pipelines has to determine the version of the YAML file that should be used and a version for each repository that should be checked out. If a change to the `self` repository triggers a pipeline, then the commit that triggered the pipeline is used to determine the version of the YAML file. If a change to any other repository resource triggers the pipeline, then the latest version of YAML from the **default branch** of `self` repository is used.

When an update to one of the repositories triggers a pipeline, then the following variables are set based on triggering repository:

- `Build.Repository.ID`
- `Build.Repository.Name`
- `Build.Repository.Provider`

- Build.Repository.Uri
- Build.SourceBranch
- Build.SourceBranchName
- Build.SourceVersion
- Build.SourceVersionMessage

For the triggering repository, the commit that triggered the pipeline determines the version of the code that is checked out. For other repositories, the `ref` defined in the YAML for that repository resource determines the default version that is checked out.

Consider the following example, where the `self` repository contains the YAML file and repositories `A` and `B` contain additional source code.

```
trigger:
- main
- feature

resources:
  repositories:
    - repository: A
      type: git
      name: MyProject/A
      ref: main
      trigger:
        - main

    - repository: B
      type: git
      name: MyProject/B
      ref: release
      trigger:
        - main
        - release
```

The following table shows which versions are checked out for each repository by a pipeline using the above YAML file, unless you explicitly override the behavior during `checkout`.

CHANGE MADE TO	PIPELINE TRIGGERED	VERSION OF YAML	VERSION OF SELF	VERSION OF A	VERSION OF B
<code>main</code> in <code>self</code>	Yes	commit from <code>main</code> that triggered the pipeline	commit from <code>main</code> that triggered the pipeline	latest from <code>main</code>	latest from <code>release</code>
<code>feature</code> in <code>self</code>	Yes	commit from <code>feature</code> that triggered the pipeline	commit from <code>feature</code> that triggered the pipeline	latest from <code>main</code>	latest from <code>release</code>
<code>main</code> in <code>A</code>	Yes	latest from <code>main</code>	latest from <code>main</code>	commit from <code>main</code> that triggered the pipeline	latest from <code>release</code>

CHANGE MADE TO	PIPELINE TRIGGERED	VERSION OF YAML	VERSION OF SELF	VERSION OF A	VERSION OF B
main in B	Yes	latest from main	latest from main	latest from main	commit from main that triggered the pipeline
release in B	Yes	latest from main	latest from main	latest from main	commit from release that triggered the pipeline

You can also trigger the pipeline when you create or update a pull request in any of the repositories. To do this, declare the repository resources in the YAML files as in the examples above, and configure a branch policy in the repository (Azure Repos only).

Repository details

When you check out multiple repositories, some details about the `self` repository are available as [variables](#). When you use multi-repo triggers, some of those variables have information about the triggering repository instead. Details about all of the repositories consumed by the job are available as a [template context object](#) called `resources.repositories`.

For example, to get the ref of a non-`self` repository, you could write a pipeline like this:

```
resources:
  repositories:
    - repository: other
      type: git
      name: MyProject/OtherTools

variables:
  tools.ref: ${ resources.repositories['other'].ref }

steps:
  - checkout: self
  - checkout: other
  - bash: |
    echo "Tools version: $TOOLS_REF"
```

FAQ

- [Why can't I check out a repository from another project? It used to work.](#)
- [Why am I prompted to authorize resources the first time I try to check out a different repository?](#)

Why can't I check out a repository from another project? It used to work.

Azure Pipelines provides a **Limit job authorization scope to current project** setting, that when enabled, doesn't permit the pipeline to access resources outside of the project that contains the pipeline. This setting can be set at either the organization or project level. If this setting is enabled, you won't be able to check out a repository in another project unless you explicitly grant access. For more information, see [Job authorization scope](#).

Why am I prompted to authorize resources the first time I try to check out a different repository?

When you check out Azure Repos Git repositories other than the one containing the pipeline, you may be prompted to authorize access to that resource before the pipeline runs for the first time. These prompts are

displayed on the pipeline run summary page.

This pipeline needs permission to access a resource before this run can continue

[View](#)

There was a **resource authorization issue**: "The pipeline is not valid. Could not find a pool with name Azure Pipelines. The pool does not exist or has not been authorized for use. For authorization details, refer to <https://aka.ms/yamlauthz>."

[Authorize resources](#)

Choose **View** or **Authorize resources**, and follow the prompts to authorize the resources.

Waiting for review

X



Permit access?

X

Granting permission here will permit the use of Repository 'devops-project-samples' for all waiting and future runs of this pipeline.

[Cancel](#) [Permit](#)

For more information, see [Troubleshooting authorization for a YAML pipeline](#).

Build pipeline history

6/2/2020 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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**.

FAQ

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.

I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

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](#).

My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

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](#).

Specify events that trigger pipelines

5/13/2021 • 2 minutes to read • [Edit Online](#)

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Use triggers to run a pipeline automatically. Azure Pipelines supports many types of triggers. Based on your pipeline's type, select the appropriate trigger from the list below:

Classic build pipelines and YAML pipelines

Continuous integration (CI) triggers vary based on the type of repository you build in your pipeline.

- [CI triggers in Azure Repos Git](#)
- [CI triggers in GitHub](#)
- [CI triggers in Bitbucket Cloud](#)
- [CI triggers in TFVC](#)

Pull request validation (PR) triggers also vary based on the type of repository.

- [PR triggers in Azure Repos Git](#)
- [PR triggers in GitHub](#)
- [PR triggers in Bitbucket Cloud](#)

Gated check-in is supported for TFVC repositories.

Comment triggers are supported only for GitHub repositories.

Scheduled triggers are independent of the repository and allow you to run a pipeline according to a schedule.

Pipeline triggers in YAML pipelines and [build completion triggers](#) in classic build pipelines allow you to trigger one pipeline upon the completion of another.

Branch consideration for triggers in YAML pipelines

YAML pipelines can have different versions of the pipeline in different branches, which can affect which version of the pipeline's triggers are evaluated and which version of the pipeline should run.

TRIGGER TYPE	PIPELINE YAML VERSION
CI triggers (<code>trigger</code>)	The version of the pipeline in the pushed branch is used.
PR triggers (<code>pr</code>)	The version of the pipeline in the source branch for the pull request is used.
GitHub pull request comment triggers	The version of the pipeline in the source branch for the pull request is used.

Trigger type	Pipeline YAML version
Scheduled triggers	See Branch considerations for scheduled triggers .
Pipeline completion triggers	See Branch considerations for pipeline completion triggers .
Azure Pipelines Action	See Branch considerations .

Classic release pipelines

[Continuous deployment triggers](#) help you start classic releases after a classic build or YAML pipeline completes.

[Scheduled release triggers](#) allow you to run a release pipeline according to a schedule.

[Pull request release triggers](#) are used to deploy a pull request directly using classic releases.

[Stage triggers in classic release](#) are used to configure how each stage in a classic release is triggered.

Configure schedules for pipelines

6/24/2021 • 19 minutes to read • [Edit Online](#)

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Azure Pipelines provides several types of triggers to configure how your pipeline starts.

- Scheduled triggers start your pipeline based on a schedule, such as a nightly build. This article provides guidance on using scheduled triggers to run your pipelines based on a schedule.
- Event-based triggers start your pipeline in response to events, such as creating a pull request or pushing to a branch. For information on using event-based triggers, see [Triggers in Azure Pipelines](#).

You can combine scheduled and event-based triggers in your pipelines, for example to validate the build every time a push is made ([CI trigger](#)), when a pull request is made ([PR trigger](#)), and a nightly build (Scheduled trigger). If you want to build your pipeline only on a schedule, and not in response to event-based triggers, ensure that your pipeline does not have any other triggers enabled. For example, YAML pipelines in a GitHub repository have CI triggers and PR triggers enabled by default. For information on disabling default triggers, see [Triggers in Azure Pipelines](#) and navigate to the section that covers your repository type.

Scheduled triggers

- [YAML](#)
- [Classic](#)

IMPORTANT

Scheduled triggers defined using the pipeline settings UI take precedence over YAML scheduled triggers.

If your YAML pipeline has both YAML scheduled triggers and UI defined scheduled triggers, only the UI defined scheduled triggers are run. To run the YAML defined scheduled triggers in your YAML pipeline, you must remove the scheduled triggers defined in the pipeline settings UI. Once all UI scheduled triggers are removed, a push must be made in order for the YAML scheduled triggers to start being evaluated.

To delete UI scheduled triggers from a YAML pipeline, see [UI settings override YAML scheduled triggers](#).

Scheduled triggers configure a pipeline to run on a schedule defined using [cron syntax](#).

```
schedules:  
- cron: string # cron syntax defining a schedule  
  displayName: string # friendly name given to a specific schedule  
  branches:  
    include: [ string ] # which branches the schedule applies to  
    exclude: [ string ] # which branches to exclude from the schedule  
  always: boolean # whether to always run the pipeline or only if there have been source code changes since  
  the last successful scheduled run. The default is false.
```

Scheduled pipelines in YAML have the following constraints.

- The time zone for cron schedules is UTC.
- If you specify an `exclude` clause without an `include` clause for `branches`, it is equivalent to specifying `*` in the `include` clause.
- You cannot use pipeline variables when specifying schedules.
- If you use [templates in your YAML file](#), then the schedules must be specified in the main YAML file and not in the template files.

Branch considerations for scheduled triggers

Scheduled triggers are evaluated for a branch when the following events occur.

- A pipeline is created.
- A pipeline's YAML file is updated, either from a push, or by editing it in the pipeline editor.
- A pipeline's YAML file path is [updated to reference a different YAML file](#). This change only updates the default branch, and therefore only picks up schedules in the updated YAML file for the default branch. If any other branches subsequently merge the default branch, for example `git pull origin main`, the scheduled triggers from the newly referenced YAML file are evaluated for that branch.
- A new branch is created.

After one of these events occurs in a branch, any scheduled runs for that branch are added, if that branch matches the branch filters for the scheduled triggers contained in the YAML file in that branch.

IMPORTANT

Scheduled runs for a branch are added only if the branch matches the branch filters for the scheduled triggers in the YAML file in that particular branch.

For example, a pipeline is created with the following schedule, and this version of the YAML file is checked into the `main` branch. This schedule builds the `main` branch on a daily basis.

```
# YAML file in the main branch
schedules:
- cron: "0 0 * * *"
  displayName: Daily midnight build
  branches:
    include:
    - main
```

Next, a new branch is created based off of `main`, named `new-feature`. The scheduled triggers from the YAML file in the new branch are read, and since there is no match for the `new-feature` branch, no changes are made to the scheduled builds, and the `new-feature` branch is not built using a scheduled trigger.

If `new-feature` is added to the `branches` list and this change is pushed to the `new-feature` branch, the YAML file is read, and since `new-feature` is now in the branches list, a scheduled build is added for the `new-feature` branch.

```
# YAML file in the new-feature-branch
schedules:
- cron: "0 0 * * *"
  displayName: Daily midnight build
  branches:
    include:
    - main
    - new-feature
```

Now consider that a branch named `release` is created based off `main`, and then `release` is added to the branch filters in the YAML file in the `main` branch, but not in the newly created `release` branch.

```
# YAML file in the release branch
schedules:
- cron: "0 0 * * *"
  displayName: Daily midnight build
  branches:
    include:
    - main

# YAML file in the main branch with release added to the branches list
schedules:
- cron: "0 0 * * *"
  displayName: Daily midnight build
  branches:
    include:
    - main
    - release
```

Because `release` was added to the branch filters in the `main` branch, but not to the branch filters in the `release` branch, the `release` branch won't be built on that schedule. Only when the `feature` branch is added to the branch filters in the YAML file in the **feature branch** will the scheduled build be added to the scheduler.

Scheduled builds are not supported in YAML syntax in Azure DevOps Server 2019. After you create your YAML build pipeline, you can use pipeline settings to specify a scheduled trigger.

YAML pipelines are not available on TFS.

Examples

- [YAML](#)
- [Classic](#)

The following example defines two schedules:

```
schedules:
- cron: "0 0 * * *"
  displayName: Daily midnight build
  branches:
    include:
    - main
    - releases/*
    exclude:
    - releases/ancient/*
- cron: "0 12 * * 0"
  displayName: Weekly Sunday build
  branches:
    include:
    - releases/*
  always: true
```

The first schedule, **Daily midnight build**, runs a pipeline at midnight every day, but only if the code has changed since the last successful scheduled run, for `main` and all `releases/*` branches, except those under `releases/ancient/*`.

The second schedule, **Weekly Sunday build**, runs a pipeline at noon on Sundays, whether the code has changed or not since the last run, for all `releases/*` branches.

NOTE

The time zone for cron schedules is UTC, so in these examples, the midnight build and the noon build are at midnight and noon in UTC.

For more examples, see [Migrating from the classic editor](#).

Scheduled builds are not supported in YAML syntax in Azure DevOps Server 2019. After you create your YAML build pipeline, you can use pipeline settings to specify a scheduled trigger.

YAML pipelines are not available on TFS.

Cron syntax

- [YAML](#)
- [Classic](#)

Each Azure Pipelines scheduled trigger cron expression is a space-delimited expression with five entries in the following order.

```
mm HH DD MM DW
 \ \ \ \ \_ Days of week
 \ \ \ \ \_ Months
 \ \ \ \_ Days
 \ \ \_ Hours
 \_ Minutes
```

FIELD	ACCEPTED VALUES
Minutes	0 through 59
Hours	0 through 23
Days	1 through 31
Months	1 through 12, full English names, first three letters of English names
Days of week	0 through 6 (starting with Sunday), full English names, first three letters of English names

Values can be in the following formats.

FORMAT	EXAMPLE	DESCRIPTION
Wildcard	*	Matches all values for this field
Single value	5	Specifies a single value for this field
Comma delimited	3,5,6	Specifies multiple values for this field. Multiple formats can be combined, like 1,3-6

FORMAT	EXAMPLE	DESCRIPTION
Ranges	<code>1-3</code>	The inclusive range of values for this field
Intervals	<code>*/4</code> or <code>1-5/2</code>	Intervals to match for this field, such as every 4th value or the range 1-5 with a step interval of 2

EXAMPLE	CRON EXPRESSION
Build every Monday, Wednesday, and Friday at 6:00 PM	<code>0 18 * * Mon,Wed,Fri</code> , <code>0 18 * * 1,3,5</code> , or <code>0 18 * * 1-5/2</code>
Build every 6 hours	<code>0 0,6,12,18 * * *</code> , <code>0 */6 * * *</code> or <code>0 0-18/6 * * *</code>
Build every 6 hours starting at 9:00 AM	<code>0 9,15,21 * * *</code> or <code>0 9-21/6 * * *</code>

For more information on supported formats, see [Crontab Expression](#).

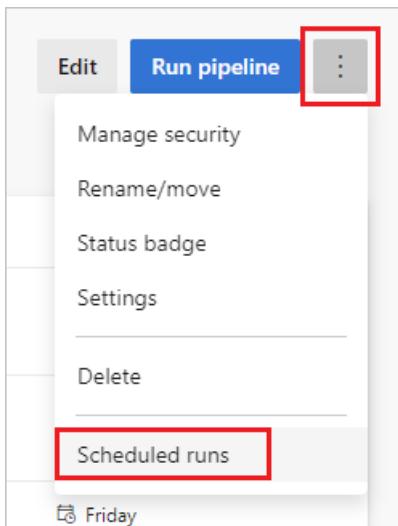
Scheduled builds are not supported in YAML syntax in Azure DevOps Server 2019. After you create your YAML build pipeline, you can use pipeline settings to specify a scheduled trigger.

YAML pipelines are not available on TFS.

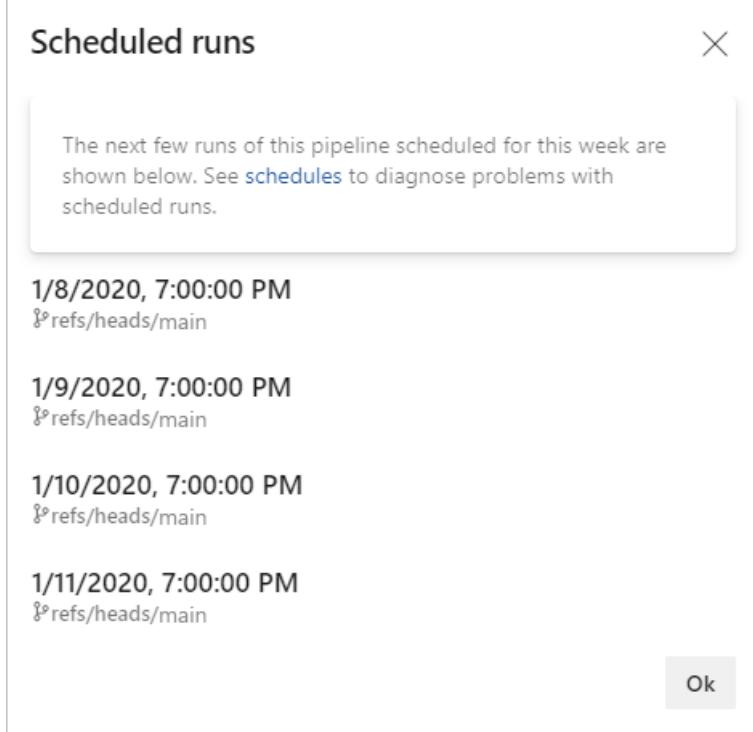
Scheduled runs view

- [YAML](#)
- [Classic](#)

You can view a preview of upcoming scheduled builds by choosing **Scheduled runs** from the context menu on the [pipeline details page](#) for your pipeline.



After you create or update your scheduled triggers, you can verify them using this view.



This example displays the scheduled runs for the following schedule.

```
schedules:  
- cron: "0 0 * * *"  
displayName: Daily midnight build  
branches:  
  include:  
  - main
```

The **Scheduled runs** windows displays the times converted to the local time zone set on the computer used to browse to the Azure DevOps portal. This example displays a screenshot taken in the EST time zone.

Scheduled builds are not supported in YAML syntax in Azure DevOps Server 2019. After you create your YAML build pipeline, you can use pipeline settings to specify a scheduled trigger.

YAML pipelines are not available on TFS.

Running even when there are no code changes

NOTE

Scheduled builds always run on schedule regardless of code changes on TFS 2018.1 and lower.

By default, your pipeline does not run as scheduled if there have been no code changes since the last successful scheduled run. For instance, consider that you have scheduled a pipeline to run every night at 9:00pm. During the weekdays, you push various changes to your code. The pipeline runs as per schedule. During the weekends, you do not make any changes to your code. If there have been no code changes since the scheduled run on Friday, then the pipeline does not run as scheduled during the weekend.

- [YAML](#)
- [Classic](#)

To force a pipeline to run even when there are no code changes, you can use the `always` keyword.

```
schedules:  
- cron: ...  
...  
always: true
```

Scheduled builds are not supported in YAML syntax in this version of Azure DevOps Server. After you create your YAML build pipeline, you can use pipeline settings to specify a scheduled trigger.

YAML pipelines are not available on TFS.

Limits on the number of scheduled runs

There are certain limits on how often you can schedule a pipeline to run. These limits have been put in place to prevent misuse of Azure Pipelines resources - particularly the Microsoft-hosted agents. This limit is around 1000 runs per pipeline per week.

Migrating from the classic editor

The following examples show you how to migrate your schedules from the classic editor to YAML.

- [Example: Nightly build of Git repo in multiple time zones](#)
- [Example: Nightly build with different frequencies](#)

Example: Nightly build of Git repo in multiple time zones

In this example, the classic editor scheduled trigger has two entries, producing the following builds.

- Every Monday - Friday at 3:00 AM (UTC + 5:30 time zone), build branches that meet the `features/india/*` branch filter criteria

The screenshot shows the 'When to build' section of a scheduled trigger. It includes a summary line '🕒 Mon through Fri at 3:00'. Below it are fields for selecting days of the week (Monday through Friday are checked, Saturday and Sunday are unchecked) and setting the time to '03h 00m'. A dropdown menu indicates the time zone is '(UTC+05:30) Chennai, Kolkata, Mumbai, New De...'. There is also a checked checkbox for 'Only schedule builds if the source or pipeline has changed'. At the bottom, there's a 'Branch filters' section with a 'Type' dropdown set to 'Include' and a 'Branch specification' input field containing the value 'features/india/*'. A '+' button is available for adding more filters.

- Every Monday - Friday at 3:00 AM (UTC - 5:00 time zone), build branches that meet the `features/nc/*` branch filter criteria

⌚ Mon through Fri at 3:00

When to build

Mon Tue Wed Thu Fri Sat Sun

03h 00m (UTC-05:00) Eastern Time (US & Canada)

Only schedule builds if the source or pipeline has changed

Branch filters

Type	Branch specification
Include	 features/nc/* <input type="button" value="▼"/> <input type="button" value="Delete"/>

[+ Add](#)

The equivalent YAML scheduled trigger is:

```
schedules:
- cron: "30 21 * * Sun-Thu"
  displayName: M-F 3:00 AM (UTC + 5:30) India daily build
  branches:
    include:
      - /features/india/*
- cron: "0 8 * * Mon-Fri"
  displayName: M-F 3:00 AM (UTC - 5) NC daily build
  branches:
    include:
      - /features/nc/*
```

In the first schedule, **M-F 3:00 AM (UTC + 5:30) India daily build**, the cron syntax (`mm HH DD MM DW`) is

`30 21 * * Sun-Thu`.

- Minutes and Hours - `30 21` - This maps to `21:30 UTC` (`9:30 PM UTC`). Since the specified time zone in the classic editor is **UTC + 5:30**, we need to subtract 5 hours and 30 minutes from the desired build time of 3:00 AM to arrive at the desired UTC time to specify for the YAML trigger.
- Days and Months are specified as wildcards since this schedule doesn't specify to run only on certain days of the month or on a specific month.
- Days of the week - `Sun-Thu` - because of the timezone conversion, for our builds to run at 3:00 AM in the **UTC + 5:30 India time zone**, we need to specify starting them the previous day in UTC time. We could also specify the days of the week as `0-4` or `0,1,2,3,4`.

In the second schedule, **M-F 3:00 AM (UTC - 5) NC daily build**, the cron syntax is `0 8 * * Mon-Fri`.

- Minutes and Hours - `0 8` - This maps to `8:00 AM UTC`. Since the specified time zone in the classic editor is **UTC - 5:00**, we need to add 5 hours from the desired build time of 3:00 AM to arrive at the desired UTC time to specify for the YAML trigger.
- Days and Months are specified as wildcards since this schedule doesn't specify to run only on certain days of the month or on a specific month.
- Days of the week - `Mon-Fri` - Because our timezone conversions don't span multiple days of the week for our desired schedule, we don't need to do any conversion here. We could also specify the days of the week as `1-5` or `1,2,3,4,5`.

IMPORTANT

The UTC time zones in YAML scheduled triggers don't account for daylight savings time.

TIP

When using 3 letter days of the week and wanting a span of multiple days through Sun, Sun should be considered the first day of the week e.g. For a schedule of midnight EST, Thursday to Sunday, the cron syntax is `0 5 * * Sun,Thu-Sat`

Example: Nightly build with different frequencies

In this example, the classic editor scheduled trigger has two entries, producing the following builds.

- Every Monday - Friday at 3:00 AM UTC, build branches that meet the `main` and `releases/*` branch filter criteria

⌚ Mon through Fri at 3:00

When to build

Mon Tue Wed Thu Fri Sat Sun

03h 00m (UTC) Coordinated Universal Time

Only schedule builds if the source or pipeline has changed

Branch filters

Type	Branch specification
Include	main <input type="button"/>
Include	releases/* <input type="button"/>

+ Add

- Every Sunday at 3:00 AM UTC, build the `releases/lastversion` branch, even if the source or pipeline hasn't changed

⌚ Sun at 3:00

When to build

Mon Tue Wed Thu Fri Sat Sun

03h 00m (UTC) Coordinated Universal Time

Only schedule builds if the source or pipeline has changed

Branch filters

Type	Branch specification
Include	releases/lastversion <input type="button"/>

+ Add

The equivalent YAML scheduled trigger is:

```

schedules:
- cron: "0 3 * * Mon-Fri"
  displayName: M-F 3:00 AM (UTC) daily build
  branches:
    include:
    - main
    - /releases/*
- cron: "0 3 * * Sun"
  displayName: Sunday 3:00 AM (UTC) weekly latest version build
  branches:
    include:
    - /releases/lastversion
  always: true

```

In the first schedule, **M-F 3:00 AM (UTC) daily build**, the cron syntax is `0 3 * * Mon-Fri`.

- Minutes and Hours - `0 3` - This maps to `3:00 AM UTC`. Since the specified time zone in the classic editor is **UTC**, we don't need to do any time zone conversions.
- Days and Months are specified as wildcards since this schedule doesn't specify to run only on certain days of the month or on a specific month.
- Days of the week - `Mon-Fri` - because there is no timezone conversion, the days of the week map directly from the classic editor schedule. We could also specify the days of the week as `1,2,3,4,5`.

In the second schedule, **Sunday 3:00 AM (UTC) weekly latest version build**, the cron syntax is `0 3 * * Sun`

- Minutes and Hours - `0 3` - This maps to `3:00 AM UTC`. Since the specified time zone in the classic editor is **UTC**, we don't need to do any time zone conversions.
- Days and Months are specified as wildcards since this schedule doesn't specify to run only on certain days of the month or on a specific month.
- Days of the week - `Sun` - Because our timezone conversions don't span multiple days of the week for our desired schedule, we don't need to do any conversion here. We could also specify the days of the week as `0`.
- We also specify `always: true` since this build is scheduled to run whether or not the source code has been updated.

FAQ

- [I defined a schedule in the YAML file. But it didn't run. What happened?](#)
- [My YAML schedules were working fine. But, they stopped working now. How do I debug this?](#)
- [My code hasn't changed, yet a scheduled build is triggered. Why?](#)
- [I see the planned run in the Scheduled runs panel. However, it does not run at that time. Why?](#)
- [Schedules defined in YAML pipeline work for one branch but not the other. How do I fix this?](#)

I defined a schedule in the YAML file. But it didn't run. What happened?

- Check the next few runs that Azure Pipelines has scheduled for your pipeline. You can find these by selecting the **Scheduled runs** action in your pipeline. The list is filtered down to only show you the upcoming few runs over the next few days. If this does not meet your expectation, it is probably the case that you have mistyped your cron schedule, or you do not have the schedule defined in the correct branch. Read the topic above to understand how to configure schedules. Reevaluate your cron syntax. All the times for cron schedules are in **UTC**.
- Make a small trivial change to your YAML file and push that update into your repository. If there was any problem in reading the schedules from the YAML file earlier, it should be fixed now.
- If you have any schedules defined in the UI, then your YAML schedules are not honored. Ensure that you

do not have any UI schedules by navigating to the editor for your pipeline and then selecting [Triggers](#).

- There is a limit on the number of runs you can schedule for a pipeline. Read more about [limits](#).
- If there are no changes to your code, they Azure Pipelines may not start new runs. Learn how to [override](#) this behavior.

My YAML schedules were working fine. But, they stopped working now. How do I debug this?

- If you did not specify `always:true`, your pipeline won't be scheduled unless there are any updates made to your code. Check whether there have been any code changes and how you [configured the schedules](#).
- There is a [limit](#) on how many times you can schedule your pipeline. Check if you have exceeded those limits.
- Check if someone enabled additional schedules in the UI. Open the editor for your pipeline, and select [Triggers](#). If they defined schedules in the UI, then your YAML schedules won't be honored.
- Check if your pipeline is paused or disabled. Select [Settings](#) for your pipeline.
- Check the next few runs that Azure Pipelines has scheduled for your pipeline. You can find these by selecting the [Scheduled runs](#) action in your pipeline. If you do not see the schedules that you expected, make a small trivial change to your YAML file, and push the update to your repository. This should re-sync the schedules.
- If you use GitHub for storing your code, it is possible that Azure Pipelines may have been throttled by GitHub when it tried to start a new run. Check if you can start a new run manually.

My code hasn't changed, yet a scheduled build is triggered. Why?

- You might have enabled an option to [always](#) run a scheduled build even if there are no code changes. If you use a YAML file, verify the syntax for the schedule in the YAML file. If you use classic pipelines, verify if you checked this option in the scheduled triggers.
- You might have updated the build pipeline or some property of the pipeline. This will cause a new run to be scheduled even if you have not updated your source code. Verify the [History](#) of changes in the pipeline using the classic editor.
- You might have updated the service connection used to connect to the repository. This will cause a new run to be scheduled even if you have not updated your source code.
- Azure Pipelines first checks if there are any updates to your code. If Azure Pipelines is unable to reach your repository or get this information, it will either start a scheduled run anyway or it will create a failed run to indicate that it is unable to reach the repository. If you notice that a run was created and that failed immediately, this is likely the reason. It is a dummy build to let you know that Azure Pipelines is unable to reach your repository.

I see the planned run in the Scheduled runs panel. However, it does not run at that time. Why?

- The [Scheduled runs](#) panel shows all potential schedules. However, it may not actually run unless you have made real updates to the code. To force a schedule to always run, ensure that you have set the [always](#) property in the YAML pipeline, or checked the option to always run in a classic pipeline.

Schedules defined in YAML pipeline work for one branch but not the other. How do I fix this?

Schedules are defined in YAML files, and these files are associated with branches. If you want a pipeline to be scheduled for a particular branch, say `features/x`, then make sure that the YAML file **in that branch** has the cron schedule defined in it, and that it has the correct branch inclusions for the schedule. The YAML file in the `features/x` branch should have the following in this example:

```
schedules:  
- cron: "0 12 * * 0"    # replace with your schedule  
branches:  
  include:  
    - features/X
```

For more information, see [Branch considerations for scheduled triggers](#).

Trigger one pipeline after another

5/13/2021 • 7 minutes to read • [Edit Online](#)

Large products have several components that are dependent on each other. These components are often independently built. When an upstream component (a library, for example) changes, the downstream dependencies have to be rebuilt and revalidated.

In situations like these, add a pipeline trigger to run your pipeline upon the successful completion of the **triggering pipeline**.

NOTE

Previously, you may have navigated to the classic editor for your YAML pipeline and configured **build completion triggers** in the UI. While that model still works, it is no longer recommended. The recommended approach is to specify **pipeline triggers** directly within the YAML file. Build completion triggers as defined in the classic editor have various drawbacks, which have now been addressed in pipeline triggers. For instance, there is no way to trigger a pipeline on the same branch as that of the triggering pipeline using build completion triggers.

Configure pipeline resource triggers

To trigger a pipeline upon the completion of another pipeline, specify the triggering pipeline as a [pipeline resource](#).

The following example has two pipelines - `app-ci` (the pipeline defined by the YAML snippet), and `security-lib-ci` (the triggering pipeline referenced by the pipeline resource). We want the `app-ci` pipeline to run automatically every time a new version of `security-lib-ci` is built.

```
# this is being defined in app-ci pipeline
resources:
  pipelines:
    - pipeline: securitylib # Name of the pipeline resource
      source: security-lib-ci # Name of the pipeline referenced by the pipeline resource
      project: FabrikamProject # Required only if the source pipeline is in another project
      trigger: true # Run app-ci pipeline when any run of security-lib-ci completes
```

- `pipeline: securitylib` specifies the name of the pipeline resource, and is used when referring to the pipeline resource from other parts of the pipeline, such as pipeline resource variables.
- `source: security-lib-ci` specifies the name of the pipeline referenced by this pipeline resource. You can retrieve a pipeline's name from the Azure DevOps portal in several places, such as the [Pipelines landing page](#). By default, pipelines are named after the repository that contains the pipeline. To update a pipeline's name, see [Pipeline settings](#).
- `project: FabrikamProject` - If the triggering pipeline is in another Azure DevOps project, you must specify the project name. This property is optional if both the source pipeline and the triggered pipeline are in the same project.
- `trigger: true` - Use this syntax to trigger the pipeline when any version of the source pipeline completes. See the following sections in this article to learn how to filter which versions of the source pipeline completing will trigger a run. When filters are specified, the source pipeline run must match all of the filters to trigger a run.

If the triggering pipeline and the triggered pipeline use the same repository, then both the pipelines will run using the same commit when one triggers the other. This is helpful if your first pipeline builds the code, and the second pipeline tests it. However, if the two pipelines use different repositories, then the triggered pipeline will use the version of the code in the branch specified by the **Default branch for manual and scheduled builds** setting, as described in the following [Branch considerations for pipeline completion triggers](#) section.

Branch filters

You can optionally specify the branches to include or exclude when configuring the trigger. If you specify branch filters, a new pipeline is triggered whenever a source pipeline run is successfully completed that matches the branch filters. In the following example, the `app-ci` pipeline runs if the `security-lib-ci` completes on any `releases/*` branch, except for `releases/old*`.

```
# this is being defined in app-ci pipeline
resources:
  pipelines:
    - pipeline: securitylib
      source: security-lib-ci
      trigger:
        branches:
          include:
            - releases/*
          exclude:
            - releases/old*
```

NOTE

If your branch filters aren't working, try using the prefix `refs/heads/`. For example, use `refs/heads/releases/old*` instead of `releases/old*`.

Tag filters

NOTE

Tag filter support for pipeline resources requires [Azure DevOps Server 2020 Update 1](#) or greater.

The `tags` property of the `trigger` filters which pipeline completion events can trigger your pipeline. If the triggering pipeline matches all of the tags in the `tags` list, the pipeline runs.

```
resources:
  pipelines:
    - pipeline: MyCIAlias
      source: Farbrikam-CI
      trigger:
        tags:      # This filter is used for triggering the pipeline run
        - Production # Tags are AND'ed
        - Signed
```

NOTE

The pipeline resource also has a `tags` property. The `tags` property of the pipeline resource is used to determine which pipeline run to retrieve artifacts from, when the pipeline is triggered manually or by a scheduled trigger. For more information, see [Resources: pipelines](#) and [Evaluation of artifact version](#).

Stage filters

NOTE

Stages filters for pipeline resource triggers requires Azure DevOps Server 2020 Update 1 or greater.

You can trigger your pipeline when one or more stages of the triggering pipeline complete by using the `stages` filter. If you provide multiple stages, the triggered pipeline runs when all of the listed stages complete.

```
resources:  
  pipelines:  
    - pipeline: MyCIAlias  
      source: Farbrikam-CI  
      trigger:  
        stages:          # This stage filter is used when evaluating conditions for  
        - PreProduction # triggering your pipeline. On successful completion of all the stages  
        - Production   # provided, your pipeline will be triggered.
```

Branch considerations

Pipeline completion triggers use the **Default branch for manual and scheduled builds** setting to determine which branch's version of a YAML pipeline's branch filters to evaluate when determining whether to run a pipeline as the result of another pipeline completing. By default this setting points to the default branch of the repository.

When a pipeline completes, the Azure DevOps runtime evaluates the pipeline resource trigger branch filters of any pipelines with pipeline completion triggers that reference the completed pipeline. A pipeline can have multiple versions in different branches, so the runtime evaluates the branch filters in the pipeline version in the branch specified by the **Default branch for manual and scheduled builds** setting. If there is a match, the pipeline runs, but the version of the pipeline that runs may be in a different branch depending on whether the triggered pipeline is in the same repository as the completed pipeline.

- If the two pipelines are in different repositories, the triggered pipeline version in the branch specified by **Default branch for manual and scheduled builds** is run.
- If the two pipelines are in the same repository, the triggered pipeline version in the same branch as the triggering pipeline is run, even if that branch is different than the **Default branch for manual and scheduled builds**, and even if that version does not have branch filters that match the completed pipeline's branch. This is because the branch filters from the **Default branch for manual and scheduled builds** branch are used to determine if the pipeline should run, and not the branch filters in the version that is in the completed pipeline branch.

If your pipeline completion triggers don't seem to be firing, check the value of the **Default branch for manual and scheduled builds** setting for the triggered pipeline. The branch filters in that branch's version of the pipeline are used to determine whether the pipeline completion trigger initiates a run of the pipeline. By default, **Default branch for manual and scheduled builds** is set to the default branch of the repository, but you can change it after the pipeline is created.

A typical scenario in which the pipeline completion trigger doesn't fire is when a new branch is created, the pipeline completion trigger branch filters are modified to include this new branch, but when the first pipeline completes on a branch that matches the new branch filters, the second pipeline doesn't trigger. This happens if the branch filters in the pipeline version in the **Default branch for manual and scheduled builds** branch don't match the new branch. To resolve this trigger issue you have the following two options.

- Update the branch filters in the pipeline in the **Default branch for manual and scheduled builds** branch

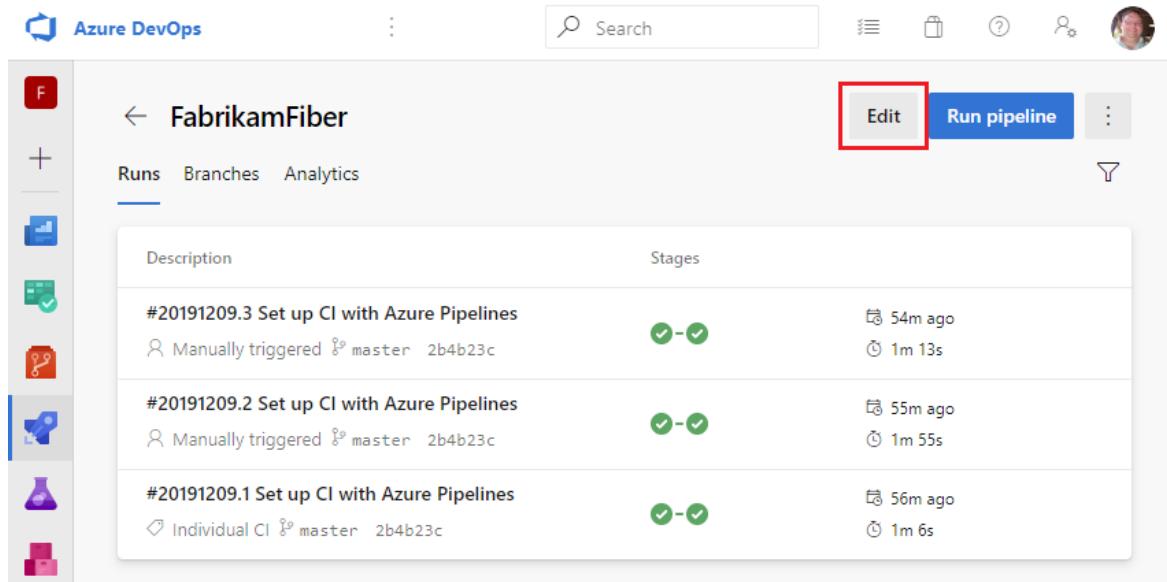
so that they match the new branch.

- Update the **Default branch for manual and scheduled builds** setting to a branch that has a version of the pipeline with the branch filters that match the new branch.

Default branch for manual and scheduled builds setting

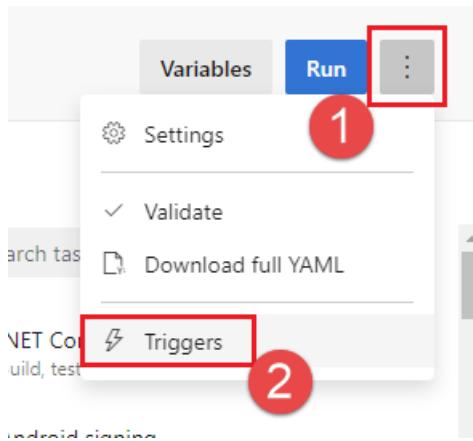
To view and update the **Default branch for manual and scheduled builds** setting:

1. Navigate to the [pipeline details](#) for your pipeline, and choose **Edit**.



The screenshot shows the Azure DevOps interface for a pipeline named 'FabrikamFiber'. On the left, there's a sidebar with various icons. The main area displays a table of recent pipeline runs. The first run is titled '#20191209.3 Set up CI with Azure Pipelines', triggered manually, and completed successfully 54m ago. The second run is '#20191209.2 Set up CI with Azure Pipelines', also triggered manually, completed successfully 55m ago. The third run is '#20191209.1 Set up CI with Azure Pipelines', triggered individually, completed successfully 56m ago. At the top right, there are buttons for 'Edit' (which is highlighted with a red box) and 'Run pipeline'.

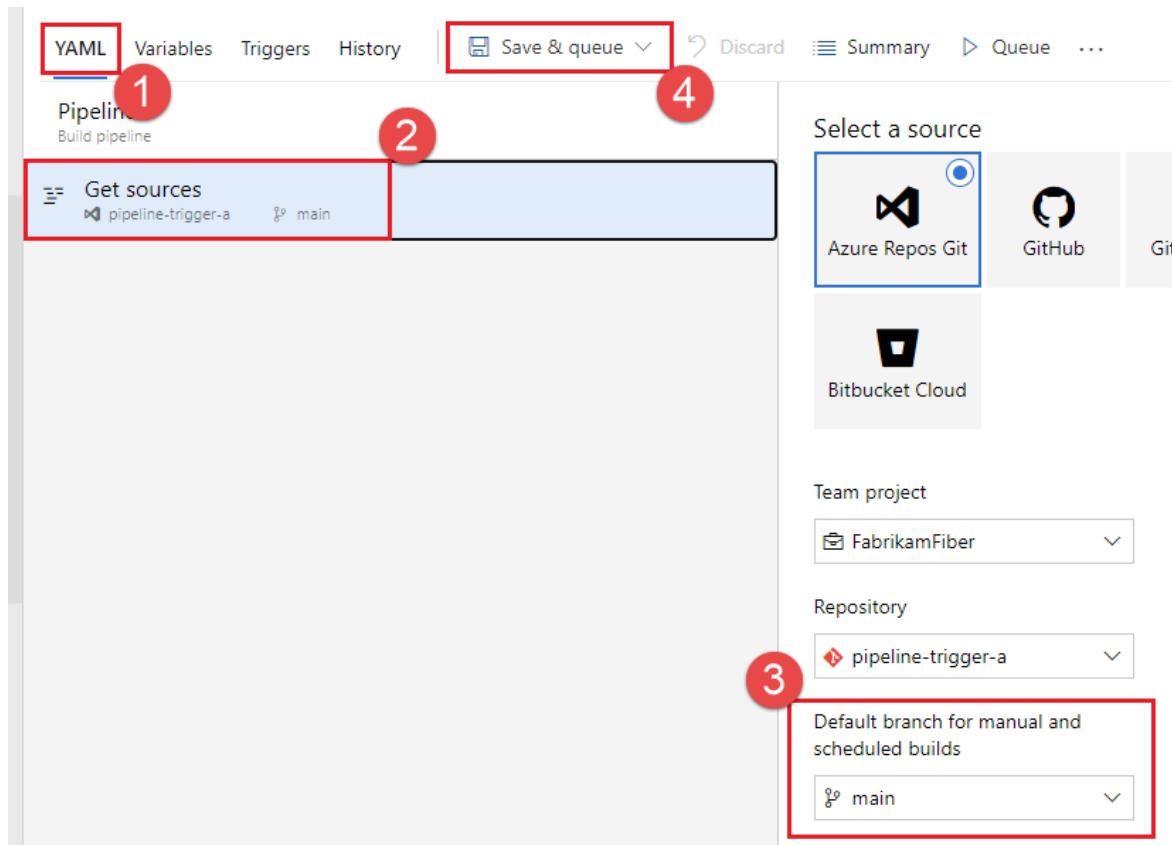
2. Choose ... and select **Triggers**.



This screenshot shows the 'Triggers' configuration dialog. It includes tabs for 'Variables', 'Run', and a three-dot menu icon (highlighted with a red box and labeled '1'). Below these are sections for 'Settings', 'Validate', and 'Download full YAML'. At the bottom, there's a list of triggers, with the 'Triggers' button highlighted with a red box and labeled '2'.

3. Select **YAML, Get sources**, and view the **Default branch for manual and scheduled builds** setting.

If you change it, choose **Save** or **Save & queue** to save the change.



Combining trigger types

When you specify both CI triggers and pipeline triggers in your pipeline, you can expect new runs to be started every time a push is made that matches the filters the CI trigger, and a run of the source pipeline is completed that matches the filters of the pipeline completion trigger.

For example, consider two pipelines named **A** and **B** that are in the same repository, both have CI triggers, and **B** has a pipeline completion trigger configured for the completion of pipeline **A**. If you make a push to the repository:

- A new run of **A** is started, based on its CI trigger.
- At the same time, a new run of **B** is started, based on its CI trigger. This run consumes the artifacts from a previous run of pipeline **A**.
- When **A** completes, it triggers another run of **B**, based on the pipeline completion trigger in **B**.

To prevent triggering two runs of **B** in this example, you must remove its CI trigger or pipeline trigger.

Trigger one pipeline after another (classic)

4/22/2021 • 2 minutes to read • [Edit Online](#)

Large products have several components that are dependent on each other. These components are often independently built. When an upstream component (a library, for example) changes, the downstream dependencies have to be rebuilt and revalidated.

In situations like these, add a pipeline trigger to run your pipeline upon the successful completion of the **triggering pipeline**.

Add a build completion trigger

In the classic editor, pipeline triggers are called **build completion triggers**. You can select any other build in the same project to be the triggering pipeline.

After you add a **build completion** trigger, select the **triggering build**. If the triggering build is sourced from a Git repo, you can also specify **branch filters**. If you want to use wildcard characters, then type the branch specification (for example, `features/modules/*`) and then press Enter.

NOTE

Keep in mind that in some cases, a single [multi-job build](#) could meet your needs. However, a build completion trigger is useful if your requirements include different configuration settings, options, or a different team to own the dependent pipeline.

Download artifacts from the triggering build

In many cases, you'll want to download artifacts from the triggering build. To do this:

1. Edit your build pipeline.
2. Add the **Download Build Artifacts** task to one of your jobs under **Tasks**.
3. For **Download artifacts produced by**, select **Specific build**.
4. Select the team **Project** that contains the triggering build pipeline.
5. Select the triggering **Build pipeline**.
6. Select **When appropriate, download artifacts from the triggering build**.
7. Even though you specified that you want to download artifacts from the triggering build, you must still select a value for **Build**. The option you choose here determines which build will be the source of the artifacts whenever your triggered build is run because of any other reason than `BuildCompletion` (e.g. `Manual`, `IndividualCI`, `Schedule`, and so on).
8. Specify the **Artifact name** and make sure it matches the name of the artifact published by the triggering build.
9. Specify the **Destination directory** to which you want to download the artifacts. For example:
`$(Build.BinariesDirectory)`

Release triggers

3/6/2021 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

NOTE

This topic covers classic release pipelines. To understand triggers in YAML pipelines, see [pipeline triggers](#).

Release triggers are an automation tool to deploy your application. When the trigger conditions are met, the pipeline will deploy your artifacts to the environment/stages you already specified.

Continuous deployment triggers

Continuous deployment triggers allow you to create a release every time a new build artifact is available. This feature is currently available only to build from Azure DevOps, TFS and Git-based repositories.

The screenshot shows the 'Continuous deployment trigger' configuration page. On the left, there's a sidebar with 'Artifacts' and a '+ Add' button. Below it is a card for 'DotNetSample-ASP.NET Core-CI' with a download icon and a '⚡' icon (which is highlighted with a red box). At the bottom of this sidebar is a 'Schedule not set' button. The main area is titled 'Continuous deployment trigger' and shows 'Build: DotNetSample-ASP.NET Core-CI'. A toggle switch is set to 'Enabled'. Below it, a description reads 'Creates a release every time a new build is available.' Under 'Build branch filters', there are three tabs: 'Type' (set to 'Build branch'), 'Build branch' (with a dropdown showing 'Include' and a note 'The build pipeline's default bra...'), and 'Build tags' (empty). A 'Branch filter' dropdown is open, showing 'The build pipeline's default branch'. There are also '+ Add' and 'Branch filter' buttons.

Build branch filters allow you to trigger a release only for a build that is from one of the branches selected here.

You also have the option to specify branch tags. If you do so, a release will be triggered only if a new build tagged with the keywords specified here, is available.

NOTE

Automatically creating a release does not mean it will be automatically deployed to a stage. You must set up stages triggers to deploy your app to the various stages.

Scheduled release triggers

Scheduled release trigger allow you to create new releases at specific times.

Select the schedule icon under the **Artifacts** section. Toggle the Enabled/Disabled button and specify your release schedule. You can set up multiple schedules to trigger a release.

The screenshot shows the 'FabrikamFiber' pipeline configuration. On the left, there's an 'Artifacts' section containing a 'FabrikamCode' artifact and a 'Schedule set' item, which is highlighted with a red box. On the right, the 'Scheduled release trigger' configuration is displayed. It includes an 'Enabled' toggle switch (which is turned on), a time range selector ('Mon through Fri at 3:00'), and a date range selector ('03h 00m UTC Coordinated Universal Time'). A red box highlights the '+ Add a new time' button.

Pull request triggers

If you chose to enable the pull-request triggers, a release will be created every time a selected artifact is available as part of a pull request workflow.

The screenshot shows the 'Pull request trigger' configuration for the 'DotNetSample-ASP.NET Core-CI' build. It features an 'Enabled' toggle switch (turned on) and a description explaining it creates a release every time a new version of the selected artifact is available as part of a pull request workflow. Below this are sections for 'Target Branch Filters' (set to 'master') and 'Stages'. A yellow callout box at the bottom states '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.'

To use a pull request trigger, you must also enable it for specific stages. We will go through stage triggers in the next section. You may also want to set up a [branch policies](#) for your branches.

Stage triggers

Stage triggers allow you set up specific conditions to trigger deployment to a specific stage.

Pre-deployment conditions

Production

Triggers ^

Define the trigger that will start deployment to this stage

Production 1 job, 1 task

Select trigger ⓘ

After release

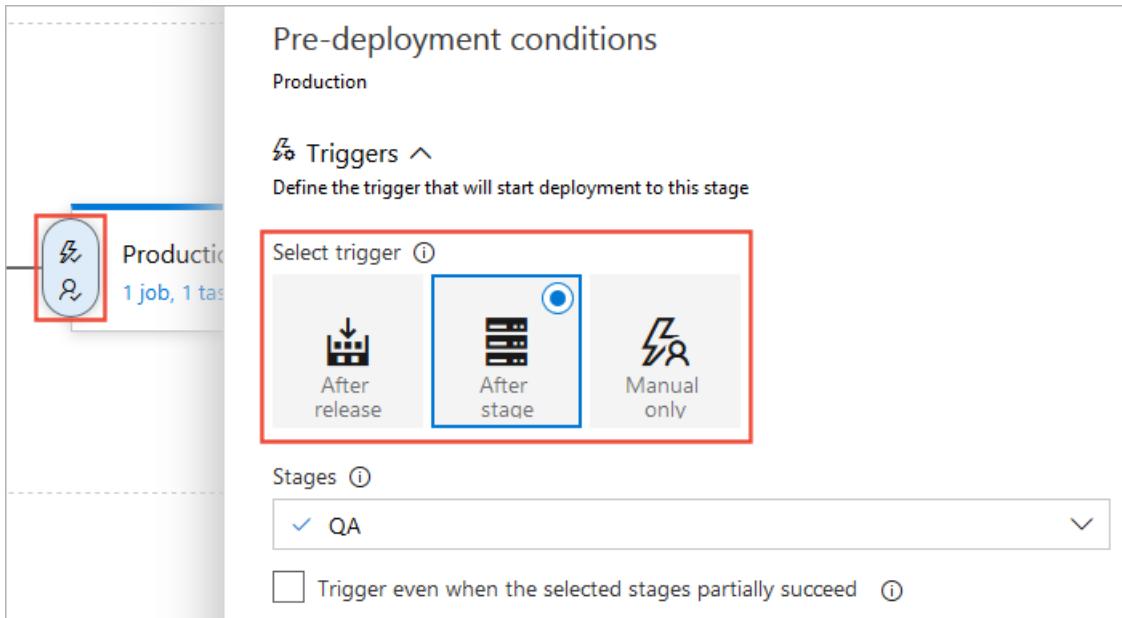
After stage

Manual only

Stages ⓘ

QA

Trigger even when the selected stages partially succeed ⓘ



- **Select trigger:** Set the trigger that will start the deployment to this stage automatically. Select "Release" to deploy to the stage every time a new release is created. Use the "Stage" option to deploy after deployments to selected stages are successful. To allow only manual deployments, select "Manual".
- **Artifacts filter:** Select artifact condition(s) to trigger a new deployment. A release will be deployed to this stage only if all artifact conditions are met.

Stages | + Add ▾

QA 2 jobs, 2 tasks

Triggers ^

Define the trigger that will start deployment to this stage

Select trigger ⓘ

After release

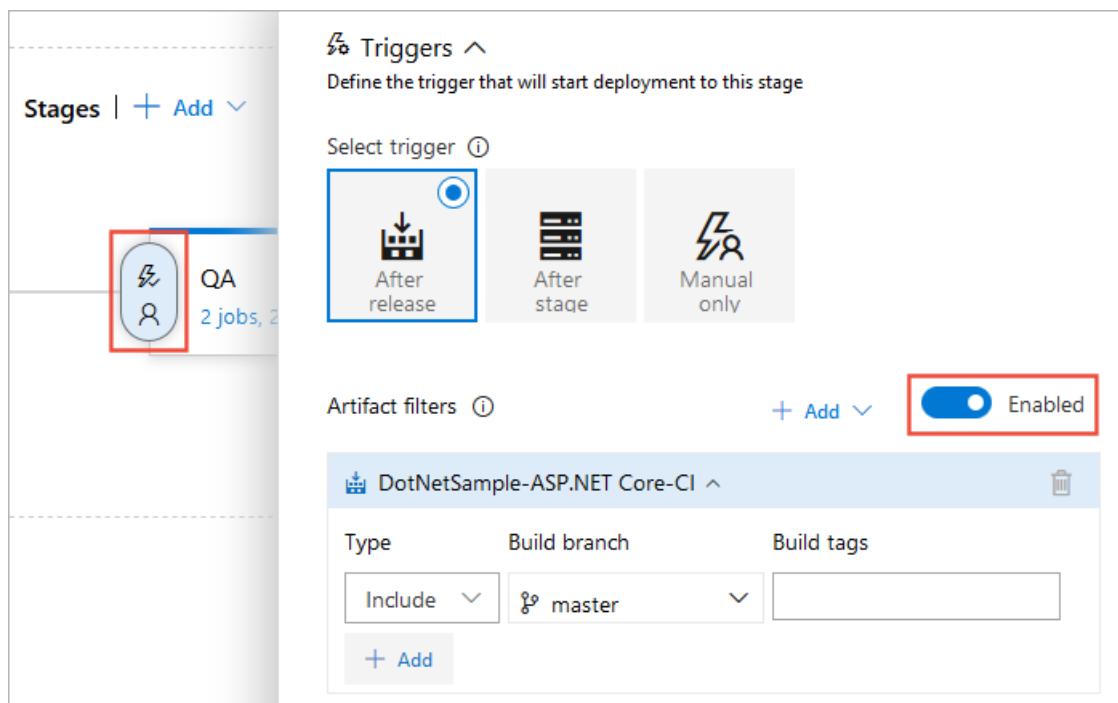
After stage

Manual only

Artifact filters ⓘ

+ Add ▾ Enabled

Type	Build branch	Build tags
Include	master	
+ Add		



- **Schedule:** Trigger a new deployment to this stage at a specific time.

The screenshot shows the 'Triggers' configuration for a pipeline stage. A red box highlights the 'After release' trigger option under 'Select trigger'. The 'QA' stage is selected in the dropdown. The 'Schedule' toggle is set to 'Enabled', indicated by a red box around the switch. The schedule details show it runs from Monday through Friday at 3:00 UTC.

Triggers ^
Define the trigger that will start deployment to this stage

Select trigger ⓘ

After release

After stage

Manual only

Stages ⓘ

QA

Trigger even when the selected stages partially succeed ⓘ

Artifact filters ⓘ

Disabled

Schedule ⓘ

Enabled

Mon through Fri at 3:00

Mon Tue Wed Thu Fri Sat

Sun

03h 00m

(UTC) Coordinated Universal Time

- **Pull-request deployment:** Enabling this will allow pull request based releases to be deployed to this stage. Keep it disabled if this is a critical or production stage.

The screenshot shows the 'Stages' configuration for a pipeline. A red box highlights the 'QA' stage. The 'Pre-deployment conditions' section shows 'QA'. The 'Triggers' configuration is identical to the previous screenshot, with 'After release' selected. The 'Pull request deployment' toggle is set to 'Enabled', indicated by a red box around the switch.

All pipelines > SampleApp - 1

Save

Pipeline Tasks Variables Retention Options History

Stages | + Add ⌂

QA
1 job, 1 task

Pre-deployment conditions

QA

Triggers ^
Define the trigger that will start deployment to this stage

Select trigger ⓘ

After release

After stage

Manual only

Artifact filters ⓘ

Disabled

Schedule ⓘ

Disabled

Pull request deployment ⓘ

Enabled

- **Pre-deployment approvals:** Select the users who can approve or reject deployments to this stage. By default, all users must approve the deployment. If a group is added, one user in the group must approve the deployment. You can also specify the timeout (the maximum time that an approval is allowed to be pending before it is automatically rejected) and approval policies.

🔗 Pre-deployment approvals ▲

 Enabled

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

Approvers 

Search users and groups for approvers

 Enter at least one approver.

Timeout 

30

Days 

Approval policies

- 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 

- **Gates:** Allow you to set up specific gates to evaluate before the deployment.

→] Gates ▲

 Enabled

Define gates to evaluate before the deployment.

[Learn more](#)

The delay before evaluation 

5

Minutes 

Deployment gates 

 Add 

- **Deployment queue settings:** Allow you to configure actions when multiple releases are queued for deployment.

▫ Deployment queue settings ^

Define behavior when multiple releases are queued for deployment

Number of parallel deployments (i)

Specific Unlimited

Maximum number of parallel deployments

1

Subsequent releases

Deploy all in sequence
 Deploy latest and cancel the others

NOTE

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.

Help and support

- See our [troubleshooting](#) page
- Get advice on [Stack Overflow](#), and feel free to post your questions, search for answers, or suggest a feature on our [Azure DevOps Developer Community](#). [Support](#) page.

Task types & usage

3/6/2021 • 9 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

A **task** is the building block for defining automation in a 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 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 run a [job](#), all the tasks are run in sequence, one after the other. To run the same set of tasks in parallel on multiple agents, or to run some tasks without using an agent, see [jobs](#).

By default, all tasks run in the same context, whether that's on the [host](#) or in a [job container](#). You may optionally use [step targets](#) to control context for an individual task.

Learn more about how to specify properties for a task with the [YAML schema](#).

When you run a [job](#), 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 the task GUID or a fully-qualified name for the custom task to avoid this risk:

```
steps:  
- task: myPublisherId.myExtensionId.myContributionId.myTaskName@1 #format example  
- task: qetza.replacetokens.replacetokens-task.replacetokens@3 #working example
```

To find `myPublisherId` and `myExtensionId`, select **Get** on a task in the marketplace. The values after the `itemName` in your URL string are `myPublisherId` and `myExtensionId`. You can also find the fully-qualified name by adding the task to a [Release pipeline](#) and selecting **View YAML** when editing the task.

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.

You can set which minor version gets used by specifying the full version number of a task after the `@` sign (example: `GoTool@0.3.1`). You can only use task versions that exist for your organization.

- [YAML](#)
- [Classic](#)

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
```

YAML pipelines aren't available in TFS.

Task control options

Each task offers you some **Control Options**.

- [YAML](#)
- [Classic](#)

Control options are available as keys on the `task` section.

```
- task: string # reference to a task and version, e.g. "VSBuild@1"  
  condition: expression # see below  
  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 # how long to wait before timing out the task  
  target: string # 'host' or the name of a container resource to target
```

The timeout period begins when the task starts running. It does not include the time the task is queued or is waiting for an agent.

In this YAML, `PublishTestResults@2` will run even if the previous step fails because of the [succeededOrFailed\(\)](#) condition.

```
steps:  
- task: UsePythonVersion@0  
  inputs:  
    versionSpec: '3.7'  
    architecture: 'x64'  
- task: PublishTestResults@2  
  inputs:  
    testResultsFiles: "**/TEST-*.xml"  
  condition: succeededOrFailed()
```

NOTE

For the full schema, see [YAML schema for task](#).

Conditions

- Only when all previous dependencies have succeeded. This is the default if there is not a condition set in the YAML.
- Even if a previous dependency has failed, unless the run was canceled. Use `succeededOrFailed()` in the YAML for this condition.
- Even if a previous dependency has failed, even if the run was canceled. Use `always()` in the YAML for this condition.
- Only when a previous dependency has failed. Use `failed()` in the YAML for this condition.
- [Custom conditions](#) which are composed of expressions

Step target

Tasks run in an execution context, which is either the agent host or a container. An individual step may override its context by specifying a `target`. Available options are the word `host` to target the agent host plus any containers defined in the pipeline. For example:

```
resources:  
  containers:  
    - container: pycontainer  
      image: python:3.8  
  
  steps:  
    - task: SampleTask@1  
      target: host  
    - task: AnotherTask@1  
      target: pycontainer
```

Here, the `SampleTask` runs on the host and `AnotherTask` runs in a container.

YAML pipelines aren't available in TFS.

Build tool installers (Azure Pipelines)

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

- [YAML](#)
- [Classic](#)

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 Nodejs version if it is not already on the agent. The [Command Line](#) script logs the location of the Nodejs version on disk.

YAML pipelines aren't available in TFS.

Tool installer tasks

For a list of our tool installer tasks, see [Tool installer tasks](#).

Disabling in-box and Marketplace tasks

On the organization settings page, you can disable Marketplace tasks, in-box tasks, or both. Disabling Marketplace tasks can help [increase security](#) of your pipelines. If you disable both in-box and Marketplace tasks, only tasks you install using [`tfx`](#) will be available.

Related articles

- [Jobs](#)
- [Task groups](#)
- [Built-in task catalog](#)

Help and support

- See our [troubleshooting](#) page
- Get advice on [Stack Overflow](#), and feel free to post your questions, search for answers, or suggest a feature on our [Azure DevOps Developer Community](#). [Support](#) page.

Task groups for builds and releases

2/26/2020 • 8 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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

- 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.

All pipelines > SampleApp - 1

Pipeline Tasks Variables Retention Options History

Production Deployment process

Run on agent +

Deploy Azure App Service
Azure App Service Deploy

Cloud Load Test mytestfi...
Cloud-based Load Test

Stage name: Production

Parameters: **Unlink all**

Azure subscription: Visual Studio Enterprise

App type

- 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**.

All pipelines > SampleApp - 1

Pipeline Tasks Variables Retention Options History

Production Deployment process

Run on agent +

Deploy Azure App Service
Azure App Service Deploy

Cloud Load Test mytestfi...
Cloud-based Load Test

Cloud-based Load Test

Version: 1.*

Display name: Cloud Load Test mytestfile

VS Team Services Connection

Load test files folder: \$(System.DefaultWorkingDirectory)

Load test file: mytestfile

Active Run Settings: As specified in the load test file

Enable selected task(s) (checked)

Disable selected task(s)

Remove selected task(s)

Clone task(s)

+ Create task group (highlighted with a red box)

Manage task group

- 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.
- After you choose **Create**, the new task group is created and replaces the selected tasks in your pipeline.
- All the '\$(vars)' from the underlying tasks, excluding the **predefined variables**, will surface as the

mandatory parameters for the newly created task group.

For example, let's say you have a task input \$(foobar), which you don't intend to parameterize. However, when you create a task group, the task input is converted into task group parameter 'foobar'. Now, you can provide the default value for the task group parameter 'foobar' as \$(foobar). This ensures that at runtime, the expanded task gets the same input it's intended to.

6. 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 item, with the '...' option highlighted by a red box. The menu contains three items: 'Delete', 'Export', and 'Security'. The 'Export' option is currently selected.

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. The left sidebar lists the tasks included in the group: 'Deploy Azure App Service' and 'Quick Web Performance...'. The main panel displays the task group's properties. The 'Name' field is set to 'Standard deployment tasks'. The 'Description' field contains the text 'Fabrikam standard set of tasks for deployment'. The 'Category' field is set to '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.

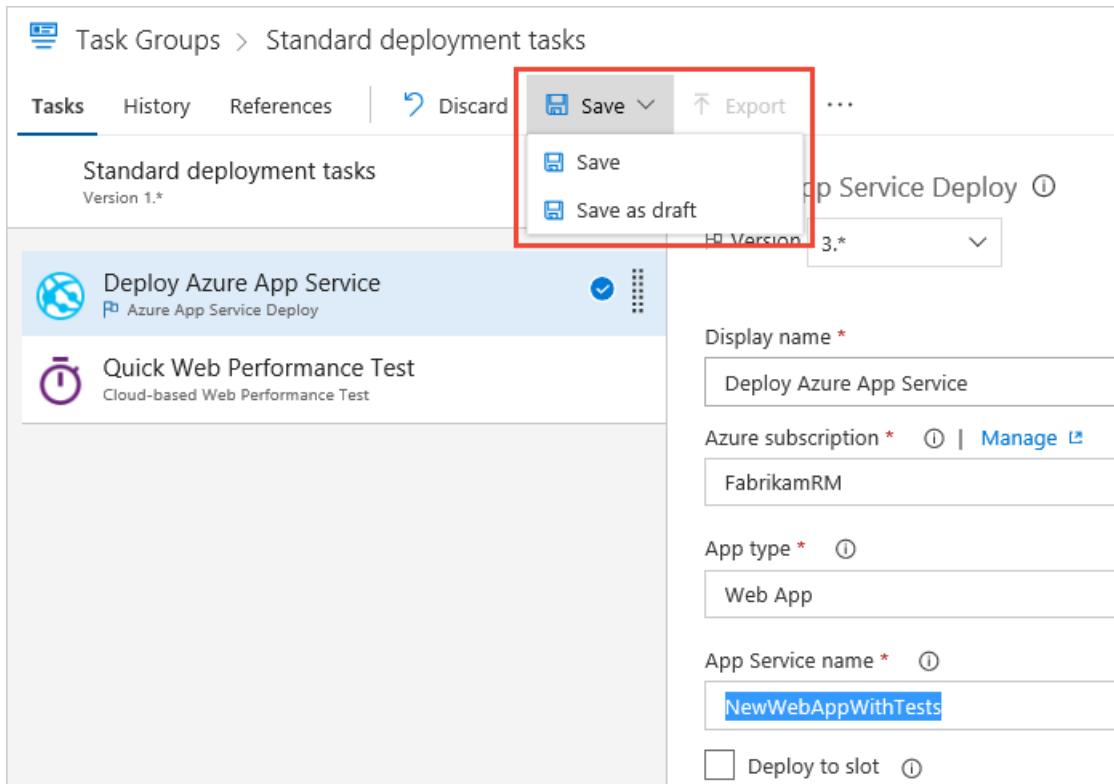
All the variable parameters of the task group will show up as mandatory parameters in the pipeline definition. You can also set the default value for the task group parameters.

- 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

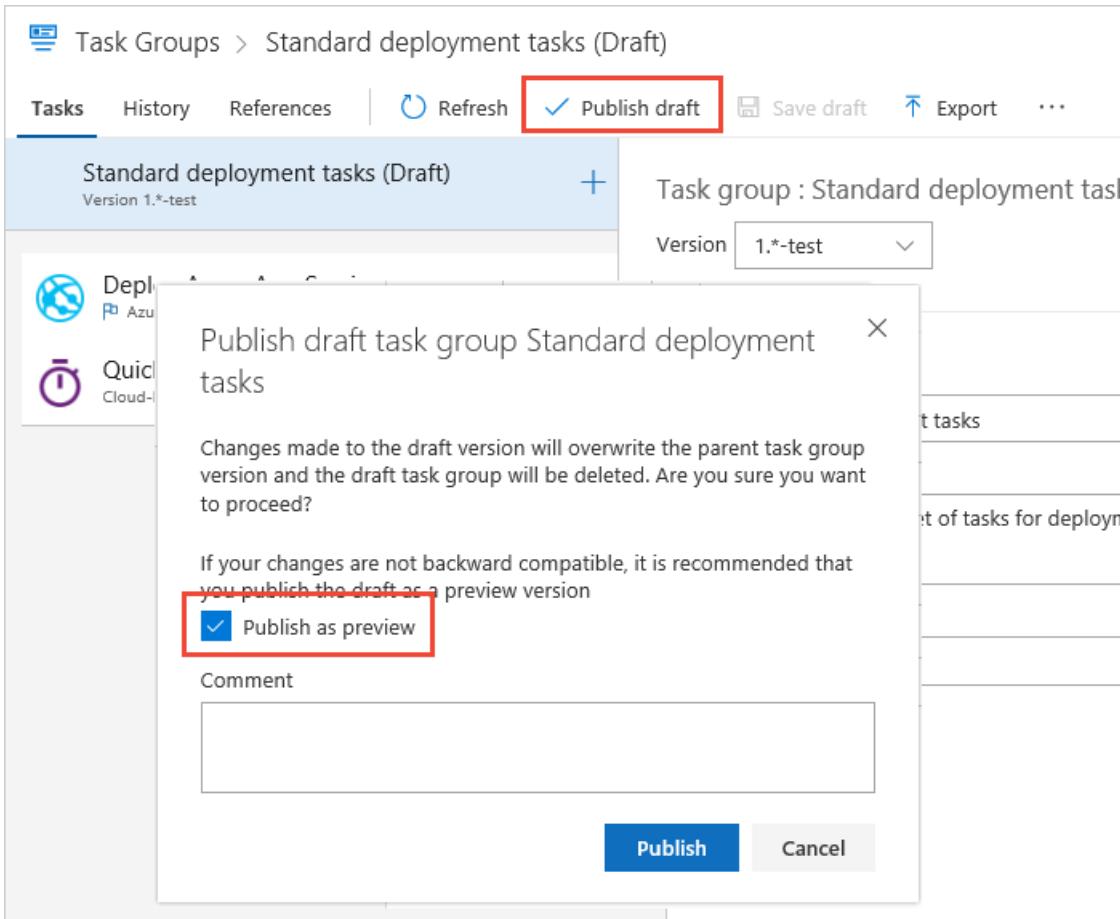
All of the built-in tasks in Azure Pipelines and TFS are [versioned](#). This allows build and release pipelines to continue to use the existing version of a task while new versions are developed, tested, and released. In Azure Pipelines, you can version your own custom task groups so that they behave 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 a 'Standard deployment tasks' group. The 'Save' dropdown menu is open, highlighting the 'Save as draft' option. The group has a version of '1.*'. Below the group, there are two tasks: 'Deploy Azure App Service' and 'Quick Web Performance Test'. On the right, there are configuration fields for the 'Deploy Azure App Service' task, including 'Display name' (set to 'Deploy Azure App Service'), 'Azure subscription' (set to 'FabrikamRM'), 'App type' (set to 'Web App'), 'App Service name' (set to 'NewWebAppWithTests'), and a checked 'Deploy to slot' checkbox.

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.



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 'All definitions > Fabrikam' pipeline configuration page. The 'Tasks' tab is selected. A 'Task group: Standard de...' item is selected, showing its details. The 'Version' dropdown is set to '1.*' and has a red box around it. A sub-menu is open, showing '2.* (preview)' highlighted with a red box. Other options in the menu include '1.*' and '3.* (preview)'. The 'Display name' is '1.*'.

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.

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.

Working with taskgroup versions

Any taskgroup update can be a minor or major version update.

Minor version

Action: You directly save the task group after edit instead of saving it as draft.

Effect: The version number doesn't change. Let's say you have a taskgroup of version `1.0`. You can have any number of minor version updates i.e. `1.1`, `1.2`, `1.3` etc. In your pipeline, the taskgroup version shows as `1.*`. The latest changes will show up in the pipeline definition automatically.

Reason: This is supposed to be a small change in the task group and you expect the pipelines to use this new change without editing the version in the pipeline definition.

Major version

Action: You save the task group as draft and then create a preview, validate the task group and then publish the preview as a major version.

Effect: The task group bumps up to a new version. Let's say you have a task group of version `1.*`. A new version gets published as `2.*`, `3.*`, `4.*` etc. And a notification about availability of new version shows up in all the pipeline definitions where this task group is used. User has to explicitly update to new version of the taskgroup in pipelines.

Reason: When you have a substantial change which might break the existing pipelines, you would like to test it out and roll out as a new version. Users can choose to upgrade to new version or choose to stay on the same version. This functionality is same as a normal task version update.

However, if your taskgroup update is not a breaking change but you would like to validate first and then enforce pipelines to consume the latest changes, you can follow below steps.

1. Update the task group with your desired changes and save it as a draft. A new draft task group '-Draft' will be created which contains the changes you have done. And this draft task group is accessible for you to consume in your pipelines.
2. Now, instead of publishing as preview, you can directly consume this draft task group in your test pipeline.
3. Validate this new draft task group in your test pipeline and once you are confident, go back to your main task group and do the same changes and save it directly. This will be taken as minor version update.
4. The new changes will now show up in all the pipelines where this task group is used.
5. Now you can delete your draft task group.

Related topics

- [Tasks](#)
- [Task jobs](#)

Help and support

- See our [troubleshooting](#) page
- Get advice on [Stack Overflow](#), and feel free to post your questions, search for answers, or suggest a feature on our [Azure DevOps Developer Community](#). [Support](#) page.

Template types & usage

6/8/2021 • 25 minutes to read • [Edit Online](#)

Templates let you define reusable content, logic, and parameters. Templates function in two ways. You can insert reusable content with a template or you can use a template to control what is allowed in a pipeline. The second approach is useful for [building secure pipelines with templates](#).

If a template is used to include content, it functions like an include directive in many programming languages. Content from one file is inserted into another file. When a template controls what is allowed in a pipeline, the template defines logic that another file must follow.

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.

Parameters

You can specify parameters and their data types in a template and pass those parameters to a pipeline. You can also [use parameters outside of templates](#).

Passing parameters

Parameters must contain a name and data type. In `azure-pipelines.yml`, when the parameter `yesNo` is set to a boolean value, the build succeeds. When `yesNo` is set to a string such as `apples`, the build fails.

```
# File: simple-param.yml
parameters:
- name: yesNo # name of the parameter; required
  type: boolean # data type of the parameter; required
  default: false

steps:
- script: echo ${{ parameters.yesNo }}
```

```
# File: azure-pipelines.yml
trigger:
- master

extends:
  template: simple-param.yml
  parameters:
    yesNo: false # set to a non-boolean value to have the build fail
```

Parameters to select a template at runtime

You can call different templates from a pipeline YAML depending on a condition. In this example, the `experimental.yml` YAML will run when the parameter `experimentalTemplate` is true.

```
#azure-pipeline.yml
parameters:
- name: experimentalTemplate
  displayName: 'Use experimental build process?'
  type: boolean
  default: false

steps:
- ${{ if eq(parameters.experimentalTemplate, true) }}:
  - template: experimental.yml
- ${{ if not(eq(parameters.experimentalTemplate, true)) }}:
  - template: stable.yml
```

Parameter data types

DATA TYPE	NOTES
<code>string</code>	string
<code>number</code>	may be restricted to <code>values:</code> , otherwise any number-like string is accepted
<code>boolean</code>	<code>true</code> or <code>false</code>
<code>object</code>	any YAML structure
<code>step</code>	a single step
<code>stepList</code>	sequence of steps
<code>job</code>	a single job
<code>jobList</code>	sequence of jobs
<code>deployment</code>	a single deployment job
<code>deploymentList</code>	sequence of deployment jobs
<code>stage</code>	a single stage
<code>stageList</code>	sequence of stages

The `step`, `stepList`, `job`, `jobList`, `deployment`, `deploymentList`, `stage`, and `stageList` data types all use standard YAML schema format. This example includes `string`, `number`, `boolean`, `object`, `step`, and `stepList`.

```

parameters:
- name: myString
  type: string
  default: a string
- name: myMultiString
  type: string
  default: default
  values:
    - default
    - ubuntu
- name: myNumber
  type: number
  default: 2
  values:
    - 1
    - 2
    - 4
    - 8
    - 16
- name: myBoolean
  type: boolean
  default: true
- name: myObject
  type: object
  default:
    foo: FOO
    bar: BAR
    things:
      - one
      - two
      - three
    nested:
      one: apple
      two: pear
      count: 3
- name: myStep
  type: step
  default:
    script: echo my step
- name: mySteplist
  type: stepList
  default:
    - script: echo step one
    - script: echo step two

trigger: none

jobs:
- job: stepList
  steps: ${{ parameters.mySteplist }}
- job: myStep
  steps:
    - ${{ parameters.myStep }}

```

You can iterate through an object and print out each string in the object.

```

parameters:
- name: listOfStrings
  type: object
  default:
  - one
  - two

steps:
- ${{ each value in parameters.listOfStrings }}:
  - script: echo ${{ value }}

```

Extend from a template

To increase security, you can enforce that a pipeline extends from a particular template. The file `start.yml` defines the parameter `buildSteps`, which is then used in the pipeline `azure-pipelines.yml`. In `start.yml`, if a `buildStep` gets passed with a script step, then it is rejected and the pipeline build fails. When extending from a template, you can increase security by adding a [required template approval](#).

```

# File: start.yml
parameters:
- name: buildSteps # the name of the parameter is buildSteps
  type: stepList # data type is StepList
  default: [] # default value of buildSteps
stages:
- stage: secure_buildstage
  pool: Hosted VS2017
  jobs:
  - job: secure_buildjob
    steps:
    - script: echo This happens before code
      displayName: 'Base: Pre-build'
    - script: echo Building
      displayName: 'Base: Build'

    - ${{ each step in parameters.buildSteps }}:
      - ${{ each pair in step }}:
        ${{ if ne(pair.value, 'CmdLine@2') }}:
          ${{ pair.key }}: ${{ pair.value }}
        ${{ if eq(pair.value, 'CmdLine@2') }}:
          '${{ pair.value }}': error

    - script: echo This happens after code
      displayName: 'Base: Signing'

```

```

# File: azure-pipelines.yml
trigger:
- master

extends:
  template: start.yml
parameters:
  buildSteps:
  - bash: echo Test #Passes
    displayName: succeed
  - bash: echo "Test"
    displayName: succeed
  - task: CmdLine@2
    displayName: Test 3 - Will Fail
    inputs:
      script: echo "Script Test"

```

Extend from a template with resources

You can also use `extends` to extend from a template in your Azure pipeline that contains resources.

```
# File: azure-pipelines.yml
trigger:
- none

extends:
  template: resource-template.yml
```

```
# File: resource-template.yml
resources:
  pipelines:
    - pipeline: my-pipeline
      source: sourcePipeline

  steps:
    - script: echo "Testing resource template"
```

Insert a template

You can copy content from one YAML and reuse it in a different YAML. This saves you from having to manually include the same logic in multiple places. The `include-npm-steps.yml` file template contains steps that are reused in `azure-pipelines.yml`.

```
# File: templates/include-npm-steps.yml

steps:
- script: npm install
- script: yarn install
- script: npm run compile
```

```
# File: azure-pipelines.yml

jobs:
- job: Linux
  pool:
    vmImage: 'ubuntu-latest'
  steps:
    - template: templates/include-npm-steps.yml # Template reference
- job: Windows
  pool:
    vmImage: 'windows-latest'
  steps:
    - template: templates/include-npm-steps.yml # Template reference
```

Step reuse

You can insert a template to 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-10.14'
  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 with templates.

```

# File: templates/jobs.yml
jobs:
- job: Ubuntu
  pool:
    vmImage: 'ubuntu-latest'
  steps:
    - bash: echo "Hello Ubuntu"

- job: Windows
  pool:
    vmImage: 'windows-latest'
  steps:
    - bash: echo "Hello Windows"

```

```

# File: azure-pipelines.yml

jobs:
- template: templates/jobs.yml # Template reference

```

When working with multiple jobs, remember to remove the name of the job in the template file, so as to avoid conflict

```
# File: templates/jobs.yml
jobs:
- job:
  pool:
    vmImage: 'ubuntu-latest'
  steps:
  - bash: echo "Hello Ubuntu"

- job: Windows
  pool:
    vmImage: 'windows-latest'
  steps:
  - bash: echo "Hello Windows"
```

```
# File: azure-pipelines.yml

jobs:
- template: templates/jobs.yml # Template reference
- template: templates/jobs.yml # Template reference
- template: templates/jobs.yml # Template reference
```

Stage reuse

Stages can also be reused with templates.

```
# File: templates/stages1.yml
stages:
- stage: Angular
  jobs:
  - job: angularinstall
    steps:
    - script: npm install angular
```

```
# File: templates/stages2.yml
stages:
- stage: Build
  jobs:
  - job: build
    steps:
    - script: npm run build
```

```
# File: azure-pipelines.yml
trigger:
- master

pool:
  vmImage: 'ubuntu-latest'

stages:
- stage: Install
  jobs:
  - job: npminstall
    steps:
    - task: Npm@1
      inputs:
        command: 'install'
- template: templates/stages1.yml
- template: templates/stages2.yml
```

Job, stage, and step templates with parameters

```

# File: templates/npm-with-params.yml

parameters:
- name: name # defaults for any parameters that aren't specified
  default: ''
- name: vmImage
  default: ''

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-10.14'

- template: templates/npm-with-params.yml # Template reference
  parameters:
    name: Windows
    vmImage: 'vs2017-win2016'

```

You can also use parameters with step or stage templates. For example, steps with parameters:

```

# File: templates/steps-with-params.yml

parameters:
- name: 'runExtendedTests' # defaults for any parameters that aren't specified
  type: boolean
  default: false

steps:
- script: npm test
- ${{ if eq(parameters.runExtendedTests, true) }}:
  - script: npm test --extended

```

When you consume the template in your pipeline, specify values for the template parameters.

```

# File: azure-pipelines.yml

steps:
- script: npm install

- template: templates/steps-with-params.yml # Template reference
  parameters:
    runExtendedTests: 'true'

```

NOTE

Scalar parameters without a specified type are treated as strings. For example, `eq(true, parameters['myparam'])` will return `true`, even if the `myparam` parameter is the word `false`, if `myparam` is not explicitly made `boolean`. Non-empty strings are cast to `true` in a Boolean context. That `expression` could be rewritten to explicitly compare strings: `eq(parameters['myparam'], 'true')`.

Parameters are not limited to scalar strings. See the list of [data types](#). For example, using the `object` type:

```
# azure-pipelines.yml
jobs:
- template: process.yml
parameters:
  pool: # this parameter is called `pool`
  vmImage: ubuntu-latest # and it's a mapping rather than a string

# process.yml
parameters:
- name: 'pool'
  type: object
  default: {}

jobs:
- job: build
  pool: ${{ parameters.pool }}
```

Variable reuse

Variables can be defined in one YAML and included in another template. This could be useful if you want to store all of your variables in one file. If you are using a template to include variables in a pipeline, the included template can only be used to define variables. You can use steps and more complex logic when you are [extending from a template](#). Use [parameters](#) instead of variables when you want to restrict type.

In this example, the variable `favoriteVeggie` is included in `azure-pipelines.yml`.

```
# File: vars.yml
variables:
  favoriteVeggie: 'brussels sprouts'
```

```
# File: azure-pipelines.yml

variables:
- template: vars.yml # Template reference

steps:
- script: echo My favorite vegetable is ${{ variables.favoriteVeggie }}.
```

Reference template paths

Template paths should be relative to the file that does the including. Here is an example nested hierarchy.

```
|  
+-- fileA.yml  
|  
+-- dir1/  
|  
+-- fileB.yml  
|  
+-- dir2/  
|  
+-- fileC.yml
```

Then, in `fileA.yml` you can reference `fileB.yml` and `fileC.yml` like this.

```
steps:  
- template: dir1/fileB.yml  
- template: dir1/dir2/fileC.yml
```

If `fileC.yml` is your starting point, you can include `fileA.yml` and `fileB.yml` like this.

```
steps:  
- template: ../../fileA.yml  
- template: ../fileB.yml
```

When `fileB.yml` is your starting point, you can include `fileA.yml` and `fileC.yml` like this.

```
steps:  
- template: ./fileA.yml  
- template: dir2/fileC.yml
```

Use 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:  
- name: 'vmImage'  
  default: 'ubuntu 16.04'  
  type: string  
  
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
      ref: refs/tags/v1.0 # optional ref to pin to

jobs:
- template: common.yml@templates # Template reference
  parameters:
    vmImage: 'vs2017-win2016'

```

For `type: github`, `name` is `<identity>/<repo>` as in the examples above. For `type: git` (Azure Repos), `name` is `<project>/<repo>`. If that project is in a separate Azure DevOps organization, you'll need to configure a [service connection](#) of type `Azure Repos/Team Foundation Server` with access to the project and include that in YAML:

```

resources:
  repositories:
    - repository: templates
      name: Contoso/BuildTemplates
      endpoint: myServiceConnection # Azure DevOps service connection
jobs:
- template: common.yml@templates

```

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.

If you want to use a particular, fixed version of the template, be sure to pin to a `ref`. The `refs` are either branches (`refs/heads/<name>`) or tags (`refs/tags/<name>`). If you want to pin a specific commit, first create a tag pointing to that commit, then pin to that tag.

NOTE

If no `ref` is specified, the pipeline will default to using `refs/heads/master`.

You may also use `@self` to refer to the repository where the main pipeline was found. This is convenient for use in `extends` templates if you want to refer back to contents in the extending pipeline's repository. For example:

```

# Repo: Contoso/Central
# File: template.yml
jobs:
- job: PreBuild
  steps: []

  # Template reference to the repo where this template was
  # included from - consumers of the template are expected
  # to provide a "BuildJobs.yml"
- template: BuildJobs.yml@self

- job: PostBuild
  steps: []

```

```

# Repo: Contoso/MyProduct
# File: azure-pipelines.yml
resources:
  repositories:
    - repository: templates
      type: git
      name: Contoso/Central

extends:
  template: template.yml@templates

```

```

# Repo: Contoso/MyProduct
# File: BuildJobs.yml
jobs:
- job: Build
  steps: []

```

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. Only predefined variables can be used in template expressions.

NOTE

Expressions are only expanded for `stages`, `jobs`, `steps`, and `containers` (inside `resources`). You cannot, for example, use an expression inside `trigger` or a resource like `repositories`. Additionally, on Azure DevOps 2020 RTW, you can't use template expressions inside `containers`.

For example, you define a template:

```

# File: steps/msbuild.yml

parameters:
- name: 'solution'
  default: '**/*.sln'
  type: string

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

```

Context

Within a template expression, you have access to the `parameters` context that contains the values of parameters passed in. Additionally, you have access to the `variables` context that contains all the variables specified in the YAML file plus many of the [predefined variables](#) (noted on each variable in that topic). Importantly, it doesn't have runtime variables such as those stored on the pipeline or given when you start a run. Template expansion happens [very early in the run](#), so those variables aren't available.

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:
- name: 'solution'
  default: ''
  type: string

steps:
- bash: |
  if [ -z "$SOLUTION" ]; then
    echo "##vso[task.logissue type=error;]Missing template parameter \"solution\""
    echo "##vso[task.complete result=Failed;]"
  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 show 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:
- name: 'restoreProjects'
  default: ''
  type: string
- name: 'buildProjects'
  default: ''
  type: string

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:
- name: 'preBuild'
  type: stepList
  default: []
- name: 'preTest'
  type: stepList
  default: []
- name: 'preSign'
  type: stepList
  default: []

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:
- name: 'additionalVariables'
  type: object
  default: {}

jobs:
- job: build
  variables:
    configuration: debug
    arch: x86
    ${{ insert }}: ${{ parameters.additionalVariables }}
  steps:
- task: msbuild@1
- task: vstest@2

```

```

jobs:
- template: jobs/build.yml
  parameters:
    additionalVariables:
      TEST_SUITE: L0,L1

```

Conditional insertion

If you want to conditionally insert into a sequence or a mapping in a template, use insertions and expression evaluation. You can also use `if` statements [outside of templates](#) as long as you use template syntax.

For example, to insert into a sequence in a template:

```
# File: steps/build.yml

parameters:
- name: 'toolset'
  default: msbuild
  type: string
  values:
  - msbuild
  - dotnet

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 in a template:

```
# File: steps/build.yml

parameters:
- name: 'debug'
  type: boolean
  default: 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
```

You can also use conditional insertion for variables. In this example, `start` always prints and `this is a test`

only prints when the `foo` variable equals `test`.

```
variables:
- name: foo
  value: test

pool:
  vmImage: 'ubuntu-latest'

steps:
- script: echo "start" # always runs
- ${{ if eq(variables.foo, 'test') }}:
  - script: echo "this is a test" # runs when foo=test
```

Iterative insertion

The `each` directive allows iterative insertion based on a YAML sequence (array) or mapping (key-value pairs).

For example, you can wrap the steps of each job with additional pre- and post-steps:

```
# job.yml
parameters:
- name: 'jobs'
  type: jobList
  default: []

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()
```

```
# azure-pipelines.yml
jobs:
- template: job.yml
parameters:
  jobs:
  - job: A
    steps:
    - script: echo This will get sandwiched between SetupMyBuildTools and PublishMyTelemetry.
  - job: B
    steps:
    - script: echo So will this!
```

You can also manipulate the properties of whatever you're iterating over. For example, to add additional dependencies:

```

# job.yml
parameters:
- name: 'jobs'
  type: jobList
  default: []

jobs:
- job: SomeSpecialTool           # Run your special tool in its own job first
  steps:
  - task: RunSpecialTool@1
- ${{ each job in parameters.jobs }}: # Then do each job
  - ${{ each pair in job }}:          # Insert all properties other than "dependsOn"
    ${{ if ne(pair.key, 'dependsOn') }}:
      ${{ pair.key }}: ${{ pair.value }}
  dependsOn:                      # Inject dependency
  - SomeSpecialTool
- ${{ if job.dependsOn }}:
  - ${{ job.dependsOn }}

```

```

# azure-pipelines.yml
jobs:
- template: job.yml
  parameters:
    jobs:
    - job: A
      steps:
      - script: echo This job depends on SomeSpecialTool, even though it's not explicitly shown here.
    - job: B
      dependsOn:
      - A
      steps:
      - script: echo This job depends on both Job A and on SomeSpecialTool.

```

Escape a value

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}}' }}`

Imposed limits

Templates and template expressions can cause explosive growth to the size and complexity of a pipeline. To help prevent runaway growth, Azure Pipelines imposes the following limits:

- No more than 100 separate YAML files may be included (directly or indirectly)
- No more than 20 levels of template nesting (templates including other templates)
- No more than 10 megabytes of memory consumed while parsing the YAML (in practice, this is typically between 600KB - 2MB of on-disk YAML, depending on the specific features used)

Template parameters

You can pass parameters to 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. As a result, only [predefined variables](#) can be used in parameters.

To use parameters across multiple pipelines, see how to create a [variable group](#).

Job, stage, and step templates with parameters

```

# 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'

```

You can also use parameters with step or stage templates. For example, steps with parameters:

```

# File: templates/steps-with-params.yml

parameters:
  runExtendedTests: 'false' # defaults for any parameters that aren't specified

steps:
- script: npm test
- ${{ if eq(parameters.runExtendedTests, 'true') }}:
  - script: npm test --extended

```

When you consume the template in your pipeline, specify values for the template parameters.

```

# File: azure-pipelines.yml

steps:
- script: npm install

- template: templates/steps-with-params.yml # Template reference
  parameters:
    runExtendedTests: 'true'

```

NOTE

Scalar parameters are always treated as strings. For example, `eq(parameters['myparam'], true)` will almost always return `true`, even if the `myparam` parameter is the word `false`. Non-empty strings are cast to `true` in a Boolean context. That `expression` could be rewritten to explicitly compare strings: `eq(parameters['myparam'], 'true')`.

Parameters are not limited to scalar strings. As long as the place where the parameter expands expects a mapping, the parameter can be a mapping. Likewise, sequences can be passed where sequences are expected. For example:

```
# azure-pipelines.yml
jobs:
- template: process.yml
parameters:
  pool: # this parameter is called `pool`
  vmImage: ubuntu-latest # and it's a mapping rather than a string

# process.yml
parameters:
  pool: {}

jobs:
- job: build
  pool: ${{ parameters.pool }}
```

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
      ref: refs/tags/v1.0 # optional ref to pin to

jobs:
- template: common.yml@templates # Template reference
  parameters:
    vmImage: 'vs2017-win2016'

```

For `type: github`, `name` is `<identity>/<repo>` as in the examples above. For `type: git` (Azure Repos), `name` is `<project>/<repo>`. The project must be in the same organization; cross-organization references are not supported.

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.

If you want to use a particular, fixed version of the template, be sure to pin to a ref. Refs are either branches (`refs/heads/<name>`) or tags (`refs/tags/<name>`). If you want to pin a specific commit, first create a tag pointing to that commit, then pin to that tag.

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

```

Context

Within a template expression, you have access to the `parameters` context which contains the values of parameters passed in. Additionally, you have access to the `variables` context which contains all the variables specified in the YAML file plus the [system variables](#). Importantly, it doesn't have runtime variables such as those stored on the pipeline or given when you start a run. Template expansion happens [very early in the run](#), so those variables aren't available.

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.logissue type=error;]Missing template parameter \"solution\""
    echo "#vso[task.complete result=Failed;]"
  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 show 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:
  additionalVariables: {}

jobs:
- job: build
  variables:
    configuration: debug
    arch: x86
    ${{ insert }}: ${{ parameters.additionalVariables }}
  steps:
    - task: msbuild@1
    - task: vstest@2
```

```
jobs:
- template: jobs/build.yml
parameters:
  additionalVariables:
    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 YAML sequence (array) or mapping (key-value pairs).

For example, you can wrap the steps of each job with additional pre- and post-steps:

```
# job.yml
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()
```

```
# azure-pipelines.yml
jobs:
- template: job.yml
parameters:
  jobs:
    - job: A
      steps:
        - script: echo This will get sandwiched between SetupMyBuildTools and PublishMyTelemetry.
    - job: B
      steps:
        - script: echo So will this!
```

You can also manipulate the properties of whatever you're iterating over. For example, to add additional

dependencies:

```
# job.yml
parameters:
  jobs: []

jobs:
- job: SomeSpecialTool          # Run your special tool in its own job first
  steps:
    - task: RunSpecialTool@1
- ${{ each job in parameters.jobs }}: # Then do each job
  - ${{ each pair in job }}:        # Insert all properties other than "dependsOn"
    ${{ if ne(pair.key, 'dependsOn') }}:
      ${{ pair.key }}: ${{ pair.value }}
  dependsOn:                      # Inject dependency
    - SomeSpecialTool
- ${{ if job.dependsOn }}:
  - ${{ job.dependsOn }}
```

```
# azure-pipelines.yml
jobs:
- template: job.yml
parameters:
  jobs:
    - job: A
      steps:
        - script: echo This job depends on SomeSpecialTool, even though it's not explicitly shown here.
    - job: B
      dependsOn:
        - A
      steps:
        - script: echo This job depends on both Job A and on SomeSpecialTool.
```

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}}' }}`

Limits

Templates and template expressions can cause explosive growth to the size and complexity of a pipeline. To help prevent runaway growth, Azure Pipelines imposes the following limits:

- No more than 50 separate YAML files may be included (directly or indirectly)
- No more than 10 megabytes of memory consumed while parsing the YAML (in practice, this is typically between 600KB - 2MB of on-disk YAML, depending on the specific features used)
- No more than 2000 characters per template expression are allowed

Add a custom pipelines task extension

6/30/2021 • 19 minutes to read • [Edit Online](#)

Azure DevOps Services | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2017

Learn how to install extensions to your organization for custom build or release tasks in Azure DevOps. These tasks appear next to Microsoft-provided tasks in the **Add Step** wizard.

The screenshot shows the 'Add tasks' interface with the 'Build' tab selected. The interface includes a 'Refresh' button and tabs for All, Build, Utility, Test, Package, Deploy, Tool, and Marketplace. A red box highlights the 'Fabrikam Test Build Task' entry, which is described as a 'Test build task from extension.'

Task	Description
.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.
Android Signing	Sign and align Android APK files
Ant	Build with Apache Ant
CMake	Build with the CMake cross-platform build system
Docker	Build, tag, push, or run Docker images, or run a Docker command. Task can be used with Docker or Azure Container registry.
Docker Compose	Build, push or run multi-container Docker applications. Task can be used with Docker or Azure Container registry.
Fabrikam Test Build Task	Test build task from extension.
Grunt	The JavaScript Task Runner

To learn more about the new cross-platform build/release system, see [What is Azure Pipelines?](#).

NOTE

This article covers agent tasks in agent-based extensions. For information on server tasks/server-based extensions, check out the [Server Task GitHub Documentation](#).

Prerequisites

To create extensions for Azure DevOps, you need the following software and tools:

- An organization in Azure DevOps. For more information, see [Create an organization](#).
- A text editor. For many of the tutorials, we use **Visual Studio Code**, which provides intellisense and debugging support. Go to code.visualstudio.com to download the latest version.
- The [latest version](#) of Nodejs. The production environment uses only **Node10** or Node6 (by using the

"Node" in the "execution" object instead of `Node10`).

- TypeScript Compiler 2.2.0 or greater, although we recommend version 4.0.2 or newer for tasks that use Node10. Go to [npmjs.com](#) to download the compiler.
- [Cross-platform CLI for Azure DevOps](#) to package your extensions. You can install `tfx-cli` by using `npm`, a component of Node.js, by running `npm i -g tfx-cli`.
- A `home` directory for your project. The `home` directory of a build or release task extension should look like the following example after you complete the steps in this tutorial:

```
|--- README.md  
|--- images  
|   |--- extension-icon.png  
|--- buildandreleasetask           // where your task scripts are placed  
|--- vss-extension.json            // extension's manifest
```

- The [vss-web-extension-sdk package installation](#).

IMPORTANT

The dev machine needs to run Node v10.21.0 to ensure that the code written is compatible with the production environment on the agent and the latest non-preview version of `azure-pipelines-task-lib`.

Develop in Unix vs. Windows

We did this walk-through on Windows with PowerShell. We attempted to make it generic for all platforms, but the syntax for getting environment variables is different.

If you're using a Mac or Linux, replace any instances of `$env:<var>=<val>` with `export <var>=<val>`.

Step 1: Create a custom task

Set up your task. Do every part of Step 1 within the `buildandreleasetask` folder.

Create task scaffolding

Create the folder structure for the task and install the required libraries and dependencies.

Create a directory and `package.json` file

From within your `buildandreleasetask` folder, run the following command:

```
npm init
```

`npm init` creates the `package.json` file. You can accept all of the default `npm init` options.

TIP

The agent doesn't automatically install the required modules because it's expecting your task folder to include the node modules. To mitigate this, copy the `node_modules` to `buildandreleasetask`. As your task gets bigger, it's easy to exceed the size limit (50MB) of a VSIX file. Before you copy the node folder, you may want to run

```
npm install --production or npm prune --production, or you can write a script to build and pack everything.
```

Add `azure-pipelines-task-lib`

We provide a library, `azure-pipelines-task-lib`, that should be used to create tasks. Add it to your library.

```
npm install azure-pipelines-task-lib --save
```

Add typings for external dependencies

Ensure that TypeScript typings are installed for external dependencies.

```
npm install @types/node --save-dev  
npm install @types/q --save-dev
```

Create a `.gitignore` file and add `node_modules` to it. Your build process should do an `npm install` and a `typings install` so that `node_modules` are built each time and don't need to be checked in.

```
echo node_modules > .gitignore
```

Choose typescript version

Tasks can use typescript versions 2.3.4 or 4.0.2. You can install the chosen typescript version using this command:

```
npm install typescript@4.0.2 --save-dev
```

If you skip this step, typescript version 2.3.4 gets used by default.

Create tsconfig.json compiler options

This file ensures that your TypeScript files are compiled to JavaScript files.

```
tsc --init
```

For example, we want to compile to the ES6 standard instead of ES5. To ensure the ES6 standard happens, open the newly generated `tsconfig.json` and update the `target` field to "es6."

NOTE

To have the command run successfully, make sure that TypeScript is installed globally with npm on your local machine.

Task implementation

Now that the scaffolding is complete, we can create our custom task.

task.json

Next, we create a `task.json` file in the `buildandreleasetask` folder. The `task.json` file describes the build or release task and is what the build/release system uses to render configuration options to the user and to know which scripts to execute at build/release time.

Copy the following code and replace the `{{placeholders}}` with your task's information. The most important placeholder is the `taskguid`, and it must be unique. You can generate the `taskguid` by using [Microsoft's online GuidGen tool](#).

```
{
    "$schema": "https://raw.githubusercontent.com/Microsoft/azure-pipelines-task-lib/master/tasks.schema.json",
    "id": "{{taskguid}}",
    "name": "{{taskname}}",
    "friendlyName": "{{taskfriendlyname}}",
    "description": "{{taskdescription}}",
    "helpMarkDown": "",
    "category": "Utility",
    "author": "{{taskauthor}}",
    "version": {
        "Major": 0,
        "Minor": 1,
        "Patch": 0
    },
    "instanceNameFormat": "Echo $(samplestring)",
    "inputs": [
        {
            "name": "samplestring",
            "type": "string",
            "label": "Sample String",
            "defaultValue": "",
            "required": true,
            "helpMarkDown": "A sample string"
        }
    ],
    "execution": {
        "Node10": {
            "target": "index.js"
        }
    }
}
```

task.json components Following are descriptions of some of the components of the `task.json` file:

PROPERTY	DESCRIPTION
<code>id</code>	A unique GUID for your task.
<code>name</code>	Name with no spaces.
<code>friendlyName</code>	Descriptive name (spaces allowed).
<code>description</code>	Detailed description of what your task does.
<code>author</code>	Short string describing the entity developing the build or release task, for example: "Microsoft Corporation."
<code>instanceNameFormat</code>	How the task is displayed within the build or release step list. You can use variable values by using <code>\$(variablename)</code> .
<code>groups</code>	Describes groups that task properties may be logically grouped by in the UI.
<code>inputs</code>	Inputs to be used when your build or release task runs. This task expects an input with the name <code>samplestring</code> .
<code>execution</code>	Execution options for this task, including scripts.

NOTE

For a more in-depth look into the task.json file, or to learn how to bundle multiple versions in your extension, check out the [build/release task reference](#).

index.ts

Create an `index.ts` file by using the following code as a reference. This code runs when the task is called.

```
import tl = require('azure-pipelines-task-lib/task');

async function run() {
    try {
        const inputString: string | undefined = tl.getInput('samplestring', true);
        if (inputString == 'bad') {
            tl.setResult(tl.TaskResult.Failed, 'Bad input was given');
            return;
        }
        console.log('Hello', inputString);
    }
    catch (err) {
        tl.setResult(tl.TaskResult.Failed, err.message);
    }
}

run();
```

Compile

Enter "tsc" from the buildandreleasetask folder to compile an `index.js` file from `index.ts`.

Run the task

An agent can run the task with `node index.js` from PowerShell.

In the following example, the task fails because inputs weren't supplied (`samplestring` is a required input).

```
node index.js
##vso[task.debug]agent.workFolder=undefined
##vso[task.debug]loading inputs and endpoints
##vso[task.debug]loaded 0
##vso[task.debug]task result: Failed
##vso[task.issue type=error;]Input required: samplestring
##vso[task.complete result=Failed;]Input required: samplestring
```

As a fix, we can set the `samplestring` input and run the task again.

```
$env:INPUT_SAMPLESTRING="Human"
node index.js
##vso[task.debug]agent.workFolder=undefined
##vso[task.debug]loading inputs and endpoints
##vso[task.debug]loading INPUT_SAMPLESTRING
##vso[task.debug]loaded 1
##vso[task.debug]Agent.ProxyUrl=undefined
##vso[task.debug]Agent.CAInfo=undefined
##vso[task.debug]Agent.ClientCert=undefined
##vso[task.debug]Agent.SkipCertValidation=undefined
##vso[task.debug]samplestring=Human
Hello Human
```

This time, the task succeeded because `samplestring` was supplied, and it correctly outputted "Hello Human"!

Step 2: Unit test your task scripts

The goal of unit testing is to quickly test the task script, not the external tools it's calling. We want to test all aspects of both success and failure paths.

Install test tools

We use [Mocha](#) as the test driver in this walk through.

```
npm install mocha --save-dev -g
npm install sync-request --save-dev
npm install @types/mocha --save-dev
```

Create test suite

Create a `tests` folder containing a `_suite.ts` file with the following contents:

```
import * as path from 'path';
import * as assert from 'assert';
import * as ttm from 'azure-pipelines-task-lib/mock-test';

describe('Sample task tests', function () {

    before(function() {

    });

    after(() => {

        it('should succeed with simple inputs', function(done: Mocha.Done) {
            // Add success test here
        });

        it('it should fail if tool returns 1', function(done: Mocha.Done) {
            // Add failure test here
        });
    });

});
```

TIP

Your test folder should be located in the `buildandreleasetask` folder. If you get a `sync-request` error, you can work around it by adding `sync-request` to the `buildandreleasetask` folder with the command `npm i --save-dev sync-request`.

Create success test

The success test validates that when the tool has the appropriate inputs, it succeeds with no errors or warnings and returns the correct output.

Create a file containing your task mock runner. This file creation simulates running the task and mocks all calls to outside methods.

Create a `success.ts` file in your test directory with the following contents:

```

import ma = require('azure-pipelines-task-lib/mock-answer');
import tmrm = require('azure-pipelines-task-lib/mock-run');
import path = require('path');

let taskPath = path.join(__dirname, '..', 'index.js');
let tmr: tmrm.TaskMockRunner = new tmrm.TaskMockRunner(taskPath);

tmr.setInput('samplestring', 'human');

tmr.run();

```

Next, add the following example success test to your `_suite.ts` file to run the task mock runner:

```

it('should succeed with simple inputs', function(done: Mocha.Done) {
  this.timeout(1000);

  let tp = path.join(__dirname, 'success.js');
  let tr: ttm.MockTestRunner = new ttm.MockTestRunner(tp);

  tr.run();
  console.log(tr.succeeded);
  assert.equal(tr.succeeded, true, 'should have succeeded');
  assert.equal(tr.warningIssues.length, 0, "should have no warnings");
  assert.equal(tr.errorIssues.length, 0, "should have no errors");
  console.log(tr.stdout);
  assert.equal(tr.stdout.indexOf('Hello human') >= 0, true, "should display Hello human");
  done();
});

```

Create failure test

The failure test validates that when the tool gets bad or incomplete input, it fails in the expected way with helpful output.

First, we create our task mock runner. To do so, create a `failure.ts` file in your test directory with the following contents:

```

import ma = require('azure-pipelines-task-lib/mock-answer');
import tmrm = require('azure-pipelines-task-lib/mock-run');
import path = require('path');

let taskPath = path.join(__dirname, '..', 'index.js');
let tmr: tmrm.TaskMockRunner = new tmrm.TaskMockRunner(taskPath);

tmr.setInput('samplestring', 'bad');

tmr.run();

```

Next, add the following to your `_suite.ts` file to run the task mock runner:

```
it('it should fail if tool returns 1', function(done: Mocha.Done) {
  this.timeout(1000);

  let tp = path.join(__dirname, 'failure.js');
  let tr: ttm.MockTestRunner = new ttm.MockTestRunner(tp);

  tr.run();
  console.log(tr.succeeded);
  assert.equal(tr.succeeded, false, 'should have failed');
  assert.equal(tr.warningIssues.length, 0, "should have no warnings");
  assert.equal(tr.errorIssues.length, 1, "should have 1 error issue");
  assert.equal(tr.errorIssues[0], 'Bad input was given', 'error issue output');
  assert.equal(tr.stdout.indexOf('Hello bad'), -1, "Should not display Hello bad");

  done();
});
```

Run the tests

To run the tests, run the following commands:

```
tsc
mocha tests/_suite.js
```

Both tests should pass. If you want to run the tests with more verbose output (what you'd see in the build console), set the environment variable: `TASK_TEST_TRACE=1`.

```
$env:TASK_TEST_TRACE=1
```

Step 3: Create the extension manifest file

The extension manifest contains all of the information about your extension. It includes links to your files, including your task folders and images folders. Ensure you've created an images folder with extension-icon.png. The following example is an extension manifest that contains the build or release task.

Copy the following json code and save it as your `vss-extension.json` file in your `home` directory. Don't create this file in the buildandreleasetask folder.

```
{
  "manifestVersion": 1,
  "id": "build-release-task",
  "name": "Fabrikam Build and Release Tools",
  "version": "0.0.1",
  "publisher": "fabrikam",
  "targets": [
    {
      "id": "Microsoft.VisualStudio.Services"
    }
  ],
  "description": "Tools for building/releasing with Fabrikam. Includes one build/release task.",
  "categories": [
    "Azure Pipelines"
  ],
  "icons": {
    "default": "images/extension-icon.png"
  },
  "files": [
    {
      "path": "buildAndReleaseTask"
    }
  ],
  "contributions": [
    {
      "id": "custom-build-release-task",
      "type": "ms.vss-distributed-task.task",
      "targets": [
        "ms.vss-distributed-task.tasks"
      ],
      "properties": {
        "name": "buildAndReleaseTask"
      }
    }
  ]
}
```

NOTE

The **publisher** here must be changed to your publisher name. If you want to create a publisher now, go to [create your publisher](#) for instructions.

Contributions

PROPERTY	DESCRIPTION
<code>id</code>	Identifier of the contribution. Must be unique within the extension. Doesn't need to match the name of the build or release task. Typically the build or release task name is in the ID of the contribution.
<code>type</code>	Type of the contribution. Should be ms.vss-distributed-task.task .
<code>targets</code>	Contributions "targeted" by this contribution. Should be ms.vss-distributed-task.tasks .
<code>properties.name</code>	Name of the task. This name must match the folder name of the corresponding self-contained build or release task pipeline.

Files

PROPERTY	DESCRIPTION
path	Path of the file or folder relative to the <code>home</code> directory.

NOTE

For more information about the [extension manifest file](#), such as its properties and what they do, check out the [extension manifest reference](#).

Step 4: Package your extension

After you've written your extension, the next step toward getting it into the Visual Studio Marketplace is to package all of your files together. All extensions are packaged as VSIX 2.0-compatible .vsix files. Microsoft provides a cross-platform command-line interface (CLI) to package your extension.

Packaging your extension into a .vsix file is effortless after you have the [tfx-cli](#), go to your extension's home directory, and run the following command:

```
tfx extension create --manifest-globs vss-extension.json
```

NOTE

An extension or integration's version must be incremented on every update. When you're updating an existing extension, either update the version in the manifest or pass the `--rev-version` command line switch. This increments the *patch* version number of your extension and saves the new version to your manifest. You must rev both the task version and extension version for an update to occur.

`tfx extension create --manifest-globs vss-extension.json --rev-version` only updates the extension version and not the task version. For more information, see [Build Task in GitHub](#).

After you have your packaged extension in a .vsix file, you're ready to publish your extension to the Marketplace.

Step 5: Publish your extension

Create your publisher

All extensions, including extensions from Microsoft, are identified as being provided by a publisher. If you aren't already a member of an existing publisher, you'll create one.

1. Sign in to the [Visual Studio Marketplace Publishing Portal](#).
2. If you aren't already a member of an existing publisher, you're prompted to create a publisher. If you're not prompted to create a publisher, scroll down to the bottom of the page and select **Publish extensions** under **Related Sites**.
 - Specify an identifier for your publisher, for example: `mycompany-myteam`.
 - This identifier is used as the value for the `publisher` attribute in your extensions' manifest file.
 - Specify a display name for your publisher, for example: `My Team`.
3. Review the [Marketplace Publisher Agreement](#) and select **Create**.

Your publisher is defined. In a future release, you can grant permissions to view and manage your publisher's extensions. It's easier and more secure to publish extensions under a common publisher, without the need to

share a set of credentials across users.

Upload your extension

After creating a publisher, you can upload your extension to the Marketplace.

- Find the **Upload new extension** button, go to your packaged .vsix file, and select **Upload**.

You can also upload your extension via the command line by using the `tfx extension publish` command instead of `tfx extension create` to package and publish your extension in one step. You can optionally use `--share-with` to share your extension with one or more accounts after publishing. You'll need a personal access token, too. For more information, see [Acquire a personal access token](#).

```
tfx extension publish --manifest-globs your-manifest.json --share-with yourOrganization
```

Share your extension

Now that you've uploaded your extension, it's in the Marketplace, but no one can see it. Share it with your organization so that you can install and test it.

- Right-click your extension and select **Share**, and enter your organization information. You can share it with other accounts that you want to have access to your extension, too.

IMPORTANT

Publishers must be verified to share extensions publicly. To learn more, see [Package/Publish/Install](#).

Now that your extension is in the Marketplace and shared, anyone who wants to use it must install it.

Step 6: Create a build and release pipeline to publish the extension to Marketplace

Create a build and release pipeline on Azure DevOps to help maintain the custom task on the Marketplace.

Prerequisites

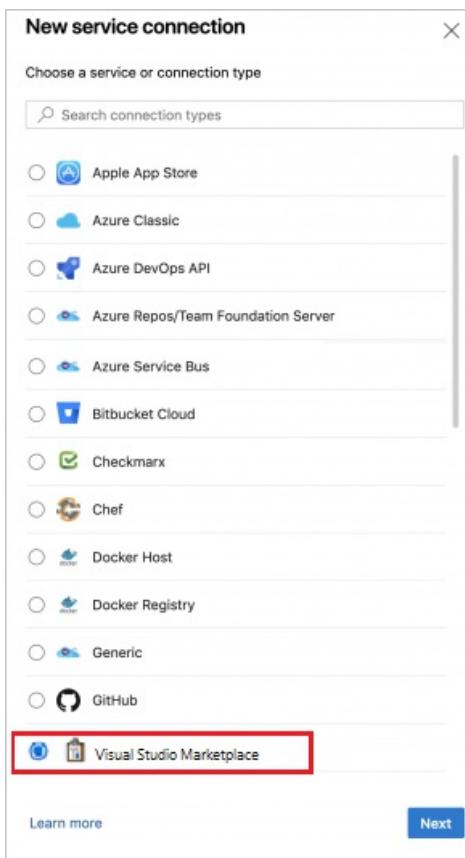
- A project in your organization. If you need to create one, see [Create a project](#).
- An [Azure DevOps Extension Tasks](#) extension installed in your organization.

Create a pipeline library variable group to hold the variables used by the pipeline. For more information about creating a variable group, see [Add and use variable groups](#). Keep in mind that you can make variable groups from the Azure DevOps Library tab or through the CLI. After a variable group is made, use any variables within that group in your pipeline. Read more on [How use a variable group](#).

Declare the following variables in the variable group:

- `publisherId` : ID of your marketplace publisher.
- `extensionId` : ID of your extension, as declared in the vss-extension.json file.
- `extensionName` : Name of your extension, as declared in the vss-extension.json file.
- `artifactName` : Name of the artifact being created for the VSIX file.

Create a new Visual Studio Marketplace service connection and grant access permissions for all pipelines. For more information about creating a service connection, see [Service connections](#).



New Visual Studio Marketplace service connection

Marketplace URL
https://marketplace.visualstudio.com

Authentication

Personal access token

Azure DevOps personal access token. Required permissions:
Publish: All accessible organisations, Marketplace (Publish)
Share: All accessible organisations, Marketplace Publish
Install: All accessible organisations or a specific organisation, Extensions (read and manage), Marketplace (acquire)
Query Version: All accessible organisations, Marketplace (read)
IsValid Extension: All accessible organisations, Marketplace (read)

[More information](#)

Details

Service connection name

Description (optional)

Security

Grant access permission to all pipelines

[Learn more](#) [Troubleshoot](#) [Back](#) [Save](#)

Use the following example to create a new pipeline with YAML. Learn more about how to [Create your first pipeline](#) and [YAML schema](#).

```

trigger:
- master

pool:
  vmImage: "ubuntu-latest"

variables:
- group: variable-group # Rename to whatever you named your variable group in the prerequisite stage of
step 6

stages:
- stage: Run_and_publish_unit_tests
  jobs:
    - job:
      steps:
        - task: TfxInstaller@3
          inputs:
            version: "v0.7.x"
        - task: Npm@1
          inputs:
            command: 'install'
            workingDir: '/TaskDirectory' # Update to the name of the directory of your task
        - task: Bash@3
          displayName: Compile Javascript
          inputs:
            targetType: "inline"
            script: |
              cd TaskDirectory # Update to the name of the directory of your task
              tsc
        - task: Npm@1
          inputs:
            command: 'custom'
            workingDir: '/TestsDirectory' # Update to the name of the directory of your task's tests
            customCommand: 'testScript' # See the definition in the explanation section below - it may be
called test
          - task: PublishTestResults@2
            inputs:
              testResultsFormat: 'JUnit'
              testResultsFiles: '**/ResultsFile.xml'
    - stage: Package_extension_and_publish_build_artifacts
      jobs:
        - job:
          steps:
            - task: TfxInstaller@3
              inputs:
                version: "v0.7.x"
            - task: Npm@1
              inputs:
                command: 'install'
                workingDir: '/TaskDirectory' # Update to the name of the directory of your task
            - task: Bash@3
              displayName: Compile Javascript
              inputs:
                targetType: "inline"
                script: |
                  cd TaskDirectory # Update to the name of the directory of your task
                  tsc
            - task: QueryAzureDevOpsExtensionVersion@3
              inputs:
                connectTo: 'VsTeam'
                connectedServiceName: 'ServiceConnection' # Change to whatever you named the service
connection
              publisherId: '$(PublisherID)'
              extensionId: '$(ExtensionID)'
              versionAction: 'Patch'
              outputVariable: 'Task.Extension.Version'
        - task: PackageAzureDevOpsExtension@3
          inputs:

```

```

rootFolder: '$(System.DefaultWorkingDirectory)'
publisherId: '$(PublisherID)'
extensionId: '$(ExtensionID)'
extensionName: '$(ExtensionName)'
extensionVersion: '$(Task.Extension.Version)'
updateTasksVersion: true
updateTasksVersionType: 'patch'
extensionVisibility: 'private' # Change to public if you're publishing to the marketplace
extensionPricing: 'free'
- task: CopyFiles@2
  displayName: "Copy Files to: $(Build.ArtifactStagingDirectory)"
  inputs:
    Contents: "**/*.vsix"
    TargetFolder: "$(Build.ArtifactStagingDirectory)"
- task: PublishBuildArtifacts@1
  inputs:
    PathToPublish: '$(Build.ArtifactStagingDirectory)'
    ArtifactName: '$(ArtifactName)'
    publishLocation: 'Container'
- stage: Download_build_artifacts_and_publish_the_extension
  jobs:
    - job:
      steps:
        - task: TfxInstaller@3
          inputs:
            version: "v0.7.x"
        - task: DownloadBuildArtifacts@0
          inputs:
            buildType: "current"
            downloadType: "single"
            artifactName: '$(ArtifactName)'
            downloadPath: "$(System.DefaultWorkingDirectory)"
        - task: PublishAzureDevOpsExtension@3
          inputs:
            connectTo: 'VsTeam'
            connectedServiceName: 'ServiceConnection' # Change to whatever you named the service
connection
          fileType: 'vsix'
          vsixFile: '/Publisher.*.vsix'
          publisherId: '$(PublisherID)'
          extensionId: '$(ExtensionID)'
          extensionName: '$(ExtensionName)'
          updateTasksVersion: false
          extensionVisibility: 'private' # Change to public if you're publishing to the marketplace
          extensionPricing: 'free'

```

For more help with triggers, such as CI and PR triggers, see [Specify events that trigger pipelines](#).

NOTE

Each job uses a new user agent and requires dependencies to be installed.

Pipeline stages

This section helps you understand how the pipeline stages work.

Stage: Run and publish unit tests

This stage runs unit tests and publishes test results to Azure DevOps.

To run unit tests, add a custom script to the package.json file. For example:

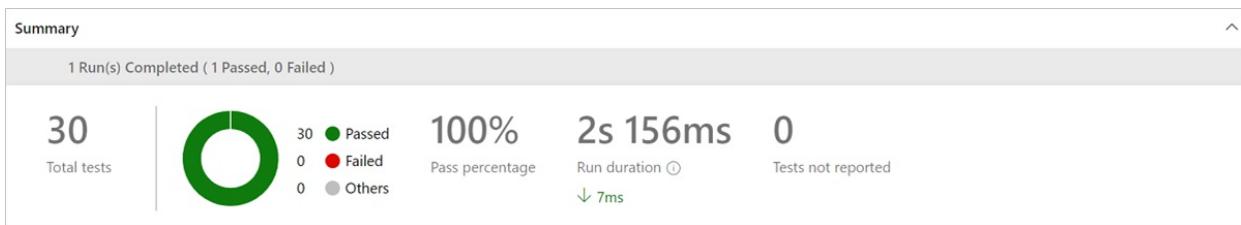
```

"scripts": {
  "testScript": "mocha ./TestFile --reporter xunit --reporter-option output=ResultsFile.xml"
},

```

1. Add "Use Node CLI for Azure DevOps (tfx-cli)" to install the tfx-cli onto your build agent.
2. Add the "npm" task with the "install" command and target the folder with the package.json file.
3. Add the "Bash" task to compile the TypeScript into JavaScript.
4. Add the "npm" task with the "custom" command, target the folder that contains the unit tests, and input `testScript` as the command. Use the following inputs:
 - Command: custom
 - Working folder that contains package.json: /TestsDirectory
 - Command and arguments: testScript
5. Add the "Publish Test Results" task. If you're using the Mocha XUnit reporter, ensure that the result format is "JUnit" and not "XUnit." Set the search folder to the root directory. Use the following inputs:
 - Test result format: JUnit
 - Test results files: **/ResultsFile.xml
 - Search folder: \$(System.DefaultWorkingDirectory)

After the test results have been published, the output under the tests tab should look like this:



Stage: Package the extension and publish build artifacts

1. Add "Use Node CLI for Azure DevOps (tfx-cli)" to install the tfx-cli onto your build agent.
2. Add the "npm" task with the "install" command and target the folder with the package.json file.
3. Add the "Bash" task to compile the TypeScript into JavaScript.
4. Add the "Query Extension Version" task to query the existing extension version. Use the following inputs:
 - Connect to: Visual Studio Marketplace
 - Visual Studio Marketplace (Service connection): Service Connection
 - Publisher ID: ID of your Visual Studio Marketplace publisher
 - Extension ID: ID of your extension in the vss-extension.json file
 - Increase version: Patch
 - Output Variable: Task.Extension.Version
5. Add the "Package Extension" task to package the extensions based on manifest Json. Use the following inputs:
 - Root manifests folder: Points to root directory that contains manifest file. For example, \$(System.DefaultWorkingDirectory) is the root directory.
 - Manifest file(s): vss-extension.json.
 - Publisher ID: ID of your Visual Studio Marketplace publisher.
 - Extension ID: ID of your extension in the vss-extension.json file.
 - Extension Name: Name of your extension in the vss-extension.json file.

- Extension Version: \$(Task.Extension.Version).
- Override tasks version: checked (true).
- Override Type: Replace Only Patch (1.0.r).
- Extension Visibility: If the extension is still in development, set the value to private. To release the extension to the public, set the value to public.

6. Add the "Copy files" task to copy published files. Use the following inputs:

- Contents: All of the files that need to be copied for publishing them as an artifact
- Target folder: The folder that the files get copied to
 - For example: \$(Build.ArtifactStagingDirectory)

7. Add "Publish build artifacts" to publish the artifacts for use in other jobs or pipelines. Use the following inputs:

- Path to publish: The path to the folder that contains the files that are being published.
 - For example: \$(Build.ArtifactStagingDirectory).
- Artifact name: The name given to the artifact.
- Artifacts publish location: Choose "Azure Pipelines" to use the artifact in future jobs.

Stage: Download build artifacts and publish the extension

1. Add "Use Node CLI for Azure DevOps (tfx-cli)" to install the tfx-cli onto your build agent.

2. Add the "Download build artifacts" task to download the artifacts onto a new job. Use the following inputs:

- Download artifacts produced by: If you're downloading the artifact on a new job from the same pipeline, select "Current build." If you're downloading on a new pipeline, select "Specific build."
- Download type: Choose "Specific artifact" to download all files that were published.
- Artifact name: The published artifact's name.
- Destination directory: The folder where the files should be downloaded.

3. The last task that you need is the "Publish Extension" task. Use the following inputs:

- Connect to: Visual Studio Marketplace
- Visual Studio Marketplace connection: ServiceConnection
- Input file type: VSIX file
- VSIX file: /Publisher.*.vsix
- Publisher ID: ID of your Visual Studio Marketplace publisher
- Extension ID: ID of your extension in the vss-extension.json file
- Extension Name: Name of your extension in the vss-extension.json file
- Extension visibility: Either private or public

Optional: Install and test your extension

Install an extension that is shared with you in just a few steps:

1. From your organization control panel (https://dev.azure.com/{organization}/_admin), go to the project collection administration page.
2. In the **Extensions** tab, find your extension in the "Extensions Shared With Me" group and select the extension link.
3. Install the extension.

If you can't see the **Extensions** tab, make sure you're in the control panel (the administration page at the project collection level, https://dev.azure.com/{organization}/_admin) and not the administration page for a project.

If you don't see the **Extensions** tab, then extensions aren't enabled for your organization. You can get early access to the extensions feature by joining the Visual Studio Partner Program.

For build and release tasks to package and publish Azure DevOps Extensions to the Visual Studio Marketplace, you can download [Azure DevOps Extension Tasks](#).

Related articles

- [Extension Manifest Reference](#)
- [Build/Release Task JSON Schema](#)
- [Build/Release Task Examples](#)

Specify jobs in your pipeline

5/18/2021 • 24 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

You can organize your pipeline into jobs. Every pipeline has at least one job. A job is a series of steps that run sequentially as a unit. In other words, a job is the smallest unit of work that can be scheduled to run.

You can organize your build or release pipeline into jobs. Every pipeline has at least one job. A job is a series of steps that run sequentially as a unit. In other words, a job is the smallest unit of work that can be scheduled to run.

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 release pipeline into jobs. Every release pipeline has at least one job. Jobs are not supported in a build pipeline in this version of TFS.

NOTE

You must install Update 2 to use jobs in a release pipeline in TFS 2017. Jobs in build pipelines are available in Azure Pipelines, TFS 2018.2, and newer versions.

Define a single job

- [YAML](#)
- [Classic](#)

In the simplest case, a pipeline has a single job. In that case, you do not have to explicitly use the `job` keyword unless you are using a [template](#). You can directly specify the steps in your YAML file.

This YAML file has a job that runs on a [Microsoft-hosted agent](#) and outputs `Hello world`.

```
pool:  
  vmImage: 'ubuntu-16.04'  
steps:  
- bash: echo "Hello world"
```

You may want to specify additional properties on that job. In that case, you can use the `job` keyword.

```
jobs:
- job: myJob
  timeoutInMinutes: 10
  pool:
    vmImage: 'ubuntu-16.04'
  steps:
    - bash: echo "Hello world"
```

Your pipeline may have multiple jobs. In that case, use the `jobs` keyword.

```
jobs:
- job: A
  steps:
    - bash: echo "A"

- job: B
  steps:
    - bash: echo "B"
```

Your pipeline may have multiple stages, each with multiple jobs. In that case, use the `stages` keyword.

```
stages:
- stage: A
  jobs:
    - job: A1
    - job: A2

- stage: B
  jobs:
    - job: B1
    - job: B2
```

The full syntax to specify a job is:

```
- job: string # name of the job, A-Z, a-z, 0-9, and underscore
  displayName: string # friendly name to display in the UI
  dependsOn: string | [ string ]
  condition: string
  strategy:
    parallel: # parallel strategy
    matrix: # matrix strategy
    maxParallel: number # maximum number simultaneous matrix legs to run
    # note: `parallel` and `matrix` are mutually exclusive
    # you may specify one or the other; including both is an error
    # `maxParallel` is only valid with `matrix`
  continueOnError: boolean # 'true' if future jobs should run even if this job fails; defaults to 'false'
  pool: pool # agent pool
  workspace:
    clean: outputs | resources | all # what to clean up before the job runs
    container: containerReference # container 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 | variableReference ]
    steps: [ script | bash | pwsh | powershell | checkout | task | templateReference ]
    services: { string: string | container } # container resources to run as a service container
```

The full syntax to specify a job is:

```

- job: string # name of the job, A-Z, a-z, 0-9, and underscore
  displayName: string # friendly name to display in the UI
  dependsOn: string | [ string ]
  condition: string
  strategy:
    parallel: # parallel strategy
    matrix: # matrix strategy
    maxParallel: number # maximum number simultaneous matrix legs to run
    # note: `parallel` and `matrix` are mutually exclusive
    # you may specify one or the other; including both is an error
    # `maxParallel` is only valid with `matrix`
  continueOnError: boolean # 'true' if future jobs should run even if this job fails; defaults to 'false'
  pool: pool # agent pool
  workspace:
    clean: outputs | resources | all # what to clean up before the job runs
    container: containerReference # container 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 | variableReference ]
  steps: [ script | bash | pwsh | powershell | checkout | task | templateReference ]
  services: { string: string | container } # container resources to run as a service container
  uses: # Any resources (repos or pools) required by this job that are not already referenced
    repositories: [ string ] # Repository references to Azure Git repositories
    pools: [ string ] # Pool names, typically when using a matrix strategy for the job

```

If the primary intent of your job is to deploy your app (as opposed to build or test your app), then you can use a special type of job called **deployment job**.

The syntax for a deployment job is:

```

- deployment: string      # instead of job keyword, use deployment keyword
  pool:
    name: string
    demands: string | [ string ]
  environment: string
  strategy:
    runOnce:
      deploy:
        steps:
          - script: echo Hi!

```

Although you can add steps for deployment tasks in a `job`, we recommend that you instead use a [deployment job](#). A deployment job has a few benefits. For example, you can deploy to an environment, which includes benefits such as being able to see the history of what you've deployed.

YAML is not supported in this version of TFS.

Types of jobs

Jobs can be of different types, depending on where they run.

- [YAML](#)
- [Classic](#)

- **Agent pool jobs** run on an agent in an agent pool.
- **Server jobs** run on the Azure DevOps Server.
- **Container jobs** run in a container on an agent in an agent pool. For more information about choosing containers, see [Define container jobs](#).

- [YAML](#)
- [Classic](#)
- **Agent pool jobs** run on an agent in an agent pool.
- **Server jobs** run on the Azure DevOps Server.
- **Agent pool jobs** run on an agent in the agent pool. These jobs are available in build and release pipelines.
- **Server jobs** run on TFS. These jobs are available in build and release pipelines.
- **Deployment group jobs** run on machines in a deployment group. These jobs are available only in release pipelines.
- **Agent pool jobs** run on an agent in the agent pool. These jobs are only available release pipelines.

Agent pool jobs

These are the most common type of jobs and they run on an agent in an agent pool.

- When using Microsoft-hosted agents, each job in a pipeline gets a fresh agent.
- Use [demands](#) with self-hosted agents to specify what capabilities an agent must have to run your job. You may get the same agent for consecutive jobs, depending on whether there is more than one agent in your agent pool that matches your pipeline's demands. If there is only one agent in your pool that matches the pipeline's demands, the pipeline will wait until this agent is available.

NOTE

Demands and capabilities are designed for use with self-hosted agents so that jobs can be matched with an agent that meets the requirements of the job. When using Microsoft-hosted agents, you select an image for the agent that matches the requirements of the job, so although it is possible to add capabilities to a Microsoft-hosted agent, you don't need to use capabilities with Microsoft-hosted agents.

- [YAML](#)
- [Classic](#)

```
pool:
  name: myPrivateAgents    # your job runs on an agent in this pool
  demands: agent.os -equals Windows_NT    # the agent must have this capability to run the job
  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](#).

Server jobs

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 are supported in a server job at present.

Agentless jobs supported tasks

Currently, only the following tasks are supported out of the box for agentless jobs:

- [Delay task](#)
- [Invoke Azure Function task](#)
- [Invoke REST API task](#)
- [Publish To Azure Service Bus task](#)
- [Query Azure Monitor Alerts task](#)
- [Query Work Items task](#)

Because tasks are extensible, you can add more agentless tasks by using extensions. The default timeout for agentless jobs is 60 minutes.

- [YAML](#)
- [Classic](#)

The full syntax to specify a server 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  
  pool: server
```

YAML is not yet supported in TFS.

Dependencies

When you define multiple jobs in a single stage, you can specify dependencies between them. Pipelines must contain at least one job with no dependencies.

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](#)
- [Classic](#)

The syntax for defining multiple jobs and their dependencies is:

```
jobs:  
- job: string  
  dependsOn: string  
  condition: string
```

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-10.14'  
  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 builds are not yet available on 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](#)
- [Classic](#)

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
```

YAML builds are not yet available on TFS.

Timeouts

To avoid taking up resources when your job is unresponsive 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 60 minutes on Microsoft-hosted agents with a private project or private repository (unless [additional capacity](#) is paid for)

The timeout period begins when the job starts running. It does not include the time the job is queued or is waiting for an agent.

- [YAML](#)
- [Classic](#)

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 the deployment task is set to keep running if a previous task has failed. When not specified, the default is 5 minutes.

```
jobs:
- job: Test
  timeoutInMinutes: 10 # how long to run the job before automatically cancelling
  cancelTimeoutInMinutes: 2 # how much time to give 'run always even if cancelled tasks' before stopping them
```

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-job configuration

From a single job you author, you can run multiple jobs on multiple agents in parallel. Some examples include:

- **Multi-configuration builds:** You can 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:** You can run multiple deployments in parallel, for example, to different geographic regions.
- **Multi-configuration testing:** You can run test multiple configurations in parallel.

- [YAML](#)
- [Classic](#)

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
```

NOTE

Matrix configuration names (like `us_IE` above) must contain only basic Latin alphabet letters (A-Z, a-z), numbers, and underscores (`_`). They must start with a letter. Also, they must be 100 characters or less.

It's also possible to use [output variables](#) to generate a matrix. This can be handy if you need to generate the matrix using a script.

`matrix` will accept a runtime expression containing a stringified JSON object. That JSON object, when expanded, must match the matrixing syntax. In the example below, we've hard-coded the JSON string, but it could be generated by a scripting language or command-line program.

```
jobs:  
- job: generator  
  steps:  
    - bash: echo "##vso[task.setVariable variable=legs;isOutput=true]{'a':{'myvar':'A'}, 'b':{'myvar':'B'}}"  
      name: mtrx  
    # This expands to the matrix  
    #   a:  
    #     myvar: A  
    #   b:  
    #     myvar: B  
- job: runner  
  dependsOn: generator  
  strategy:  
    matrix: $[ dependencies.generator.outputs['mtrx.legs'] ]  
  steps:  
    - script: echo $(myvar) # echos A or B depending on which leg is running
```

YAML is not 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](#)
- [Classic](#)

The `parallel` strategy enables a job to be duplicated many times. 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 five times with the values of `System.JobPositionInPhase` and `System.TotalJobsInPhase` set appropriately.

```
jobs:  
- job: Test  
  strategy:  
    parallel: 5
```

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](#)
- [Classic](#)

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.

For information about using a `condition`, see [Specify conditions](#).

Workspace

When you run an agent pool job, it creates a workspace on the agent. The workspace is a directory in which it downloads the source, runs steps, and produces outputs. The workspace directory can be referenced in your job using `Pipeline.Workspace` variable. Under this, various subdirectories are created:

When you run an agent pool job, it creates a workspace on the agent. The workspace is a directory in which it downloads the source, runs steps, and produces outputs. The workspace directory can be referenced in your job using `Agent.BuildDirectory` variable. Under this, various subdirectories are created:

- `Build.SourcesDirectory` is where tasks download the application's source code.
 - `Build.ArtifactStagingDirectory` is where tasks download artifacts needed for the pipeline or upload artifacts before they are published.
 - `Build.BinariesDirectory` is where tasks write their outputs.
 - `Common.TestResultsDirectory` is where tasks upload their test results.
- [YAML](#)
- [Classic](#)

When you run a pipeline on a **self-hosted agent**, by default, none of the subdirectories are cleaned in between two consecutive runs. As a result, you can do incremental builds and deployments, provided that tasks are implemented to make use of that. You can override this behavior using the `workspace` setting on the job.

IMPORTANT

The workspace clean options are applicable only for self-hosted agents. When using Microsoft-hosted agents, job are always run on a new agent.

```
- job: myJob
  workspace:
    clean: outputs | resources | all # what to clean up before the job runs
```

When you specify one of the `clean` options, they are interpreted as follows:

- `outputs` : Delete `Build.BinariesDirectory` before running a new job.
- `resources` : Delete `Build.SourcesDirectory` before running a new job.
- `all` : Delete the entire `Pipeline.Workspace` directory before running a new job.

`$(Build.ArtifactStagingDirectory)` and `$(Common.TestResultsDirectory)` are always deleted and recreated prior to every build regardless of any of these settings.

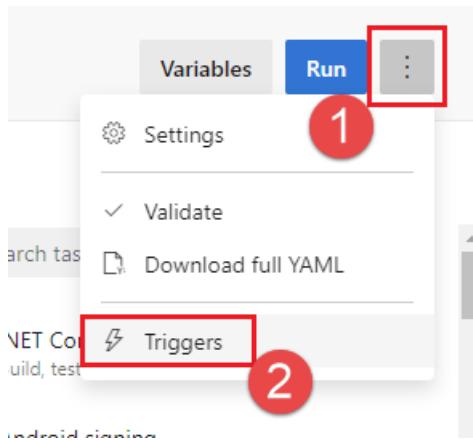
```
jobs:
- deployment: deploy
  pool:
    vmImage: 'Ubuntu-16.04'
  workspace:
    clean: all
  environment: staging
```

NOTE

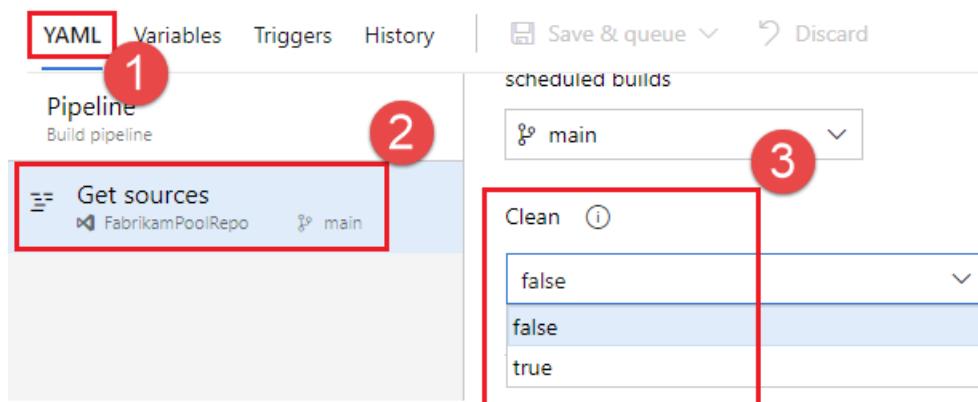
Depending on your agent capabilities and pipeline demands, each job may be routed to a different agent in your self-hosted pool. As a result, you may get a new agent for subsequent pipeline runs (or stages or jobs in the same pipeline), so **not** cleaning is not a guarantee that subsequent runs, jobs, or stages will be able to access outputs from previous runs, jobs, or stages. You can configure agent capabilities and pipeline demands to specify which agents are used to run a pipeline job, but unless there is only a single agent in the pool that meets the demands, there is no guarantee that subsequent jobs will use the same agent as previous jobs. For more information, see [Specify demands](#).

In addition to workspace clean, you can also configure cleaning by configuring the **Clean** setting in the pipeline settings UI. When the **Clean** setting is **true** it is equivalent to specifying `clean: true` for every `checkout` step in your pipeline. To configure the **Clean** setting:

1. Edit your pipeline, choose ..., and select **Triggers**.



2. Select **YAML**, **Get sources**, and configure your desired **Clean** setting. The default is **false**.



YAML is not yet supported in TFS.

Artifact download

This example YAML file publishes the artifact `Website` and then downloads the artifact to `$(Pipeline.Workspace)`. The Deploy job only runs if the Build job is successful.

- [YAML](#)
- [Classic](#)

```

# 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()

```

YAML is not yet supported in TFS.

For information about using **dependsOn** and **condition**, see [Specify conditions](#).

Access to OAuth token

You can allow scripts running in a job to access the current Azure Pipelines or TFS OAuth security token. The token can be used to authenticate to the Azure Pipelines REST API.

- [YAML](#)
- [Classic](#)

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: |
  $url =
"$(env: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:SYSTEM_ACCESSTOKEN"
  }
  Write-Host "Pipeline = $($pipeline | ConvertTo-Json -Depth 100)"
env:
  SYSTEM_ACCESSTOKEN: $(system.accesstoken)

```

YAML is not yet supported in TFS.

What's next

- [Deployment group jobs](#)

- Conditions

Define container jobs (YAML)

6/2/2021 • 6 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019

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 Azure Pipelines. Over time, you may find that you want more control over the context where your tasks run. YAML pipelines offer container jobs for this level of control.

On Linux and Windows agents, jobs may be run [on the host](#) or [in a container](#). (On macOS and Red Hat Enterprise Linux 6, container jobs are not available.) Containers provide isolation from the host and allow you to pin specific versions of tools and dependencies. Host jobs require less initial setup and infrastructure to maintain.

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. You cannot have nested containers. Containers are not supported when an agent is already running inside a container.

If you need fine-grained control at the individual step level, [step targets](#) allow you to choose container or host for each step.

Requirements

Linux-based containers

The Azure Pipelines system requires a few things in Linux-based containers:

- Bash
- glibc-based
- Can run Node.js (which the agent provides)
- Does not define an `ENTRYPOINT`
- `USER` has access to `groupadd` and other privileges commands without `sudo`

And on your agent host:

- Ensure Docker is installed
- The agent must have permission to access the Docker daemon

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. Containers with a `ENTRYPOINT` might not work, since Azure Pipelines will `docker create` an awaiting container and `docker exec` a series of commands which expect the container is always up and running.

NOTE

For Windows-based Linux containers, Node.js must be pre-installed.

Windows Containers

Azure Pipelines can also run [Windows Containers](#). [Windows Server version 1803](#) or higher is required. Docker must be installed. Be sure your pipelines agent has permission to access the Docker daemon.

The Windows container must support running Node.js. A base Windows Nano Server container is missing dependencies required to run Node.

Hosted agents

Only `windows-2019` and `ubuntu-*` images support running containers. The macOS image does not support running containers.

Single job

A simple example:

```
pool:  
  vmImage: 'ubuntu-18.04'  
  
  container: ubuntu:18.04  
  
  steps:  
    - script: printenv
```

This tells the system to fetch the `ubuntu` image tagged `18.04` from [Docker Hub](#) and then start the container.

When the `printenv` command runs, it will happen inside the `ubuntu:18.04` container.

A Windows example:

```
pool:  
  vmImage: 'windows-2019'  
  
  container: mcr.microsoft.com/windows/servercore:ltsc2019  
  
  steps:  
    - script: set
```

NOTE

Windows requires that the kernel version of the host and container match. Since this example uses the Windows 2019 image, we will use the `2019` tag for the container.

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. (And we don't have to mention the `jobs` keyword, since there's only a single job defined.)

```
pool:
  vmImage: 'ubuntu-18.04'

strategy:
  matrix:
    ubuntu14:
      containerImage: ubuntu:14.04
    ubuntu16:
      containerImage: ubuntu:16.04
    ubuntu18:
      containerImage: ubuntu:18.04

  container: ${ variables['containerImage'] }

steps:
- script: printenv
```

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:

```
container:
  image: registry:ubuntu1604
  endpoint: private_dockerhub_connection

steps:
- script: echo hello
```

or

```
container:
  image: myprivate.azurecr.io/windowsservercore:1803
  endpoint: my_acr_connection

steps:
- script: echo hello
```

Other container registries may also work. Amazon ECR doesn't currently work, as there are additional client tools required to convert AWS credentials into something Docker can use to authenticate.

NOTE

The Red Hat Enterprise Linux 6 build of the agent won't run container job. Choose another Linux flavor, such as Red Hat Enterprise Linux 7 or above.

Options

If you need to control container startup, you can specify `options`.

```

container:
  image: ubuntu:18.04
  options: --hostname container-test --ip 192.168.0.1

steps:
- script: echo hello

```

Running `docker create --help` will give you the list of supported options. You can use any option available with the `docker create` command.

Reusable container definition

In the following example, the containers are defined in the resources section. Each container is then referenced later, by referring to its assigned alias. (Here, we explicitly list the `jobs` keyword for clarity.)

```

resources:
  containers:
    - container: u14
      image: ubuntu:14.04

    - container: u16
      image: ubuntu:16.04

    - container: u18
      image: ubuntu:18.04

  jobs:
    - job: RunInContainer
      pool:
        vmImage: 'ubuntu-18.04'

  strategy:
    matrix:
      ubuntu14:
        containerResource: u14
      ubuntu16:
        containerResource: u16
      ubuntu18:
        containerResource: u18

    container: ${ variables['containerResource'] }

  steps:
    - script: printenv

```

Non glibc-based containers

The Azure Pipelines agent supplies a copy of Node.js, which is required to run tasks and scripts. The version of Node.js is compiled against the C runtime we use in our hosted cloud, typically glibc. Some variants of Linux use other C runtimes. For instance, Alpine Linux uses musl.

If you want to use a non-glibc-based container as a job container, you will need to arrange a few things on your own. First, you must supply your own copy of Node.js. Second, you must add a label to your image telling the agent where to find the Node.js binary. Finally, stock Alpine doesn't come with other dependencies that Azure Pipelines depends on: bash, sudo, which, and groupadd.

Bring your own Node.js

You are responsible for adding a Node binary to your container. Node 14 is a safe choice. You can start from the `node:14-alpine` image.

Tell the agent about Node.js

The agent will read a container label "com.azure.dev.pipelines.handler.node.path". If this label exists, it must be the path to the Node.js binary. For example, in an image based on `node:10-alpine`, add this line to your Dockerfile:

```
LABEL "com.azure.dev.pipelines.agent.handler.node.path"="/usr/local/bin/node"
```

Add requirements

Azure Pipelines assumes a Bash-based system with common administration packages installed. Alpine Linux in particular doesn't come with several of the packages needed. Installing `bash`, `sudo`, and `shadow` will cover the basic needs.

```
RUN apk add bash sudo shadow
```

If you depend on any in-box or Marketplace tasks, you'll also need to supply the binaries they require.

Full example of a Dockerfile

```
FROM node:10-alpine

RUN apk add --no-cache --virtual .pipeline-deps readline linux-pam \
    && apk add bash sudo shadow \
    && apk del .pipeline-deps

LABEL "com.azure.dev.pipelines.agent.handler.node.path"="/usr/local/bin/node"

CMD [ "node" ]
```

Multiple jobs with agent pools on a single hosted agent

The container job uses the underlying host agent Docker config.json for image registry authorization, which logs out at the end of the Docker registry container initialization. Subsequent registry image pulls authorization might be denied for "unauthorized authentication" because the Docker config.json file registered in the system for authentication has already been logged out by one of the other container jobs that are running in parallel.

The solution is to set the Docker environment variable `DOCKER_CONFIG` that is specific to each agent pool service running on the hosted agent. Export the `DOCKER_CONFIG` in each agent pool's runsvc.sh script:

```
#insert anything to set up env when running as a service
export DOCKER_CONFIG=./docker
```

Add stages, dependencies, & conditions

4/7/2021 • 9 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

The concept of stages varies depending on whether you use YAML pipelines or classic release pipelines.

- [YAML](#)
- [Classic](#)

You can organize pipeline jobs into stages. Stages are the major divisions in a pipeline: "build this app", "run these tests", and "deploy to pre-production" are good examples of stages. They are logical boundaries in your pipeline where you can pause the pipeline and perform various checks.

Every pipeline has at least one stage even if you do not explicitly define it. You can also arrange stages into a dependency graph so that one stage runs before another one.

NOTE

Support for stages was added in Azure DevOps Server 2019.1.

This version of TFS doesn't support YAML.

Specify stages

- [YAML](#)
- [Classic](#)

NOTE

Support for stages was added in Azure DevOps Server 2019.1.

In the simplest case, you do not need any logical boundaries in your pipeline. In that case, you do not have to explicitly use the `stage` keyword. You can directly specify the jobs in your YAML file.

```
# this has one implicit stage and one implicit job
pool:
  vmImage: 'ubuntu-16.04'
steps:
- bash: echo "Hello world"
```

```
# this pipeline has one implicit stage
jobs:
- job: A
  steps:
  - bash: echo "A"

- job: B
  steps:
  - bash: echo "B"
```

If you organize your pipeline into multiple stages, you use the `stages` keyword.

```
stages:
- stage: A
  jobs:
  - job: A1
  - job: A2

- stage: B
  jobs:
  - job: B1
  - job: B2
```

If you choose to specify a `pool` at the stage level, then all jobs defined in that stage will use that pool unless otherwise specified at the job-level.

NOTE

In Azure DevOps Server 2019, pools can only be specified at job level.

```
stages:
- stage: A
  pool: StageAPool
  jobs:
  - job: A1 # will run on "StageAPool" pool based on the pool defined on the stage
  - job: A2 # will run on "JobPool" pool
    pool: JobPool
```

The full syntax to specify a stage is:

```
stages:
- stage: string # name of the stage, A-Z, a-z, 0-9, and underscore
  displayName: string # friendly name to display in the UI
  dependsOn: string | [ string ]
  condition: string
  pool: string | pool
  variables: { string: string } | [ variable | variableReference ]
  jobs: [ job | templateReference]
```

This version of TFS doesn't support YAML pipelines.

Specify dependencies

- [YAML](#)
- [Classic](#)

NOTE

Support for stages was added in Azure DevOps Server 2019.1.

When you define multiple stages in a pipeline, by default, they run sequentially in the order in which you define them in the YAML file. The exception to this is when you add dependencies. With dependencies, stages run in the order of the `dependsOn` requirements.

Pipelines must contain at least one stage with no dependencies.

The syntax for defining multiple stages and their dependencies is:

```
stages:  
- stage: string  
  dependsOn: string  
  condition: string
```

Example stages that run sequentially:

```
# if you do not use a dependsOn keyword, stages run in the order they are defined  
stages:  
- stage: QA  
  jobs:  
    - job:  
      ...  
  
- stage: Prod  
  jobs:  
    - job:  
      ...
```

Example stages that run in parallel:

```
stages:  
- stage: FunctionalTest  
  jobs:  
    - job:  
      ...  
  
- stage: AcceptanceTest  
  dependsOn: []    # this removes the implicit dependency on previous stage and causes this to run in parallel  
  jobs:  
    - job:  
      ...
```

Example of fan-out and fan-in:

```

stages:
- stage: Test

- stage: DeployUS1
dependsOn: Test    # this stage runs after Test

- stage: DeployUS2
dependsOn: Test    # this stage runs in parallel with DeployUS1, after Test

- stage: DeployEurope
dependsOn:          # this stage runs after DeployUS1 and DeployUS2
- DeployUS1
- DeployUS2

```

This version of TFS doesn't support YAML pipelines.

Conditions

You can specify the conditions under which each stage runs. By default, a stage runs if it does not depend on any other stage, or if all of the stages that it depends on have completed and succeeded. You can customize this behavior by forcing a stage to run even if a previous stage fails or by specifying a custom condition.

If you customize the default condition of the preceding steps for a stage, you remove the conditions for completion and success. So, if you use a custom condition, it's common to use

`and(succeeded(),custom_condition)` to check whether the preceding stage ran successfully. Otherwise, the stage runs regardless of the outcome of the preceding stage.

NOTE

Conditions for failed ('JOBNAME/STAGENAME') and succeeded ('JOBNAME/STAGENAME') as shown in the following example work only for [YAML pipelines](#).

- [YAML](#)
- [Classic](#)

NOTE

Support for stages was added in Azure DevOps Server 2019.1.

Example to run a stage based upon the status of running a previous stage:

```

stages:
- stage: A

# stage B runs if A fails
- stage: B
  condition: failed()

# stage C runs if B succeeds
- stage: C
  dependsOn:
  - A
  - B
  condition: succeeded('B')

```

Example of using a [custom condition](#):

```
stages:  
- stage: A  
  
- stage: B  
  condition: and(succeeded(), eq(variables['build.sourceBranch'], 'refs/heads/master'))
```

This version of TFS doesn't support YAML pipelines.

Specify queuing policies

- [YAML](#)
- [Classic](#)

YAML pipelines don't support queuing policies. Each run of a pipeline is independent from and unaware of other runs. In other words, your two successive commits may trigger two pipelines, and both of them will execute the same sequence of stages without waiting for each other. While we work to bring queuing policies to YAML pipelines, we recommend that you use [manual approvals](#) in order to manually sequence and control the order the execution if this is of importance.

This version of TFS does not support YAML pipelines.

Specify approvals

- [YAML](#)
- [Classic](#)

You can manually control when a stage should run using approval checks. This is commonly used to control deployments to production environments. Checks are a mechanism available to the *resource owner* to control if and when a stage in a pipeline can consume a resource. As an owner of a resource, such as an environment, you can define checks that must be satisfied before a stage consuming that resource can start.

Currently, manual approval checks are supported on environments. For more information, see [Approvals](#).

Approvals are not yet supported in YAML pipelines in this version of Azure DevOps Server.

This version of TFS doesn't support YAML pipelines.

Deployment jobs

6/2/2021 • 12 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020

IMPORTANT

- Job and stage names cannot contain keywords (example: `deployment`).
- Each job in a stage must have a unique name.

In YAML pipelines, we recommend that you put your deployment steps in a special type of [job](#) called a deployment job. A deployment job is a collection of steps that are run sequentially against the environment. A deployment job and a [traditional job](#) can exist in the same stage.

Deployment jobs provide the following benefits:

- **Deployment history:** You get the deployment history across pipelines, down to a specific resource and status of the deployments for auditing.
- **Apply deployment strategy:** You define how your application is rolled out.

NOTE

We currently support only the `runOnce`, `rolling`, and the `canary` strategies.

A deployment job doesn't automatically clone the source repo. You can checkout the source repo within your job with `checkout: self`.

Schema

Here is the full syntax to specify a deployment job:

```

jobs:
  - deployment: string    # name of the deployment job, A-Z, a-z, 0-9, and underscore. The word "deploy" is a keyword and is unsupported as the deployment name.
    displayName: string    # friendly name to display in the UI
    pool:
      name: string         # Use only global level variables for defining a pool name. Stage/job level variables are not supported to define pool name.
      demands: string | [ string ]
    workspace:
      clean: outputs | resources | all # what to clean up before the job runs
      dependsOn: string
      condition: string
      continueOnError: boolean          # 'true' if future jobs should run even if this job fails;
      defaults to 'false'
      container: containerReference # container to run this job inside
      services: { string: string | container } # container resources to run as a service container
      timeoutInMinutes: nonEmptyString     # how long to run the job before automatically cancelling
      cancelTimeoutInMinutes: nonEmptyString # how much time to give 'run always even if cancelled tasks' before killing them
      variables: # several syntaxes, see specific section
      environment: string # target environment name and optionally a resource name to record the deployment history; format: <environment-name>.<resource-name>
    strategy:
      runOnce:    #rolling, canary are the other strategies that are supported
      deploy:
        steps: [ script | bash | pwsh | powershell | checkout | task | templateReference ]

```

Deployment strategies

When you're deploying application updates, it's important that the technique you use to deliver the update will:

- Enable initialization.
- Deploy the update.
- Route traffic to the updated version.
- Test the updated version after routing traffic.
- In case of failure, run steps to restore to the last known good version.

We achieve this by using lifecycle hooks that can run steps during deployment. Each of the lifecycle hooks resolves into an agent job or a [server job](#) (or a container or validation job in the future), depending on the `pool` attribute. By default, the lifecycle hooks will inherit the `pool` specified by the `deployment` job.

Deployment jobs use the `$(Pipeline.Workspace)` system variable.

Descriptions of lifecycle hooks

`preDeploy` : Used to run steps that initialize resources before application deployment starts.

`deploy` : Used to run steps that deploy your application. Download artifact task will be auto injected only in the `deploy` hook for deployment jobs. To stop downloading artifacts, use `- download: none` or choose specific artifacts to download by specifying [Download Pipeline Artifact task](#).

`routeTraffic` : Used to run steps that serve the traffic to the updated version.

`postRouteTraffic` : Used to run the steps after the traffic is routed. Typically, these tasks monitor the health of the updated version for defined interval.

`on: failure` or `on: success` : Used to run steps for rollback actions or clean-up.

RunOnce deployment strategy

`runOnce` is the simplest deployment strategy wherein all the lifecycle hooks, namely `preDeploy` `deploy`,

`routeTraffic`, and `postRouteTraffic`, are executed once. Then, either `on: success` or `on: failure` is executed.

```
strategy:  
  runOnce:  
    preDeploy:  
      pool: [ server | pool ] # See pool schema.  
      steps:  
        - script: [ script | bash | pwsh | powershell | checkout | task | templateReference ]  
    deploy:  
      pool: [ server | pool ] # See pool schema.  
      steps:  
        ...  
    routeTraffic:  
      pool: [ server | pool ]  
      steps:  
        ...  
    postRouteTraffic:  
      pool: [ server | pool ]  
      steps:  
        ...  
  on:  
    failure:  
      pool: [ server | pool ]  
      steps:  
        ...  
    success:  
      pool: [ server | pool ]  
      steps:  
        ...
```

If you are using self-hosted agents, you can use the workspace clean options to clean your deployment workspace.

```
jobs:  
- deployment: deploy  
  pool:  
    vmImage: 'Ubuntu-16.04'  
    workspace:  
      clean: all  
    environment: staging
```

Rolling deployment strategy

A rolling deployment replaces instances of the previous version of an application with instances of the new version of the application on a fixed set of virtual machines (rolling set) in each iteration.

We currently only support the rolling strategy to VM resources.

For example, a rolling deployment typically waits for deployments on each set of virtual machines to complete before proceeding to the next set of deployments. You could do a health check after each iteration and if a significant issue occurs, the rolling deployment can be stopped.

Rolling deployments can be configured by specifying the keyword `rolling:` under the `strategy:` node. The `strategy.name` variable is available in this strategy block, which takes the name of the strategy. In this case, `rolling`.

```
strategy:  
  rolling:  
    maxParallel: [ number or percentage as x% ]  
    preDeploy:  
      steps:  
        - script: [ script | bash | pwsh | powershell | checkout | task | templateReference ]  
    deploy:  
      steps:  
        ...  
    routeTraffic:  
      steps:  
        ...  
    postRouteTraffic:  
      steps:  
        ...  
  on:  
    failure:  
      steps:  
        ...  
    success:  
      steps:  
        ...
```

All the lifecycle hooks are supported and lifecycle hook jobs are created to run on each VM.

`preDeploy`, `deploy`, `routeTraffic`, and `postRouteTraffic` are executed once per batch size defined by `maxParallel`. Then, either `on: success` or `on: failure` is executed.

With `maxParallel: <# or % of VMs>`, you can control the number/percentage of virtual machine targets to deploy to in parallel. This ensures that the app is running on these machines and is capable of handling requests while the deployment is taking place on the rest of the machines, which reduces overall downtime.

NOTE

There are a few known gaps in this feature. For example, when you retry a stage, it will re-run the deployment on all VMs not just failed targets.

Canary deployment strategy

Canary deployment strategy is an advanced deployment strategy that helps mitigate the risk involved in rolling out new versions of applications. By using this strategy, you can roll out the changes to a small subset of servers first. As you gain more confidence in the new version, you can release it to more servers in your infrastructure and route more traffic to it.

You can only use the canary deployment strategy for Kubernetes resources.

```

strategy:
  canary:
    increments: [ number ]
  preDeploy:
    pool: [ server | pool ] # See pool schema.
    steps:
      - script: [ script | bash | pwsh | powershell | checkout | task | templateReference ]
  deploy:
    pool: [ server | pool ] # See pool schema.
    steps:
      ...
  routeTraffic:
    pool: [ server | pool ]
    steps:
      ...
  postRouteTraffic:
    pool: [ server | pool ]
    steps:
      ...
  on:
    failure:
      pool: [ server | pool ]
      steps:
        ...
    success:
      pool: [ server | pool ]
      steps:
        ...

```

Canary deployment strategy supports the `preDeploy` lifecycle hook (executed once) and iterates with the `deploy`, `routeTraffic`, and `postRouteTraffic` lifecycle hooks. It then exits with either the `success` or `failure` hook.

The following variables are available in this strategy:

- `strategy.name` : Name of the strategy. For example, canary.
- `strategy.action` : The action to be performed on the Kubernetes cluster. For example, deploy, promote, or reject.
- `strategy.increment` : The increment value used in the current interaction. This variable is available only in `deploy`, `routeTraffic`, and `postRouteTraffic` lifecycle hooks.

Examples

RunOnce deployment strategy

The following example YAML snippet showcases a simple use of a deploy job by using the `runOnce` deployment strategy. The example includes a checkout step.

```

jobs:
  # Track deployments on the environment.
  - deployment: DeployWeb
    displayName: deploy Web App
    pool:
      vmImage: 'Ubuntu-16.04'
      # Creates an environment if it doesn't exist.
      environment: 'smarthotel-dev'
    strategy:
      # Default deployment strategy, more coming...
    runOnce:
      deploy:
        steps:
          - checkout: self
          - script: echo my first deployment

```

With each run of this job, deployment history is recorded against the `smarthotel-dev` environment.

NOTE

- It's also possible to create an environment with empty resources and use that as an abstract shell to record deployment history, as shown in the previous example.

The next example demonstrates how a pipeline can refer both an environment and a resource to be used as the target for a deployment job.

```

jobs:
  - deployment: DeployWeb
    displayName: deploy Web App
    pool:
      vmImage: 'Ubuntu-16.04'
      # Records deployment against bookings resource - Kubernetes namespace.
      environment: 'smarthotel-dev.bookings'
    strategy:
    runOnce:
      deploy:
        steps:
          # No need to explicitly pass the connection details.
          - task: KubernetesManifest@0
            displayName: Deploy to Kubernetes cluster
            inputs:
              action: deploy
              namespace: ${k8sNamespace}
              manifests: |
                $(System.ArtifactsDirectory)/manifests/*
              imagePullSecrets: |
                $(imagePullSecret)
              containers: |
                $(containerRegistry)}/${imageRepository}:$(tag)

```

This approach has the following benefits:

- Records deployment history on a specific resource within the environment, as opposed to recording the history on all resources within the environment.
- Steps in the deployment job **automatically inherit** the connection details of the resource (in this case, a Kubernetes namespace, `smarthotel-dev.bookings`), because the deployment job is linked to the environment. This is useful in the cases where the same connection detail is set for multiple steps of the job.

Rolling deployment strategy

The rolling strategy for VMs updates up to five targets in each iteration. `maxParallel` will determine the number of targets that can be deployed to, in parallel. The selection accounts for absolute number or percentage of targets that must remain available at any time excluding the targets that are being deployed to. It is also used to determine the success and failure conditions during deployment.

```
jobs:
- deployment: VMDeploy
  displayName: web
  environment:
    name: smarhotel-dev
  resourceType: VirtualMachine
  strategy:
    rolling:
      maxParallel: 5 #for percentages, mention as x%
      preDeploy:
        steps:
          - download: current
            artifact: drop
          - script: echo initialize, cleanup, backup, install certs
    deploy:
      steps:
        - task: IISWebAppDeploymentOnMachineGroup@0
          displayName: 'Deploy application to Website'
          inputs:
            WebSiteName: 'Default Web Site'
            Package: '$(Pipeline.Workspace)/drop/**/*.zip'
    routeTraffic:
      steps:
        - script: echo routing traffic
  postRouteTraffic:
    steps:
      - script: echo health check post-route traffic
  on:
    failure:
      steps:
        - script: echo Restore from backup! This is on failure
    success:
      steps:
        - script: echo Notify! This is on success
```

Canary deployment strategy

In the next example, the canary strategy for AKS will first deploy the changes with 10-percent pods, followed by 20 percent, while monitoring the health during `postRouteTraffic`. If all goes well, it will promote to 100 percent.

```

jobs:
- deployment:
  environment: smarthotel-dev.bookings
  pool:
    name: smarthotel-devPool
  strategy:
    canary:
      increments: [10,20]
      preDeploy:
        steps:
          - script: initialize, cleanup...
    deploy:
      steps:
        - script: echo deploy updates...
  -task:KubernetesManifest@0
  inputs:
    action:$(strategy.action)
    namespace:'default'
    strategy:$(strategy.name)
    percentage:$(strategy.increment)
    manifests:'manifest.yml'
    postRouteTraffic:
      pool: server
      steps:
        - script: echo monitor application health...
  on:
    failure:
      steps:
        - script: echo clean-up, rollback...
    success:
      steps:
        - script: echo checks passed, notify...

```

Use pipeline decorators to inject steps automatically

[Pipeline decorators](#) can be used in deployment jobs to auto-inject any custom step (for example, vulnerability scanner) to every [lifecycle hook](#) execution of every deployment job. Since pipeline decorators can be applied to all pipelines in an organization, this can be leveraged as part of enforcing safe deployment practices.

In addition, deployment jobs can be run as a [container job](#) along with [services side-car](#) if defined.

Support for output variables

Define output variables in a deployment job's [lifecycle hooks](#) and consume them in other downstream steps and jobs within the same stage.

To share variables between stages, output an [artifact](#) in one stage and then consume it in a subsequent stage, or use the `stageDependencies` syntax described in [variables](#).

While executing deployment strategies, you can access output variables across jobs using the following syntax.

- For `runOnce` strategy:

```
[$dependencies.<job-name>.outputs['<lifecycle-hookname>.<step-name>.<variable-name>']] (for example,
[$dependencies.JobA.outputs['Deploy.StepA.VariableA']] )
```

- For `runOnce` strategy plus a `resourceType`:

```
[$dependencies.<job-name>.outputs['<lifecycle-hookname>_<resource-name>.<step-name>.<variable-name>']] .
(for example, [$dependencies.JobA.outputs['Deploy_VM1.StepA.VariableA']] )
```

- For `canary` strategy:

```
[$dependencies.<job-name>.outputs['<lifecycle-hookname>_<increment-value>.<step-name>.<variable-name>']]
```

- For `rolling` strategy:

```
$[dependencies.<job-name>.outputs['<lifecycle-hookname>_<resource-name>.<step-name>.<variable-name>']]
```

```
# Set an output variable in a lifecycle hook of a deployment job executing canary strategy.
- deployment: A
  pool:
    vmImage: 'ubuntu-16.04'
  environment: staging
  strategy:
    canary:
      increments: [10,20] # Creates multiple jobs, one for each increment. Output variable can be
referenced with this.
    deploy:
      steps:
        - bash: echo "##vso[task.setvariable variable=myOutputVar;isOutput=true]this is the deployment
variable value"
          name: setvarStep
        - bash: echo $(setvarStep.myOutputVar)
          name: echovar

# Map the variable from the job.
- job: B
  dependsOn: A
  pool:
    vmImage: 'ubuntu-16.04'
  variables:
    myVarFromDeploymentJob: $[ dependencies.A.outputs['deploy_10.setvarStep.myOutputVar'] ]
  steps:
    - script: "echo $(myVarFromDeploymentJob)"
      name: echovar
```

For a `runOnce` job, specify the name of the job instead of the lifecycle hook:

```
# Set an output variable in a lifecycle hook of a deployment job executing runOnce strategy.
- deployment: A
  pool:
    vmImage: 'ubuntu-16.04'
  environment: staging
  strategy:
    runOnce:
      deploy:
        steps:
          - bash: echo "##vso[task.setvariable variable=myOutputVar;isOutput=true]this is the deployment
variable value"
            name: setvarStep
          - bash: echo $(setvarStep.myOutputVar)
            name: echovar

# Map the variable from the job.
- job: B
  dependsOn: A
  pool:
    vmImage: 'ubuntu-16.04'
  variables:
    myVarFromDeploymentJob: $[ dependencies.A.outputs['A.setvarStep.myOutputVar'] ]
  steps:
    - script: "echo $(myVarFromDeploymentJob)"
      name: echovar
```

When you define an environment in a deployment job, the syntax of the output variable varies depending on how the environment gets defined. In this example, `env1` uses shorthand notation and `env2` includes the full syntax with a defined resource type.

```

stages:
- stage: StageA
  jobs:
    - deployment: A1
      pool:
        vmImage: 'ubuntu-16.04'
      environment: env1
      strategy:
        runOnce:
          deploy:
            steps:
              - bash: echo "##vso[task.setvariable variable=myOutputVar;isOutput=true]this is the deployment variable value"
                name: setvarStep
              - bash: echo $(System.JobName)
      - deployment: A2
        pool:
          vmImage: 'ubuntu-16.04'
        environment:
          name: env2
          resourceType: virtualmachine
        strategy:
          runOnce:
            deploy:
              steps:
                - script: echo "##vso[task.setvariable variable=myOutputVarTwo;isOutput=true]this is the second deployment variable value"
                  name: setvarStepTwo

- job: B1
  dependsOn: A1
  pool:
    vmImage: 'ubuntu-16.04'
  variables:
    myVarFromDeploymentJob: $[ dependencies.A1.outputs['A1.setvarStep.myOutputVar'] ]

  steps:
    - script: "echo $(myVarFromDeploymentJob)"
      name: echovar

- job: B2
  dependsOn: A2
  pool:
    vmImage: 'ubuntu-16.04'
  variables:
    myVarFromDeploymentJob: $[ dependencies.A2.outputs['A2.setvarStepTwo.myOutputVar'] ]
    myOutputVarTwo: $[ dependencies.A2.outputs['Deploy_vmsfortesting.setvarStepTwo.myOutputVarTwo'] ]

  steps:
    - script: "echo $(myOutputVarTwo)"
      name: echovartwo

```

When you output a variable from a deployment job, referencing it from the next job uses different syntax depending on if you want to set a variable or use it as a condition for the stage.

```

stages:
- stage: StageA
  jobs:
    - job: A1
      steps:
        - pwsh: echo "##vso[task.setvariable variable=RunStageB;isOutput=true]true"
          name: setvarStep
        - bash: echo $(System.JobName)

- stage: StageB
  dependsOn:
    - StageA

# when referring to another stage, stage name is included in variable path
condition: eq(dependencies.StageA.outputs['A1.setvarStep.RunStageB'], 'true')

# Variables reference syntax differs slightly from inter-stage condition syntax
variables:
  myOutputVar: $[stageDependencies.StageA.A1.outputs['setvarStep.RunStageB']]
jobs:
- deployment: B1
  pool:
    vmImage: 'ubuntu-16.04'
  environment: envB
  strategy:
    runOnce:
      deploy:
        steps:
          - bash: echo $(myOutputVar)

```

Learn more about how to [set a multi-job output variable](#)

FAQ

My pipeline is stuck with the message "Job is pending...". How can I fix this?

This can happen when there is a name conflict between two jobs. Verify that any deployment jobs in the same stage have a unique name and that job and stage names do not contain keywords. If renaming does not fix the problem, review [troubleshooting pipeline runs](#).

Use a decorator to inject steps into a pipeline

3/8/2021 • 4 minutes to read • [Edit Online](#)

Azure DevOps Services | Azure DevOps Server 2020

TIP

Check out our newest documentation on extension development using the [Azure DevOps Extension SDK](#).

Pipeline decorators let you add steps to the beginning and end of every job. This process is different than adding steps to a single definition because it applies to all pipelines in an organization.

Suppose our organization requires running a virus scanner on all build outputs that could be released. Pipeline authors don't need to remember to add that step. We create a decorator that automatically injects the step. Our pipeline decorator injects a custom task that does virus scanning at the end of every pipeline job.

Author a pipeline decorator

This example assumes you're familiar with the [contribution models](#).

Start by [creating an extension](#). After you follow the tutorial, you'll have a `vss-extension.json` file. In this file, add contribution for our new pipeline decorator.

`vss-extension.json`

```
{
  "manifestVersion": 1,
  "contributions": [
    {
      "id": "my-required-task",
      "type": "ms.azure-pipelines.pipeline-decorator",
      "targets": [
        "ms.azure-pipelines-agent-job.post-job-tasks"
      ],
      "properties": {
        "template": "my-decorator.yml"
      }
    }
  ],
  "files": [
    {
      "path": "my-decorator.yml",
      "addressable": true,
      "contentType": "text/plain"
    }
  ]
}
```

Contribution options

Let's take a look at the properties and what they're used for:

PROPERTY	DESCRIPTION
----------	-------------

PROPERTY	DESCRIPTION
<code>id</code>	Contribution identifier. Must be unique among contributions in this extension.
<code>type</code>	Specifies that this contribution is a pipeline decorator. Must be the string <code>ms.azure-pipelines.pipeline-decorator</code> .
<code>targets</code>	Decorators can run before your job, after, or both. See the table below for available options.
<code>properties</code>	The only property required is a <code>template</code> . The template is a YAML file included in your extension, which defines the steps for your pipeline decorator. It's a relative path from the root of your extension folder.

Targets

TARGET	DESCRIPTION
<code>ms.azure-pipelines-agent-job.pre-job-tasks</code>	Run before other tasks in a classic build or YAML pipeline. Due to differences in how source code checkout happens, this target runs before checkout in a YAML pipeline but after checkout in a classic build pipeline.
<code>ms.azure-pipelines-agent-job.post-checkout-tasks</code>	Run after the last <code>checkout</code> task in a classic build or YAML pipeline.
<code>ms.azure-pipelines-agent-job.post-job-tasks</code>	Run after other tasks in a classic build or YAML pipeline.
<code>ms.azure-release-pipelines-agent-job.pre-job-tasks</code>	Run before other tasks in a classic RM pipeline.
<code>ms.azure-release-pipelines-agent-job.post-job-tasks</code>	Run after other tasks in a classic RM pipeline.

In this example, we use `ms.azure-pipelines-agent-job.post-job-tasks` only because we want to run at the end of all build jobs.

This extension contributes a pipeline decorator. Next, we'll create a template YAML file to define the decorator's behavior.

Decorator YAML

In the extension's properties, we chose the name "my-decorator.yml". Create that file in the root of your contribution. It holds the set of steps to run after each job. We'll start with a basic example and work up to the full task.

my-decorator.yml (initial version)

```
steps:
- task: CmdLine@2
  displayName: 'Run my script (injected from decorator)'
  inputs:
    script: dir
```

Installing the decorator

To add a pipeline decorator to your organization, you must [install an extension](#). Only private extensions can contribute pipeline decorators. The extension must be authored and shared with your organization before it can be used.

Once the extension has been shared with your organization, [search for the extension](#) and install it.

Save the file, then [build and install the extension](#). Create and run a basic pipeline. The decorator automatically injects our `dir` script at the end of every job. A pipeline run looks similar to:

```
✓ Prepare job · succeeded
✓ Initialize job · succeeded
✓ Checkout · succeeded
✓ Command Line Script · succeeded
✓ Bash Script · succeeded
✓ Run my script (injected from decorator) · succeeded
✓ Post-job: Checkout · succeeded
✓ Report build status · succeeded
```

NOTE

The decorator runs on every job in every pipeline in the organization. In later steps, we'll add logic to control when and how the decorator runs.

Conditional injection

In our example, we only need to run the virus scanner if the build outputs might be released to the public. Let's say that only builds from the default branch (typically `master`) are ever released. We should limit the decorator to jobs running against the default branch.

The updated file looks like this:

my-decorator.yml (revised version)

```
steps:
- ${{ if eq(resources.repositories['self'].ref, resources.repositories['self'].defaultBranch) }}:
  - script: dir
    displayName: 'Run my script (injected from decorator)'
```

You can start to see the power of this extensibility point. Use the context of the current job to conditionally inject steps at runtime. Use [YAML expressions](#) to make decisions about what steps to inject and when. See [pipeline](#)

[decorator expression context](#) for a full list of available data.

There's another condition we need to consider: what if the user already included the virus scanning step? We shouldn't waste time running it again. In this simple example, we'll pretend that any `script` task found in the job is running the virus scanner. (In a real implementation, you'd have a custom task to check for that instead.)

The script task's ID is `d9bafed4-0b18-4f58-968d-86655b4d2ce9`. If we see another script task, we shouldn't inject ours.

my-decorator.yml (final version)

```
steps:  
- ${{ if and(eq(resources.repositories['self'].ref, resources.repositories['self'].defaultBranch),  
not(containsValue(job.steps.*.task.id, 'd9bafed4-0b18-4f58-968d-86655b4d2ce9')) )}}:  
  - script: dir  
    displayName: 'Run my script (injected from decorator)'
```

Debugging

While authoring your pipeline decorator, you'll likely need to debug. You also may want to see what data you have available in the context.

You can set the `system.debugContext` variable to `true` when you queue a pipeline. Then, look at the pipeline summary page.

You see something similar to the following image:

✓	Prepare job	· succeeded
✓	Initialize job	· succeeded
⌚	Pipeline decorator context (macOS/Linux)	· skipped ⓘ
✓	Pipeline decorator context (Windows)	· succeeded
✓	Checkout	· succeeded
✓	Command Line Script	· succeeded
✓	Virus Scan (injected from decorator)	· succeeded
✓	Post-job: Checkout	· succeeded
✓	Report build status	· succeeded

Select the task to see the logs, which report the available context is available and runtime values.

Helpful Links

Learn more about [YAML expression syntax](#).

Pipeline decorator expression context

3/8/2021 • 3 minutes to read • [Edit Online](#)

Azure DevOps Services

Pipeline decorators have access to context about the pipeline in which they run. As a pipeline decorator author, you can use this context to make decisions about the decorator's behavior. The information available in context is different for pipelines and for release. Also, decorators run after task names are resolved to task GUIDs. When your decorator wants to reference a task, it should use the GUID rather than the name or keyword.

TIP

Check out our newest documentation on extension development using the [Azure DevOps Extension SDK](#).

Resources

Pipeline resources are available on the `resources` object.

Repositories

Currently, there's only one key: `repositories`. `repositories` is a map from repo ID to information about the repository.

In a designer build, the primary repo alias is `__designer_repo`. In a YAML pipeline, the primary repo is called `self`. In a release pipeline, repositories aren't available. [Release artifact variables](#) are available.

For example, to print the name of the `self` repo in a YAML pipeline:

```
steps:
- script: echo ${{ resources.repositories['self'].name }}
```

Repositories contain these properties:

```
resources['repositories']['self'] =
{
  "alias": "self",
  "id": "<repo guid>",
  "type": "Git",
  "version": "<commit hash>",
  "name": "<repo name>",
  "project": "<project guid>",
  "defaultBranch": "<default ref of repo, like 'refs/heads/main'>",
  "ref": "<current pipeline ref, like 'refs/heads/topic'>",
  "versionInfo": {
    "author": "<author of tip commit>",
    "message": "<commit message of tip commit>"
  },
  "checkoutOptions": {}
}
```

Job

Job details are available on the `job` object.

The data looks similar to:

```

job =
{
  "steps": [
    {
      "environment": null,
      "inputs": {
        "script": "echo hi"
      },
      "type": "Task",
      "task": {
        "id": "d9bafed4-0b18-4f58-968d-86655b4d2ce9",
        "name": "CmdLine",
        "version": "2.146.1"
      },
      "condition": null,
      "continueOnError": false,
      "timeoutInMinutes": 0,
      "id": "5c09f0b5-9bc3-401f-8cfb-09c716403f48",
      "name": "CmdLine",
      "displayName": "CmdLine",
      "enabled": true
    }
  ]
}

```

For instance, to conditionally add a task only if it doesn't already exist:

```

- ${{ if not(containsValue(job.steps.*.task.id, 'f3ab91e7-bed6-436a-b651-399a66fe6c2a')) }}:
  - script: echo conditionally inserted

```

Variables

[Pipeline variables](#) are also available.

For instance, if the pipeline had a variable called `myVar`, its value would be available to the decorator as `variables['myVar']`.

For example, to give a decorator an opt-out, we could look for a variable. Pipeline authors who wish to opt out of the decorator can set this variable, and the decorator won't be injected. If the variable isn't present, then the decorator is injected as usual.

my-decorator.yml

```

- ${{ if ne(variables['skipInjecting'], 'true') }}
  - script: echo Injected the decorator

```

Then, in a pipeline in the organization, the author can request the decorator not to inject itself.

pipeline-with-opt-out.yml

```

variables:
  skipInjecting: true
steps:
- script: echo This is the only step. No decorator is added.

```

Task names and GUIDs

Decorators run after tasks have already been turned into GUIDs. Consider the following YAML:

```

steps:
- checkout: self
- bash: echo This is the Bash task
- task: PowerShell@2
inputs:
  targetType: inline
  script: Write-Host This is the PowerShell task

```

Each of those steps maps to a task. Each task has a unique GUID. Task names and keywords map to task GUIDs before decorators run. If a decorator wants to check for the existence of another task, it must search by task GUID rather than by name or keyword.

For normal tasks (which you specify with the `task` keyword), you can look at the task's `task.json` to determine its GUID. For special keywords like `checkout` and `bash` in the previous example, you can use the following GUIDs:

KEYWORD	GUID	TASK NAME
<code>checkout</code>	<code>6D15AF64-176C-496D-B583-FD2AE21D4DF4</code>	n/a, see note below
<code>bash</code>	<code>6C731C3C-3C68-459A-A5C9-BDE6E6595B5B</code>	Bash
<code>script</code>	<code>D9BAFED4-0B18-4F58-968D-86655B4D2CE9</code>	CmdLine
<code>powershell</code>	<code>E213FF0F-5D5C-4791-802D-52EA3E7BE1F1</code>	PowerShell
<code>pwsh</code>	<code>E213FF0F-5D5C-4791-802D-52EA3E7BE1F1</code>	PowerShell
<code>publish</code>	<code>ECDC45F6-832D-4AD9-B52B-EE49E94659BE</code>	PublishPipelineArtifact
<code>download</code>	<code>61F2A582-95AE-4948-B34D-A1B3C4F6A737</code>	DownloadPipelineArtifact

After resolving task names and keywords, the previous YAML becomes:

```

steps:
- task: 6D15AF64-176C-496D-B583-FD2AE21D4DF4@1
  inputs:
    repository: self
- task: 6C731C3C-3C68-459A-A5C9-BDE6E6595B5B@3
  inputs:
    targetType: inline
    script: echo This is the Bash task
- task: E213FF0F-5D5C-4791-802D-52EA3E7BE1F1@2
  inputs:
    targetType: inline
    script: Write-Host This is the PowerShell task

```

TIP

Each of these GUIDs can be found in the `task.json` for the corresponding [in-box task](#). The only exception is `checkout`, which is a native capability of the agent. Its GUID is built into the Azure Pipelines service and agent.

Specify conditions

5/3/2021 • 6 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017.3

You can specify the conditions under which each stage, job, or step runs. By default, a job or stage runs if it does not depend on any other job or stage, or if all of the jobs or stages that it depends on have completed and succeeded. By default, a step runs if nothing in its job has failed yet and the step immediately preceding it has finished. You can customize this behavior by forcing a stage, job, or step to run even if a previous dependency fails or by specifying a custom condition.

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

- [YAML](#)
- [Classic](#)

You can specify conditions under which a step, job, or stage will run.

- Only when all previous dependencies have succeeded. This is the default if there is not a condition set in the YAML.
- Even if a previous dependency has failed, unless the run was canceled. Use `succeededOrFailed()` in the YAML for this condition.
- Even if a previous dependency has failed, even if the run was canceled. Use `always()` in the YAML for this condition.
- Only when a previous dependency has failed. Use `failed()` in the YAML for this condition.
- Custom conditions

By default, steps, jobs, and stages run if all previous steps/jobs have succeeded. It's as if you specified "condition: succeeded()" (see [Job status functions](#)).

```
jobs:  
- job: Foo  
  
  steps:  
    - script: echo Hello!  
      condition: always() # this step will always run, even if the pipeline is canceled  
  
    - job: Bar  
      dependsOn: Foo  
      condition: failed() # this job will only run if Foo fails
```

You can also use variables in conditions.

```

variables:
  isMain: ${eq(variables['Build.SourceBranch'], 'refs/heads/main')}

stages:
- stage: A
  jobs:
    - job: A1
      steps:
        - script: echo Hello Stage A!

- stage: B
  condition: and(succeeded(), eq(variables.isMain, true))
  jobs:
    - job: B1
      steps:
        - script: echo Hello Stage B!
        - script: echo $(isMain)

```

Conditions are evaluated to decide whether to start a stage, job, or step. This means that nothing computed at runtime inside that unit of work will be available. For example, if you have a job which sets a variable using a runtime expression using `$[]` syntax, you can't use that variable in your custom condition.

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 in YAML pipelines. 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, job, or stage should 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 [cancel timeout](#) so that these kinds of tasks have enough time to complete after the user cancels a run.

Examples

Run for the main branch, if succeeding

```
and(succeeded(), eq(variables['Build.SourceBranch'], 'refs/heads/main'))
```

Run if the branch is not main, if succeeding

```
and(succeeded(), ne(variables['Build.SourceBranch'], 'refs/heads/main'))
```

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.

Run if a variable is null

```
variables:  
- name: testNull  
  value: ''  
  
jobs:  
- job: A  
  steps:  
  - script: echo testNull is blank  
    condition: eq('${{ variables.testNull }}', '')
```

Use a template parameter as part of a condition

When you declare a parameter in the same pipeline that you have a condition, parameter expansion happens before conditions are considered. In this case, you can embed parameters inside conditions. The script in this YAML file will run because `parameters.doThing` is true.

```
parameters:  
- name: doThing  
  default: true  
  type: boolean  
  
steps:  
- script: echo I did a thing  
  condition: and(succeeded(), eq('${{ parameters.doThing }}', true))
```

However, when you pass a parameter to a template, the parameter will not have a value when the condition gets evaluated. As a result, if you set the parameter value in both the template and the pipeline YAML files, the pipeline value from the template will get used in your condition.

```
# parameters.yml  
parameters:  
- name: doThing  
  default: false # value passed to the condition  
  type: boolean  
  
jobs:  
- job: B  
  steps:  
  - script: echo I did a thing  
    condition: and(succeeded(), eq('${{ parameters.doThing }}', true))
```

```

# azure-pipeline.yml
parameters:
- name: doThing
  default: true # will not be evaluated in time
  type: boolean

trigger:
- none

extends:
  template: parameters.yml

```

Use the output variable from a job in a condition in a subsequent job

You can make a variable available to future jobs and specify it in a condition. Variables available to future jobs must be marked as [multi-job output variables](#) using `isOutput=true`.

```

jobs:
- job: Foo
  steps:
    - bash: |
        echo "This is job Foo."
        echo "##vso[task.setvariable variable=doThing;isOutput=true]Yes" #set variable doThing to Yes
        name: DetermineResult
- job: Bar
  dependsOn: Foo
  condition: eq(dependencies.Foo.outputs['DetermineResult.doThing'], 'Yes') #map doThing and check the value
  steps:
    - script: echo "Job Foo ran and doThing is Yes."

```

FAQ

I've got a conditional step that runs even when a job is canceled. Does my conditional step affect a job that I canceled in the queue?

No. If you cancel a job while it's in the queue, then the entire job is canceled, including conditional steps.

I've got a conditional step that should run even when the deployment is canceled. How do I specify this?

If you defined the pipelines using a YAML file, then this is supported. This scenario is not yet supported for release pipelines.

How can I trigger a job if a previous job succeeded with issues?

You can use the result of the previous job. For example, in this YAML file, the condition

`eq(dependencies.A.result, 'SucceededWithIssues')` allows the job to run because Job A succeeded with issues.

```

jobs:
- job: A
  displayName: Job A
  continueOnError: true # next job starts even if this one fails
  steps:
    - script: echo Job A ran
    - script: exit 1

- job: B
  dependsOn: A
  condition: eq(dependencies.A.result, 'SucceededWithIssues') # targets the result of the previous job
  displayName: Job B
  steps:
    - script: echo Job B ran

```

I've got a conditional step that runs even when a job is canceled. How do I manage to cancel all jobs at once?

You'll experience this issue if the condition that's configured in the stage doesn't include a job status check function. To resolve the issue, add a job status check function to the condition. If you cancel a job while it's in the queue, the entire job is canceled, including all the other stages, with this function configured. For more information, see [Job status functions](#).

```
stages:
- stage: Stage1
  displayName: Stage 1
  dependsOn: []
  condition: and(contains(variables['build.sourceBranch'], 'refs/heads/main'), succeeded())
  jobs:
    - job: ShowVariables
      displayName: Show variables
      steps:
        - task: CmdLine@2
          displayName: Show variables
          inputs:
            script: 'printenv'

- stage: Stage2
  displayName: stage 2
  dependsOn: Stage1
  condition: contains(variables['build.sourceBranch'], 'refs/heads/main')
  jobs:
    - job: ShowVariables
      displayName: Show variables 2
      steps:
        - task: CmdLine@2
          displayName: Show variables 2
          inputs:
            script: 'printenv'

- stage: Stage3
  displayName: stage 3
  dependsOn: Stage2
  condition: and(contains(variables['build.sourceBranch'], 'refs/heads/main'), succeeded())
  jobs:
    - job: ShowVariables
      displayName: Show variables 3
      steps:
        - task: CmdLine@2
          displayName: Show variables 3
          inputs:
            script: 'printenv'
```

Related articles

- [Specify jobs in your pipeline](#)
- [Add stages, dependencies, & conditions](#)

Specify demands

3/6/2021 • 2 minutes to read • [Edit Online](#)

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Use demands to make sure that the capabilities your pipeline needs are present on the agents that run it.

Demands are asserted automatically by tasks or manually by you.

NOTE

Demands and capabilities are designed for use with self-hosted agents so that jobs can be matched with an agent that meets the requirements of the job. When using Microsoft-hosted agents, you select an image for the agent that matches the requirements of the job, so although it is possible to add capabilities to a Microsoft-hosted agent, you don't need to use capabilities with Microsoft-hosted agents.

Task demands

Some tasks won't run unless one or more demands are met by the agent. For example, the [Visual Studio Build](#) task demands that `msbuild` and `visualstudio` are installed on the agent.

Manually entered demands

You might need to use self-hosted agents with special capabilities. For example, your pipeline may require **SpecialSoftware** on agents in the `Default` pool. Or, if you have multiple agents with different operating systems in the same pool, you may have a pipeline that requires a Linux agent.

- [YAML](#)
- [Classic](#)

To add a single demand to your YAML build pipeline, add the `demands:` line to the `pool` section.

```
pool:  
  name: Default  
  demands: SpecialSoftware # Check if SpecialSoftware capability exists
```

Or if you need to add multiple demands, add one per line.

```
pool:  
  name: MyPool  
  demands:  
    - myCustomCapability # check for existence of capability  
    - agent.os -equals Darwin # check for specific string in capability
```

NOTE

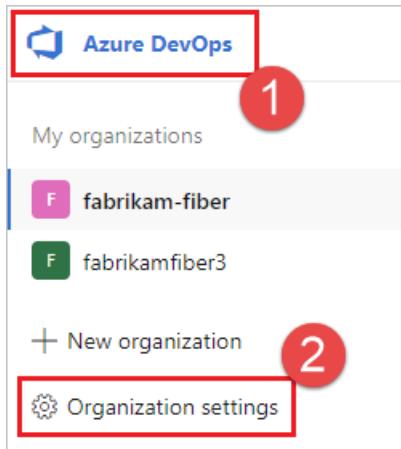
Checking for the existence of a capability (exists) and checking for a specific string in a capability (equals) are the only two supported operations for demands.

For more information and examples, see [YAML schema - Demands](#).

Register each agent that has the capability.

1. In your web browser, navigate to Agent pools:

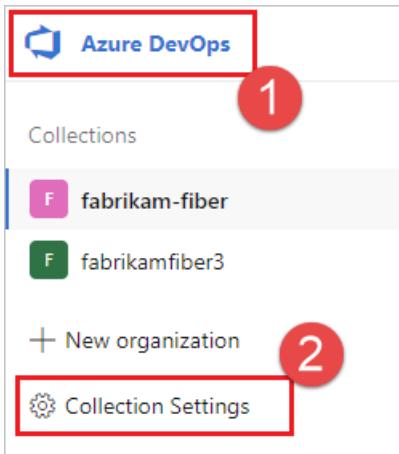
1. Choose Azure DevOps, Organization settings.



2. Choose Agent pools.

The screenshot shows the 'Agent pools' page in the Azure DevOps interface. The navigation bar at the top includes 'FabrikamFiber / Organization Settings / Agent pools'. On the left, a sidebar menu lists 'Security', 'Boards', 'Pipelines', and 'Agent pools', with 'Agent pools' highlighted by a red box and a red circle with the number '1'. The main content area displays a table titled 'Agent pools' with columns for 'Name'. It lists four entries: 'Azure Pipelines' (with a cloud icon), 'Default' (with a computer monitor icon), 'Test' (with a computer monitor icon), and 'Test2' (with a computer monitor icon). The 'Test' and 'Test2' rows are partially blurred.

1. Choose Azure DevOps, Collection settings.



2. Choose Agent pools.

Azure DevOps

FabrikamFiber / Organization Settings / Agent pools

Security

Policies

Permissions

Boards

Process

Pipelines

Agent pools

Settings

Deployment pools

Parallel jobs

Agent pools

Name	Type
Azure Pipelines	Azure Pipelines
Default	Hosted
Test	Hosted
Test2	Hosted

1. Choose Azure DevOps, Collection settings.

Azure DevOps

Collections

fabrikam-fiber

fabrikamfiber3

+ New organization

Collection Settings

2. Choose Agent pools.

Azure DevOps FabrikamFiber / Organization Settings / Agent pools

Security
Policies
Permissions

Boards
Process

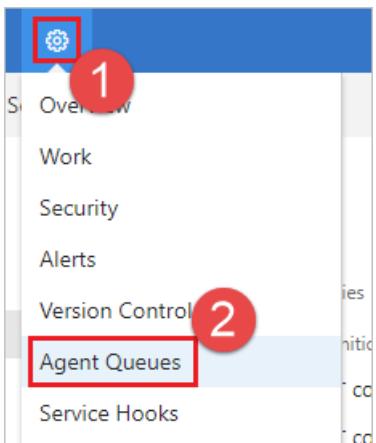
Pipelines

Agent pools **(Red Box)**
Settings
Deployment pools
Parallel jobs

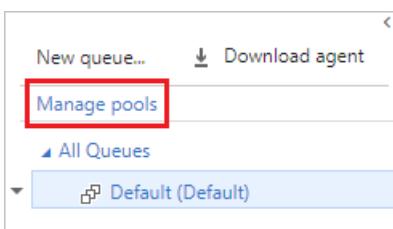
Agent pools

Name
Azure Pipelines
Default
Test
Test2

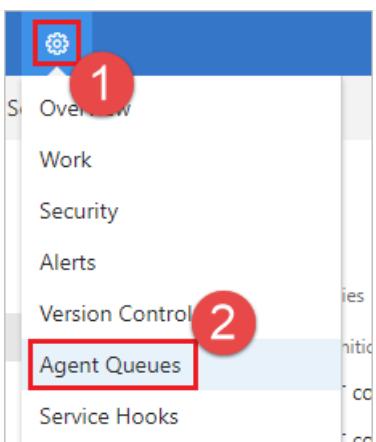
1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



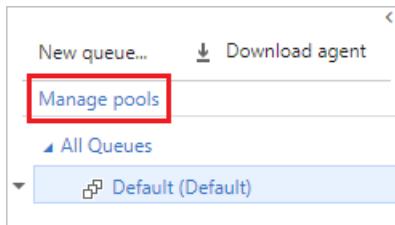
2. Choose **Manage pools**.



1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



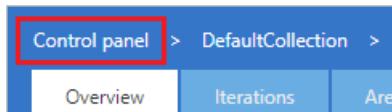
2. Choose **Manage pools**.



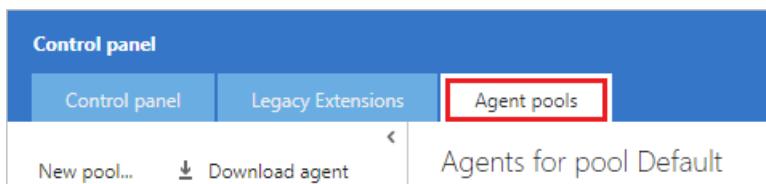
1. Navigate to your project and choose **Manage project** (gear icon).



2. Choose **Control panel**.



3. Select **Agent pools**.



2. Navigate to the capabilities tab for the agent:

1. From the **Agent pools** tab, select the desired agent pool.

A screenshot of the Azure DevOps interface showing the 'Agent pools' page. On the left is a sidebar with a red circle labeled '1' over the 'Agent pools' item. The main area shows a table of agent pools with a red circle labeled '2' over the 'Default' row. The table has columns for Name, Image, and Status. The 'Default' pool is selected.

Name	Image	Status
Azure Pipelines	Cloud icon	Azure Pipelines
Default	Mobile icon	(highlighted)
Test	Mobile icon	
Test2	Mobile icon	

2. Select **Agents** and choose the desired agent.

The screenshot shows the 'Default' pool settings page. At the top, there are tabs: 'Jobs', 'Agents' (which is highlighted with a red box and circled with a red number '1'), 'Details', 'Security', and 'Settings'. Below the tabs, there's a table with two rows. The first row contains the name '160724-AT14-SP3' and the status 'Offline'. The second row contains the name '20160317-W10' and the status 'Offline'. A red box labeled '2' highlights the first row.

3. Choose the **Capabilities** tab.

The screenshot shows the '160724-AT14-SP3' agent settings page. At the top, the agent name is displayed. Below it, the 'Capabilities' tab is selected (highlighted with a red box and circled with a red number '1'). The 'User-defined capabilities' section has a '+ Add new capability' button. The 'System capabilities' section lists the following properties:

Name	Value
Agent.Name	160724-AT14-SP3
Agent.Version	2.107.0
Agent.ComputerName	160724-AT14-SP3

NOTE

Microsoft-hosted agents don't display system capabilities. For a list of software installed on Microsoft-hosted agents, see [Use a Microsoft-hosted agent](#).

1. From the **Agent pools** tab, select the desired pool.

Azure DevOps FabrikamFiber / Organization Settings / Agent pools

Security
Policies
Permissions
Boards
Process
Pipelines
Agent pools (1)
Settings
Deployment pools
Parallel jobs

Agent pools

Name
Azure Pipelines (2)
Default (2)
Test
Test2

2. Select **Agents** and choose the desired agent.

Default

Jobs Agents Details Security Settings (1)

Name
160724-AT14-SP3 (2)
● Offline
20160317-W10
● Offline

3. Choose the **Capabilities** tab.

160724-AT14-SP3

Jobs Capabilities

User-defined capabilities

No user-defined capabilities

Add a new capability

System capabilities

Search by keyword

Name	Value
Agent.Name	160724-AT14-SP3
Agent.Version	2.107.0
Agent.ComputerName	160724-AT14-SP3

1. From the Agent pools tab, select the desired pool.

Azure DevOps / FabrikamFiber / Organization Settings / Agent pools

Security

Policies

Permissions

Boards

Process

Pipelines

Agent pools

Settings

Deployment pools

Parallel jobs

1

2

Azure Pipelines

Default

Test

Test2

2. Select Agents and choose the desired agent.

Default

Jobs Agents Details Security Settings

Name
160724-AT14-SP3 ● Offline
20160317-W10 ● Offline

3. Choose the **Capabilities** tab.

160724-AT14-SP3

Jobs Capabilities

User-defined capabilities

No user-defined capabilities

Add a new capability

System capabilities

Name	Value
Agent.Name	160724-AT14-SP3
Agent.Version	2.107.0
Agent.ComputerName	160724-AT14-SP3

Select the desired agent, and choose the **Capabilities** tab.

Agents for pool Default

Agents Roles

Enabled	Name	Current Status	X
<input checked="" type="checkbox"/>	20160317-W10	Idle	X
<input checked="" type="checkbox"/>	DEV15-TFS-RTM	Idle	X
<input checked="" type="checkbox"/>	DEV15VS	Idle	X

Requests Capabilities

USER CAPABILITIES

Shows information about user-defined capabilities supported by this host

+ Add capability

Save changes Undo changes

SYSTEM CAPABILITIES

Shows information about the capabilities provided by this host

AegisRoot	C:\AegisTools
Agent.ComputerName	20160317-W10

Select the desired agent, and choose the **Capabilities** tab.

Agents for pool Default

Enabled	Name	Current Status	X
<input checked="" type="checkbox"/>	20160317-W10	Idle	X
<input checked="" type="checkbox"/>	DEV15-TFS-RTM	Idle	X
<input checked="" type="checkbox"/>	DEV15VS	Idle	X

Requests Capabilities **2**

USER CAPABILITIES
Shows information about user-defined capabilities supported by this host

+ Add capability

Save changes Undo changes

SYSTEM CAPABILITIES
Shows information about the capabilities provided by this host

AegisRoot	C:\AegisTools
Agent.ComputerName	20160317-W10

From the Agent pools tab, select the desired agent, and choose the Capabilities tab.

Control panel

Control panel Legacy Extensions **1** Agent pools

New pool... Download agent

All Pools **2** Default

Enabled	Name	Current Status	X
<input checked="" type="checkbox"/>	Agent-TFSDEV14UP...	Idle	X

Requests Capabilities **4**

USER CAPABILITIES
Shows information about user-defined capabilities supported by this host

+ Add capability

Save changes Undo changes

SYSTEM CAPABILITIES
Shows information about the capabilities provided by this host

3. Add something like the following entry:

FIRST BOX	SECOND BOX
SpecialSoftware	C:\Program Files (x86)\SpecialSoftware

TIP

For classic non-YAML build definitions, when you manually queue a build you can change the demands on that run.

Library

4/16/2021 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Library is a collection of 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 TFS 2017 and earlier. They are available to build and release pipelines in TFS 2018 and in Azure Pipelines. Task groups and service connections are available to build and release pipelines in TFS 2015 and newer, and in 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 it does not include **Reader** and **User** permissions and 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

- Get advice on [Stack Overflow](#), and feel free to post your questions, search for answers, or suggest a feature on our [Azure DevOps Developer Community Support](#) page.

Define variables

5/20/2021 • 38 minutes to read • [Edit Online](#)

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Variables give you a convenient way to get key bits of data into various parts of the pipeline. The most common use of variables is to define a value that you can then use in your pipeline. All variables are stored as strings and are mutable. The value of a variable can change from run to run or job to job of your pipeline.

When you define the same variable in multiple places with the same name, the most locally scoped variable wins. So, a variable defined at the job level can override a variable set at the stage level. A variable defined at the stage level will override a variable set at the pipeline root level. A variable set in the pipeline root level will override a variable set in the Pipeline settings UI.

You can use variables with [expressions](#) to conditionally assign values and further customize pipelines.

Variables are different from [runtime parameters](#), which are typed and available during template parsing.

User-defined variables

When you define a variable, you can use [different syntaxes \(macro, template expression, or runtime\)](#) and what syntax you use will determine where in the pipeline your variable will render.

In YAML pipelines, you can set variables at the root, stage, and job level. You can also specify variables outside of a YAML pipeline in the UI. When you set a variable in the UI, that variable can be encrypted and set as secret.

[Secret variables](#) are not automatically decrypted in YAML pipelines and need to be passed to your YAML file with `env:` or a variable at the root level.

User-defined variables can be [set as read-only](#). There are [naming restrictions for variables](#) (example: you can't use `secret` at the start of a variable name).

You can [use a variable group](#) to make variables available across multiple pipelines.

You can use [templates](#) to define variables that are used in multiple pipelines in one file.

System variables

In addition to user-defined variables, Azure Pipelines has system variables with predefined values. If you are using YAML or classic build pipelines, see [predefined variables](#) for a comprehensive list of system variables. If you are using classic release pipelines, see [release variables](#).

System variables are set with their current value when you run the pipeline. Some variables are set automatically. As a pipeline author or end user, you change the value of a system variable before the pipeline is run.

System variables are read-only.

Environment variables

Environment variables are specific to the operating system you are using. They are injected into a pipeline in platform-specific ways. The format corresponds to how environment variables get formatted for your specific scripting platform.

On UNIX systems (macOS and Linux), environment variables have the format `$NAME`. On Windows, the format is `%NAME%` for batch and `$env:NAME` in PowerShell.

System and user-defined variables also get injected as environment variables for your platform. When variables are turned into environment variables, variable names become uppercase, and periods turn into underscores.

For example, the variable name `any.variable` becomes the variable name `$ANY_VARIABLE`.

Variable naming restrictions

User-defined variables can consist of letters, numbers, `.`, and `_` characters. Don't use variable prefixes that are reserved by the system. These are: `endpoint`, `input`, `secret`, and `securefile`. Any variable that begins with one of these strings (regardless of capitalization) will not be available to your tasks and scripts.

Understand variable syntax

Azure Pipelines supports three different ways to reference variables: macro, template expression, and runtime expression. Each syntax can be used for a different purpose and has some limitations.

In a pipeline, template expression variables (`${{ variables.var }}`) get processed at compile time, before runtime starts. Macro syntax variables (`$(var)`) get processed during runtime before a task runs. Runtime expressions (`[$variables.var]`) also get processed during runtime but were designed for use with [conditions](#) and [expressions](#). When you use a runtime expression, it must take up the entire right side of a definition.

In this example, you can see that the template expression still has the initial value of the variable after the variable is updated. The value of the macro syntax variable updates. The template expression value does not change because all template expression variables get processed at compile time before tasks run. In contrast, macro syntax variables are evaluated before each task runs.

```
variables:
- name: one
  value: initialValue

steps:
- script: |
    echo ${{ variables.one }} # outputs initialValue
    echo $(one)
  displayName: First variable pass
- bash: echo '##vso[task.setvariable variable=one]secondValue'
  displayName: Set new variable value
- script: |
    echo ${{ variables.one }} # outputs initialValue
    echo $(one) # outputs secondValue
  displayName: Second variable pass
```

Macro syntax variables

Most documentation examples use macro syntax (`$(var)`). Macro syntax is designed to interpolate variable values into task inputs and into other variables.

Variables with macro syntax get processed before a task executes during runtime. Runtime happens [after template expansion](#). When the system encounters a macro expression, it replaces the expression with the contents of the variable. If there's no variable by that name, then the macro expression is left unchanged. For example, if `$(var)` can't be replaced, `$(var)` won't be replaced by anything.

Macro syntax variables remain unchanged with no value because an empty value like `$()` might mean something to the task you are running and the agent should not assume you want that value replaced. For example, if you use `$(foo)` to reference variable `foo` in a Bash task, replacing all `$()` expressions in the input to the task could break your Bash scripts.

Macro variables are only expanded when they are used for a value, not as a keyword. Values appear on the right side of a pipeline definition. The following is valid: `key: $(value)`. The following isn't valid: `$(key): value`.

Macro variables are not expanded when used to display a job name inline. Instead, you must use the `displayName` property.

NOTE

Variables are only expanded for `stages`, `jobs`, and `steps`. You cannot, for example, use macro syntax inside a `resource` or `trigger`.

In this example, macro syntax is used with Bash, PowerShell, and a script task. The syntax for calling a variable with macro syntax is the same for all three.

```
variables:  
- name: projectName  
  value: contoso  
  
steps:  
- bash: echo $(projectName)  
- powershell: echo $(projectName)  
- script: echo $(projectName)
```

Template expression syntax

You can use template expression syntax to expand both [template parameters](#) and variables (`${{ variables.var }}`). Template variables are processed at compile time, and are replaced before runtime starts. Template expressions are designed for reusing parts of YAML as templates.

Template variables silently coalesce to empty strings when a replacement value isn't found. Template expressions, unlike macro and runtime expressions, can appear as either keys (left side) or values (right side). The following is valid: `${{ variables.key }} : ${{ variables.value }}`.

Runtime expression syntax

You can use runtime expression syntax for variables that are expanded at runtime (`[$variables.var]`). Runtime expression variables silently coalesce to empty strings when a replacement value isn't found. Runtime expressions are designed to be used in the conditions of jobs, to support conditional execution of jobs, or whole stages.

Runtime expression variables are only expanded when they are used for a value, not as a keyword. Values appear on the right side of a pipeline definition. The following is valid: `key: [$variables.value]`. The following isn't valid: `[$variables.key]: value`. The runtime expression must take up the entire right side of a key-value pair. For example, `key: [$variables.value]` is valid but `key: [$variables.value] foo` is not.

SYNTAX	EXAMPLE	WHEN IS IT PROCESSED?	WHERE DOES IT EXPAND IN A PIPELINE DEFINITION?	HOW DOES IT RENDER WHEN NOT FOUND?
macro	<code>\$(var)</code>	runtime before a task executes	value (right side)	prints <code>\$var</code>

Syntax	Example	When is it processed?	Where does it expand in a pipeline definition?	How does it render when not found?
template expression	<code>\${{ variables.var }}</code>	compile time	key or value (left or right side)	empty string
runtime expression	<code>[\$variables.var]</code>	runtime	value (right side)	empty string

What syntax should I use?

Use macro syntax if you are providing input for a task.

Choose a runtime expression if you are working with [conditions](#) and [expressions](#). The exception to this is if you have a pipeline where it will cause a problem for your empty variable to print out. For example, if you have conditional logic that relies on a variable having a specific value or no value. In that case, you should use a runtime expression.

If you are defining a variable in a template, use a template expression.

Set variables in pipeline

- [YAML](#)
- [Classic](#)
- [Azure DevOps CLI](#)

In the most common case, you set the variables and use them within the YAML file. This allows you to track changes to the variable in your version control system. You can also define variables in the pipeline settings UI (see the Classic tab) and reference them in your YAML.

Here's an example that shows how to set two variables, `configuration` and `platform`, and use them later in steps. To use a variable in a YAML statement, wrap it in `$()`. Variables can't be used to define a `repository` in a YAML statement.

```
# Set variables once
variables:
  configuration: debug
  platform: x64

steps:

# Use them once
- task: MSBuild@1
  inputs:
    solution: solution1.sln
    configuration: $(configuration) # Use the variable
    platform: $(platform)

# Use them again
- task: MSBuild@1
  inputs:
    solution: solution2.sln
    configuration: $(configuration) # Use the variable
    platform: $(platform)
```

Variable scopes

In the YAML file, you can set a variable at various scopes:

- At the root level, to make it available to all jobs in the pipeline.
- At the stage level, to make it available only to a specific stage.
- At the job level, to make it available only to a specific job.

When a variable is defined at the top of a YAML, it will be available to all jobs and stages in the pipeline and is a global variable. Global variables defined in a YAML are not visible in the pipeline settings UI.

Variables at the job level override variables at the root and stage level. Variables at the stage level override variables at the root level.

```

variables:
  global_variable: value    # this is available to all jobs

jobs:
- job: job1
  pool:
    vmImage: 'ubuntu-16.04'
  variables:
    job_variable1: value1    # this is only available in job1
  steps:
    - bash: echo $(global_variable)
    - bash: echo $(job_variable1)
    - bash: echo $JOB_VARIABLE1 # variables are available in the script environment too

- job: job2
  pool:
    vmImage: 'ubuntu-16.04'
  variables:
    job_variable2: value2    # this is only available in job2
  steps:
    - bash: echo $(global_variable)
    - bash: echo $(job_variable2)
    - bash: echo $GLOBAL_VARIABLE

```

Specify variables

In the preceding examples, the `variables` keyword is followed by a list of key-value pairs. The keys are the variable names and the values are the variable values.

There is another syntax, useful when you want to use [variable templates](#) or [variable groups](#). This syntax should be used at the root level of a pipeline.

In this alternate syntax, the `variables` keyword takes a list of variable specifiers. The variable specifiers are `name` for a regular variable, `group` for a variable group, and `template` to include a variable template. The following example demonstrates all three.

```

variables:
# a regular variable
- name: myvariable
  value: myvalue
# a variable group
- group: myvariablegroup
# a reference to a variable template
- template: myvariabletemplate.yml

```

Learn more about [variable reuse with templates](#).

Access variables through the environment

Notice that variables are also made available to scripts through environment variables. The syntax for using these environment variables depends on the scripting language.

The name is upper-cased, and the `.` is replaced with the `_`. This is automatically inserted into the process environment. Here are some examples:

- Batch script: `%VARIABLE_NAME%`
- PowerShell script: `$env:VARIABLE_NAME`
- Bash script: `$VARIABLE_NAME`

IMPORTANT

Predefined variables that contain file paths are translated to the appropriate styling (Windows style `C:\foo\` versus Unix style `/foo/`) based on agent host type and shell type. If you are running bash script tasks on Windows, you should use the environment variable method for accessing these variables rather than the pipeline variable method to ensure you have the correct file path styling.

YAML is not supported in TFS.

Set secret variables

- [YAML](#)
- [Classic](#)
- [Azure DevOps CLI](#)

Don't set secret variables in your YAML file. Operating systems often log commands for the processes that they run, and you wouldn't want the log to include a secret that you passed in as an input. Use the script's environment or map the variable within the `variables` block to pass secrets to your pipeline.

You need to set secret variables in the pipeline settings UI for your pipeline. These variables are scoped to the pipeline in which you set them. You can also set [secret variables in variable groups](#).

To set secrets in the web interface, follow these steps:

1. Go to the **Pipelines** page, select the appropriate pipeline, and then select **Edit**.
2. Locate the **Variables** for this pipeline.
3. Add or update the variable.
4. Select the lock icon to store the variable in an encrypted manner.
5. Save the pipeline.

Secret variables are encrypted at rest with a 2048-bit RSA key. Secrets are available on the agent for tasks and scripts to use. Be careful about who has access to alter your pipeline.

IMPORTANT

We make an effort to mask secrets from appearing in Azure Pipelines output, but you still need to take precautions.

Never echo secrets as output. Some operating systems log command line arguments. Never pass secrets on the command line. Instead, we suggest that you map your secrets into environment variables.

We never mask substrings of secrets. If, for example, "abc123" is set as a secret, "abc" isn't masked from the logs. This is to avoid masking secrets at too granular of a level, making the logs unreadable. For this reason, secrets should not contain structured data. If, for example, "{ "foo": "bar" }" is set as a secret, "bar" isn't masked from the logs.

Unlike a normal variable, they are not automatically decrypted into environment variables for scripts. You need to explicitly map secret variables.

The following example shows how to use a secret variable called `mySecret` in PowerShell and Bash scripts.

Unlike a normal pipeline variable, there's no environment variable called `MYSECRET`.

```
variables:
  GLOBAL_MYSECRET: $(mySecret) # this will not work because the secret variable needs to be mapped as env
  GLOBAL_MY_MAPPED_ENV_VAR: $(nonSecretVariable) # this works because it's not a secret.

steps:
  - powershell: |
      Write-Host "Using an input-macro works: $(mySecret)"
      Write-Host "Using the env var directly does not work: $env:MYSECRET"
      Write-Host "Using a global secret var mapped in the pipeline does not work either: $env:GLOBAL_MYSECRET"
      Write-Host "Using a global non-secret var mapped in the pipeline works: $env:GLOBAL_MY_MAPPED_ENV_VAR"
      Write-Host "Using the mapped env var for this task works and is recommended: $env:MY_MAPPED_ENV_VAR"
    env:
      MY_MAPPED_ENV_VAR: $(mySecret) # the recommended way to map to an env variable

  - bash: |
      echo "Using an input-macro works: $(mySecret)"
      echo "Using the env var directly does not work: $MYSECRET"
      echo "Using a global secret var mapped in the pipeline does not work either: $GLOBAL_MYSECRET"
      echo "Using a global non-secret var mapped in the pipeline works: $GLOBAL_MY_MAPPED_ENV_VAR"
      echo "Using the mapped env var for this task works and is recommended: $MY_MAPPED_ENV_VAR"
    env:
      MY_MAPPED_ENV_VAR: $(mySecret) # the recommended way to map to an env variable
```

The output from both tasks in the preceding script would look like this:

```
Using an input-macro works: ***
Using the env var directly does not work:
Using a global secret var mapped in the pipeline does not work either:
Using a global non-secret var mapped in the pipeline works: foo
Using the mapped env var for this task works and is recommended: ***
```

You can also use secret variables outside of scripts. For example, you can map secret variables to tasks using the `variables` definition. This example shows how to use secret variables `$(vmsUser)` and `$(vmsAdminPass)` in an Azure file copy task.

```
variables:
  VMS_USER: $(vmsUser)
  VMS_PASS: $(vmsAdminPass)

pool:
  vmImage: 'ubuntu-latest'

steps:
  - task: AzureFileCopy@4
    inputs:
      SourcePath: 'my/path'
      azureSubscription: 'my-subscription'
      Destination: 'AzureVMs'
      storage: 'my-storage'
      resourceGroup: 'my-rg'
      vmsAdminUserName: $(VMS_USER)
      vmsAdminPassword: $(VMS_PASS)
```

Reference secret variables in variable groups

This example shows how to reference a variable group in your YAML file, and also add variables within the YAML. There are two variables used from the variable group: `user` and `token`. The `token` variable is secret,

and is mapped to the environment variable `$env:MY_MAPPED_TOKEN` so that it can be referenced in the YAML.

This YAML makes a REST call to retrieve a list of releases, and outputs the result.

```
variables:
- group: 'my-var-group' # variable group
- name: 'devopsAccount' # new variable defined in YAML
  value: 'contoso'
- name: 'projectName' # new variable defined in YAML
  value: 'contosoads'

steps:
- task: PowerShell@2
  inputs:
    targetType: 'inline'
    script: |
      # Encode the Personal Access Token (PAT)
      # $env:USER is a normal variable in the variable group
      # $env:MY_MAPPED_TOKEN is a mapped secret variable
      $base64AuthInfo = [Convert]::ToBase64String([Text.Encoding]::ASCII.GetBytes("{0}:{1}" -f
$env:USER,$env:MY_MAPPED_TOKEN))

      # Get a list of releases
      $uri = "https://vsrm.dev.azure.com/$(devopsAccount)/$(projectName)/_apis/release/releases?api-
version=5.1"

      # Invoke the REST call
      $result = Invoke-RestMethod -Uri $uri -Method Get -ContentType "application/json" -Headers
@{Authorization=("Basic {0}" -f $base64AuthInfo)}

      # Output releases in JSON
      Write-Host $result.value
env:
  MY_MAPPED_TOKEN: $(token) # Maps the secret variable $(token) from my-var-group
```

IMPORTANT

By default with GitHub repositories, secret variables associated with your pipeline aren't made available to pull request builds of forks. For more information, see [Contributions from forks](#).

YAML is not supported in TFS.

Share variables across pipelines

To share variables across multiple pipelines in your project, use the web interface. Under **Library**, use [variable groups](#).

Use output variables from tasks

Some tasks define output variables, which you can consume in downstream steps, jobs, and stages. In YAML, you can access variables across jobs and stages by using [dependencies](#).

Some tasks define output variables, which you can consume in downstream steps and jobs within the same stage. In YAML, you can access variables across jobs by using [dependencies](#).

Some tasks define output variables, which you can consume in downstream steps within the same job.

NOTE

By default, each stage in a pipeline depends on the one just before it in the YAML file. If you need to refer to a stage that isn't immediately prior to the current one, you can override this automatic default by adding a `dependsOn` section to the stage.

NOTE

The following examples use standard pipeline syntax. If you're using deployment pipelines, both variable and conditional variable syntax will differ. For information about the specific syntax to use, see [Deployment jobs](#).

- [YAML](#)
- [Classic](#)
- [Azure DevOps CLI](#)

For these examples, assume we have a task called `MyTask`, which sets an output variable called `MyVar`. Learn more about the syntax in [Expressions - Dependencies](#).

Use outputs in the same job

```
steps:  
- task: MyTask@1 # this step generates the output variable  
  name: ProduceVar # because we're going to depend on it, we need to name the step  
- script: echo $(ProduceVar.MyVar) # this step uses the output variable
```

Use outputs in a different job

```
jobs:  
- job: A  
  steps:  
    # assume that MyTask generates an output variable called "MyVar"  
    # (you would learn that from the task's documentation)  
    - task: MyTask@1  
      name: ProduceVar # because we're going to depend on it, we need to name the step  
- job: B  
  dependsOn: A  
  variables:  
    # map the output variable from A into this job  
    varFromA: $[ dependencies.A.outputs['ProduceVar.MyVar'] ]  
  steps:  
    - script: echo $(varFromA) # this step uses the mapped-in variable
```

Use outputs in a different stage

To use the output from a different stage at the job level, you use the `stageDependencies` syntax.

```
stages:
- stage: One
  jobs:
    - job: A
      steps:
        - task: MyTask@1 # this step generates the output variable
          name: ProduceVar # because we're going to depend on it, we need to name the step
- stage: Two
  - job: B
    variables:
      # map the output variable from A into this job
      varFromA: ${stageDependencies.One.A.outputs['ProduceVar.MyVar']}
    steps:
      - script: echo $(varFromA) # this step uses the mapped-in variable
```

You can also pass variables between stages with a file input. To do so, you'll need to define variables in the second stage at the job level and then pass the variables as `env:` inputs.

```
## script-a.sh
echo "##vso[task.setvariable variable=sauce;isOutput=true]crushed tomatoes"
```

```
## script-b.sh
echo 'Hello file version'
echo $skipMe
echo $StageSauce
```

```

## azure-pipelines.yml
stages:

- stage: one
  jobs:
    - job: A
      steps:
        - task: Bash@3
          inputs:
            filePath: 'script-a.sh'
            name: setvar
        - bash: |
          echo "##vso[task.setvariable variable=skipsubsequent;isOutput=true]true"
          name: skipstep

- stage: two
  jobs:
    - job: B
      variables:
        - name: StageSauce
          value: ${ stageDependencies.one.A.outputs['setvar.sauce'] }
        - name: skipMe
          value: ${ stageDependencies.one.A.outputs['skipstep.skipsubsequent'] }
      steps:
        - task: Bash@3
          inputs:
            filePath: 'script-b.sh'
            name: fileversion
            env:
              StageSauce: $(StageSauce) # predefined in variables section
              skipMe: $(skipMe) # predefined in variables section
        - task: Bash@3
          inputs:
            targetType: 'inline'
            script: |
              echo 'Hello inline version'
              echo $(skipMe)
              echo $(StageSauce)

```

List variables

You can list all of the variables in your pipeline with the [az pipelines variable list](#) command. To get started, see [Get started with Azure DevOps CLI](#).

```

az pipelines variable list [--org]
                          [--pipeline-id]
                          [--pipeline-name]
                          [--project]

```

Parameters

- **org**: Azure DevOps organization URL. You can configure the default organization using `az devops configure -d organization=ORG_URL`. Required if not configured as default or picked up using `git config`. Example: `--org https://dev.azure.com/MyOrganizationName/`.
- **pipeline-id**: Required if **pipeline-name** is not supplied. ID of the pipeline.
- **pipeline-name**: Required if **pipeline-id** is not supplied, but ignored if **pipeline-id** is supplied. Name of the pipeline.
- **project**: Name or ID of the project. You can configure the default project using `az devops configure -d project=NAME_OR_ID`. Required if not configured as default or picked up by using `git config`.

Example

The following command lists all of the variables in the pipeline with ID 12 and shows the result in table format.

```
az pipelines variable list --pipeline-id 12 --output table

Name      Allow Override  Is Secret  Value
-----
MyVariable  False          False      platform
NextVariable False          True       platform
Configuration False         False      config.debug
```

Set variables in scripts

A script in your pipeline can define a variable so that it can be consumed by one of the subsequent steps in the pipeline. All variables set by this method are treated as strings. To set a variable from a script, you use a command syntax and print to stdout.

- [YAML](#)
- [Classic](#)
- [Azure DevOps CLI](#)

Set a job-scoped variable from a script

To set a variable from a script, you use the `task.setvariable` [logging command](#). This will update the environment variables for subsequent jobs. Subsequent jobs will have access to the new variable with [macro syntax](#) and in tasks as environment variables.

When `issecret` is set to true, the value of the variable will be saved as secret and masked from the log. For more information on secret variables, see [logging commands](#).

```
steps:
# Create a variable
- bash: |
  echo "##vso[task.setvariable variable=sauce]crushed tomatoes"

# Use the variable
# "$(sauce)" is replaced by the contents of the `sauce` variable by Azure Pipelines
# before handing the body of the script to the shell.
- bash: |
  echo my pipeline variable is $(sauce)
```

Subsequent steps will also have the pipeline variable added to their environment. You cannot use the variable in the step that it is defined.

```
steps:
# Create a variable
# Note that this does not update the environment of the current script.
- bash: |
  echo "##vso[task.setvariable variable=sauce]crushed tomatoes"

# An environment variable called `SAUCE` has been added to all downstream steps
- bash: |
  echo "my environment variable is $SAUCE"
- pwsh: |
  Write-Host "my environment variable is $env:SAUCE"
```

Set a multi-job output variable

If you want to make a variable available to future jobs, you must mark it as an output variable by using `isOutput=true`. Then you can map it into future jobs by using the `$[]` syntax and including the step name that set the variable. Multi-job output variables only work for jobs in the same stage.

To pass variables to jobs in different stages, use the [stage dependencies](#) syntax.

NOTE

By default, each stage in a pipeline depends on the one just before it in the YAML file. Therefore, each stage can use output variables from the prior stage. To access further stages, you will need to alter the dependency graph, for instance, if stage 3 requires a variable from stage 1, you will need to declare an explicit dependency on stage 1.

When you create a multi-job output variable, you should assign the expression to a variable. In this YAML,

`$[dependencies.A.outputs['setvarStep.myOutputVar']]` is assigned to the variable `$(myVarFromJobA)`.

```
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-18.04'
  variables:
    myVarFromJobA: $[ dependencies.A.outputs['setvarStep.myOutputVar'] ] # map in the variable
                                         # remember, expressions require
single quotes
  steps:
    - script: echo $(myVarFromJobA)
      name: echovar
```

If you're setting a variable from one stage to another, use [stageDependencies](#).

```
stages:
- stage: A
  jobs:
    - job: A1
      steps:
        - bash: echo "##vso[task.setvariable variable=myStageOutputVar;isOutput=true]this is a stage output
var"
          name: printvar

- stage: B
  dependsOn: A
  variables:
    myVarfromStageA: $[ stageDependencies.A.A1.outputs['printvar.myStageOutputVar'] ]
  jobs:
    - job: B1
      steps:
        - script: echo $(myVarfromStageA)
```

If you're setting a variable from a [matrix](#) or [slice](#), then, to reference the variable when you access it from a downstream job, you must include:

- The name of the job.
- The step.

```

jobs:

# Set an output variable from a job with a matrix
- job: A
  pool:
    vmImage: 'ubuntu-18.04'
  strategy:
    maxParallel: 2
  matrix:
    debugJob:
      configuration: debug
      platform: x64
    releaseJob:
      configuration: release
      platform: x64
  steps:
    - bash: echo "##vso[task.setvariable variable=myOutputVar;isOutput=true]this is the $(configuration) value"
      name: setvarStep
    - bash: echo ${setvarStep.myOutputVar}
      name: echovar

# Map the variable from the debug job
- job: B
  dependsOn: A
  pool:
    vmImage: 'ubuntu-18.04'
  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-18.04'
    parallel: 2 # Two slices
  steps:
    - bash: 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-18.04'
  variables:
    myVarFromJobsA1: ${dependencies.A.outputs['job1.setvarStep.myOutputVar']}
  steps:
    - script: "echo ${myVarFromJobsA1}"
      name: echovar

```

Be sure to prefix the job name to the output variables of a [deployment](#) job. In this case, the job name is A:

```

jobs:

# Set an output variable from a deployment
- deployment: A
  pool:
    vmImage: 'ubuntu-18.04'
  environment: staging
  strategy:
    runOnce:
      deploy:
        steps:
          - bash: echo "##vso[task.setvariable variable=myOutputVar;isOutput=true]this is the deployment variable value"
            name: setvarStep
          - bash: echo ${setvarStep.myOutputVar}
            name: echovar

# Map the variable from the job for the first slice
- job: B
  dependsOn: A
  pool:
    vmImage: 'ubuntu-18.04'
  variables:
    myVarFromDeploymentJob: ${dependencies.A.outputs['A.setvarStep.myOutputVar']}
  steps:
    - bash: "echo ${myVarFromDeploymentJob}"
      name: echovar

```

YAML is not supported in TFS.

Set variables by using expressions

- [YAML](#)
- [Classic](#)
- [Azure DevOps CLI](#)

You can set a variable by using an expression. We already encountered one case of this to set a variable to the output of another from a previous job.

```

- job: B
  dependsOn: A
  variables:
    myVarFromJobsA1: ${dependencies.A.outputs['job1.setvarStep.myOutputVar']} # remember to use single quotes

```

You can use any of the supported expressions for setting a variable. Here's an example of setting a variable to act as a counter that starts at 100, gets incremented by 1 for every run, and gets reset to 100 every day.

```

jobs:
- job:
  variables:
    a: ${counter(format('{0:yyyyMMdd}', pipeline.startTime), 100)}
  steps:
    - bash: echo $(a)

```

For more information about counters, dependencies, and other expressions, see [expressions](#).

YAML is not supported in TFS.

Configure settable variables for steps

You can define `settableVariables` within a step or specify that no variables can be set.

In this example, the script cannot set a variable.

```
steps:
- script: echo This is a step
  target:
    settableVariables: none
```

In this example, the script allows the variable `sauce` but not the variable `secretSauce`. You'll see a warning on the pipeline run page.

Warnings 1

 Setting variable 'secretSauce' has been disabled by the task or build definition.
SetVars

```
steps:
- bash: |
  echo "##vso[task.setvariable variable=Sauce;]crushed tomatoes"
  echo "##vso[task.setvariable variable=secretSauce;]crushed tomatoes with garlic"
  target:
    settableVariables:
      - sauce
    name: SetVars
- bash:
  echo "Sauce is $(sauce)"
  echo "secretSauce is $(secretSauce)"
  name: OutputVars
```

Allow at queue time

- [YAML](#)
- [Classic](#)
- [Azure DevOps CLI](#)

If a variable appears in the `variables` block of a YAML file, its value is fixed and can't be overridden at queue time. Best practice is to define your variables in a YAML file but there are times when this doesn't make sense. For example, you may want to define a secret variable and not have the variable exposed in your YAML. Or, you may need to manually set a variable value during the pipeline run.

You have two options for defining queue-time values. You can define a variable in the UI and select the option to **Let users override this value when running this pipeline** or you can use [runtime parameters](#) instead. If your variable is not a secret, the best practice is to use [runtime parameters](#).

To set a variable at queue time, add a new variable within your pipeline and select the override option.

Name	myVar
Value	1234
<input type="checkbox"/> Keep this value secret <input checked="" type="checkbox"/> Let users override this value when running this pipeline	

To allow a variable to be set at queue time, make sure the variable doesn't also appear in the `variables` block of a pipeline or job. If you define a variable in both the variables block of a YAML and in the UI, the value in the YAML will have priority.

YAML is not supported in TFS.

Expansion of variables

- [YAML](#)
- [Classic](#)
- [Azure DevOps CLI](#)

When you set a variable with the same name in multiple scopes, the following precedence applies (highest precedence first).

1. Job level variable set in the YAML file
2. Stage level variable set in the YAML file
3. Pipeline level variable set in the YAML file
4. Variable set at queue time
5. Pipeline variable set in Pipeline settings UI

In the following example, the same variable `a` is set at the pipeline level and job level in YAML file. It's also set in a variable group `G`, and as a variable in the Pipeline settings UI.

```

variables:
  a: 'pipeline yaml'

stages:
- stage: one
  displayName: one
  variables:
    - name: a
      value: 'stage yaml'

jobs:
- job: A
  variables:
    - name: a
      value: 'job yaml'
  steps:
    - bash: echo $(a)      # This will be 'job yaml'
  
```

When you set a variable with the same name in the same scope, the last set value will take precedence.

```

stages:
- stage: one
  displayName: Stage One
  variables:
    - name: a
      value: alpha
    - name: a
      value: beta
  jobs:
    - job: I
      displayName: Job I
      variables:
        - name: b
          value: uno
        - name: b
          value: dos
      steps:
        - script: echo ${a} #outputs beta
        - script: echo ${b} #outputs dos

```

NOTE

When you set a variable in the YAML file, don't define it in the web editor as settable at queue time. You can't currently change variables that are set in the YAML file at queue time. If you need a variable to be settable at queue time, don't set it in the YAML file.

Variables are expanded once when the run is started, and again at the beginning of each step. For example:

```

jobs:
- job: A
  variables:
    a: 10
  steps:
    - bash: |
        echo $(a)           # This will be 10
        echo '#vso[task.setvariable variable=a]20'
        echo $(a)           # This will also be 10, since the expansion of $(a) happens before the step
    - bash: echo $(a)     # This will be 20, since the variables are expanded just before the step

```

There are two steps in the preceding example. The expansion of `$(a)` happens once at the beginning of the job, and once at the beginning of each of the two steps.

Because variables are expanded at the beginning of a job, you can't use them in a strategy. In the following example, you can't use the variable `a` to expand the job matrix, because the variable is only available at the beginning of each expanded job.

```

jobs:
- job: A
  variables:
    a: 10
  strategy:
    matrix:
      x:
        some_variable: $(a)    # This does not work

```

If the variable `a` is an output variable from a previous job, then you can use it in a future job.

```
- job: A
  steps:
    - powershell: echo "##vso[task.setvariable variable=a;isOutput=true]10"
      name: a_step

# Map the variable into job B
- job: B
  dependsOn: A
  variables:
    some_variable: ${ dependencies.A.outputs['a_step.a'] }
```

Recursive expansion

On the agent, variables referenced using `$()` syntax are recursively expanded. For example:

```
variables:
  myInner: someValue
  myOuter: $(myInner)

steps:
- script: echo $(myOuter) # prints "someValue"
  displayName: Variable is $(myOuter) # display name is "Variable is someValue"
```

YAML is not supported in TFS.

Use a variable group's secret and nonsecret variables in an Azure Pipeline

5/20/2021 • 5 minutes to read • [Edit Online](#)

In this sample, use the Microsoft Azure DevOps CLI (azure-devops extension) to create an Azure Pipeline that accesses a variable group containing both secret and nonsecret variables.

This script demonstrates three operations:

- Defining a [Azure Pipeline](#) using [YAML](#) files
- Creating a [variable group](#) with nonsecret and secret variables for use in a pipeline
- Running the pipeline using [Azure DevOps CLI](#), which also opens a web page for monitoring the pipeline run's processing and output

Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#).
[Launch Cloud Shell](#)
- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
 - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).
 - When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
 - Run [az version](#) to find the version and dependent libraries that are installed. To upgrade to the latest version, run [az upgrade](#).
- [Sign up for Azure Pipelines](#) to get an Azure DevOps organization.
- [Create an Azure DevOps personal access token](#) (PAT) for authentication.
- Use or create a [GitHub](#) repository to place the Azure Pipeline in.
- In the GitHub Marketplace, [install the Azure Pipelines application](#) so that Azure Pipelines can access your GitHub repository.

Sample script

First, save the following YAML file, which defines the Azure Pipeline, to `azure-pipelines.yml` in the root directory of your local repository. Then add and commit the file in GitHub, and push the file to the remote GitHub repository.

```

parameters:
- name: image
  displayName: 'Pool image'
  default: ubuntu-latest
  values:
  - windows-latest
  - vs2017-win2016
  - ubuntu-latest
  - ubuntu-16.04
  - macOS-latest
  - macOS-10.14
- name: test
  displayName: Run Tests?
  type: boolean
  default: false

variables:
- group: "Contoso Variable Group"
- name: va
  value: ${variables.a}
- name: vb
  value: ${variables.b}
- name: vcontososecret
  value: ${variables.contososecret}

trigger:
- master

pool:
  vmImage: ubuntu-latest

steps:
- script: |
    echo "Hello, world!"
    echo "Pool image: ${{ parameters.image }}"
    echo "Run tests? ${{ parameters.test }}"
    displayName: 'Show runtime parameter values'

- script: |
    echo "a=$(va)"
    echo "b=$(vb)"
    echo "contososecret=$(vcontososecret)"
    echo
    echo "Count up to the value of the variable group's nonsecret variable *a*:"
    for number in {1..$(va)}
    do
      echo "$number"
    done
    echo "Count up to the value of the variable group's nonsecret variable *b*:"
    for number in {1..$(vb)}
    do
      echo "$number"
    done
    echo "Count up to the value of the variable group's secret variable *contososecret*:"
    for number in {1..$(vcontososecret)}
    do
      echo "$number"
    done
  displayName: 'Test variable group variables (secret and nonsecret)'

env:
  SYSTEM_ACCESTOKEN: $(System.AccessToken)

```

After you have published the YAML file in GitHub, replace the placeholders in the following Bash script, and then run the script.

```
#!/bin/bash
```

```

# Provide variable definitions.
devopsToken="<azure-devops-personal-access-token>"
devopsOrg="https://dev.azure.com/<my-azure-devops-account-or-organization-name>"
resourceGroupLocation="<resource-group-location-name-or-id>"
storageAccountLocation="<storage-account-location-name-or-id>"
pipelineName="<my-build>"
githubOrg="<my-github-organization-name>"
githubRepo="<my-github-repository-name>"
repoName="$githubOrg/$githubRepo"
repoType="github"
branch="master"

# Declare other variables.
uniqueId=$RANDOM
resourceGroupName="contoso-storage-rg$uniqueId"
storageAccountName="contosostoracct$uniqueId" # needs to be lowercase
devopsProject="Contoso DevOps Project $uniqueId"
serviceConnectionName="Contoso Service Connection $uniqueId"
variableGroupName="Contoso Variable Group"

# Sign in to Azure CLI and follow the directions. May be unnecessary in some environments.
echo "Sign in. (For Cloud Shell, provide the device login code.)"
az login

# Create a resource group and a storage account.
az group create --name "$resourceGroupName" --location "$resourceGroupLocation"
az storage account create --name "$storageAccountName" \
    --resource-group "$resourceGroupName" --location "$storageAccountLocation"

# Set the environment variable used for Azure DevOps token authentication.
export AZURE_DEVOPS_EXT_GITHUB_PAT=$devopsToken

# Create the Azure DevOps project. Set the default organization and project.
projectId=$(az devops project create \
    --name "$devopsProject" --organization "$devopsOrg" --visibility public --query id)
projectId=${projectId:1:-1} # Just set to GUID; drop enclosing quotes.
az devops configure --defaults organization="$devopsOrg" project="$devopsProject"
pipelineRunUrlPrefix="$devopsOrg/$projectId/_build/results?buildId="

# Create GitHub service connection (requires AZURE_DEVOPS_EXT_GITHUB_PAT to be set).
githubServiceEndpointId=$(az devops service-endpoint github create \
    --name "$serviceConnectionName" --github-url "https://www.github.com/$repoName" --query id)
githubServiceEndpointId=${githubServiceEndpointId:1:-1} # Just set to GUID; drop enclosing quotes.

# Create the pipeline.
pipelineId=$(az pipelines create \
    --name "$pipelineName" \
    --skip-first-run \
    --repository $repoName \
    --repository-type $repoType \
    --branch $branch \
    --service-connection $githubServiceEndpointId \
    --yml-path azure-pipelines.yml \
    --query id)

# Create a variable group with 2 non-secret variables and 1 secret variable.
# (contososecret < a < b). Then run the pipeline.
variableGroupId=$(az pipelines variable-group create \
    --name "$variableGroupName" --authorize true --variables a=12 b=29 --query id)
az pipelines variable-group variable create \
    --group-id $variableGroupId --name contososecret --secret true --value 17
pipelineRunId1=$(az pipelines run --id $pipelineId --open --query id)
echo "Go to the pipeline run's web page to view the output results for the 1st run."
echo "If the web page doesn't automatically appear, go to:"
echo "    ${pipelineRunUrlPrefix}${pipelineRunId1}"
read -p "Press Enter to change the value of one of the variable group's nonsecret variables, then run again:"
```

```

# Change the value of one of the variable group's nonsecret variables.
az pipelines variable-group variable update \
    --group-id $variableGroupId --name a --value 22
pipelineRunId2=$(az pipelines run --id $pipelineId --open --query id)
echo "Go to the pipeline run's web page to view the output results for the 2nd run."
echo "If the web page doesn't automatically appear, go to:"
echo "    ${pipelineRunUrlPrefix}${pipelineRunId2}"
read -p "Press Enter to change the value of the variable group's secret variable, then run once more:"

# Change the value of the variable group's secret variable.
az pipelines variable-group variable update \
    --group-id $variableGroupId --name contososecret --value 35
pipelineRunId3=$(az pipelines run --id $pipelineId --open --query id)
echo "Go to the pipeline run's web page to view the output results for the 3rd run."
echo "If the web page doesn't automatically appear, go to:"
echo "    ${pipelineRunUrlPrefix}${pipelineRunId3}"
read -p "Press Enter to continue:"

```

Clean up resources

After the script sample has been run, the following commands can be used to remove the resource group and all resources associated with it.

```

az pipelines variable-group delete --group-id $variableGroupId --yes
az pipelines delete --id $pipelineId --yes
az devops service-endpoint delete --id $githubServiceEndpointId --yes
az devops project delete --id $projectId --yes
export AZURE_DEVOPS_EXT_GITHUB_PAT=""
az storage account delete --name $storageAccountName --resource-group $resourceGroupName --yes
az group delete --name $resourceGroupName --yes
az devops configure --defaults organization="" project=""

```

Azure CLI references used in this article

- [az devops configure](#)
- [az devops project create](#)
- [az devops project delete](#)
- [az devops service-endpoint github create](#)
- [az group create](#)
- [az group delete](#)
- [az login](#)
- [az pipelines create](#)
- [az pipelines delete](#)
- [az pipelines run](#)
- [az pipelines variable-group create](#)
- [az pipelines variable-group delete](#)
- [az pipelines variable-group variable create](#)
- [az pipelines variable-group variable update](#)
- [az storage account create](#)
- [az storage account delete](#)

Use predefined variables

6/3/2021 • 85 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Variables give you a convenient way to get key bits of data into various parts of your pipeline. This is a list of predefined variables that are available for your use. There may be a few other predefined variables, but they are mostly for internal use.

These variables are automatically set by the system and read-only. (The exceptions are `Build.Clean` and `System.Debug`.)

In YAML pipelines, you can reference predefined variables as environment variables. For example, the variable `Build.ArtifactStagingDirectory` becomes the variable `BUILD_ARTIFACTSTAGINGDIRECTORY`.

For classic pipelines, you can use [release variables](#) in your deploy tasks to share the common information (e.g. — Environment Name, Resource Group, etc).

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 [Clean the local repo on the agent](#).

This variable modifies how the build agent cleans up source. To learn more, see [Clean the local repo on the agent](#).

System.AccessToken

`System.AccessToken` is a special variable that carries the security token used by the running build.

- [YAML](#)
- [Classic](#)

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 script 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)
```

You can configure the default scope for `System.AccessToken` using [build job authorization scope](#).

System.Debug

For more detailed logs to debug pipeline problems, define `System.Debug` and set it to `true`.

1. Edit your pipeline.
2. Select **Variables**.
3. Add a new variable with the name `System.Debug` and value `true`.

[← New variable](#)

Name

`System.Debug`

Value

`true`

4. Save the new variable.

Agent variables (DevOps Services)

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. This variable has the same value as <code>Pipeline.Workspace</code> . For example: <code>/home/vsts/work/1</code>
Agent.ContainerMapping	A mapping from container resource names in YAML to their Docker IDs at runtime. For example: <pre>{ "one_container": { "id": "bdbb357d73a0bd3550a1a5b778b62a4c88ed2051c7802a 0659f1ff6e76910190" }, "another_container": { "id": "82652975109ec494876a8ccb875459c945982952e0a72 ad74c91216707162bb" } }</pre>
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 name of the running job. This will usually be "Job" or "__default", but in multi-config scenarios, will be the configuration.
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"> • <code>Windows_NT</code> • <code>Darwin</code> • <code>Linux</code> <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"> • <code>X86</code> • <code>X64</code> • <code>ARM</code>
Agent.TempDirectory	<p>A temporary folder that is cleaned after each pipeline job. This directory is used by tasks such as .NET Core CLI task to hold temporary items like test results before they are published.</p> <p>For example: <code>/home/vsts/work/_temp</code> for Ubuntu</p>
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> <p>Note: This directory is not guaranteed to be writable by pipeline tasks (eg. when mapped into a container)</p>

Build variables (DevOps Services)

VARIABLE	DESCRIPTION	AVAILABLE IN TEMPLATES?
Build.ArtifactStagingDirectory	<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: 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, and 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>	No
Build.BuildId	The ID of the record for the completed build.	No
Build.BuildNumber	<p>The name of the completed build, also known as the run number. You can specify what is included in this value.</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, and 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>	No
Build.BuildUri	<p>The URL for the build. For example: <code>vstfs:///Build/Build/1430</code>.</p> <p>This variable is agent-scoped, and 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>	No

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, and 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>	No
Build.ContainerId	The ID of the container for your artifact. When you upload an artifact in your pipeline, it is added to a container that is specific for that particular artifact.	No
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>	Yes
Build.DefinitionVersion	The version of the build pipeline.	Yes
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>	Yes
Build.QueuedById	See " How are the identity variables set? ".	Yes

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>ResourceTrigger</code> : The build was triggered by a resource trigger or it was triggered by another build. <p>See Build pipeline triggers, Improve code quality with branch policies.</p>	Yes
Build.Repository.Clean	<p>The value you've selected for Clean in the source repository settings.</p> <p>This variable is agent-scoped, and 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>	No
Build.Repository.LocalPath	<p>The local path on the agent where your source code files are downloaded. For example:</p> <pre>c:\agent_work\1\s</pre> <p>By default, new build pipelines update only the changed files. You can modify how files are downloaded on the Repository tab. Important note: if you only check out one Git repository, this path will be the exact path to the code. If you check out multiple repositories, it will revert to its default value, which is</p> <pre>\$(Pipeline.Workspace)/s .</pre> <p>This variable is agent-scoped, and 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>	No

Build.Repository.ID	<p>The unique identifier of the repository.</p> <p>This won't change, even if the name of the repository does.</p> <p>This variable is agent-scoped, and 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>	No
Build.Repository.Name	<p>The name of the triggering repository.</p> <p>This variable is agent-scoped, and 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>	No
Build.Repository.Provider	<p>The type of the triggering repository.</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, and 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>	No
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, and 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>	No
Build.Repository.Uri	<p>The URL for the triggering repository.</p> <p>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, and 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>	No

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>	Yes
Build.RequestedForEmail	See " How are the identity variables set? ".	Yes
Build.RequestedForId	See " How are the identity variables set? ".	Yes
Build.SourceBranch	<p>The branch of the triggering repo 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> • When your pipeline is triggered by a tag: <code>refs/tags/your-tag-name</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>	Yes

Build.SourceBranchName	<p>The name of the branch in the triggering repo 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>. In <code>refs/heads/feature/tools</code> this value is <code>tools</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, <pre>Gated_2016-06-06_05.20.51.4369;username@live.com</pre> <p>or</p> <pre>myshelveset;username@live.com</pre> <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>	Yes
Build.SourcesDirectory	<p>The local path on the agent where your source code files are downloaded. For example:</p> <pre>c:\agent_work\1\s</pre> <p>By default, new build pipelines update only the changed files.</p> <p>Important note: if you only check out one Git repository, this path will be the exact path to the code. If you check out multiple repositories, it will revert to its default value, which is</p> <pre>\$(Pipeline.Workspace)/s</pre> <p>This variable is agent-scoped, and 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.Repository.LocalPath.</p>	No

Build.SourceVersion	<p>The latest version control change of the triggering repo 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, and 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>	Yes
Build.SourceVersionMessage	<p>The comment of the commit or changeset for the triggering repo. We truncate the message to the first line or 200 characters, whichever is shorter.</p> <p>The Build.SourceVersionMessage corresponds to the message on Build.SourceVersion commit. The Build.SourceVersion commit for a PR build is the merge commit (not the commit on the source branch).</p> <p>This variable is agent-scoped, and 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. Also, this variable is only available on the step level and is neither available in the job nor stage levels (i.e. the message is not extracted until the job had started and checked out the code).</p> <p>Note: This variable is available in TFS 2015.4.</p>	No

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: 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, and 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>	No
Build.Repository.Git.SubmoduleCheckout	<p>The value you've selected for Checkout submodules on the repository tab. With multiple repos checked out, this value tracks the triggering repository's setting.</p> <p>This variable is agent-scoped, and 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>	No
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>	No

Build.TriggeredBy.BuildId	<p>If the build was triggered by another build, then this variable is set to the BuildID of the triggering build. In Classic pipelines, this variable is triggered by a build completion trigger.</p> <p>This variable is agent-scoped, and 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>If you are triggering a YAML pipeline using <code>resources</code>, you should use the resources variables instead.</p>	No
Build.TriggeredBy.DefinitionId	<p>If the build was triggered by another build, then this variable is set to the DefinitionID of the triggering build. In Classic pipelines, this variable is triggered by a build completion trigger.</p> <p>This variable is agent-scoped, and 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>If you are triggering a YAML pipeline using <code>resources</code>, you should use the resources variables instead.</p>	No
Build.TriggeredBy.DefinitionName	<p>If the build was triggered by another build, then this variable is set to the name of the triggering build pipeline. In Classic pipelines, this variable is triggered by a build completion trigger.</p> <p>This variable is agent-scoped, and 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>If you are triggering a YAML pipeline using <code>resources</code>, you should use the resources variables instead.</p>	No
Build.TriggeredBy.BuildNumber	<p>If the build was triggered by another build, then this variable is set to the number of the triggering build. In Classic pipelines, this variable is triggered by a build completion trigger.</p> <p>This variable is agent-scoped, and 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>If you are triggering a YAML pipeline using <code>resources</code>, you should use the resources variables instead.</p>	No

Build.TriggeredBy.ProjectID	<p>If the build was triggered by another build, then this variable is set to ID of the project that contains the triggering build. In Classic pipelines, this variable is triggered by a build completion trigger.</p> <p>This variable is agent-scoped, and 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>If you are triggering a YAML pipeline using <code>resources</code>, you should use the resources variables instead.</p>	No
Common.TestResultsDirectory	<p>The local path on the agent where the test results are created. For example:</p> <pre>c:\agent_work\1\TestResults</pre> <p>This variable is agent-scoped, and 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>	No

Pipeline variables (DevOps Services)

VARIABLE	DESCRIPTION
Pipeline.Workspace	<p>Workspace directory for a particular pipeline. This variable has the same value as <code>Agent.BuildDirectory</code>.</p> <p>For example, <code>/home/vsts/work/1</code>.</p>

Deployment job variables (DevOps Services)

These variables are scoped to a specific [Deployment job](#) and will be resolved only at job execution time.

VARIABLE	DESCRIPTION
Environment.Name	Name of the environment targeted in the deployment job to run the deployment steps and record the deployment history. For example, <code>smarhotel-dev</code> .
Environment.Id	ID of the environment targeted in the deployment job. For example, <code>10</code> .
Environment.ResourceName	Name of the specific resource within the environment targeted in the deployment job to run the deployment steps and record the deployment history. For example, <code>bookings</code> which is a Kubernetes namespace that has been added as a resource to the environment <code>smarhotel-dev</code> .
Environment.ResourceId	ID of the specific resource within the environment targeted in the deployment job to run the deployment steps. For example, <code>4</code> .

Strategy.Name	The name of the deployment strategy: <code>canary</code> , <code>runOnce</code> , or <code>rolling</code> .
Strategy.CycleName	The current cycle name in a deployment. Options are <code>PreIteration</code> , <code>Iteration</code> , or <code>PostIteration</code> .

System variables (DevOps Services)

VARIABLE	DESCRIPTION	AVAILABLE IN TEMPLATES?
System.AccessToken	<p>Use the OAuth token to access the REST API.</p> <p>Use <code>System.AccessToken</code> from YAML scripts.</p> <p>This variable is agent-scoped, and 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>	Yes
System.CollectionId	The GUID of the TFS collection or Azure DevOps organization.	Yes
System.CollectionUri	<p>The URI of the TFS collection or Azure DevOps organization. For example:</p> <pre><code>https://dev.azure.com/fabrikamfiber/</code></pre> <p>.</p>	Yes
System.DefaultWorkingDirectory	<p>The local path on the agent where your source code files are downloaded. For example:</p> <pre><code>c:\agent_work\1\s</code></pre> <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>	No
System.DefinitionId	The ID of the build pipeline.	Yes
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, <code>gates</code> during evaluation of gates, and <code>release</code> for other (Agent and Agentless) jobs.	Yes

System.JobAttempt	Set to 1 the first time this job is attempted, and increments every time the job is retried.	No
System.JobDisplayName	The human-readable name given to a job.	No
System.JobId	A unique identifier for a single attempt of a single job. The value is unique to the current pipeline.	No
System.JobName	The name of the job, typically used for expressing dependencies and accessing output variables.	No
System.PhaseAttempt	<p>Set to 1 the first time this phase is attempted, and increments every time the job is retried.</p> <p>Note: "Phase" is a mostly-redundant concept which represents the design-time for a job (whereas job was the runtime version of a phase). We've mostly removed the concept of "phase" from Azure Pipelines. Matrix and multi-config jobs are the only place where "phase" is still distinct from "job". One phase can instantiate multiple jobs which differ only in their inputs.</p>	No
System.PhaseDisplayName	The human-readable name given to a phase.	No
System.PhaseName	A string-based identifier for a job, typically used for expressing dependencies and accessing output variables.	No
System.StageAttempt	Set to 1 the first time this stage is attempted, and increments every time the job is retried.	No
System.StageDisplayName	The human-readable name given to a stage.	No
System.StageName	A string-based identifier for a stage, typically used for expressing dependencies and accessing output variables.	Yes
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> .	Yes
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).	No

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. This variable is only available in a YAML pipeline if the PR is affected by a branch policy.	No
System.PullRequest.SourceBranch	<p>The branch that is being reviewed in a pull request. For example:</p> <pre>refs/heads/users/raisa/new-feature</pre> <p>for Azure Repos. (This variable is initialized only if the build ran because of a Git PR affected by a branch policy). This variable is only available in a YAML pipeline if the PR is affected by a branch policy.</p>	No
System.PullRequest.SourceRepositoryUri	<p>The URL to the repo that contains the pull request. For example:</p> <pre>https://dev.azure.com/ouraccount/_git/OurProject</pre>	No
System.PullRequest.TargetBranch	<p>The branch that is the target of a pull request. For example:</p> <pre>refs/heads/master</pre> <p>when your repository is in Azure Repos and</p> <pre>master</pre> <p>when your repository is in GitHub. This variable is initialized only if the build ran because of a Git PR affected by a branch policy. This variable is only available in a YAML pipeline if the PR is affected by a branch policy.</p>	No
System.TeamFoundationCollectionUri	<p>The URI of the TFS collection or Azure DevOps organization. For example:</p> <pre>https://dev.azure.com/fabrikamfiber/</pre>	Yes
	<p>This variable is agent-scoped, and 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.	Yes
System.TeamProjectId	The ID of the project that this build belongs to.	Yes
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, and 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>	No

Checks variables (DevOps Services)

VARIABLE	DESCRIPTION
Checks.StageAttempt	<p>Set to 1 the first time this stage is attempted, and increments every time the stage is retried.</p> <p>This variable can only be used within an approval or check for an environment. For example, you could use <code>\$(Checks.StageAttempt)</code> within an Invoke REST API check.</p> <div style="border: 1px solid black; height: 20px; margin-top: 10px;"></div>

Agent variables (DevOps Server 2020)

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. This variable has the same value as <code>Pipeline.Workspace</code>.</p> <p>For example: <code>/home/vsts/work/1</code></p>
Agent.HomeDirectory	<p>The directory the agent is installed into. This contains the agent software. For example: <code>c:\agent</code>.</p>
Agent.Id	<p>The ID of the agent.</p>
Agent.JobName	<p>The name of the running job. This will usually be "Job" or "<code>_default</code>", but in multi-config scenarios, will be the configuration.</p>
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	<p>The name of the machine on which the agent is installed.</p>
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.TempDirectory	<p>A temporary folder that is cleaned after each pipeline job. This directory is used by tasks such as .NET Core CLI task to hold temporary items like test results before they are published.</p> <p>For example: <code>/home/vsts/work/_temp</code> for Ubuntu</p>
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> <p>Note: This directory is not guaranteed to be writable by pipeline tasks (eg. when mapped into a container)</p>

Build variables (DevOps Server 2020)

VARIABLE	DESCRIPTION	AVAILABLE IN TEMPLATES?

Build.ArtifactStagingDirectory	<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: 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, and 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>	No
Build.BuildId	The ID of the record for the completed build.	No
Build.BuildNumber	<p>The name of the completed build, also known as the run number. You can specify what is included in this value.</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, and 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>	No
Build.BuildUri	<p>The URI for the build. For example: <code>vstfs:///Build/Build/1430</code>.</p> <p>This variable is agent-scoped, and 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>	No

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, and 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>	No
Build.ContainerId	The ID of the container for your artifact. When you upload an artifact in your pipeline, it is added to a container that is specific for that particular artifact.	No
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>	Yes
Build.DefinitionVersion	The version of the build pipeline.	Yes
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>	Yes
Build.QueuedById	See " How are the identity variables set? ".	Yes

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>ResourceTrigger</code> : The build was triggered by a resource trigger or it was triggered by another build. <p>See Build pipeline triggers, Improve code quality with branch policies.</p>	Yes
Build.Repository.Clean	<p>The value you've selected for Clean in the source repository settings.</p> <p>This variable is agent-scoped, and 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>	No
Build.Repository.LocalPath	<p>The local path on the agent where your source code files are downloaded. For example:</p> <pre>c:\agent_work\1\s</pre> <p>By default, new build pipelines update only the changed files. You can modify how files are downloaded on the Repository tab. Important note: if you only check out one Git repository, this path will be the exact path to the code. If you check out multiple repositories, it will revert to its default value, which is</p> <pre>\$(Pipeline.Workspace)/s .</pre> <p>This variable is agent-scoped, and 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>	No

Build.Repository.ID	<p>The unique identifier of the repository.</p> <p>This won't change, even if the name of the repository does.</p> <p>This variable is agent-scoped, and 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>	No
Build.Repository.Name	<p>The name of the triggering repository.</p> <p>This variable is agent-scoped, and 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>	No
Build.Repository.Provider	<p>The type of the triggering repository.</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, and 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>	No
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, and 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>	No
Build.Repository.Uri	<p>The URL for the triggering repository.</p> <p>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, and 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>	No

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>	Yes
Build.RequestedForEmail	See " How are the identity variables set? ".	Yes
Build.RequestedForId	See " How are the identity variables set? ".	Yes
Build.SourceBranch	<p>The branch of the triggering repo 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> • When your pipeline is triggered by a tag: <code>refs/tags/your-tag-name</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>	Yes

Build.SourceBranchName	<p>The name of the branch in the triggering repo 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>. In <code>refs/heads/feature/tools</code> this value is <code>tools</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, <pre>Gated_2016-06-06_05.20.51.4369;username@live.com</pre> <p>or</p> <pre>myshelveset;username@live.com</pre> <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>	Yes
Build.SourcesDirectory	<p>The local path on the agent where your source code files are downloaded. For example:</p> <pre>c:\agent_work\1\s</pre> <p>By default, new build pipelines update only the changed files. You can modify how files are downloaded on the Repository tab. Important note: if you only check out one Git repository, this path will be the exact path to the code. If you check out multiple repositories, it will revert to its default value, which is</p> <pre>\$(Pipeline.Workspace)/s</pre> . <p>This variable is agent-scoped, and 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.Repository.LocalPath.</p>	No

Build.SourceVersion	<p>The latest version control change of the triggering repo 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, and 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>	Yes
Build.SourceVersionMessage	<p>The comment of the commit or changeset for the triggering repo. We truncate the message to the first line or 200 characters, whichever is shorter.</p> <p>This variable is agent-scoped, and 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. Also, this variable is only available on the step level and is neither available in the job nor stage levels (i.e. the message is not extracted until the job had started and checked out the code).</p> <p>Note: This variable is available in TFS 2015.4.</p>	No
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: 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, and 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>	No

Build.Repository.Git.SubmoduleCheckout	<p>The value you've selected for Checkout submodules on the repository tab. With multiple repos checked out, this value tracks the triggering repository's setting.</p> <p>This variable is agent-scoped, and 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>	No
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>	No
Build.TriggeredBy.BuildId	<p>If the build was triggered by another build, then this variable is set to the BuildID of the triggering build. In Classic pipelines, this variable is triggered by a build completion trigger.</p> <p>This variable is agent-scoped, and 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>	No
Build.TriggeredBy.DefinitionId	<p>If the build was triggered by another build, then this variable is set to the DefinitionID of the triggering build. In Classic pipelines, this variable is triggered by a build completion trigger.</p> <p>This variable is agent-scoped, and 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>	No
Build.TriggeredBy.DefinitionName	<p>If the build was triggered by another build, then this variable is set to the name of the triggering build pipeline. In Classic pipelines, this variable is triggered by a build completion trigger.</p> <p>This variable is agent-scoped, and 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>	No

Build.TriggeredBy.BuildNumber	<p>If the build was triggered by another build, then this variable is set to the number of the triggering build. In Classic pipelines, this variable is triggered by a build completion trigger.</p> <p>This variable is agent-scoped, and 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>	No
Build.TriggeredBy.ProjectID	<p>If the build was triggered by another build, then this variable is set to ID of the project that contains the triggering build. In Classic pipelines, this variable is triggered by a build completion trigger.</p> <p>This variable is agent-scoped, and 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>	No
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, and 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>	No

Pipeline variables (DevOps Server 2020)

VARIABLE	DESCRIPTION
Pipeline.Workspace	<p>Workspace directory for a particular pipeline. This variable has the same value as <code>Agent.BuildDirectory</code>.</p> <p>For example, <code>/home/vsts/work/1</code>.</p>

Deployment job variables (DevOps Server 2020)

These variables are scoped to a specific [Deployment job](#) and will be resolved only at job execution time.

VARIABLE	DESCRIPTION
Environment.Name	Name of the environment targeted in the deployment job to run the deployment steps and record the deployment history. For example, <code>smarthotel-dev</code> .
Environment.Id	ID of the environment targeted in the deployment job. For example, <code>10</code> .

Environment.ResourceName	Name of the specific resource within the environment targeted in the deployment job to run the deployment steps and record the deployment history. For example, <code>bookings</code> which is a Kubernetes namespace that has been added as a resource to the environment <code>smarthotel-dev</code> .
Environment.ResourceId	ID of the specific resource within the environment targeted in the deployment job to run the deployment steps. For example, <code>4</code> .

System variables (DevOps Server 2020)

VARIABLE	DESCRIPTION	AVAILABLE IN TEMPLATES?
System.AccessToken	<p>Use the OAuth token to access the REST API.</p> <p>Use <code>System.AccessToken</code> from YAML scripts.</p> <p>This variable is agent-scoped, and 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>	Yes
System.CollectionId	The GUID of the TFS collection or Azure DevOps organization	Yes
System.CollectionUri	A string Team Foundation Server collection URI.	Yes
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>	No
System.DefinitionId	The ID of the build pipeline.	Yes
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, <code>gates</code> during evaluation of gates, and <code>release</code> for other (Agent and Agentless) jobs.	Yes

System.JobAttempt	Set to 1 the first time this job is attempted, and increments every time the job is retried.	No
System.JobDisplayName	The human-readable name given to a job.	No
System.JobId	A unique identifier for a single attempt of a single job. The value is unique to the current pipeline.	No
System.JobName	The name of the job, typically used for expressing dependencies and accessing output variables.	No
System.PhaseAttempt	<p>Set to 1 the first time this phase is attempted, and increments every time the job is retried.</p> <p>Note: "Phase" is a mostly-redundant concept which represents the design-time for a job (whereas job was the runtime version of a phase). We've mostly removed the concept of "phase" from Azure Pipelines. Matrix and multi-config jobs are the only place where "phase" is still distinct from "job". One phase can instantiate multiple jobs which differ only in their inputs.</p>	No
System.PhaseDisplayName	The human-readable name given to a phase.	No
System.PhaseName	A string-based identifier for a job, typically used for expressing dependencies and accessing output variables.	No
System.StageAttempt	Set to 1 the first time this stage is attempted, and increments every time the job is retried.	No
System.StageDisplayName	The human-readable name given to a stage.	No
System.StageName	A string-based identifier for a stage, typically used for expressing dependencies and accessing output variables.	Yes
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> .	Yes
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).	No

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. This variable is only available in a YAML pipeline if the PR is affected by a branch policy.	No
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). This variable is only available in a YAML pipeline if the PR is affected by a branch policy.	No
System.PullRequest.SourceRepositoryUri	The URL to the repo that contains the pull request. For example: <code>https://dev.azure.com/ouraccount/_git/OurProject</code>	No
System.PullRequest.TargetBranch	The branch that is the target of a pull request. For example: <code>refs/heads/master</code> when your repository is in Azure Repos and <code>master</code> when your repository is in GitHub. This variable is initialized only if the build ran because of a Git PR affected by a branch policy . This variable is only available in a YAML pipeline if the PR is affected by a branch policy.	No
System.TeamFoundationCollectionUri	The URI of the team foundation collection. For example: <code>https://dev.azure.com/fabrikamfiber/</code> This variable is agent-scoped, and 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.	Yes
System.TeamProject	The name of the project that contains this build.	Yes
System.TeamProjectId	The ID of the project that this build belongs to.	Yes
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, and 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>	No

Agent variables (DevOps Server 2019)

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 name of the running job. This will usually be "Job" or " <code>_default</code> ", but in multi-config scenarios, will be the configuration.
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. If you are using a self-hosted agent, then this name is specified by you. See agents .
Agent.OS	The operating system of the agent host. Valid values are: <ul style="list-style-type: none">• Windows_NT• Darwin• Linux If you're running in a container, the agent host and container may be running different operating systems.
Agent.OSArchitecture	The operating system processor architecture of the agent host. Valid values are: <ul style="list-style-type: none">• X86• X64• ARM
Agent.TempDirectory	A temporary folder that is cleaned after each pipeline job. This directory is used by tasks such as .NET Core CLI task to hold temporary items like test results before they are published.

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> <p>This directory is not guaranteed to be writable by pipeline tasks (eg. when mapped into a container)</p>

Build variables (DevOps Server 2019)

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: <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.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 part of the build number or as a version control tag.</p>

Build.BuildUri	<p>The URI for the build. For example: <code>vstfs:///Build/Build/1430</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.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	<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	<p>See "How are the identity variables set?".</p>
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 <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://dev.azure.com/fabrikamfiber/_git/Scripts</code> TFVC: <code>https://dev.azure.com/fabrikamfiber/</code>
Build.RequestedFor	<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.RequestedForEmail	<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 <code>label format</code> will fail.</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 (/) 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>. In <code>refs/heads/feature/tools</code> this value is <code>tools</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. We truncate the message to the first line or 200 characters, whichever is shorter.</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>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.SourceTvcShelveset	<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 BuildID 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	<p>If the build was triggered by another build, then this variable is set to the DefinitionID 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.DefinitionName	<p>If the build was triggered by another build, then this variable is set to the name of the triggering build pipeline.</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.BuildNumber	<p>If the build was triggered by another build, then this variable is set to the number 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.ProjectID	<p>If the build was triggered by another build, then this variable is set to ID of the project that contains 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>
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 (DevOps Server 2019)

VARIABLE	DESCRIPTION

System.AccessToken	<p>Use the OAuth token to access the REST API.</p> <p>Use <code>System.AccessToken</code> from YAML scripts.</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	<p>The GUID of the TFS collection or Azure DevOps organization</p>
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	<p>The ID of the build pipeline.</p>
System.HostType	<p>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.</p>
System.PullRequest.IsFork	<p>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>.</p>
System.PullRequest.PullRequestId	<p>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.)</p>
System.PullRequest.PullRequestNumber	<p>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.</p>
System.PullRequest.SourceBranch	<p>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.)</p>
System.PullRequest.SourceRepositoryURI	<p>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.)</p>
System.PullRequest.TargetBranch	<p>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.</p>

System.TeamFoundationCollectionUri	The URI of the team foundation collection. For example: <code>https://dev.azure.com/fabrikamfiber/</code> .
System.TeamProject	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.TeamProjectId	The name of the project that contains this build.
TF_BUILD	The ID of the project that this build belongs to. Set to <code>True</code> 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 (TFS 2018)

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.TempDirectory	A temporary folder that is cleaned after each pipeline job. This directory is used by tasks such as .NET Core CLI task to hold temporary items like test results before they are published.
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 (TFS 2018)

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: <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.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: <code>vstfs:///Build/Build/1430</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.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	<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	The version of the build pipeline.
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 from the UI or an API call. • <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 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 <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/Script</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?".</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	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>. In <code>refs/heads/feature/tools</code> this value is <code>tools</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. We truncate the message to the first line or 200 characters, whichever is shorter.</p> <p>This variable is agent-scoped, and 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>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: 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.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></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 (TFS 2018)

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	<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 (TFS 2017)

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: <code>c:\agent_work\1</code></p>
Agent.ComputerName	The name of the machine on which the agent is installed.
Agent.HomeDirectory	<p>The directory the agent is installed into. This contains the agent software. For example: <code>c:\agent</code>.</p>
Agent.Id	The ID of the agent.
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.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: <code>c:\agent_work</code> .

Build variables (TFS 2017)

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: <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: 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: <code>vstfs:///Build/Build/1430</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.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	The event that caused the build to run. Available in TFS 2017.3 . <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. See Build pipeline triggers , Improve code quality with branch policies .
Build.Repository.Clean	The value you've selected for Clean in the source repository settings . 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.Repository.LocalPath	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. This variable is synonymous with <code>Build.SourcesDirectory</code> .
Build.Repository.Name	The name of the repository . 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.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/Script</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?".</p> <p>Note: This value can contain whitespace or other invalid label characters. In these cases, the <code>label format</code> 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 (/) 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>. In <code>refs/heads/feature/tools</code> this value is <code>tools</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. We truncate the message to the first line or 200 characters, whichever is shorter.</p> <p>This variable is agent-scoped, and 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>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: 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.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 (TFS 2017)

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	<p>The URL to the repo that contains the pull request. For example:</p> <pre>http://our-server:8080/tfs/DefaultCollection/_git/OurProject</pre> <p>. (This variable is initialized only if the build ran because of a Azure Repos Git PR affected by a branch policy.)</p>
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:</p> <pre>http://our-server:8080/tfs/DefaultCollection/</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>
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 (TFS 2015)

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 (TFS 2015)

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\a • 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.</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 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:</p> <pre>C:\TfsData\Agents\Agent-MACHINENAME_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	<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	<p>See "How are the identity variables set?".</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:</p> <pre>c:\agent_work\1\s</pre> <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 (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/Script</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?".</p> <p>Note: This value can contain whitespace or other invalid label characters. In these cases, the label format will fail.</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>myshelvset;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>. In <code>refs/heads/feature/tools</code> this value is <code>tools</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	<p>Note: This variable is available in TFS 2015 RTM, but not in TFS 2015.4.</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. We truncate the message to the first line or 200 characters, whichever is shorter.</p> <p>This variable is agent-scoped, and 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>Note: This variable is available in TFS 2015.4.</p>

Build.StagingDirectory

TFS 2015 RTM

The local path on the agent you can use as an output folder for compiled binaries. For example:

C:\TfsData\Build_work\6c3842c6\staging .

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](#).

TFS 2015.4

The local path on the agent where any artifacts are copied to before being pushed to their destination. For example:

C:\TfsData\Agents\Agent-MACHINENAME_work\1\a

This directory is purged before each new build, so you don't have to clean it up yourself.

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](#).

In **TFS 2015.4**, `Build.ArtifactStagingDirectory` and `Build.StagingDirectory` are interchangeable.

All versions of TFS 2015

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.Repository.Git.SubmoduleCheckout

The value you've selected for **Checkout submodules** 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.

Build.SourceTfvcShelveset

Defined if your [repository](#) is Team Foundation Version Control.

If you are running a [gated build](#) or a [shelveset build](#), this is set to the name of the [shelveset](#) you are building.

Note: This variable yields a value that is invalid for build use in a build number format.

Common.TestResultsDirectory

The local path on the agent where the test results are created. For example: C:\agent_work\1\TestResults .

Available in TFS 2015.4.

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 (TFS 2015)

VARIABLE	DESCRIPTION
----------	-------------

System.AccessToken	Available in TFS 2015.4. Use the OAuth token to access the REST API. 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	<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	Set to <code>True</code> 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.

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: <code>[DefaultCollection]\Project Collection Service Accounts</code>	The person who pushed or checked in the changes.
In Git or by a branch policy build .	The system identity, for example: <code>[DefaultCollection]\Project Collection Service Accounts</code>	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: <code>[DefaultCollection]\Project Collection Service Accounts</code>	The system identity, for example: <code>[DefaultCollection]\Project Collection Service Accounts</code>
Because you clicked the Queue build button	You	You

Runtime parameters

6/15/2021 • 6 minutes to read • [Edit Online](#)

Runtime parameters let you have more control over what values can be passed to a pipeline. With runtime parameters you can:

- Supply different values to scripts and tasks at runtime
- Control parameter types, ranges allowed, and defaults
- Dynamically select jobs and stages with template expressions

You can specify [parameters in templates](#) and in the pipeline. Parameters have data types such as number and string, and they can be restricted to a subset of values. The `parameters` section in a YAML defines what parameters are available.

Parameters are only available at template parsing time. Parameters are expanded just before the pipeline runs so that values surrounded by `${{ }}` are replaced with parameter values. Use [variables](#) if you need your values to be more widely available during your [pipeline run](#).

NOTE

This guidance does not apply to classic pipelines. For parameters in classic pipelines, see [Process parameters \(classic\)](#).

Parameters must contain a name and data type. Parameters cannot be optional. A default value needs to be assigned in your YAML file or when you run your pipeline. If you do not assign a default value or set `default` to `false`, the first available value will be used.

Use parameters in pipelines

Set runtime parameters at the beginning of a YAML. This example pipeline accepts the value of `image` and then outputs the value in the job. The `trigger` is set to none so that you can select the value of `image` when you manually trigger your pipeline to run.

```

parameters:
- name: image
  displayName: Pool Image
  type: string
  default: ubuntu-latest
  values:
  - windows-latest
  - vs2017-win2016
  - ubuntu-latest
  - ubuntu-16.04
  - macOS-latest
  - macOS-10.14

trigger: none

jobs:
- job: build
  displayName: build
  pool:
    vmImage: ${{ parameters.image }}
  steps:
  - script: echo building $(Build.BuildNumber) with ${{ parameters.image }}

```

When the pipeline runs, you select the Pool Image. If you do not make a selection, the default option, `ubuntu-latest` gets used.

Run pipeline X

Select parameters below and manually run the pipeline

Branch/tag

⚙️ master
 ▼

Select the branch, commit, or tag

Pool Image

ubuntu-latest
 ▼

- windows-latest
- vs2017-win2016
- ubuntu-latest
- ubuntu-16.04
- macOS-latest
- macOS-10.14

Enable system diagnostics

Cancel
Run

Use conditionals with parameters

You can also use parameters as part of conditional logic. With conditionals, part of a YAML will only run if it meets the `if` criteria.

Use parameters to determine what steps run

This pipeline only runs a step when the boolean parameter `test` is true.

```
parameters:
- name: image
  displayName: Pool Image
  values:
  - windows-latest
  - vs2017-win2016
  - ubuntu-latest
  - ubuntu-16.04
  - macOS-latest
  - macOS-10.14
- name: test
  displayName: Run Tests?
  type: boolean
  default: false

trigger: none

jobs:
- job: build
  displayName: Build and Test
  pool:
    vmImage: ${{ parameters.image }}
  steps:
  - script: echo building $(Build.BuildNumber)
  - ${{ if eq(parameters.test, true) }}:
    - script: echo "Running all the tests"
```

Use parameters to set what configuration is used

You can also use parameters to set which job runs. In this example, a different job runs depending on the value of `config`.

```
parameters:
- name: configs
  type: string
  default: 'x86,x64'

trigger: none

jobs:
- ${{ if contains(parameters.configs, 'x86') }}:
  - job: x86
    steps:
    - script: echo Building x86...
- ${{ if contains(parameters.configs, 'x64') }}:
  - job: x64
    steps:
    - script: echo Building x64...
- ${{ if contains(parameters.configs, 'arm') }}:
  - job: arm
    steps:
    - script: echo Building arm...
```

Selectively exclude a stage

You can also use parameters to set whether a stage runs. In this example, the Performance Test stage runs if the parameter `runPerfTests` is true.

```
parameters:
- name: runPerfTests
  type: boolean
  default: false

trigger: none

stages:
- stage: Build
  displayName: Build
  jobs:
  - job: Build
    steps:
    - script: echo running Build

- stage: UnitTest
  displayName: Unit Test
  dependsOn: Build
  jobs:
  - job: UnitTest
    steps:
    - script: echo running UnitTest

- ${{ if eq(parameters.runPerfTests, true) }}:
- stage: PerfTest
  displayName: Performance Test
  dependsOn: Build
  jobs:
  - job: PerfTest
    steps:
    - script: echo running PerfTest

- stage: Deploy
  displayName: Deploy
  dependsOn: UnitTest
  jobs:
  - job: Deploy
    steps:
    - script: echo running UnitTest
```

Loop through parameters

You can also loop through your string, number, and boolean parameters.

- [Script](#)
- [PowerShell](#)

In this example, you loop through parameters and print out each parameter name and value.

```

# start.yaml
parameters:
- name: myStringName
  type: string
  default: a string value
- name: myMultiString
  type: string
  default: default
  values:
    - default
    - ubuntu
- name: myNumber
  type: number
  default: 2
  values:
    - 1
    - 2
    - 4
    - 8
    - 16
- name: myBoolean
  type: boolean
  default: true

steps:
- ${{ each parameter in parameters }}:
  - script: echo ${{ parameter.Key }}
  - script: echo ${{ parameter.Value }}

```

```

# azure-pipeline.yaml
trigger: none

extends:
  template: start.yaml

```

Check for an empty parameter object

You can use the `length()` expression to check whether an object parameter has no value.

```

parameters:
- name: foo
  type: object
  default: []

steps:
- checkout: none
- ${{ if eq(length(parameters.foo), 0) }}:
  - script: echo Foo is empty
    displayName: Foo is empty

```

Parameter data types

DATA TYPE	NOTES
<code>string</code>	string
<code>number</code>	may be restricted to <code>values:</code> , otherwise any number-like string is accepted

DATA TYPE	NOTES
<code>boolean</code>	<code>true</code> or <code>false</code>
<code>object</code>	any YAML structure
<code>step</code>	a single step
<code>stepList</code>	sequence of steps
<code>job</code>	a single job
<code>jobList</code>	sequence of jobs
<code>deployment</code>	a single deployment job
<code>deploymentList</code>	sequence of deployment jobs
<code>stage</code>	a single stage
<code>stageList</code>	sequence of stages

The `step`, `stepList`, `job`, `jobList`, `deployment`, `deploymentList`, `stage`, and `stageList` data types all use standard YAML schema format. This example includes string, number, boolean, object, step, and stepList.

```

parameters:
- name: myString
  type: string
  default: a string
- name: myMultiString
  type: string
  default: default
  values:
    - default
    - ubuntu
- name: myNumber
  type: number
  default: 2
  values:
    - 1
    - 2
    - 4
    - 8
    - 16
- name: myBoolean
  type: boolean
  default: true
- name: myObject
  type: object
  default:
    foo: FOO
    bar: BAR
    things:
      - one
      - two
      - three
    nested:
      one: apple
      two: pear
      count: 3
- name: myStep
  type: step
  default:
    script: echo my step
- name: mySteplist
  type: stepList
  default:
    - script: echo step one
    - script: echo step two

trigger: none

jobs:
- job: stepList
  steps: ${{ parameters.mySteplist }}
- job: myStep
  steps:
    - ${{ parameters.myStep }}

```

FAQ

How can I use variables inside of templates?

There are times when it may be useful to set parameters to values based on variables. Parameters are expanded early in processing a [pipeline run](#) so not all variables will be available. To see what predefined variables are available in templates, see [Use predefined variables](#).

In this example, the predefined variables `Build.SourceBranch` and `Build.Reason` are used in conditions in `template.yml`.

```
# File: azure-pipelines.yml
trigger:
- main

extends:
  template: template.yml
```

```
# File: template.yml
steps:
- script: echo Build.SourceBranch = $(Build.SourceBranch) # outputs refs/heads/main
- script: echo Build.Reason = $(Build.Reason) # outputs IndividualCI
- ${{ if eq(variables['Build.SourceBranch'], 'refs/heads/main') }}:
  - script: echo I run only if Build.SourceBranch = refs/heads/main
- ${{ if eq(variables['Build.Reason'], 'IndividualCI') }}:
  - script: echo I run only if Build.Reason = IndividualCI
- script: echo I run after the conditions
```

Process parameters

3/6/2021 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

NOTE

This guidance does not apply to YAML pipelines. For parameters in YAML pipelines, see [runtime parameters](#).

You can link all important arguments for tasks used across the build definition as process parameters, which are then shown at one place - the Pipeline view. This means you can quickly edit these arguments without needing to click through all the tasks.

Templates come with a set of predefined process parameters.

Process parameters differ from variables in the kind of input supported by them. Variables only take in string inputs while process parameters in addition to string inputs support more data types like check boxes and drop-down list boxes.

NOTE

The **Link** and **Unlink** functionality applies to build pipelines only. It does not apply to release pipelines.

To link more arguments across all tasks to new or existing process parameters, select **Link** from the task argument.

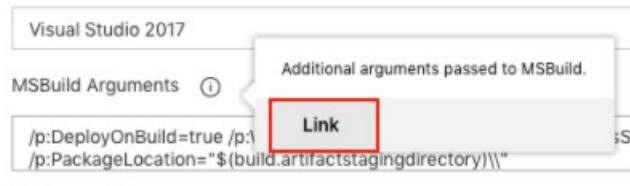
Set a process parameter from a template in a build pipeline

You can create a process parameter from a task input in a build pipeline. Process parameters are useful if you will have the same input values in multiple tasks. You can link more arguments across all tasks to new or existing process parameters.

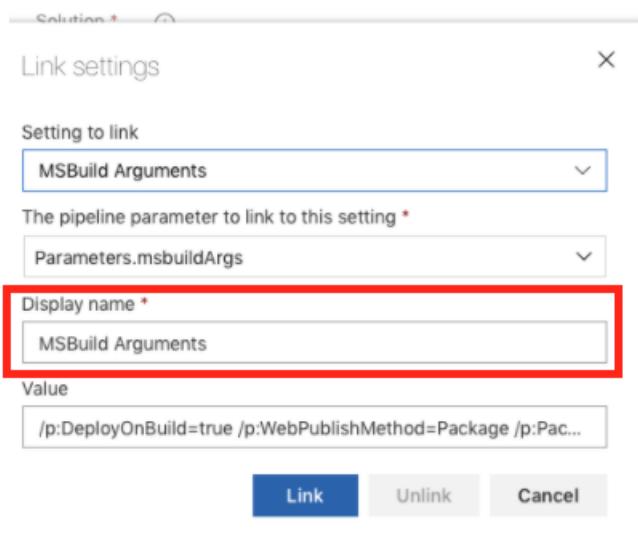
The screenshot shows the Azure DevOps pipeline editor interface. On the left, a list of tasks is visible, including 'Get sources', 'NuGet restore', 'Build solution', 'Test Assemblies' (which is currently selected), 'Publish symbols path:', 'Copy Files to:', 'Publish Artifact: drop', and 'Add Task'. At the top right, there are buttons for 'Save & queue', 'Discard', 'Summary', 'Queue', and more. A context menu is open over the 'Test Assemblies' task, with the 'Link' option highlighted.

To link a process parameter:

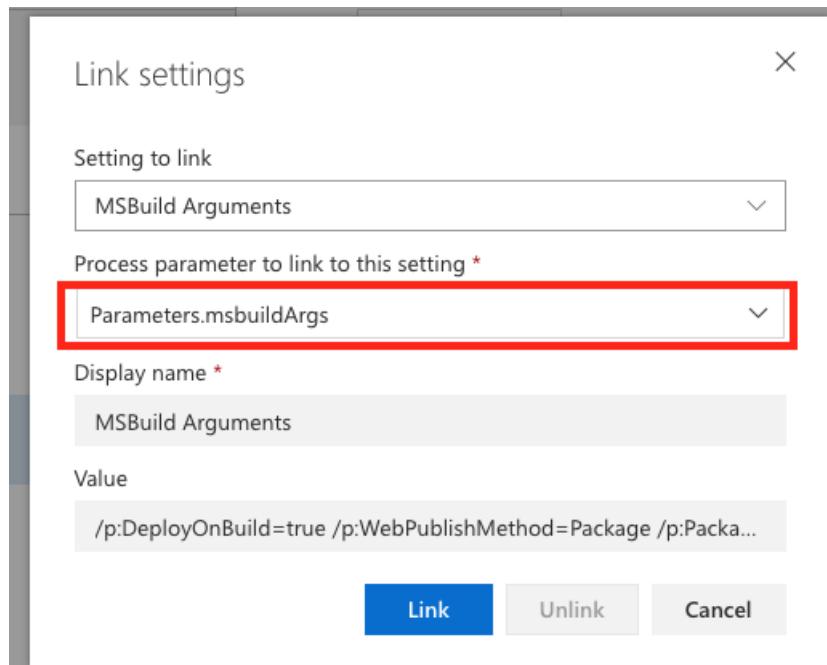
1. Select the *info* icon that is part of a task input field.
2. In the window that opens, select **Link**.



3. Configure the **Link settings**. You can set a new **Display name** so that you can identify the process parameter later on.



4. You'll now have the option to link your process parameter when you reuse the same task. To link a process parameter, select the *info* icon and click **Link**.
5. Set the **Process parameter to link to this setting** to reference your existing process parameter and select **Link**.



The screenshot shows the Azure DevOps interface for a build pipeline named 'FabrikamFibre-Visual Studio-CI'. The 'Tasks' tab is active. On the left, a list of tasks is shown: Get sources, NuGet restore, Build solution, Test Assemblies, Publish symbols path, Copy Files to, Publish Artifact: drop, and Add Task. On the right, the 'Process' section is detailed. It includes fields for 'Name' (set to 'FabrikamFibre-Visual Studio-CI'), 'Parameters' (empty), 'Solution' (set to '***.sln'), 'Assemblies with unit tests' (set to '**\\$(BuildConfiguration)\test*.dll;!**\obj**'), and 'Test Assembly' (highlighted with an orange box and set to '**\test*.dll;!**\obj**'). Other tabs like 'Variables', 'Triggers', 'Options', 'Retention', and 'History' are also visible.

Select **Unlink** if you need to disconnect an argument from a process parameter.

You can also unlink arguments from process parameters.

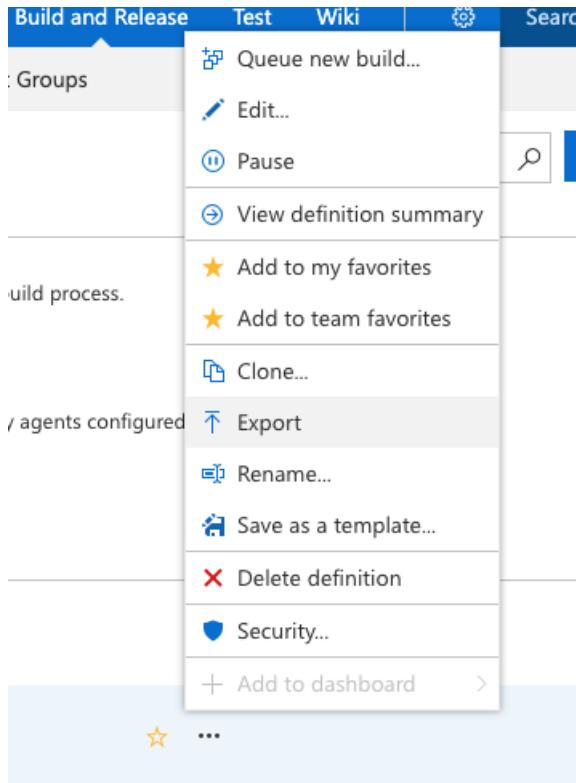
1. Select the link icon.
2. Choose the **Unlink** option. In this example, the `Parameter.solution` process parameter is unlinked.

The screenshot shows the Azure DevOps pipeline editor. On the left, there's a list of tasks: 'Get sources', 'NuGet restore ***.sln', 'Build solution ***.sln', 'Test Assemblies', 'Publish symbols path:', 'Copy Files to: \$(build.artifacts...', 'Publish Artifact: drop', and '+ Add Task'. The 'NuGet restore' task is selected. On the right, the 'NuGet Installer' task configuration is shown. A tooltip for the 'Path to solution or packages.config' field is displayed, containing the text: 'The path to the Visual Studio solution file or NuGet packages.config' and 'This setting is linked to the 'Solution' (Parameters.solution) process parameter.' An orange box highlights the 'Unlink' button.

Define a new process parameter

You can define a new process parameter using the `json` definition for your pipeline.

1. Export the existing template from the Build Definitions page.



2. Open the JSON in a text editor. Add a new input in the process parameter section of your file. If you have existing process parameters, you can duplicate one of them. For example, this is a new parameter definition based on the `msBuildArgs` parameter.

```
{
  "aliases":[
    ],
    "options":{

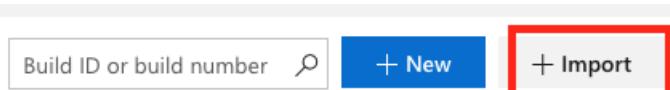
      },
      "properties":{

        },
        "name":"myNewParam",
        "label":"MyProcessParam",
        "defaultValue":"/p:DeployOnBuild=true /p:WebPublishMethod=Package
/p:PackageAsSingleFile=true /p:SkipInvalidConfigurations=true
/p:PackageLocation="\$(build.artifactstagingdirectory)\\\"",
        "type":"string",
        "helpMarkDown":"Additional arguments for process parameters.",
        "visibleRule":"",
        "groupName": ""
  }
}
```

3. Add the new parameter as an input value for any tasks where you want to use it. For example, here the new value of `msbuildArgs` is `myNewParam`.

```
"inputs":{
  "solution": "$(Parameters.solution)",
  "vsVersion": "latest",
  "msbuildArgs": "$(Parameters.myNewParam)",
  "platform": "$(BuildPlatform)",
  "configuration": "$(BuildConfiguration)",
  "clean": "false",
  "maximumCpuCount": "false",
  "restoreNugetPackages": "false",
  "msbuildArchitecture": "x86",
  "logProjectEvents": "true",
  "createLogFile": "false"
}
```

4. Return to the Build Definitions page and select **Import**.



5. Add your `json` file and import the pipeline.
6. Within your imported pipeline, you now have the option to link to your new process parameter. You may need to unlink and then link again for the process parameter to appear.

Classic release and artifacts variables

5/20/2021 • 14 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Classic release and artifacts variables are a convenient way to exchange and transport data throughout your pipeline. Each variable is stored as a string and its value can change between runs of your pipeline.

Variables are different from [Runtime parameters](#) which are only available at template parsing time.

NOTE

This is a reference article that covers the classic release and artifacts variables. To understand variables in YAML pipelines, see [user-defined variables](#).

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**.

TIP

You can view the [current values of all variables](#) for a release, and use a default variable to [run a release in debug mode](#).

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. To view the full list, see [View the current values of all variables](#).

Default variables - System

VARIABLE NAME	DESCRIPTION
System.TeamFoundationServerUri	<p>The URL of the service connection in TFS or Azure Pipelines. Use this from your scripts or tasks to call Azure Pipelines REST APIs.</p> <p>Example: <code>https://fabrikam.vssrm.visualstudio.com/</code></p>
System.TeamFoundationCollectionUri	<p>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.</p> <p>Example: <code>https://dev.azure.com/fabrikam/</code></p>
System.CollectionId	<p>The ID of the collection to which this build or release belongs. Not available in TFS 2015.</p> <p>Example: <code>6c6f3423-1c84-4625-995a-f7f143a1e43d</code></p>
System.DefinitionId	<p>The ID of the release pipeline to which the current release belongs. Not available in TFS 2015.</p> <p>Example: <code>1</code></p>
System.TeamProject	<p>The name of the project to which this build or release belongs.</p> <p>Example: <code>Fabrikam</code></p>
System.TeamprojectId	<p>The ID of the project to which this build or release belongs. Not available in TFS 2015.</p> <p>Example: <code>79f5c12e-3337-4151-be41-a268d2c73344</code></p>
System.ArtifactsDirectory	<p>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.</p> <p>Example: <code>C:\agent_work\r1\a</code></p>
System.DefaultWorkingDirectory	<p>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.</p> <p>Example: <code>C:\agent_work\r1\a</code></p>
System.WorkFolder	<p>The working directory for this agent, where subfolders are created for every build or release. Same as Agent.RootDirectory and Agent.WorkFolder.</p> <p>Example: <code>C:\agent_work</code></p>

VARIABLE NAME	DESCRIPTION
System.Debug	<p>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.</p> <p>Example: <code>true</code></p>

Default variables - Release

VARIABLE NAME	DESCRIPTION
Release.AttemptNumber	<p>The number of times this release is deployed in this stage. Not available in TFS 2015.</p> <p>Example: <code>1</code></p>
Release.DefinitionEnvironmentId	<p>The ID of the stage in the corresponding release pipeline. Not available in TFS 2015.</p> <p>Example: <code>1</code></p>
Release.DefinitionId	<p>The ID of the release pipeline to which the current release belongs. Not available in TFS 2015.</p> <p>Example: <code>1</code></p>
Release.DefinitionName	<p>The name of the release pipeline to which the current release belongs.</p> <p>Example: <code>fabrikam-cd</code></p>
Release.Deployment.RequestedFor	<p>The display name of the identity that triggered (started) the deployment currently in progress. Not available in TFS 2015.</p> <p>Example: <code>Mateo Escobedo</code></p>
Release.Deployment.RequestedForEmail	<p>The email address of the identity that triggered (started) the deployment currently in progress. Not available in TFS 2015.</p> <p>Example: <code>mateo@fabrikam.com</code></p>
Release.Deployment.RequestedForId	<p>The ID of the identity that triggered (started) the deployment currently in progress. Not available in TFS 2015.</p> <p>Example: <code>2f435d07-769f-4e46-849d-10d1ab9ba6ab</code></p>
Release.DeploymentID	<p>The ID of the deployment. Unique per job.</p> <p>Example: <code>254</code></p>
Release.DeployPhaseID	<p>The ID of the phase where deployment is running.</p> <p>Example: <code>127</code></p>

VARIABLE NAME	DESCRIPTION
Release.EnvironmentId	The ID of the stage instance in a release to which the deployment is currently in progress. Example: <code>276</code>
Release.EnvironmentName	The name of stage to which deployment is currently in progress. Example: <code>Dev</code>
Release.EnvironmentUri	The URI of the stage instance in a release to which deployment is currently in progress. Example: <code>vstfs://ReleaseManagement/Environment/276</code>
Release.Environments.{stage-name}.status	The deployment status of the stage. Example: <code>InProgress</code>
Release.PrimaryArtifactSourceAlias	The alias of the primary artifact source Example: <code>fabrikam_web</code>
Release.Reason	The reason for the deployment. Supported values are: <code>ContinuousIntegration</code> - the release started in Continuous Deployment after a build completed. <code>Manual</code> - the release started manually. <code>None</code> - the deployment reason has not been specified. <code>Scheduled</code> - the release started from a schedule.
Release.ReleaseDescription	The text description provided at the time of the release. Example: <code>Critical security patch</code>
Release.ReleaseId	The identifier of the current release record. Example: <code>118</code>
Release.ReleaseName	The name of the current release. Example: <code>Release-47</code>
Release.ReleaseUri	The URI of current release. Example: <code>vstfs://ReleaseManagement/Release/118</code>
Release.ReleaseWebURL	The URL for this release. Example: <code>https://dev.azure.com/fabrikam/f3325c6c/_release?releaseId=392&_a=release-summary</code>
Release.RequestedFor	The display name of identity that triggered the release. Example: <code>Mateo Escobedo</code>

VARIABLE NAME	DESCRIPTION
Release.RequestedForEmail	The email address of identity that triggered the release. Example: <code>mateo@fabrikam.com</code>
Release.RequestedForId	The ID of identity that triggered the release. Example: <code>2f435d07-769f-4e46-849d-10d1ab9ba6ab</code>
Release.SkipArtifactsDownload	Boolean value that specifies whether or not to skip downloading of artifacts to the agent. Example: <code>FALSE</code>
Release.TriggeringArtifact.Alias	The alias of the artifact which triggered the release. This is empty when the release was scheduled or triggered manually. Example: <code>fabrikam_app</code>

Default variables - Release stage

VARIABLE NAME	DESCRIPTION
Release.Environments.{stage name}.Status	The status of deployment of this release within a specified stage. Not available in TFS 2015. Example: <code>NotStarted</code>

Default variables - Agent

VARIABLE NAME	DESCRIPTION
Agent.Name	The name of the agent as registered with the agent pool . This is likely to be different from the computer name. Example: <code>fabrikam-agent</code>
Agent.MachineName	The name of the computer on which the agent is configured. Example: <code>fabrikam-agent</code>
Agent.Version	The version of the agent software. Example: <code>2.109.1</code>
Agent.JobName	The name of the job that is running, such as Release or Build. Example: <code>Release</code>
Agent.HomeDirectory	The folder where the agent is installed. This folder contains the code and resources for the agent. Example: <code>C:\agent</code>

VARIABLE NAME	DESCRIPTION
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. Example: 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. Example: 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. Example: C:\agent_work
Agent.DeploymentGroupId	The ID of the deployment group the agent is registered with. This is available only in deployment group jobs. Not available in TFS 2018 Update 1. Example: 1

Default variables - General Artifact

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 the `{alias}` placeholder with the value you specified for the [artifact alias](#) or with the default value generated for the release pipeline.

VARIABLE NAME	DESCRIPTION
Release.Artifacts.{alias}.DefinitionId	The identifier of the build pipeline or repository. Azure Pipelines example: 1 GitHub example: fabrikam/asp
Release.Artifacts.{alias}.DefinitionName	The name of the build pipeline or repository. Azure Pipelines example: fabrikam-ci TFVC example: \$/Fabrikam Git example: fabrikam GitHub example: fabrikam/asp (main)

VARIABLE NAME	DESCRIPTION
Release.Artifacts.{alias}.BuildNumber	<p>The build number or the commit identifier.</p> <p>Azure Pipelines example: 20170112.1 Jenkins/TeamCity example: 20170112.1 TFVC example: Changeset 3 Git example: 38629c964 GitHub example: 38629c964</p>
Release.Artifacts.{alias}.BuildId	<p>The build identifier.</p> <p>Azure Pipelines example: 130 Jenkins/TeamCity example: 130 GitHub example: 38629c964d21fe405ef830b7d0220966b82c9e11</p>
Release.Artifacts.{alias}.BuildURI	<p>The URL for the build.</p> <p>Azure Pipelines example: vstfs://build-release/Build/130 GitHub example: https://github.com/fabrikam/asp</p>
Release.Artifacts.{alias}.SourceBranch	<p>The full path and name of the branch from which the source was built.</p> <p>Azure Pipelines example: refs/heads/main</p>
Release.Artifacts.{alias}.SourceBranchName	<p>The name only of the branch from which the source was built.</p> <p>Azure Pipelines example: main</p>
Release.Artifacts.{alias}.SourceVersion	<p>The commit that was built.</p> <p>Azure Pipelines example: bc0044458ba1d9298cdc649cb5dcf013180706f7</p>
Release.Artifacts.{alias}.Repository.Provider	<p>The type of repository from which the source was built.</p> <p>Azure Pipelines example: Git</p>
Release.Artifacts.{alias}.RequestedForID	<p>The identifier of the account that triggered the build.</p> <p>Azure Pipelines example: 2f435d07-769f-4e46-849d-10d1ab9ba6ab</p>
Release.Artifacts.{alias}.RequestedFor	<p>The name of the account that requested the build.</p> <p>Azure Pipelines example: Mateo Escobedo</p>

VARIABLE NAME	DESCRIPTION
Release.Artifacts.{alias}.Type	<p>The type of artifact source, such as Build.</p> <p>Azure Pipelines example: <code>Build</code> Jenkins example: <code>Jenkins</code> TeamCity example: <code>TeamCity</code> TFVC example: <code>TFVC</code> Git example: <code>Git</code> GitHub example: <code>Github</code></p>
Release.Artifacts.{alias}.PullRequest.TargetBranch	<p>The full path and name of the branch that is the target of a pull request. This variable is initialized only if the release is triggered by a pull request flow.</p> <p>Azure Pipelines example: <code>refs/heads/main</code></p>
Release.Artifacts.{alias}.PullRequest.TargetBranchName	<p>The name only of the branch that is the target of a pull request. This variable is initialized only if the release is triggered by a pull request flow.</p> <p>Azure Pipelines example: <code>main</code></p>

See also [Artifact source alias](#)

Default variables - Primary Artifact

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

VARIABLE NAME	SAME AS
Build.Type	Release.Artifacts.{Primary artifact alias}.Type
Build.PullRequest.TargetBranch	Release.Artifacts.{Primary artifact alias}.PullRequest.TargetBranch
Build.PullRequest.TargetBranchName	Release.Artifacts.{Primary artifact alias}.PullRequest.TargetBranchName

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)`.

Powershell Script

Type	Inline Script
Arguments	<code>-a "\$(Release.Artifacts.ASPNET4.CI.DEFINITIONNAME)"</code>
Inline Script	<pre>param (\$a) Write-Host "\$a"</pre>

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	<code>Write-Host "\$env:RELEASE_ARTIFACTS_ASPNET4_CI_DEFINITIONNAME"</code>

Note that the original name of the artifact source alias, `ASPNET4.CI`, is replaced by `ASPNET4_CI`.

View the current values of all 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 interface. At the top, it says "SampleApp - 1 > Release-2 > QA > Succeeded". Below that is a navigation bar with Pipeline, Tasks, Variables, Logs, Tests, Deploy, Cancel, Refresh, and Download buttons. The "Logs" tab is selected. On the left, there's a sidebar titled "Deployment process" with a "Succeeded" status. Under "Run on agent", there's also a "Succeeded" status. The main area is titled "Run on agent" and shows a log entry for "Initialize job" which is also marked as "succeeded". This log entry is highlighted with a red box.

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 for the "Initialize job" step. It displays a series of environment variable mappings. The log entries are:

```

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_TOOLDIRECTORY] --> [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.

TIP

If you get an error related to an Azure RM service connection, see [How to: Troubleshoot Azure Resource Manager service connections](#).

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. In the Pipeline Variables page, open the Scope drop-down list and select "Release". By default, when you add a variable, it is set to Release scope.
- Share values across all of the tasks within one specific stage by using **stage variables**. Use a stage-level variable for values that vary from stage to stage (and are the same for all the tasks in a stage). You define and manage these variables in the **Variables** tab of a release pipeline. In the Pipeline Variables page, open the Scope drop-down list and select the required stage. When you add a variable, set the Scope to the appropriate environment.

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.

IMPORTANT

The values of the hidden (secret) variables are securely stored 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.

NOTE

Creating custom variables can overwrite standard variables. For example, the PowerShell **Path** environment variable. If you create a custom `Path` variable on a Windows agent, it will overwrite the `$env:Path` variable and PowerShell won't be able to run.

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 (e.g., job or stage) will collide and the result may be unpredictable. Ensure that you use different names for variables across all your variable groups.

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)
```

Use Azure Key Vault secrets in Azure Pipelines

6/14/2021 • 5 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019

NOTE

This article will guide you through working with Azure key vault in your pipeline. If you want to set secret variables or reference variable groups, see [Define variables](#) for more details.

Azure Key Vault allows users to securely store, manage, and access sensitive information. Secrets can be API keys, credentials, certificates, etc.

Azure Key Vault service supports two types of containers: vaults and managed HSM (hardware security module) pools. Vaults support storing software and HSM-backed keys, secrets, and certificates, while managed HSM pools only support HSM-backed keys.

In this tutorial, you will learn how to:

- Create an Azure Key Vault using Azure CLI
- Add a secret and configure access to Azure key vault
- Use secrets in your pipeline

Prerequisites

- An Azure DevOps organization. If you don't have one, you can [create one for free](#).

Create an Azure Key Vault

Azure key vaults can be created and managed through the Azure portal or Azure CLI. We will use Azure CLI in this tutorial to create our Azure Key vault.

Sign in to the [Azure Portal](#), and then select the [Cloud Shell](#) button in the upper-right corner.

1. If you have more than one Azure subscription associated with your account, use the command below to specify a default subscription. You can use `az account list` to generate a list of your subscriptions.

```
az account set --subscription <your_subscription_name_or_ID>
```

2. Run the following command to set your default Azure region. You can use `az account list-locations` to generate a list of available regions.

```
az config set defaults.location=<your_region>
```

For example, this command will select the westus2 region:

```
az config set defaults.location=westus2
```

3. Run the following command to create a new resource group. A resource group is a container that holds

related resources for an Azure solution.

```
az group create --name <your-resource-group>
```

- Run the following command to create a new key vault.

```
az keyvault create \  
  --name <your-key-vault> \  
  --resource-group <your-resource-group>
```

- Run the following command to create a new secret in your key vault. Secrets are stored as a key value pair. In the example below, *Password* is the key and *mysecretpassword* is the value.

```
az keyvault secret set \  
  --name "Password" \  
  --value "mysecretpassword" \  
  --vault-name <your-key-vault-name>
```

Create a project

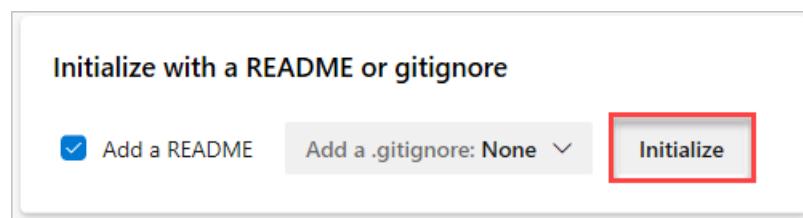
Sign in to [Azure Pipelines](#). Your browser will then navigate to <https://dev.azure.com/your-organization-name> and displays your Azure DevOps dashboard.

If you don't have any projects in your organization yet, select **Create a project to get started** to create a new project. Otherwise, select the **New project** button in the upper-right corner of the dashboard.

Create a repo

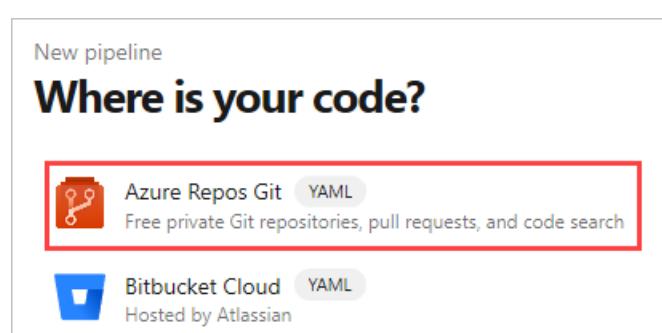
We will use YAML to create our pipeline but first we need to create a new repo.

- Sign in to your Azure DevOps organization and navigate to your project.
- Go to **Repos**, and then select **Initialize** to initialize a new repo with a README.



Create a new pipeline

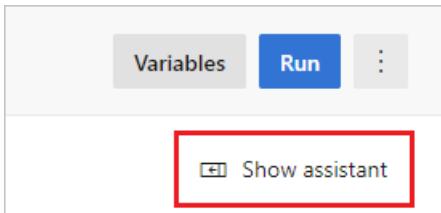
- Go to **Pipelines**, and then select **New Pipeline**.
- Select **Azure Repos Git**.



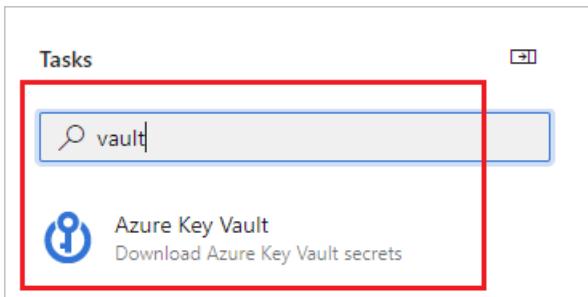
3. Select the repo you created earlier. It should have the same name as your Azure DevOps project.
4. Select **Starter pipeline**.
5. The default pipeline will include a few scripts that run echo commands. Those are not needed so we can delete them. Your new YAML file will now look like this:

```
trigger:  
- main  
  
pool:  
vmImage: 'ubuntu-latest'  
  
steps:
```

6. Select **Show assistant** to expand the assistant panel. This panel provides convenient and searchable list of pipeline tasks.



7. Search for **vault** and select the **Azure Key Vault** task.



8. Select and authorize your Azure subscription then select the Azure key vault task and select **Add** to add it to your pipeline. This task allows the pipeline to connect to your Azure Key Vault and retrieve secrets to use as pipeline variables.

NOTE

The **Make secrets available to whole job** feature is not currently supported in Azure DevOps Server 2019 and 2020.

The screenshot shows the 'Azure Key Vault' task configuration page. It includes fields for 'Azure subscription' (Fabrikam-subscription), 'Key vault' (PipelinesKeyVault7954), 'Secrets filter' (*), and a checkbox for 'Make secrets available to whole job'. There are also 'About this task' and 'Add' buttons.

9. Your YAML file should look something like the following

```
trigger:
- main

pool:
  vmImage: ubuntu-latest

steps:
- task: AzureKeyVault@2
  inputs:
    azureSubscription: 'Your-Azure-Subscription'
    KeyVaultName: 'Your-Key-Vault-Name'
    SecretsFilter: '*'
    RunAsPreJob: false

- task: CmdLine@2
  inputs:
    script: 'echo $(Your-Secret-Name) > secret.txt'

- task: CopyFiles@2
  inputs:
    Contents: secret.txt
    targetFolder: '$(Build.ArtifactStagingDirectory)'

- task: PublishBuildArtifacts@1
  inputs:
    PathtoPublish: '$(Build.ArtifactStagingDirectory)'
    ArtifactName: 'drop'
    publishLocation: 'Container'
```

10. Do not save or run your pipeline just yet. We must first give our pipeline the right permissions to access Azure Key Vault. Keep your browser tab open, we will resume the remaining steps once we set up the key vault permissions.

Set up Azure Key Vault access policies

In order to access our Azure Key Vault, we must first set up a service principal to give access to Azure Pipelines. Follow [this guide](#) to create your service principal and then proceed with the next steps in this section.

1. Go to [Azure portal](#).

2. Use the search bar to search for the key vault you created earlier.

3. Under **Settings** Select **Access policies**.

4. Select **Add Access Policy** to add a new policy.

5. For **Secret permissions**, select **Get** and **List**.

6. Select the option to select a service principal and search for the one you created in the beginning of this section. A security principal is an object that represents a user, group, service, or application that's requesting access to Azure resources.

7. Select **Add** to create the access policy, then **Save**.

Run and review the pipeline

1. Return to the previous tab where we left off.

2. Select **Save** then **Save** again to commit your changes and trigger the pipeline.

NOTE

You may be asked to allow the pipeline access to Azure resources, if prompted select **Allow**. You will only have to approve your pipeline once.

3. Select the **CmdLine** job to view the logs.

4. Return to pipeline summary and select the published artifact.

5. Under **Job** select the **secret.txt** file to open it.

The screenshot shows the 'Artifacts' section of the Azure Pipelines interface. Under the 'Published' tab, there is a table with two rows. The first row is collapsed, showing a 'Job' entry with a size of 17 B. The second row is expanded, showing a file named 'secret.txt' with a size of 17 B. A red box highlights the 'secret.txt' file.

Name	Size
Job	17 B
secret.txt	17 B

6. The text file should contain our secret: *mysecretpassword* from earlier.

WARNING

This tutorial is for educational purposes only. For security best practices and how to safely work with secrets, see [Manage secrets in your server apps with Azure Key Vault](#).

If you encounter an error indicating that the user or group does not have secrets list permission on key vault, run the following commands to authorize your application to access the key or secret in the Azure Key Vault:

```
$ErrorActionPreference="Stop";
Login-AzureRmAccount -SubscriptionId your-subscription-ID;
$spn=(Get-AzureRmADServicePrincipal -SPN service-principal-ID);
$spnObjectId=$spn.Id;
Set-AzureRmKeyVaultAccessPolicy -VaultName key-vault-tutorial -ObjectId $spn.ObjectId -PermissionsToSecrets
get,list;
```

Clean up resources

Follow the steps below to delete the resources you created:

1. If you created a new organization to host your project, see [how to delete your organization](#), otherwise [delete your project](#).
2. All Azure resources created during this tutorial are hosted under a single resource group *PipelinesKeyVaultResourceGroup*. Run the following command to delete the resource group and all of its resources.

```
az group delete --name PipelinesKeyVaultResourceGroup
```

Next steps

[Artifacts in Azure Pipelines](#) Publish and download artifacts in Azure Pipelines Release artifacts and artifact sources

Use Azure Key Vault secrets in your Pipeline

7/2/2021 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019

With Azure Key Vault, you can securely store and manage your sensitive information such as passwords, API keys, certificates, etc. using Azure Key Vault, you can easily create and manage encryption keys to encrypt your data. Azure Key Vault can also be used to manage certificates for all your resources.

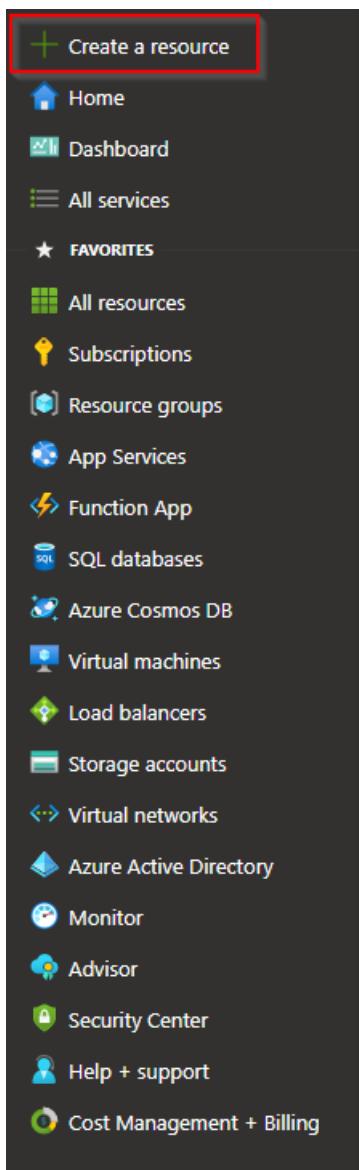
Prerequisites

- An Azure DevOps organization. [Create one for free](#) if you don't already have one.
- Your own project. [Create a project](#) if you don't already have one.
- Your own repository. [Create a new Git repo](#) if you don't already have one.
- An Azure subscription. [Create a free Azure account](#) if you don't already have one.

Create an Azure Key Vault

- [Azure portal](#)
- [Azure CLI](#)

1. Navigate to [Azure portal](#).
2. Select **Create a resource** in the left navigation pane.



3. Search for Key Vault and then click Enter.

A screenshot of the 'Create a resource' search results page. The search bar at the top is highlighted with a red box and contains the text 'Key Vault'. To the right of the search bar is a 'Getting Started? Try our Quickstart center' link with a yellow ribbon icon. Below the search bar, there are sections for 'Popular services' and 'Popular offers'. The 'Popular services' section includes links for Virtual machine, Kubernetes Service, Azure Cosmos DB, Function App, and SQL Database, each with a small icon and 'Create | Learn more' buttons. The 'Popular offers' section includes links for Ubuntu Server 18.04 LTS, Check Point Security Management, Citrix Virtual Apps Essentials, Windows Server 2016 Datacenter, and OpenVPN Access Server 2.8.5, each with a small icon and 'Create | Learn more' buttons.

4. Select **Create** to create a new Azure Key Vault.

Key Vault

Microsoft



Key Vault

Microsoft

★ ★ ★ ★ 4.1 (585 ratings)

Azure benefit eligible

Create

Overview Plans Usage Information + Support Reviews

5. Select your **Subscription** and then add a new **Resource group**. Enter a **Key vault name** and select a **Region** and a **Pricing tier**. Select **Review + create** when you are done.

Home > Create a resource > Key Vault >

Create key vault

...

Basics Access policy Networking Tags Review + create

Azure Key Vault is a cloud service used to manage keys, secrets, and certificates. Key Vault eliminates the need for developers to store security information in their code. It allows you to centralize the storage of your application secrets which greatly reduces the chances that secrets may be leaked. Key Vault also allows you to securely store secrets and keys backed by Hardware Security Modules or HSMs. The HSMs used are Federal Information Processing Standards (FIPS) 140-2 Level 2 validated. In addition, key vault provides logs of all access and usage attempts of your secrets so you have a complete audit trail for compliance.

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *	<input type="text" value="rabououn-devops"/>
Resource group *	<input type="text" value="Create new"/>

Instance details

Key vault name * ⓘ	<input type="text" value="Enter the name"/>
Region *	<input type="text" value="East US"/>
Pricing tier * ⓘ	<input type="text" value="Standard"/>

Recovery options

Soft delete protection will automatically be enabled on this key vault. This feature allows you to recover or permanently delete a key vault and secrets for the duration of the retention period. This protection applies to the key vault and the secrets stored within the key vault.

To enforce a mandatory retention period and prevent the permanent deletion of key vaults or secrets prior to the retention period elapsing, you can turn on purge protection. When purge protection is enabled, secrets cannot be purged by users or by Microsoft.

Review + create

< Previous

Next : Access policy >

6. Select **Go to resource** when the deployment of your new resource is completed.

The screenshot shows the Azure portal interface for a deployment named 'kv-demo-repo'. The 'Deployment' blade is active, indicating the deployment is complete. It displays the deployment name, subscription, and resource group. Below the deployment details, there are sections for 'Deployment details' (with a download link) and 'Next steps'. A prominent blue button labeled 'Go to resource' is located at the bottom of the blade, which is highlighted with a red rectangular border.

Configure Key Vault access permissions

Before proceeding with the next steps, we must first create a service principal to be able to query our Azure Key Vault from Azure Pipelines. Follow the steps in the following how-to to [Create a service principal](#) and then continue with the next steps.

1. Navigate to [Azure portal](#).
2. Select the key vault you created in the previous step.
3. Select **Access policies**.

The screenshot shows the 'Access policies' section within the 'Settings' of a Key Vault. The 'Access policies' option is highlighted with a red rectangular border. Other settings listed include Keys, Secrets, Certificates, Networking, Security, Properties, and Locks.

4. Select **Add Access Policy** to add a new policy.

5. Add a **Get** and **List** to **Secret permissions**.

Add access policy ...

Add access policy

Configure from template (optional)

Key permissions 0 selected

Secret permissions 2 selected

Certificate permissions

Select principal *

Authorized application (i)

Add

Select all

Get

List

Set

Delete

Recover

Backup

Restore

Secret Management Operations

Purge

Privileged Secret Operations

6. Under **Select principal**, select to add a service principal and choose the one you created earlier.

7. Select **Save** when you are done.

Query and use secrets in your pipeline

Using the [Azure Key Vault task](#) we can fetch the value of our secret and use it in subsequent tasks in our pipeline. One thing to keep in mind is that secrets must be explicitly mapped to env variable as shown in the example below.

```

pool:
  vmImage: 'ubuntu-latest'

steps:
- task: AzureKeyVault@1
  inputs:
    azureSubscription: 'repo-kv-demo'                                ## YOUR_SERVICE_CONNECTION_NAME
    KeyVaultName: 'kv-demo-repo'                                     ## YOUR_KEY_VAULT_NAME
    SecretsFilter: 'secretDemo'                                       ## YOUR_SECRET_NAME
    RunAsPreJob: false

- task: DotNetCoreCLI@2
  inputs:
    command: 'build'
    projects: '**/*.csproj'

- task: DotNetCoreCLI@2
  inputs:
    command: 'run'
    projects: '**/*.csproj'
  env:
    mySecret: $(secretDemo)

- bash: |
  echo "Secret Found! $MY_MAPPED_ENV_VAR"
  env:
    MY_MAPPED_ENV_VAR: $(mySecret)

```

The output from the last bash command should look like this:

```
Secret Found! ***
```

Related articles

- [Manage service connections](#)
- [Define variables](#)
- [Publish Pipeline Artifacts](#)

Release gates and approvals overview

5/26/2021 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

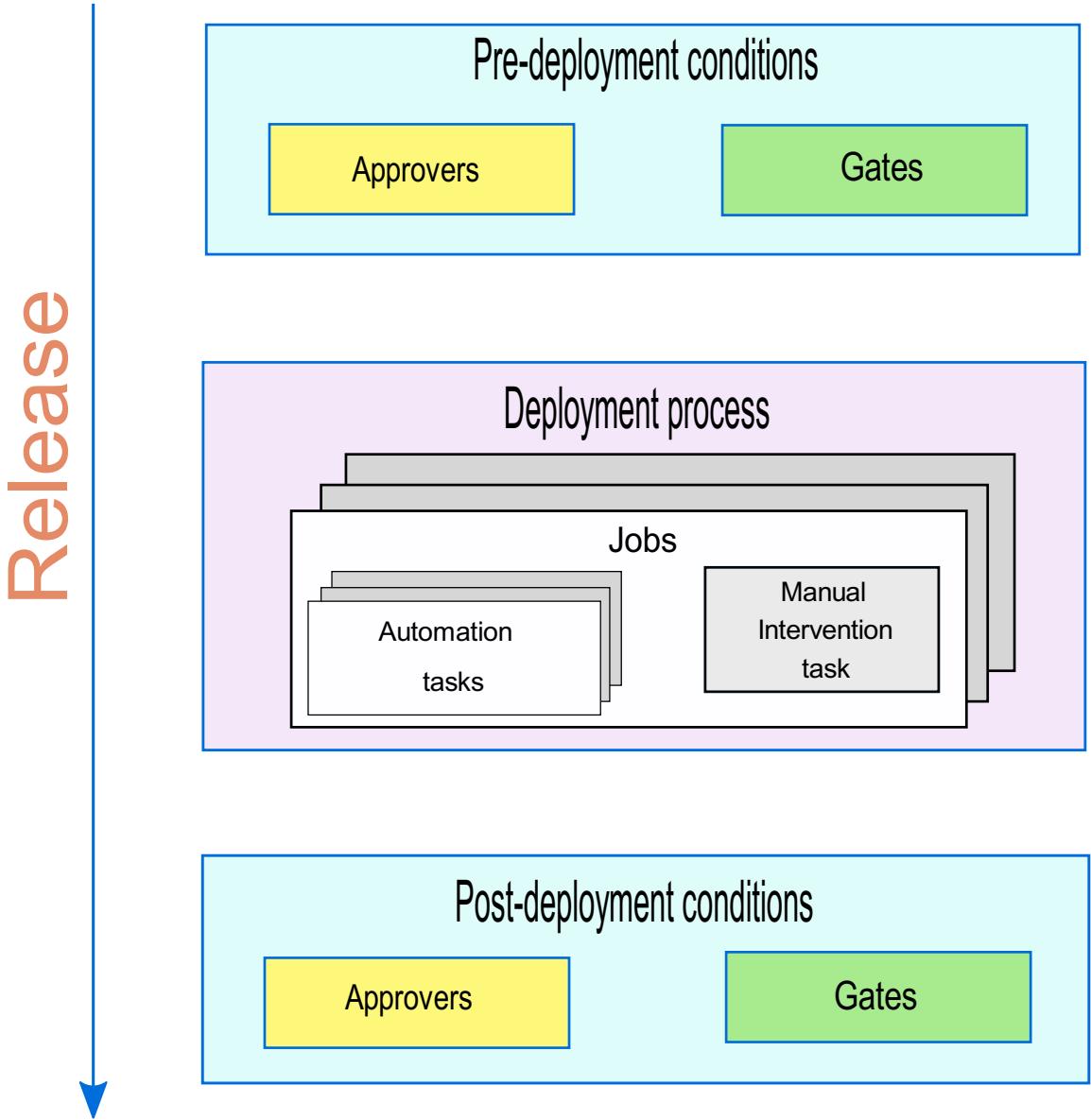
NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Release pipelines enable teams to continuously deploy their application across different stages with lower risk and with faster pace. Deployments to each stage can be fully automated by using jobs and tasks.

Teams can also take advantage of the Approvals and Gates feature to control the workflow 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 that specific conditions are met. In addition, teams can configure manual validations to pause the deployment pipeline and prompt users to carry out manual tasks then resume or reject the deployment.

The following diagram illustrates the release pipeline workflow.



By using gates, approvals, 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
A user must manually validate the change request and approve the deployment to a certain stage.	Pre-deployment approvals
A user must manually sign off after deployment before the release is triggered to other stages.	Post-deployment approvals
A team wants 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
A team wants to ensure there are no reported incidents after deployment, before triggering a release.	Post-deployment gates

SCENARIO	FEATURE(S) TO USE
After deployment, a team wants to wait for a specified time before prompting users to sign out.	Post-deployment gates and post-deployment approvals
During deployment, a user must manually follow specific instructions and then resume the deployment.	Manual Intervention or Manual Validation
During deployment, a team wants to prompt users to enter a value for a parameter used by the deployment tasks, or allow users to edit the release.	Manual Intervention or Manual Validation
During deployment, a team wants to wait for monitoring or information portals to detect any active incidents, before continuing with other deployment jobs.	Planned

You can combine all three techniques within a release pipeline to fully achieve your own deployment requirements.

In addition, you can install an extension that integrates with *ServiceNow* to help you control and manage your deployments through Service Management methodologies such as ITIL. For more information, see [Integrate with ServiceNow change management](#).

NOTE

The time delay before pre-deployment gates are executed is capped at 48 hours. If you need to delay the overall launch of your gates instead, it is recommended to use a [delay task](#) in your release pipeline.

```
# Delay further execution of a workflow by a fixed time
jobs:
- job: RunsOnServer
  pool: Server
  steps:
  - task: Delay@1
    inputs:
      delayForMinutes: '0'
```

Related articles

- [Release deployment control using approvals](#)
- [Release deployment control using gates](#)
- [Configure a manual intervention](#)
- [Add stages, dependencies, & conditions](#)
- [Release triggers](#)
- [Releases in Azure Pipelines](#)

Next steps

[Define approvals and checks](#) [Use approvals and gates to control your deployment](#)

Define approvals and checks

3/6/2021 • 10 minutes to read • [Edit Online](#)

Azure Pipelines

A pipeline is made up of stages. A pipeline author can control whether a stage should run by defining [conditions](#) on the stage. Another way to control if and when a stage should run is through [approvals and checks](#).

Pipelines rely on resources such as environments, service connections, agent pools, variable groups, and secure files. Checks enable the *resource owner* to control if and when a stage in any pipeline can consume a resource. As an owner of a resource, you can define checks that must be satisfied before a stage consuming that resource can start. For example, a *manual approval check* on an [environment](#) would ensure that deployment to that environment only happens after the designated user(s) has reviewed the changes being deployed.

A stage can consist of many jobs, and each job can consume several resources. Before the execution of a stage can begin, all checks on all the resources used in that stage must be satisfied. Azure Pipelines pauses the execution of a pipeline prior to each stage, and waits for all pending checks to be completed. Checks are re-evaluation based on the retry interval specified in each check. If all checks are not successful till the [timeout](#) specified, then that stage is not executed. If any of the checks terminally fails (for example, if you reject an approval on one of the resources), then that stage is not executed.

Approvals and other checks are not defined in the yaml file. Users modifying the pipeline yaml file cannot modify the checks performed before start of a stage. Administrators of resources manage checks using the web interface of Azure Pipelines.

IMPORTANT

Checks can be configured on environments, service connections and agent pools. Service connections cannot be specified by variable.

Approvals

You can manually control when a stage should run using approval checks. This is commonly used to control deployments to production environments.

1. In your Azure DevOps project, go to the resource (eg environment) that needs to be protected.
2. Navigate to [Approvals and Checks](#) for the resource.

A screenshot of the Azure DevOps interface. The top navigation bar shows 'Pipelines / Environments / production voting-app'. Below this, the 'production voting-app' environment page is displayed. On the left, there's a sidebar with various icons. The main area shows a table with one row, where the first column is 'Name' and the second is 'Latest job'. A tooltip for the job indicates it's '#AzureDevops_IgniteDemo2019_bbq-beets.voting-app-kubernetes_20191030.2 on bbq-beets.voter'. On the right, there's a context menu with options: 'Add resource' (blue button), 'Edit', 'Security', 'Approvals and checks' (which is highlighted with a red box), and 'Delete'.

The screenshot shows the 'Approvals and checks' option in the context menu of an environment resource, which is highlighted with a red box.

3. Select **Create**, provide the approvers and an optional message, and select **Create** again to complete the addition of the manual approval check.

You can add multiple approvers to an environment. 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 run to move forward.

Using the advanced options, you can configure minimum number of approvers to complete the approval. A group is considered as one approver.

You can also restrict the user who requested (initiated or created) the run from completing the approval. This option is commonly used for segregation of roles amongst the users.

When you run a pipeline, the execution of that run pauses before entering a stage that uses the environment. Users configured as approvers must review and approve or reject the deployment. If you have multiple runs executing simultaneously, you must approve or reject each of them independently. If all required approvals are not complete within the **Timeout** specified for the approval and all other checks succeed, the stage is marked skipped.

Branch control

Using the branch control check, you can ensure all the resources linked with the pipeline are built from the **allowed** branches and that they branches have protection enabled. This helps in control the release readiness and quality of deployments. In case multiple resources are linked with the pipeline, source for all the resources is verified. If you have linked another pipeline, then the branch of the specific run being deployed is verified for protection.

To define the branch control check:

1. In your Azure DevOps project, go to the resource (eg environment) that needs to be protected.
2. Navigate to **Approvals and Checks** for the resource.
3. Choose the **Branch control** check and provide a comma separated list of allowed branches. You can mandate that the branch should have protection enabled and the behavior of the check in case protection status for one of the branches is not known.

Branch control

Display name *

Allowed branches * ⓘ

Verify branch protection * ⓘ

Ignore unknown protection status * ⓘ

Control options

At run time, the check would validate branches for all linked resources in the run against the allowed list. If any one of the branches do not match the criteria, the check fails and the stage is marked failed.

NOTE

The check requires the branch names to be fully qualified. Make sure the format for branch name is

```
refs/heads/<branch name>
```

Business hours

In case you want all deployments to your environment to happen in a specific time window only, then business hours check is the ideal solution. When you run a pipeline, the execution of the stage that uses the resource waits for business hours. If you have multiple runs executing simultaneously, each of them is independently verified. At the start of the business hours, the check is marked successful for all the runs.

Business Hours

X

Display name *

Business Hours

Days * ⓘ

Monday (+4)



Time Zone * ⓘ

(UTC) Coordinated Universal Time



Start Time * ⓘ

04:00

End Time * ⓘ

11:00

Control options



If execution of the stage has not started at the end of business hours (held up by to some other check), then the business hours approval is automatically withdrawn and a re-evaluation is scheduled for the next day. The check fails if execution of the stage does not start within the **Timeout** period specified for the check, and the stage is marked failed.

Invoke Azure function

Azure functions are the serverless computation platform offered by Azure. With Azure functions, you can run small pieces of code (called "functions") without worrying about application infrastructure. Given the high flexibility, Azure functions provide a great way to author your own checks. You include the logic of the check in Azure function such that each execution is triggered on http request, has a short execution time and returns a response. While defining the check, you can parse the response body to infer if the check is successful. The evaluation can be repeated periodically using the Time between evaluations setting in control options. [Learn More](#)

Invoke Azure Function

X

Query parameters ⓘ

Body ⓘ

Advanced

Completion event * ⓘ

ApiResponse



Success criteria ⓘ

Control options

Time between evaluations (minutes)

5

Timeout (minutes)

43200

Linked Variable Group



The checks fails if the stage has not started execution within the specified **Timeout** period.

NOTE

User defined pipeline variables are not accessible to the check. You can only access the pre-defined variables and variables from the linked variable group in the request body.

Invoke REST API

Invoke REST API check enables you to integrate with any of your existing services. Periodically, make a call to a REST API and continue if it returns a successful response. [Learn More](#)

The evaluation can be repeated periodically using the **Time between evaluations** setting in control options. The checks fails if the stage has not started execution within the specified **Timeout** period.

NOTE

User defined pipeline variables are not accessible to the check. You can only access the pre-defined variables and variables from the linked variable group in the request body.

Query Azure Monitor Alerts

Azure Monitor offers visualization, query, routing, alerting, autoscale, and automation on data from the Azure infrastructure and each individual Azure resources. Alerts are a standard means to detect issues with the health of infrastructure or application, and take corrective actions. Canary deployments and staged rollouts are common deployment strategies used to lower risk of regressions to critical applications. After deploying to a stage (set of customers), the application is observed for a period of time. Health of the application after deployment is used to decide whether the update should be made to the next stage or not.

Query Azure Monitor Alerts helps you observe Azure Monitor and ensure no alerts are raised for the application after a deployment. The check succeeds if no alert rules are activated at the time of evaluation. [Learn More](#)

The evaluation is repeated after **Time between evaluations** setting in control options. The checks fails if the stage has not started execution within the specified **Timeout** period.

Required template

With the required template check, you can enforce pipelines to use a specific YAML template. When this check is in place, a pipeline will fail if it doesn't extend from the referenced template.

To define a required template approval:

1. In your Azure DevOps project, go to the [service connection](#) that you want to restrict.
2. Open **Approvals and Checks** in the menu next to **Edit**.
3. In the **Add your first check** menu, select **Required template**.
4. Enter details on how to get to your required template file.
 - **Repository type:** The location of your repository (GitHub, Azure, or Bitbucket).
 - **Repository:** The name of your repository that contains your template.
 - **Ref:** The branch or tag of the required template.
 - **Path to required template:** The name of your template.

You can have multiple required templates for the same service connection. In this example, the required template is `required.yml`.

Required template

Repository type

Azure Repos Git

Repository

IncludeTemplates

Ref

refs/heads/master

Path to required template

required.yml

Add

Evaluate artifact

You can evaluate artifact(s) to be deployed to an environment against custom policies.

NOTE

Currently, this works with container image artifacts only

To define a custom policy evaluation over the artifact(s), follow the below steps.

1. In your Azure DevOps Services project, navigate to the environment that needs to be protected. Learn more about [creating an environment](#).

The screenshot shows the 'Environments' page in the Azure DevOps Pipelines section. On the left, there's a sidebar with icons for Pipelines, Environments, Artifacts, and more. The main area is titled 'Environments'. It lists several environments, each with a name, status (indicated by a green circle with a checkmark), and last activity time. The environment named 'production voting-app' is highlighted with a red rectangular box. The status for this environment is also green with a checkmark. Other environments listed include 'bbq-beets', 'voting-app-kubernetes', and several others whose names are partially visible or blurred.

2. Navigate to **Approvals and checks** for the environment.

The screenshot shows the Azure DevOps interface for managing environments. In the top navigation bar, you can see the path: Pipelines / Environments / production voting-app. Below this, there's a search bar and some global settings. On the left, there's a sidebar with various icons for different services. The main content area shows a table with one row, and in the top right, there's a context menu with options like 'Edit', 'Security', 'Approvals and checks' (which is highlighted with a red box), and 'Delete'.

3. Select Evaluate artifact.

This screenshot shows the 'Approvals and checks' page for the 'production voting-app'. It features a central illustration of a person sitting on the ground with a dog, surrounded by clouds. To the right, there's a section titled 'Add your first check' with a sub-section for 'Approvals'. At the bottom of this section, the 'Evaluate artifact (preview)' option is highlighted with a red box. The overall layout includes a sidebar on the left and a top navigation bar.

4. Paste the policy definition and click Save. [See more](#) about writing policy definitions.

This screenshot shows the 'Evaluate artifact policies' dialog box. It has a 'Display name' field set to 'allow whitelisted registries'. Under 'Policy definitions', there's a code editor containing a JSON snippet. The code defines a whitelist of registry URLs and handles violations and check registries. At the bottom right, there are 'Cancel' and 'Save' buttons.

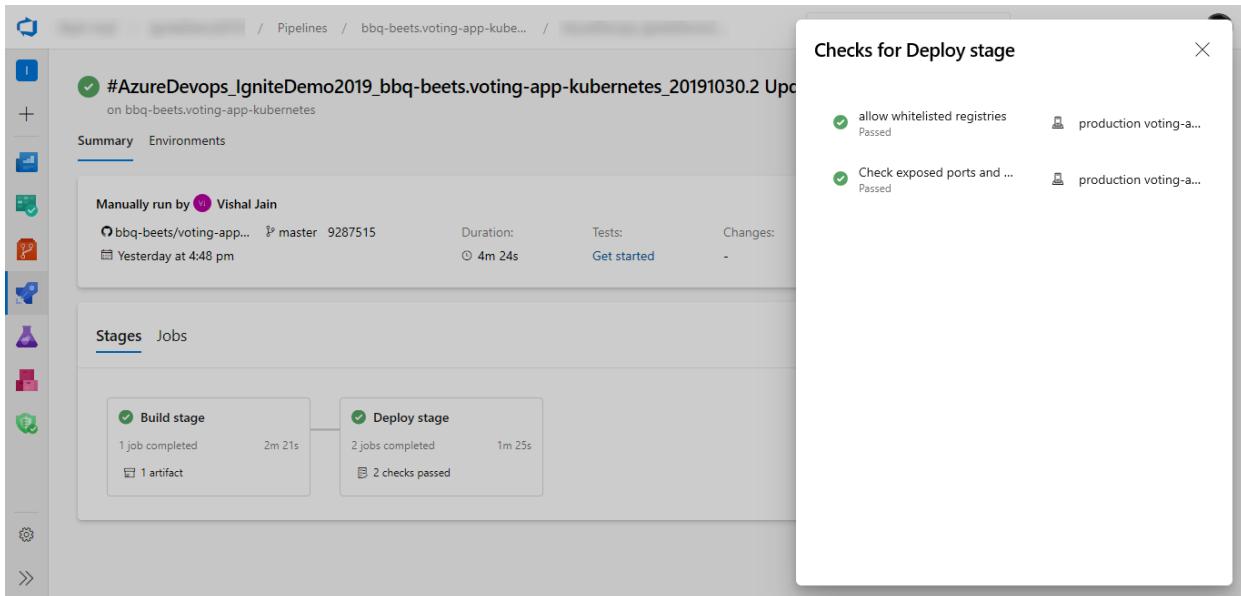
```

{
  "display_name": "allow whitelisted registries",
  "policy_definitions": [
    {
      "name": "package test",
      "definition": {
        "import input as values": true,
        "whitelist": [
          "https://.azuredcr.io",
          "https://.azuredcr.io"
        ],
        "violations[msg]": {
          "whitelistViolations := checkregistries",
          "msg := whitelistViolations"
        },
        "checkregistries[errors]": {
          "trace.Sprintf(\"Whitelisted registries: %s\", [concat(\" \", whitelist)])",
          "resourceUri := values[index].image.resourceUri"
        }
      }
    }
  ]
}
  
```

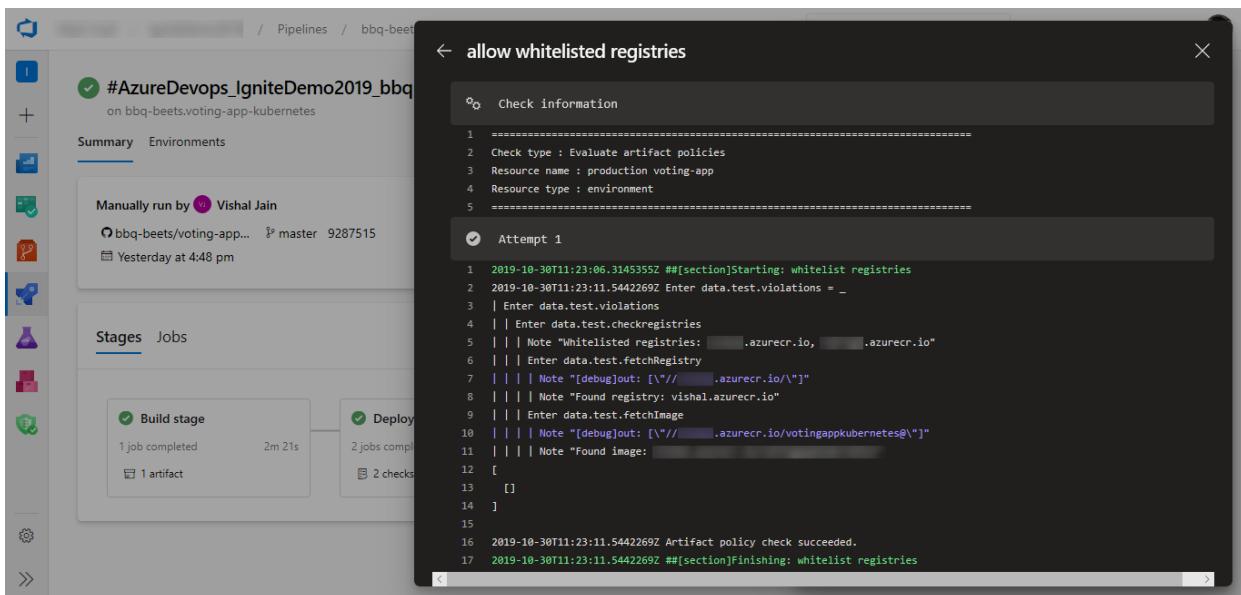
When you run a pipeline, the execution of that run pauses before entering a stage that uses the environment.

The specified policy is evaluated against the available image metadata. The check passes when the policy is successful and fails otherwise. The stage is marked failed if the check fails.

- Passed
- Failed



You can also see the complete logs of the policy checks from the pipeline view.



Exclusive lock

The **exclusive lock** check allows only a single run from the pipeline to proceed. All stages in all runs of that pipeline which use the resource are paused. When the stage using the lock completes, then another stage can proceed to use the resource. Also, only one stage will be allowed to continue. Any other stages which tried to take the lock will be cancelled.

ServiceNow Change Management

This checks needs the [ServiceNow Change Management extension](#) to be installed from the marketplace

The **servicenow change management** check allows for an integration of ServiceNow change management process in the pipelines. By adding the check, a new change request in ServiceNow can be automatically created

at the start of the stage. The pipeline waits for the change process to complete before starting the stage. More details are available [here](#).

FAQ

The checks defined did not start. What happened?

The evaluation of checks starts once the stage conditions are satisfied. You should confirm run of the stage started after the checks were added on the resource and that the resource is consumed in the stage.

How can I use checks for scheduling a stage?

Using the business hours check, you can control the time for start of stage execution. You can achieve the same behavior as [predefined schedule on a stage](#) in designer releases.

How can I take advance approvals for a stage scheduled to run in future?

This scenario can be enabled

1. The business hours check enables all stages deploying to a resource to be scheduled for execution between the time window
2. When approvals configured on the same resource, then the stage would wait for approvals before starting.
3. You can configure both the checks on a resource. The stage would wait on approvals and business hours. It would start in the next scheduled window after approvals are complete.

Can I wait for completion of security scanning on the artifact being deployed?

In order to wait for completion of security scanning on the artifact being deployed, you would need to use an external scanning service like AquaScan. The artifact being deployed would need to be uploaded at a location accessible to the scanning service before the start of checks, and can be identified using [pre-defined variables](#). Using the Invoke REST API check, you can add a check to wait on the API in the security service and pass the artifact identifier as an input.

How can I use output variables from previous stages in a check?

By default, only pre-defined variables are available to checks. You can use a linked variable group to access other variables. The output variable from the previous stage can be written to the variable group and accessed in the check.

Release deployment control using gates

5/26/2021 • 5 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](#).
- **Security and compliance assessment**: Assess Azure Policy compliance on resources within the scope of a given subscription and resource group, and optionally at a specific resource level. For more details, see [Security Compliance and Assessment task](#).

You can [create your own gates](#) with Marketplace extensions.

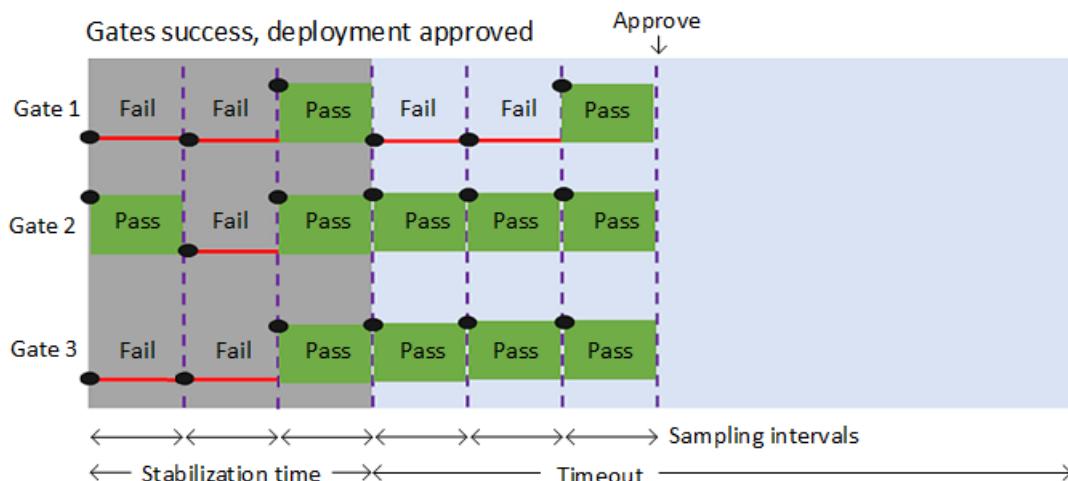
The evaluation options that apply to all the gates you've added are:

- **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 to approve.

For information about viewing gate results and logs, see [View the logs for approvals](#) and [Monitor and track deployments](#).

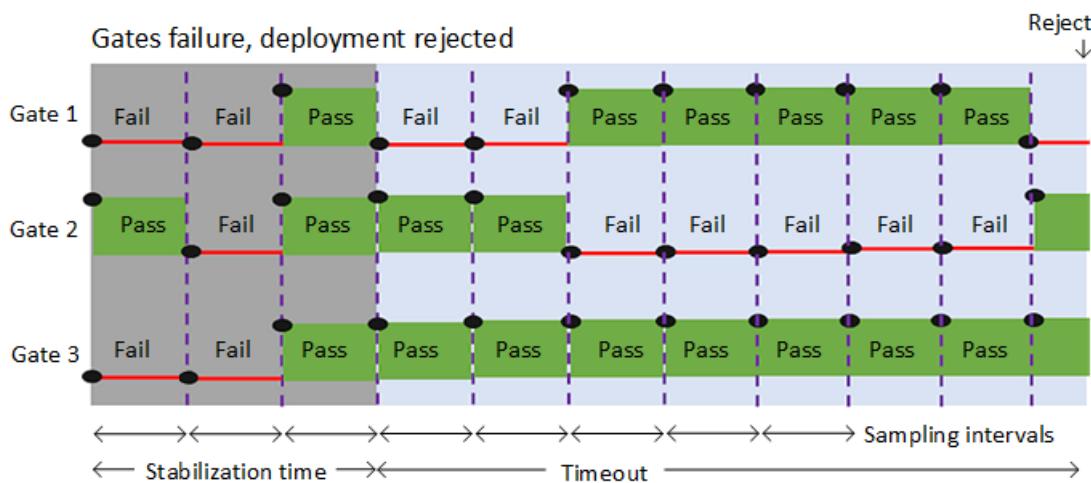
Gate evaluation flow examples

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.



Use gates and approvals to control your deployment

5/26/2021 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

By using a combination of manual deployment approvals, gates, and manual intervention in your release pipeline, you can quickly and easily configure your deployment to meet all the specific pre-deployment requirements for your DevOps CI/CD workflow.

In this tutorial, you will learn about:

- Pre-deployment gates
- Manual intervention
- Manual validation
- Deployment logs

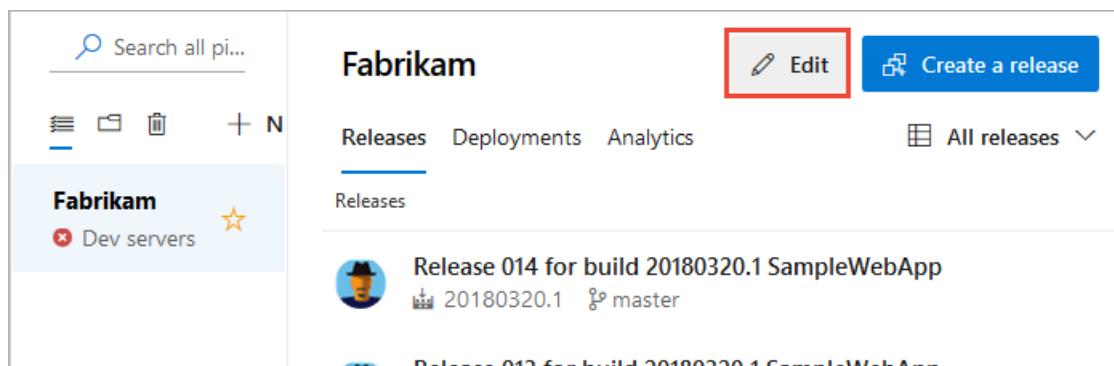
Prerequisites

- Complete the [Define your multi-stage continuous deployment \(CD\) pipeline](#) tutorial.
- A work item query. You can create a new query here: [Create managed queries with the query editor](#) if you don't already have one.

Set up gates

You can use gates to ensure that the release pipeline meets specific criteria before deployment without requiring user intervention.

1. Select **Releases**, and then select your release pipeline and then **Edit** to open the pipeline editor.



2. Select the pre-deployment icon for your stage to open the pre-deployment conditions panel, and then select the toggle button to **Enable gates**.

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. Enter the delay time before the added gates are evaluated. This time is to allow gate functions to initialize and stabilize before returning results.

Gates ▾ Define gates to evaluate before the deployment. [Learn more](#) Enabled

The delay before evaluation ⓘ Minutes ▾

Deployment gates ⓘ [+ Add](#)

4. Select **Add**, and then choose the **Query Work Items** task.

Gates ▾ Define gates to evaluate before the deployment. [Learn more](#) Enabled

The delay before evaluation ⓘ Minutes ▾

Deployment gates ⓘ [+ Add](#)

Deployment queue settings ⓘ Define behavior when multiple releases are triggered.

- Invoke Azure Function
- Invoke REST API
- Query Classic Azure Monitor Alerts
- Query Work Items** Executes a work item query and checks for the number of items returned.
- Security and compliance assessment

5. Select an existing work item query or [create your own](#). Depending on how many work items you expect the query to return, set your maximum and minimum thresholds.

Deployment gates ⓘ

+ Add ▾

Query Work Items Enabled

Query Work Items ⓘ

Task version 0.* ▾

Display name *

Query Work Items

Query * ⓘ

Active Bugs ▾

Upper threshold * ⓘ

0

Advanced ^

Lower threshold * ⓘ

0

Output Variables ^

Reference name ⓘ

Variables list

There are no output variables associated with this task [more information](#) ▾

Evaluation options ▾

6. Select the **Evaluation options** section, and then specify the timeout and the sampling interval. The minimum values you can specify are 6-minutes timeout and 5-minutes sampling interval.

Evaluation options ▾

The time between re-evaluation of gates ⓘ

Minutes ▾

Minimum duration for steady results after a successful gates evaluation ⓘ

Minutes ▾

The timeout after which gates fail ⓘ

Hours ▾

Gates and approvals ⓘ

Before gates, ask for approvals

On successful gates, ask for approvals

Ignore gates outcome and ask for approvals

7. Select **Save** when you are done.

All definitions > SampleApp - 1

Save + Release

Set up manual intervention

Depending on the scenario, sometimes you may need to add manual intervention to your release pipeline. You can do this by adding the **Manual Intervention** task to your pipeline.

1. Select your release pipeline, and then select **Tasks** and choose your stage.

All pipelines > SampleApp - 1

Pipeline Tasks Variables Retention Options History

QA Deployment

Production

Run on agent

Azure App Service

Version 3.*

2. Select the ellipses (...) from your stage definition, and then select **Add agentless job**.

All pipelines > SampleApp - 1

Pipeline Tasks Variables Retention Options History

QA Deployment process

Run on agent

Deploy Azure App Service

Stage name

Add an agent job

Add a deployment group job

Add an agentless job

NOTE

The **Manual Intervention** task can only be used in an [agentless job](#).

3. Drag and drop the agentless job to the start of the deployment process. Select the + sign, and then select **Add the Manual Intervention task**.

The screenshot shows the 'Tasks' tab selected in the pipeline editor. A modal window titled 'Add tasks' is open, displaying a list of available tasks. The 'Manual Intervention' task is highlighted with a red box around its icon and name. Other visible tasks include 'Delay', 'Invoke Azure Function', 'Invoke REST API', and 'Agentless job'. A blue 'Add' button is at the bottom right of the modal.

- Configure the task by entering the Instructions to be displayed when it the task is triggered.

The screenshot shows the 'Tasks' tab selected in the pipeline editor. The 'Manual Intervention' task is now configured with the following settings:

- Display name:** Manual Intervention
- Instructions:** Ensure QA database updates have completed before continuing deployment
- Notify users:** Mateo Escobedo
- On timeout:** Reject

You can specify a list of users to be notified when the deployment is pending manual approval. You can also specify a timeout and the action (approve or reject) that occur if no intervention within the timeout

period. See the [Manual Intervention](#) task for more details.

5. Select **Save** when you are done.

Set up manual validation

You can use the [manual validation](#) task in your YAML to pause the pipeline run and wait for manual approval. Manual validation is especially useful in scenarios where you want to pause a pipeline and validate configuration settings or build packages before starting a computation-intensive job.

The `waitForValidation` job pauses the run and triggers a prompt within the Pipeline UI to review and validate the task. The email addresses listed in `notifyUsers` receive a notification to approve or deny the pipeline run.

1 manual validation needs your review before this run can continue

Review

```
pool:
  vmImage: ubuntu-latest

jobs:
- job: waitForValidation
  displayName: Wait for external validation
  pool: server
  timeoutInMinutes: 4320 # job times out in 3 days
  steps:
    - task: ManualValidation@0
      timeoutInMinutes: 1440 # task times out in 1 day
      inputs:
        notifyUsers: |
          someone@example.com
        instructions: 'Please validate the build configuration and resume'
        onTimeout: 'resume'
```

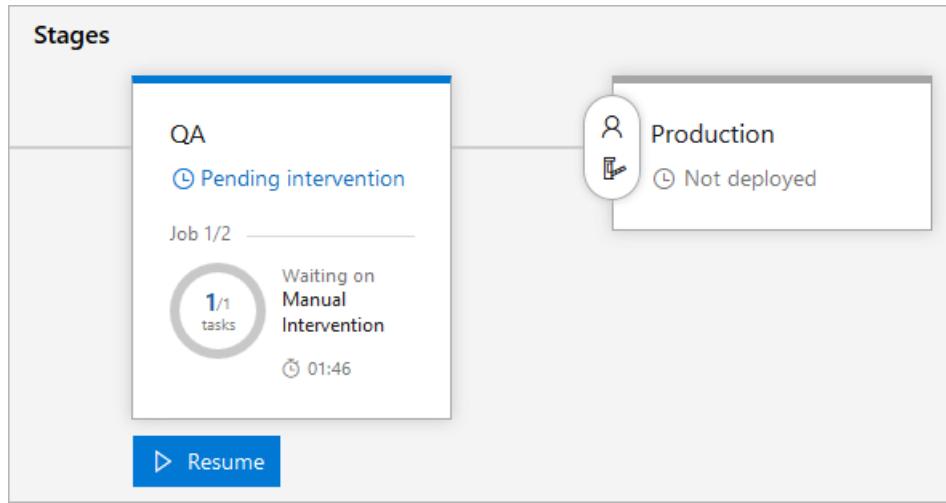
View deployment logs

Deployment logs can be very useful when debugging deployment issues. You can also use the logs to audit the run and verify approvals and how they were granted and by whom.

1. Select **Releases**, and then select your release pipeline.

The screenshot shows the Azure DevOps Releases page for the 'Fabrikam' pipeline. At the top, there's a search bar labeled 'Search all pipelines'. Below it, a sidebar shows the pipeline structure with 'Fabrikam' selected and 'Dev servers' listed. The main area displays two releases: 'Release 014 for build 20180320.1 SampleWebApp' and 'Release 013 for build 20180320.1 SampleWebApp'. The fourth stage of the fourth release is highlighted with a red box and labeled 'Release 014 for build 20180320.1 SampleWebApp'.

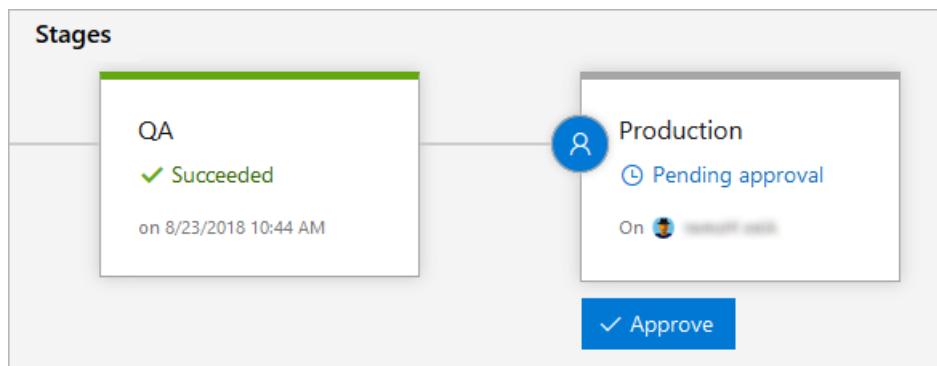
2. This view will show you a live status of each stage in your pipeline. The QA stage in this example is pending intervention. Select **Resume**.



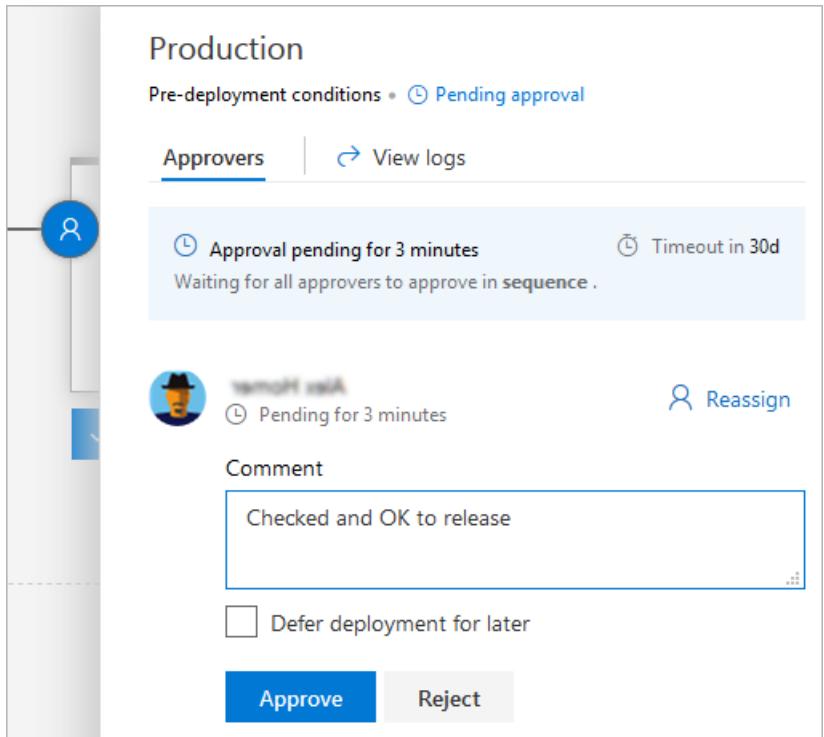
3. Enter your comment, and then select **Resume**

This screenshot shows the 'QA' stage details. At the top, there are links for 'Manual Intervention' and 'View logs'. A message box states 'Manual Intervention pending.' Below it, under 'Instructions', is the note 'Ensure QA database updates have completed before continuing deployment'. In the 'Comment' section, the text 'Databases checked, OK to deploy' is entered. At the bottom are two buttons: a blue 'Resume' button and a grey 'Reject' button.

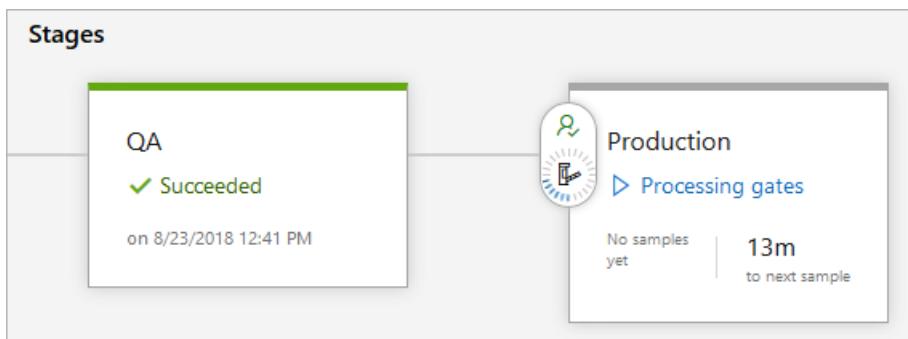
4. Return to your release pipeline. The **QA** stage deployment succeeded and the pre-deployment approvals are triggered for the **Production** stage.



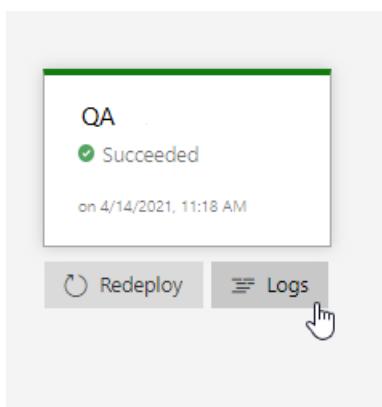
5. Select **Approve** and enter your comment and then select **Approve** to continue deployment.



6. Return to your release pipeline. The live status indicates that the gates are being processed for the **Production** stage before the release continues.



7. Return to your release pipeline, and hover over the stage you want to audit and then select **Logs** to view the logs.



Next step

[Integrate with ServiceNow](#) [Integrate with Jenkins \(Classic\)](#)

Release deployment control using approvals

5/26/2021 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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. These users must have the [View releases](#) permission.

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. For information about the ID of the requester for CI/CD releases, see [How are the identity variables set?](#)
- 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](#).

Description	Trigger	Delivery settings
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 articles

- [Approvals and gates overview](#)
- [Manual intervention](#)
- [Stages](#)
- [Triggers](#)

Help and support

- See our [troubleshooting](#) page
- Get advice on [Stack Overflow](#), and feel free to post your questions, search for answers, or suggest a feature on our [Azure DevOps Developer Community](#). [Support](#) page.

Pipeline run sequence

3/20/2021 • 12 minutes to read • [Edit Online](#)

Runs represent one execution of a pipeline. During a run, the pipeline is processed, and agents process one or more jobs. A pipeline run includes [jobs, steps, and tasks](#). Runs power both continuous integration (CI) and continuous delivery (CD) pipelines.

When you run a pipeline, many things happen under the covers. While you often won't need to know about them, occasionally it's useful to have the big picture. At a high level, Azure Pipelines will:

- [Process the pipeline](#)
- [Request one or more agents to run jobs](#)
- Hand off jobs to agents and collect the results

On the agent side, for each job, an agent will:

- [Get ready for the job](#)
- [Run each step in the job](#)
- [Report results to Azure Pipelines](#)

Jobs may [succeed, fail, or be canceled](#). There are also situations where a job [may not complete](#). Understanding how this happens can help you troubleshoot issues.

Let's break down each action one by one.

Process the pipeline

To turn a pipeline into a run, Azure Pipelines goes through several steps in this order:

1. First, expand [templates](#) and evaluate [template expressions](#).
2. Next, evaluate dependencies at the [stage](#) level to pick the first stage(s) to run.
3. For each stage selected to run, two things happen:
 - All resources used in all jobs are gathered up and validated for [authorization](#) to run.
 - Evaluate [dependencies at the job level](#) to pick the first job(s) to run.
4. For each job selected to run, expand [multi-configs](#) (`strategy: matrix` or `strategy: parallel` in YAML) into multiple runtime jobs.
5. For each runtime job, evaluate [conditions](#) to decide whether that job is eligible to run.
6. [Request an agent](#) for each eligible runtime job.

As runtime jobs complete, Azure Pipelines will see if there are new jobs eligible to run. If so, steps 4 - 6 repeat with the new jobs. Similarly, as stages complete, steps 2 - 6 will be repeated for any new stages.

This ordering helps answer a common question: why can't I use certain variables in my template parameters? Step 1, template expansion, operates solely on the text of the YAML document. Runtime variables don't exist during that step. After step 1, template parameters have been resolved and no longer exist.

It also answers another common issue: why can't I use [variables](#) to resolve service connection / environment names? Resources are authorized before a stage can start running, so stage- and job-level variables aren't

available. Pipeline-level variables can be used, but only those explicitly included in the pipeline. Variable groups are themselves a resource subject to authorization, so their data is likewise not available when checking resource authorization.

Request an agent

Whenever Azure Pipelines needs to run a job, it will ask the [pool](#) for an [agent](#). ([Server jobs](#) are an exception, since they run on the Azure Pipelines server itself.) [Microsoft-hosted](#) and [self-hosted](#) agent pools work slightly differently.

Microsoft-hosted agent pool requests

First, the service checks on your organization's parallel jobs. It adds up all running jobs on all Microsoft-hosted agents and compares that with the number of parallel jobs purchased. If there are no available parallel slots, the job has to wait on a slot to free up.

Once a parallel slot is available, the job is routed to the requested agent type. Conceptually, the Microsoft-hosted pool is one giant, global pool of machines. (In reality, it's many different physical pools split by geography and operating system type.) Based on the `vmImage` (in YAML) or pool name (in the classic editor) requested, an agent is selected.

All agents in the Microsoft pool are fresh, new virtual machines that haven't run any pipelines before. When the job completes, the agent VM will be discarded.

Self-hosted agent pool requests

Similar to the [Microsoft-hosted pool](#), the service first checks on your organization's parallel jobs. It adds up all running jobs on all self-hosted agents and compares that with the number of parallel jobs purchased. If there are no available parallel slots, the job has to wait on a slot to free up.

Once a parallel slot is available, the self-hosted pool is examined for a compatible agent. Self-hosted agents offer [capabilities](#), which are strings indicating that particular software is installed or settings are configured. The pipeline has [demands](#), which are the capabilities required to run the job. If a free agent whose capabilities match the pipeline's demands cannot be found, the job will continue waiting. If there are no agents in the pool whose capabilities match the demands, the job will fail.

Self-hosted agents are typically reused from run to run. This means that a pipeline job can have side effects: warming up caches, having most commits already available in the local repo, and so on.

Prepare to run a job

Once an agent has accepted a job, it has some preparation work to do. The agent downloads (and caches for next time) all the [tasks](#) needed to run the job. It creates working space on disk to hold the source code, artifacts, and outputs used in the run. Then it begins [running steps](#).

Run each step

Steps are run sequentially, one after another. Before a step can start, all the previous steps must be finished (or skipped).

Steps are implemented by [tasks](#). Tasks themselves are implemented as Node.js or PowerShell scripts. The task system routes inputs and outputs to the backing scripts. It also provides some common services such as altering the system path and creating new [pipeline variables](#).

Each step runs in its own process, isolating it from the environment left by previous steps. Because of this

process-per-step model, environment variables are not preserved between steps. However, tasks and scripts have a mechanism to communicate back to the agent: [logging commands](#). When a task or script writes a logging command to standard out, the agent will take whatever action is requested.

There is an agent command to create new pipeline variables. Pipeline variables will be automatically converted into environment variables in the next step. In order to set a new variable `myVar` with a value of `myValue`, a script can do this:

```
echo '##vso[task.setVariable variable=myVar]myValue'
```

```
Write-Host "##vso[task.setVariable variable=myVar]myValue"
```

Report and collect results

Each step can report warnings, errors, and failures. Errors and warnings are reported to the pipeline summary page, marking the task as "succeeded with issues". Failures are also reported to the summary page, but they mark the task as "failed". A step is a failure if it either explicitly reports failure (using a `##vso` command) or ends the script with a non-zero exit code.

As steps run, the agent is constantly sending output lines to the service. That's why you can see a live feed of the console. At the end of each step, the entire output from the step is also uploaded as a log file. Logs can be downloaded once the pipeline has finished. Other items that the agent can upload include [artifacts](#) and [test results](#). These are also available after the pipeline completes.

State and conditions

The agent keeps track of each step's success or failure. As steps succeed with issues or fail, the job's status will be updated. The job always reflects the "worst" outcome from each of its steps: if a step fails, the job also fails.

Before running a step, the agent will check that step's [condition](#) to determine whether it should run. By default, a step will only run when the job's status is succeeded or succeeded with issues. Many jobs have cleanup steps that need to run no matter what else happened, so they can specify a condition of "always()". Cleanup steps might also be set to run only on [cancellation](#). A succeeding cleanup step cannot save the job from failing; jobs can never go back to success after entering failure.

Timeouts and disconnects

Each job has a timeout. If the job has not completed in the specified time, the server will cancel the job. It will attempt to signal the agent to stop, and it will mark the job as canceled. On the agent side, this means canceling all remaining steps and uploading any remaining [results](#).

Jobs have a grace period known as the cancel timeout in which to complete any cancellation work. (Remember, steps can be marked to run [even on cancellation](#).) After the timeout plus the cancel timeout, if the agent has not reported that work has stopped, the server will mark the job as a failure.

Because Azure Pipelines distributes work to agent machines, from time to time, agents may stop responding to the server. This can happen if the agent's host machine goes away (power loss, VM turned off) or if there's a network failure. To help detect these conditions, the agent sends a heartbeat message once per minute to let the server know it's still operating. If the server doesn't receive a heartbeat for five consecutive minutes, it assumes the agent will not come back. The job is marked as a failure, letting the user know they should retry the pipeline.

Manage runs through the CLI

Using the Azure DevOps CLI, you can list the pipeline runs in your project and view details about a specific run. You can also add and delete tags in your pipeline run.

Prerequisites

- You must have installed the Azure DevOps CLI extension as described in [Get started with Azure DevOps CLI](#).
- Sign into Azure DevOps using `az login`.
- For the examples in this article, set the default organization using
`az devops configure --defaults organization=YourOrganizationURL`.

List pipeline runs

List the pipeline runs in your project with the `az pipelines runs list` command. To get started, see [Get started with Azure DevOps CLI](#).

```
az pipelines runs list [--branch]
                      [--org]
                      [--pipeline-ids]
                      [--project]
                      [--query-order {FinishTimeAsc, FinishTimeDesc, QueueTimeAsc, QueueTimeDesc,
StartTimeAsc, StartTimeDesc}]
                      [--reason {all, batchedCI, buildCompletion, checkInShelveset, individualCI, manual,
pullRequest, schedule, triggered, userCreated, validateShelveset}]
                      [--requested-for]
                      [--result {canceled, failed, none, partiallySucceeded, succeeded}]
                      [--status {all, cancelling, completed, inProgress, none, notStarted, postponed}]
                      [--tags]
                      [--top]
```

Optional parameters

- **branch**: Filter by builds for this branch.
- **org**: Azure DevOps organization URL. You can configure the default organization using
`az devops configure -d organization=ORG_URL`. Required if not configured as default or picked up using
`git config`. Example: `--org https://dev.azure.com/MyOrganizationName/`.
- **pipeline-ids**: Space-separated IDs of definitions for which to list builds.
- **project**: Name or ID of the project. You can configure the default project using
`az devops configure -d project=NAME_OR_ID`. Required if not configured as default or picked up using
`git config`.
- **query-order**: Define the order in which pipeline runs are listed. Accepted values are *FinishTimeAsc*, *FinishTimeDesc*, *QueueTimeAsc*, *QueueTimeDesc*, *StartTimeAsc*, and *StartTimeDesc*.
- **reason**: Only list builds for this specified reason. Accepted values are *batchedCI*, *buildCompletion*, *checkInShelveset*, *individualCI*, *manual*, *pullRequest*, *schedule*, *triggered*, *userCreated*, and *validateShelveset*.
- **requested-for**: Limit to the builds requested for a specified user or group.
- **result**: Limit to the builds with a specified result. Accepted values are *canceled*, *failed*, *none*, *partiallySucceeded*, and *succeeded*.
- **status**: Limit to the builds with a specified status. Accepted values are *all*, *cancelling*, *completed*, *inProgress*, *none*, *notStarted*, and *postponed*.
- **tags**: Limit to the builds with each of the specified tags. Space separated.
- **top**: Maximum number of builds to list.

Example

The following command lists the first three pipeline runs that have a status of **completed** and a result of **succeeded**, and returns the result in table format.

```
az pipelines runs list --status completed --result succeeded --top 3 --output table
```

Run ID	Number	Status	Result	Pipeline ID	Pipeline Name	Source Branch	Branch
Queued Time			Reason				
125	20200124.1	completed	succeeded	12	Githubname.pipelines-java	master	2020-
01-23 18:56:10.067588		manual					
123	20200123.2	completed	succeeded	12	Githubname.pipelines-java	master	2020-
01-23 11:55:56.633450		manual					
122	20200123.1	completed	succeeded	12	Githubname.pipelines-java	master	2020-
01-23 11:48:05.574742		manual					

Show pipeline run details

Show the details for a pipeline run in your project with the [az pipelines runs show](#) command. To get started, see [Get started with Azure DevOps CLI](#).

```
az pipelines runs show --id
  [--open]
  [--org]
  [--project]
```

Parameters

- id:** Required. ID of the pipeline run.
- open:** Optional. Opens the build results page in your web browser.
- org:** Azure DevOps organization URL. You can configure the default organization using
`az devops configure -d organization=ORG_URL`. Required if not configured as default or picked up using
`git config`. Example: `--org https://dev.azure.com/MyOrganizationName/`.
- project:** Name or ID of the project. You can configure the default project using
`az devops configure -d project=NAME_OR_ID`. Required if not configured as default or picked up using
`git config`.

Example

The following command shows details for the pipeline run with the ID 123 and returns the results in table format. It also opens your web browser to the build results page.

```
az pipelines runs show --id 122 --open --output table
```

Run ID	Number	Status	Result	Pipeline ID	Pipeline Name	Source Branch	Branch
Queued Time			Reason				
123	20200123.2	completed	succeeded	12	Githubname.pipelines-java	master	2020-
01-23 11:55:56.633450		manual					

Add tag to pipeline run

Add a tag to a pipeline run in your project with the [az pipelines runs tag add](#) command. To get started, see [Get started with Azure DevOps CLI](#).

```
az pipelines runs tag add --run-id
  --tags
  [--org]
  [--project]
```

Parameters

- **run-id**: Required. ID of the pipeline run.
- **tags**: Required. Tags to be added to the pipeline run (comma-separated values).
- **org**: Azure DevOps organization URL. You can configure the default organization using
`az devops configure -d organization=ORG_URL`. Required if not configured as default or picked up using
`git config`. Example: `--org https://dev.azure.com/MyOrganizationName/`.
- **project**: Name or ID of the project. You can configure the default project using
`az devops configure -d project=NAME_OR_ID`. Required if not configured as default or picked up using
`git config`.

Example

The following command adds the tag **YAML** to the pipeline run with the ID **123** and returns the result in JSON format.

```
az pipelines runs tag add --run-id 123 --tags YAML --output json

[
  "YAML"
]
```

List pipeline run tags

List the tags for a pipeline run in your project with the [az pipelines runs tag list](#) command. To get started, see [Get started with Azure DevOps CLI](#).

```
az pipelines runs tag list --run-id
  [--org]
  [--project]
```

Parameters

- **run-id**: Required. ID of the pipeline run.
- **org**: Azure DevOps organization URL. You can configure the default organization using
`az devops configure -d organization=ORG_URL`. Required if not configured as default or picked up using
`git config`. Example: `--org https://dev.azure.com/MyOrganizationName/`.
- **project**: Name or ID of the project. You can configure the default project using
`az devops configure -d project=NAME_OR_ID`. Required if not configured as default or picked up using
`git config`.

Example

The following command lists the tags for the pipeline run with the ID **123** and returns the result in table format.

```
az pipelines runs tag list --run-id 123 --output table

Tags
-----
YAML
```

Delete tag from pipeline run

Delete a tag from a pipeline run in your project with the [az pipelines runs tag delete](#) command. To get started, see [Get started with Azure DevOps CLI](#).

```
az pipelines runs tag delete --run-id  
    --tag  
    [--org]  
    [--project]
```

Parameters

- **run-id**: Required. ID of the pipeline run.
- **tag**: Required. Tag to be deleted from the pipeline run.
- **org**: Azure DevOps organization URL. You can configure the default organization using
`az devops configure -d organization=ORG_URL`. Required if not configured as default or picked up using
`git config`. Example: `--org https://dev.azure.com/MyOrganizationName/`.
- **project**: Name or ID of the project. You can configure the default project using
`az devops configure -d project=NAME_OR_ID`. Required if not configured as default or picked up using
`git config`.

Example

The following command deletes the **YAML** tag from the pipeline run with ID 123.

```
az pipelines runs tag delete --run-id 123 --tag YAML
```

Access repositories, artifacts, and other resources

3/25/2021 • 13 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

At run-time, each job in a pipeline may access other resources in Azure DevOps. For example, a job may:

- Check out source code from a Git repository
- Add a tag to the repository
- Access a feed in Azure Artifacts
- Upload logs from the agent to the service
- Upload test results and other artifacts from the agent to the service
- Update a work item

Azure Pipelines uses job access tokens to perform these tasks. A **job access token** is a security token that is dynamically generated by Azure Pipelines for each job at run time. The agent on which the job is running uses the job access token in order to access these resources in Azure DevOps. You can control which resources your pipeline has access to by controlling how permissions are granted to job access tokens.

The token's permissions are derived from (a) job authorization scope and (b) the permissions you set on project or collection build service account.

Job authorization scope

You can set the job authorization scope to be **collection** or **project**. By setting the scope to **collection**, you choose to let pipelines access all repositories in the collection or organization. By setting the scope to **project**, you choose to restrict access to only those repositories that are in the same project as the pipeline.

- [YAML](#)
- [Classic](#)

Job authorization scope can be set for the entire Azure DevOps organization or for a specific project.

NOTE

In Azure DevOps Server 2020, **Limit job authorization scope to current project** applies only to YAML pipelines and classic build pipelines. It does not apply to classic release pipelines. Classic release pipelines always run with project collection scope.

To set job authorization scope for the organization:

- Navigate to your organization settings page in the Azure DevOps user interface.
- Select **Settings** under **Pipelines**.
- Enable **Limit job authorization scope to current project** to limit the scope to project. This is the

recommended setting, as it enhances security for your pipelines.

To set job authorization scope for a specific project:

- Navigate to your project settings page in the Azure DevOps user interface.
- Select **Settings** under **Pipelines**.
- Enable **Limit job authorization scope to current project** to limit the scope to project. This is the recommended setting, as it enhances security for your pipelines.
- To set job authorization scope at the organization level for all projects, choose **Organization settings > Pipelines > Settings**.
- To set job authorization scope for a specific project, choose **Project settings > Pipelines > Settings**.

Enable one or more of the following settings. Enabling these settings are recommended, as it enhances security for your pipelines.

- **Limit job authorization scope to current project for non-release pipelines** - This setting applies to YAML pipelines and classic build pipelines, and does not apply to [classic release pipelines](#).
- **Limit job authorization scope to current project for release pipelines** - This setting applies to [classic release pipelines](#) only.

NOTE

If the scope is set to **project** at the organization level, you cannot change the scope in each project.

IMPORTANT

If the scope is not restricted at either the organization level or project level, then every job in your YAML pipeline gets a collection scoped job access token. In other words, your pipeline has access to any repository in any project of your organization. If an adversary is able to gain access to a single pipeline in a single project, they will be able to gain access to any repository in your organization. This is why, it is recommended that you restrict the scope at the highest level (organization settings) in order to contain the attack to a single project.

If you use Azure DevOps Server 2019, then all YAML jobs run with the job authorization scope set to **collection**. In other words, these jobs have access to all repositories in your project collection. You cannot change this in Azure DevOps Server 2019.

YAML pipelines are not available in TFS.

NOTE

If your pipeline is in a **public project**, then the job authorization scope is automatically restricted to **project** no matter what you configure in any setting. Jobs 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 organization.

Limit job authorization scope to referenced Azure DevOps repositories

In addition to the job authorization scope settings described in the previous section, Azure Pipelines provides a **Limit job authorization scope to referenced Azure DevOps repositories** setting.

Pipelines can access any Azure DevOps repositories in authorized projects unless **Limit job authorization scope to referenced Azure DevOps repositories** is enabled. With this option enabled, you can reduce the scope of access for all pipelines to only Azure DevOps repositories explicitly referenced by a `checkout` step or a `uses` statement in the pipeline job that uses that repository.

For more information, see [Azure Repos Git repositories - Limit job authorization scope to referenced Azure DevOps repositories](#).

IMPORTANT

Limit job authorization scope to referenced Azure DevOps repositories is enabled by default for new organizations and projects created after May 2020.

Scoped build identities

Azure DevOps uses two built-in identities to execute pipelines.

- A **collection-scoped identity**, which has access to all projects in the collection (or organization for Azure DevOps Services)
- A **project-scoped identity**, which has access to a single project

These identities are allocated permissions necessary to perform build/release execution time activities when calling back to the Azure DevOps system. There are built-in default permissions, and you may also manage your own permissions as needed.

The **collection-scoped identity** name has the following format:

- `Project Collection Build Service ({OrgName})`
- For example, if the organization name is `fabrikam-tailspin`, this account has the name `Project Collection Build Service (fabrikam-tailspin)`.

The **project-scoped identity** name has the following format:

- `{Project Name} Build Service ({Org Name})`
- For example, if the organization name is `fabrikam-tailspin` and the project name is `SpaceGameWeb`, this account has the name `SpaceGameWeb Build Service (fabrikam-tailspin)`.

By default, the collection-scoped identity is used, unless configured otherwise as described in the previous [Job authorization scope](#) section.

Manage build service account permissions

One result of setting project-scoped access may be that the project-scoped identity may not have permissions to a resource that the collection-scoped one did have.

You may want to change the permissions of job access token in scenarios such as the following:

- You want your pipeline to access a feed that is in a different project.
- You want your pipeline to be restricted from changing code in the repository.
- You want your pipeline to be restricted from creating work items.

To update the permissions of the job access token:

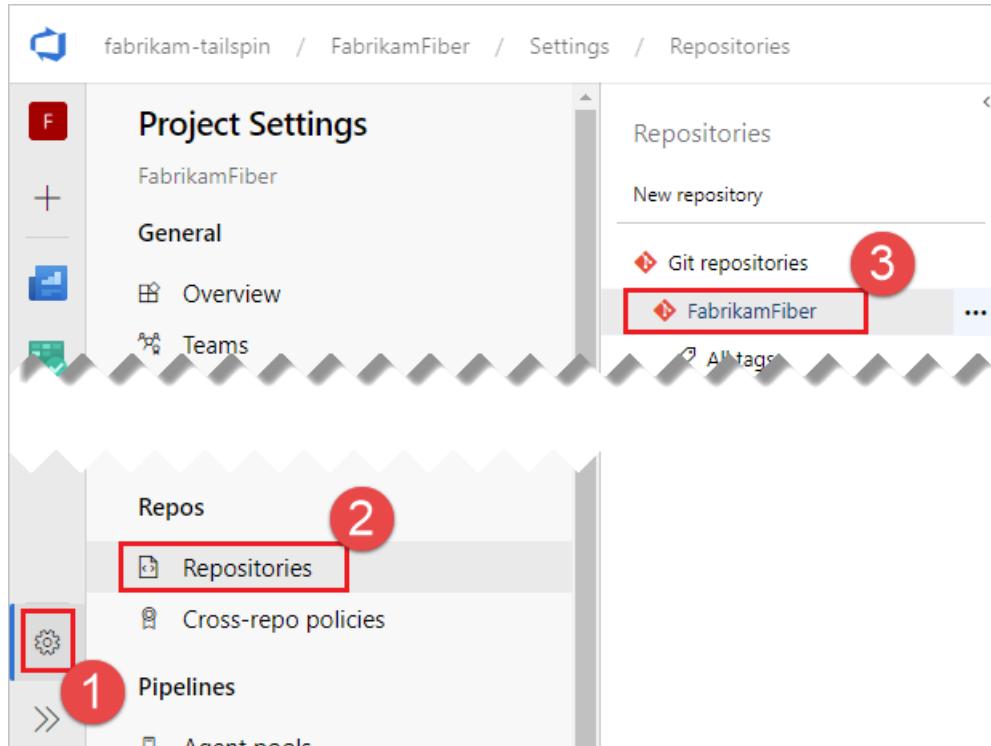
- First, determine the job authorization scope for your pipeline. See the section above to understand job authorization scope. If the job authorization scope is **collection**, then the corresponding build service account to manage permissions on is **Project Collection Build Service (your-collection-name)**. If the job authorization scope is **project**, then the build service account to manage permissions on is **Your-project-name Build Service (your-collection-name)**.
- To restrict or grant additional access to **Project Collection Build Service (your-collection-name)**:

- Select **Manage security** in the overflow menu on **Pipelines** page.
- Under **Users**, select **Project Collection Build Service (your-collection-name)**.
- Make any changes to the pipelines-related permissions for this account.
- Navigate to organization settings for your Azure DevOps organization (or collection settings for your project collection).
- Select **Permissions under Security**.
- Under the **Users** tab, look for **Project Collection Build Service (your-collection-name)**.
- Make any changes to the non-pipelines-related permissions for this account.
- Since **Project Collection Build Service (your-collection-name)** is a user in your organization or collection, you can add this account explicitly to any resource - for e.g., to a feed in Azure Artifacts.
- To restrict or grant additional access to **Your-project-name Build Service (your-collection-name)**:
 - The build service account on which you can manage permissions will only be created after you run the pipeline once. Make sure that you already ran the pipeline once.
 - Select **Manage security** in the overflow menu on **Pipelines** page.
 - Under **Users**, select **Your-project-name Build Service (your-collection-name)**.
 - Make any changes to the pipelines-related permissions for this account.
 - Navigate to organization settings for your Azure DevOps organization (or collection settings for your project collection).
 - Select **Permissions under Security**.
 - Under the **Users** tab, look for **Your-project-name build service (your-collection-name)**.
 - Make any changes to the non-pipelines-related permissions for this account.
 - Since **Your-project-name Build Service (your-collection-name)** is a user in your organization or collection, you can add this account explicitly to any resource - for e.g., to a feed in Azure Artifacts.

Example - Configure permissions to access another repo in the same project project collection

In this example, the `fabrikam-tailspin/SpaceGameWeb` project-scoped build identity is granted permission to access the `FabrikamFiber` repository in the `fabrikam-tailspin/FabrikamFiber` project.

1. In the **FabrikamFiber** project, navigate to **Project settings**, **Repositories**, **FabrikamFiber**.



2. Choose the + icon, start to type in the name **SpaceGameWeb**, and select the **SpaceGameWeb Build**

Service account.

Security for FabrikamFiber repository

Security

Inheritance ⓘ

+

SpaceGameWeb

[SpaceGameWeb]\SpaceGameWeb Team

SpaceGameWeb Build Service (fabrikam-tails...) 38128e35-9da4-4a3b-8730-cac6c2495f93

C Contributors

3. Configure the desired permissions for that user.

Security for FabrikamFiber repository

Security

Inheritance ⓘ

+

Filter

Azure DevOps Groups

Users

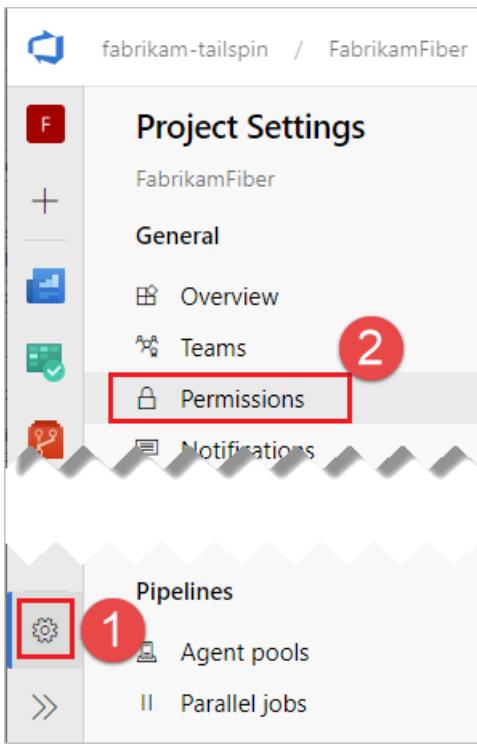
SpaceGameWeb Build Service (fabrikam-tailspin)

Permission	Setting
Bypass policies when completing pull requests	Not set
Bypass policies when pushing	Not set
Contribute	Not set
Contribute to pull requests	Not set
Create branch	Not set
Create tag	Not set
Delete repository	Not set

Example - Configure permissions to access other resources in the same project collection

In this example, the `fabrikam-tailspin/SpaceGameWeb` project-scoped build identity is granted permissions to access other resources in the `fabrikam-tailspin/FabrikamFiber` project.

1. In the **FabrikamFiber** project, navigate to **Project settings, Permissions**.



2. Choose **Users**, start to type in the name **SpaceGameWeb**, and select the **SpaceGameWeb Build Service** account. If you don't see any search results initially, select **Expand search**.

The screenshot shows the 'Permissions' page with the 'Users' tab selected (red circle 1). A search bar contains the text 'SpaceGameWeb' (red circle 2). Below the search bar, it says 'Total 1 Filtered 1'. A search result for 'SpaceGameWeb Build Service (fabrikam-tailspin)' is listed (red circle 3).

3. Configure the desired permissions for that user.



SpaceGameWeb Build Service (fabrikam-tailspin)

Permissions Member of

General

Delete team project	Not set
Edit project-level information	Not set
Manage project properties	Not set
Rename team project	Not set
Suppress notifications for work item updates	Not set
Update project visibility	Not set
View project-level information	Not set

Boards

Bypass rules on work item updates	Not set
Change process of team project.	Not set

FAQ

How do I determine the job authorization scope of my YAML pipeline?

- If your project is a public project, the job authorization scope is always **project** regardless of any other settings.

All YAML pipelines in Azure DevOps Server 2019 run under **collection** job authorization scope.

- Check the Pipeline settings under your **Azure DevOps Organization settings**:
 - If **Limit job authorization scope to current project** is enabled, then the scope is **project**.
 - If **Limit job authorization scope to current project** is not enabled, then check the Pipeline settings under your **Project settings** in Azure DevOps:
 - If **Limit job authorization scope to current project** is enabled, then the scope is **project**.
 - Otherwise, the scope is **collection**.
- If the pipeline is in a private project, check the Pipeline settings under your **Azure DevOps Organization settings**:
 - If **Limit job authorization scope to current project for non-release pipelines** is enabled, then the scope is **project**.
 - If **Limit job authorization scope to current project for non-release pipelines** is not enabled, then check the Pipeline settings under your **Project settings** in Azure DevOps:
 - If **Limit job authorization scope to current project for non-release pipelines** is enabled, then the scope is **project**.
 - Otherwise, the scope is **collection**.

How do I determine the job authorization scope of my classic build pipeline?

- If the pipeline is in a public project, then the job authorization scope is **project** regardless of any other settings.

- Open the editor for the pipeline and navigate to the **Options** tab.
 - If the **Build job authorization scope** is **Current project**, then scope is **project**.
 - Otherwise, scope is **collection**.
- Check the Pipeline settings under your Azure DevOps **Organization settings**:
 - If **Limit job authorization scope to current project** is enabled, then the scope is **project**.
 - If **Limit job authorization scope to current project** is not enabled, then check the Pipeline settings under your **Project settings** in Azure DevOps:
 - If **Limit job authorization scope to current project** is enabled, then the scope is **project**.
 - If **Limit job authorization scope to current project** is not enabled, open the editor for the pipeline, and navigate to the **Options** tab.
 - If the **Build job authorization scope** is **Current project**, then scope is **project**.
 - Otherwise, scope is **collection**.
- If the pipeline is in a private project, check the Pipeline settings under your Azure DevOps **Organization settings**:
 - If **Limit job authorization scope to current project for non-release pipelines** is enabled, then the scope is **project**.
 - If **Limit job authorization scope to current project for non-release pipelines** is not enabled, then check the Pipeline settings under your **Project settings** in Azure DevOps:
 - If **Limit job authorization scope to current project for non-release pipelines** is enabled, then the scope is **project**.
 - If **Limit job authorization scope to current project for non-release pipelines** is not enabled, open the editor for the pipeline, and navigate to the **Options** tab.
 - If the **Build job authorization scope** is **Current project**, then scope is **project**.
 - Or else, scope is **collection**.

How do I determine the job authorization scope of my classic release pipeline?

Classic release pipelines in Azure DevOps Server 2020 and below run with **collection** scope.

- If the pipeline is in a public project, then the job authorization scope is **project** regardless of any other settings.
- If the pipeline is in a private project, check the Pipeline settings under your Azure DevOps **Organization settings**:
 - If **Limit job authorization scope to current project for release pipelines** is enabled, then the scope is **project**.
 - If **Limit job authorization scope to current project for release pipelines** is not enabled, then check the Pipeline settings under your **Project settings** in Azure DevOps:
 - If **Limit job authorization scope to current project for release pipelines** is enabled, then the scope is **project**.
 - Otherwise, the scope is **collection**.

Pipeline reports

11/2/2020 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019

Teams track their pipeline health and efficiency to ensure continuous delivery to their customers. You can gain visibility into your team's pipeline(s) using Pipeline analytics. The source of information for pipeline analytics is the set of runs for your pipeline. These analytics are accrued over a period of time, and form the basis of the rich insights offered. Pipelines reports show you metrics, trends, and can help you identify insights to improve the efficiency of your pipeline.

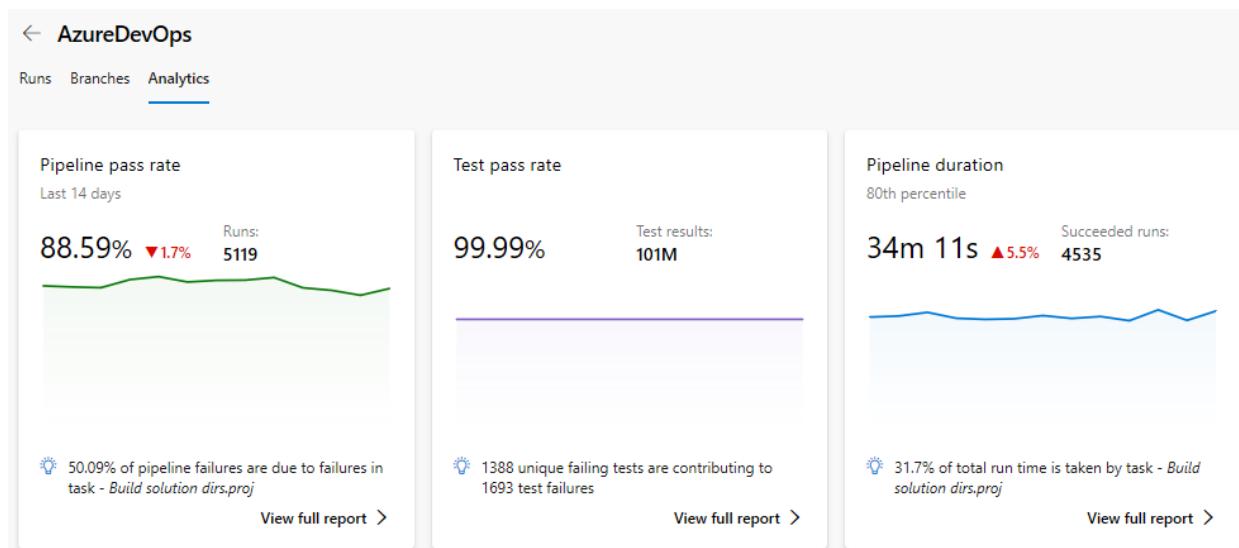
Prerequisites

Ensure that you have installed the [Analytics Marketplace extension](#) for Azure DevOps Server.

View pipeline reports

A summary of the pass rate can be viewed in the **Analytics** tab of a pipeline. To drill into the trend and insights, click on the card to view the full report.

A summary of the pass rate and duration can be viewed in the **Analytics** tab of a pipeline. To drill into the trend and insights, click on the card to view the full report.



Runs Branches Analytics

Test failures

Last 14 days

Pass rate:

99.99%

Test results:

154K

💡 3371 unique failing tests are contributing to 5453 test failures

Pipeline pass rate report

The **Pipeline pass rate** report provides a granular view of the pipeline pass rate and its trend over time. You can also view which specific task failure contributes to a high number of pipeline run failures, and use that insight to fix the top failing tasks.

The report contain the following sections:

- **Summary:** Provides the key metrics of pass rate of the pipeline over the specified period. The default view shows data for 14 days, which you can modify.

Pass rate trend



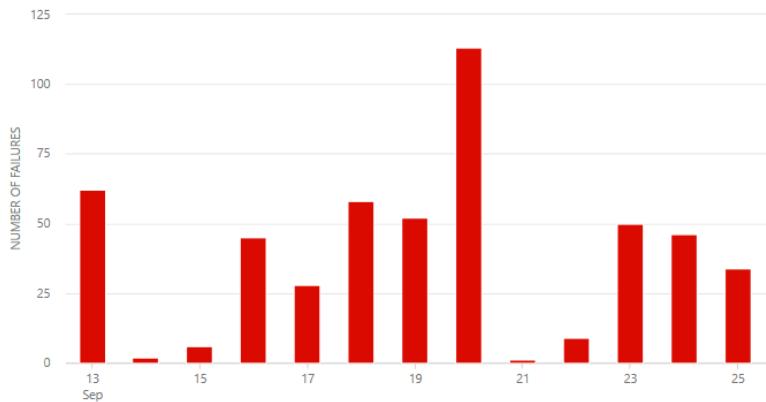
88.45% ↑ 5.9%
3,938 of 4,452 passed



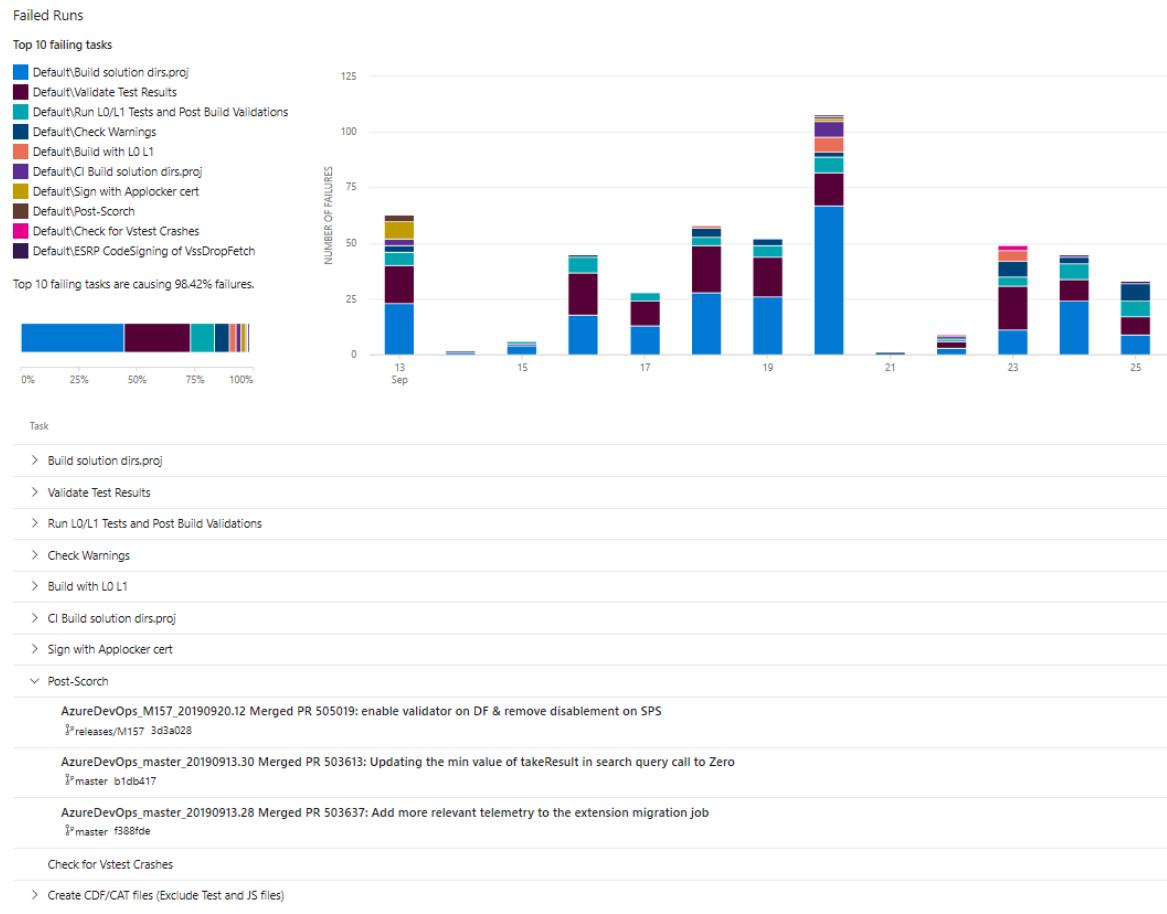
- **Failure trend:** Shows the number of failures per day. This data is divided by stages if multiple stages are applicable for the pipeline.

Failure Trend

💡 44.77% of pipeline failures are due to failures in task - `Build solution dirs.proj`



- **Top failing tasks & their failed runs:** Lists the top failing tasks, their trend and provides pointers to their failed runs. Analyze the failures in the build to fix your failing task and improve the pass rate of the pipeline.



Pipeline duration report

The **Pipeline duration** report shows how long your pipeline typically takes to complete successfully. You can review the duration trend and analyze the top tasks by duration to optimize the duration of the pipeline.

Pipeline duration

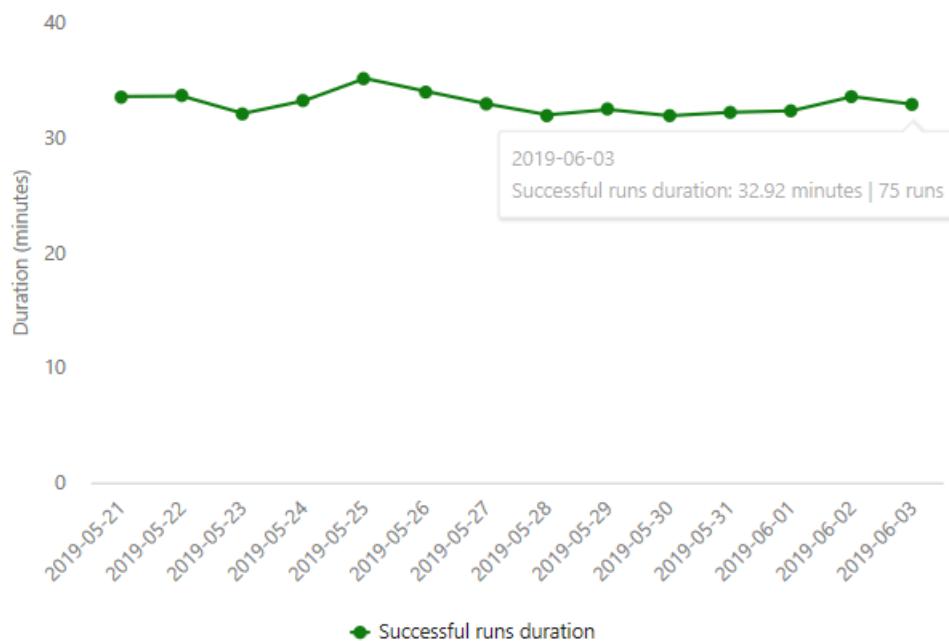
80th percentile

32 mins | 2585 succeeded runs

↑ 20.4% in the last 14 days

Task 'Check Test Warnings' duration has increased by 6 min (29.26%) in the last 14 days

Pipeline duration trend



Test failures report

The **Test failures** report provides a granular view of the top failing tests in the pipeline, along with the failure details. For more information on this report, see [Test failures](#).

Release pipelines > RM.CDP > Test Failures Report ▾

Summary ▾

Pass rate

97.52%

175K Passed
4.4K Failed
4.9K Not executed

Unique failing tests

86 tests causing 4,437 failed test results

Trend of test results and pass rate ▾

Failed result count

Pass rate percentage

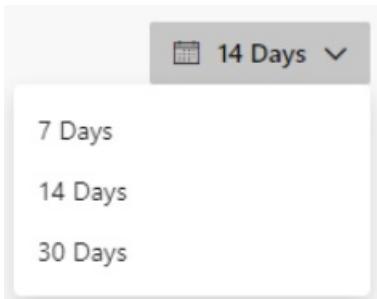
Test

Test	Failed	Pass rate	Total count	Average duration
AgentBasedPipelineShouldBeAbleToSuccessfullyReleaseWithExternalTfsArtifact	1108	53.65%	2391	123.16s
SelectiveArtifactsDownloadTest	980	52.88%	2622	69.87s
AddDeletePhaseTaskV2	403	47.66%	770	72.56s
RmoDevFabricUpgrade	263	3.3%	272	404.9s
VariableGroupV2	239	71.64%	843	66.14s

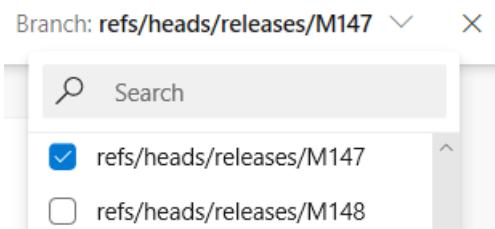
Filters

Pipelines reports can be further filtered by date range or branch.

- **Date range:** The default view shows data from the last 14 days. The filter helps change this range.



- **Branch filter:** View the report for a particular branch or a set of branches.



Help and support

- See our [troubleshooting](#) page
- Get advice on [Stack Overflow](#), and get support via the [Developer Community](#)

Add widgets to a dashboard

11/2/2020 • 7 minutes to read • [Edit Online](#)

Azure DevOps Services | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

Widgets smartly format data to provide access to easily consumable data. You add widgets to your team dashboards to gain visibility into the status and trends occurring as you develop your software project.

Each widget provides access to a chart, user-configurable information, or a set of links that open a feature or function. You can add one or more charts or widgets to your dashboard. Up to 200 widgets total. You add several widgets at a time simply by selecting each one. See [Manage dashboards](#) to determine the permissions you need to add and remove widgets from a dashboard.

Prerequisites

- You must be a member of a project. If you don't have a project yet, [create one](#).
- If you haven't been added as a project member, [get added now](#).
- Anyone with access to a project, including [stakeholders](#), can view dashboards.
- To add, edit, or manage a team dashboard, you must have **Basic** access or greater and be a [team admin](#), a project admin, or have [dashboard permissions](#). In general, you need to be a team member for the currently selected team to edit dashboards.
- You must be a member of a project. If you don't have a project yet, [create one](#).
- If you haven't been added as a project member, [get added now](#).
- Anyone with access to a project, including [stakeholders](#), can view dashboards.
- To add, edit, or manage a team dashboard, you must have **Basic** access or greater and be a team admin, a project admin, or have [dashboard permissions](#). In general, you need to be a team admin for the currently selected team to edit dashboards. Request your current team or project admin to add you as a [team admin](#).
- You must be a member of a project. If you don't have a project yet, [create one](#).
- If you haven't been added as a project member, [get added now](#).
- Anyone with access to a project, including [stakeholders](#), can view dashboards.
- To add, edit, or manage a team dashboard, you must have **Basic** access or greater and be added to the [team administrator role for the team](#).

NOTE

Widgets specific to a service are disabled if the service they depend on has been disabled. For example, if **Boards** is disabled, New Work item and all work tracking Analytics widgets are disabled and won't appear in the widget catalog. If Analytics is disabled or not installed, then all Analytics widgets are disabled.

To re-enable a service, see [Turn an Azure DevOps service on or off](#). For Analytics, see [enable or install Analytics](#).

Select a dashboard

All dashboards are associated with a team. You need to be a team administrator, project administrator, or a team member with permissions to modify a dashboard.

1. Open a web browser, connect to your project, and choose **Overview > Dashboards**. The dashboard directory page opens.

The screenshot shows the Azure DevOps interface for the 'Fabrikam Fiber' project. On the left, a sidebar lists various project sections: Overview, Summary, Dashboards (which is selected and highlighted with a red box), Analytics views, Wiki, Boards, Repos, Pipelines, Test Plans, and Artifacts. The main content area is titled 'Dashboards' and displays a list of available dashboards. The columns are 'Name' and 'Team'. The list includes:

Name	Team
Analytics	Fabrikam Team
My favorite dashboards (2)	
Overview	Account Management
Team Guidance	Fabrikam Team
Account Management (1)	
Overview	Account Management
Customer Profile (1)	
Overview	Customer Profile
Fabrikam Team (5)	
Analytics	Fabrikam Team
Bug status	Fabrikam Team

If you need to switch to a different project, choose the Azure DevOps logo to [browse all projects](#).

2. Choose the dashboard you want to modify.

Open a web browser, connect to your project, and choose **Dashboards**.

The screenshot shows the top navigation bar of the Azure DevOps interface for the 'Fabrikam Fiber' project. The tabs include: Overview (selected), Dashboards (highlighted with a red box), Code, Work, Build & Release, Test, and Wiki*. The 'Dashboards' tab is the active one.

Select the team whose dashboards you want to view. To switch your team focus, see [Switch project or team focus](#).

Choose the name of the dashboard to modify it.

For example, here we choose to view the Work in Progress dashboard.

The screenshot shows the top navigation bar for the 'Fabrikam Fiber / Web' project. The tabs include: Overview (selected), Bugs, Work in Progress (highlighted with a red box), and Test. The 'Work in Progress' tab is the active one.

If you need to switch to a different project, choose the Azure DevOps logo to [browse all projects](#).

Add a widget

To add widgets to the dashboard, choose  **Edit**.

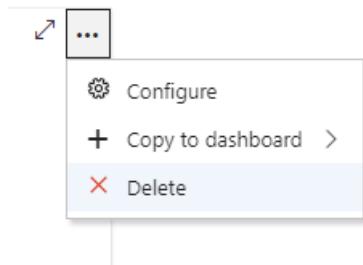
The widget catalog will automatically open. Add all the widgets that you want and drag their tiles into the sequence you want.

When you're finished with your additions, choose **Done Editing** to exit dashboard editing. This will dismiss the widget catalog. You can then [configure the widgets](#) as needed.

TIP

When you're in dashboard edit mode, you can remove, rearrange, and configure widgets, as well as add new widgets. Once you leave edit mode, the widget tiles remain locked, reducing the chances of accidentally moving a widget.

To remove a widget, choose the  actions icon and select the **Delete** option from the menu.



Choose  to modify a dashboard. Choose  to add a widget to the dashboard.

The [widget catalog](#) describes all the available widgets, many of which are scoped to the selected team context.

Or, you can drag and drop a widget from the catalog onto the dashboard.

Add an Analytics widget

This example shows how to add the Velocity widget available from Analytics to a dashboard.

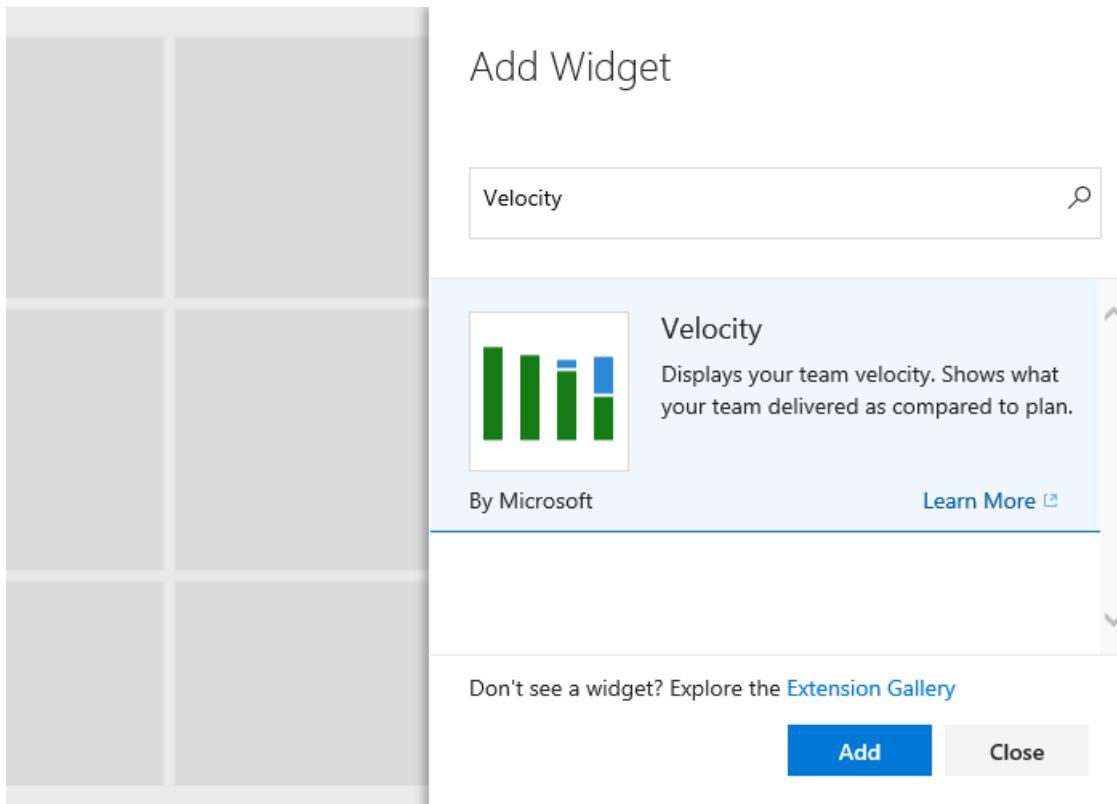
1. Connect to the web portal for your project and choose **Overview > Dashboards**.

The screenshot shows the Azure DevOps interface for the 'Fabrikam Fiber' project. The left sidebar lists various project management sections like Overview, Summary, Dashboards, Analytics views, Wiki, Boards, Repos, Pipelines, Test Plans, and Artifacts. The 'Dashboards' section is highlighted with a red box. The main content area is titled 'Dashboards' and displays a list of available dashboards, grouped by category. The categories include 'Continue where you left off', 'My favorite dashboards (2)', 'Account Management (1)', 'Customer Profile (1)', and 'Fabrikam Team (5)'. Each dashboard entry shows its name, a small icon, a star rating, and the team it belongs to.

Name	Team
Analytics	Fabrikam Team
Overview	Account Management
Team Guidance	Fabrikam Team
Overview	Account Management
Overview	Customer Profile
Analytics	Fabrikam Team
Bug status	Fabrikam Team

If you need to switch to a different project, choose the Azure DevOps logo to [browse all projects and teams](#).

2. Make sure that the [Analytics Marketplace extension](#) has been installed. The Analytics widgets won't be available until it is installed.
3. [Choose the dashboard](#) that you want to modify.
4. Choose Edit to modify a dashboard. The widget catalog opens.
5. In the right pane search box, type **Velocity** to quickly locate the Velocity widget within the widget catalog.



6. Choose the widget, then **Add** to add it to the dashboard. Or, you can drag-and-drop it onto the dashboard.

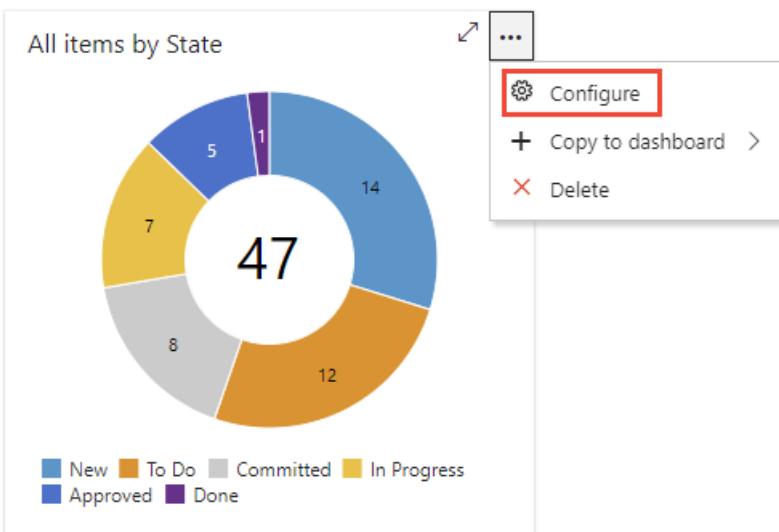
7. Next, configure the widget. For details, see the following articles:

- [Configure burndown or burnup](#)
- [Configure cumulative flow](#)
- [Configure lead/cycle time](#)
- [Configure velocity](#)
- [Configure test trend results](#)

Configure a widget

Most widgets support configuration, which may include specifying the title, setting the widget size, and other widget-specific variables.

To configure a widget, add the widget to a dashboard, choose open the **...** menu, and select **Configure**.



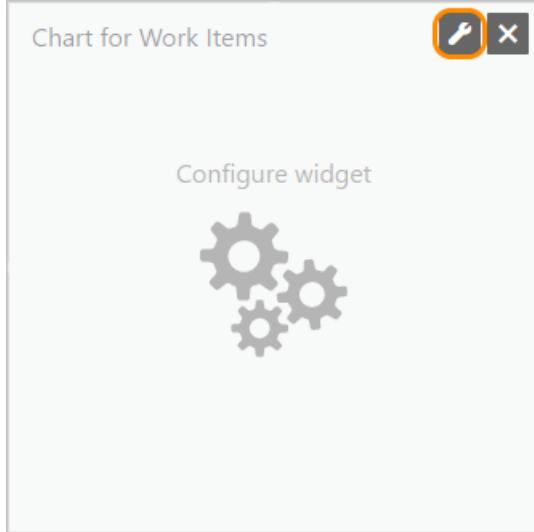
Additional information is provided to configure the following widgets:

- [Burndown/burnup](#)
- [Cumulative flow](#)
- [Lead time or cycle time](#)
- [Velocity widget](#)
- [Test trend results](#)

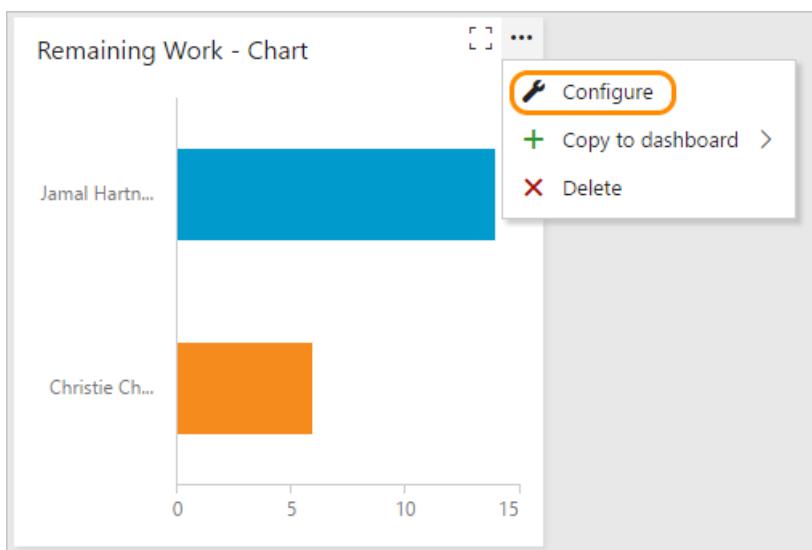
Additional information is provided to configure the following widgets:

- [Burndown/burnup](#)
- [Cumulative flow](#)
- [Lead time or cycle time](#)
- [Velocity widget](#)

To configure a widget, add the widget to a dashboard and then choose the  configure icon.



Once you've configured the widget, you can edit it by opening the actions menu.



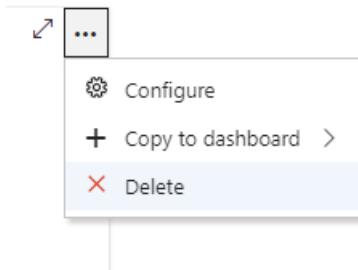
Move or delete a widget

To move a widget, you need to enable the dashboard edit mode. To delete a widget, simply select the delete option provided from the widget's options menu.

Just as you have to be a team or project admin to add items to a dashboard, you must have admin permissions to remove items.

Choose  **Edit** to modify your dashboard. You can then add widgets or drag tiles to reorder their sequence on the dashboard.

To remove a widget, choose the  actions icon and select the **Delete** option from the menu.

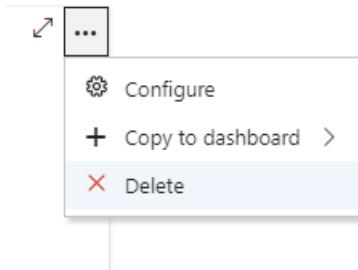


When you're finished with your changes, choose **Done Editing** to exit dashboard editing.



Choose  to modify your dashboard. You can then drag tiles to reorder their sequence on the dashboard.

To remove a widget, choose the  actions icon and select the **Delete** option from the menu.



To remove a widget, choose the widget's  or  delete icons.

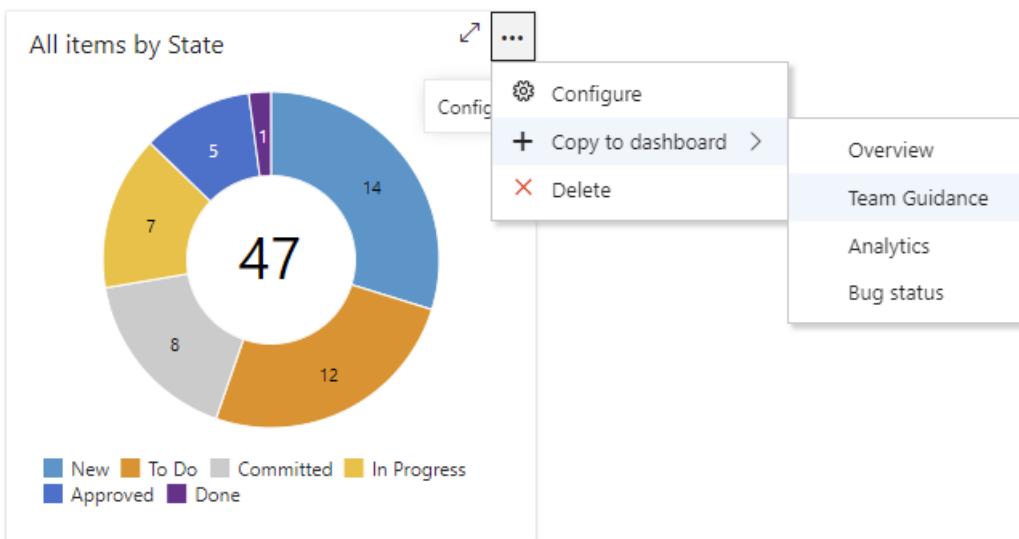


When you're finished with your changes, choose  to exit dashboard editing.

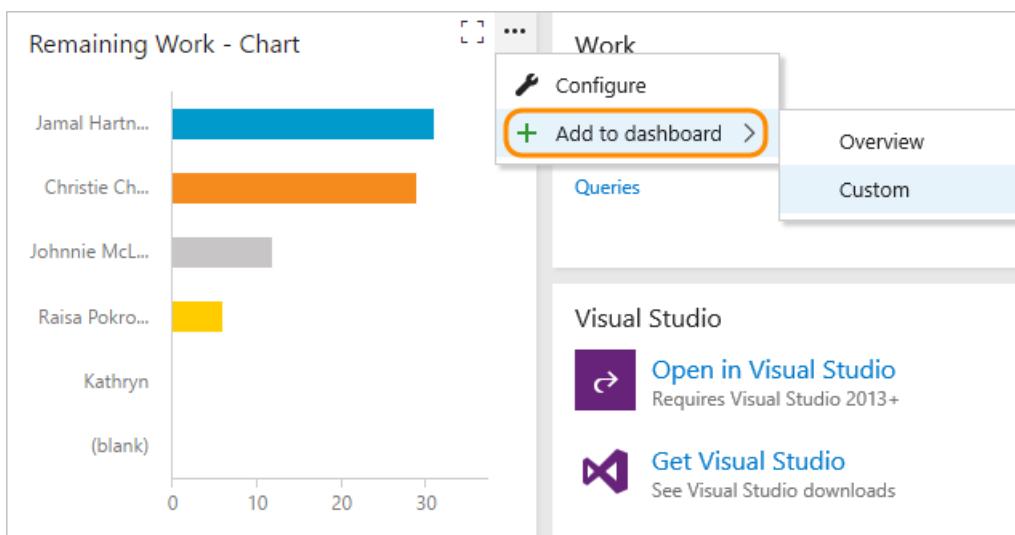
Copy a widget

You can copy a widget to the same dashboard or to another team dashboard. If you want to move widgets you have configured to another dashboard, this is how you do it. Before you begin, add the dashboard you want to copy or move the widget to. Once you've copied the widget, you can delete it from the current dashboard.

To copy a configured widget to another team dashboard, choose the  actions icon and select **Copy to dashboard** and then the dashboard to copy it to.



To copy a configured widget to another team dashboard, choose the **...** actions icon and select **Add to dashboard** and then the dashboard to copy it to.



Widget size

Some widgets are pre-sized and can't be changed. Others are configurable through their configuration dialog.

For example, the Chart for work items widget allows you to select an area size ranging from 2 x 2 to 4 x 4 (tiles).

Configuration

Title

Remaining Work - Chart

Size

2 x 2

2 x 2

2 x 3

2 x 4

3 x 2

3 x 3

3 x 4

4 x 2

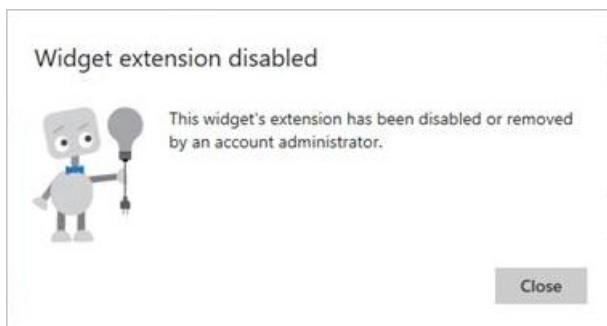
4 x 3

Extensibility and Marketplace widgets

In addition to the widgets described in the Widget catalog, you can add widgets from the [Marketplace](#), or create your own widgets using the [Widget REST APIs](#).

Disabled Marketplace widget

If your organization owner or project collection administrator disables a marketplace widget, you'll see the following image:



To regain access to it, request your admin to reinstate or reinstall the widget.

Try this next

[Review the widget catalog](#) or [Review Marketplace widgets](#)

Related articles

- [Analytics-based widgets](#)
- [What is Analytics?](#)
- [Burndown guidance](#)
- [Cumulative flow & lead/cycle time guidance](#)
- [Velocity guidance](#)
- [Burndown guidance](#)

- Cumulative flow & lead/cycle time guidance
- Velocity guidance

Widgets based on Analytics

3/25/2021 • 4 minutes to read • [Edit Online](#)

Azure DevOps Services | Azure DevOps Server 2020 | Azure DevOps Server 2019

Analytics supports several dashboard widgets that take advantage of the power of the service. Using these widgets, you and your team can gain valuable insights into the health and status of your work.

Analytics supports several dashboard widgets that take advantage of the power of the service. Once you [enable or install Analytics](#) on a project collection, you can add these widgets to your dashboard. You must be an organization owner or a member of the [Project Collection Administrator group](#) to add extensions or enable the service. Using these widgets, you and your team can gain valuable insights into the health and status of your work.

You add an Analytics widget to a dashboard the same way you add any other type of widget. For details, see [Add a widget to your dashboard](#).

Prerequisites

- Analytics widget data is calculated from the Analytics service.
- For Analytics widgets to display correctly, the Analytics service must be enabled on your on-premises Azure DevOps Server. To learn more, see [Install/uninstall or enable/disable the Analytics service](#).

NOTE

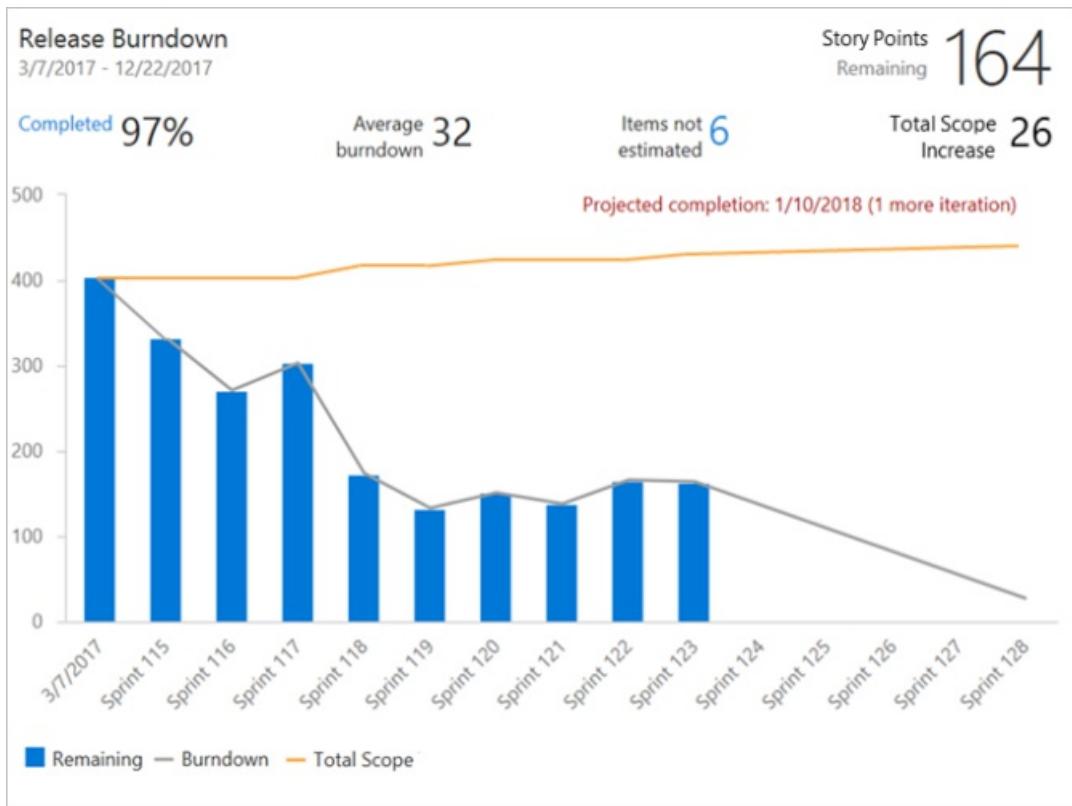
If **Boards** is disabled, then **Analytics views** will also be disabled and all widgets associated with work item tracking won't appear in the widget catalog and will become disabled. To re-enable a service, see [Turn an Azure DevOps service on or off](#).

Burndown

The Burndown widget lets you display a trend of remaining work across multiple teams and multiple sprints. You can use it to create a release burndown, a bug burndown, or a burndown on any scope of work over time. It will help you answer questions like:

- Will we complete the scope of work by the targeted completion date? If not, what is the projected completion date?
- What kind of scope creep does my project have?
- What is the projected completion date for my project?

Burndown widget showing a release Burndown



To learn more, see [Configure a Burndown or Burnup widget](#).

Burnup

The Burnup widget lets you display a trend of completed work across multiple teams and multiple sprints. You can use it to create a release burnup, a bug burnup, or a burnup on any scope of work over time. When completed work meets total scope, your project is done!

Burnup widget showing a release Burnup

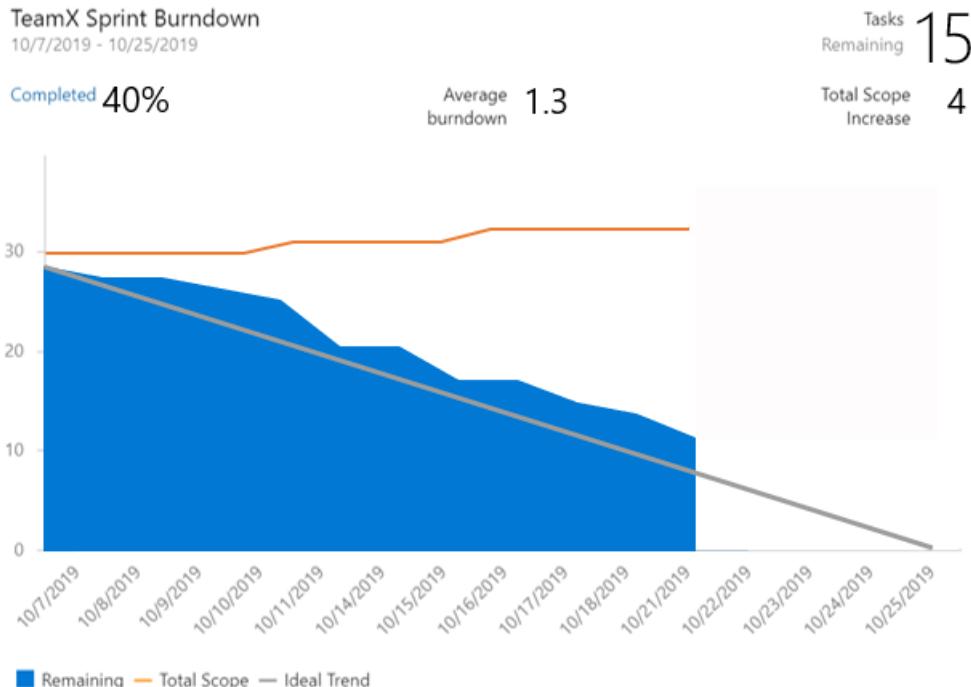


To learn more, see [Configure a Burndown or Burnup widget](#).

Sprint Burndown widget

The Analytics-based Sprint Burndown widget adds a team's burndown chart for a sprint to the dashboard. This widget supports several configuration options, including selecting a team, iteration, and time period. Teams use the burndown chart to mitigate risk and check for scope creep throughout the sprint cycle.

Sprint Burndown widget



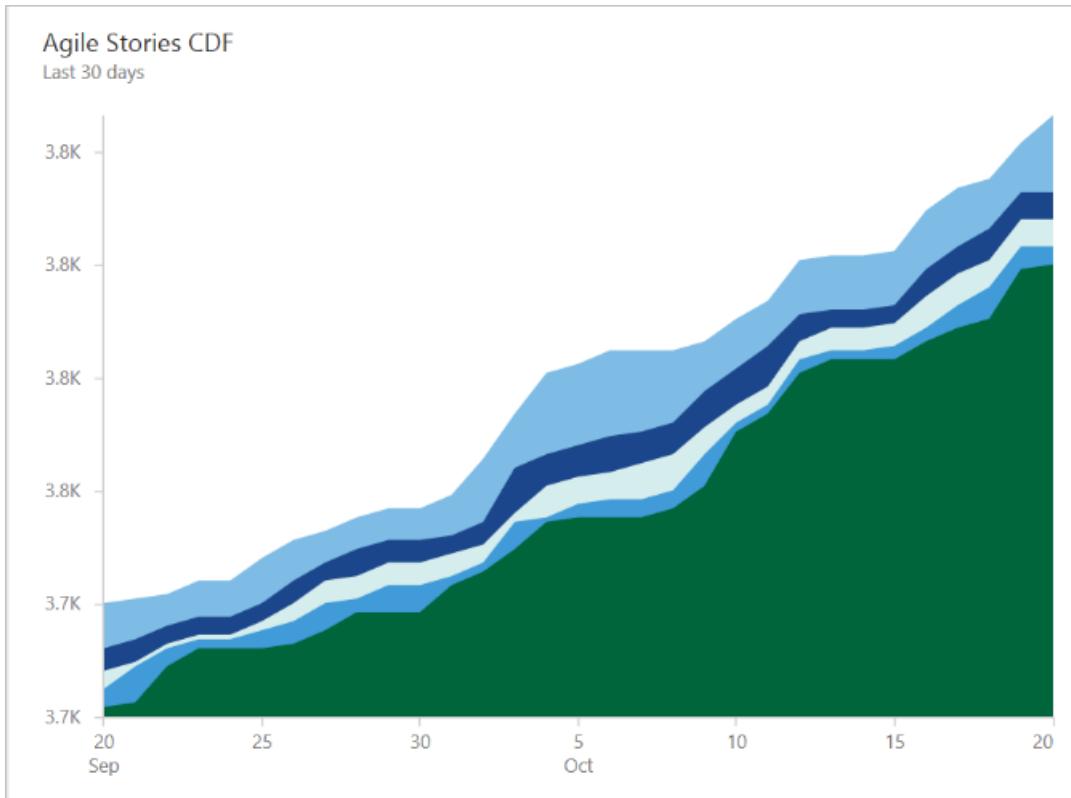
To learn more, see [Configure and monitor sprint burndown](#).

Cumulative Flow Diagram (CFD)

The CFD widget shows the count of work items (over time) for each column of a Kanban board. This allows you to see patterns in your team's development cycle over time. It will help you answer questions like:

- Is there a bottleneck in my process?
- Am I consistently delivering value to my users?

Cumulative flow diagram widget showing 30 days of data



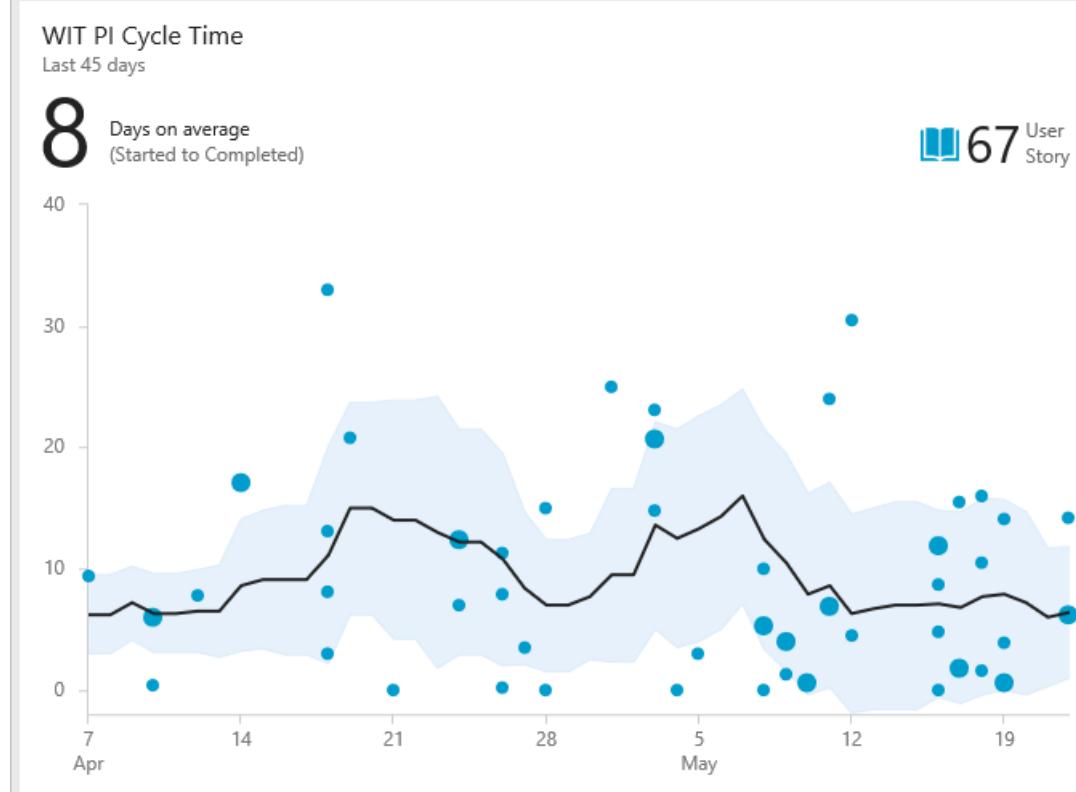
To learn more, see [Cumulative flow diagram widget](#).

Cycle Time

The Cycle time widget will help you analyze the time it takes for your team to complete work items once they begin actively working on them. A lower cycle time is typically indicative of a healthier team process. Using the Cycle time widget you will be able to answer questions like:

- On average, how long does it take my team to build a feature or fix a bug?
- Are bugs costing my team a lot of development time?

Cycle time widget showing 30 days of data



To learn more, see [Cycle time and lead time control charts](#).

Lead Time

The Lead time widget will help you analyze the time it takes to deliver work from your backlog. Lead time measures the total time elapsed from the creation of work items to their completion. Using the Lead time widget you will be able to answer questions like:

- How long does it take for work requested by a customer to be delivered?
- Did work items take longer than usual to complete?

Lead time widget showing 60 days of data

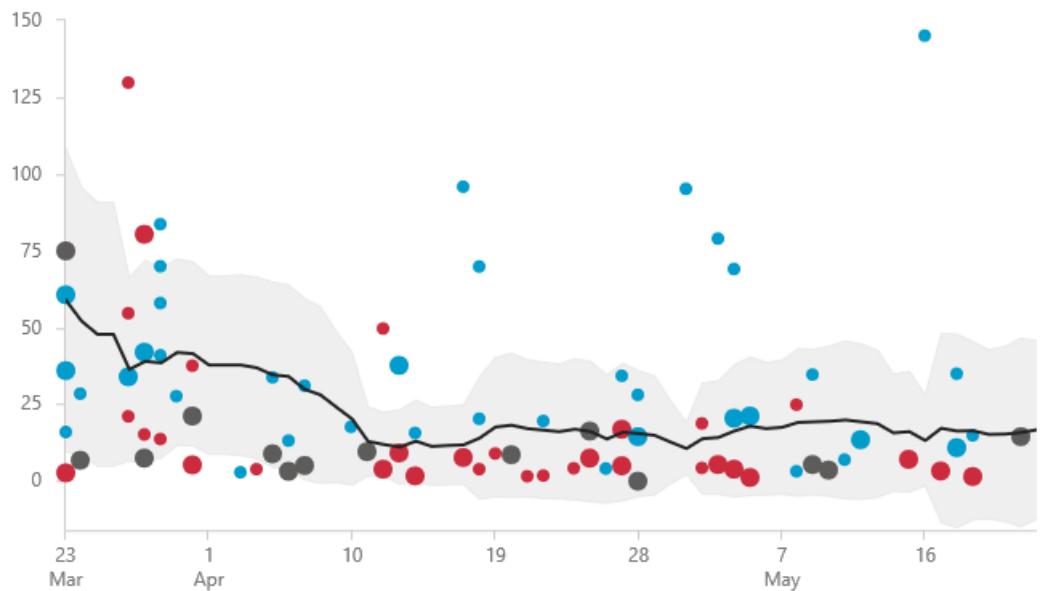
Compliance Tooling Lead Time

Last 60 days

24

Days on average
(Created to Completed)

 78 Bug  68 User Story



To learn more, see [Cycle time and lead time control charts](#).

Velocity

The Velocity widget will help you learn how much work your team can complete during a sprint. The widget shows the team's velocity by Story Points, work item count, or any custom field. You can also compare the work delivered against your plan and track work completed late. Using the Velocity widget, you will be able to answer questions like:

- On average, what is the velocity of my team?
- Is my team consistently delivering what we planned?
- How much work can we commit to deliver in upcoming sprints?

Velocity widget showing 8 sprints of data based on Story Points



To learn more, see [Configure and view Velocity widgets](#).

Test Results Trend (Advanced)

With the Test Results Trend (Advanced) widget, you can track the test quality of your pipelines over time.

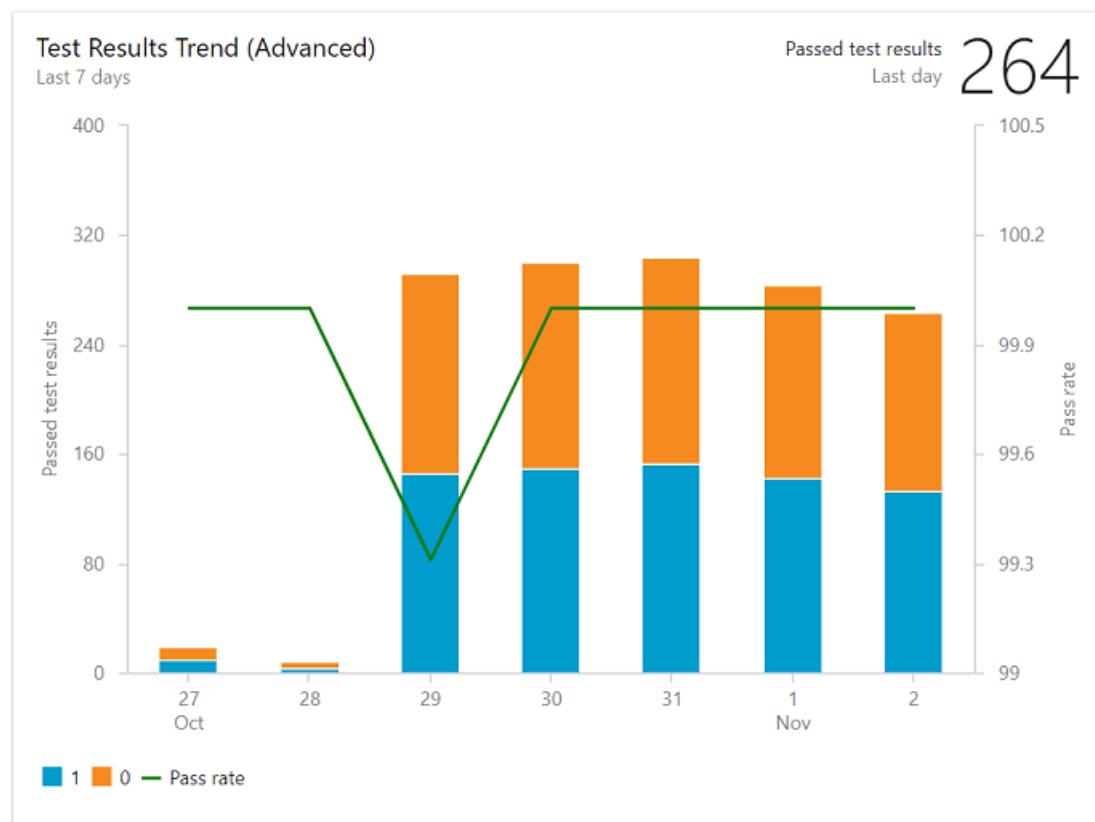
Tracking test quality and improving test collateral are essential tasks to maintaining a healthy DevOps pipeline.

The widget shows a trend of your test results for either build or release pipelines. You can track the daily count of tests, pass rates, and test duration. The highly configurable widget allows you to use it for a wide variety of scenarios.

You can find outliers in your test results and answer questions like:

- *What tests taking longer to run than usual?*
- *What micro services are affecting my pass rate?*

Test trend widget showing passed test results and pass rate for the last 7 days grouped by Priority



To learn more, see [Configure a test results widget](#).

Azure Pipelines ecosystem support

11/2/2020 • 2 minutes to read • [Edit Online](#)

Build and deploy your apps. Find guidance based on your language and platform.

Build your app



.NET Core



Anaconda



Android



ASP.NET



C/C++ with GCC



C/C++ with VC++



Containers



Go



Java



JavaScript and Node.js



PHP



Python



Ruby



UWP



Xamarin



Xcode



.NET Core



Android



ASP.NET



C/C++ with GCC



C/C++ with VC++



Containers



Go



Java



JavaScript and Node.js



PHP



Python



Ruby



UWP



Xamarin



Xcode

Deploy your app



Kubernetes



Azure Stack



Azure SQL database



Azure Web Apps



Linux VM



npm



NuGet



Virtual Machine Manager



VMware



Web App for Containers



Windows VM



Azure SQL database



Azure Web Apps



Linux VM



npm



NuGet



Virtual Machine Manager



VMware



Web App for Containers



Windows VM

Build, test, and deploy .NET Core apps

5/11/2021 • 17 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

Use a pipeline to automatically build and test your .NET Core projects. Learn how to:

- Set up your build environment with [Microsoft-hosted](#) or [self-hosted](#) agents.
- Restore dependencies, build your project, and test with the [.NET Core CLI task](#) or a [script](#).
- Use the [publish code coverage task](#) to publish code coverage results.
- Package and deliver your code with the [.NET Core CLI task](#) and the [publish build artifacts task](#).
- Publish to a [NuGet feed](#).
- Deploy your [web app to Azure](#).

NOTE

For help with .NET Framework projects, see [Build ASP.NET apps with .NET Framework](#).

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

NOTE

This guidance applies to TFS version 2017.3 and newer.

Create your first pipeline

Are you new to Azure Pipelines? If so, then we recommend you try this section before moving on to other sections.

Get the code

Fork this repo in GitHub:

Import this repo into your Git repo in Azure DevOps Server 2019:

Import this repo into your Git repo in TFS:

```
https://github.com/MicrosoftDocs/pipelines-dotnet-core
```

Sign in to Azure Pipelines

Sign in to [Azure Pipelines](#). After you sign in, your browser goes to <https://dev.azure.com/my-organization-name> and displays your Azure DevOps dashboard.

Within your selected organization, create a *project*. If you don't have any projects in your organization, you see a

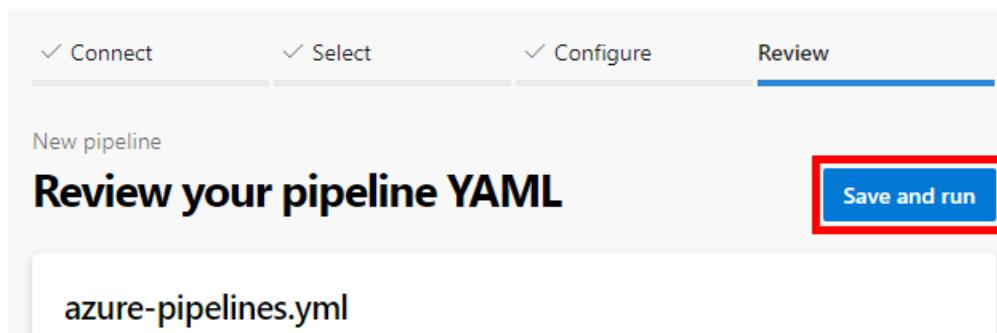
Create a project to get started screen. Otherwise, select the **Create Project** button in the upper-right corner of the dashboard.

Create the pipeline

1. Sign in to your Azure DevOps organization and navigate to your project.
2. Go to **Pipelines**, and then select **New Pipeline**.
3. Walk through the steps of the wizard by first selecting **GitHub** as the location of your source code.
4. You might be redirected to GitHub to sign in. If so, enter your GitHub credentials.
5. When the list of repositories appears, select your repository.
6. You might be redirected to GitHub to install the Azure Pipelines app. If so, select **Approve & install**.

When the **Configure** tab appears, select **ASP.NET Core**.

1. When your new pipeline appears, take a look at the YAML to see what it does. When you're ready, select **Save and run**.



2. You're prompted to commit a new `azure-pipelines.yml` file to your repository. After you're happy with the message, select **Save and run** again.

If you want to watch your pipeline in action, select the build job.

You just created and ran a pipeline that we automatically created for you, because your code appeared to be a good match for the [ASP.NET Core template](#).

You now have a working YAML pipeline (`azure-pipelines.yml`) in your repository that's ready for you to customize!

3. When you're ready to make changes to your pipeline, select it in the **Pipelines** page, and then **Edit** the `azure-pipelines.yml` file.
4. See the sections below to learn some of the more common ways to customize your pipeline.

YAML

1. Add an `azure-pipelines.yml` file in your repository. Customize this snippet for your build.

```

trigger:
- master

pool: Default

variables:
  buildConfiguration: 'Release'

# do this before all your .NET Core tasks
steps:
- task: DotNetCoreInstaller@2
  inputs:
    version: '2.2.402' # replace this value with the version that you need for your project
- script: dotnet build --configuration $(buildConfiguration)
  displayName: 'dotnet build $(buildConfiguration)'

```

2. Create a pipeline (if you don't know how, see [Create your first pipeline](#)), and for the template select **YAML**.
3. Set the **Agent pool** and **YAML file path** for your pipeline.
4. Save the pipeline and queue a build. When the **Build #nnnnnnnn.n has been queued** message appears, select the number link to see your pipeline in action.
5. When you're ready to make changes to your pipeline, [Edit it](#).
6. See the sections below to learn some of the more common ways to customize your pipeline.

Classic

1. Create a pipeline (if you don't know how, see [Create your first pipeline](#)). Select **Empty Pipeline** for the template.
 2. In the task catalog, find and add the **.NET Core** task. This task will run `dotnet build` to build the code in the sample repository.
 3. Save the pipeline and queue a build. When the **Build #nnnnnnnn.n has been queued** message appears, select the number link to see your pipeline in action.
- You now have a working pipeline that's ready for you to customize!
4. When you're ready to make changes to your pipeline, [Edit it](#).
 5. See the sections below to learn some of the more common ways to customize your pipeline.

Build environment

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 include several released versions of the .NET Core SDKs preinstalled.

Ubuntu 18.04 is set here in the YAML file.

```

pool:
  vmImage: 'ubuntu-18.04' # examples of other options: 'macOS-10.15', 'windows-2019'

```

See [Microsoft-hosted agents](#) for a complete list of images and [Pool](#) for further examples.

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

[UseDotNet@2](#) task to your YAML file.

To install the preview version of the 5.0.x SDK for building and 3.0.x for running tests that target .NET Core 3.0.x, add this snippet:

```
steps:
- task: UseDotNet@2
  inputs:
    version: '5.0.x'
    includePreviewVersions: true # Required for preview versions

- task: UseDotNet@2
  inputs:
    version: '3.0.x'
    packageType: runtime
```

Windows agents already include a .NET Core runtime. To install a newer SDK, set `performMultiLevelLookup` to `true` in this snippet:

```
steps:
- task: UseDotNet@2
  displayName: 'Install .NET Core SDK'
  inputs:
    version: 5.0.x
    performMultiLevelLookup: true
    includePreviewVersions: true # Required for preview versions
```

TIP

As an alternative, you can set up a [self-hosted agent](#) and save the cost of running the tool installer. See [Linux](#), [MacOS](#), or [Windows](#). You can also use self-hosted agents to save additional time if you have a large repository or you run incremental builds. A self-hosted agent can also help you in using the preview or private SDKs that are not officially supported by Azure DevOps or you have available on your corporate or on-premises environments only.

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 and project-specific tools that are specified in the project file 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.

This pipeline uses an artifact feed for `dotnet restore` in the [.NET Core CLI task](#).

```
trigger:
- master

pool:
  vmImage: 'windows-latest'

variables:
  buildConfiguration: 'Release'

steps:
- task: DotNetCoreCLI@2
  inputs:
    command: 'restore'
    feedsToUse: 'select'
    vstsFeed: 'my-vsts-feed' # A series of numbers and letters

- task: DotNetCoreCLI@2
  inputs:
    command: 'build'
    arguments: '--configuration $(buildConfiguration)'
    displayName: 'dotnet build $(buildConfiguration)'
```

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 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.

To restore packages from an external custom feed, use the **.NET Core** task:

```

# do this before your build tasks
steps:
- task: DotNetCoreCLI@2
  displayName: Restore
  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](#).

1. Select **Tasks** in the pipeline. Select the job that runs your build tasks. Then select **+** to add a new task to that job.
2. In the task catalog, find and add the **.NET Core** task.
3. Select the task and, for **Command**, select **restore**.
4. Specify any other options you need for this task. Then save the build.

NOTE

Make sure the custom feed is specified in your `NuGet.config` file and that credentials are specified in the NuGet service connection.

Build your project

You build your .NET Core project either by running the `dotnet build` command in your pipeline or by using the .NET Core task.

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

```

steps:
- task: DotNetCoreCLI@2
  displayName: Build
  inputs:
    command: build
    projects: '**/*.csproj'
    arguments: '--configuration $(buildConfiguration)' # Update this to match your need

```

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

```

steps:
- task: DotNetCoreCLI@2
  displayName: 'Install dotnetsay'
  inputs:
    command: custom
    custom: tool
    arguments: 'install -g dotnetsay'

```

Build

1. Select **Tasks** in the pipeline. Select the job that runs your build tasks. Then select **+** to add a new task to that job.

2. In the task catalog, find and add the **.NET Core** task.
3. Select the task and, for **Command**, select **build or publish**.
4. Specify any other options you need for this task. Then save the build.

Install a tool

To install a .NET Core global tool like `dotnetsay` in your build running on Windows, take the following steps:

1. Add the **.NET Core** task and set the following properties:

- **Command**: custom.
 - **Path to projects**: *leave empty*.
- **Custom command**: tool.
- **Arguments**: `install -g dotnetsay`.

2. Add a **Command Line** task and set the following properties:

- **Script**: `dotnetsay`.

Run your tests

If you have test projects in your repository, then use the **.NET Core** task to run unit tests by using testing frameworks like MSTest, xUnit, and NUnit. For this functionality, the test project must reference [Microsoft.NET.Test.SDK](#) version 15.8.0 or higher. Test results are automatically published to the service. These results are then made available to you in the build summary and can be used for troubleshooting failed tests and test-timing analysis.

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

```
steps:  
# ...  
# do this after other tasks such as building  
- 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:

```
steps:  
# ...  
# do this after your tests have run  
- script: dotnet test <test-project> --logger trx  
- task: PublishTestResults@2  
  condition: succeededOrFailed()  
  inputs:  
    testRunner: VSTest  
    testResultsFiles: '**/*.trx'
```

Use the **.NET Core** task with **Command** set to **test**. **Path to projects** should refer to the test projects in your solution.

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.

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

```
steps:  
# ...  
# do this after other tasks such as building  
- 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:

```
steps:  
# ...  
# do this after your tests have run  
- script: dotnet test <test-project> --logger trx --collect "Code coverage"  
- task: PublishTestResults@2  
  inputs:  
    testRunner: VSTest  
    testResultsFiles: '**/*.trx'
```

1. Add the .NET Core task to your build job and set the following properties:

- **Command:** test.
- **Path to projects:** *Should refer to the test projects in your solution.*
- **Arguments:** `--configuration $(BuildConfiguration) --collect "Code coverage"`.

2. Ensure that the **Publish test results** option remains selected.

Collect code coverage metrics with Coverlet

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 use this functionality, the coverage tool must be configured to generate results in Cobertura or JaCoCo coverage format.

To run tests and publish code coverage with Coverlet:

- Add a reference to the `coverlet.msbuild` NuGet package in your test project(s) for .NET projects below .NET 5. For .NET 5, add a reference to the `coverlet.collector` NuGet package.
- Add this snippet to your `azure-pipelines.yml` file:
 - [.NET 5](#)
 - [.NET < 5](#)

```

- task: UseDotNet@2
  inputs:
    version: '5.0.x'
    includePreviewVersions: true # Required for preview versions

- task: DotNetCoreCLI@2
  displayName: 'dotnet build'
  inputs:
    command: 'build'
    configuration: $(buildConfiguration)

- task: DotNetCoreCLI@2
  displayName: 'dotnet test'
  inputs:
    command: 'test'
    arguments: '--configuration $(buildConfiguration) --collect:"XPlat Code Coverage"
/p:CoverletOutputFormat=cobertura /p:CoverletOutput=$(Build.SourcesDirectory)/TestResults/Coverage/'
  publishTestResults: true
  projects: 'MyTestLibrary' # update with your test project directory

- task: PublishCodeCoverageResults@1
  displayName: 'Publish code coverage report'
  inputs:
    codeCoverageTool: 'Cobertura'
    summaryFileLocation: '$(Build.SourcesDirectory)/**/coverage.cobertura.xml'

```

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.

Publish artifacts to Azure Pipelines

To publish the output of your .NET **build**,

- Run `dotnet publish --output $(Build.ArtifactStagingDirectory)` on CLI or add the DotNetCoreCLI@2 task with publish command.
- Publish the artifact by using Publish artifact task.

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

```

steps:

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

# this code takes all the files in $(Build.ArtifactStagingDirectory) and uploads them as an artifact of your
# build.
- task: PublishPipelineArtifact@1
  inputs:
    targetPath: '$(Build.ArtifactStagingDirectory)'
    artifactName: 'myWebsiteName'

```

NOTE

The `dotNetCoreCLI@2` task has a `publishWebProjects` input that is set to **true** by default. This publishes *all* web projects in your repo by default. You can find more help and information in the [open source task on GitHub](#).

To copy more files to Build directory before publishing, use [Utility: copy files](#).

Publish to a NuGet feed

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

```
steps:  
# ...  
# do this near the end of your pipeline in most cases  
- script: dotnet pack /p:PackageVersion=$(version) # define version variable elsewhere in your pipeline  
- task: NuGetAuthenticate@0  
  input:  
    nuGetServiceConnections: '<Name of the NuGet service connection>'  
- 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:

```
steps:  
# ...  
# do this after you've built your app, near the end of your pipeline in most cases  
# for example, you do this before you deploy to an Azure web app on Windows  
- 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](#).

Publish artifacts to Azure Pipelines

Use the **Publish Artifacts** task to publish the output of your build to Azure Pipelines or TFS.

Publish to a NuGet feed

If you want to publish your code to a NuGet feed, take the following steps:

1. Use a .NET Core task with **Command** set to pack.
2. [Publish your package to a NuGet feed](#).

Deploy a web app

1. Use a .NET Core task with **Command** set to publish.
2. Make sure you've selected the option to create a .zip file archive.
3. To publish this archive to a web app, see [Azure Web Apps deployment](#).

Build an image and push to container registry

For your app, you can also [build an image](#) and [push it to a container registry](#).

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 data centers that Azure Pipelines run 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 `packages.config` 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.

FAQ

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 ASP.NET apps with .NET Framework

3/6/2021 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

NOTE

This article focuses on building .NET Framework projects with Azure Pipelines. For help with .NET Core projects, see [.NET Core](#).

NOTE

This guidance applies to TFS version 2017.3 and newer.

Create your first pipeline

Are you new to Azure Pipelines? If so, then we recommend you try this section before moving on to other sections.

Get the code

Fork this repo in GitHub:

Import this repo into your Git repo in Azure DevOps Server 2019:

Import this repo into your Git repo in TFS:

```
https://github.com/Microsoft/devops-project-samples.git
```

The sample repo includes several different projects, and the sample application for this article is located at:

```
https://github.com/Microsoft/devops-project-samples
```

You'll use the code in `/dotnet/aspnet/webapp/`. Your `azure-pipelines.yml` file needs to run from within the `dotnet/aspnet/webapp/Application` folder for the build to complete successfully.

The sample app is a Visual Studio solution that has two projects:

- An ASP.NET Web Application project that targets .NET Framework 4.5
- A Unit Test project

[Sign in to Azure Pipelines](#)

Sign in to [Azure Pipelines](#). After you sign in, your browser goes to <https://dev.azure.com/my-organization-name> and displays your Azure DevOps dashboard.

Within your selected organization, create a *project*. If you don't have any projects in your organization, you see a **Create a project to get started** screen. Otherwise, select the **Create Project** button in the upper-right corner of the dashboard.

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 [Create your first pipeline](#) and select the **ASP.NET** 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 `windows-2019` for Windows Server 2019 with Visual Studio 2019
- Use `vs2017-win2016` for Windows Server 2016 with Visual Studio 2017

You can also use a [self-hosted agent](#) to run your builds. This is 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.

Restore dependencies

You can use the [NuGet task](#) to install and update NuGet package dependencies. You can also download NuGet packages from Azure Artifacts, NuGet.org, or some other external or internal NuGet repository with the NuGet

task.

This code restores a solution from a project-scoped feed in the same organization.

```
# Restore from a project scoped feed in the same organization
- task: NuGetCommand@2
  inputs:
    command: 'restore'
    feedsToUse: 'select'
    vstsFeed: 'my-project/my-project-scoped-feed'
    includeNuGetOrg: false
    restoreSolution: '**/*.sln'
```

Build, test, and deploy JavaScript and Node.js apps

4/27/2021 • 23 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [Azure DevOps Server 2020](#) | [Azure DevOps Server 2019](#) | [TFS 2018](#) | [TFS 2017](#)

Use a pipeline to build and test JavaScript and Node.js apps, and then deploy or publish to targets. Learn how to:

- Set up your build environment with [Microsoft-hosted](#) or [self-hosted](#) agents.
- Use the [npm task](#) or a [script](#) to download packages for your build.
- Implement [JavaScript frameworks](#): Angular, React, or Vue.
- Run unit tests and publish them with the [publish test results task](#).
- Use the [publish code coverage task](#) to publish code coverage results.
- Publish [npm packages](#) with Azure Artifacts.
- Create a .zip file archive that is ready for publishing to a web app with the [Archive Files task](#) and [deploy to Azure](#).

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

NOTE

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

Create your first pipeline

Are you new to Azure Pipelines? If so, then we recommend you try this section to create before moving on to other sections.

- [Get the code](#)
- [See an example](#)

Fork this repo in GitHub:

<https://github.com/MicrosoftDocs/pipelines-javascript>

Sign in to Azure Pipelines

Sign in to [Azure Pipelines](#). After you sign in, your browser goes to <https://dev.azure.com/my-organization-name> and displays your Azure DevOps dashboard.

Within your selected organization, create a *project*. If you don't have any projects in your organization, you see a **Create a project to get started** screen. Otherwise, select the **Create Project** button in the upper-right corner of the dashboard.

Create the pipeline

1. The following code is a simple Node server implemented with the Express.js framework. Tests for the app are written through the Mocha framework. To get started, fork this repo in GitHub.

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

2. Sign in to your Azure DevOps organization and navigate to your project.
3. In your project, navigate to the **Pipelines** page. Then choose the action to create a new pipeline.
4. Walk through the steps of the wizard by first selecting **GitHub** as the location of your source code.
5. You might be redirected to GitHub to sign in. If so, enter your GitHub credentials.
6. When the list of repositories appears, select your Node.js sample repository.
7. Azure Pipelines will analyze the code in your repository and recommend **Node.js** template for your pipeline. Select that template.
8. Azure Pipelines will generate a YAML file for your pipeline. Select **Save and run**, then select **Commit directly to the main branch**, and then choose **Save and run** again.
9. A new run is started. Wait for the run to finish.

When you're done, you'll have a working YAML file (`azure-pipelines.yml`) in your repository that's ready for you to customize.

TIP

To make changes to the YAML file as described in this topic, select the pipeline in the **Pipelines** page, and then **Edit the `azure-pipelines.yml` file**.

YAML

1. The following code is a simple Node server implemented with the Express.js framework. Tests for the app are written through the Mocha framework. To get started, fork this repo in GitHub.

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

2. Add an `azure-pipelines.yml` file in your repository. This YAML assumes that you have Node.js with npm installed on your server.

```
trigger:  
- main  
  
pool: Default  
  
- script: |  
  npm install  
  npm run build  
displayName: 'npm install and build'
```

3. Create a pipeline (if you don't know how, see [Create your first pipeline](#)), and for the template select **YAML**.
4. Set the **Agent pool** and **YAML file path** for your pipeline.
5. Save the pipeline and queue a build. When the **Build #nnnnnnnn.n has been queued** message

appears, select the number link to see your pipeline in action.

6. When you're ready to make changes to your pipeline, [Edit it](#).
7. See the sections below to learn some of the more common ways to customize your pipeline.

Classic

1. The following code is a simple Node server implemented with the Express.js framework. Tests for the app are written through the Mocha framework. To get started, fork this repo in GitHub.

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

2. After you have the sample code in your own repository, create a pipeline by using the instructions in [Create your first pipeline](#) and select the **Empty process** template.

3. Select **Process** under the **Tasks** tab in the pipeline editor and change the properties as follows:

- **Agent queue:** `Hosted Ubuntu 1604`

4. Add the following tasks to the pipeline in the specified order:

- **npm**

- **Command:** `install`

- **npm**

- **Display name:** `npm test`

- **Command:** `custom`

- **Command and arguments:** `test`

- **Publish Test Results**

- Leave all the default values for properties

- **Archive Files**

- **Root folder or file to archive:** `$(System.DefaultWorkingDirectory)`

- **Prepend root folder name to archive paths:** Unchecked

- **Publish Build Artifacts**

- Leave all the default values for properties

5. Save the pipeline and queue a build to see it in action.

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. You can use either Windows or Linux agents to run your builds.

Update the following snippet in your `azure-pipelines.yml` file to select the appropriate image.

```
pool:  
  vmImage: 'ubuntu-latest' # examples of other options: 'macOS-10.15', 'vs2017-win2016'
```

Tools that you commonly use to build, test, and run JavaScript apps - like npm, Node, Yarn, and Gulp - are pre-installed on [Microsoft-hosted agents](#) in Azure Pipelines. 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.

You can also use a [self-hosted](#) agent.

Use a specific version of Node.js

If you need a version of Node.js and npm that is not already installed on the Microsoft-hosted agent, use the [Node tool installer task](#). 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: '12.x' # replace this value with the version that you need for your project
```

If you need a version of Node.js/npm that is not already installed on the agent:

1. In the pipeline, select **Tasks**, choose the phase that runs your build tasks, and then select **+** to add a new task to that phase.
2. In the task catalog, find and add the **Node Tool Installer** task.
3. Select the task and specify the version of the Node.js runtime that you want to install.

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 by using a strategy and the [Node tool installer task](#).

```
pool:
  vmImage: 'ubuntu-latest'
strategy:
  matrix:
    node_12_x:
      node_version: 12.x
    node_13_x:
      node_version: 13.x

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

- script: npm install
```

See [multi-configuration execution](#).

Install tools on your build agent

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.

You can use a [script](#) or the [npm task](#).

Using a script to install with package.json

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

Using the npm task to install with package.json

```
- task: Npm@1
  inputs:
    command: 'install'
```

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.

NOTE

On Microsoft-hosted Linux agents, preface the command with `sudo`, like `sudo npm install -g`.

```
- 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.

Use the [npm](#) or [command line](#) tasks in your pipeline to install tools on your build agent.

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 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.

To install npm packages by using a script in your 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>
```

Use the [npm](#) or [npm authenticate](#) task in your pipeline to download and install packages.

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 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

Use a 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
```

Use the [CLI](#) or [Bash](#) task in your pipeline to invoke [Yarn](#).

Run JavaScript compilers

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
```

Use the [npm](#) task in your pipeline if you have a compile script defined in your project's package.json to build the code. Use the [Bash](#) task to compile your code if you don't have a separate script defined in your project configuration.

Run unit tests

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 using the built-in [publish test results](#) task.

If your test framework doesn't support JUnit output, 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 cypress-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. 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`.

```
- script: npm test
```

Publish test results

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

```
- task: PublishTestResults@2
  condition: succeededOrFailed()
  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 or JaCoCo reporting output, so ensure that your code coverage tool runs with the necessary options to generate the right output. (For example, `--report cobertura`.)

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

Use the [Publish Test Results](#) and [Publish Code Coverage Results](#) tasks in your pipeline to publish test results along with code coverage results by using Istanbul.

Set the Control Options for the Publish Test Results task to run the task even if a previous task has failed, unless the deployment was canceled.

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

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 pipeline.

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

Use the [CLI](#) or [Bash](#) task in your pipeline to invoke your packaging tool, such as `webpack` or Angular's `ng build`.

JavaScript frameworks

Angular

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

NOTE

On Microsoft-hosted Linux agents, preface the command with `sudo`, like `sudo npm install -g`.

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

Add the following tasks to your pipeline:

- **npm**
 - **Command:** `custom`
 - **Command and arguments:** `install -g @angular/cli`
- **npm**
 - **Command:** `install`
- **bash**
 - **Type:** `inline`
 - **Script:** `ng build --prod`

For tests in your 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 Angular starter app:

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.

React and Vue

All the dependencies for your React and Vue apps are captured in your `package.json` file. Your `azure-pipelines.yml` file contains the standard Node.js script:

```
- script: |
  npm install
  npm run build
  displayName: 'npm install and build'
```

The build files are in a new folder, `dist` (for Vue) or `build` (for React). This snippet builds an artifact, `www`, that is ready for release. It uses the [Node Installer](#), [Copy Files](#), and [Publish Build Artifacts](#) tasks.

```

trigger:
- main

pool:
  vmImage: 'ubuntu-latest'

steps:
- task: NodeTool@0
  inputs:
    versionSpec: '10.x'
  displayName: 'Install Node.js'

- script: |
  npm install
  npm run build
  displayName: 'npm install and build'

- task: CopyFiles@2
  inputs:
    Contents: 'build/**' # Pull the build directory (React)
    TargetFolder: '$(Build.ArtifactStagingDirectory)'

- task: PublishBuildArtifacts@1
  inputs:
    PathToPublish: $(Build.ArtifactStagingDirectory) # dist or build files
    ArtifactName: 'www' # output artifact named www

```

To release, point your release task to the `dist` or `build` artifact and use the [Azure Web App Deploy task](#).

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.

```

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

```

Add the following tasks to your pipeline:

- **npm**
 - **Command:** `custom`
 - **Command and arguments:** `install -g webpack webpack-cli --save-dev`
- **bash**
 - **Type:** `inline`
 - **Script:** `npx webpack --config webpack.config.js`

Build task runners

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

Gulp

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'
```

The simplest way to create a pipeline if your app uses Gulp is to use the [Node.js with gulp](#) build template when creating the pipeline. This will automatically add various tasks to invoke Gulp commands and to publish artifacts. In the task, select **Enable Code Coverage** to enable code coverage by using Istanbul.

Grunt

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
```

The simplest way to create a pipeline if your app uses Grunt is to use the [Node.js with Grunt](#) build template when creating the pipeline. This will automatically add various tasks to invoke Gulp commands and to publish artifacts. In the task, select the **Publish to TFS/Team Services** option to publish test results, and select **Enable Code Coverage** to enable code coverage by using Istanbul.

Package and deliver your code

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

Publish files to Azure Pipelines

To simply upload the entire working directory of files, use the [Publish Build Artifacts](#) task and 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 with the [Copy Files](#) task, and then use the [Publish Build Artifacts 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](#) and [How can I version my npm packages as part of the build process?](#)

Deploy a web app

To create a .zip file archive that is ready for publishing to a web app, use the [Archive Files](#) task:

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

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

Publish artifacts to Azure Pipelines

Use the [Publish Build Artifacts task](#) to publish files from your build to Azure Pipelines or TFS.

Publish to an npm registry

To create and publish an npm package, use the [npm task](#). 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, use the [Archive Files task](#). To publish this archive to a web app, see [Azure Web App deployment](#).

Build and push image to container registry

Once your source code is building successfully and your unit tests are in place and successful, you can also [build an image](#) and [push it to a container registry](#).

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 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.
- If you're using `nvm` to manage different versions of Node.js, consider switching to the [Node Tool Installer](#) task instead. (`nvm` is installed for historical reasons on the macOS image.) `nvm` manages multiple Node.js versions by adding shell aliases and altering `PATH`, which interacts poorly with the way [Azure Pipelines runs each task in a new process](#).

The [Node Tool Installer](#) task handles this model correctly. However, if your work requires the use of `nvm`, you can add the following script to the beginning of each pipeline:

```
steps:
- bash: |
  NODE_VERSION=12 # or whatever your preferred version is
  npm config delete prefix # avoid a warning
  . ${NVM_DIR}/nvm.sh
  nvm use ${NODE_VERSION}
  nvm alias default ${NODE_VERSION}
  VERSION_PATH="$(nvm_version_path ${NODE_VERSION})"
  echo "##vso[task.prependPath]$VERSION_PATH"
```

Then, `node` and other command-line tools will work for the rest of the pipeline job. In each step where you need to use the `nvm` command, you'll need to start the script with:

```
- bash: |
  . ${NVM_DIR}/nvm.sh
  nvm <command>
```

FAQ

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

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

Where can I learn more about tasks?

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

How do I fix a pipeline failure with the message 'FATAL ERROR: CALL_AND_RETRY_LAST Allocation failed - JavaScript heap out of memory'

This happens when the Node.js package has exceeded the memory usage limit. To resolve the issue, add a variable like `NODE_OPTIONS` and assign it a value of `--max_old_space_size=16384`.

How can I version my npm packages as part of the build process?

One option is to use a combination of version control and [npm version](#). At the end of a pipeline run, you can update your repo with the new version. In this YAML, there is a GitHub repo and the package gets deployed to npmjs. Note that your build will fail if there is a mismatch between your package version on npmjs and your `package.json` file.

```
variables:
  MAP_NPMTOKEN: $(NPMTOKEN) # Mapping secret var

trigger:
- none

pool:
  vmImage: 'ubuntu-latest'

steps: # Checking out connected repo
- checkout: self
  persistCredentials: true
  clean: true

- task: npmAuthenticate@0
  inputs:
    workingFile: .npmrc
    customEndpoint: 'my-npm-connection'

- task: NodeTool@0
  inputs:
    versionSpec: '12.x'
    displayName: 'Install Node.js'

- script: |
  npm install
  displayName: 'npm install'

- script: |
  npm pack
  displayName: 'Package for release'

- bash: | # Grab the package version
  v=`node -p "const p = require('./package.json'); p.version;"`  

  echo "#!/usr/bin/bash -e  

  set -o pipefail  

  export NPM_TOKEN=$MAP_NPMTOKEN  

  curl -X POST https://api.npmjs.com/v1/_apis/packaging/nuget/v1/nuget/$(echo $v | sed 's/\./-/g')/publish  

  echo 'Published package '$v'
```

```
echo ##vso[task.setVariable variable=packageversion]${{version}}
```

- task: CopyFiles@2
 - inputs:
 - contents: '*.tgz'
 - targetFolder: \$(Build.ArtifactStagingDirectory)/npm
 - displayName: 'Copy archives to artifacts staging directory'
- task: CopyFiles@2
 - inputs:
 - sourceFolder: '\$(Build.SourcesDirectory)'
 - contents: 'package.json'
 - targetFolder: \$(Build.ArtifactStagingDirectory)/npm
 - displayName: 'Copy package.json'
- task: PublishBuildArtifacts@1
 - inputs:
 - PathToPublish: \$(Build.ArtifactStagingDirectory)/npm'
 - artifactName: npm
 - displayName: 'Publish npm artifact'
- script: | # Config can be set in .npmrc
 - npm config set //registry.npmjs.org/:_authToken=\$(MAP_NPMTOKEN)
 - npm config set scope "@myscope"
 - # npm config list
 - # npm --version
 - npm version patch --force
 - npm publish --access public
- task: CmdLine@2 # Push changes to GitHub (substitute your repo)
 - inputs:
 - script: |
 - git config --global user.email "username@contoso.com"
 - git config --global user.name "Azure Pipeline"
 - git add package.json
 - git commit -a -m "Test Commit from Azure DevOps"
 - git push -u origin HEAD:main

Build Python apps

6/1/2021 • 6 minutes to read • [Edit Online](#)

Azure Pipelines

You can use Azure Pipelines to build, test, and deploy Python apps and scripts as part of your CI/CD system. This article focuses on creating a simple pipeline.

If you want an end-to-end walkthrough, see [Use CI/CD to deploy a Python web app to Azure App Service on Linux](#).

To create and activate an Anaconda environment and install Anaconda packages with `conda`, see [Run pipelines with Anaconda environments](#).

Create your first pipeline

Are you new to Azure Pipelines? If so, then we recommend you try this section before moving on to other sections.

Get the code

Import this repo into your Git repo in Azure DevOps Server 2019:

Import this repo into your Git repo:

```
https://github.com/Microsoft/python-sample-vscode-flask-tutorial
```

Sign in to Azure Pipelines

Sign in to [Azure Pipelines](#). After you sign in, your browser goes to <https://dev.azure.com/my-organization-name> and displays your Azure DevOps dashboard.

Within your selected organization, create a *project*. If you don't have any projects in your organization, you see a **Create a project to get started** screen. Otherwise, select the **Create Project** button in the upper-right corner of the dashboard.

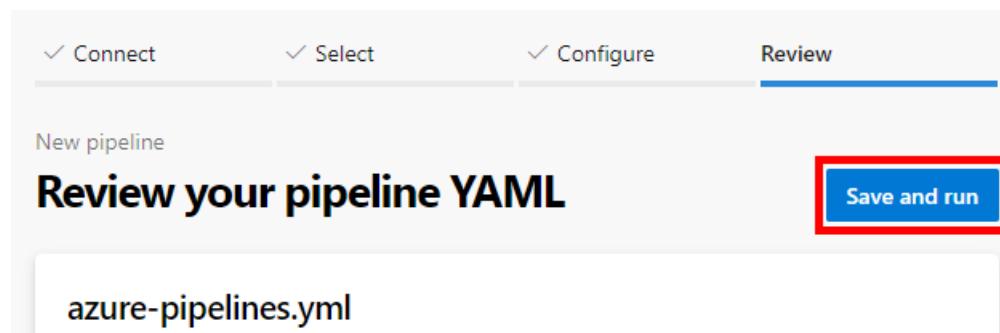
Create the pipeline

1. Sign in to your Azure DevOps organization and navigate to your project.
2. Go to **Pipelines**, and then select **New Pipeline**.
3. Walk through the steps of the wizard by first selecting **GitHub** as the location of your source code.
4. You might be redirected to GitHub to sign in. If so, enter your GitHub credentials.
5. When the list of repositories appears, select your repository.
6. You might be redirected to GitHub to install the Azure Pipelines app. If so, select **Approve & install**.

When the **Configure** tab appears, select **Python package**. This will create a Python package to test on multiple Python versions.

7. When your new pipeline appears, take a look at the YAML to see what it does. When you're ready, select

Save and run.



8. You're prompted to commit a new `azure-pipelines.yml` file to your repository. After you're happy with the message, select **Save and run** again.

If you want to watch your pipeline in action, select the build job.

You just created and ran a pipeline that we automatically created for you, because your code appeared to be a good match for the [Python package](#) template.

You now have a working YAML pipeline (`azure-pipelines.yml`) in your repository that's ready for you to customize!

9. When you're ready to make changes to your pipeline, select it in the **Pipelines** page, and then **Edit** the `azure-pipelines.yml` file.

See the sections below to learn some of the more common ways to customize your pipeline.

YAML

1. Add an `azure-pipelines.yml` file in your repository. Customize this snippet for your build.

```
trigger:  
- master  
  
pool: Default  
  
steps:  
- script: python -m pip install --upgrade pip  
  displayName: 'Install dependencies'  
  
- script: pip install -r requirements.txt  
  displayName: 'Install requirements'
```

2. Create a pipeline (if you don't know how, see [Create your first pipeline](#)), and for the template select **YAML**.
3. Set the **Agent pool** and **YAML file path** for your pipeline.
4. Save the pipeline and queue a build. When the **Build #nnnnnnnn.n has been queued** message appears, select the number link to see your pipeline in action.
5. When you're ready to make changes to your pipeline, **Edit** it.
6. See the sections below to learn some of the more common ways to customize your pipeline.

Build environment

You don't have to set up anything for Azure Pipelines to build Python projects. Python is preinstalled on [Microsoft-hosted build agents](#) for Linux, macOS, or Windows. To see which Python versions are preinstalled, see

[Use a Microsoft-hosted agent](#).

Use a specific Python version

To use a specific version of Python in your pipeline, add the [Use Python Version task](#) to `azure-pipelines.yml`. This snippet sets the pipeline to use Python 3.6:

```
steps:  
- task: UsePythonVersion@0  
  inputs:  
    versionSpec: '3.6'
```

Use multiple Python versions

To run a pipeline with multiple Python versions, for example to test a package against those versions, define a `job` with a `matrix` of Python versions. Then set the `UsePythonVersion` task to reference the `matrix` variable.

```
jobs:  
- job: 'Test'  
  pool:  
    vmImage: 'ubuntu-16.04' # other options: 'macOS-10.14', 'vs2017-win2016'  
  strategy:  
    matrix:  
      Python27:  
        python.version: '2.7'  
      Python35:  
        python.version: '3.5'  
      Python36:  
        python.version: '3.6'  
  
  steps:  
- task: UsePythonVersion@0  
  inputs:  
    versionSpec: '$(python.version)'
```

You can add tasks to run using each Python version in the matrix.

Run Python scripts

To run Python scripts in your repository, use a `script` element and specify a filename. For example:

```
- script: python src/example.py
```

You can also run inline Python scripts with the [Python Script task](#):

```
- task: PythonScript@0  
  inputs:  
    scriptSource: 'inline'  
    script: |  
      print('Hello world 1')  
      print('Hello world 2')
```

To parameterize script execution, use the `PythonScript` task with `arguments` values to pass arguments into the executing process. You can use `sys.argv` or the more sophisticated `argparse` library to parse the arguments.

```
- task: PythonScript@0
  inputs:
    scriptSource: inline
    script: |
      import sys
      print ('Executing script file is:', str(sys.argv[0]))
      print ('The arguments are:', str(sys.argv))
      import argparse
      parser = argparse.ArgumentParser()
      parser.add_argument("--world", help="Provide the name of the world to greet.")
      args = parser.parse_args()
      print ('Hello ', args.world)
  arguments: --world Venus
```

Install dependencies

You can use scripts to install specific PyPI packages with `pip`. For example, this YAML installs or upgrades `pip` and the `setuptools` and `wheel` packages.

```
- script: python -m pip install --upgrade pip setuptools wheel
  displayName: 'Install tools'
```

Install requirements

After you update `pip` and friends, a typical next step is to install dependencies from `requirements.txt`.

```
- script: pip install -r requirements.txt
  displayName: 'Install requirements'
```

Run tests

You can use scripts to install and run various tests in your pipeline.

Run lint tests with flake8

To install or upgrade `flake8` and use it to run lint tests, use this YAML:

```
- script: |
  python -m pip install flake8
  flake8 .
  displayName: 'Run lint tests'
```

Test with pytest and collect coverage metrics with pytest-cov

Use this 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 pytest-azurepipelines
  pip install pytest-cov
  pytest --doctest-modules --junitxml=junit/test-results.xml --cov=. --cov-report=xml
  displayName: 'pytest'
```

Run tests with Tox

Azure Pipelines can run parallel Tox test jobs to split up the work. On a development computer, you have to run your test environments in series. This sample uses `tox -e py` to run whichever version of Python is active for the current job.

```

- job:

  pool:
    vmImage: 'ubuntu-16.04'
  strategy:
    matrix:
      Python27:
        python.version: '2.7'
      Python35:
        python.version: '3.5'
      Python36:
        python.version: '3.6'
      Python37:
        python.version: '3.7'

  steps:
    - task: UsePythonVersion@0
      displayName: 'Use Python $(python.version)'
      inputs:
        versionSpec: '$(python.version)'

    - script: pip install tox
      displayName: 'Install Tox'

    - script: tox -e py
      displayName: 'Run Tox'

```

Publish test results

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

```

- task: PublishTestResults@2
  condition: succeededOrFailed()
  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. You can see coverage metrics in the build summary, and download HTML reports for further analysis.

```

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

```

Package and deliver code

To authenticate with `twine`, use the [Twine Authenticate task](#) to store authentication credentials in the `PYPIRC_PATH` environment variable.

```

- task: TwineAuthenticate@0
  inputs:
    artifactFeed: '<Azure Artifacts feed name>'
    pythonUploadServiceConnection: '<twine service connection from external organization>'

```

Then, add a custom `script` that uses `twine` to publish your packages.

```
- script: |
  twine upload -r "<feed or service connection name>" --config-file $(PYPIRC_PATH) <package path/files>
```

You can also use Azure Pipelines to [build an image](#) for your Python app and [push it to a container registry](#).

Related extensions

- [PyLint Checker](#) (Darren Fuller)
- [Python Test](#) (Darren Fuller)
- [Azure DevOps plugin for PyCharm \(IntelliJ\)](#) (Microsoft)
- [Python in Visual Studio Code](#) (Microsoft)

Use CI/CD to deploy a Python web app to Azure App Service on Linux

6/9/2021 • 17 minutes to read • [Edit Online](#)

Azure Pipelines

In this article, you use Azure Pipelines continuous integration and continuous delivery (CI/CD) to deploy a Python web app to Azure App Service on Linux. You begin by running app code from a GitHub repository locally. You then provision a target App Service through the Azure portal. Finally, you create an Azure Pipelines CI/CD pipeline that automatically builds the code and deploys it to the App Service whenever there's a commit to the repository.

Create a repository for your app code

If you already have a Python web app to use, make sure it's committed to a GitHub repository.

NOTE

If your app uses Django and a SQLite database, it won't work for this walkthrough. For more information, see [considerations for Django](#) later in this article. If your Django app uses a separate database, you can use it with this walkthrough.

If you need an app to work with, you can fork and clone the repository at <https://github.com/Microsoft/python-sample-vscode-flask-tutorial>. The code is from the tutorial [Flask in Visual Studio Code](#).

To test the example app locally, from the folder containing the code, run the following appropriate commands for your operating system:

```
# Mac/Linux
sudo apt-get install python3-venv # If needed
python3 -m venv .env
source .env/bin/activate
pip install -r requirements.txt
export FLASK_APP=hello_app.webapp
python3 -m flask run
```

```
# Windows
py -3 -m venv .env
.env\scripts\activate
pip install -r requirements.txt
$env:FLASK_APP = "hello_app.webapp"
python -m flask run
```

Open a browser and navigate to <http://localhost:5000> to view the app. When you're finished, close the browser, and stop the Flask server with **Ctrl+C**.

Provision the target Azure App Service

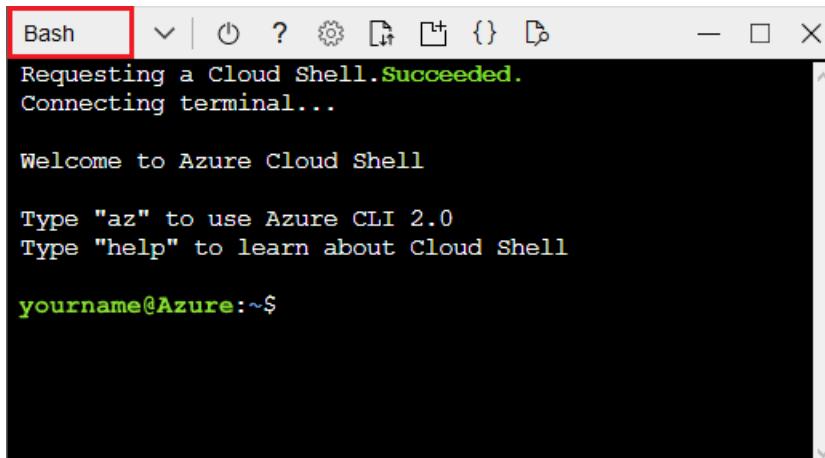
The quickest way to create an App Service instance is to use the Azure command-line interface (CLI) through the interactive Azure Cloud Shell. In the following steps, you use [az webapp up](#) to both provision the App Service

and perform the first deployment of your app.

1. Sign in to the Azure portal at <https://portal.azure.com>.
2. Open the Azure CLI by selecting the Cloud Shell button on the portal's toolbar:



3. The Cloud Shell appears along the bottom of the browser. Select **Bash** from the dropdown:



4. In the Cloud Shell, clone your repository using `git clone`. For the example app, use:

```
git clone https://github.com/<your-alias>/python-sample-vscode-flask-tutorial
```

Replace `<your-alias>` with the name of the GitHub account you used to fork the repository.

TIP

To paste into the Cloud Shell, use **Ctrl+Shift+V**, or right-click and select **Paste** from the context menu.

NOTE

The Cloud Shell is backed by an Azure Storage account in a resource group called `cloud-shell-storage-<your-region>`. That storage account contains an image of the Cloud Shell's file system, which stores the cloned repository. There is a small cost for this storage. You can delete the storage account at the end of this article, along with other resources you create.

5. In the Cloud Shell, change directories into the repository folder that has your Python app, so the `az webapp up` command will recognize the app as Python.

```
cd python-sample-vscode-flask-tutorial
```

6. In the Cloud Shell, use `az webapp up` to create an App Service and initially deploy your app.

```
az webapp up -n <your-appservice>
```

Change `<your-appservice>` to a name for your app service that's unique across Azure. Typically, you use a personal or company name along with an app identifier, such as `<your-name>-flaskpipelines`. The app URL becomes `<your-appservice>.azurewebsites.net`.

When the command completes, it shows JSON output in the Cloud Shell.

TIP

If you encounter a "Permission denied" error with a `.zip` file, you may have tried to run the command from a folder that doesn't contain a Python app. The `az webapp up` command then tries to create a Windows app service plan, and fails.

7. If your app uses a custom startup command, set the `az webapp config` property. For example, the `python-sample-vscode-flask-tutorial` app contains a file named `startup.txt` that contains its specific startup command, so you set the `az webapp config` property to `startup.txt`.

- a. From the first line of output from the previous `az webapp up` command, copy the name of your resource group, which is similar to `<your-name>_rg_Linux_<your-region>`.
- b. Enter the following command, using your resource group name, your app service name (`<your-appservice>`), and your startup file or command (`startup.txt`).

```
az webapp config set -g <your-resource-group> -n <your-appservice> --startup-file <your-startup-file-or-command>
```

Again, when the command completes, it shows JSON output in the Cloud Shell.

8. To see the running app, open a browser and go to `http://<your-appservice>.azurewebsites.net`. If you see a generic page, wait a few seconds for the App Service to start, and refresh the page.

NOTE

For a detailed description of the specific tasks performed by the `az webapp up` command, see [Provision an App Service with single commands](#) at the end of this article.

Create an Azure DevOps project and connect to Azure

To deploy to Azure App Service from Azure Pipelines, you need to establish a *service connection* between the two services.

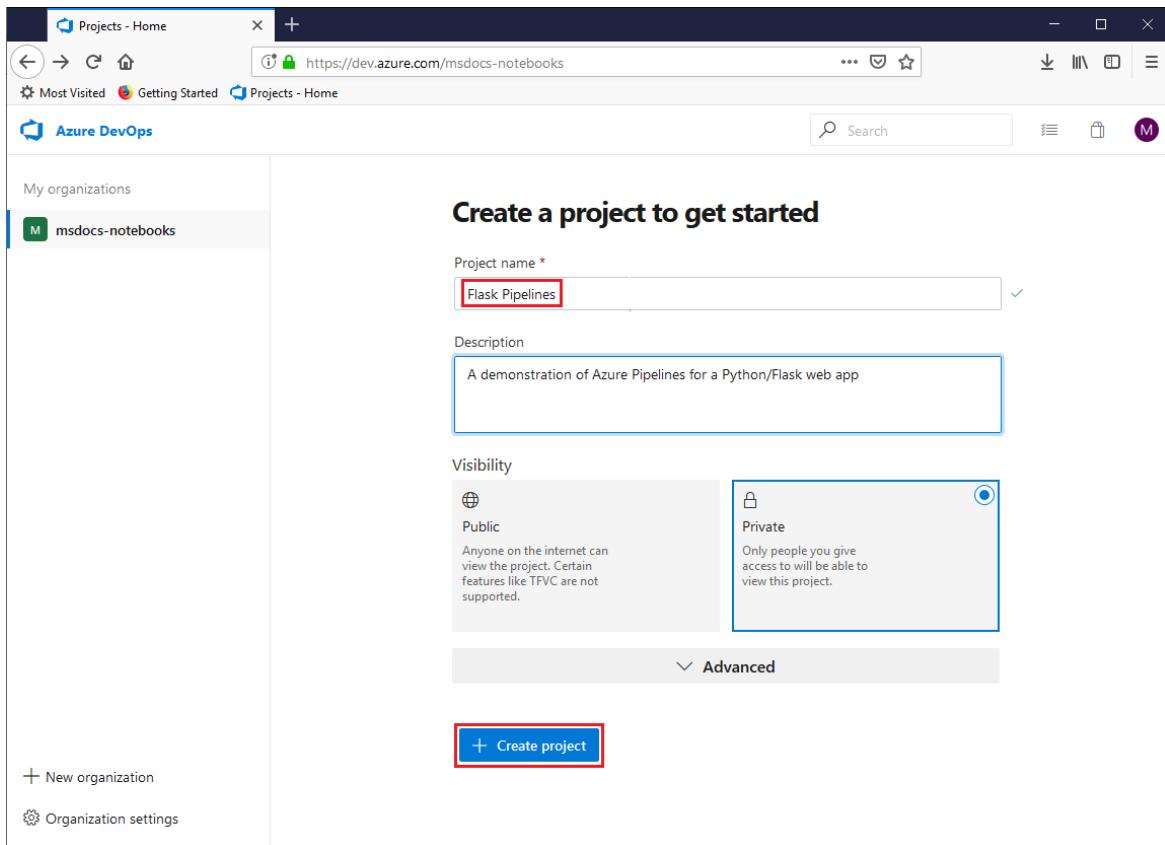
1. In a browser, go to dev.azure.com. If you don't yet have an account on Azure DevOps, select **Start free** and get a free account. If you have an account already, select **Sign in to Azure DevOps**.

IMPORTANT

To simplify the service connection, use the same email address for Azure DevOps as you use for Azure.

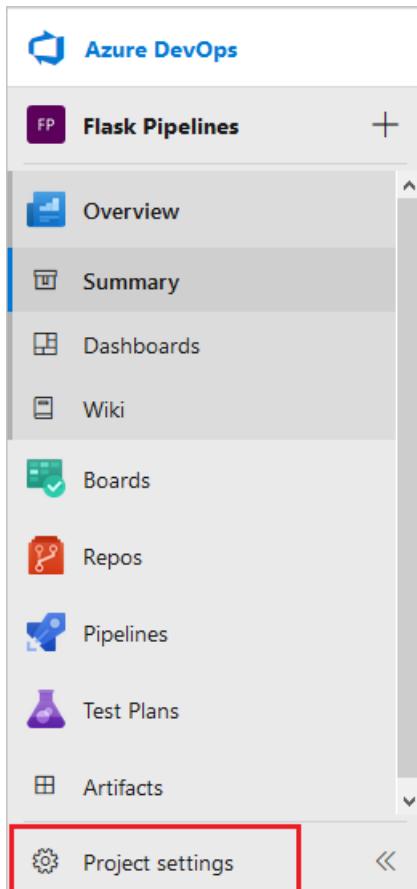
2. Once you sign in, the browser displays your Azure DevOps dashboard, at the URL `https://dev.azure.com/<your-organization-name>`. An Azure DevOps account can belong to one or more *organizations*, which are listed on the left side of the Azure DevOps dashboard. If more than one organization is listed, select the one you want to use for this walkthrough. By default, Azure DevOps creates a new organization using the email alias you used to sign in.

A project is a grouping for boards, repositories, pipelines, and other aspects of Azure DevOps. If your organization doesn't have any projects, enter the project name *Flask Pipelines* under **Create a project to get started**, and then select **Create project**.



If your organization already has projects, select **New project** on the organization page. In the **Create new project** dialog box, enter the project name *Flask Pipelines*, and select **Create**.

3. From the new project page, select **Project settings** from the left navigation.



4. On the **Project Settings** page, select **Pipelines > Service connections**, then select **New service connection**, and then select **Azure Resource Manager** from the dropdown.

Project Settings > Service connections

General

Service connections XAML build services

+ New service connection ▾

Azure Classic

Azure Repos/Team Foundation Ser...

Azure Resource Manager

Azure Service Bus

Bitbucket Cloud

Chef

Docker Host

Docker Registry

Generic

GitHub

Boards

Project configuration

Team configuration

GitHub connections

Pipelines

Service connections

Agent pools

Retention and parallel jobs

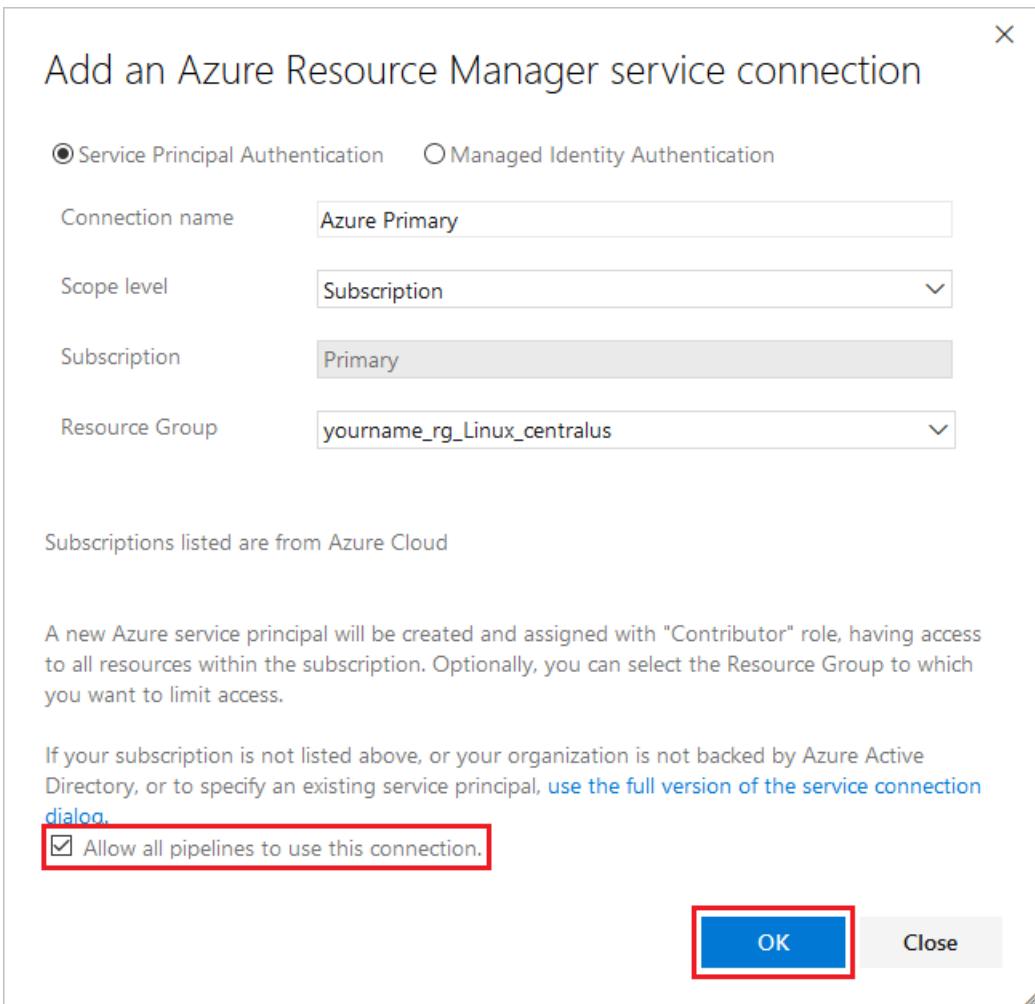
Release retention

No service connections were found.

The screenshot shows the 'Service connections' page in the Azure DevOps Project Settings. On the left, there's a sidebar with sections like General, Boards, and Pipelines. Under Pipelines, 'Service connections' is highlighted with a red box. The main area shows a list of service connection types: Azure Classic, Azure Repos/Team Foundation Ser..., Azure Resource Manager (which is also highlighted with a red box), Azure Service Bus, Bitbucket Cloud, Chef, Docker Host, Docker Registry, Generic, and GitHub. A search bar is at the top right of this list. To the right of the list, it says 'No service connections were found.'

5. In the Add an Azure Resource Manager service connection dialog box:

- Give the connection a name. Make note of the name to use later in the pipeline.
- For **Scope level**, select **Subscription**.
- Select the subscription for your App Service from the **Subscription** drop-down list.
- Under **Resource Group**, select your resource group from the dropdown.
- Make sure the option **Allow all pipelines to use this connection** is selected, and then select **OK**.



The new connection appears in the **Service connections** list, and is ready for Azure Pipelines to use from the project.

NOTE

If you need to use an Azure subscription from a different email account, follow the instructions on [Create an Azure Resource Manager service connection with an existing service principal](#).

Create a Python-specific pipeline to deploy to App Service

1. From your project page left navigation, select **Pipelines**.

The screenshot shows the Azure DevOps interface for the 'Flask Pipelines' project. The left sidebar contains links for Overview, Summary, Dashboards, Wiki, Boards, Repos, Pipelines (which is highlighted with a red box), Test Plans, and Artifacts. The main content area features a cartoon illustration of a person working at a desk with a laptop, a dog, and clouds. The title 'Flask Pipelines' is displayed, along with a 'Private' button, an 'Invite' button, and a star icon. Below the illustration, the text 'Welcome to the project!' is shown, followed by a question 'What service would you like to start with?' and four buttons: Boards, Repos, Pipelines (which is highlighted with a blue box), and Test Plans.

2. Select **Create Pipeline**:



Create your first Pipeline

Automate your build and release processes using our wizard, and go from code to cloud-hosted within minutes.

[Create Pipeline](#)

3. On the **Where is your code** screen, select **GitHub**. You may be prompted to sign into GitHub.

Connect Select Configure Review

New pipeline

Where is your code?

 Azure Repos Git YAML
Free private Git repositories, pull requests, and code search

 Bitbucket Cloud YAML
Hosted by Atlassian

 GitHub YAML
Home to the world's largest community of developers

 GitHub Enterprise Server YAML
The self-hosted version of GitHub Enterprise

 Other Git
Any Internet-facing Git repository

 Subversion
Centralized version control by Apache

4. On the **Select a repository** screen, select the repository that contains your app, such as your fork of the example app.

✓ Connect Select Configure Review

New pipeline

Select a repository

Filter by keywords My repositories

Repository	Last updated
Microsoft/vscode-docs	2h ago
Microsoft/vscode-website	private Yesterday
Mycode/python-sample-vscode-flask-tutorial	fork Yesterday

5. You may be prompted to enter your GitHub password again as a confirmation, and then GitHub prompts you to install the **Azure Pipelines** extension:

Search or jump to... Pull requests Issues Marketplace Explore

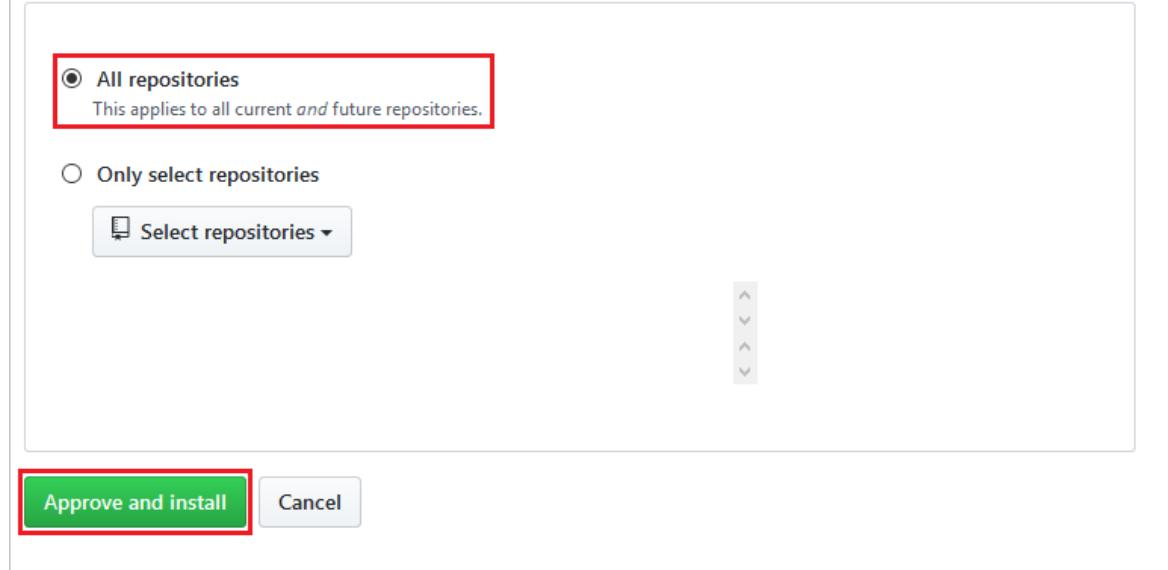
Personal settings Profile Account Emails Notifications

 Azure Pipelines
Installed 7 days ago Developed by AzurePipelines https://azure.microsoft.com/services/devops/pipelines/

Continuously build, test, and deploy to any platform and cloud
Azure Pipelines offers cloud-hosted pipelines for Linux, macOS, and Windows with 10 free parallel jobs and unlimited minutes for open source projects.

On this screen, scroll down to the **Repository access** section, choose whether to install the extension on all repositories or only selected ones, and then select **Approve and install**:

Repository access



6. On the **Configure your pipeline** screen, select **Python to Linux Web App on Azure**.

Your new pipeline appears. When prompted, select the Azure subscription in which you created your Web App.

- Select the Web App
- Select Validate and configure

Azure Pipelines creates an **azure-pipelines.yml** file that defines your CI/CD pipeline as a series of *stages*, *Jobs*, and *steps*, where each step contains the details for different *tasks* and *scripts*. Take a look at the pipeline to see what it does. Make sure all the default inputs are appropriate for your code.

YAML pipeline explained

The YAML file contains the following key elements:

- The `trigger` at the top indicates the commits that trigger the pipeline, such as commits to the `main` branch.
- The `variables` that parameterize the YAML template

TIP

To avoid hard-coding specific variable values in your YAML file, you can define variables in the pipeline's web interface instead. For more information, see [Variables - Secrets](#).

- The `stages`
 - Build `stage`, which builds your project, and a Deploy stage, which deploys it to Azure as a Linux web app.
 - Deploy `stage` that also creates an Environment with default name same as the Web App. You can choose to modify the environment name.
- Each stage has a `pool` element that specifies one or more virtual machines (VMs) in which the pipeline runs the `steps`. By default, the `pool` element contains only a single entry for an Ubuntu VM. You can use a pool to run tests in multiple environments as part of the build, such as using different Python versions for creating a package.
- The `steps` element can contain children like `task`, which runs a specific task as defined in the Azure Pipelines [task reference](#), and `script`, which runs an arbitrary set of commands.

- The first task under Build stage is [UsePythonVersion](#), which specifies the version of Python to use on the build agent. The `@<n>` suffix indicates the version of the task. The `@0` indicates preview version. Then we have script-based task that creates a virtual environment and installs dependencies from file (requirements.txt).

```

steps:
- task: UsePythonVersion@0
  inputs:
    versionSpec: '$(pythonVersion)'
    displayName: 'Use Python $(pythonVersion)'
- script: |
  python -m venv antenv
  source antenv/bin/activate
  python -m pip install --upgrade pip
  pip install setup
  pip install -r requirements.txt
  workingDirectory: $(projectRoot)
  displayName: "Install requirements"

```

- Next step creates the `.zip` file that the steps under deploy stage of the pipeline deploys. To create the `.zip` file, add an [ArchiveFiles](#) task to the end of the YAML file:

```

- task: ArchiveFiles@2
  inputs:
    rootFolderOrFile: '$(Build.SourcesDirectory)'
    includeRootFolder: false
    archiveType: 'zip'
    archiveFile: '$(Build.ArtifactStagingDirectory)/Application$(Build.BuildId).zip'
    replaceExistingArchive: true
    verbose: # (no value); this input is optional
- publish: $(Build.ArtifactStagingDirectory)/Application$(Build.BuildId).zip
  displayName: 'Upload package'
  artifact: drop

```

You use `$()` in a parameter value to reference variables. The built-in `Build.SourcesDirectory` variable contains the location on the build agent where the pipeline cloned the app code. The `archiveFile` parameter indicates where to place the `.zip` file. In this case, the `archiveFile` parameter uses the built-in variable `Build.ArtifactsStagingDirectory`.

IMPORTANT

When deploying to Azure App Service, be sure to use `includeRootFolder: false`. Otherwise, the contents of the `.zip` file are put in a folder named `s`, for "sources," which is replicated on the App Service. The App Service on Linux container then can't find the app code.

- Then we have the task to upload the artifacts.
- In the Deploy stage, we use the `deployment` keyword to define a [deployment job](#) targeting an [environment](#). By using the template, an environment with same name as the Web app is automatically created if it doesn't already exist. Alternatively you can pre-create the environment and provide the `environmentName`
- Within the deployment job, first task is [UsePythonVersion](#), which specifies the version of Python to use on the build agent.
- We then use the [AzureWebApp](#) task to deploy the `.zip` file to the App Service you identified by the `azureServiceConnectionId` and `webAppName` variables at the beginning of the pipeline file. Paste the

following code at the end of the file:

```
jobs:
- deployment: DeploymentJob
pool:
  vmImage: $(vmImageName)
environment: $(environmentName)
strategy:
  runOnce:
    deploy:
      steps:

        - task: UsePythonVersion@0
          inputs:
            versionSpec: '$(pythonVersion)'
            displayName: 'Use Python version'

        - task: AzureWebApp@1
          displayName: 'Deploy Azure Web App : {{ webAppName }}'
          inputs:
            azureSubscription: $(azureServiceConnectionId)
            appName: $(webAppName)
            package: $(Pipeline.Workspace)/drop/$(Build.BuildId).zip

          # The following parameter is specific to the Flask example code. You may
          # or may not need a startup command for your app.

          startUpCommand: 'gunicorn --bind=0.0.0.0 --workers=4 startup:app'
```

The `StartupCommand` parameter shown here is specific to the *python-vscode-flask-tutorial* example code, which defines the app in the *startup.py* file. By default, Azure App Service looks for the Flask app object in a file named *app.py* or *application.py*. If your code doesn't follow this pattern, you need to customize the startup command. Django apps may not need customization at all. For more information, see [How to configure Python on Azure App Service - Customize startup command](#).

Also, because the *python-vscode-flask-tutorial* repository contains the same startup command in a file named *startup.txt*, you could specify that file in the `StartupCommand` parameter rather than the command, by using `StartupCommand: 'startup.txt'`.

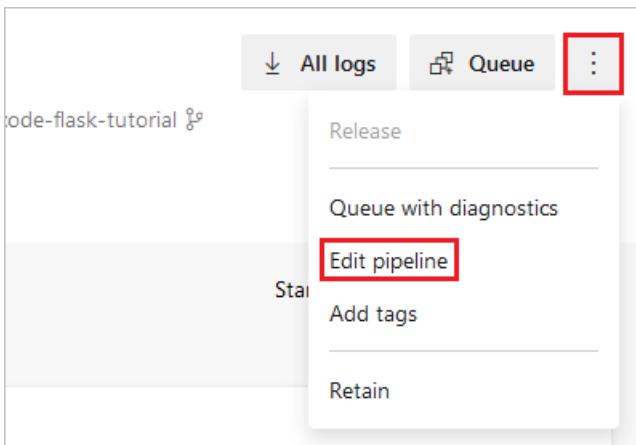
Run the pipeline

You're now ready to try it out!

1. Select **Save** at upper right in the editor, and in the pop-up window, add a commit message and select **Save**.
2. Select **Run** on the pipeline editor, and select **Run** again in the **Run pipeline** dialog box. Azure Pipelines queues another pipeline run, acquires an available build agent, and has that build agent run the pipeline.

The pipeline takes a few minutes to complete, especially the deployment steps. You should see green checkmarks next to each of the steps.

If there's an error, you can quickly return to the YAML editor by selecting the vertical dots at upper right and selecting **Edit pipeline**:



3. From the build page, select the **Azure Web App task** to display its output. To visit the deployed site, hold down the **Ctrl** key and select the URL after **App Service Application URL**.

If you're using the Flask example, the app should appear as follows:



IMPORTANT

If your app fails because of a missing dependency, then your `requirements.txt` file was not processed during deployment. This behavior happens if you created the web app directly on the portal rather than using the `az webapp up` command as shown in this article.

The `az webapp up` command specifically sets the build action `SCM_DO_BUILD_DURING_DEPLOYMENT` to `true`. If you provisioned the app service through the portal, however, this action is not automatically set.

The following steps set the action:

1. Open the [Azure portal](#), select your App Service, then select Configuration.
2. Under the **Application Settings** tab, select **New Application Setting**.
3. In the popup that appears, set **Name** to `SCM_DO_BUILD_DURING_DEPLOYMENT`, set **Value** to `true`, and select **OK**.
4. Select **Save** at the top of the **Configuration** page.
5. Run the pipeline again. Your dependencies should be installed during deployment.

Run a post-deployment script

A post-deployment script can, for example, define environment variables expected by the app code. Add the script as part of the app code and execute it using startup command.

To avoid hard-coding specific variable values in your YAML file, you can instead define variables in the pipeline's web interface and then refer to the variable name in the script. For more information, see [Variables - Secrets](#).

Considerations for Django

As noted earlier in this article, you can use Azure Pipelines to deploy Django apps to Azure App Service on Linux, provided that you're using a separate database. You can't use a SQLite database, because App Service locks the `db.sqlite3` file, preventing both reads and writes. This behavior doesn't affect an external database.

As described in [Configure Python app on App Service - Container startup process](#), App Service automatically looks for a `wsgi.py` file within your app code, which typically contains the app object. If you need to customize the startup command in any way, use the `StartupCommand` parameter in the `AzureWebApp@1` step of your YAML pipeline file, as described in the previous section.

When using Django, you typically want to migrate the data models using `manage.py migrate` after deploying the app code. You can add `startUpCommand` with post-deployment script for this purpose:

```
startUpCommand: python3.6 manage.py migrate
```

Run tests on the build agent

As part of your build process, you may want to run tests on your app code. Tests run on the build agent, so you probably need to first install your dependencies into a virtual environment on the build agent computer. After the tests run, delete the virtual environment before you create the `.zip` file for deployment. The following script elements illustrate this process. Place them before the `ArchiveFiles@2` task in the `azure-pipelines.yml` file. For more information, see [Run cross-platform scripts](#).

```
# The | symbol is a continuation character, indicating a multi-line script.
# A single-line script can immediately follow "- script:".
- script: |
    python3.6 -m venv .env
    source .env/bin/activate
    pip3.6 install setuptools
    pip3.6 install -r requirements.txt

    # The displayName shows in the pipeline UI when a build runs
    displayName: 'Install dependencies on build agent'

- script: |
    # Put commands to run tests here
    displayName: 'Run tests'

- script: |
    echo Deleting .env
    deactivate
    rm -rf .env
    displayName: 'Remove .env before zip'
```

You can also use a task like `PublishTestResults@2` to make test results appear in the pipeline results screen. For more information, see [Build Python apps - Run tests](#).

Provision an App Service with single commands

The `az webapp up` command used earlier in this article is a convenient method to provision the App Service and initially deploy your app in a single step. If you want more control over the deployment process, you can use single commands to accomplish the same tasks. For example, you might want to use a specific name for the resource group, or create an App Service within an existing App Service Plan.

The following steps perform the equivalent of the `az webapp up` command:

1. Create a resource group.

A resource group is a collection of related Azure resources. Creating a resource group makes it easy to delete all those resources at once when you no longer need them. In the Cloud Shell, run the following command to create a resource group in your Azure subscription. Set a location for the resource group by specifying the value of `<your-region>`. JSON output appears in the Cloud Shell when the command

completes successfully.

```
az group create -l <your-region> -n <your-resource-group>
```

2. Create an App Service Plan.

An App Service runs inside a VM defined by an App Service Plan. Run the following command to create an App Service Plan, substituting your own values for `<your-resource-group>` and `<your-appservice-plan>`. The `--is-linux` is required for Python deployments. If you want a pricing plan other than the default F1 Free plan, use the `sku` argument. The `--sku B1` specifies the lower-price compute tier for the VM. You can easily delete the plan later by deleting the resource group.

```
az appservice plan create -g <your-resource-group> -n <your-appservice-plan> --is-linux --sku B1
```

Again, you see JSON output in the Cloud Shell when the command completes successfully.

3. Create an App Service instance in the plan.

Run the following command to create the App Service instance in the plan, replacing `<your-appservice>` with a name that's unique across Azure. Typically, you use a personal or company name along with an app identifier, such as `<your-name>-flaskpipelines`. The command fails if the name is already in use. By assigning the App Service to the same resource group as the plan, it's easy to clean up all the resources at once.

```
az webapp create -g <your-resource-group> -p <your-appservice-plan> -n <your-appservice> --runtime "Python|3.6"
```

NOTE

If you want to deploy your code at the same time you create the app service, you can use the `--deployment-source-url` and `--deployment-source-branch` arguments with the `az webapp create` command. For more information, see [az webapp create](#).

TIP

If you see the error message "The plan (name) doesn't exist", and you're sure that the plan name is correct, check that the resource group specified with the `-g` argument is also correct, and the plan you identify is part of that resource group. If you misspell the resource group name, the command doesn't find the plan in that nonexistent resource group, and gives this particular error.

4. If your app requires a custom startup command, use the `az webapp config set` command, as described earlier in [Provision the target Azure App Service](#). For example, to customize the App Service with your resource group, app name, and startup command, run:

```
az webapp config set -g <your-resource-group> -n <your-appservice> --startup-file <your-startup-command-or-file>
```

The App Service at this point contains only default app code. You can now use Azure Pipelines to deploy your specific app code.

Clean up resources

To avoid incurring ongoing charges for any Azure resources you created in this walkthrough, such as a B1 App Service Plan, delete the resource group that contains the App Service and the App Service Plan. To delete the resource group from the Azure portal, select **Resource groups** in the left navigation. In the resource group list, select the ... to the right of the resource group you want to delete, select **Delete resource group**, and follow the prompts.

You can also use `az group delete` in the Cloud Shell to delete resource groups.

To delete the storage account that maintains the file system for Cloud Shell, which incurs a small monthly charge, delete the resource group that begins with `cloud-shell-storage-`.

Next steps

- [Build Python apps](#)
- [Learn about build agents](#)
- [Configure Python app on App Service](#)

Run pipelines with Anaconda environments

5/11/2021 • 3 minutes to read • [Edit Online](#)

Azure Pipelines

Learn how to set up and use Anaconda with Python in your pipeline. Anaconda is a Python distribution for data science and machine learning.

Get started

Follow these instructions to set up a pipeline for a sample Python app with Anaconda environment.

1. The code in the following repository is a simple Python app. To get started, fork this repo to your GitHub account.

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

2. Sign in to your Azure DevOps organization and navigate to your project.
3. In your project, navigate to the **Pipelines** page. Then choose the action to create a new pipeline.
4. Walk through the steps of the wizard by first selecting **GitHub** as the location of your source code.
5. You might be redirected to GitHub to sign in. If so, enter your GitHub credentials.
6. When the list of repositories appears, select your Java sample repository.
7. Azure Pipelines will analyze the code in your repository and detect an existing `azure-pipelines.yml` file.
8. Select **Run**.
9. A new run is started. Wait for the run to finish.

TIP

To make changes to the YAML file as described in this topic, select the pipeline in the **Pipelines** page, and then **Edit the `azure-pipelines.yml` file**.

Add conda to your system path

On [hosted agents](#), conda is left out of `PATH` by default to keep its Python version from conflicting with other installed versions. The `task.prependpath` agent command will make it available to all subsequent steps.

- [Hosted Ubuntu 18.04](#)
- [Hosted macOS](#)
- [Hosted VS2017](#)

```
- bash: echo "##vso[task.prependpath]$CONDA/bin"  
displayName: Add conda to PATH
```

Create an environment

From command-line arguments

The `conda create` command will create an environment with the arguments you pass it.

- [Hosted Ubuntu 18.04](#)
- [Hosted macOS](#)
- [Hosted VS2017](#)

```
- bash: conda create --yes --quiet --name myEnvironment  
displayName: Create Anaconda environment
```

From YAML

You can check in an `environment.yml` file to your repo that defines the configuration for an Anaconda environment.

```
- script: conda env create --quiet --file environment.yml  
displayName: Create Anaconda environment
```

NOTE

If you are using a self-hosted agent and don't remove the environment at the end, you'll get an error on the next build since the environment already exists. To resolve, use the `--force` argument:

```
conda env create --quiet --force --file environment.yml .
```

Install packages from Anaconda

The following YAML installs the `scipy` package in the conda environment named `myEnvironment`.

- [Hosted Ubuntu 18.04](#)
- [Hosted macOS](#)
- [Hosted VS2017](#)

```
- bash: |  
  source activate myEnvironment  
  conda install --yes --quiet --name myEnvironment scipy  
displayName: Install Anaconda packages
```

Run pipeline steps in an Anaconda environment

NOTE

Each build step runs in its own process. When you activate an Anaconda environment, it will edit `PATH` and make other changes to its current process. Therefore, an Anaconda environment must be activated separately for each step.

- [Hosted Ubuntu 18.04](#)
- [Hosted macOS](#)
- [Hosted VS2017](#)

```
- bash: |
  source activate myEnvironment
  python -m pytest --junitxml=junit/unit-test.xml
  displayName: pytest

- task: PublishTestResults@2
  inputs:
    testResultsFiles: 'junit/*.xml'
  condition: succeededOrFailed()
```

FAQs

Why am I getting a "Permission denied" error?

On Hosted macOS, the agent user doesn't have ownership of the directory where Miniconda is installed. For a fix, see the "Hosted macOS" tab under [Add conda to your system path](#).

Why does my build stop responding on a `conda create` or `conda install` step?

If you forget to pass `--yes`, conda will stop and wait for user interaction.

Why is my script on Windows stopping after it activates the environment?

On Windows, `activate` is a Batch script. You must use the `call` command to resume running your script after activating. See examples of using `call` [above](#).

How can I run my tests with multiple versions of Python?

See [Build Python apps in Azure Pipelines](#).

Build C++ Windows apps

4/27/2021 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

This guidance explains how to automatically build C++ projects for Windows.

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 [Create your first pipeline](#) 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. From the **Execution plan** section, select **Multi-configuration** to change the options for the job:

- Specify **Multipliers**: `BuildConfiguration, BuildPlatform`
 - Specify **Maximum number of agents**
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, perform these steps:

1. Click the [Copy Files task](#). Specify the following arguments:

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

Build Java apps

11/2/2020 • 6 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | 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 automatically build Java projects. (If you're working on an Android project, see [Build, test, and deploy Android apps](#).)

Create your first pipeline

Are you new to Azure Pipelines? If so, then we recommend you try this section to create before moving on to other sections.

Get the code

Fork this repo in GitHub:

Import this repo into your Git repo in Azure DevOps Server 2019:

Import this repo into your Git repo in TFS:

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

Sign in to Azure Pipelines

Sign in to [Azure Pipelines](#). After you sign in, your browser goes to <https://dev.azure.com/my-organization-name> and displays your Azure DevOps dashboard.

Within your selected organization, create a *project*. If you don't have any projects in your organization, you see a **Create a project to get started** screen. Otherwise, select the **Create Project** button in the upper-right corner of the dashboard.

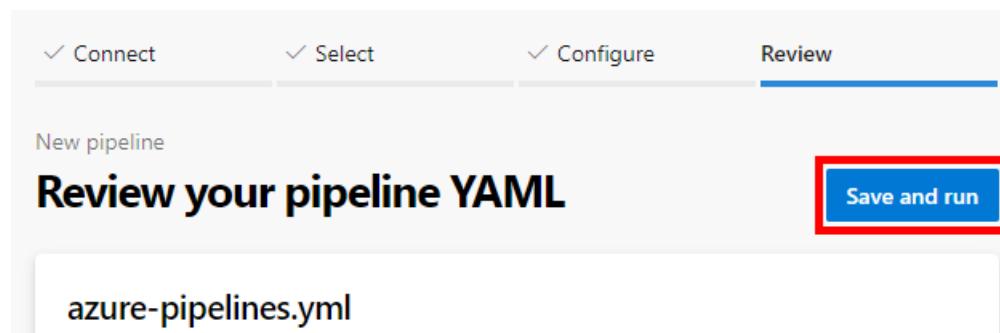
Create the pipeline

1. Sign in to your Azure DevOps organization and navigate to your project.
2. Go to **Pipelines**, and then select **New Pipeline**.
3. Walk through the steps of the wizard by first selecting **GitHub** as the location of your source code.
4. You might be redirected to GitHub to sign in. If so, enter your GitHub credentials.
5. When the list of repositories appears, select your repository.
6. You might be redirected to GitHub to install the Azure Pipelines app. If so, select **Approve & install**.

When the **Configure** tab appears, select **Maven**.

1. When your new pipeline appears, take a look at the YAML to see what it does. When you're ready, select

Save and run.



2. You're prompted to commit a new `azure-pipelines.yml` file to your repository. After you're happy with the message, select **Save and run** again.

If you want to watch your pipeline in action, select the build job.

You just created and ran a pipeline that we automatically created for you, because your code appeared to be a good match for the [Maven](#) template.

You now have a working YAML pipeline (`azure-pipelines.yml`) in your repository that's ready for you to customize!

3. When you're ready to make changes to your pipeline, select it in the **Pipelines** page, and then **Edit** the `azure-pipelines.yml` file.

4. See the sections below to learn some of the more common ways to customize your pipeline.

1. Create a pipeline (if you don't know how, see [Create your first pipeline](#), and for the template select **Maven**. This template automatically adds the tasks you need to build the code in the sample repository.
2. Save the pipeline and queue a build. When the **Build #nnnnnnnn.n has been queued** message appears, select the number link to see your pipeline in action.

You now have a working pipeline that's ready for you to customize!

3. When you're ready to make changes to your pipeline, **Edit** it.
4. See the sections below to learn some of the more common ways to customize your pipeline.

Build environment

You can use Azure Pipelines to build Java apps without needing to set up any infrastructure of your own. You can build on Windows, Linux, or MacOS images. The Microsoft-hosted agents in Azure Pipelines have modern JDKs and other tools for Java pre-installed. To know which versions of Java are installed, see [Microsoft-hosted agents](#).

Update the following snippet in your `azure-pipelines.yml` file to select the appropriate image.

```
pool:  
  vmImage: 'ubuntu-16.04' # other options: 'macOS-10.14', 'vs2017-win2016'
```

See [Microsoft-hosted agents](#) for a complete list of images.

As an alternative to using Microsoft-hosted agents, you can set up [self-hosted agents](#) with Java installed. You can also use self-hosted agents to save additional time if you have a large repository or you run incremental builds.

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

Build your code

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 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](#).

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'
```

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](#).

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'
```

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'
```

Next Steps

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

Next we recommend that you learn more about creating a CI/CD pipeline for the deployment target you choose:

- [Build and deploy to a Java web app](#)
- [Build and deploy Java to Azure Functions](#)
- [Build and deploy Java to Azure Kubernetes service](#)

Build and deploy to a Java web app

2/26/2020 • 5 minutes to read • [Edit Online](#)

Azure Pipelines

A web app is a lightweight way to host a web application. In this step-by-step guide you'll learn how to create a pipeline that continuously builds and deploys your Java app. Your team can then automatically build each commit in GitHub, and if you want, automatically deploy the change to an Azure App Service. You can use whatever runtime you prefer: Tomcat, or Java SE.

Prerequisites

- A GitHub account, where you can create a repository. If you don't have one, you can [create one for free](#).
- An Azure DevOps organization. If you don't have one, you can [create one for free](#). (An Azure DevOps organization is different from your GitHub organization. Give them the same name if you want alignment between them.)
If your team already has one, then make sure you're an administrator of the Azure DevOps project that you want to use.
- An ability to run pipelines on Microsoft-hosted agents. You can either purchase a [parallel job](#) or you can request a free tier. To request a free tier, follow the instructions in [this article](#). Please note that it may take us 2-3 business days to grant the free tier.
- An Azure account. If you don't have one, you can [create one for free](#).

TIP

If you're new at this, the easiest way to get started is to use the same email address as the owner of both the Azure Pipelines organization and the Azure subscription.

Get the code

Select the runtime you want to use.

- [Tomcat](#)
- [Java SE](#)

If you already have an app in GitHub that you want to deploy, you can create a pipeline for that code.

If you are a new user, fork this repo in GitHub:

```
https://github.com/spring-petclinic/spring-framework-petclinic
```

Create an Azure App Service

Sign in to the [Azure Portal](#), and then select the [Cloud Shell](#) button in the upper-right corner.

Create an Azure App Service on Linux.

- [Tomcat](#)
- [Java SE](#)

```
# Create a resource group
az group create --location westus --name myapp-rg

# Create an app service plan of type Linux
az appservice plan create -g myapp-rg -n myapp-service-plan --is-linux

# Create an App Service from the plan with Tomcat and JRE 8 as the runtime
az webapp create -g myapp-rg -p myapp-service-plan -n my-app-name --runtime "TOMCAT|8.5-jre8"
```

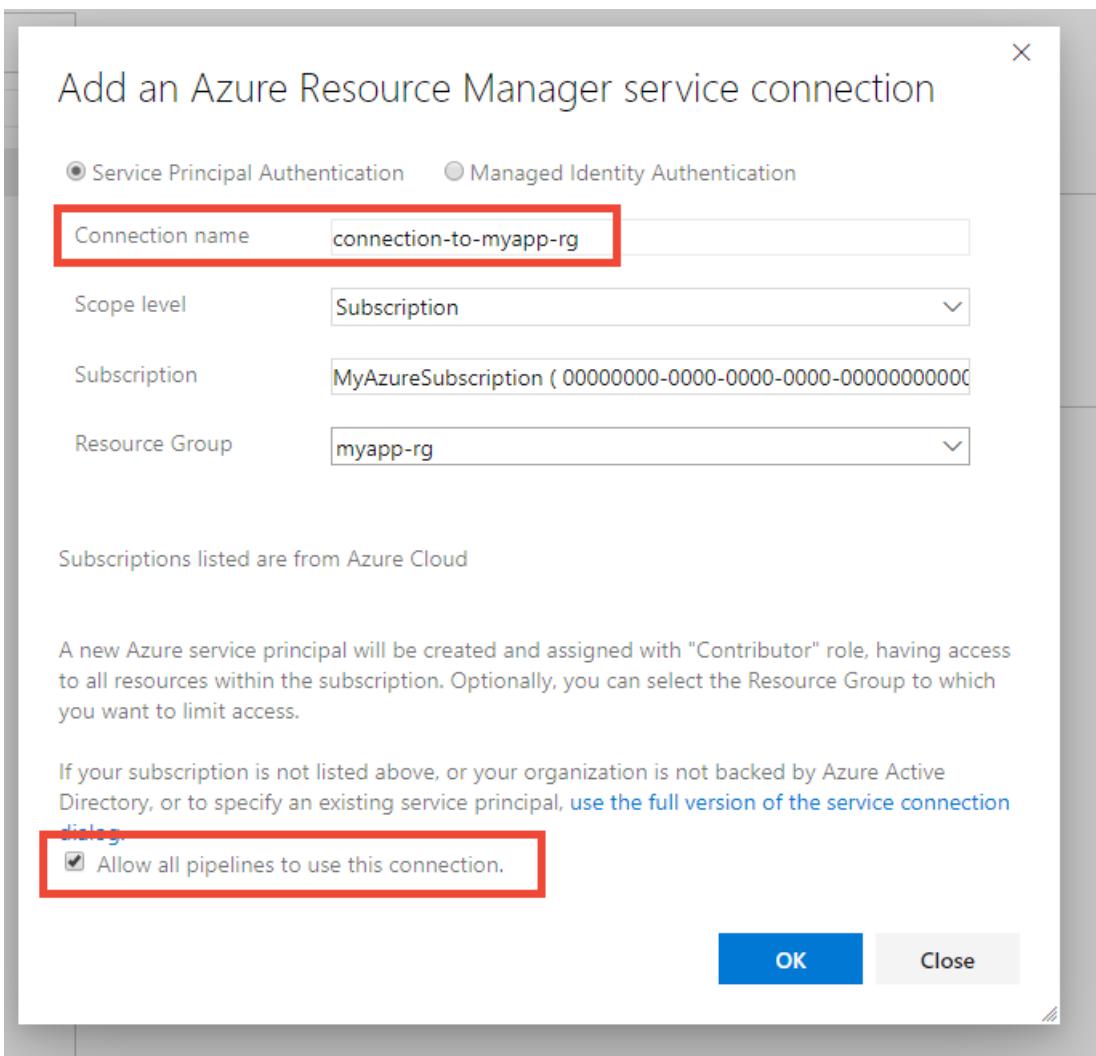
Sign in to Azure Pipelines and connect to Azure

Sign in to [Azure Pipelines](#). After you sign in, your browser goes to <https://dev.azure.com/my-organization-name> and displays your Azure DevOps dashboard.

Within your selected organization, create a *project*. If you don't have any projects in your organization, you see a **Create a project to get started** screen. Otherwise, select the **Create Project** button in the upper-right corner of the dashboard.

Now create the service connection:

1. From your project dashboard, select **Project settings** on the bottom left.
2. On the settings page, select **Pipelines > Service connections**, select **New service connection**, and then select **Azure Resource Manager**.
3. The *Add an Azure Resource Manager service connection** dialog box appears.
 - **Name** Type a name and then copy and paste it into a text file so you can use it later.
 - **Scope** Select Subscription.
 - **Subscription** Select the subscription in which you created the App Service.
 - **Resource Group** Select the resource group you created earlier
 - Select **Allow all pipelines to use this connection**.



TIP

If you need to create a connection to an Azure subscription that's owned by someone else, see [Create an Azure Resource Manager service connection with an existing service principal](#).

Create the pipeline

1. Sign in to your Azure DevOps organization and navigate to your project.
2. Go to **Pipelines**, and then select **New Pipeline**.
3. Walk through the steps of the wizard by first selecting **GitHub** as the location of your source code.
4. You might be redirected to GitHub to sign in. If so, enter your GitHub credentials.
5. When the list of repositories appears, select your repository.
6. You might be redirected to GitHub to install the Azure Pipelines app. If so, select **Approve & install**.

When the **Configure** tab appears, select **Show more**, and then select **Maven package Java project Web App to Linux on Azure**. Your new pipeline appears.

1. When prompted, select the Azure subscription in which you created your Web App.
2. Select the Web App.
3. Select **Validate and configure**.

As Azure Pipelines creates an `azure-pipelines.yml` file, which defines your CI/CD pipeline, it:

- Includes a Build stage, which builds your project, and a Deploy stage, which deploys it to Azure as a Linux web app.
 - As part of the Deploy stage, it also creates an [Environment](#) with default name same as the Web App. You can choose to modify the environment name.
4. Take a look at the pipeline to see what it does. Make sure that all the default inputs are appropriate for your code.
5. After you've looked at what the pipeline does, select **Save and run**, after which you're prompted for a commit message because Azure Pipelines adds the *azure-pipelines.yml* file to your repository. After editing the message, select **Save and run** again to see your pipeline in action.

See the pipeline run, and your app deployed

As your pipeline runs, watch as your build stage, and then your deployment stage, go from blue (running) to green (completed). You can select the stages and jobs to watch your pipeline in action.

After the pipeline has run, check out your site!

```
https://my-app-name.azurewebsites.net
```

Also explore deployment history for the App by navigating to the "Environment". From the pipeline summary:

1. Select the **Environments** tab.
2. Select **View environment**.

Clean up resources

Whenever you're done with the resources you created above, you can use the following command to delete them:

```
az group delete --name myapp-rg
```

Type `y` when prompted.

Build and deploy Java to Azure Functions

4/27/2021 • 5 minutes to read • [Edit Online](#)

Azure Pipelines

You can use Azure Functions to run small pieces of code in the cloud without the overhead of running a server. In this step-by-step guide you'll learn how to create a pipeline that continuously builds and deploys a your Java function app. Your team can then automatically build each commit in GitHub, and if you want, automatically deploy the change to Azure Functions.

Prerequisites

- A GitHub account, where you can create a repository. If you don't have one, you can [create one for free](#).
- An Azure DevOps organization. If you don't have one, you can [create one for free](#). (An Azure DevOps organization is different from your GitHub organization. Give them the same name if you want alignment between them.)
If your team already has one, then make sure you're an administrator of the Azure DevOps project that you want to use.
- An ability to run pipelines on Microsoft-hosted agents. You can either purchase a [parallel job](#) or you can request a free tier. To request a free tier, follow the instructions in [this article](#). Please note that it may take us 2-3 business days to grant the free tier.
- An Azure account. If you don't have one, you can [create one for free](#).

TIP

If you're new at this, the easiest way to get started is to use the same email address as the owner of both the Azure Pipelines organization and the Azure subscription.

Get the code

If you already have an app in GitHub that you want to deploy, you can create a pipeline for that code.

If you are a new user, fork this repo in GitHub:

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

Create an Azure Functions app

Sign in to the [Azure Portal](#), and then select the [Cloud Shell](#) button in the upper-right corner.

Create an Azure App Service on Linux. Select the runtime you want to use.

```
# Create a resource group
az group create --location westus --name myapp-rg

# Create a storage account
az storage account create --name mystorage --location westeurope --resource-group myapp-rg --sku Standard_LRS

# Create an Azure Functions app
az functionapp create --resource-group myapp-rg --consumption-plan-location westeurope \
--name my-app-name --storage-account mystorage --runtime java
```

Update pom.xml

Update `pom.xml` to reflect your application. Edit the `pom.xml` file in your forked repository with your values for `FUNCTION_APP_NAME`, `REGION`, and `RESOURCE_GROUP`.

```
<functionAppName>FUNCTION_APP_NAME</functionAppName>
<functionAppRegion>REGION</functionAppRegion>
<stagingDirectory>${project.build.directory}/azure-functions/${functionAppName}</stagingDirectory>
<functionResourceGroup>RESOURCE_GROUP</functionResourceGroup>
```

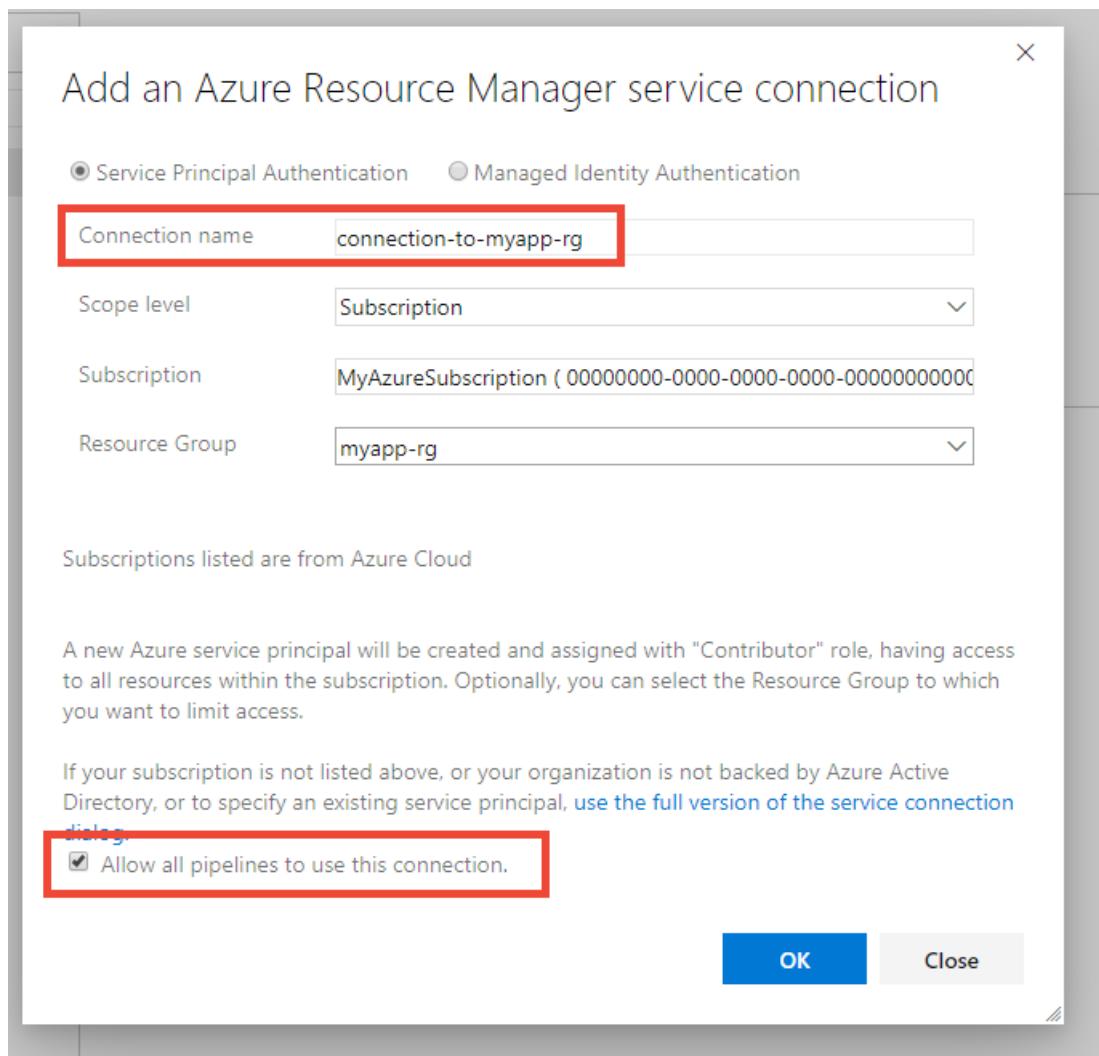
Sign in to Azure Pipelines and connect to Azure

Sign in to [Azure Pipelines](#). After you sign in, your browser goes to <https://dev.azure.com/my-organization-name> and displays your Azure DevOps dashboard.

Within your selected organization, create a *project*. If you don't have any projects in your organization, you see a **Create a project to get started** screen. Otherwise, select the **Create Project** button in the upper-right corner of the dashboard.

Now create the service connection:

1. From your project dashboard, select **Project settings** on the bottom left.
2. On the settings page, select **Pipelines > Service connections**, select **New service connection**, and then select **Azure Resource Manager**.
3. The *Add an Azure Resource Manager service connection** dialog box appears.
 - **Name** Type a name and then copy and paste it into a text file so you can use it later.
 - **Scope** Select Subscription.
 - **Subscription** Select the subscription in which you created the App Service.
 - **Resource Group** Select the resource group you created earlier
 - Select **Allow all pipelines to use this connection**.



TIP

If you need to create a connection to an Azure subscription that's owned by someone else, see [Create an Azure Resource Manager service connection with an existing service principal](#).

Create the pipeline

1. Sign in to your Azure DevOps organization and navigate to your project.
2. Go to **Pipelines**, and then select **New Pipeline**.
3. Walk through the steps of the wizard by first selecting **GitHub** as the location of your source code.
4. You might be redirected to GitHub to sign in. If so, enter your GitHub credentials.
5. When the list of repositories appears, select your repository.
6. You might be redirected to GitHub to install the Azure Pipelines app. If so, select **Approve & install**.

When the **Configure** tab appears, select **Maven**. Your new pipeline appears.

1. When prompted, select the Azure subscription in which you created your Web App.
2. Select the Web App.
3. Select **Validate and configure**.

As Azure Pipelines creates an `azure-pipelines.yml` file, which defines your CI/CD pipeline, it:

- Includes a Build stage, which builds your project, and a Deploy stage, which deploys it to Azure as a Linux web app.
 - As part of the Deploy stage, it also creates an [Environment](#) with default name same as the Web App. You can choose to modify the environment name.
4. Take a look at the pipeline to see what it does. Make sure that all the default inputs are appropriate for your code.
5. After you've looked at what the pipeline does, select **Save and run**, after which you're prompted for a commit message because Azure Pipelines adds the *azure-pipelines.yml* file to your repository. After editing the message, select **Save and run** again to see your pipeline in action.

You just created and ran a pipeline that we automatically created for because your code appeared to be a good match for the [Maven Azure Pipelines template](#).

Edit the pipeline

After the pipeline has run, select the vertical ellipses in the upper-right corner of the window and then select **Edit pipeline**.

Set some variables for your deployment

```
# at the top of your YAML file
# set some variables that you'll need when you deploy
variables:
  # the name of the service connection that you created above
  serviceConnectionToAzure: name-of-your-service-connection
  # the name of your web app here is the same one you used above
  # when you created the web app using the Azure CLI
  appName: my-app-name
  # the name of your functionApp is what you provided as
  # stagingDirectory in pom.xml
  functionName: 'javafuctions'
  # ...
```

Deploy to Azure Functions

```
# ...
# add these as the last steps
# to deploy to your app service
- task: CopyFiles@2
  displayName: Copy Files
  inputs:
    SourceFolder: $(system.defaultworkingdirectory)/target/azure-functions/$(functionAppName)/
    Contents: '**'
    TargetFolder: $(build.artifactstagingdirectory)

- task: PublishBuildArtifacts@1
  displayName: Publish Artifact
  inputs:
    PathtoPublish: $(build.artifactstagingdirectory)

- task: AzureFunctionApp@1
  displayName: Azure Function App deploy
  inputs:
    azureSubscription: $(serviceConnectionToAzure)
    appType: functionApp
    appName: $(appName)
    package: $(build.artifactstagingdirectory)/$(appName)
```

Run the pipeline and check out your site

You're now ready to save your changes and try it out!

1. Select **Save** in the upper-right corner of the editor.
2. In the dialog box that appears, add a **Commit message** such as `add deployment to our pipeline`, and then select **Save**.
3. In the pipeline editor, select **Run**.

When the **Build #nnnnnnnn.n has been queued** message appears, select the number link to see your pipeline in action.

After the pipeline has run, test the function app running on Azure. For example, in bash or from a command prompt enter:

```
curl -w '\n' https://my-app-name-0000000000000000.azurewebsites.net/api/HttpTrigger-Java -d fromYourPipeline
```

Your function then returns:

```
Hello PipelineCreator
```

Clean up resources

Whenever you're done with the resources you created above, you can use the following command to delete them:

```
az group delete --name myapp-rg
```

Type `y` when prompted.

Build, test, and deploy Android apps

3/26/2021 • 6 minutes to read • [Edit Online](#)

This guidance explains how to automatically build, test, and deploy Android apps.

Get started

Follow these instructions to set up a pipeline for a sample Android app.

1. The code in the following repository is a simple Android app. To get started, fork this repo to your GitHub account.

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

2. Sign in to your Azure DevOps organization and navigate to your project.
3. In your project, navigate to the **Pipelines** page. Then choose the action to create a new pipeline.
4. Walk through the steps of the wizard by first selecting **GitHub** as the location of your source code.
5. You might be redirected to GitHub to sign in. If so, enter your GitHub credentials.
6. When the list of repositories appears, select your Java sample repository.
7. Azure Pipelines will analyze the code in your repository and recommend starter templates for your pipeline. Select the **Android** template.
8. Azure Pipelines will generate a YAML file for your pipeline. Select **Save and run**, then select **Commit directly to the main branch**, and then choose **Save and run** again.
9. A new run is started. Wait for the run to finish.

When you're done, you'll have a working YAML file (`azure-pipelines.yml`) in your repository that's ready for you to customize.

TIP

To make changes to the YAML file as described in this topic, select the pipeline in the **Pipelines** page, and then **Edit** the `azure-pipelines.yml` file.

Gradle

Gradle is a common build tool used for building Android projects. See the **Gradle** task for more about these options.

```

# https://docs.microsoft.com/azure/devops/pipelines/ecosystems/android
pool:
  vmImage: 'macOS-10.14'

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.

Create the [Bash Task](#) and copy paste the code below in order to install and run the emulator. Don't forget to arrange the emulator parameters to fit your testing environment. The emulator will be started as a background process and available in subsequent tasks.

```
#!/usr/bin/env bash

# Install AVD files
echo "y" | $ANDROID_HOME/tools/bin/sdkmanager --install 'system-images;android-27;google_apis;x86'

# Create emulator
echo "no" | $ANDROID_HOME/tools/bin/avdmanager create avd -n xamarin_android_emulator -k 'system-
images;android-27;google_apis;x86' --force

$ANDROID_HOME/emulator/emulator -list-avds

echo "Starting emulator"

# Start emulator in background
nohup $ANDROID_HOME/emulator/emulator -avd xamarin_android_emulator -no-snapshot > /dev/null 2>&1 &
$ANDROID_HOME/platform-tools/adb wait-for-device shell 'while [[ -z $(getprop sys.boot_completed | tr -d
'\r') ]]; do sleep 1; done; input keyevent 82'

$ANDROID_HOME/platform-tools/adb devices

echo "Emulator started"
```

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.

[Sign up with App Center](#) first.

```

# App Center test
# Test app packages with Visual Studio App Center
- task: AppCenterTest@1
  inputs:
    appFile:
    #artifactsDirectory: '$(Build.ArtifactStagingDirectory)/AppCenterTest'
    #prepareTests: true # 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
    #uitestStorePath: # Optional
    #uitestStorePassword: # Optional
    #uitestKeyAlias: # Optional
    #uitestKeyPassword: # Optional
    #uitestToolsDirectory: # Optional
    #signInfo: # Optional
    #xcUITestBuildDirectory: # Optional
    #xcUITestIpaFile: # Optional
    #prepareOptions: # Optional
    #runTests: true # 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 Visual Studio 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
    #isMandatory: false # Optional
    #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)

FAQ

How do I create app bundles?

You can build and sign your app bundle with an inline script and a secure file. To do this, you'll need to first download your keystore and [store it as a secure file in the Library](#). You'll also need to create variables for `keystore.password`, `key.alias`, and `key.password` in a [variable group](#).

Next, you'll use the [Download Secure File](#) and [Bash](#) tasks to download your keystore and build and sign your app bundle.

In this YAML file, you download an `app.keystore` secure file and use a bash script to generate an app bundle. Then, you use [Copy Files](#) to copy the app bundle. From there, you can create and save an artifact with [Publish Build Artifact](#) or use the [Google Play extension](#) to publish.

```
- task: DownloadSecureFile@1
  name: keyStore
  displayName: "Download keystore from secure files"
  inputs:
    secureFile: app.keystore

- task: Bash@3
  displayName: "Build and sign App Bundle"
  inputs:
    targetType: "inline"
    script: |
      msbuild -restore $(Build.SourcesDirectory)/myAndroidApp/*.csproj -t:SignAndroidPackage -
      p:AndroidPackageFormat=aab -p:Configuration=$(buildConfiguration) -p:AndroidKeyStore=True -
      p:AndroidSigningKeyStore=$(keyStore.secureFilePath) -p:AndroidSigningStorePass=$(keystore.password) -
      p:AndroidSigningKeyAlias=$(key.alias) -p:AndroidSigningKeyPass=$(key.password)

- task: CopyFiles@2
  displayName: 'Copy deliverables'
  inputs:
    SourceFolder: '$(Build.SourcesDirectory)/myAndroidApp/bin/$(buildConfiguration)'
    Contents: '*.aab'
    TargetFolder: 'drop'
```

Build and test Go projects

11/2/2020 • 5 minutes to read • [Edit Online](#)

Azure Pipelines

Use a pipeline to automatically build and test your Go projects.

Create your first pipeline

Are you new to Azure Pipelines? If so, then we recommend you try this section before moving on to other sections.

Import this repo into your Git repo:

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

Sign in to Azure Pipelines

Sign in to [Azure Pipelines](#). After you sign in, your browser goes to <https://dev.azure.com/my-organization-name> and displays your Azure DevOps dashboard.

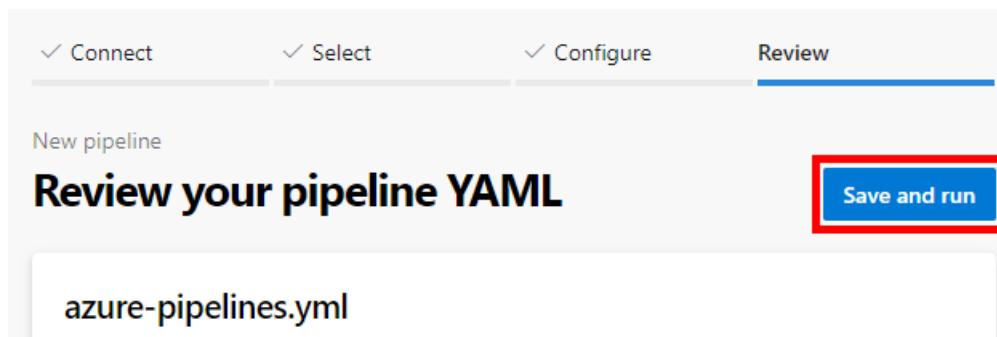
Within your selected organization, create a *project*. If you don't have any projects in your organization, you see a **Create a project to get started** screen. Otherwise, select the **Create Project** button in the upper-right corner of the dashboard.

Create the pipeline

1. Sign in to your Azure DevOps organization and navigate to your project.
2. Go to **Pipelines**, and then select **New Pipeline**.
3. Walk through the steps of the wizard by first selecting **GitHub** as the location of your source code.
4. You might be redirected to GitHub to sign in. If so, enter your GitHub credentials.
5. When the list of repositories appears, select your repository.
6. You might be redirected to GitHub to install the Azure Pipelines app. If so, select **Approve & install**.

When the **Configure** tab appears, select **Go**.

7. When your new pipeline appears, take a look at the YAML to see what it does. When you're ready, select **Save and run**.



8. You're prompted to commit a new `azure-pipelines.yml` file to your repository. After you're happy with the message, select **Save and run** again.

If you want to watch your pipeline in action, select the build job.

You just created and ran a pipeline that we automatically created for you, because your code appeared to be a good match for the [Go](#) template.

You now have a working YAML pipeline (`azure-pipelines.yml`) in your repository that's ready for you to customize!

9. When you're ready to make changes to your pipeline, select it in the **Pipelines** page, and then **Edit** the `azure-pipelines.yml` file.

See the sections below to learn some of the more common ways to customize your pipeline.

TIP

To make changes to the YAML file as described in this topic, select the pipeline in **Pipelines** page, and then select **Edit** to open an editor for the `azure-pipelines.yml` file.

Build environment

You can use Azure Pipelines to build your Go projects without needing to set up any infrastructure of your own. You can use Linux, macOS, or Windows agents to run your builds.

Update the following snippet in your `azure-pipelines.yml` file to select the appropriate image.

```
pool:  
  vmImage: 'ubuntu-latest'
```

Modern versions of Go are pre-installed on [Microsoft-hosted agents](#) in Azure Pipelines. For the exact versions of Go that are pre-installed, refer to [Microsoft-hosted agents](#).

Set up Go

- [Go 1.11+](#)
- [Go < 1.11](#)

Starting with Go 1.11, you no longer need to define a `$GOPATH` environment, set up a workspace layout, or use the `dep` module. Dependency management is now built-in.

This YAML implements the `go get` command to download Go packages and their dependencies. It then uses `go build` to generate the content that is published with `PublishBuildArtifacts@1` task.

```
trigger:
- master

pool:
  vmImage: 'ubuntu-latest'

steps:
- task: GoTool@0
  inputs:
    version: '1.13.5'
- task: Go@0
  inputs:
    command: 'get'
    arguments: '-d'
    workingDirectory: '$(System.DefaultWorkingDirectory)'
- task: Go@0
  inputs:
    command: 'build'
    workingDirectory: '$(System.DefaultWorkingDirectory)'
- task: CopyFiles@2
  inputs:
    TargetFolder: '$(Build.ArtifactStagingDirectory)'
- task: PublishBuildArtifacts@1
  inputs:
    artifactName: drop
```

Build

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

```
- task: Go@0
  inputs:
    command: 'build'
    workingDirectory: '$(System.DefaultWorkingDirectory)'
```

Test

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

```
- task: Go@0
  inputs:
    command: 'test'
    arguments: '-v'
    workingDirectory: '$(modulePath)'
```

Build an image and push to container registry

For your Go app, you can also [build an image](#) and [push it to a container registry](#).

Related extensions

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

Build and test PHP apps

3/6/2021 • 3 minutes to read • [Edit Online](#)

Azure Pipelines

Create a pipeline that continuously builds, deploys, and tests your PHP projects.

For an end-to-end walkthrough of deploying to Azure App Service with a pipeline, see [Build and deploy to a PHP web app](#).

Create your first pipeline

Are you new to Azure Pipelines? If so, then we recommend you try this section before moving on to other sections.

Get the code

Fork this repo in GitHub:

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

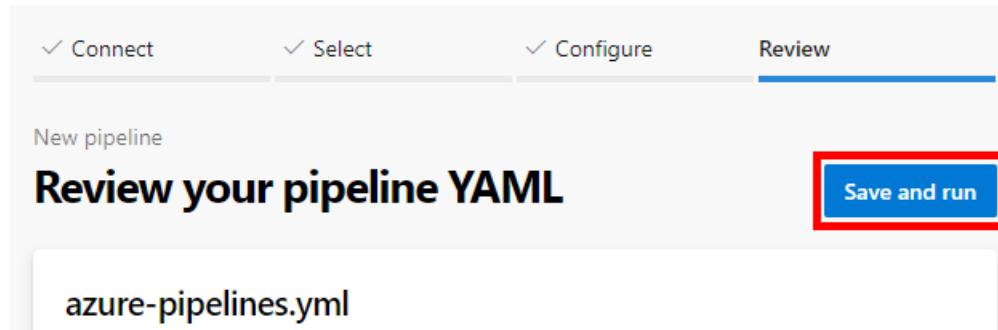
Sign in to Azure Pipelines

Sign in to [Azure Pipelines](#). After you sign in, your browser goes to <https://dev.azure.com/my-organization-name> and displays your Azure DevOps dashboard.

Within your selected organization, create a *project*. If you don't have any projects in your organization, you see a **Create a project to get started** screen. Otherwise, select the **Create Project** button in the upper-right corner of the dashboard.

When the **Configure** tab appears, select **PHP**.

1. When your new pipeline appears, take a look at the YAML to see what it does. When you're ready, select **Save and run**.



2. You're prompted to commit a new *azure-pipelines.yml* file to your repository. After you're happy with the message, select **Save and run** again.

If you want to watch your pipeline in action, select the build job.

You now have a working YAML pipeline (`azure-pipelines.yml`) in your repository that's ready for you to customize!

3. When you're ready to make changes to your pipeline, select it in the **Pipelines** page, and then **Edit the `azure-pipelines.yml` file**.

4. See the sections below to learn some of the more common ways to customize your pipeline.

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.

```
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 phpdbs /usr/bin/phpdbs$(phpVersion)  
    sudo update-alternatives --set php-cgi /usr/bin/php-cgi$(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 phunit

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

```
- script: ./phunit  
  displayName: 'Run tests with phunit'
```

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
```

Using a custom composer location

If your composer.json is in a subfolder instead of the root directory, you can use the `--working-dir` argument to tell composer what directory to use. For example, if your composer.json is inside the subfolder `pkgs`

```
composer install --no-interaction --working-dir=pkgs
```

You can also specify the absolute path, using the built-in system variables:

```
composer install --no-interaction --working-dir='${system.defaultWorkingDirectory}/pkgs'
```

Build image and push to container registry

For your PHP app, you can also [build an image](#) and [push it to a container registry](#).

Build and deploy to a PHP web app

3/6/2021 • 5 minutes to read • [Edit Online](#)

Azure Pipelines

A web app is a lightweight way to host a web application. In this step-by-step guide, you'll learn how to create a pipeline that continuously builds and deploys your PHP app. Your team can then automatically build each commit in GitHub, and if you want, automatically deploy the change to an Azure App Service. You can select different PHP versions.

Prerequisites

- A GitHub account, where you can create a repository. If you don't have one, you can [create one for free](#).
- An Azure DevOps organization. If you don't have one, you can [create one for free](#). (An Azure DevOps organization is different from your GitHub organization. Give them the same name if you want alignment between them.)
If your team already has one, then make sure you're an administrator of the Azure DevOps project that you want to use.
- An ability to run pipelines on Microsoft-hosted agents. You can either purchase a [parallel job](#) or you can request a free tier. To request a free tier, follow the instructions in [this article](#). Please note that it may take us 2-3 business days to grant the free tier.
- An Azure account. If you don't have one, you can [create one for free](#).

TIP

If you're new at this, the easiest way to get started is to use the same email address as the owner of both the Azure Pipelines organization and the Azure subscription.

Get the code

If you already have an app in GitHub that you want to deploy, you can try creating a pipeline for that code.

However, if you are a new user, then you might get a better start by using our sample code. In that case, fork this repo in GitHub:

```
https://github.com/Azure-Samples/php-docs-hello-world
```

Create an Azure App Service

Sign in to the [Azure Portal](#), and then select the [Cloud Shell](#) button in the upper-right corner.

Create an Azure App Service on Linux.

```
# Create a resource group
az group create --location westus --name myapp-rg

# Create an app service plan of type Linux
az appservice plan create -g myapp-rg -n myapp-service-plan --is-linux

# Create an App Service from the plan with PHP as the runtime
az webapp create -g myapp-rg -p myapp-service-plan -n my-app-name --runtime "PHP|7.0"
```

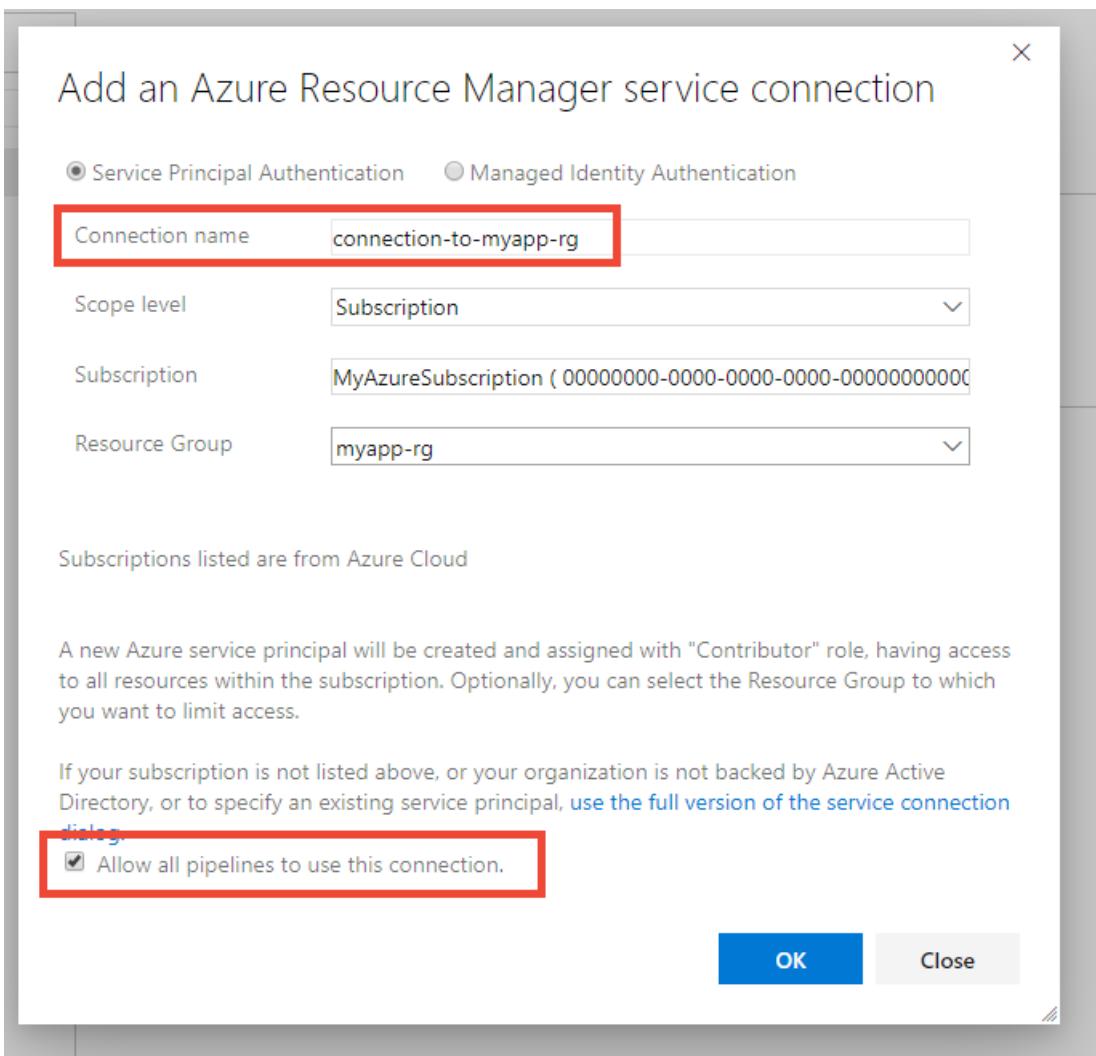
Sign in to Azure Pipelines and connect to Azure

Sign in to [Azure Pipelines](#). After you sign in, your browser goes to <https://dev.azure.com/my-organization-name> and displays your Azure DevOps dashboard.

Within your selected organization, create a *project*. If you don't have any projects in your organization, you see a **Create a project to get started** screen. Otherwise, select the **Create Project** button in the upper-right corner of the dashboard.

Now create the service connection:

1. From your project dashboard, select **Project settings** on the bottom left.
2. On the settings page, select **Pipelines > Service connections**, select **New service connection**, and then select **Azure Resource Manager**.
3. The *Add an Azure Resource Manager service connection** dialog box appears.
 - **Name** Type a name and then copy and paste it into a text file so you can use it later.
 - **Scope** Select Subscription.
 - **Subscription** Select the subscription in which you created the App Service.
 - **Resource Group** Select the resource group you created earlier
 - Select **Allow all pipelines to use this connection**.



TIP

If you need to create a connection to an Azure subscription that's owned by someone else, see [Create an Azure Resource Manager service connection with an existing service principal](#).

Create the pipeline

1. Sign in to your Azure DevOps organization and navigate to your project.
2. Go to **Pipelines**, and then select **New Pipeline**.
3. Walk through the steps of the wizard by first selecting **GitHub** as the location of your source code.
4. You might be redirected to GitHub to sign in. If so, enter your GitHub credentials.
5. When the list of repositories appears, select your repository.
6. You might be redirected to GitHub to install the Azure Pipelines app. If so, select **Approve & install**.

When the **Configure** tab appears, select **PHP**. Your new pipeline appears.

1. Take a look at the pipeline to see what it does.
2. After you've looked at what the pipeline does, select **Save and run** to see the pipeline in action.
3. Select **Save and run**, after which you're prompted for a commit message because Azure Pipelines added the *azure-pipelines.yml* file to your repository. After editing the message, select **Save and run** again.

You just created and ran a pipeline that we automatically created for you, because your code appeared to be a good match for the [PHP Azure Pipelines template](#).

Edit the pipeline

After the pipeline has run, select the vertical ellipses in the upper-right corner of the window and then select **Edit pipeline**.

Set some variables for your deployment

```
# at the top of your YAML file
# set some variables that you'll need when you deploy
variables:
  # the name of the service connection that you created above
  serviceConnectionToAzure: name-of-your-service-connection
  # the name of your web app here is the same one you used above
  # when you created the web app using the Azure CLI
  appName: my-app-name
  # the name of your functionApp is what you provided as
  # stagingDirectory in pom.xml
  functionAppName: 'javafunctions'
  # ...
```

Deploy to your app service

```
# add these as the last steps (below all the other `task` items under `steps`)
# to deploy to your app service
- task: ArchiveFiles@1
  displayName: Archive files
  inputs:
    rootFolder: $(System.DefaultWorkingDirectory)
    includeRootFolder: false
    archiveType: zip

- task: PublishBuildArtifacts@1
  displayName: Publish Artifact
  inputs:
    PathtoPublish: $(build.artifactstagingdirectory)

- task: AzureWebApp@1
  displayName: Azure Web App Deploy
  inputs:
    azureSubscription: $(serviceConnectionToAzure)
    appType: webAppLinux
    appName: $(appName)
    package: $(build.artifactstagingdirectory)/**/*.zip
```

Run the pipeline and check out your site

You're now ready to save your changes and try it out!

1. Select **Save** in the upper-right corner of the editor.
2. In the dialog box that appears, add a **Commit message** such as `add deployment to our pipeline`, and then select **Save**.
3. In the pipeline editor, select **Run**.

When the **Build #nnnnnnnn.n has been queued** message appears, select the number link to see your pipeline in action.

After the pipeline has run, check out your site!

```
https://my-app-name.azurewebsites.net/
```

Clean up resources

Whenever you're done with the resources you created above, you can use the following command to delete them:

```
az group delete --name myapp-rg
```

Type `y` when prompted.

Build and test Ruby apps

3/6/2021 • 3 minutes to read • [Edit Online](#)

Azure Pipelines

This guidance explains how to automatically build Ruby projects.

Get started

Follow these instructions to set up a pipeline for a Ruby app.

1. The code in the following repository is a simple Ruby app. To get started, fork this repo to your GitHub account.

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

2. Sign in to your Azure DevOps organization and navigate to your project.
3. In your project, navigate to the [Pipelines](#) page. Then choose the action to create a new pipeline.
4. Walk through the steps of the wizard by first selecting **GitHub** as the location of your source code.
5. You might be redirected to GitHub to sign in. If so, enter your GitHub credentials.
6. When the list of repositories appears, select your Ruby sample repository.
7. Azure Pipelines will analyze the code in your repository and recommend **Ruby** template for your pipeline. Select that template.
8. Azure Pipelines will generate a YAML file for your pipeline. Select **Save and run**, then select **Commit directly to the main branch**, and then choose **Save and run** again.
9. A new run is started. Wait for the run to finish.

When you're done, you'll have a working YAML file (`azure-pipelines.yml`) in your repository that's ready for you to customize.

TIP

To make changes to the YAML file as described in this topic, select the pipeline in the [Pipelines](#) page, and then **Edit the `azure-pipelines.yml` file**.

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/ecosystems/ruby
pool:
  vmImage: 'ubuntu-16.04' # other options: 'macOS-10.14', '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: |
  CALL 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
  condition: succeededOrFailed()
  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 an image and push to container registry

For your Ruby app, you can also [build an image](#) and [push it to a container registry](#).

Quickstart: Build and deploy Xamarin apps with a pipeline

5/24/2021 • 6 minutes to read • [Edit Online](#)

Azure Pipelines

Get started with [Xamarin](#) and Azure Pipelines by using building a pipeline to deploy a Xamarin app. You can deploy Android and iOS apps in the same or separate pipelines.

Prerequisites

Before you begin, you need:

- An Azure account with an active subscription. [Create an account for free](#).
- An active Azure DevOps organization. [Sign up for Azure Pipelines](#).

Get code

Fork this repo in GitHub:

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

This sample is the `FirstApp` sample from the <https://github.com/xamarin/xamarin-forms-samples> repository. For more information on getting started with Xamarin, see [Build your first Xamarin.Forms App](#).

Sign in to Azure Pipelines

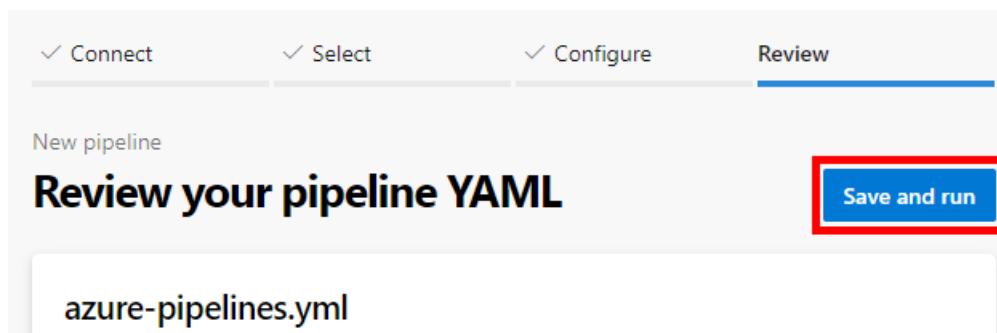
Sign in to [Azure Pipelines](#). After you sign in, your browser goes to <https://dev.azure.com/my-organization-name> and displays your Azure DevOps dashboard.

Within your selected organization, create a *project*. If you don't have any projects in your organization, you see a **Create a project to get started** screen. Otherwise, select the **Create Project** button in the upper-right corner of the dashboard.

Create the pipeline

1. Sign in to your Azure DevOps organization and navigate to your project.
2. Go to **Pipelines**, and then select **New Pipeline**.
3. Walk through the steps of the wizard by first selecting **GitHub** as the location of your source code.
4. You might be redirected to GitHub to sign in. If so, enter your GitHub credentials.
5. When the list of repositories appears, select your repository.
6. You might be redirected to GitHub to install the Azure Pipelines app. If so, select **Approve & install**.
7. When the **Configure** tab appears, select **Xamarin.Android** to build an Android project or **Xamarin.iOS** to build an iOS project. If you want to use the included sample `.yml` files, choose **Existing Azure Pipelines YAML file** and choose from one of the included sample pipelines (Android, iOS, or a combined pipeline that builds both).
8. When your new pipeline appears, take a look at the YAML to see what it does. When you're ready, select

Save and run.



9. If you created a new YAML file, you're prompted to commit a new `azure-pipelines.yml` file to your repository. After you're happy with the message, select **Save and run** again.

If you want to watch your pipeline in action, select the build job. You now have a working YAML pipeline (`azure-pipelines.yml`) in your repository that's ready for you to customize!

10. When you're ready to make changes to your pipeline, select it in the **Pipelines** page, and then **Edit** the `azure-pipelines.yml` file.
11. See the sections below to learn some of the more common ways to customize your pipeline.

Set up Xamarin tools

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/ecosystems/xamarin
pool:
  vmImage: 'macOS-10.15' # For Windows, use 'windows-2019'
```

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@1

- task: NuGetCommand@2
  inputs:
    restoreSolution: '**/*.sln'

- task: XamarinAndroid@1
  inputs:
    configFile: '**/*Droid*.csproj'
    outputDirectory: '$(outputDirectory)'
    configuration: '$(buildConfiguration)'
    msbuildVersionOption: '16.0'

```

Sign a Xamarin.Android app

See [Sign your mobile Android app during CI](#) for information about signing your app.

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: '**/*iOS.csproj'
    configuration: '$(buildConfiguration)'
    packageApp: false
    buildForSimulator: true

```

Sign and provision a Xamarin.iOS app - The PackageApp option

To generate a signed and provisioned Xamarin.iOS app .ipa package, set `packageApp` to `true` and make sure prior to this task you installed the right Apple Provisioning Profile and Apple Certificates that match your App Bundle ID into the agent running the job.

To fulfill these mandatory requisites, use the Microsoft Provided tasks for [installing an Apple Provisioning Profile](#) and [installing Apple Certificates](#).

```

- task: XamariniOS@2
  inputs:
    solutionFile: '**/*iOS.csproj'
    configuration: 'AppStore'
    packageApp: true

```

TIP

The Xamarin.iOS build task will only generate an .ipa package if the agent running the job has the [appropriate provisioning profile and Apple certificate installed](#). If you enable the packageApp option and the agent does not have the appropriate apple provisioning profile(.mobileprovision) and apple certificate(.p12) the build may report succeeded but there will be no .ipa generated.

For Microsoft Hosted agents the .ipa package is by default located under path:

```
{ios.csproj root}/bin/{Configuration}/{iPhone/iPhoneSimulator}/
```

You can configure the output path by adding an argument to the Xamarin.iOS task:

- [YAML](#)
- [Classic](#)

```
- task: XamariniOS@2
  inputs:
    solutionFile: '**/*iOS.csproj'
    configuration: 'AppStore'
    packageApp: true
    args: /p:IpaPackageDir="/Users/vsts/agent/2.153.2/work/1/a"
```

This example locates the .ipa in the Build Artifact Staging Directory ready to be pushed into Azure DevOps as an artifact to each build run. To push it into Azure DevOps simply add a [Publish Artifact task](#) to the end of your pipeline.

For more information about signing and provisioning your iOS app, see [Sign your mobile iOS app during CI](#).

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.

```

# Xamarin.Android and Xamarin.iOS
# Build a Xamarin.Android and Xamarin.iOS app.
# Add steps that test, sign, and distribute the app, save build artifacts, and more:
# https://docs.microsoft.com/azure/devops/pipelines/ecosystems/xamarin

jobs:

- job: Android
  pool:
    vmImage: 'windows-2019'

  variables:
    buildConfiguration: 'Release'
    outputDirectory: '$(build.binariesDirectory)/$(buildConfiguration)'

  steps:
    - task: NuGetToolInstaller@1

    - task: NuGetCommand@2
      inputs:
        restoreSolution: '**/*.sln'

    - task: XamarinAndroid@1
      inputs:
        projectFile: '**/*droid*.csproj'
        outputDirectory: '$(outputDirectory)'
        configuration: '$(buildConfiguration)'
        msbuildVersionOption: '16.0'

    - task: AndroidSigning@3
      inputs:
        apkSign: false
        zipAlign: false
        apkFiles: '$(outputDirectory)/*.apk'

    - task: PublishBuildArtifacts@1
      inputs:
        pathToPublish: '$(outputDirectory)'

- job: iOS
  pool:
    vmImage: 'macOS-10.15'

  steps:
    # To manually select a Xamarin SDK version on the Hosted macOS agent, enable this script with the SDK
    # version you want to target
    # https://go.microsoft.com/fwlink/?linkid=871629
    - script: sudo $AGENT_HOME DIRECTORY/scripts/select-xamarin-sdk.sh 5_4_1
      displayName: 'Select Xamarin SDK version'
      enabled: false

    - task: NuGetToolInstaller@1

    - task: NuGetCommand@2
      inputs:
        restoreSolution: '**/*.sln'

    - task: XamariniOS@2
      inputs:
        solutionFile: '**/*.sln'
        configuration: 'Release'
        buildForSimulator: true
        packageApp: false

```

Clean up resources

If you don't need the example code, delete your GitHub repository and Azure Pipelines project.

Next steps

[Learn more about using Xcode in pipelines](#)

[Learn more about using Android in pipelines](#)

Build, test, and deploy Xcode apps

2/26/2020 • 6 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

This guidance explains how to automatically build Xcode projects.

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/ecosystems/xcode
pool:
  vmImage: 'macOS-10.14'
```

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

An Xcode app must be signed and provisioned to run on a device or be published to the App Store. The signing and provisioning process needs access to your P12 signing certificate and one or more provisioning profiles. The [Install Apple Certificate](#) and [Install Apple Provisioning Profile](#) tasks make these available to Xcode during a build.

The following snippet installs an Apple P12 certificate and provisioning profile in the build agent's Keychain. Then, it builds, signs, and provisions the app with Xcode. Finally, the certificate and provisioning profile are automatically removed from the Keychain at the end of the build, regardless of whether the build succeeded or failed. For more details, see [Sign your mobile app during CI](#).

```

# The `certSecureFile` and `provProfileSecureFile` files are uploaded to the Azure Pipelines secure files
library where they are encrypted.
# The `P12Password` variable is set in the Azure Pipelines pipeline editor and marked 'secret' to be
encrypted.
steps:
- task: InstallAppleCertificate@2
  inputs:
    certSecureFile: 'chrisid_iOSDev_Nov2018.p12'
    certPwd: $(P12Password)

- task: InstallAppleProvisioningProfile@1
  inputs:
    provProfileSecureFile: '6ffac825-ed27-47d0-8134-95fcf37a666c.mobileprovision'

- task: Xcode@5
  inputs:
    actions: 'build'
    scheme: ''
    sdk: 'iphoneos'
    configuration: 'Release'
    xcWorkspacePath: '**/*.{xcodeproj/project.xcworkspace}'
    xcodeVersion: 'default' # Options: 8, 9, 10, default, specifyPath
    signingOption: 'default' # Options: nosign, default, manual, auto
    useXcpretty: 'false' # Makes it easier to diagnose build failures

```

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'
```

Carthage

If your project uses Carthage with a private Carthage repository, you can set up authentication by setting an environment variable named `GITHUB_ACCESS_TOKEN` with a value of a token that has access to the repository. Carthage will automatically detect and use this environment variable.

Do not add the secret token directly to your pipeline YAML. Instead, create a new pipeline variable with its lock enabled on the Variables pane to encrypt this value. See [secret variables](#).

Here is an example that uses a secret variable named `myGitHubAccessToken` for the value of the `GITHUB_ACCESS_TOKEN` environment variable.

```
- script: carthage update --platform iOS
  env:
    GITHUB_ACCESS_TOKEN: $(myGitHubAccessToken)
```

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.

[Sign up with App Center](#) first.

```

# App Center test
# Test app packages with Visual Studio App Center
- task: AppCenterTest@1
  inputs:
    appFile:
    #artifactsDirectory: '$(Build.ArtifactStagingDirectory)/AppCenterTest'
    #prepareTests: true # 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
    #uitestStorePath: # Optional
    #uitestStorePassword: # Optional
    #uitestKeyAlias: # Optional
    #uitestKeyPassword: # Optional
    #uitestToolsDirectory: # Optional
    #signInfo: # Optional
    #xcUITestBuildDirectory: # Optional
    #xcUITestIpaFile: # Optional
    #prepareOptions: # Optional
    #runTests: true # 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 Visual Studio 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
    #isMandatory: false # Optional
    #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.

See [limitations](#) of using this task with Apple two-factor authentication, since Apple authentication is region specific and fastlane session tokens expire quickly and must be recreated and reconfigured.

```

- 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
    appIdentifier: 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
    appIdentifier: 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)

Quickstart: Trigger an Azure Pipelines run from GitHub Actions

5/21/2021 • 2 minutes to read • [Edit Online](#)

Get started using [GitHub Actions](#) and Azure Pipelines together. GitHub Actions help you automate your software development workflows from within GitHub. You can deploy workflows in the same place where you store code and collaborate on pull requests and issues.

If you have both Azure Pipelines and GitHub Actions workflows, you might want to trigger a pipeline run from within a GitHub action. For example, you might have a specific set of Azure DevOps pipeline tasks that you want to trigger from your GitHub Actions workflow.

To accomplish this, you can trigger a pipeline run with the [Azure Pipelines Action](#).

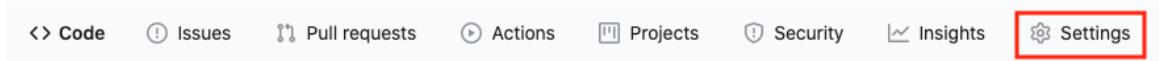
Prerequisites

- A working Azure pipeline. [Create your first pipeline](#).
- A GitHub account with a repository. [Join GitHub](#) and [create a repository](#).
- An Azure DevOps personal access token (PAT) to use with your GitHub action. [Create a PAT](#).

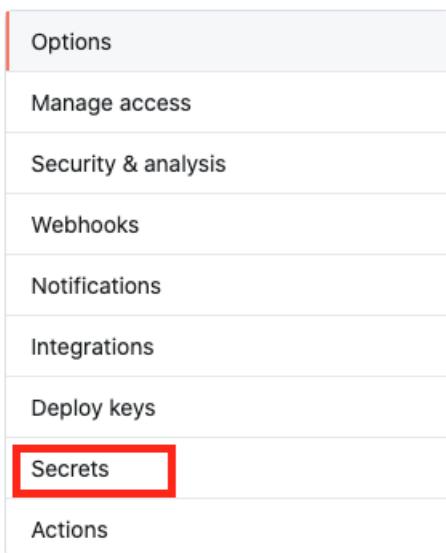
Authenticate with Azure Pipelines

You'll use a [personal access token](#) (PAT) to connect your GitHub account to Azure DevOps. You can generate a PAT from within Azure DevOps and then store it as a GitHub secret. Within your GitHub workflow, you'll reference the secret so that your GitHub action can authenticate with your Azure DevOps project.

1. Open your GitHub repository and go to **Settings**.



2. Select **Secrets** and then **New repository secret**.



3. Paste in your PAT and give it the name `AZURE_DEVOPS_TOKEN`.

4. Save by selecting **Add secret**.

Create a GitHub workflow that triggers a pipeline

[GitHub workflows](#) are a series of actions (like tasks in Azure Pipelines). You'll use the [Azure Pipelines Action](#) to trigger a pipeline run.

This example walks through creating a new workflow from within GitHub Actions and can be adapted to meet your needs. The relevant section for connecting to Azure Pipelines is the Azure Pipelines action.

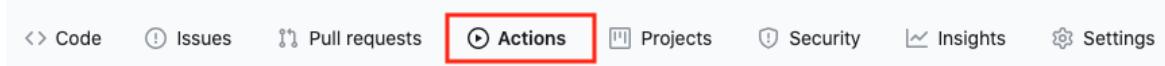
1. In your repository on GitHub, create a new YAML file in the `.github/workflows` directory.
2. Copy the following contents into your YAML file. Customize the `azure-devops-project-url` and `azure-pipeline-name` values. The `azure-devops-project-url` shouldn't have a trailing slash.

```
name: CI

# Run this workflow every time a commit is pushed to main or a pull request is opened against main
on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  build:
    name: Call Azure Pipeline
    runs-on: ubuntu-latest
    steps:
      - name: Azure Pipelines Action
        uses: Azure/pipelines@v1
        with:
          azure-devops-project-url: https://dev.azure.com/organization/project-name
          azure-pipeline-name: 'My Pipeline'
          azure-devops-token: ${{ secrets.AZURE_DEVOPS_TOKEN }}
```

3. Commit and push your workflow file.
4. The workflow will run every time a commit is pushed to main or a pull request is opened against main. To verify that your action ran, open your GitHub repository and select **Actions**.



5. Select the workflow title to see more information about the run. You should see a green check mark for the Azure Pipelines Action. Open the Action to see a direct link to the pipeline run.



Branch considerations

The pipeline your branch runs on depends on whether your pipeline is in the same repo as your GitHub workflow file.

- If the pipeline and the GitHub workflow are in different repositories, the triggered pipeline version in the branch specified by **Default branch for manual and scheduled builds** is run.
- If the pipeline and the GitHub workflow are in the same repository, the triggered pipeline version in the same branch as the triggering pipeline is run.

To configure the **Default branch for manual and scheduled builds** setting, see [Default branch for manual and scheduled builds setting](#).

Clean up resources

If you're not going to continue to use your GitHub workflow, [disable the workflow](#).

Next steps

[Deploy to Azure using GitHub Actions](#)

Build multiple branches

3/6/2021 • 8 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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 main 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 main branch and delete the topic branch.

- [YAML](#)
- [Classic](#)

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 `main` branch. This will cause any changes to `main` and `feature/*` branches to be automatically built.

```
trigger:  
- main  
- feature/*
```

YAML builds are not yet available on TFS.

Automatically build a change in topic branch

You're now ready for CI for both the main 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 `feature/`, e.g., `feature/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 main. Practicing CI for your topic branches helps to minimize risk when merging back to main.

Exclude or include tasks for builds based on the branch being built

The main 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 main branch during a build run. You can use a single build with multiple branches and skip or perform certain tasks based on conditions.

- [YAML](#)
- [Classic](#)

Edit the `azure-pipelines.yml` file in your `main` 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/main'))
```

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 main 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](#)
- [Classic](#)

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: `main` and `feature/*`:

```
pr:  
- main  
- feature/*
```

For more details, see [Triggers](#).

YAML builds are not yet available on TFS.

Azure Pipelines or TFS repository

1. Navigate to the **Repos** hub in Azure Repos or TFS.
2. Choose your **repository** and select **Branches**. Choose the **main branch**.
3. You will implement a branch policy to protect the main 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 main 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 main. If the build fails the merge is blocked.

Once the work is completed in the topic branch and merged to main, 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 are evaluated in order, applying the first matching policy to each build. The default rule at the bottom matches all builds.

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 **feature/*** 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

3/6/2021 • 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

- A GitHub account, where you can create a repository. If you don't have one, you can [create one for free](#).
- An Azure DevOps organization. If you don't have one, you can [create one for free](#). (An Azure DevOps organization is different from your GitHub organization. Give them the same name if you want alignment between them.)

If your team already has one, then make sure you're an administrator of the Azure DevOps project that you want to use.

- An ability to run pipelines on Microsoft-hosted agents. You can either purchase a [parallel job](#) or you can request a free tier. To request a free tier, follow the instructions in [this article](#). Please note that it may take us 2-3 business days to grant the free tier.

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 a pipeline

In the sample repo, there's no pipeline yet. You're going to add jobs that run on three platforms.

1. Go to your fork of the sample code on GitHub.
2. Choose 'Create new file'. Name the file `azure-pipelines.yml`, and give it the contents below.

```

# Build NodeJS Express app using Azure Pipelines
# https://docs.microsoft.com/azure/devops/pipelines/ecosystems/javascript?view=azure-devops
strategy:
  matrix:
    linux:
      imageName: 'ubuntu-16.04'
    mac:
      imageName: 'macos-10.14'
    windows:
      imageName: 'vs2017-win2016'

pool:
  vmImage: $(imageName)

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 jobs run at the same time in parallel.

Note: `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.

Connect Select Configure Review

New pipeline

Where is your code?

 Azure Repos Git 
Free private Git repositories, pull requests, and code search

 Bitbucket Cloud 
Hosted by Atlassian

 GitHub 
Home to the world's largest community of developers

 GitHub Enterprise Server 
The self-hosted version of GitHub Enterprise

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 main branch**.

9. The YAML file is pushed to your GitHub repository, and a new build is automatically started. Wait for the build to finish.

FAQ

Can I build my multi-platform pipeline on both self-hosted and Microsoft-hosted agents?

You can, you would need to specify both a `vmImage` and a `Pool` variable, like the following example. For the hosted agent, specify `Azure Pipelines` as the pool name, and for self-hosted agents, leave the `vmImage` blank. The blank `vmImage` for the self-hosted agent may result in some unusual entries in the logs but they won't affect the pipeline.

```
strategy:
matrix:
  microsofthosted:
    poolName: Azure Pipelines
    vmImage: ubuntu-latest

  selfhosted:
    poolName: FabrikamPool
    vmImage:

pool:
  name: $(poolName)
  vmImage: $(vmImage)

steps:
- checkout: none
- script: echo test
```

Next steps

You've just learned the basics of using multiple platforms with Azure Pipelines. From here, you can learn more about:

- [Jobs](#)
- [Cross-platform scripting](#)
- [Templates](#) to remove the duplication
- Building [Node.js](#) apps
- Building [.NET Core](#), [Go](#), [Java](#), or [Python](#) apps

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

Publish Pipeline Artifacts

6/29/2021 • 2 minutes to read • [Edit Online](#)

Azure Artifacts enable developers to store and manage their packages and control who they want to share it with. Pipeline Artifacts are generally generated after you build your application. The output can then deployed or consumed by another job in your pipeline.

Publish Artifacts

NOTE

Publish Artifacts is not supported in release pipelines. It is only supported in build pipelines, yaml pipelines, and multi-stage pipelines.

You can publish your Artifacts at any stage of your pipeline using YAML or the classic editor. If you want to publish your Artifacts manually, you can use the Azure CLI to run the `az pipelines runs artifact` command. You will not be billed for storing your Pipeline Artifacts or Pipeline caching.

- [YAML](#)
- [Classic](#)

```
steps:  
- task: PublishPipelineArtifact@1  
  inputs:  
    targetPath: '$(Pipeline.Workspace)'  
    artifactType: 'pipeline'  
    artifactName: 'drop'
```

- **targetPath**: Path to the folder or file you want to publish. (Required)
- **artifactType**: Artifacts publish location. (Required). Options: pipeline, filepath. Default value: pipeline.
- **artifactName**: Name of your Artifact. (Optional)

NOTE

The `PublishBuildArtifacts@1` task is deprecated, we recommend that you use the `PublishPipelineArtifact@1` task for faster performance.

Publish Artifacts from the command line

If you want to manually publish your Artifact, run the following command in an elevated command prompt:

```
az pipelines runs artifact upload --artifact-name your_artifact_name --path your_path_to_publish --run-id ''
```

View published Artifacts

When your pipeline run is completed you can view or download your published Artifact as follows

1. Select your pipeline run, and then select the **Summary** tab.

2. Select the published Artifact in the related section.

The screenshot shows the 'Summary' tab of an Azure Pipeline run. At the top, it displays the run was manually triggered by Rami Bououni. Below that, it shows the repository and version information: ramiMSFT/mslearn-tailspin-spacegame-web, build-pipeline, commit 8a520f2. It also shows the time started and elapsed (Today at 10:51 AM, 3m 15s), related work items (0), and tests and coverage (Get started). A red box highlights the '1 published' link under the tests and coverage section. Below this, there is a 'Jobs' section showing a single job named 'Job' which completed successfully in 3m 8s.

3. Expand the drop folder and find your Artifact.

The screenshot shows the 'Published' artifacts list. It lists two artifacts under the 'drop' folder: 'Tailspin.SpaceGame.Web.zip' (size 2 MB) under the 'Debug' subfolder, and another 'Tailspin.SpaceGame.Web.zip' (size 2 MB) under the 'Release' subfolder. Both artifacts are highlighted with red boxes.

4. Download your pipeline Artifact and explore its content.

Related articles

- [Releases in Azure Pipelines](#)
- [Multi-stage release pipeline](#)
- [Deploy from multiple branches](#)

Service containers

3/6/2021 • 4 minutes to read • [Edit Online](#)

Azure Pipelines

If your pipeline requires the support of one or more services, in many cases you'll want to create, connect to, and clean up each service on a per-job basis. For instance, a pipeline may run integration tests that require access to a database and a memory cache. The database and memory cache need to be freshly created for each job in the pipeline.

A container provides a simple and portable way to run a service that your pipeline depends on. A *service container* enables you to automatically create, network, and manage the lifecycle of your containerized service. Each service container is accessible by only the job that requires it. Service containers work with any kind of [job](#), but they're most commonly used with [container jobs](#).

Requirements

Service containers must define a `CMD` or `ENTRYPOINT`. The pipeline will `docker run` the provided container without additional arguments.

Azure Pipelines can run Linux or [Windows Containers](#). Use either hosted Ubuntu for Linux containers, or the Hosted Windows Container pool for Windows containers. (The Hosted macOS pool does not support running containers.)

- [YAML](#)
- [Classic](#)

Single container job

A simple example of using [container jobs](#):

```
resources:
  containers:
    - container: my_container
      image: buildpack-deps:focal
    - container: nginx
      image: nginx

  pool:
    vmImage: 'ubuntu-20.04'

  container: my_container
  services:
    nginx: nginx

  steps:
    - script: |
        curl nginx
  displayName: Show that nginx is running
```

This pipeline fetches the `nginx` and `buildpack-deps` containers from [Docker Hub](#) and then starts the containers. The containers are networked together so that they can reach each other by their `services` name.

From inside this job container, the `nginx` host names resolves to the correct services using Docker networking. All containers on the network automatically expose all ports to each other.

Single job

You can also use service containers without a job container. A simple example:

```
resources:
  containers:
    - container: nginx
      image: nginx
      ports:
        - 8080:80
      env:
        NGINX_PORT: 80
    - container: redis
      image: redis
      ports:
        - 6379

  pool:
    vmImage: 'ubuntu-18.04'

  services:
    nginx: nginx
    redis: redis

  steps:
    - script: |
        curl localhost:8080
        echo ${AGENT_SERVICES_REDIS_PORTS_6379}
```

This pipeline starts the latest `nginx` containers. Since the job is not running in a container, there's no automatic name resolution. This example shows how you can instead reach services by using `localhost`. In the above example we provide the port explicitly (for example, `8080:80`).

An alternative approach is to let a random port get assigned dynamically at runtime. You can then access these dynamic ports by using [variables](#). In a Bash script, you can access a variable by using the process environment. These variables take the form: `agent.services.<serviceName>.ports.<port>`. In the above example, `redis` is assigned a random available port on the host. The `agent.services.redis.ports.6379` variable contains the port number.

Multiple jobs

Service containers are also useful for running the same steps against multiple versions of the same service. In the following example, the same steps run against multiple versions of PostgreSQL.

```

resources:
  containers:
    - container: my_container
      image: ubuntu:18.04
    - container: pg12
      image: postgres:12
    - container: pg11
      image: postgres:11

  pool:
    vmImage: 'ubuntu-18.04'

  strategy:
    matrix:
      postgres12:
        postgresService: pg12
      postgres11:
        postgresService: pg11

  container: my_container

  services:
    postgres: ${ variables['postgresService'] }

  steps:
    - script: printenv

```

Ports

When specifying a container resource or an inline container, you can specify an array of `ports` to expose on the container.

```

resources:
  containers:
    - container: my_service
      image: my_service:latest
      ports:
        - 8080:80
        - 5432

  services:
    redis:
      image: redis
      ports:
        - 6379/tcp

```

Specifying `ports` is not required if your job is running in a container because containers on the same Docker network automatically expose all ports to each other by default.

If your job is running on the host, then `ports` are required to access the service. A port takes the form `<hostPort>:<containerPort>` or just `<containerPort>`, with an optional `/<protocol>` at the end, for example `6379/tcp` to expose `tcp` over port `6379`, bound to a random port on the host machine.

For ports bound to a random port on the host machine, the pipeline creates a variable of the form `agent.services.<serviceName>.ports.<port>` so that it can be accessed by the job. For example, `agent.services.redis.ports.6379` resolves to the randomly assigned port on the host machine.

Volumes

Volumes are useful for sharing data between services, or for persisting data between multiple runs of a job.

You can specify volume mounts as an array of `volumes`. Volumes can either be named Docker volumes, anonymous Docker volumes, or bind mounts on the host.

```
services:  
  my_service:  
    image: myservice:latest  
    volumes:  
      - mydockervolume:/data/dir  
      - /data/dir  
      - /src/dir:/dst/dir
```

Volumes take the form `<source>:<destinationPath>`, where `<source>` can be a named volume or an absolute path on the host machine, and `<destinationPath>` is an absolute path in the container.

NOTE

If you use our hosted pools, then your volumes will not be persisted between jobs because the host machine is cleaned up after the job is completed.

Other options

Service containers share the same container resources as container jobs. This means that you can use the same [additional options](#).

Healthcheck

Optionally, if any service container specifies a [HEALTHCHECK](#), the agent waits until the container is healthy before running the job.

Run cross-platform scripts

11/2/2020 • 4 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | 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](#)
- [Classic](#)

```
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](#)
- [Classic](#)

```
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 or pwsh

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 or [Windows Subsystem for Linux](#) 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](#)
- [Classic](#)

```
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
```

PowerShell Core (`pwsh`) is also an option. It requires each agent to have PowerShell Core installed.

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](#)
- [Classic](#)

```
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 | grep inet | 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 pipeline

3/6/2021 • 5 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015 | [Previous versions \(XAML builds\)](#)

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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.

Add a PowerShell script

- [YAML](#)
- [Classic](#)

The syntax for including PowerShell Core is slightly different from the syntax for Windows PowerShell. You can run Windows PowerShell on a [Windows build agent](#). PowerShell Core runs on any platform.

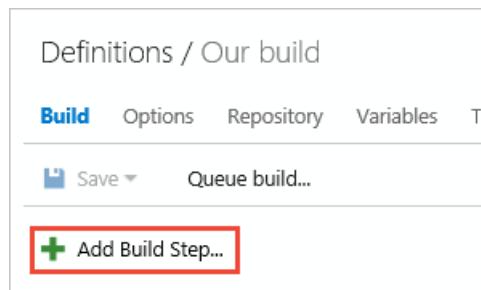
1. Push your PowerShell script to your repo.
2. Add a `pwsh` or `powershell` step. The `pwsh` keyword is a shortcut for the [PowerShell task](#) for PowerShell Core. The `powershell` keyword is another shortcut for the [PowerShell task](#) but Windows PowerShell and will only work on a Windows agent.

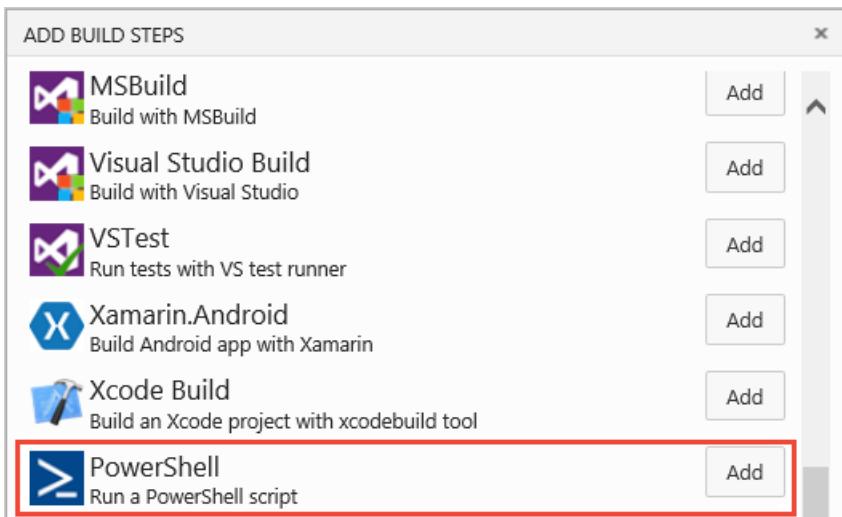
```
# for PowerShell Core
steps:
- pwsh: ./my-script.ps1

# for Windows PowerShell
steps:
- powershell: .\my-script.ps1
```

You can run Windows 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 PowerShell script: version assemblies

Here's an example script to version your assemblies. For the script to run successfully, you'll need to update your build number to use a format with four periods (example:

```
$(BuildDefinitionName)_{(Year:yyyy).{Month}.{DayOfMonth}}{Rev:.r} ). Build number can also be referred to as run number.
```

- [YAML](#)
- [Classic](#)

You can [customize your build number](#) within a YAML pipeline with the `name` property.

```
name: $(BuildDefinitionName)_{(Year:yyyy).{Month}.{DayOfMonth}}{Rev:.r}

pool:
  vmImage: windows-latest

steps:
- script: echo $(Build.BuildNumber) //output updated 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:BUILDSOURCESDIRECTORY -and $Env:BUILDNUMBER))
{
  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 '$Env:BUILD_BUILDDATE - For example, enter something like:'
Write-Host '$Env:BUILD_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:BUILD_BUILDDATE)
{
    Write-Error ("BUILD_BUILDDATE environment variable is missing.")
    exit 1
}
Write-Verbose "BUILD_BUILDDATE: $Env:BUILD_BUILDDATE"

# Get and validate the version data
$VersionData = [regex]::matches($Env:BUILD_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."
}

```

Example PowerShell script: access REST API

In this example, you'll use the `$env:SYSTEM_ACESSTOKEN` variable to access the [Azure Pipelines REST API](#).

- [YAML](#)
- [Classic](#)

You can use `$env:SYSTEM_ACESSTOKEN` in your script in a YAML pipeline to access the OAuth token.

```
- task: PowerShell@2
  inputs:
    targetType: 'inline'
    script: |
      $url =
      "$($env:SYSTEM_TEAMFOUNDATIONCOLLECTIONURI)$env:SYSTEM_TEAMPROJECTID/_apis/build/definitions/$($env:SYSTEM_DEFINITIONID)?api-version=5.0"
      Write-Host "URL: $url"
      $pipeline = Invoke-RestMethod -Uri $url -Headers @{
        Authorization = "Bearer $env:SYSTEM_ACESSTOKEN"
      }
      Write-Host "Pipeline = $($pipeline | ConvertTo-Json -Depth 100)"
  env:
    SYSTEM_ACESSTOKEN: $(System.AccessToken)
```

FAQ

What variables are available for me to use in my scripts?

You can use [predefined variables](#) in your scripts. For more information on available variables and how to use them, see [Use predefined variables](#).

How do I set a variable so that it can be read by subsequent scripts and tasks?

To learn more about defining build variables in a script, see [Define and modify your build variables in a script](#).

To learn more about defining release variables in a script, see [Define and modify your release variables in a script](#).

Which branch of the script does the build run?

The build will use the active branch of your code. If your pipeline run uses the `main` branch, your script will also use the `main` branch.

What kinds of parameters can I use?

You can use named parameters. Other kinds of parameters, such as switch parameters, aren't supported. You'll see errors if you try to use switch parameters.

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

3/31/2021 • 5 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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 main.

Git is available on [Microsoft-hosted agents](#) and on [on-premises agents](#).

Enable scripts to run Git commands

NOTE

Before you begin, be sure your account's default identity is set with:

```
git config --global user.email "you@example.com"  
git config --global user.name "Your Name"
```

Grant version control permissions to the build service

1. Go to the project settings page for your organization at [Organization Settings > General > Projects](#).



2. Select the project you want to edit.

<input type="checkbox"/>	Name ↑	Description
<input type="checkbox"/>	P	ContosoAds

3. Within **Project Settings**, select **Repositories**. Select the repository you want to run Git commands on.

4. Select **Permissions** to edit your repository permissions.

All Repositories

Repositories Settings Policies Permissions

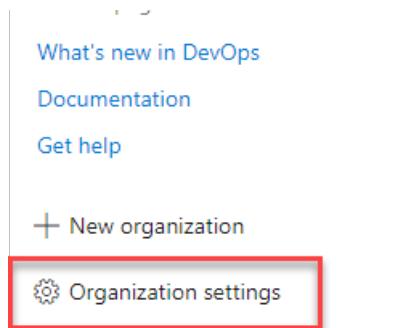
ContosoAds

5. Search for **Project Collection Build Service**. Choose the identity **Project Collection Build Service ({your organization})**. By default, this identity can read from the repo but cannot push any changes back to it. Grant permissions needed for the Git commands you want to run. Typically you'll want to grant:

- **Create branch:** Allow
- **Contribute:** Allow
- **Read:** Allow
- **Create tag:** Allow

Go to the Version Control control panel tab

- Azure Repos: https://dev.azure.com/{your-organization}/{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:

The screenshot shows the 'Organization Settings' page under 'Projects'. The left sidebar has sections for 'General' (Overview, Projects, Users, Global notifications, Usage, Extensions), 'Repos' (Repositories, Policies), and 'Active Directory'. The 'Projects' section shows two projects: 'webappcoretest' (Scrum, Online) and 'Work Item Test' (Agile SSH, Online). The 'Repositories' link in the 'Repos' section is 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**. By default, this identity can read from the repo but cannot push any changes back to it.

The screenshot shows the 'Security for all Git repositories' settings in VSTS. On the left, there's a sidebar with 'Repositories' and a 'New repository' button. Under 'Git repositories', several projects are listed: 'OurProject', 'Portal-Client', 'Portal-Server', 'Store-Client', and 'Store-Server'. The main area is titled 'Security for all Git repositories' with 'Inheritance' dropdowns. A search bar is at the top. Below it, under 'VSTS Groups', 'Project Collection Administrators' is selected. A list of users follows, with 'Project Collection Build Service (Application...)' highlighted by a red box.

Grant permissions needed for the Git commands you want to run. Typically you'll want to grant:

- **Create branch:** Allow
- **Contribute:** Allow
- **Read:** Allow
- **Create tag:** 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](#)
- [Classic](#)

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](#)
- [Classic](#)

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 main

You want a CI build to merge to main 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/main (
    ECHO Building main branch so no merge is needed.
    EXIT
)
SET sourceBranch=origin/%BUILD_SOURCEBRANCH:refs/heads/=%
ECHO GIT CHECKOUT MAIN
git checkout main
ECHO GIT STATUS
git status
ECHO GIT MERGE
git merge %sourceBranch% -m "Merge to main"
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: <input type="text" value="merge.bat"/>

FAQ

Can I run Git commands if my remote repo is in GitHub or another Git service such as Bitbucket Cloud?

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. Here are examples:

- `git commit -m "This is a commit message ***NO_CI***"`
- `git merge origin/features/hello-world -m "Merge to main ***NO_CI***"`

Add `[skip ci]` to your commit message or description. Here are examples:

- `git commit -m "This is a commit message [skip ci]"`
- `git merge origin/features/hello-world -m "Merge to main [skip ci]"`

You can also use any of the variations below. This is supported for commits to Azure Repos Git, Bitbucket Cloud, GitHub, and GitHub Enterprise Server.

- `[skip ci]` or `[ci skip]`
- `skip-checks: true` or `skip-checks:true`

- `[skip azurepipelines]` or `[azurepipelines skip]`
- `[skip azpipelines]` or `[azpipelines skip]`
- `[skip azp]` or `[azp skip]`
- `***NO_CI***`

How does enabling scripts to run Git commands affect how the build pipeline gets build sources?

When you set `system.usegit` 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.

I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

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](#).

My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

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](#).

Pipeline caching

6/23/2021 • 14 minutes to read • [Edit Online](#)

Pipeline caching can help reduce build time by allowing the outputs or downloaded dependencies from one run to be reused in later runs, thereby reducing or avoiding the cost to recreate or redownload the same files again. Caching is especially useful in scenarios where the same dependencies are downloaded over and over at the start of each run. This is often a time consuming process involving hundreds or thousands of network calls.

Caching can be effective at improving build time provided the time to restore and save the cache is less than the time to produce the output again from scratch. Because of this, caching may not be effective in all scenarios and may actually have a negative impact on build time.

Caching is currently supported in CI and deployment jobs, but not classic release jobs.

When to use artifacts versus caching

Pipeline caching and [pipeline artifacts](#) perform similar functions but are designed for different scenarios and should not be used interchangeably. In general:

- **Use pipeline artifacts** when you need to take specific files produced in one job and share them with other jobs (and these other jobs will likely fail without them).
- **Use pipeline caching** when you want to improve build time by reusing files from previous runs (and not having these files will not impact the job's ability to run).

NOTE

Caching is currently free, and caches are stored in Azure blob storage.

Use Cache task

Caching is added to a pipeline using the `Cache` pipeline task. This task works like any other task and is added to the `steps` section of a job.

When a cache step is encountered during a run, the task will restore the cache based on the provided inputs. If no cache is found, the step completes and the next step in the job is run. After all steps in the job have run and assuming a successful job status, a special "save cache" step is run for each "restore cache" step that was not skipped. This step is responsible for saving the cache.

NOTE

Caches are immutable, meaning that once a cache is created, its contents cannot be changed.

Configure Cache task

The `Cache` task has two required inputs: `key` and `path`.

Path input

`path` the path of the folder to cache. Can be an absolute or a relative path. Relative paths are resolved against `$(System.DefaultWorkingDirectory)`. You can use predefined variables however wildcards are not supported.

Key input

`key` should be set to the identifier for the cache you want to restore or save. Keys are composed of a

combination of string values, file paths, or file patterns, where each segment is separated by a `|` character.

- **Strings:**

fixed value (like the name of the cache or a tool name) or taken from an environment variable (like the current OS or current job name)

- **File paths:**

path to a specific file whose contents will be hashed. This file must exist at the time the task is run. Keep in mind that *any* key segment that "looks like a file path" will be treated like a file path. In particular, this includes segments containing a `.`. This could result in the task failing when this "file" does not exist.

TIP

To avoid a path-like string segment from being treated like a file path, wrap it with double quotes, for example:

```
"my.key" | $(Agent.OS) | key.file
```

- **File patterns:**

comma-separated list of glob-style wildcard pattern that must match at least one file. For example:

- `**/yarn.lock` : all yarn.lock files under the sources directory
- `*/asset.json, !bin/**` : all asset.json files located in a directory under the sources directory, except under the bin directory

The contents of any file identified by a file path or file pattern is hashed to produce a dynamic cache key. This is useful when your project has file(s) that uniquely identify what is being cached. For example, files like

`package-lock.json`, `yarn.lock`, `Gemfile.lock`, or `Pipfile.lock` are commonly referenced in a cache key since they all represent a unique set of dependencies.

Relative file paths or file patterns are resolved against `$(System.DefaultWorkingDirectory)`.

Example:

Here is an example showing how to cache dependencies installed by Yarn:

```
variables:  
  YARN_CACHE_FOLDER: $(Pipeline.Workspace)/.yarn  
  
steps:  
- task: Cache@2  
  inputs:  
    key: '"yarn" | "$(Agent.OS)" | yarn.lock'  
    restoreKeys:  
      - yarn | "$(Agent.OS)"  
    path: $(YARN_CACHE_FOLDER)  
    displayName: Cache Yarn packages  
  
- script: yarn --frozen-lockfile
```

In this example, the cache key contains three parts: a static string ("yarn"), the OS the job is running on since this cache is unique per operating system, and the hash of the `yarn.lock` file that uniquely identifies the set of dependencies in the cache.

On the first run after the task is added, the cache step will report a "cache miss" since the cache identified by this key does not exist. After the last step, a cache will be created from the files in `$(Pipeline.Workspace)/.yarn` and uploaded. On the next run, the cache step will report a "cache hit" and the contents of the cache will be downloaded and restored.

Restore keys

`restoreKeys` can be used if one wants to query against multiple exact keys or key prefixes. This is used to fallback to another key in the case that a `key` does not yield a hit. A restore key will search for a key by prefix and yield the latest created cache entry as a result. This is useful if the pipeline is unable to find an exact match but wants to use a partial cache hit instead. To insert multiple restore keys, simply delimit them by using a new line to indicate the restore key (see the example for more details). The order of which restore keys will be tried against will be from top to bottom.

Required software on self-hosted agent

ARCHIVE SOFTWARE / PLATFORM	WINDOWS	LINUX	MAC
GNU Tar	Required	Required	No
BSD Tar	No	No	Required
7-Zip	Recommended	No	No

The above executables need to be in a folder listed in the PATH environment variable. Please note that the hosted agents come with the software included, this is only applicable for self-hosted agents.

Example:

Here is an example on how to use restore keys by Yarn:

```
variables:
  YARN_CACHE_FOLDER: $(Pipeline.Workspace)/.yarn

steps:
- task: Cache@2
  inputs:
    key: '"yarn" | "$(Agent.OS)" | yarn.lock'
    restoreKeys:
      - yarn | "$(Agent.OS)"
    path: $(YARN_CACHE_FOLDER)
    displayName: Cache Yarn packages

- script: yarn --frozen-lockfile
```

In this example, the cache task will attempt to find if the key exists in the cache. If the key does not exist in the cache, it will try to use the first restore key `yarn | $(Agent.os)`. This will attempt to search for all keys that either exactly match that key or has that key as a prefix. A prefix hit can happen if there was a different `yarn.lock` hash segment. For example, if the following key `yarn | $(Agent.os) | old-yarn.lock` was in the cache where the old `yarn.lock` yielded a different hash than `yarn.lock`, the restore key will yield a partial hit. If there is a miss on the first restore key, it will then use the next restore key `yarn` which will try to find any key that starts with `yarn`. For prefix hits, the result will yield the most recently created cache key as the result.

NOTE

A pipeline can have one or more caching task(s). There is no limit on the caching storage capacity, and jobs and tasks from the same pipeline can access and share the same cache.

Cache isolation and security

To ensure isolation between caches from different pipelines and different branches, every cache belongs to a logical container called a scope. Scopes provide a security boundary that ensures a job from one pipeline cannot

access the caches from a different pipeline, and a job building a PR has read access to the caches for the PR's target branch (for the same pipeline), but cannot write (create) caches in the target branch's scope.

When a cache step is encountered during a run, the cache identified by the key is requested from the server. The server then looks for a cache with this key from the scopes visible to the job, and returns the cache (if available). On cache save (at the end of the job), a cache is written to the scope representing the pipeline and branch. See below for more details.

CI, manual, and scheduled runs

SCOPE	READ	WRITE
Source branch	Yes	Yes
main branch	Yes	No

Pull request runs

SCOPE	READ	WRITE
Source branch	Yes	No
Target branch	Yes	No
Intermediate branch (such as refs/pull/1/merge)	Yes	Yes
main branch	Yes	No

Pull request fork runs

BRANCH	READ	WRITE
Target branch	Yes	No
Intermediate branch (such as refs/pull/1/merge)	Yes	Yes
main branch	Yes	No

TIP

Because caches are already scoped to a project, pipeline, and branch, there is no need to include any project, pipeline, or branch identifiers in the cache key.

Conditioning on cache restoration

In some scenarios, the successful restoration of the cache should cause a different set of steps to be run. For example, a step that installs dependencies can be skipped if the cache was restored. This is possible using the `cacheHitVar` task input. Setting this input to the name of an environment variable will cause the variable to be set to `true` when there is a cache hit, `inexact` on a restore key cache hit, otherwise it will be set to `false`. This variable can then be referenced in a [step condition](#) or from within a script.

In the following example, the `install-deps.sh` step is skipped when the cache is restored:

```

steps:
- task: Cache@2
  inputs:
    key: mykey | mylockfile
    restoreKeys: mykey
    path: $(Pipeline.Workspace)/mycache
    cacheHitVar: CACHE_RESTORED

- script: install-deps.sh
  condition: ne(variables.CACHE_RESTORED, 'true')

- script: build.sh

```

Bundler

For Ruby projects using Bundler, override the `BUNDLE_PATH` environment variable used by Bundler to set the [path](#) [Bundler](#) will look for Gems in.

Example:

```

variables:
  BUNDLE_PATH: $(Pipeline.Workspace)/.bundle

steps:
- task: Cache@2
  inputs:
    key: 'gems | "$(Agent.OS)" | my.gemspec'
    restoreKeys: |
      gems | "$(Agent.OS)"
      gems
    path: $(BUNDLE_PATH)
    displayName: Cache gems

- script: bundle install

```

Ccache (C/C++)

[Ccache](#) is a compiler cache for C/C++. To use Ccache in your pipeline make sure `ccache` is installed, and optionally added to your `PATH` (see [Ccache run modes](#)). Set the `CCACHE_DIR` environment variable to a path under `$(Pipeline.Workspace)` and cache this directory.

Example:

```

variables:
  CCACHE_DIR: $(Pipeline.Workspace)/ccache

steps:
- bash: |
  sudo apt-get install ccache -y
  echo "##vso[task.prependpath]/usr/lib/ccache"
  displayName: Install ccache and update PATH to use linked versions of gcc, cc, etc

- task: Cache@2
  inputs:
    key: 'ccache | "$(Agent.OS)"'
    path: $(CCACHE_DIR)
    restoreKeys: |
      ccache | "$(Agent.OS)"
  displayName: ccache

```

See [Cache configuration settings](#) for more details.

Docker images

Caching Docker images dramatically reduces the time it takes to run your pipeline.

```
pool:
  vmImage: 'Ubuntu-16.04'

steps:
- task: Cache@2
  inputs:
    key: 'docker | cache'
    path: $(Pipeline.Workspace)/docker
    cacheHitVar: DOCKER_CACHE_HIT
    displayName: Cache Docker images
- script: |
    docker load -i $(Pipeline.Workspace)/docker/cache.tar
    displayName: Restore Docker image
    condition: and(not(canceled()), eq(variables.DOCKER_CACHE_HIT, 'true'))
- script: |
    mkdir -p $(Pipeline.Workspace)/docker
    docker save -o $(Pipeline.Workspace)/docker/cache.tar cache
    displayName: Save Docker image
    condition: and(not(canceled()), or(failed(), ne(variables.DOCKER_CACHE_HIT, 'true')))
```

Replace `caching-docker.yml` with your own pipeline YAML file.

Golang

For Golang projects, you can specify the packages to be downloaded in the `go.mod` file. If your `GOCACHE` variable isn't already set, set it to where you want the cache to be downloaded.

Example:

```
variables:
  GO_CACHE_DIR: $(Pipeline.Workspace)/.cache/go-build/

steps:
- task: Cache@2
  inputs:
    key: 'go | "$(Agent.OS)" | go.mod'
    restoreKeys: |
      go | "$(Agent.OS)"
    path: $(GO_CACHE_DIR)
    displayName: Cache GO packages
```

Gradle

Using Gradle's [built-in caching support](#) can have a significant impact on build time. To enable the build cache, set the `GRADLE_USER_HOME` environment variable to a path under `$(Pipeline.Workspace)` and either run your build with `--build-cache` or add `org.gradle.caching=true` to your `gradle.properties` file.

Example:

```

variables:
  GRADLE_USER_HOME: ${Pipeline.Workspace}/.gradle

steps:
- task: Cache@2
  inputs:
    key: 'gradle | "${Agent.OS}" | **/build.gradle.kts' # Swap build.gradle.kts for build.gradle when using Groovy
    restoreKeys: |
      gradle | "${Agent.OS}"
      gradle
    path: ${GRADLE_USER_HOME}
  displayName: Configure gradle caching

- task: Gradle@2
  inputs:
    gradleWrapperFile: 'gradlew'
    tasks: 'build'
    options: '--build-cache'
  displayName: Build

- script: |
  # stop the Gradle daemon to ensure no files are left open (impacting the save cache operation later)
  ./gradlew --stop
  displayName: Build

```

- **restoreKeys:** The fallback keys if the primary key fails (Optional)

NOTE

Caches are immutable, once a cache with a particular key is created for a specific scope (branch), the cache cannot be updated. This means that if the key is a fixed value, all subsequent builds for the same branch will not be able to update the cache even if the cache's contents have changed. If you want to use a fixed key value, you must use the `restoreKeys` argument as a fallback option.

Maven

Maven has a local repository where it stores downloads and built artifacts. To enable, set the `maven.repo.local` option to a path under `$(Pipeline.Workspace)` and cache this folder.

Example:

```

variables:
  MAVEN_CACHE_FOLDER: ${Pipeline.Workspace}/.m2/repository
  MAVEN_OPTS: '-Dmaven.repo.local=${MAVEN_CACHE_FOLDER}'

steps:
- task: Cache@2
  inputs:
    key: 'maven | "${Agent.OS}" | **/pom.xml'
    restoreKeys: |
      maven | "${Agent.OS}"
      maven
    path: ${MAVEN_CACHE_FOLDER}
  displayName: Cache Maven local repo

- script: mvn install -B -e

```

If you are using a [Maven task](#), make sure to also pass the `MAVEN_OPTS` variable because it gets overwritten otherwise:

```
- task: Maven@3
  inputs:
    mavenPomFile: 'pom.xml'
    mavenOptions: '-Xmx3072m $(MAVEN_OPTS)'
```

.NET/NuGet

If you use `PackageReferences` to manage NuGet dependencies directly within your project file and have `packages.lock.json` file(s), you can enable caching by setting the `NUGET_PACKAGES` environment variable to a path under `$(UserProfile)` and caching this directory.

Example:

```
variables:
  NUGET_PACKAGES: $(Pipeline.Workspace)/.nuget/packages

steps:
- task: Cache@2
  inputs:
    key: 'nuget | "$(Agent.OS)" | $(Build.SourcesDirectory)/packages.lock.json'
    restoreKeys: |
      nuget | "$(Agent.OS)"
    path: $(NUGET_PACKAGES)
  displayName: Cache NuGet packages
```

TIP

Environment variables always override any settings in the NuGet.Config file. If your pipeline failed with the error: `Information, There is a cache miss.`, you must create a pipeline variable for `NUGET_PACKAGES` to point to the new local path on the agent (exp `d:\a\1`). Your pipeline should pick up the changes then and continue the task successfully.

In the above example, the `$(Build.SourcesDirectory)` points to your project's generated lock file. See [Package reference in project files](#) for more details on how to enable lock file creation.

Node.js/npm

There are different ways to enable caching in a Node.js project, but the recommended way is to cache npm's [shared cache directory](#). This directory is managed by npm and contains a cached version of all downloaded modules. During install, npm checks this directory first (by default) for modules that can reduce or eliminate network calls to the public npm registry or to a private registry.

Because the default path to npm's shared cache directory is [not the same across all platforms](#), it is recommended to override the `npm_config_cache` environment variable to a path under `$(Pipeline.Workspace)`. This also ensures the cache is accessible from container and non-container jobs.

Example:

```

variables:
  npm_config_cache: $(Pipeline.Workspace)/.npm

steps:
- task: Cache@2
  inputs:
    key: 'npm | $(Agent.OS) | package-lock.json'
    restoreKeys: |
      npm | $(Agent.OS)"
    path: $(npm_config_cache)
  displayName: Cache npm

- script: npm ci

```

If your project does not have a `package-lock.json` file, reference the `package.json` file in the cache key input instead.

TIP

Because `npm ci` deletes the `node_modules` folder to ensure that a consistent, repeatable set of modules is used, you should avoid caching `node_modules` when calling `npm ci`.

Node.js/Yarn

Like with npm, there are different ways to cache packages installed with Yarn. The recommended way is to cache Yarn's [shared cache folder](#). This directory is managed by Yarn and contains a cached version of all downloaded packages. During install, Yarn checks this directory first (by default) for modules, which can reduce or eliminate network calls to public or private registries.

Example:

```

variables:
  YARN_CACHE_FOLDER: $(Pipeline.Workspace)/.yarn

steps:
- task: Cache@2
  inputs:
    key: 'yarn | $(Agent.OS) | yarn.lock'
    restoreKeys: |
      yarn | $(Agent.OS)"
    path: $(YARN_CACHE_FOLDER)
  displayName: Cache Yarn packages

- script: yarn --frozen-lockfile

```

Python/pip

For Python projects that use pip or Poetry, override the `PIP_CACHE_DIR` environment variable. If you use Poetry, in the `key` field, replace `requirements.txt` with `poetry.lock`.

Example

```
variables:
  PIP_CACHE_DIR: $(Pipeline.Workspace)/.pip

steps:
- task: Cache@2
  inputs:
    key: 'python | "$(Agent.OS)" | requirements.txt'
    restoreKeys: |
      python | "$(Agent.OS)"
      python
    path: $(PIP_CACHE_DIR)
    displayName: Cache pip packages

- script: pip install -r requirements.txt
```

Python/Pipenv

For Python projects that use Pipenv, override the `PIPEENV_CACHE_DIR` environment variable.

Example

```
variables:
  PIPEENV_CACHE_DIR: $(Pipeline.Workspace)/.pipenv

steps:
- task: Cache@2
  inputs:
    key: 'python | "$(Agent.OS)" | Pipfile.lock'
    restoreKeys: |
      python | "$(Agent.OS)"
      python
    path: $(PIPEENV_CACHE_DIR)
    displayName: Cache pipenv packages

- script: pipenv install
```

Python/Anaconda

Set up your pipeline caching with Anaconda environments

Example

```

variables:
  CONDA_CACHE_DIR: $(Pipeline.Workspace)/.condarc

# Add conda to system path
steps:
- script: echo "##vso[task.prependpath]$CONDA/bin"
  displayName: Add conda to PATH

- task: Cache@2
  displayName: Use cached Anaconda environment
  inputs:
    key: 'conda | "$(Agent.OS)" | environment.yml'
    restoreKeys: |
      python | "$(Agent.OS)"
      python
    path: $(CONDA_CACHE_DIR)
    cacheHitVar: CONDA_CACHE_RESTORED

- script: conda env create --quiet --file environment.yml
  displayName: Create Anaconda environment
  condition: eq(variables.CONDA_CACHE_RESTORED, 'false')

```

PHP/Composer

For PHP projects using Composer, override the `COMPOSER_CACHE_DIR` environment variable used by Composer.

Example:

```

variables:
  COMPOSER_CACHE_DIR: $(Pipeline.Workspace)/.composer

steps:
- task: Cache@2
  inputs:
    key: 'composer | "$(Agent.OS)" | composer.lock'
    restoreKeys: |
      composer | "$(Agent.OS)"
      composer
    path: $(COMPOSER_CACHE_DIR)
    displayName: Cache composer

- script: composer install

```

Known issues and feedback

If you experience problems enabling caching for your project, first check the list of [pipeline caching issues](#) in the microsoft/azure-pipelines-tasks repo. If you don't see your issue listed, [create a new issue](#).

Q&A

Q: Can I clear a cache?

A: Clearing a cache is currently not supported. However you can add a string literal (such as `version2`) to your existing cache key to change the key in a way that avoids any hits on existing caches. For example, change the following cache key from this:

```
key: 'yarn | "$(Agent.OS)" | yarn.lock'
```

to this:

```
key: 'version2 | yarn | "$(Agent.OS)" | yarn.lock'
```

Q: When does a cache expire?

A: Caches expire after seven days of no activity.

Q: Is there a limit on the size of a cache?

A: There is no enforced limit on the size of individual caches or the total size of all caches in an organization.

Configure run or build numbers

5/17/2021 • 3 minutes to read • [Edit Online](#)

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

You can customize how your pipeline runs are numbered. The default value for run number is

`$(Date:yyyyMMdd).$(Rev:r)`.

- [YAML](#)
- [Classic](#)

In YAML, this property is called `name` and is at the root level of a pipeline. If not specified, your run is given a unique integer as its name. You can give runs much more useful names that are meaningful to your team. You can use a combination of tokens, variables, and underscore characters.

```
name: $(TeamProject)_$(Build.DefinitionName)_$(SourceBranchName)_$(Date:yyyyMMdd)$(Rev:.r)

steps:
- script: echo '$(Build.BuildNumber)' # outputs customized build number like project_def_master_20200828.1
```

YAML builds are not yet available on TFS.

Example

At the time a run is started:

- Project name: Fabrikam
- Pipeline name: CIBuild
- Branch: master
- Build ID/Run ID: 752
- Date: May 5, 2019.
- Time: 9:07:03 PM.
- One run completed earlier today.

If you specify this build number format:

```
$(TeamProject)_$(Build.DefinitionName)_$(SourceBranchName)_$(Date:yyyyMMdd)$(Rev:.r)
```

Then the second run on this day would be named: **Fabrikam_CIBuild_master_20190505.2**

Tokens

The following table shows how each token is resolved based on the previous example. You can use these tokens only to define a run number; they don't work anywhere else in your pipeline.

TOKEN	EXAMPLE REPLACEMENT VALUE
<code>\$(Build.DefinitionName)</code>	CIBuild Note: The pipeline name must not contain invalid or whitespace characters.
<code>\$(Build.BuildId)</code>	752 \$(Build.BuildId) is an internal immutable ID that is also referred to as the Run ID. It is unique across the organization.
<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 run will be 3, and so on.) Use \$(Rev:r) to ensure that every completed build has a unique name. When a build starts, 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. If you use a zero-padded Rev as part of a version numbering scheme, note that some pipeline tasks or popular tools, like NuGet packages, remove the leading zeros, which causes a version number mismatch in the artifacts that are produced.
<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

Variables

You can also use user-defined and predefined variables that have a scope of "All" in your number. For example, if you've defined `My.Variable`, you could specify the following 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](#).

Expressions

If you use an expression to set the build number, you can't use some tokens because their values aren't set at the time expressions are evaluated. These tokens include `$(Build.BuildId)`, `$(Build.BuildURL)`, and `$(Build.BuildNumber)`.

FAQ

How large can a run number be?

Runs may be up to 255 characters.

In what time zone are the build number time values expressed?

The time zone is UTC.

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.

How can you reference the run number variable within a script?

The run number variable can be called with `$(Build.BuildNumber)`. You can define a new variable that includes the run number or call the run number directly. In this example, `$(MyRunNumber)` is a new variable that includes the run number.

```
# Set MyRunNumber
variables:
  MyRunNumber: '1.0.0-CI+$(Build.BuildNumber)'

steps:
- script: echo $(MyRunNumber) # display MyRunNumber
- script: echo $(Build.BuildNumber) #display Run Number
```

Build options

6/24/2021 • 2 minutes to read • [Edit Online](#)

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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
System.Title	Build \$(Build.BuildNumber) failed
System.Reason	Build failure

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 in classic build pipelines if you want to enable your script to use the build pipeline OAuth token. This check box is located under the "additional settings" section after selecting the agent job in the pipeline.

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.

For more information, see [Understand job access tokens](#).

Build (run) number

This documentation has moved to [Build \(run\) number](#).

Pipeline Run Retention

6/16/2021 • 3 minutes to read • [Edit Online](#)

Retaining a pipeline run for longer than the configured [project settings](#) is handled by the creation of **retention leases**. Temporary retention leases are often created by automatic processes and more permanent leases by manipulating the UI or when Release Management retains artifacts, but they can also be manipulated through the [REST API](#). Here are some examples of tasks that you can add to your yaml pipeline that will cause a run to retain itself.

Prerequisites

By default, members of the Contributors, Build Admins, Project Admins, and Release Admins groups can manage retention policies.

Example: Override a short project-level retention window

In this example, the project is configured to delete pipeline runs after only thirty days.

Settings

Retention policy

Days to keep artifacts, symbols and attachments

20

Days to keep runs

30

Days to keep pull request runs

1

Number of recent runs to retain per pipeline ⓘ

0

[Learn more about run retention](#)

If a pipeline in this project is important and runs should be retained for longer than thirty days, this task ensures the run will be valid for two years by [adding a new retention lease](#).

```
- task: PowerShell@2
  condition: and(succeeded(), not(canceled()))
  name: RetainOnSuccess
  displayName: Retain on Success
  inputs:
    failOnStderr: true
    targetType: 'inline'
    script: |
      $contentType = "application/json";
      $headers = @{ Authorization = 'Bearer $(System.AccessToken)' };
      $rawRequest = @{ daysValid = 365 * 2; definitionId = 1; ownerId = 'User:$(Build.RequestedForId)';
      protectPipeline = $false; runId = $(Build.BuildId) };
      $request = ConvertTo-Json @($rawRequest);
      $uri = "$($System.CollectionUri)$(System.TeamProject)/_apis/build/retention/leases?api-version=6.0-preview.1";
      Invoke-RestMethod -uri $uri -method POST -Headers $headers -ContentType $contentType -Body $request;
```

Question: can a pipeline be retained for less than the configured project values?

No, leases do not work in the reverse. If a project is configured to retain for two years, pipeline runs will not be cleaned up by the system for two years. To delete these runs earlier, manually delete them or use the equivalent REST API.

Example: Only runs on branches named `releases/*` should be retained for a long time

This is similar to above, only the condition needs to change:

```
- task: PowerShell@2
  condition: and(succeeded(), not(canceled()), startsWith(variables['Build.SourceBranchName'], 'releases/'))
  name: RetainReleaseBuildOnSuccess
  displayName: Retain Release Build on Success
  inputs:
    failOnStderr: true
    targetType: 'inline'
    script: |
      $contentType = "application/json";
      $headers = @{ Authorization = 'Bearer $(System.AccessToken)' };
      $rawRequest = @{ daysValid = 365 * 2; definitionId = $(System.DefinitionId); ownerId =
'User:$(Build.RequestedForId)'; protectPipeline = $false; runId = $(Build.BuildId) };
      $request = ConvertTo-Json @($rawRequest);
      $uri = "$(System.CollectionUri)$(System.TeamProject)/_apis/build/retention/leases?api-version=6.0-
preview.1";
      Invoke-RestMethod -uri $uri -method POST -Headers $headers -ContentType $contentType -Body $request;
```

Example: Updating the retention window for a multi-stage pipeline based on stage success

Consider a two-stage pipeline that first runs a build and then a release. When successful, the `Build` stage will retain the run for three days, but the project administrator wants a successful `Release` stage to extend the lease to one year.

The `Build` stage can retain the pipeline as in the above examples, but with one addition: by saving the new lease's `Id` in an output variable, the lease can be updated later when the release stage runs.

```
- task: PowerShell@2
  condition: and(succeeded(), not(canceled()))
  name: RetainOnSuccess
  displayName: Retain on Success
  inputs:
    failOnStderr: true
    targetType: 'inline'
    script: |
      $contentType = "application/json";
      $headers = @{ Authorization = 'Bearer $(System.AccessToken)' };
      $rawRequest = @{ daysValid = 3; definitionId = $(System.DefinitionId); ownerId =
'User:$(Build.RequestedForId)'; protectPipeline = $false; runId = $(Build.BuildId) };
      $request = ConvertTo-Json @($rawRequest);
      $uri = "$(System.CollectionUri)$(System.TeamProject)/_apis/build/retention/leases?api-version=6.0-
preview.1";
      $newLease = Invoke-RestMethod -uri $uri -method POST -Headers $headers -ContentType $contentType -Body
$request;
      $newLeaseId = $newLease.Value[0].LeaseId
      echo "##vso[task.setvariable variable=newLeaseId;isOutput=true]$newLeaseId";
```

To [update a retention lease](#) requires a different REST API call.

```

- stage: Release
dependsOn: Build
jobs:
- job: default
variables:
- name: NewLeaseId
  value: ${ stageDependencies.Build.default.outputs['RetainOnSuccess.newLeaseId']}
steps:
- task: PowerShell@2
  condition: and(succeeded(), not(canceled()))
  name: RetainOnSuccess
  displayName: Retain on Success
  inputs:
    failOnStderr: true
    targetType: 'inline'
    script: |
      $contentType = "application/json";
      $headers = @{ Authorization = 'Bearer $(System.AccessToken)' };
      $rawRequest = @{
        daysValid = 365;
        leaseId = $newLeaseId;
        ownerId = 'User:$(Build.RequestedForId)'
      };
      $request = ConvertTo-Json @($rawRequest);
      $uri = "$($System.CollectionUri)$(System.TeamProject)/_apis/build/retention/leases?api-version=6.0-preview.2";
      Invoke-RestMethod -uri $uri -method PATCH -Headers $headers -ContentType $contentType -Body
      $request;

```

Next steps

With these examples, you learned how to use custom pipeline tasks to manage run retention.

You learned how to:

- Override a short retention window
- Override retention for runs on specific branches
- Update a retention lease when a run should be retained even longer

About pipeline tests

11/2/2020 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

This article describes commonly used terms used in pipeline [test report](#) and [test analytics](#).

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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 15 possible outcomes for a test result: Aborted, Blocked, Error, Failed, Inconclusive, In progress, 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.- 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

- See our [troubleshooting](#) page
- Get advice on [Stack Overflow](#), and get support via the [Developer Community](#)

Run tests in parallel using the Visual Studio Test task

3/6/2021 • 9 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

NOTE

For TFS, this topic applies to only TFS 2017 Update 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

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

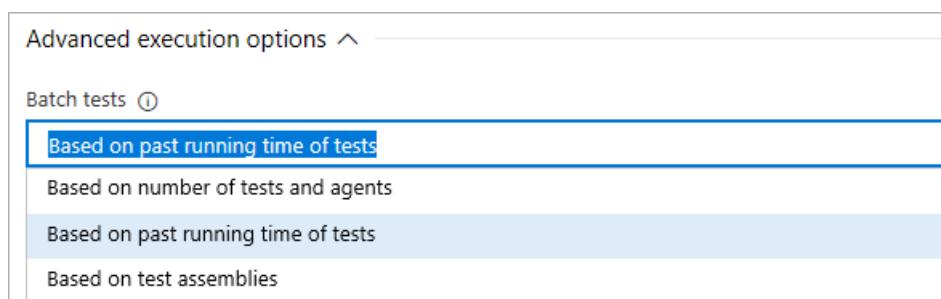
Pre-requisite

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 to 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 classic build pipelines

If you have a large test suite or long-running integration tests to run in your classic 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).

The screenshot shows the Azure DevOps Pipeline editor. On the left, a list of pipeline tasks is visible, including 'Get sources', 'Build job uses 1 agent' (selected), 'Use NuGet 4.4.1', 'NuGet restore', 'Build solution PartsUnlimited.sln', 'Publish symbols path', 'Copy Files to: \$(build.artifactstagingdirectory)', and 'Publish Artifact: drop'. The 'Build job uses 1 agent' task has a 'Run on agent' option selected. On the right, a detailed configuration pane for 'Agent job' is open. It includes fields for 'Display name' (set to 'Build job uses 1 agent'), 'Agent selection', 'Agent pool' (set to '<inherit from pipeline>'), 'Demands' (with entries for 'msbuild' and 'visualstudio'), 'Execution plan', and 'Parallelism' (set to 'None').

2. Run tests in parallel using multiple agents:

- Add an agent job

The screenshot shows the Pipeline editor with the 'Tasks' tab selected. A context menu is open over the 'Build job uses 1 agent' task, with the 'Add an agent job' option highlighted by a red box. Other options in the menu include 'Add an agentless job' and 'Learn more about jobs'.

- 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

Display name *
Parallel test job uses multiple agents

Agent selection

Agent pool | Manage
<inherit from pipeline>

Demands

Name	Condition
vstest	exists

+ Add

Execution plan

Parallelism

None Multi-configuration Multi-agent

Number of agents *

3

Continue on error

Timeout *

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 interface. On the left, a list of tasks is displayed for a 'Pipeline' build pipeline. The tasks include:

- 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, detailed configuration for the 'Download Build Artifacts' task is shown. The 'Artifact name' field is set to 'drop'. A red box highlights this field.

- Add the **Visual Studio Test** task and configure it to use the required [slicing strategy](#).

Setting up jobs for parallel testing in YAML pipelines

Specify the `parallel` strategy in the `job` and indicate how many jobs should be dispatched. You can specify as many as 99 agents to scale up testing for large test suites.

```
jobs:
- job: ParallelTesting
  strategy:
    parallel: 2
```

For more information, see [YAML schema - Job](#).

Run tests in parallel in classic 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).

The screenshot shows the 'Tasks' tab of a pipeline named 'QA'. The pipeline contains several tasks:

- 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 configuration details:

- Display name ***: Deploy app job - 1 agent.
- Agent selection**: Agent pool: Hosted VS2017.
- Demands**: Name: (empty).
- Execution plan**: Parallelism: None.
- Timeout ***: 0.

2. Run tests in parallel using multiple agents:

- Add an agent job

The screenshot shows the 'Tasks' tab of a pipeline named 'QA'. A context menu is open over the '+' button, with the 'Add an agent job' option highlighted.

Stage name: Add an agent job.

Other options in the menu:

- 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 'Tasks' tab in the Azure DevOps Pipeline interface. A parallel test job named 'Parallel test job using multiple agents - Selenium tests' is selected. In the configuration pane, the 'Execution plan' section is highlighted with a red border. It shows 'Parallelism' set to 'Multi-agent' with 3 agents assigned. Other settings include 'Display name' as 'Parallel test job using multiple agents - Selenium tests', 'Agent pool' as 'Hosted VS2017', and 'Demands' for 'vstest'.

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

- See our [troubleshooting](#) page
- Get advice on [Stack Overflow](#), and get support via the [Developer Community](#)

Run tests in parallel for any test runner

11/2/2020 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019

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.

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.

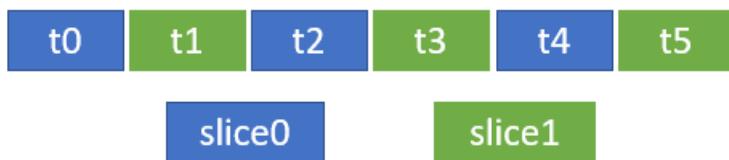
The step that runs the tests in a job needs to know which test slice should be run. The variables

`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 .NET Core sample uses `--list-tests` and `--filter` parameters of `dotnet test` to slice the tests. The tests are run using NUnit. Test results created by `DotNetCoreCLI@2` test task are then published to the server. Import (into Azure Repos or Azure DevOps Server) or fork (into GitHub) this repo:

```
https://github.com/idubnori/ParallelTestingSample-dotnet-core
```

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 Azure DevOps Server) 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 Azure DevOps Server) 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

- See our [troubleshooting](#) page
- Get advice on [Stack Overflow](#), and get support via the [Developer Community](#)

Speed up testing by using Test Impact Analysis (TIA)

11/2/2020 • 6 minutes to read • [Edit Online](#)

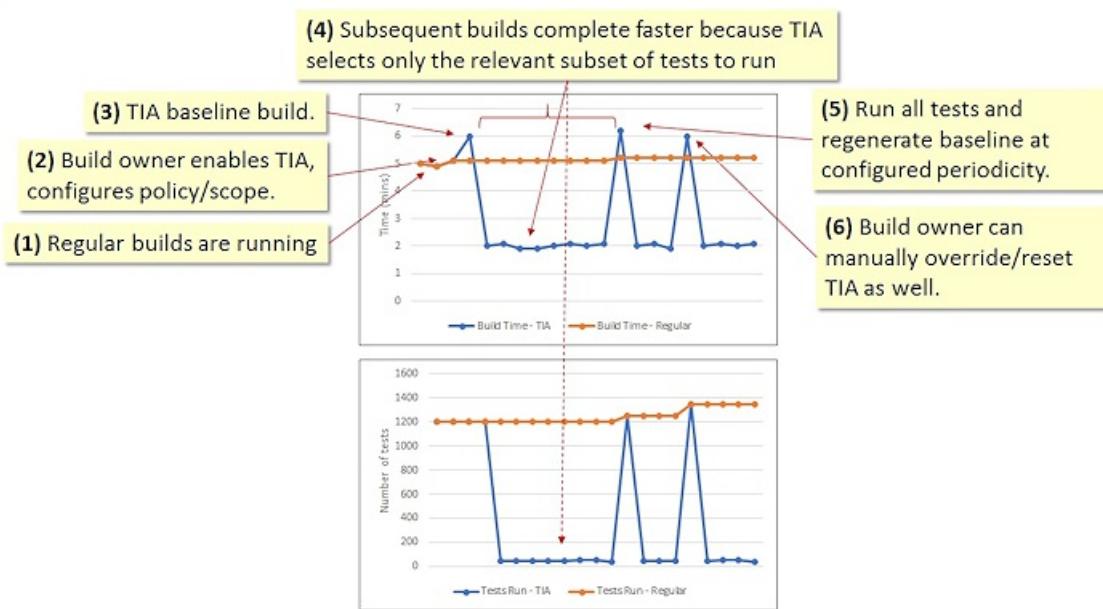
Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015 | Visual Studio 2017 | Visual Studio 2015

NOTE

Applies only to TFS 2017 Update 1 and later, and Visual Studio 2015 Update 3 and later.

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

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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, Other 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 ⓘ

[X Remove](#)

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 'Summary' tab of an Azure Pipelines build. At the top, there's a green circle with a checkmark. Below it, the project name 'AspNetWebAppSample-Azure Web App for A' is displayed along with the branch 'master' and commit hash '975e708 : Add test project'. There are buttons for 'Logs', 'Summary' (which is selected), 'Code coverage*', 'Tests', 'Artifacts', 'Release', 'Edit', and 'Queue'. A link 'Add a tag' is also present. The main area is titled 'Progression' and shows a timeline with two steps: 'Build artifacts published' (step 1) and 'Tests succeeded' (step 2). Step 2 is highlighted with a red border. It shows '100% passed' with a message '1 Run(s) Completed (1 Passed, 0 Failed, 0 Not impacted, 0 Others)'. Below this, there's a summary card with the following data:

2 Total tests +2		2 ● Passed 0 ● Failed 0 ● Others	100% Pass percentage ↑ 100%	1s 6 Run duration ↑ +1s 6
---------------------	--	--	--------------------------------	------------------------------

The screenshot shows the 'Tests' tab of an Azure Pipelines build. At the top, there's a green circle with a checkmark. Below it, the project name 'AspNetWebAppSample-Azure Web App for A' is displayed along with the branch 'master' and commit hash '975e708 : Add test project'. There are buttons for 'Logs', 'Summary' (selected), 'Code coverage*', 'Tests' (selected), 'Artifacts', 'Release', 'Edit', and 'Queue'. A link 'Add a tag' is also present. The main area is titled 'Summary' and shows a message '1 Run(s) Completed (1 Passed, 0 Failed, 0 Not impacted, 0 Others)'. Below this, there's a summary card with the following data:

2 Total tests +2		2 ● Passed 0 ● Failed 0 ● Others	100% Pass percentage ↑ 100%	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

- See our [troubleshooting](#) page
- Get advice on [Stack Overflow](#), and get support via the [Developer Community](#)

Manage flaky tests

11/2/2020 • 3 minutes to read • [Edit Online](#)

Azure Pipelines

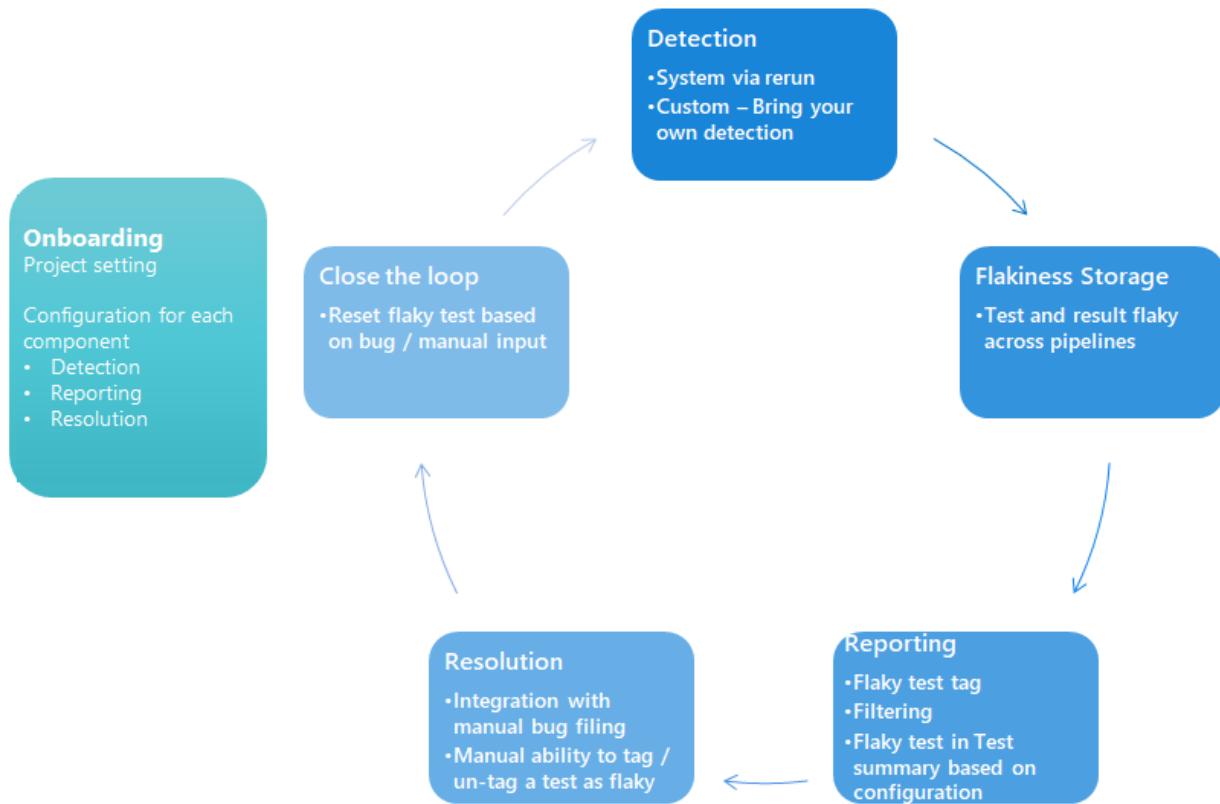
Productivity for developers relies on the ability of tests to find real problems with the code under development or update in a timely and reliable fashion. Flaky tests present a barrier to finding real problems, since the failures often don't relate to the changes being tested. A flaky test is a test that provides different outcomes, such as pass or fail, even when there are no changes in the source code or execution environment. Flaky tests also impact the quality of shipped code.

NOTE

This feature is only available on Azure DevOps Services. Typically, new features are introduced in the cloud service first, and then made available on-premises in the next major version or update of Azure DevOps Server. To learn more, see [Azure DevOps Feature Timeline](#).

The goal of bringing flaky test management in-product is to reduce developer pain caused by flaky tests and cater to the whole workflow. Flaky test management provides the following benefits.

- **Detection** - Auto detection of flaky test with rerun or extensibility to plug in your own custom detection method
- **Management of flakiness** - Once a test is marked as flaky, the data is available for all pipelines for that branch
- **Report on flaky tests** - Ability to choose if you want to prevent build failures caused by flaky tests, or use the flaky tag only for troubleshooting
- **Resolution** - Manual bug-creation or manual marking and unmarking test as flaky based on your analysis
- **Close the loop** - Reset flaky test as a result of bug resolution / manual input



Enable flaky test management

To configure flaky test management, choose **Project settings**, and select **Test management** in the Pipelines section.

Slide the On/Off button to On.

Project Settings

Fabrikam Fiber

General

Overview

Teams

Permissions

Notifications

Service hooks

Dashboards

Boards

Project configuration

Team configuration

GitHub connections

Repos

Repositories

Pipelines

Agent pools

Parallel jobs

Settings

Test management

Release retention

Service connections

Test management

Flaky test detection

On

Enable this feature to detect flaky tests using system or custom detection and configure flaky test options. [Learn more](#)

Select detection type

System detection

Use Azure Pipelines detection for flaky tests. This option is best suited if you use Pipeline or VSTest rerun capability.

Custom detection

Integrate your flaky detection system with Azure Pipelines.

Select pipelines to enable flaky test detection

All

Specific pipeline(s)

Flaky test options

Flaky tests included in test pass percentage

This option decides flaky test inclusion in test pass percentage. Uncheck to prevent pipeline failures due to flaky tests.

Allow users to manually mark/unmark flaky tests

This option allows all users in your account to manually mark or unmark tests as flaky or unfaky.

The default setting for all projects is to use flaky tests for troubleshooting.

Flaky test detection

Flaky test management supports system and custom detection.

- **System detection:** The in-product flaky detection uses test rerun data. The detection is via **VSTest task** rerunning of failed tests capability or retry of stage in the pipeline. You can select specific pipelines in the project for which you would like to detect flaky tests.

NOTE

Once a test is marked as flaky, the data is available for all pipelines for that branch to assist with troubleshooting in every pipeline.

- **Custom detection:** You can integrate your own flaky detection mechanism with Azure Pipelines and use the reporting capability. With custom detection, you need to update the test results metadata for flaky tests. For details, see [Test Results, Result Meta Data - Update REST API](#).

Project Settings

Fabrikam Fiber

- General
- Overview
- Teams
- Permissions
- Notifications
- Service hooks
- Dashboards
- Boards
- Project configuration
- Team configuration
- GitHub connections
- Repos
- Repositories
- Pipelines
- Agent pools
- Parallel jobs
- Settings
- Test management**
- Release retention

Test management

Flaky test detection

On

Enable this feature to detect flaky tests using system or custom detection and configure flaky test options. [Learn more](#)

Select detection type

System detection

 Use Azure Pipelines detection for flaky tests. This option is best suited if you use Pipeline or VSTest rerun capability.

Custom detection

 Integrate your flaky detection system with Azure Pipelines.

Flaky test options

Flaky tests included in test pass percentage

This option decides flaky test inclusion in test pass percentage. Uncheck to prevent pipeline failures due to flaky tests.

Allow users to manually mark/unmark flaky tests

This option allows all users in your account to manually mark or unmark tests as flaky or unflaky.

Identify new test cases

Track new test cases introduced in a pipeline run

This option allows to log new test cases that have been introduced for the first time in the system

Flaky test options

The **Flaky test options** specify how flaky tests are available in test reporting as well as resolution capabilities, as described in the following sections.

Flaky test management and reporting

On the Test management page under **Flaky test options**, you can set options for how flaky tests are included in the Test Summary report. Flaky test data for both passed and failed test is available in [Test results](#). The **Flaky** tag helps you identify flaky tests. By default, flaky tests are included in the Test Summary. However, if you want to ensure flaky test failures don't fail your pipeline, you can choose to not include them in your test summary and suppress the test failure. This option ensures flaky tests (both passed and failed) are removed from the pass percentage and shown in [Tests not reported](#), as shown in the following screenshot.

#AzureDevOps_merge_20190628.24 Increased retry for Spn creation PR AzureDevOps.L1 3244 targets releases/M154 Retained by release

Summary Tests Accessibility* Scans

Summary 1 Run(s) Completed (0 Passed, 1 Failed) 851 unique failing tests in the last 14 days

26,592 | 26,592 Passed 0 Failed 0 Others 100% Pass percentage 3m 1s Run duration ↑ +3m 1s 5 Tests not reported

Bug Link UnFlaky Test run Column Options Filter by test or run name Tags Test file Owner Outcome: Aborted (+1) X

Test	Duration	Failing since	Failing build	Tags
✗ L0 Run - AzureDevOps (2/26597)	0:14:13.715			
<input checked="" type="checkbox"/> ✗ PolicyListActionCreatorTests Initialize failed to load PAQs New Flaky	0:00:00.030	15m ago	Current build	
✗ PolicyListActionCreatorTests Initialize loads policies New	0:00:00.027	15m ago	Current build	

NOTE

The Test summary report is updated only for [Visual Studio Test task](#) and [Publish Test Results task](#). You may need to add a custom script to suppress flaky test failure for other scenarios.

Tests marked as flaky

You can mark or unmark a test as flaky based on analysis or context, by choosing **Flaky** (or **UnFlaky**, depending on whether the test is already marked as flaky.)

26,592 | 26,592 Passed 0 Failed 0 Others

Bug Link Flaky Filter by test or run name Test

✗ L0 Run - AzureDevOps (2/26597)
<input checked="" type="checkbox"/> ✗ PolicyListActionCreatorTests Initialize failed to load PAQs
✗ PolicyListActionCreatorTests Initialize loads policies New

When a test is marked flaky or unflaky in a pipeline, no changes are made in the current pipeline. Only on future executions of that test is the changed flaky setting evaluated. Tests marked as flaky have the *Marked flaky* tag in the user interface.

tests (+1) **66.66%** 4s 510ms 6

w Pass percentage Run duration ⓘ Tests not reported
sting

Mark test as flaky

This will mark the test as flaky for all subsequent pipeline runs from the branch and will not change the current test pass percentage.

Test name TestRunParameterOverrideParam1Value
Outcome ✘ Failed 25m ago on TFSBUILD74
Branch refs/heads/master
Test File unittestproject1.dll

Cancel OK

0:00:00.004	4h ago
0:00:00.003	4h ago

Help and support

- See our [troubleshooting](#) page
- Get advice on [Stack Overflow](#), and get support via the [Developer Community](#)

Related articles

- [Review test results](#)
- [Visual Studio Test task](#)
- [Publish Test Results task](#)
- [Test Results, Result Meta Data - Update REST API](#)

UI testing considerations

11/2/2020 • 6 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

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

Applies only to TFS 2017 Update 1 and later.

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Prerequisites

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.

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 tasks: 'Download', 'Build Ag...', and 'Deploym...'. The right window is titled 'Add artifact' and shows the configuration for the 'Build Agent' task. It includes fields for 'Account Name', 'Personal Access Token', 'Agent Name', 'Agent Name Suffix', 'Agent Pool', and 'Enable Autologon'. A red box highlights the 'Enable Autologon' dropdown, which has options: 'false', 'true', 'false', and 'INT AUTHORITY\NETWORKSERVICE'. At the bottom of the right window are 'Account Password', 'Use a saved secret', 'Type a value', and an 'Install Drive Letter' field.

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.`

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. This is because these test result formats do not have a formal definition for attachments in the results schema. You can use one of the below approaches to publish test attachments instead.

- If you are running tests in the build (CI) pipeline, you can use the [Copy and Publish Build Artifacts](#) task to publish any additional files created in your tests. These will appear in the [Artifacts](#) page of your build summary.
- Use the REST APIs to publish the necessary attachments. Code samples can be found in [this GitHub repository](#).

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

- See our [troubleshooting](#) page
- Get advice on [Stack Overflow](#), and get support via the [Developer Community](#)

UI test with Selenium

3/6/2021 • 5 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015 | Visual Studio 2017 | Visual Studio 2015

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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. Special considerations that apply when running UI tests are discussed in [UI testing considerations](#).

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).

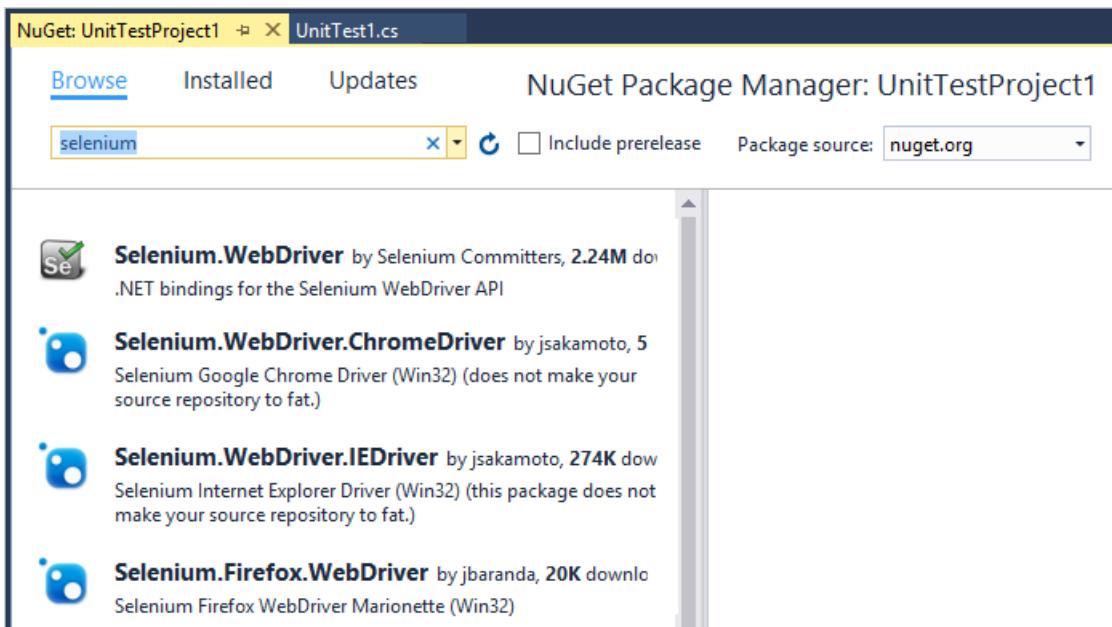
For more information about Selenium browser automation, see:

- [Selenium](#)
- [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 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 continuous deployment (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 paths to the folders containing these drivers can be obtained from the environment variables named `IEDriver` (Internet Explorer), `ChromeDriver` (Google Chrome), and `GeckoDriver` (Firefox). The drivers are **not** pre-installed on other agents such as Linux, Ubuntu, and macOS agents. Also see [UI testing considerations](#).

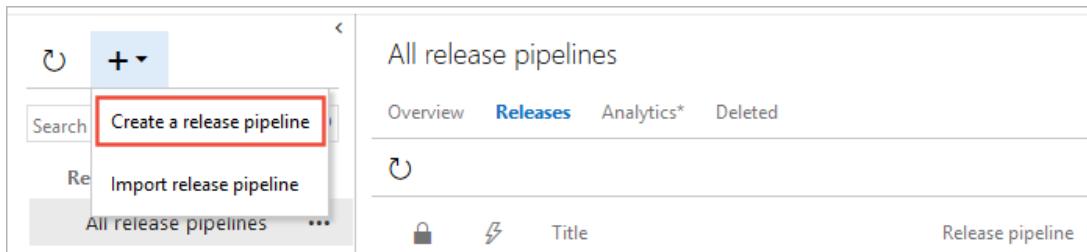
- When using a **self-hosted agent** that you deploy on your target servers, agents must be configured to run interactively with auto-logon enabled. See [Build and release agents](#) and [UI testing considerations](#).

Include the test in a release

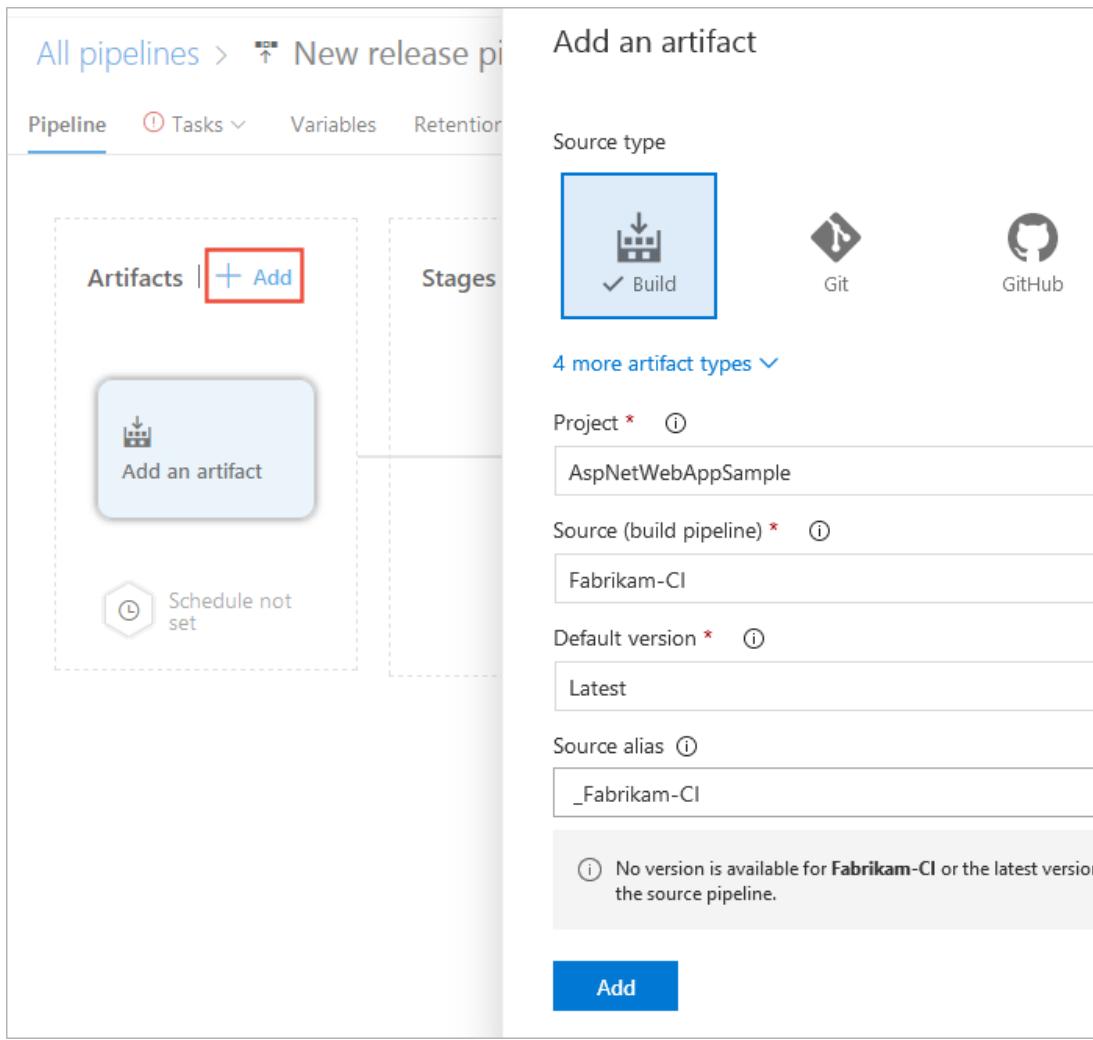
NOTE: This example uses the **Visual Studio Test Platform Installer** task and the latest version of the **Visual Studio Test** task. These tasks are not available in TFS 2015 or TFS 2017. To run Selenium tests in these versions of TFS, you must use the [Visual Studio Test Agent Deployment](#) and [Run Functional Tests](#) tasks instead.

- If you don't have an existing release pipeline that deploys your web app:

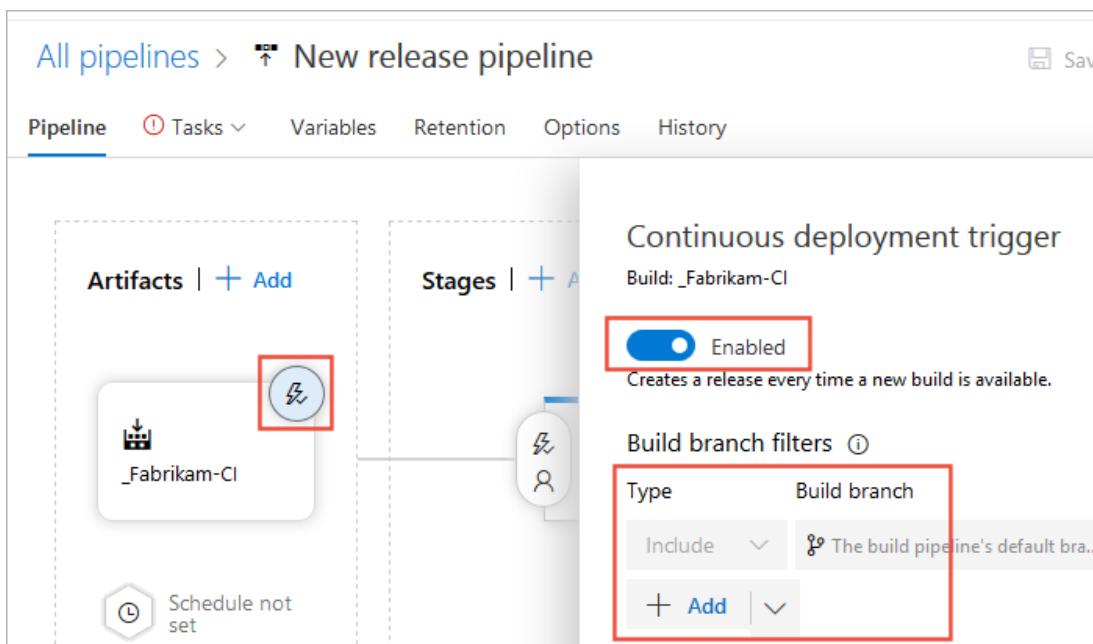
- Open the **Releases** page in the **Azure Pipelines** section in Azure DevOps or the **Build & Release** hub in TFS (see [Web portal navigation](#)) and choose the **+** icon, then choose **Create release pipeline**.



- 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

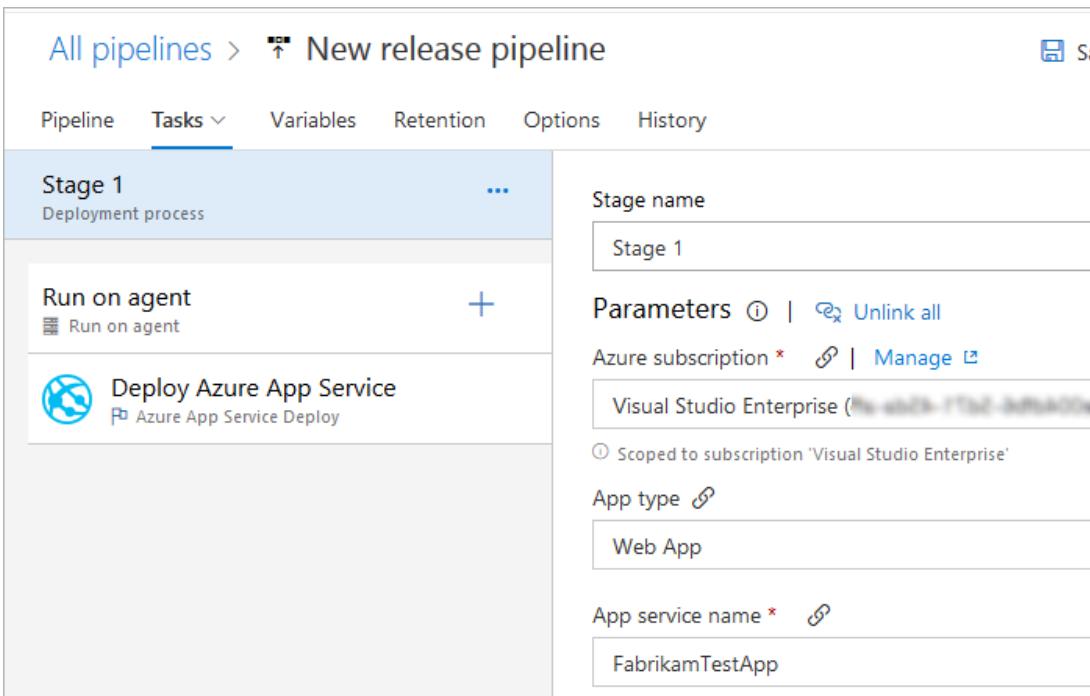
Stage name: Stage 1

Parameters: Azure subscription * | Manage | Unlink all

Azure subscription: Visual Studio Enterprise (Visual Studio Enterprise)

App type: Web App

App service name: FabrikamTestApp



2. 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

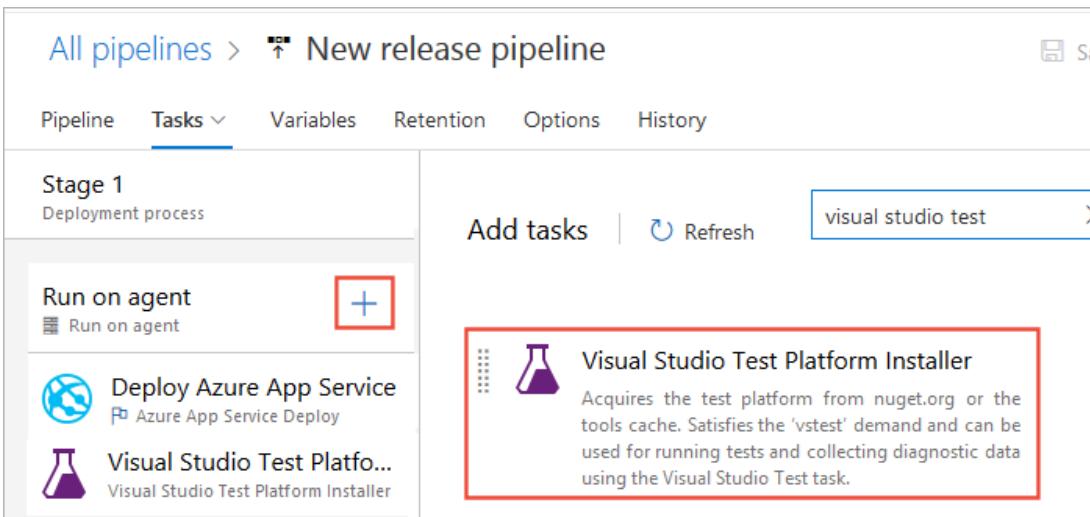
Run on agent +
Run on agent

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.

3. 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

Add tasks Refresh visual studio test

Run on agent +

Deploy Azure App Service

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 +

Deploy Azure App Service

Visual Studio Test Platfo...

VsTest - testAssemblies

Execution options

Select test platform using

Version Specific location

Test platform version ⓘ

Installed by Tools Installer

Settings file ⓘ

Override test run parameters ⓘ

How do I pass parameters to my test code from a build 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

Requirements traceability

11/2/2020 • 6 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - 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

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Agile teams running automated tests

Agile teams have characteristics including, but not limited to the following

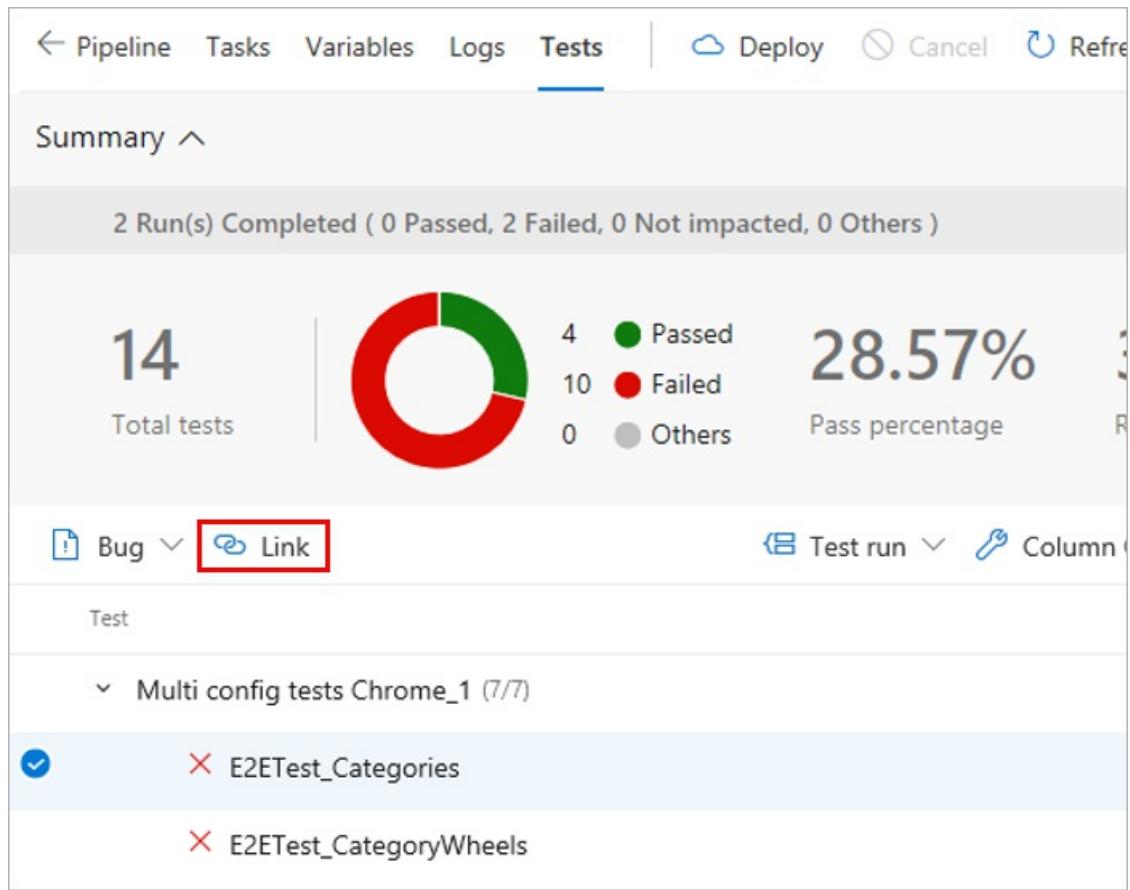
- Faster release cycles
- Continuous testing in a pipeline
- Negligible manual testing footprint; limited to exploratory testing
- High degree of automation

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	[Redacted]
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.

Bug	Link	Requirement	Column Options
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.

The screenshot shows a 'widgets' section on the left and an 'Add Widget' dialog on the right. The 'Add Widget' dialog has a search bar labeled 'Requirement' and a list of available widgets. One widget, 'Requirements quality', is highlighted with a blue border. Its details are displayed: 'Shows the quality of requirement(s) associated to automated tests.' and 'By Microsoft'. There is also a 'Learn More' link.

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**

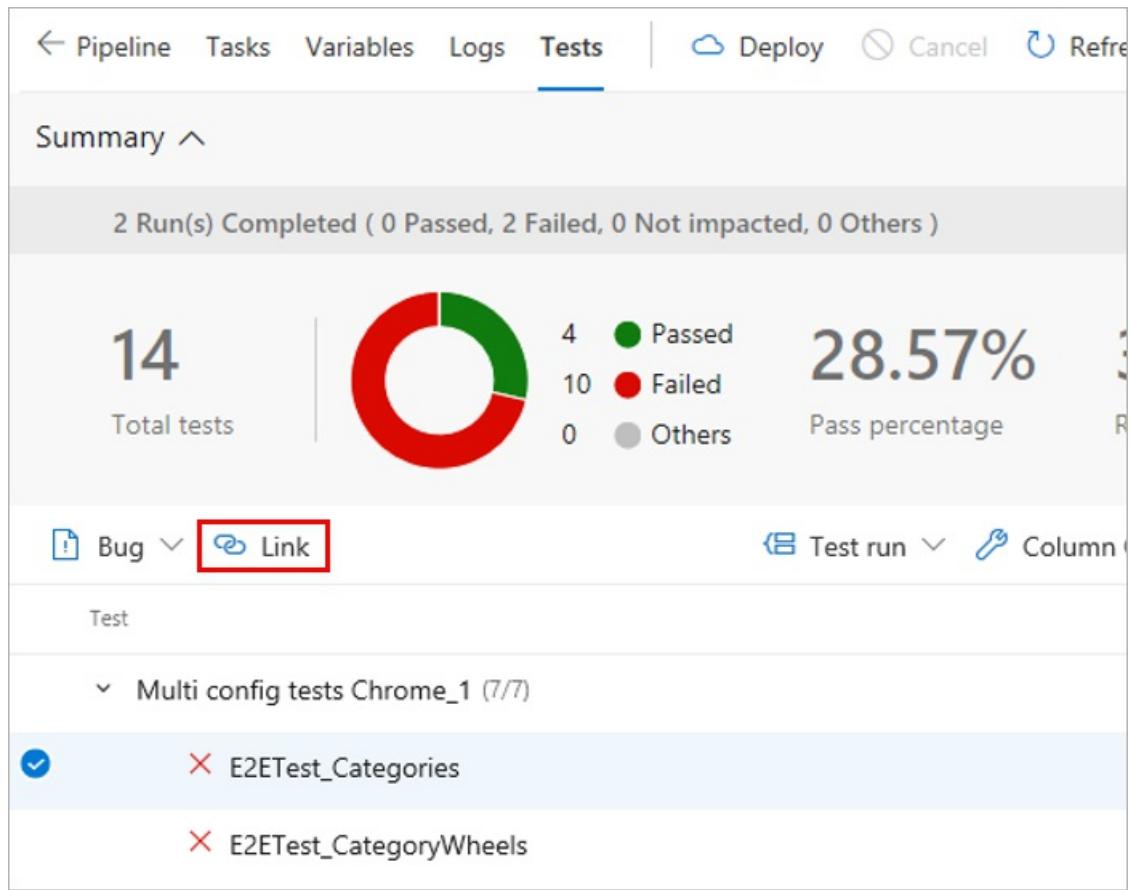
- View the widget in the team's dashboard. It lists all the Requirements in scope, along with the Pass Rate for the tests and count of Failed tests. Selecting a Failed test count opens the Tests tab for the selected build or release. The widget also helps to track the requirements without any associated test(s).

Current iteration quality

Title	Failed	Pass rate
2628: Create and track support tickets	1	<div style="width: 50%;">50%</div>
2630: Escalate a support ticket	0	<div style="width: 100%;">100%</div>
3040: Video	2	<div style="width: 50%;">50%</div>
2747: User should be able to see products in...	2	<div style="width: 71%;">71%</div>
2750: As a user, I can purchase products from...	0	<div style="width: 100%;">100%</div>
3037: Customer service on call	0	<div style="width: 100%;">100%</div>
2748: As a user, I can add items to cart	1	<div style="width: 75%;">75%</div>
+ 6 more Requirements		

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.

- 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.

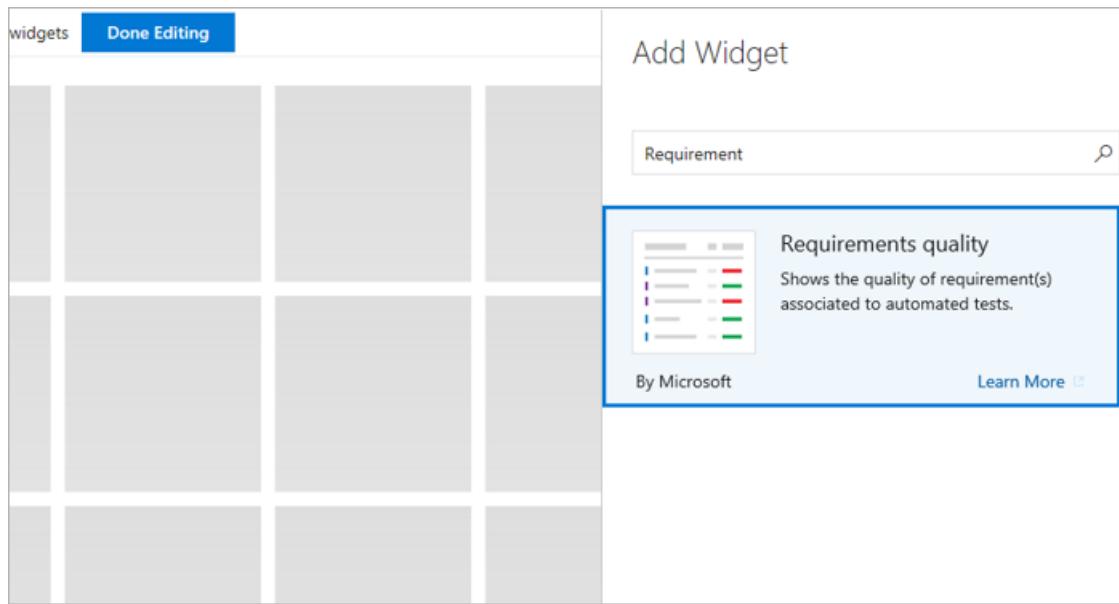
Showing 4 suggestions

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	[Redacted]
2751	As a user, I can search the...	Resolved	[Redacted]

Associate **Close**

The list shows only work items belonging to the Requirements category.

3. Teams often want to pin the summarized view of requirements traceability to a dashboard. Use the [Requirements quality](#) widget for this.



4. 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.

The configuration dialog has the following fields:

- Title:** Current iteration quality
- Size:** 3 x 2
- Requirements query:** Open User Stories
- Quality Data:** From Release (radio button selected)
- Release pipeline:** PartsUnlimited.CD

At the bottom are 'Save' and 'Cancel' buttons.

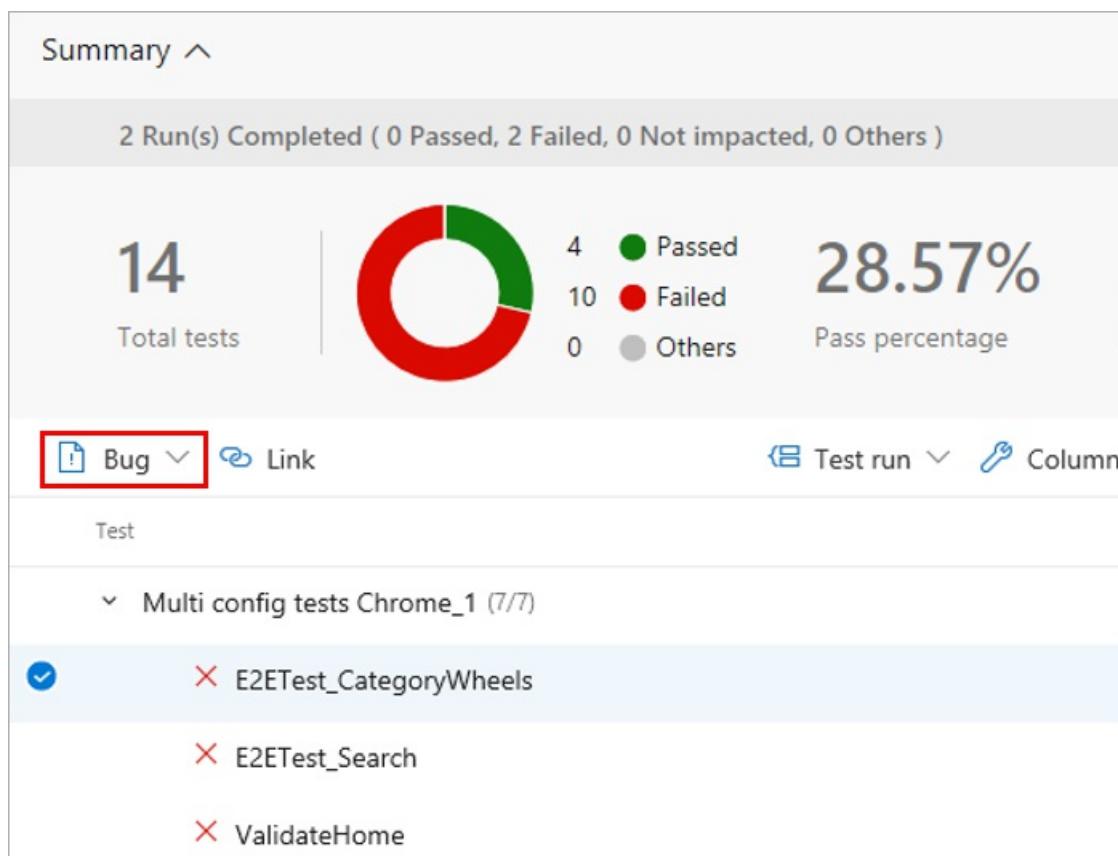
5. View the widget in the team's dashboard. It lists all the Requirements in scope, along with the Pass Rate for the tests and count of Failed tests. Selecting a Failed test count opens the Tests tab for the selected build or release. The widget also helps to track the requirements without any associated test(s).

Current iteration quality			
Title	Failed	Pass rate	
[2628: Create and track support tickets]	1	<div style="width: 50%;">50%</div>	
[2630: Escalate a support ticket]	0	<div style="width: 100%;">100%</div>	
[3040: Video]	2	<div style="width: 50%;">50%</div>	
[2747: User should be able to see products in...]	2	<div style="width: 71%;">71%</div>	
[2750: As a user, I can purchase products from...]	0	<div style="width: 100%;">100%</div>	
[3037: Customer service on call]	0	<div style="width: 100%;">100%</div>	
[2748: As a user, I can add items to cart]	1	<div style="width: 75%;">75%</div>	
+ 6 more Requirements			

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.

1. 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.



2. 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	<input checked="" type="radio"/> 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
Story Points
Priority 2
Severity 3 - Medium
Activity

3. 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

4 Passed 10 Failed 0 Skipped

Bug Link Test

Multi config tests Chrome_1 (7/7)

E2ETest_CategoryWheels

E2ETest_Search

Debug Work items Attachments History

Bugs (1)

ID	Title	State
4246	🐞 E2ETest_CategoryWheels Failed in...	New

Requirements (2)

ID	Title	State
3262	User can add cash to wallet	New
2628	Create and track support tickets	Resolved

- 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	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 ▲

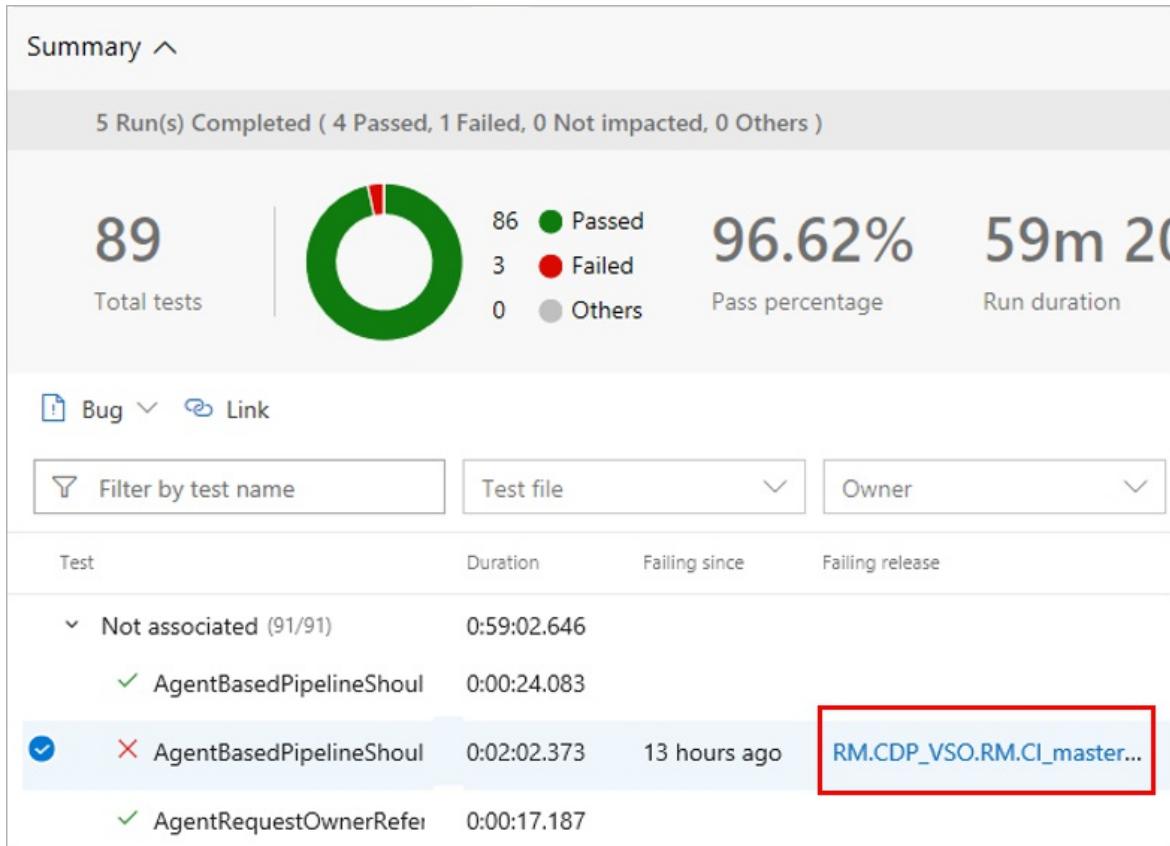
```
Test method FunctionalTests.FunctionalTests.E2ETest_CategoryWheels failed
OpenQA.Selenium.WebDriverException: Unexpected error. System.Net.WebConnectionException: Connection refused
  at System.Net.Sockets.Socket.DoConnect(EndPoint endPointSnapshot, SocketAddress socketAddress)
  at System.Net.ServicePoint.ConnectSocketInternal(Boolean 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

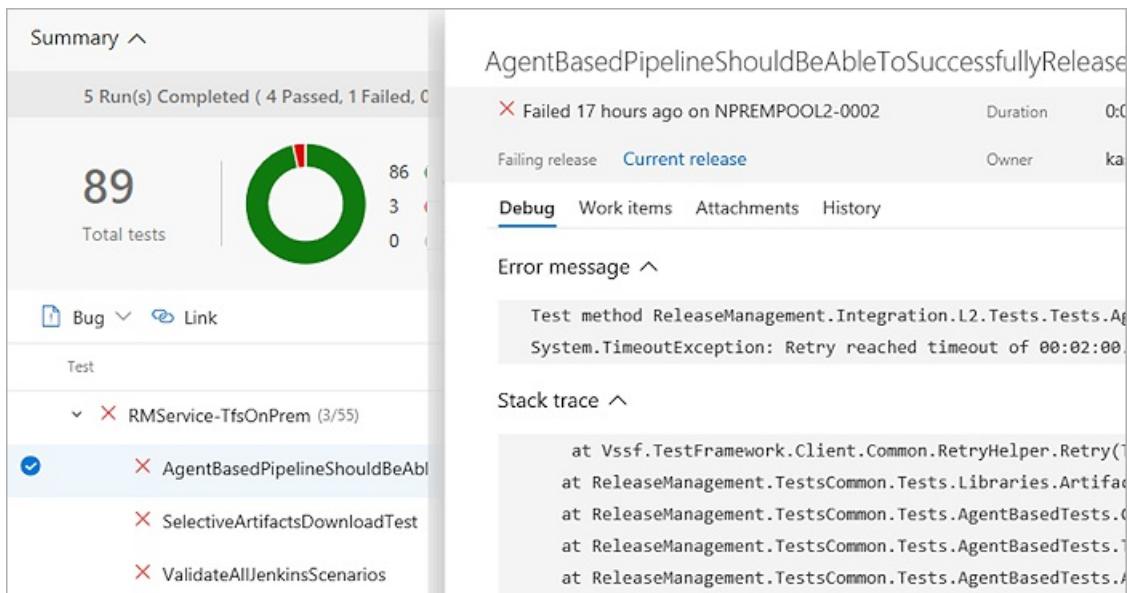
the initial set of changes - where the failure originated. This can help significantly to narrow down the scope for identifying the problematic test or source under test. To discover the first instance of test failures and trace it back to the associated code changes, visit [Tests tab](#) in build or release.

1. 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.



Test	Duration	Failing since	Failing release
Not associated (91/91)	0:59:02.646		
✓ AgentBasedPipelineShouldBeAbleToSuccessfullyRelease	0:00:24.083		
<input checked="" type="checkbox"/> ✘ AgentBasedPipelineShouldBeAbleToSuccessfullyRelease	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.



Summary ^

5 Run(s) Completed (4 Passed, 1 Failed, 0 Not impacted, 0 Others)

89 Total tests

86 Passed
3 Failed
0 Others

Bug ▾ Link

Test	Failing release	Current release	Owner
AgentBasedPipelineShouldBeAbleToSuccessfullyRelease	Failed 17 hours ago on NPREMPOOL2-0002		0:00:24.083
RMService-TfsOnPrem (3/55)			ka...

Debug Work items Attachments History

Error message ^

```
Test method ReleaseManagement.Integration.L2.Tests.AgentBasedPipelineShouldBeAbleToSuccessfullyReleaseTest.AgentBasedPipelineShouldBeAbleToSuccessfullyRelease threw an exception of type System.TimeoutException: Retry reached timeout of 00:02:00
```

Stack trace ^

```
at Vssf.TestFramework.Client.Common.RetryHelper.Retry()
at ReleaseManagement.TestsCommon.Tests.Libraries.ArtifactsDownloadTest.AgentBasedPipelineShouldBeAbleToSuccessfullyRelease()
at ReleaseManagement.TestsCommon.Tests.AgentBasedTests.AgentBasedPipelineShouldBeAbleToSuccessfullyRelease()
at ReleaseManagement.TestsCommon.Tests.AgentBasedTests.AgentBasedPipelineShouldBeAbleToSuccessfullyRelease()
at ReleaseManagement.TestsCommon.Tests.AgentBasedTests.AgentBasedPipelineShouldBeAbleToSuccessfullyRelease()
```

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.

The screenshot shows the Azure DevOps Pipeline interface for a release named "RM.CDP_VSO.RM.Cl_master_Aug22.24.1". The pipeline consists of three stages: "TfsOnPrem", "RmEqTfs", and "RmGtTfs".

- Release:** Continuous deployment for Microsoft.VisualStudio... on 8/22/2018 3:02 PM.
- Artifacts:**
 - Microsoft_vsts-azurev... (573635540) - master
 - VSO.RM.Cl (VSO.RM.Cl_master_Aug22.24) - master
 - RM.ArtifactExtensions.R... (20180806.1) - release
- Stages:**
 - TfsOnPrem**: Failed. Validate Test Results task failed on 8/22/2018 4:16 PM. Progress: 96%, Failures: 3.
 - RmEqTfs**: Failed. Validate Test Results task failed on 8/22/2018 5:17 PM. Progress: 97%, Failures: 6.
 - RmGtTfs**: Succeeded. Progress: 100%.

TfsOnPrem details:
Failed
Summary Commits Work Items
Difference in commits
Showing changes compared to RM.CDP_VSO.RM.Cl_...
VSO.RM.Cl (2 commits)
Author Clear Filters
Commit Message
1e3cc7e3 Merged PR 377628: Added perm...
9d4caaae Merged PR 377487: CI YAML De...

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

- See our [troubleshooting](#) page
- Get advice on [Stack Overflow](#), and get support via the [Developer Community](#)

Review test results

4/29/2021 • 10 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - 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.

You can also perform deeper analysis of test results by using the [Analytics Service](#). For an example of using this with your build and deploy pipelines, see [Analyze test results](#).

Read the [glossary](#) to understand test report terminology.

NOTE

Test report is available in TFS 2015 and above, however the new experience described in this topic is currently available only in Azure Pipelines.

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Published test results can be viewed in the **Tests** tab in a build or release summary.

Surface test results in the Tests tab

Test results can be surfaced in the **Tests** tab using one of the following options:

- **Automatically inferred test results.** By default, your pipeline can automatically infer the test output for a few popular test runners. This is done by parsing the error logs generated during the build operation and then checking for signatures of test failures.

Currently, Azure DevOps supports the following languages and test runners for automatically inferring the test results:

- Javascript - Mocha, Jest and Jasmine
- Python- Unittest

NOTE

This inferred test report is a limited experience. Some features available in fully-formed test reports are not present here ([more details](#)). We recommend that you publish a fully-formed test report to get the full Test and Insights experience in Pipelines. Also see:

- [Publishing fully-formed test reports for JavaScript test runners](#)
- [Publishing fully-formed test reports for Python test runners](#)

- **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.** Task that 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 by 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 for troubleshooting failures. Azure DevOps also provides other ways to surface the test information:

- The [Dashboard](#) provides visibility of 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.

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 t

1406 1316 1314 1744 2218

Logs Timeline Code coverage* Accessibility* Scan Results* Tests Build Targets

Progression



Build artifacts published ▾

2



Tests succeeded ▾

100% passed



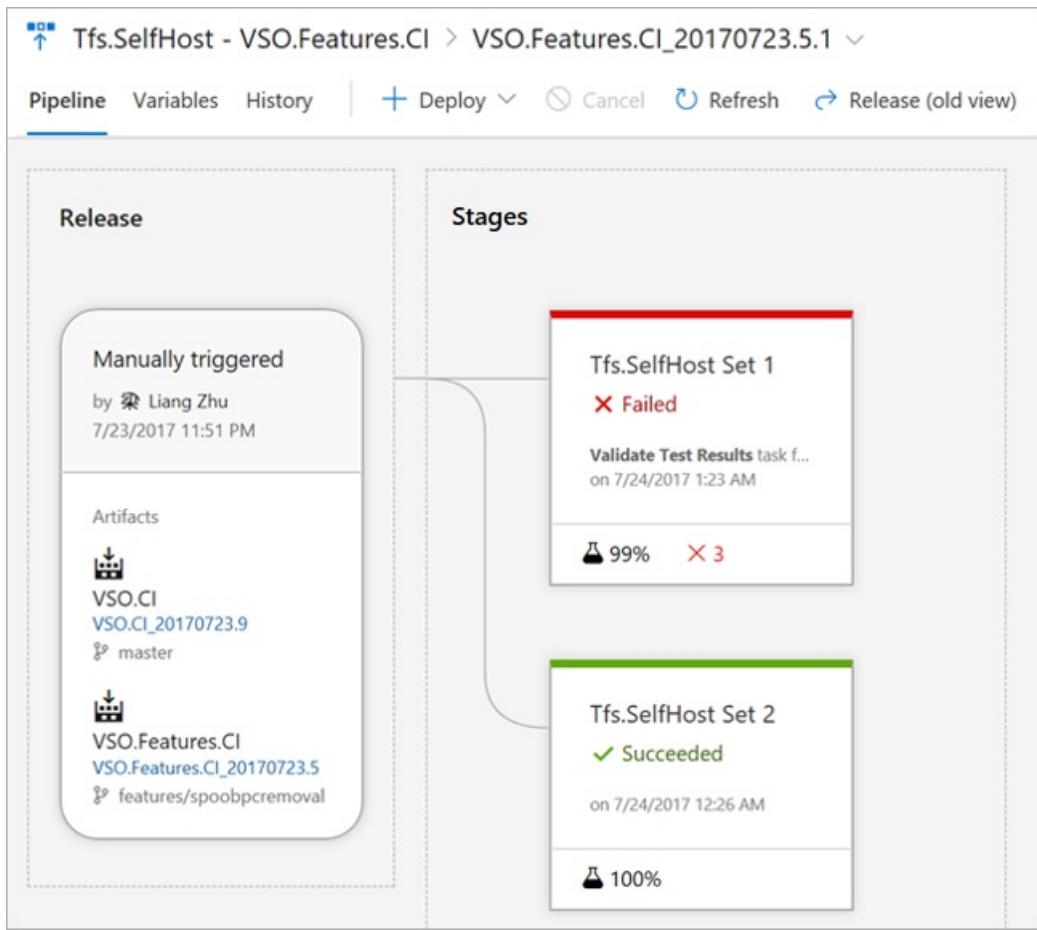
Build process succeeded ▾

4 error(s) / 0 warning(s)

- ✗ Toolsets\RunTests\ExecuteLOTests.Leaf.targets(18,5): Error : Error: Error: Timeout occurred when
- ✗ Toolsets\RunTests\ExecuteLOTests.Leaf.targets(18,5): Error : While Running:d:\v2.0\P1_work\1\sl\Tfs\Service\WebAccess\Wiki\TestScripts\Tests\AdminSecuritySourceTests.ts
- ✗ Toolsets\RunTests\ExecuteLOTests.Leaf.targets(18,5): Error :
- ✗ Toolsets\RunTests\ExecuteLOTests.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 summary page. 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 arrow. Below the header are navigation links: Pipeline, Variables, History, Deploy (with a plus sign), Cancel, Refresh, and Release (old view).

The main area is divided into two sections: "Release" on the left and "Stages" on the right.

In the "Release" section:

- A box labeled "Manually triggered" by "Liang Zhu" on "7/23/2017 11:51 PM".
- A "Artifacts" section listing:
 - VSO.CI (VSO.CI_20170723.9) from branch master.
 - VSO.Features.CI (VSO.Features.CI_20170723.5) from branch features/spoobpcremoval.

In the "Stages" section:

- A red-bordered box for "Tfs.SelfHost Set 1" which failed. It lists the "Validate Test Results task f..." on "7/24/2017 1:23 AM". Below it are metrics: 99% and 3 failed tests.
- A green-bordered box for "Tfs.SelfHost Set 2" which succeeded. It lists the success on "7/24/2017 12:26 AM". Below it is a metric of 100%.

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

◆ 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 | ↗

Summary ^

2 Run(s) Completed (1 Passed, 1 Failed, 0 Not impacted, 0 Others)

56335 Total tests +56335

56333 Passed 2 Failed 0 Others

2 Failed (+2) tests 2 New 0 Existing

99.99% Pass percentage ↑ 99.9%

Bug ↗ Link Test run ↗ Column Options ↗

Owner Failed (+1)

Test Duration

L0 Run - VSO.PR (2/50272) 0:54:46.379

FromWebApiShouldSetDefaultVersionTypeToSelectDuringReleaseCreationIfNoneExists New 0:00:00.050

GetInputValuesForDefaultVersionShouldReturnAllDefaultVersionOptions New 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

◆ 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 | ↗

Summary ^

2 Run(s) Completed (1 Passed, 1 Failed, 0 Not impacted, 0 Others)

56335 Total tests +56335

56333 Passed 2 Failed 0 Others

FromWebApiShouldSetDefaultVersionTypeToSelectDuringReleaseCreationIfNoneExists Failed an hour ago on TFSBUILD108

Failing build Current build Duration 0:00:00.050

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 at FluentAssertions.Execution.AssertionScope.FailWith(String message, Object[] args) at FluentAssertions.Primitives.StringEqualityValidator.Validate(String value, String expectedValue, String errorMessage, Object[] args) at FluentAssertions.Primitives.StringValidator.Validate() at FluentAssertions.Primitives.StringAssertions.Be(String value, String expectedValue, String errorMessage, Object[] args) at Microsoft.VisualStudio.Services.ReleaseManagement2.Data

TIP

If you use the Visual Studio Test task to run tests, diagnostic output logged from tests (using any of `Console.WriteLine`, `Trace.WriteLine` or `TestContext.WriteLine` methods), will appear as an attachment for a failed test.

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.

Test	Duration	Failing since
L0 Run - VSO.PR (4/50272)	0:54:46.379	
✓ 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.

Test debt management with bugs

To manage your test debt for failing or long running tests you can create a bug or add data to existing bug and all view all associated work items in the work item tab.

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\VisualStudio\VisualStudio\Services\ReleaseManagement2\ReleaseManagement2.csproj:line 100

Troubleshooting data for Test failure

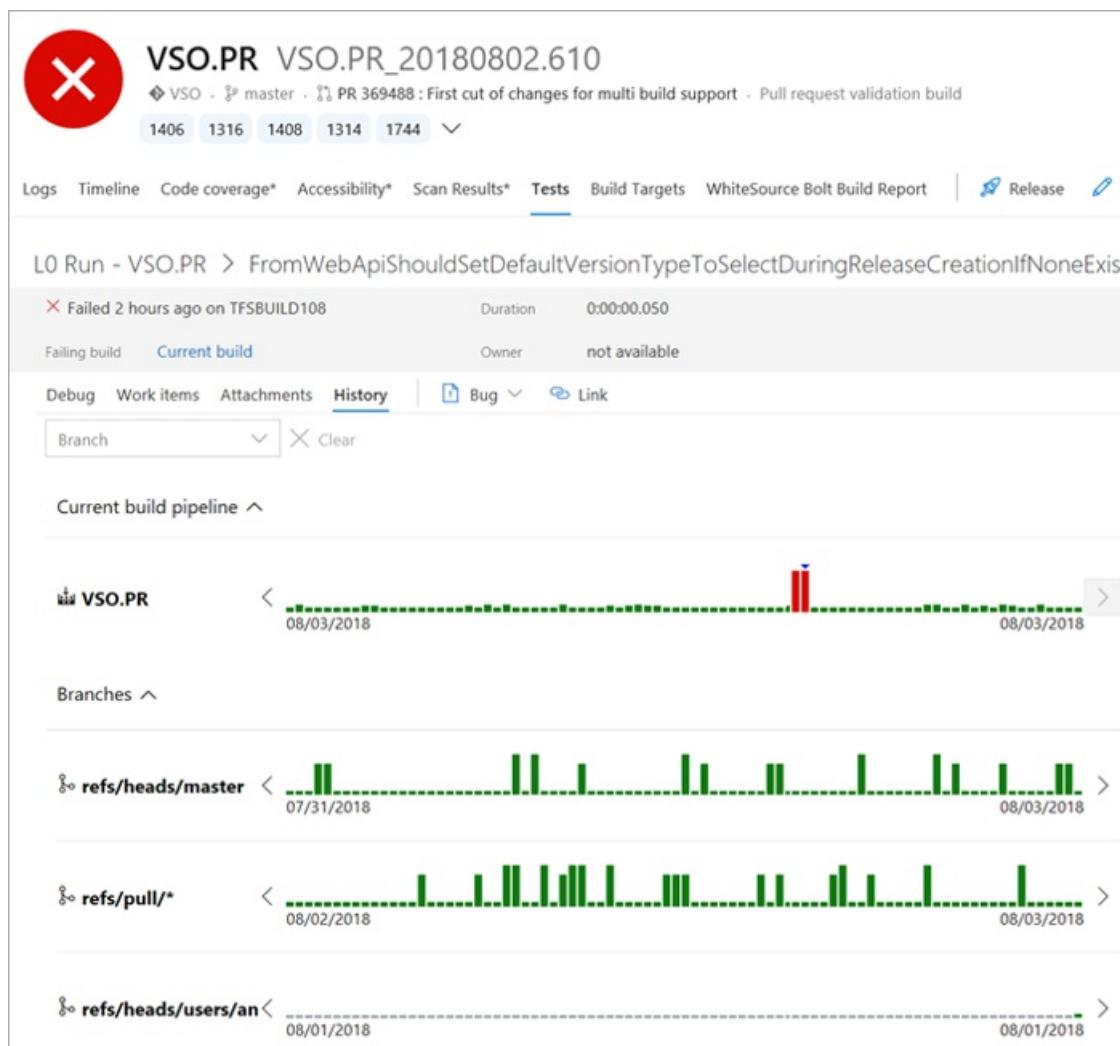
For the test failures, the error messages and stack traces are available for troubleshooting. You can also view all attachments associated with the test failure in the *Attachments* tab.

Test debt management

You can create or add to an existing bug to manage test debt for failures or long running tests. The *Work Items* tab details all bugs and requirements associated with a Test to help you analyze the requirement impact as well know status and who is working on the bug.

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.



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)

Release

Continuous deployment for Abhijit Chakraborty 6/13/2018 3:40 AM

Artifacts

VS.CI VS.CI_20180613.13 master

Stages

Tfs.ATTPC In progress for 57m

Phase 1/2

Run TFS.SelfHost 37:36

152/168 tasks

1,321 tests

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.

Builds Builds* Releases Releases* Library Task Groups Deployment Groups Packages

TFS.ATTPC - VS.CI > VS.CI_20180 > Tfs.ATTPC

← Pipeline Tasks Variables Logs Tests Refresh Release (old view)

Summary ^

5 Runs | 1 In Progress, 4 Completed

1321 Total tests | **1086** Passed | **2** Failed | 233 Not executed | 0 Others

Bug Link View more tests Test r

Test	Duration	Failing since	Failing release
TFS.SelfHost (2/1305)	0:00:00.000		
> ✗ AddMembersToProjectAsAdministratorsTest	0:08:01.047	just now	Current relea...
> ✗ AddMembersToProjectAsReadersTest	0:08:21.383	just now	Current relea...

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.

The screenshot shows the 'Tests' tab in a VSO.CI release pipeline. The summary indicates 4 runs completed with 4 passed, 0 failed, and 0 not impacted. A large green circle represents 1063 total tests, with 1063 Passed, 0 Failed, and 0 Others. The 'Test' section shows a hierarchy of tests under 'TFS.SelfHost.CodeDev'. The 'UpdateWorkItemTypeDefinitionFuzzTest' is expanded, showing two attempts: 'Attempt# 1 - ExportWorkItemTypeDefinitionFuzzTest' (Passed) and 'Attempt# 0 - ExportWorkItemTypeDefinitionFuzzTest' (Failed). The 'UpdateWorkItemTypeDefinitionFuzzTest' attempt is also expanded, showing its details. The 'Error message' and 'Stack trace' sections provide the reason for the failure.

Test	Status	Attempts
TFS.SelfHost.CodeDev (2/1306)	Passed	2
ExportWorkItemTypeDefinitionFuzzTest	Passed	1
Attempt# 1 - ExportWorkItemTypeDefinitionFuzzTest	Passed	1
Attempt# 0 - ExportWorkItemTypeDefinitionFuzzTest	Failed	1
UpdateWorkItemTypeDefinitionFuzzTest	Passed	1

UpdateWorkItemTypeDefinitionFuzzTest

Passed 2 days ago on RICP0POOL4-0070

Error message

```
Test method MS.TF.Test.WIT.REST.Tests.Security.WorkItemSystemException: Microsoft.VisualStudio.TestTools.UnitTesting.Assert.Inconclusive at Microsoft.VisualStudio.TestTools.UnitTesting.Assert.Inconclusive at MS.TF.Test.WIT.REST.Tests.Security.WorkItemTypeDefinitionFuzzTest at Common.TestFramework.L2.Fuzz.FuzzTest.<>c__DisplayClass1_0.<Run> at Common.TestFramework.L2.Fuzz.SingleThreadFuzzIteration at Common.TestFramework.L2.Fuzz.SingleThreadFuzzIteration
```

Stack trace

```
at Common.TestFramework.L2.Fuzz.SingleThreadFuzzIteration at Common.TestFramework.L2.Fuzz.SingleThreadFuzzIteration at Common.TestFramework.L2.Fuzz.FuzzTest.Start[Int32] at Common.TestFramework.L2.Fuzz.FuzzTest.Start[Int32] at MS.TF.Test.WIT.REST.Tests.Security.WorkItemTypeDefinitionFuzzTest
```

- **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

46 Passed
10 Failed
0 Others

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.

Release (old view) Edit release

(0 Others)

100%

Pass percentage

10m 11s

Run duration

4

Tests not reported

1 Run(s) Aborted

1,293

Total tests

80 Passed

1 Failed

1123 Aborted

89 Others

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.

Automatically inferred test results

Azure DevOps can automatically infer the output of tests that are running in your pipelines for a few supported test frameworks. These automatically inferred test reports require no specific configuration of your pipelines, and are a zero-effort way to get started using Test Reporting.

The screenshot shows the Test Reporting interface for a build. On the left, there's a summary card with metrics like Total tests (4), Pass percentage (50%), and Failed tests (2). Below it, a table lists test runs, with one row expanded to show details for a Jest test run. The expanded row includes a stack trace of the failing test code:

```

adds 2 + 3 to equal 5
✖ Failed 42m ago
Failing build Current build
Debug Work items Attachments History
Error message
Stack trace
● adds 2 + 3 to equal 5
expect(received).toBe(expected) // Object.is
Expected: 2
Received: 5
6 |
7 | test('adds 2 + 3 to equal 5', () => {
8 |   expect(sum(2, 3)).toBe(2);
9 | });
10 |
11 | test('adds 3 + 4 to equal 7', () => {
at Object.toBe (sum.test.js:8:23)

```

See the [list of runners for which test results are automatically inferred](#).

As only limited test metadata is present in such inferred reports, they are limited in features and capabilities. The following features are not available for inferred test reports:

- Group the test results by test file, owner, priority, and other fields
- Search and filter the test results
- Check details of passed tests
- Preview any attachments generated during the tests within the web UI itself
- Associate a test failure with a new bug, or see list of associated work items for this failure
- See build-on-build [analytics for testing in Pipelines](#)

NOTE

Some runners such as Mocha have multiple built-in console reporters such as [dot-matrix](#) and [progress-bar](#). If you have configured a non-default console output for your test runner, or you are using a custom reporter, Azure DevOps will not be able to infer the test results. It can only infer the results from the [default](#) reporter.

Related articles

- [Analyze test results](#)
- [Trace test requirements](#)
- [Review code coverage results](#)

Help and support

- See our [troubleshooting](#) page
- Get advice on [Stack Overflow](#), and get support via the [Developer Community](#)

Test Analytics

11/2/2020 • 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.

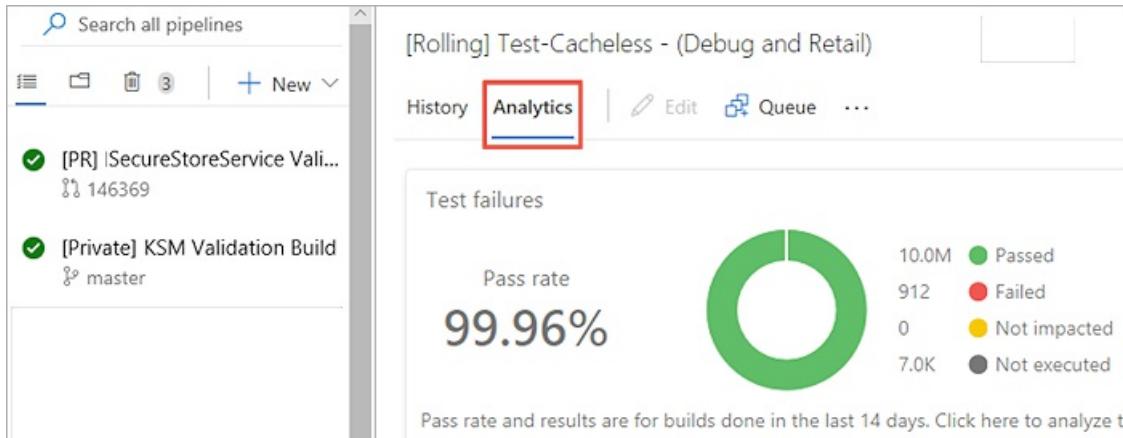
Read the [glossary](#) to understand test reports terminology.

Install the Analytics extension if required

For more information, see [The Analytics Marketplace extension](#).

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.

Release pipelines

Active All pipelines + New View pipeline Create a release All releases

Stages	Created
VSO.CI_20180829.172. VSO.CI_20180829.172	Analysis Analyze various performance indicators that give insight into selected release pipeline.
VSO.CI_20180829.171. VSO.CI_20180829.171	
VSO.CI_20180829.170. VSO.CI_20180829.170	
VSO.CI_20180829.169. VSO.CI_20180829.169	

Pass rate and results - last 14 days
99.95%

2.6M
986
0
566K

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.

Release pipelines > RM.CDP > Test Failures Report

Summary ^

Pass rate: **97.52%**

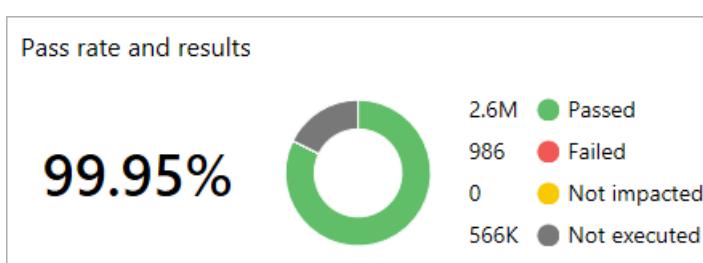
Trend of test results and pass rate ^

Test

Test	Failed	Pass rate	Total count	Average duration
AgentBasedPipelineShouldBeAbleToSuccessfullyReleaseWithExternalTfsArtifact	1108	53.65%	2391	123.16s
SelectiveArtifactsDownloadTest	980	52.88%	2622	69.87s
AddDeletePhaseTaskV2	403	47.66%	770	72.56s
RmoDevfabricUpgrade	263	3.3%	272	404.9s
VariableGroupV2	239	71.64%	843	66.14s

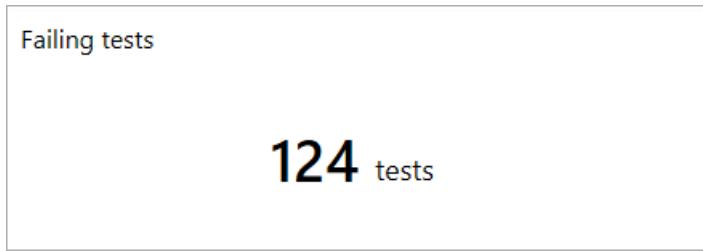
The detailed view contains two sections:

- **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.



- 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.



- Chart view: A trend of the total test failures and average pass rate on each day of the specified period.



- **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
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
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.

📅 14 Days ▾ ⚗ Failed ▾ 📄 Test file ▾

Summary ▾

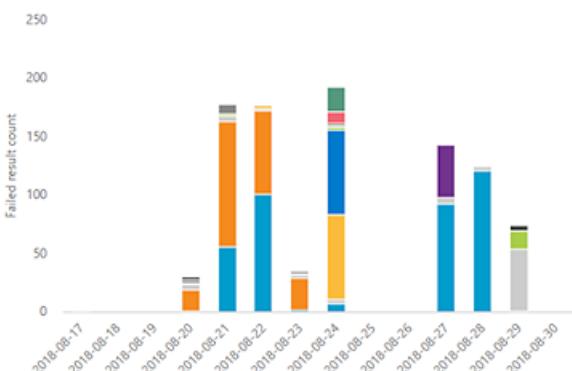
Pass rate and results

99.95%



2.6M Passed
986 Failed
0 Not impacted
566K Not executed

Result count ▾



Failing tests

124 tests

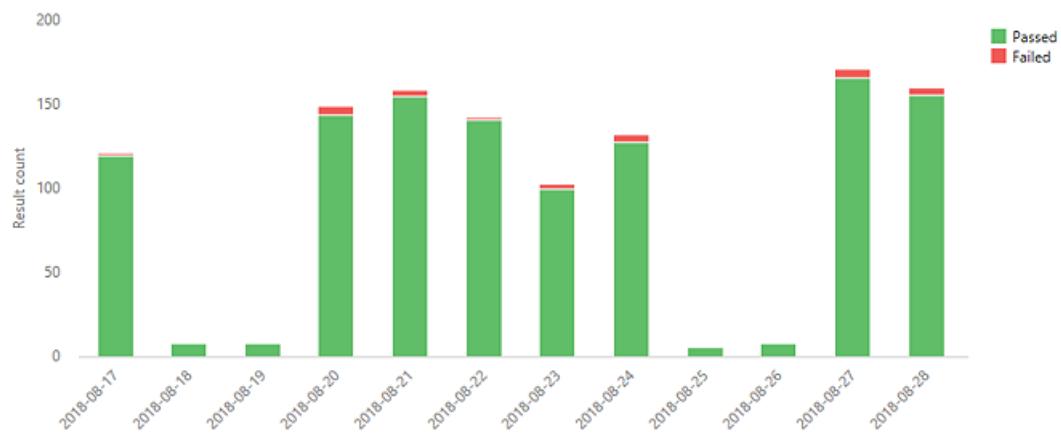
Test	Failed	Pass rate	↑	Total count	Average duration
CIWorkflow.Tfs.WebPlatform.L2.Tests.dll	383	91.5%	↑	15020	141.93s
ValidateRetentionTabExperienceForTfvcProject	368	75.49%	↑	1502	42.35s
ValidateOptionsTabExperience	15	99%	↑	1502	62.91s
Graph.Vssf.Sdk.L2.Tests.dll	87	99.03%	↑	9012	40.8s
Graph.Tfs.Client.L2.Tests.dll	9	99.7%	↑	3004	28.27s
WorkItemTracking.Tfs.ExtendedClient.L2.Tests.dll	234	99.82%	↑	135686	197.56s

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.

CachingGms_RemoteCacheInvalidation | ⏪ Back

14 Days ▾

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/

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
```

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.

TestWikiCollectionReindexing | ⏪ Back

Summary ▾

Result count ▾

Date	Result count
2018-08-01	5
2018-08-02	15
2018-08-03	2
2018-08-04	3
2018-08-05	7
2018-08-06	2
2018-08-07	7
2018-08-08	8
2018-08-09	6
2018-08-10	7
2018-08-11	7
2018-08-12	6
2018-08-13	7
2018-08-14	1
2018-08-15	5
2018-08-16	0
2018-08-17	6
2018-08-18	9
2018-08-19	6

Passed Failed

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

- See our [troubleshooting](#) page
- Get advice on [Stack Overflow](#), and get support via the [Developer Community](#)

Review code coverage results

11/2/2020 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - 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

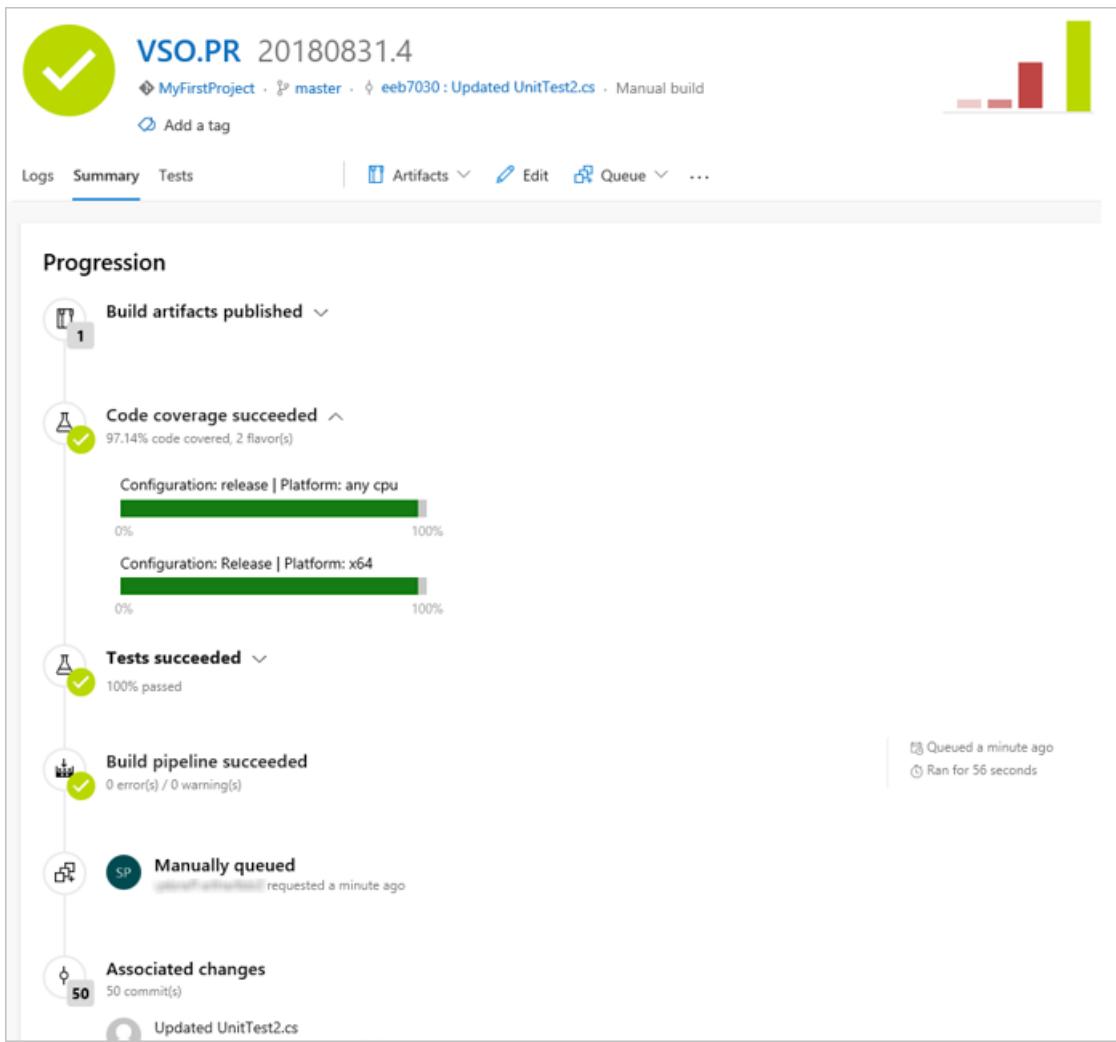
In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Example

To view an example of publishing code coverage results for your choice of language, see the **Ecosystems** 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.



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.

The code coverage summary can be viewed on the **Summary** tab on the pipeline run summary.

#20200409.3 Merged PR 40: Revert "Some fix"
on Pipeline-12 Retained

Summary Tests Code Coverage

Manually run by Jamal Hartnett [View change](#)

Repository and version	Time started and elapsed	Related	Tests and coverage
CC master · 23ac329	Apr 9 at 6:15 AM 3m 30s	0 work items 1 consumed	100% passed 91.67% covered

The results can be viewed and downloaded on the **Code coverage** tab.

#20200409.3 Merged PR 40: Revert "Some fix"
on Pipeline-12 ✘ Retained

Summary Tests **Code Coverage**

[Download code coverage results](#)

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

The code coverage artifacts published during the build can be viewed under the **Summary** tab on the pipeline run summary.

#20200506.25 Update azure-pipelines.yml for Azure Pipelines
on FabrikamPipeline ✘ Retained

[Run new](#) :

Summary Tests Code Coverage

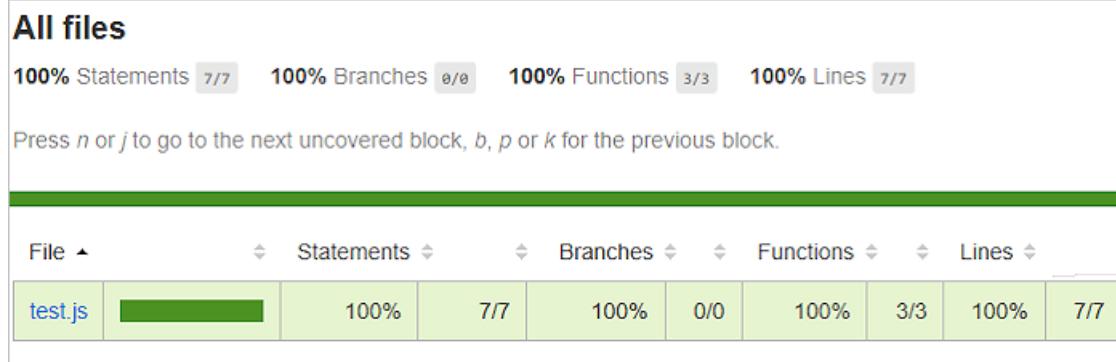
Triggered by Jamal Hartnett [View change](#)

Repository and version FabrikamFiber/FabrikamRepo master ↗ 5cce93f	Time started and elapsed May 6 at 2:22 PM 2m 43s	Related 0 work items	Tests and coverage 100% passed Setup code coverage
		1 published	

- 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.

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

- See our [troubleshooting](#) page
- Get advice on [Stack Overflow](#), and get support via the [Developer Community](#)

Code coverage for pull requests

3/17/2021 • 5 minutes to read • [Edit Online](#)

Azure Pipelines

Code coverage is an important quality metric and helps you measure the percentage of your project's code that is being tested. To ensure that quality for your project improves over time (or at the least, does not regress), it is essential that new code being brought into the system is well tested. This means that when developers raise pull requests, knowing whether their changes are covered by tests would help plug any testing holes before the changes are merged into the target branch. Repo owners may also want to set policies to prevent merging large untested changes.

Full coverage, diff coverage

Typically, coverage gets measured for the entire codebase of a project. This is **full coverage**. However, in the context of pull requests, developers are focused on the changes they are making and want to know whether the specific lines of code they have added or changed are covered. This is **diff coverage**.

Prerequisites

In order to get coverage metrics for a pull request, first configure a pipeline that validates pull requests. In this pipeline, configure the test tool you are using to collect code coverage metrics. Coverage results must then be published to the server for reporting.

To learn more about collecting and publishing code coverage results for the language of your choice, see the [Ecosystems](#) section. For example, collect and publish code coverage for [.NET core apps](#).

NOTE

While you can collect and publish code coverage results for many different languages using Azure Pipelines, the **code coverage for pull requests** feature discussed in this document is currently available only for .NET and .NET core projects using the Visual Studio code coverage results format (file extension .coverage). Support for other languages and coverage formats will be added in future milestones.

Coverage status, details and indicators

Once you have configured a pipeline that collects and publishes code coverage, it posts a code coverage status when a pull request is raised. By default, the server checks for atleast 70% of changed lines being covered by tests. The diff coverage threshold target can be changed to a value of your choice. See the settings configuration section below to learn more about this.

The screenshot shows the 'Status' section of a pull request. It displays a red 'X' icon and the message 'Coverage status check failed'. Below this, there's a 'Work Items' section with a 'No related work items' message. At the bottom, there's a 'Reviewers' section with an envelope icon.

The status check evaluates the diff coverage value for all the code files in the pull request. If you would like to view the % diff coverage value for each of the files, you can turn on details as mentioned in the configuration section. Turning on details posts details as a comment in the pull request.

A comment from 'Azure Pipelines Test Service' posted 2 minutes ago. It states: 'Diff coverage check failed. 0/1 (0.00 %) changed lines are covered up to [Update 1](#). Diff coverage target is 70.00 %'. Below this, a 'Details' section shows a table with one row for 'src/datacollector/MessageSink.cs'.

Changed files	Lines covered	Changed lines covered
src/datacollector/MessageSink.cs	0/4 (0.00 %)	0/1 (0.00 %)

In the changed files view of a pull request, lines that are changed are also annotated with coverage indicators to show whether those lines are covered.

A side-by-side code diff view. The left pane shows the original code with line numbers 10 through 23. The right pane shows the updated code with line numbers 10 through 24. A green bar highlights the changes made in lines 17-20. Coverage annotations are shown as colored bars: a grey bar covers lines 17-20 in the original code, and a green bar covers lines 17-20 in the updated code. Line 20 has a red 'x' icon indicating it was changed but not covered.

```

10  /// <inheritdoc />
11  internal class MessageSink : IMessageSink
12  {
13      /// <summary>
14      /// Data collection message as sent by DataCollection
15      /// </summary>
16      /// <param name="args">Data collection message event arguments</param>
17      public void SendMessage(DataCollectionEventArgs args)
18      {
19          DataCollectionRequestHandler.Instance.SendDataCol
20      }
21  }
22
23
24
10  /// <inheritdoc />
11  internal class MessageSink : IMessageSink
12  {
13      /// <summary>
14      /// Data collection message as sent by DataCollection
15      /// </summary>
16      /// <param name="args">Data collection message event arguments</param>
17      public void SendMessage(DataCollectionEventArgs args)
18      {
19          DataCollectionRequestHandler.Instance.SendDataCol
20      }
21
22
23
24

```

NOTE

While you can build code from a wide variety of version control systems that Azure Pipelines supports, the **code coverage for pull requests** feature discussed in this document is currently available only for Azure Repos.

Configuring coverage settings

If you would like to change the default settings of the code coverage experience for pull requests, you must include a configuration YAML file named `azurepipelines-coverage.yml` at the root of your repo. Set the desired values in this file and it will be used automatically the next time the pipeline runs.

The settings that can be changed are:

SETTING	DESCRIPTION	DEFAULT	PERMISSIBLE VALUES
status	Indicates whether code coverage status check should be posted on pull requests. Turning this off will not post any coverage checks and coverage annotations will not appear in the changed files view.	on	on, off
target	Target threshold value for diff coverage must be met for a successful coverage status to be posted.	70%	Desired % number
comments	Indicates whether a comment containing coverage details for each code file should be posted in the pull request	off	on, off

Example configuration:

```
coverage:
  status:          # Code coverage status will be posted to pull requests based on targets defined below.
  comments: on    # Off by default. When on, details about coverage for each file changed will be posted
                 as a pull request comment.
  diff:           # Diff coverage is code coverage only for the lines changed in a pull request.
  target: 60%    # Set this to a desired percentage. Default is 70 percent
```

More examples with details can be found in the [code coverage YAML samples repo](#).

NOTE

Coverage indicators light up in the changed files view regardless of whether the pull request comment details are turned on.

TIP

The coverage settings YAML is different from a YAML pipeline. This is because the coverage settings apply to your repo and will be used regardless of which pipeline builds your code. This separation also means that if you are using the classic designer-based build pipelines, you will get the code coverage status check for pull requests.

Protect a branch using a code coverage policy

Code coverage status check for pull requests is only a suggestion for developers and it does not prevent pull requests with low code coverage from being merged into the target branch. If you maintain a repo where you would like to prevent developers from merging changes that do not meet a coverage threshold, you must configure a [branch policy using the coverage status check](#).

TIP

Code coverage status posted from a pipeline follows the naming convention `{name-of-your-pipeline/codecov}`.

NOTE

Branch policies in Azure Repos (even optional policies) prevent pull requests from completing automatically if they fail. This behavior is not specific to code coverage policy.

FAQ

Which coverage tools and result formats can be used for validating code coverage in pull requests?

Code coverage for pull requests capability is currently only available for Visual Studio code coverage (.coverage) formats. This can be used if you publish code coverage using the Visual Studio Test task, the test verb of dotnet core task and the TRX option of the publish test results task. Support for other coverage tools and result formats will be added in future milestones.

If multiple pipelines are triggered when a pull request is raised, will coverage be merged across the pipelines?

If multiple pipelines are triggered when a pull request is raised, code coverage will not be merged. The capability is currently designed for a single pipeline that collects and publishes code coverage for pull requests. If you need the ability to merge coverage data across pipelines, please file a feature request on [Developer community](#).

Help and support

- See our [troubleshooting](#) page
- Get advice on [Stack Overflow](#), and get support via the [Developer Community](#)

Create and target an environment

6/2/2021 • 5 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020

An environment is a collection of resources, such as Kubernetes clusters and virtual machines, that can be targeted by deployments from a pipeline. Typical examples of environment names are Dev, Test, QA, Staging, and Production.

The advantages of using environments include the following.

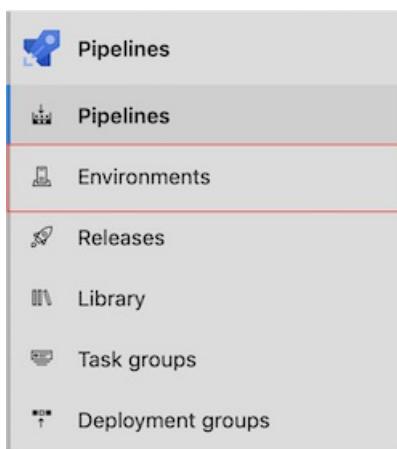
- **Deployment history** - Pipeline name and run details are recorded for deployments to an environment and its resources. In the context of multiple pipelines targeting the same environment or resource, [deployment history](#) of an environment is useful to identify the source of changes.
- **Traceability of commits and work items** - View jobs within the pipeline run that target an environment. You can also view the [commits and work items](#) that were newly deployed to the environment. Traceability also allows one to track whether a code change (commit) or feature/bug-fix (work items) reached an environment.
- **Diagnose resource health** - Validate whether the application is functioning at its desired state.
- **Permissions** - Secure environments by specifying which users and pipelines are allowed to target an environment.

Environment Resources

While an environment is a grouping of resources, the resources themselves represent actual deployment targets. The [Kubernetes resource](#) and [virtual machine resource](#) types are currently supported.

Create an environment

1. Sign in to your Azure DevOps organization and navigate to your project.
2. In your project, navigate to the Pipelines page. Then choose Environments and click on **Create Environment**.



3. After adding the name of an environment (required) and the description (optional), you can create an environment. Resources can be added to an existing environment later as well.

TIP

It is possible to create an empty environment and reference it from deployment jobs. This will let you record the deployment history against the environment.

NOTE

You can use a Pipeline to create, and deploy to environments as well. To learn more, see the [how to guide](#).

Target an environment from a deployment job

A [deployment job](#) is a collection of steps to be run sequentially. A deployment job can be used to target an entire environment (group of resources) as shown in the following YAML snippet.

```
- stage: deploy
  jobs:
    - deployment: DeployWeb
      displayName: deploy Web App
      pool:
        vmImage: 'Ubuntu-latest'
      # creates an environment if it doesn't exist
      environment: 'smarthotel-dev'
      strategy:
        runOnce:
          deploy:
            steps:
              - script: echo Hello world
```

NOTE

If the specified environment doesn't already exist, an empty environment is created using the environment name provided.

Target a specific resource within an environment from deployment job

You can scope the target of deployment to a particular resource within the environment. This allows you to record deployment history on a specific resource within the environment. The steps of the deployment job **automatically inherit** the service connection details from the resource targeted by the deployment job.

```

environment: 'smarthotel-dev.bookings'
strategy:
runOnce:
deploy:
steps:
- task: KubernetesManifest@0
displayName: Deploy to Kubernetes cluster
inputs:
action: deploy
namespace: ${k8sNamespace}
manifests: ${System.ArtifactsDirectory}/manifests/*
imagePullSecrets: ${imagePullSecret}
containers: ${containerRegistry}/${imageRepository}:${tag}
# value for kubernetesServiceConnection input automatically passed down to task by
environment.resource input

```

Environment in run details

All environments targeted by deployment jobs of a specific run of a pipeline can be found under the *Environments* tab of pipeline run details.

The screenshot shows the 'Environments' tab for a pipeline run. At the top, there's a summary bar with a checkmark icon, the run ID '#20190503.3 Update azure-pipelines.yml for Azure Pipelines', the pipeline name 'on SmartHotel360-public-web', and buttons for 'Run new' and more options. Below this, there are two sections: 'smarthotel-dev' and 'smarthotel-test'. Each section has a 'View environment' button. Under 'smarthotel-dev', there's a table showing four deployment jobs: 'deploy bookings', 'deploy notifications', 'deploy configurations', and 'deploy payments', each targeting the 'bookings' resource and taking approximately 3 seconds. Under 'smarthotel-test', there's a table showing one job: 'runSecurityTests', targeting the 'bookings' resource and taking approximately 3 seconds.

Jobs	Resource	Duration
✓ deploy bookings	bookings	⌚ 3s
✓ deploy notifications	notifications	⌚ 3s
✓ deploy configurations	configurations	⌚ 3s
✓ deploy payments	payments	⌚ 3s

Jobs	Resource	Duration
✓ runSecurityTests	bookings	⌚ 3s

If you're using an AKS private cluster, the **Environments** tab isn't available.

Approvals

You can manually control when a stage should run using approval checks. You can use approval checks to control deployments to production environments. Checks are a mechanism available to the *resource owner* to control when a stage in a pipeline consumes a resource. As the owner of a resource, such as an environment, you can [define approvals and checks](#) that must be satisfied before a stage consuming that resource starts.

Currently, manual approval checks are supported on environments. For more information, see [Approvals](#).

The creator, administrator, and user roles can manage approvals and checks. The reader role cannot manage approvals and checks.

Deployment history within environments

The deployment history view within environments provides the following advantages.

1. View jobs from all pipelines that are targeting a specific environment. Consider the scenario where two microservices, each having its own pipeline, are deploying to the same environment. In that case, the deployment history listing helps identify all pipelines that are impacting this environment and also helps

visualize the sequence of deployments by each pipeline.

The screenshot shows the 'Deployment' tab for the environment 'smarthotel-dev'. It lists five deployment runs, each with a job status and timestamp. The first run is successful, while the last four show deployment failures.

Run	Jobs	
Update azure-pipelines.yml for Azure Pipelines #20190503.3 on Smarthotel360-public-web	DeployBookings and 3 more	Yesterday 1m 7s
Update azure-pipelines.yml for Azure Pipelines #20190426.15 on Smarthotel360-public-web	DeployWeb	Apr 26 51s
Update azure-pipelines.yml for Azure Pipelines #20190426.10 on Smarthotel360-public-web	DeployWeb	Apr 26 40s
Update azure-pipelines.yml for Azure Pipelines #20190426.7 on Smarthotel360-public-web	Deploy	Apr 26 11m 4s
Update azure-pipelines.yml for Azure Pipelines #20190426.6 on Smarthotel360-public-web	Deploy	Apr 26 49s

2. Drill down into the job details reveals the listing of commits and work items that were newly deployed to the environment.

The screenshot shows the deployment details for run #9398189 targeting the 'smarthotel-dev' environment. It lists three commits from the 'Commits' tab, each with a SHA value.

Commit	SHA
Update azure-pipelines.yml for Azure Pipelines RN RoopeshNair authored on Apr 24	55a74fd7e
Update azure-pipelines.yml for Azure Pipelines RN RoopeshNair authored on Apr 24	6e8881526
Update azure-pipelines.yml for Azure Pipelines RN RoopeshNair authored on Apr 24	87c03c37e

Security

User permissions

You can control who can create, view, use, and manage the environments with user permissions. There are four roles - Creator (scope: all environments), Reader, User, and Administrator. In the specific environment's **user permissions** panel, you can set the permissions that are inherited and you can override the roles for each environment.

- Navigate to the specific **Environment** that you would like to authorize.
- Click on overflow menu button located at the top-right part of the page next to "Add resource" and choose **Security** to view the settings.
- In the **User permissions** panel, click on **+ Add** to add a **User or group** and select a suitable **Role**.

ROLE	DESCRIPTION
Creator	Global role, available from environments hub security option. Members of this role can create the environment in the project. Contributors are added as members by default. Not applicable for environments auto created from YAML pipeline.
Reader	Members of this role can view the environment.

ROLE	DESCRIPTION
User	Members of this role can use the environment when authoring yaml pipelines.
Administrator	In addition to using the environment, members of this role can manage membership of all other roles for the environment. Creators are added as members by default.

NOTE

- If you create an environment within a YAML, contributors and project administrators will be granted **Administrator** role. This is typically used in provisioning Dev/Test environments.
- If you create an environment through the UI, only the creator will be granted the **Administrator** role. You should use the UI to create protected environments like for a production environment.

Pipeline permissions

Pipeline permissions can be used to authorize all or selected pipelines for deployment to the environment.

- To remove **Open access** on the environment or resource, click the **Restrict permission** in **Pipeline permissions**.
- To allow specific pipelines to deploy to an environment or a specific resource, click **+** and choose from the list of pipelines.

FAQ

I get an error message when I try to create an environment

If you see the message "Access denied: {User} needs Create permissions to perform the action", you need to check your organization-level permissions. Go to **Organization Settings > Users** and check if you have the stakeholder role. The stakeholder role cannot create environments. Change your access level and check to see if you can now create environments. See [Troubleshoot user and permissions management](#) to learn more about user permissions.

Environment - Kubernetes resource

5/18/2021 • 5 minutes to read • [Edit Online](#)

Azure Pipelines

You can use Kubernetes resources to target Kubernetes clusters in an [environment](#) for deployment. Use pipelines to deploy to Azure Kubernetes Service (AKS) and clusters from any other cloud provider.

The Kubernetes resource view within environments provides a glimpse of the status of objects within the namespace mapped to the resource. The resource view also overlays pipeline traceability on top of these objects so that you can trace back from a Kubernetes object to the pipeline and then back to the commit.

To learn more about how resources work, see [resources in YAML](#) and [security with resources](#).

Overview

The advantages of using Kubernetes resource views within environments include:

- **Pipeline traceability** - The [Kubernetes manifest task](#) used for deployments adds more annotations to portray pipeline traceability in resource views. This can help in identifying the originating Azure DevOps organization, project, and pipeline responsible for updates made to an object within the namespace.

The screenshot shows the 'ReplicaSet details' section. It includes a 'Created' timestamp ('Yesterday by Deploy to AKS • #20190503.10 on smarthotel360.smarthotel'), 'Image' ('smarthotelacr.azurecr.io/bootcamp-demo:326'), 'Selector' ('app=bootcamp-demo pod-template-hash=8c8c5699'), and 'Labels' ('app=bootcamp-demo pod-template-hash=8c8c5699'). Below this, the 'Pods' section shows 1 running pod named 'bootcamp-demo-8c8c5699-96n8w' with a 'Status' of 'Running' and a 'Created' time of 'Yesterday at 9:40 PM'.

- **Diagnose resource health** - Workload status can be useful in quickly debugging potential mistakes or regressions that could have been introduced by a new deployment. For example, in the case of unconfigured *imagePullSecrets* resulting in ImagePullBackOff errors, pod status information can help identify the root cause for this issue.

The screenshot shows the 'Pod details' section for a pod named 'demo-f665dc65b-mcrbv'. It displays the following information:

- Created: Mar 6 at 7:55 PM
- Restart policy: Always
- QoS class: BestEffort
- Node: aks-agentpool-35291255-1
- Image: shash.azurecr.io/playground-orders:test
- Labels: app=demo pod-template-hash=f665dc65b
- Status: Pending
- Conditions: Initialized=True; Ready=False; ContainersReady=False; PodScheduled=True

A red callout box highlights the error message 'Back-off pulling image "shash.azurecr.io/playground-orders:test"'.

- **Review App** - Review app works by deploying every pull request from Git repository to a dynamic

Kubernetes resource under the environment. Reviewers can see how those changes look and work with other dependent services before they're merged into the target branch and deployed to production.

Add a Kubernetes resource

Azure Kubernetes Service

When you use Azure Kubernetes Service a [ServiceAccount](#) is created in your chosen cluster and namespace. For a [RBAC](#) enabled cluster, [RoleBinding](#) is created as well to limit the scope of the created service account to the chosen namespace. For an RBAC-disabled cluster, the ServiceAccount created has cluster-wide privileges (across namespaces).

1. In the environment details page, click on **Add resource** and choose **Kubernetes**.
2. Select **Azure Kubernetes Service** in the Provider dropdown.
3. Choose the Azure subscription, cluster, and namespace (new/existing).
4. Click on **Validate and create** to create the Kubernetes resource.
5. Verify that you see a cluster for your environment.

Resources	Deployments
Name	Latest job
 default KubernetesEnvCluster	Never deployed

Using an existing service account

While the Azure Kubernetes Service option creates a new ServiceAccount, the generic provider option lets you use an existing ServiceAccount to map a Kubernetes resource within your environment to a namespace.

If you want to set up a Kubernetes service connection outside of an environment, see the [Kubernetes service connection](#) section in [Service connections](#).

TIP

Use the generic provider (existing service account) to map a Kubernetes resource to a namespace from a non-AKS cluster.

1. In the environment details page, select **Add resource** and choose **Kubernetes**.
2. Select **Generic provider (existing service account)** for your provider.
3. Add the cluster name and namespace values.
4. Add the server URL. You can get the URL with this command:

```
kubectl config view --minify -o 'jsonpath=.clusters[0].cluster.server'
```

5. To get your secret object, you'll need to first find the service account secret name.

```
kubectl get serviceAccounts <service-account-name> -n <namespace> -o 'jsonpath=.secrets[*].name'
```

6. Next, you can retrieve the secret object using the output of the previous step.

```
kubectl get secret <service-account-secret-name> -n <namespace> -o json
```

Copy and paste the Secret object fetched in JSON form into the Secret field.

7. Click **Validate and create** to create the Kubernetes resource.

Reference your Kubernetes resources in a pipeline

If you are using Azure Kubernetes Service and building a YAML pipeline, the easiest way to configure your pipeline is to use a template. Connect to your repository and select one of the two Kubernetes Service options:

- [Deploy to Azure Kubernetes Services template](#)
- Deploy to Kubernetes - Review app with Azure DevSpaces

The templates let you set up review apps without needing to write YAML code from scratch or create explicit role bindings manually.



Deploy to Azure Kubernetes Service

Build and push image to Azure Container Registry; Deploy to Azure Kubernetes Service



Deploy to Kubernetes - Review app with Azure DevSpaces

Build and push image to Azure Container Registry; Deploy to Azure Kuberentes Services and setup Review App with Azure DevSpaces

Setup review app

Below is an example YAML snippet for adding a Review App. In this example, the first deployment job is run for non-PR branches and performs deployments against regular Kubernetes resource under environments. The second job runs only for PR branches and deploys against review app resources (namespaces inside Kubernetes cluster) generated on the fly. These resources are marked with a 'Review' label in the resource listing view of the environment. You'll need to define variables to use in the pipeline. If you use the [Deploy to Azure Kubernetes Services template](#), these variables will be configured defined for you.

```
trigger: main

resources:
- repo: self

variables:
  dockerRegistryServiceConnection: '12345' # Docker service connection identifier
  vmImageName: 'myVM' # name of your VM
  envName: 'myEnv' # name of your environment
  resourceName: 'myResource' # name of the resource you are referencing
  dockerfilePath: '**/Dockerfile'
  tag: '$(Build.BuildId)' # tag added for your build
  k8sNamespaceForPR: 'my-namespace' # namespace used in your PR
  imagePullSecret: 'my-app-secret' # image pull secret
  imageRepository: 'name-of-image-repository' # name of image repository
  containerRegistry: 'mycontainer.azurecr.io' # path to container registry

  - stage: Deploy
    displayName: Deploy stage
    dependsOn: Build

  jobs:
    - deployment: Deploy
      condition: and(succeeded(), not(startsWith(variables['Build.SourceBranch'], 'refs/pull/')))
      displayName: Deploy
      pool:
        vmImage: $(vmImageName)
        environment: $(envName).$(resourceName)
```

```

strategy:
runOnce:
  deploy:
    steps:
      - task: KubernetesManifest@0
        displayName: Create imagePullSecret
        inputs:
          action: createSecret
          secretName: $(imagePullSecret)
          dockerRegistryEndpoint: $(dockerRegistryServiceConnection)

      - task: KubernetesManifest@0
        displayName: Deploy to Kubernetes cluster
        inputs:
          action: deploy
          manifests: |
            $(Pipeline.Workspace)/manifests/deployment.yml
            $(Pipeline.Workspace)/manifests/service.yml
          imagePullSecrets: |
            $(imagePullSecret)
          containers: |
            $(containerRegistry)}/${imageRepository}:$(tag)

- deployment: DeployPullRequest
  displayName: Deploy Pull request
  condition: and(succeeded(), startsWith(variables['Build.SourceBranch'], 'refs/pull/'))
  pool:
    vmImage: $(vmImageName)

  environment: '$(envName).$(k8sNamespaceForPR)'
  strategy:
    runOnce:
      deploy:
        steps:
          - reviewApp: $(resourceName)

          - task: Kubernetes@1
            displayName: 'Create a new namespace for the pull request'
            inputs:
              command: apply
              useConfigurationFile: true
              inline: '{ "kind": "Namespace", "apiVersion": "v1", "metadata": { "name": "$(k8sNamespaceForPR)" }}'

          - task: KubernetesManifest@0
            displayName: Create imagePullSecret
            inputs:
              action: createSecret
              secretName: $(imagePullSecret)
              namespace: $(k8sNamespaceForPR)
              dockerRegistryEndpoint: $(dockerRegistryServiceConnection)

          - task: KubernetesManifest@0
            displayName: Deploy to the new namespace in the Kubernetes cluster
            inputs:
              action: deploy
              namespace: $(k8sNamespaceForPR)
              manifests: |
                $(Pipeline.Workspace)/manifests/deployment.yml
                $(Pipeline.Workspace)/manifests/service.yml
              imagePullSecrets: |
                $(imagePullSecret)
              containers: |
                $(containerRegistry)}/${imageRepository}:$(tag)

```

To use this job in an **existing** pipeline, the service connection backing the regular Kubernetes environment resource needs to be modified to "Use cluster admin credentials". Alternatively, role bindings need to be created

for the underlying service account to the review app namespace.

Next steps

- [Build and deploy to Azure Kubernetes Service](#)
- [Deploy manifests and bake manifests](#)
- [Multi-cloud Kubernetes deployments](#)
- [Deployment strategies for Kubernetes in Azure Pipelines](#)
- [Deploy ASP.NET Core apps to Azure Kubernetes Service with Azure DevOps Starter](#)
- [REST API: Kubernetes with Azure DevOps](#)

Environment - virtual machine resource

4/6/2021 • 4 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020

You can use virtual machine resources to orchestrate deployments across multiple machines with YAML pipelines. Virtual machine resources let you install agents on your own target servers so that you can drive rolling deployment to those servers.

Virtual machine resources are connected to [environments](#). Once you define an environment, you can add virtual machines to it that you can target with deployments. The deployment history view in an environment provides traceability from your virtual machine to your pipeline.

Create a virtual machine resource

NOTE

You can use this same process to set up physical machines with a registration script.

The first step in adding a virtual machine resource is to define an environment. You can define environments in [Environments under Pipelines](#).

Define an environment

1. Select **Create environment** or **New environment** (if this is not your first environment).
2. Add a **Name** (required) for the environment and a **Description**.
3. Save the new environment.

Add a resource

1. Select your environment and choose **Add resource**.
2. Select **Virtual machines** for your **Resource type**. Then select **Next**.

Name ⓘ
ContosoDeploy

Description ⓘ
VM for deployment.

Resource

-  None
You can add resources later
-  Kubernetes
Add Kubernetes namespace
-  Virtual machines
Manage virtual machines

3. Choose Windows or Linux for the **Operating System**.
4. Copy the registration script. Your script will be a [PowerShell script](#) if you've selected Windows and a Linux script if you've selected Linux.

New environment X

Virtual machine resource

Provider

Generic provider

Operating system

Windows

Registration script

1. Run `$ErrorActionPreference = "Stop"` in administrative powershell

5. Run the copied script on each of the target virtual machines that you want to register with this environment.
 - If you're installing on Windows, you'll need to run the script an PowerShell administrator.
 - If you're installing on Linux, you'll need to have permission to download and run executable scripts.

NOTE

- The Personal Access Token (PAT) of the logged in user is included in the script. The PAT expires on the day you generate the script.
- If your VM already has any other agent running on it, provide a unique name for **agent** to register with the environment.
- To learn more about installing the agent script, see [Self-hosted Linux agents](#) and [Self-hosted Windows agents](#). The agent scripts for virtual machine resources are very similar to the scripts for self-hosted agents and you can use the same commands.

6. Once your VM is registered, it will start appearing as an environment resource under the **Resources** tab of the environment.
7. To add more VMs, copy the script again by clicking **Add resource** and selecting **Virtual machines**. The Windows and Linux scripts are the same for all the VMs added to the environment.
8. When the VM script is successfully installed, your VM will appear in the list of resources associated with your environment.

The screenshot shows a user interface titled 'VMenv' with a back arrow. Below it, there are two tabs: 'Resources' (which is selected) and 'Deployments'. A table lists a single VM entry:

Name	Latest job
USHAN-PC	Never deployed

Use virtual machine in pipelines

You'll target virtual machines in your pipeline by referencing the environment. By default, the pipeline job will run for all of the virtual machines defined for an environment.

```
trigger:
- main

pool:
  vmImage: ubuntu-latest

jobs:
- deployment: VMDeploy
displayName: Deploy to VM
environment:
  name: ContosoDeploy
  resourceType: VirtualMachine
strategy:
  runOnce:
    deploy:
      steps:
        - script: echo "Hello world"
```

You can select specific sets of virtual machines from the environment to receive the deployment with tags. For example, if you only want to deploy to resources with the `windows` tag, add the `tags` parameter and the value `windows` to your pipeline.

```

trigger:
- main

pool:
  vmImage: ubuntu-latest

jobs:
- deployment: VMDeploy
displayName: Deploy to VM
environment:
  name: ContosoDeploy
  resourceType: VirtualMachine
  tags: windows # only deploy to virtual machines with this tag
strategy:
  runOnce:
    deploy:
      steps:
        - script: echo "Hello world"

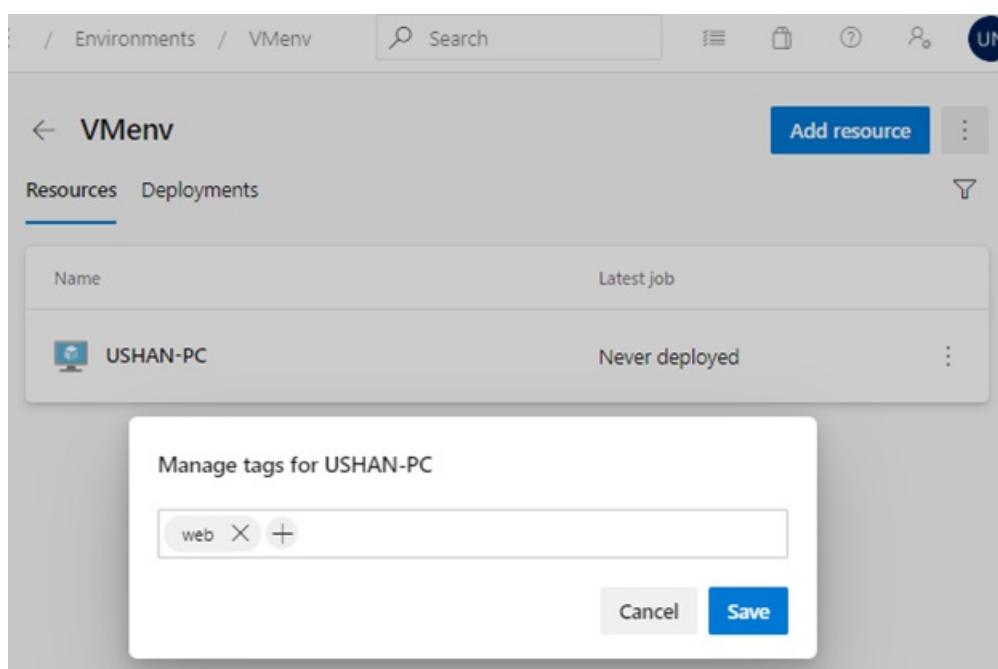
```

To learn more about deployment jobs, see the [YAML schema](#).

Add and manage tags

Tags give you a way to target specific virtual machines in an environment for deployment. You can add tags to the VM as part of the interactive registration script or through the UI. Tags are each limited to 256 characters. There is no limit to the number of tags you can use.

Add or remove tags in the UI from the resource view by selecting **More actions**  for a virtual machine resource.



The screenshot shows the Veeam ONE web interface. At the top, there's a navigation bar with 'Environments' and 'VMenv'. Below it, a search bar and various icons. The main area is titled 'VMenv' with a back arrow. It has tabs for 'Resources' (selected) and 'Deployments'. A large table lists a single VM named 'USHAN-PC' with the status 'Never deployed'. To the right of the table is a 'More actions' menu. A modal window is open over the table, titled 'Manage tags for USHAN-PC'. It contains a text input field with the tag 'web' and a delete 'X' and add '+' button. At the bottom of the modal are 'Cancel' and 'Save' buttons.

When you select multiple tags, virtual machines that include all the tags will be used in your pipeline. For example, this pipeline targets virtual machines with both the `windows` and `prod` tags. If a virtual machine only has one of these tags, it will not be targeted.

```

trigger:
- master

pool:
  vmImage: ubuntu-latest

jobs:
- deployment: VMDeploy
displayName: Deploy to VM
environment:
  name: ContosoDeploy
  resourceType: VirtualMachine
  tags: windows,prod # only deploy to virtual machines with both windows and prod tags
strategy:
  runOnce:
    deploy:
      steps:
        - script: echo "Hello world"

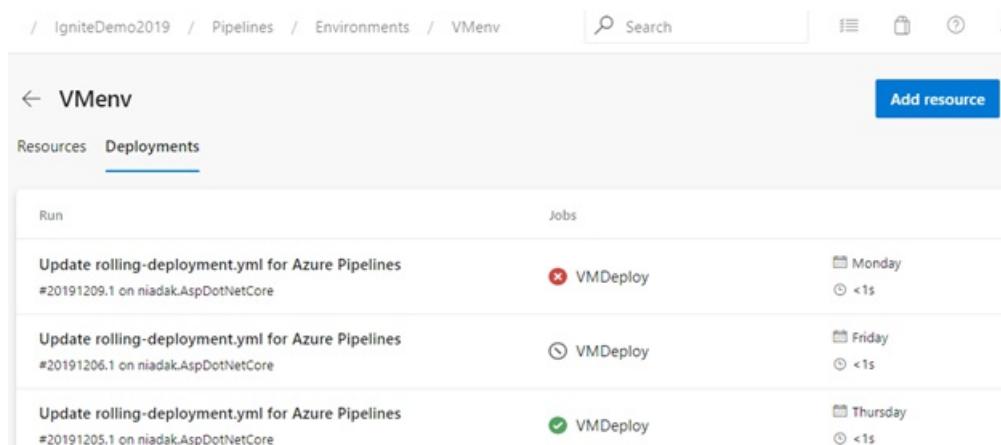
```

Apply deployment strategy

You can apply a deployment strategy to define how your application is rolled out. The `runOnce` strategy and the `rolling` strategy for VMs are both supported. [Here](#) is the reference documentation for deployment strategies and the details about various life-cycle hooks.

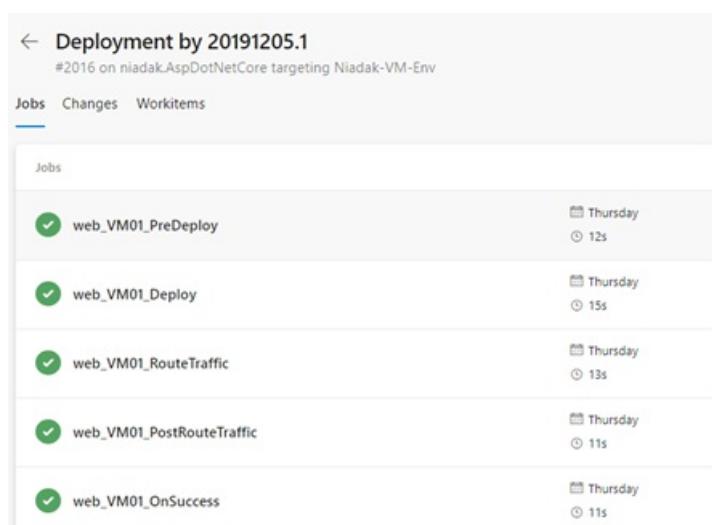
Deployment history views

The **Deployments** tab provides complete traceability of commits and work items, and a cross-pipeline deployment history per environment and resource.



The screenshot shows the Azure Pipelines interface for the 'VMEnv' environment. The top navigation bar includes links for IgniteDemo2019, Pipelines, Environments, and VMEnv. A search bar and several icons are also present. Below the navigation, there are tabs for 'Resources' and 'Deployments', with 'Deployments' being the active tab. A blue 'Add resource' button is located in the top right of this section. The main area displays a table of deployment runs:

Run	Jobs	Date	Duration
Update rolling-deployment.yml for Azure Pipelines #20191209.1 on niadak.AspDotNetCore	VMDeploy	Monday	<1s
Update rolling-deployment.yml for Azure Pipelines #20191206.1 on niadak.AspDotNetCore	VMDeploy	Friday	<1s
Update rolling-deployment.yml for Azure Pipelines #20191205.1 on niadak.AspDotNetCore	VMDeploy	Thursday	<1s



The screenshot shows the detailed deployment history for run #20191205.1. The title indicates it's for 'Deployment by 20191205.1' on 'niadak.AspDotNetCore' targeting 'Niadak-VM-Env'. The 'Jobs' tab is selected. The table lists the individual deployment jobs:

Job	Date	Duration
web_VM01_PreDeploy	Thursday	12s
web_VM01_Deploy	Thursday	15s
web_VM01_RouteTraffic	Thursday	13s
web_VM01_PostRouteTraffic	Thursday	11s
web_VM01_OnSuccess	Thursday	11s

Remove a virtual machine from an environment

To remove virtual machines that were previously added to a Windows environment, run this command from an administrator PowerShell command prompt on each machine. You'll need to run the command in the same folder path where the script to register to the environment has been previously run:

```
./config.cmd remove
```

To remove a virtual machine from a Linux environment, run this command on each machine.

```
./config.sh remove
```

Known limitations

When you retry a stage, it will rerun the deployment on all virtual machines and not just failed targets.

Next steps

- Learn more about [deployment jobs](#) and [environments](#).
- See the [YAML schema reference](#) to learn what else you can do with YAML pipelines.

Deploy to a Linux Virtual Machine

3/6/2021 • 7 minutes to read • [Edit Online](#)

NOTE

If you want to deploy your application to a Linux virtual machine using the classic editor, see [Deploy web apps to Linux VMs](#).

Azure Pipelines provides a complete, fully featured set of CI/CD automation tools for deployments to virtual machines.

You can use continuous integration (CI) and continuous deployment (CD) to build, release, and deploy your code. Learn how to set up a CI/CD pipeline for multi-machine deployments.

This article covers how to set up continuous deployment of your app to a web server running on Ubuntu. You can use these steps for any app that publishes a web deployment package.

Get your sample code

- [Java](#)
- [JavaScript](#)

If you already have an app in GitHub that you want to deploy, you can create a pipeline for that code.

If you are a new user, fork this repo in GitHub:

```
https://github.com/spring-projects/spring-petclinic
```

NOTE

Petclinic is a [Spring Boot](#) application built using [Maven](#).

Prerequisites for the Linux VM

Use Ubuntu 16.04 for this quickstart. Follow additional steps for Java or JavaScript.

- [Java](#)
 - [JavaScript](#)
- For deploying Java Spring Boot and Spring Cloud based apps, create a Linux VM in Azure using [this template](#), which provides a fully supported OpenJDK-based runtime.
 - For deploying Java servlets on Tomcat server, create a Linux VM with Java 8 using [this Azure template](#) and [configure Tomcat 9.x as a service](#).
 - For deploying Java EE-based Wildfly app, follow the [blog post](#) here. To provision the VM, use an Azure template to create a [Linux VM + Java + WebSphere 9.x](#) or a [Linux VM + Java + WebLogic 12.x](#) or a [Linux VM + Java + WildFly/JBoss 14](#)

Create an environment with virtual machines

Virtual machines can be added as resources within [environments](#) and can be targeted for multi-VM deployments. The deployment history view provides traceability from the VM to the commit.

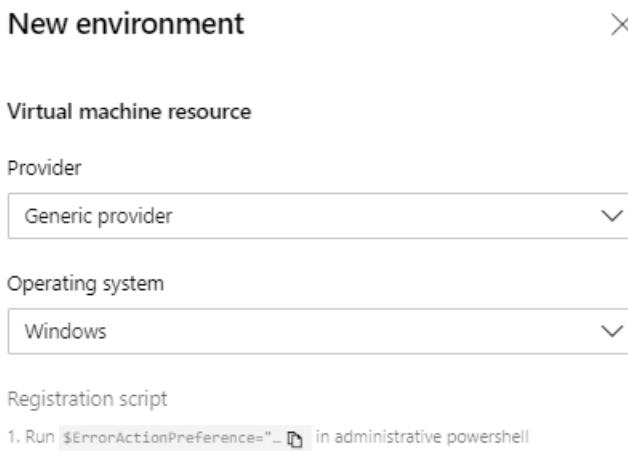
You can create an environment in **Environments** within **Pipelines**.

1. Sign into your Azure DevOps organization and navigate to your project.
2. Navigate to the Pipelines page. Select **Environments** and click **Create Environment**. Specify a **Name** (required) for the environment and a **Description**.
3. Choose **Virtual Machines** as a **Resource** to be added to the environment. Click **Next**.
4. Choose the Windows or Linux for the **Operating System** and copy PS registration script.
5. Run the copied script from an administrator PowerShell command prompt on each of the target VMs registered with this environment.

NOTE

- The Personal Access Token (PAT) of the logged in user is pre-inserted in the script and expires after three hours.
- If your VM already has any agent running on it, provide a unique name to register with environment.

6. Once VM is registered, it will start appearing as an environment resource under **Resources**.



7. To add more VMs, copy the script again. Click **Add resource** and choose **Virtual Machines**. This script is the same for all the VMs you want to add to the same environment.
8. Each machine interacts with Azure Pipelines to coordinate deployment of your app.

The screenshot shows the 'VMenv' resource list. At the top, there are tabs for 'Resources' and 'Deployments', with 'Resources' being the active tab. Below the tabs, there's a table with two columns: 'Name' and 'Latest job'. A single row is shown for a VM named 'USHAN-PC', which has the status 'Never deployed'.

9. You can add or remove tags for the VM. Click on the dots at the end of each VM resource in **Resources**. The tags you assign allow you to limit deployment to specific VMs when the environment is used in a deployment job. Tags are each limited to 256 characters, but there is no limit to the number of tags you can create.

The screenshot shows the Azure DevOps interface for managing environments. In the top navigation bar, there are links for 'Environments' and 'VMenv'. A search bar and various navigation icons are also present. Below the navigation, the 'VMenv' environment is selected. On the right, there are buttons for 'Add resource' and a three-dot menu. Under the 'Resources' tab, a table lists a single item: 'USHAN-PC' with the status 'Never deployed'. A modal window titled 'Manage tags for USHAN-PC' is open, showing a list box containing the tag 'web' and a '+' button. At the bottom of the modal are 'Cancel' and 'Save' buttons.

Define your CI build pipeline

You'll need a continuous integration (CI) build pipeline that publishes your web application and a deployment script that can be run locally on the Ubuntu server. Set up a CI build pipeline based on the runtime you want to use.

1. Sign in to your Azure DevOps organization and navigate to your project.
2. In your project, navigate to the **Pipelines** page. Then choose the action to create a new pipeline.
3. Walk through the steps of the wizard by first selecting **GitHub** as the location of your source code.
4. You may be redirected to GitHub to sign in. If so, enter your GitHub credentials.
5. When the list of repositories appears, select your desired sample app repository.
6. Azure Pipelines will analyze your repository and recommend a suitable pipeline template.

- [Java](#)
- [JavaScript](#)

Select the **starter** template and copy this YAML snippet to build your Java project and runs tests with Apache Maven:

```
- job: Build
  displayName: Build Maven Project
  steps:
    - task: Maven@3
      displayName: 'Maven Package'
      inputs:
        mavenPomFile: 'pom.xml'
    - task: CopyFiles@2
      displayName: 'Copy Files to artifact staging directory'
      inputs:
        SourceFolder: '$(System.DefaultWorkingDirectory)'
        Contents: '**/target/*.{war,jar}'
        TargetFolder: $(Build.ArtifactStagingDirectory)
    - upload: $(Build.ArtifactStagingDirectory)
      artifact: drop
```

For more guidance, follow the steps mentioned in [Build your Java app with Maven](#) for creating a build.

Define CD steps to deploy to the Linux VM

1. Edit your pipeline and include a [deployment job](#) by referencing the environment and the VM resources you created earlier:

```
jobs:  
- deployment: VMDeploy  
  displayName: web  
  environment:  
    name: <environment name>  
    resourceType: VirtualMachine  
    tags: web1  
  strategy:
```

2. You can select specific sets of virtual machines from the environment to receive the deployment by specifying the [tags](#) that you have defined for each virtual machine in the environment. [Here](#) is the complete YAML schema for Deployment job.

3. You can specify either [runOnce](#) or [rolling](#) as a deployment strategy.

[runOnce](#) is the simplest deployment strategy. All the life-cycle hooks, namely [preDeploy](#), [deploy](#), [routeTraffic](#), and [postRouteTraffic](#), are executed once. Then, either [on: success](#) or [on: failure](#) is executed.

Below is an example YAML snippet for [runOnce](#):

```
jobs:  
- deployment: VMDeploy  
  displayName: web  
  pool:  
    vmImage: 'Ubuntu-16.04'  
  environment:  
    name: <environment name>  
    resourceType: VirtualMachine  
  strategy:  
    runOnce:  
      deploy:  
        steps:  
        - script: echo my first deployment
```

4. Below is an example YAML snippet for the rolling strategy. You can update up to 5 targets gets in each iteration. [maxParallel](#) will determine the number of targets that can be deployed to, in parallel. The selection accounts for absolute number or percentage of targets that must remain available at any time excluding the targets that are being deployed to. It is also used to determine the success and failure conditions during deployment.

```

jobs:
- deployment: VMDeploy
  displayName: web
  environment:
    name: <environment name>
    resourceType: VirtualMachine
  strategy:
    rolling:
      maxParallel: 2 #for percentages, mention as x%
      preDeploy:
        steps:
          - download: current
            artifact: drop
          - script: echo initialize, cleanup, backup, install certs
    deploy:
      steps:
        - task: Bash@3
          inputs:
            targetType: 'inline'
            script: |
              # Modify deployment script based on the app type
              echo "Starting deployment script run"
              sudo java -jar '$(Pipeline.Workspace)/drop/**/target/*.jar'
  routeTraffic:
    steps:
      - script: echo routing traffic
  postRouteTraffic:
    steps:
      - script: echo health check post-route traffic
  on:
    failure:
      steps:
        - script: echo Restore from backup! This is on failure
    success:
      steps:
        - script: echo Notify! This is on success

```

With each run of this job, deployment history is recorded against the <environment name> environment that you have created and registered the VMs.

Pipeline traceability views in environment

The **Deployments** view provides complete traceability of commits and work items, and a cross-pipeline deployment history per environment.

Run	Jobs	Date	Duration
Update rolling-deployment.yml for Azure Pipelines #20191209.1 on niadak.AspNetCore	VMDeploy	Monday	<1s
Update rolling-deployment.yml for Azure Pipelines #20191206.1 on niadak.AspNetCore	VMDeploy	Friday	<1s
Update rolling-deployment.yml for Azure Pipelines #20191205.1 on niadak.AspNetCore	VMDeploy	Thursday	<1s

← Deployment by 20191205.1
#2016 on niadak.AspNetCore targeting Niadak-VM-Env

Jobs Changes Workitems

Jobs

	web_VM01_PreDeploy	Thursday 12s
	web_VM01_Deploy	Thursday 15s
	web_VM01_RouteTraffic	Thursday 13s
	web_VM01_PostRouteTraffic	Thursday 11s
	web_VM01_OnSuccess	Thursday 11s

Next Steps

To learn more about the topics in this guide see [Jobs](#), [Tasks](#), [Catalog of Tasks](#), [Variables](#), [Triggers](#), or [Troubleshooting](#).

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

Manage a virtual machine in Azure DevTest Labs

11/2/2020 • 9 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

The [Azure DevTest Labs](#) service lets you quickly provision development and test stages using reusable templates. You can use pre-created images, minimize waste with quotas and policies, and minimize costs by using automated shutdown.

By using an extension installed in Azure Pipelines or Team Foundation Server (TFS) you can easily integrate your build and release pipeline with Azure DevTest Labs. The extension installs three tasks to create a VM, create a custom image from a VM, and delete a VM. This makes it easy to, for example, quickly deploy a "golden image" for specific test task, then delete it when the test is finished.

This example shows how to create and deploy a VM, create a custom image, then delete the VM. It does so as one complete pipeline, though in reality you would use the tasks individually in your own custom build-test-deploy pipeline.

Get set up

Start by installing the [Azure DevTest Labs Tasks](#) extension from Visual Studio Marketplace, Azure DevOps tab:

- For Azure Pipelines, choose **Install**
- For TFS, choose **Download** and install the extension on your server.

Create an Azure RM template

Carry out these tasks to create the Azure Resource Manager (ARM) template that you can use to create an Azure Virtual Machine on demand.

1. Follow the steps in [these documents](#) on the Azure website to create an ARM template in your subscription.
2. Follow the steps in [these documents](#) on the Azure website to save the ARM template as a file on your computer. Name the file **CreateVMTemplate.json**.
3. Edit the **CreateVMTemplate.json** file as described in [this post](#) on Tarun Arora's blog to configure it for Windows Remote Management (WinRM).

WinRM access is required to use deploy tasks such as **Azure File Copy** and **PowerShell on Target Machines**.

4. Check the template into your source control system.
5. Open a text editor and copy the following script into it.

```

Param( [string] $labVmId)

$labVmComputeId = (Get-AzureRmResource -Id $labVmId).Properties.ComputeId

# Get lab VM resource group name
$labVmRgName = (Get-AzureRmResource -Id $labVmComputeId).ResourceGroupName

# Get the lab VM Name
$labVmName = (Get-AzureRmResource -Id $labVmId).Name

# Get lab VM public IP address
$labVMIpAddress = (Get-AzureRmPublicIpAddress -ResourceGroupName $labVmRgName
                    -Name $labVmName).IpAddress

# Get lab VM FQDN
$labVMFqdn = (Get-AzureRmPublicIpAddress -ResourceGroupName $labVmRgName
                  -Name $labVmName).DnsSettings.Fqdn

# Set a variable labVmRgName to store the lab VM resource group name
Write-Host "##vso[task.setvariable variable=labVmRgName;]$labVmRgName"

# Set a variable labVMIpAddress to store the lab VM Ip address
Write-Host "##vso[task.setvariable variable=labVMIpAddress;]$labVMIpAddress"

# Set a variable labVMFqdn to store the lab VM FQDN name
Write-Host "##vso[task.setvariable variable=labVMFqdn;]$labVMFqdn"

```

6. Check the script into your source control system. Name it something like **GetLabVMParams.ps1**.

This script, when run on the agent as part of the release pipeline, collects values that you'll need to deploy your app to the VM if you use task steps such as **Azure File Copy or PowerShell on Target Machines**. These are the tasks you typically use to deploy apps to an Azure VM, and they require values such as the VM Resource Group name, IP address, and fully-qualified domain name (FQDN).

Deploy

Carry out the following steps to create the release pipeline in Azure Pipelines.

1. Open the **Releases** tab of **Azure Pipelines** and choose the "+" icon to create a new release pipeline.
2. In the **Create release pipeline** dialog, select the **Empty** template and choose **Next**.
3. In the next page, select **Choose Later** and then choose **Create**. This creates a new release pipeline with one default stage and no linked artifacts.
4. In the new release pipeline, choose the ellipses (...) next to the stage name to open the shortcut menu and select **Configure variables**.
5. In the **Configure - stage** dialog, enter the following values for the variables you will use in the release pipeline tasks:
 - **vmName**: Enter the name you assigned to the VM when you created the ARM template in the Azure portal.
 - **userName**: Enter the username you assigned to the VM when you created the ARM template in the Azure portal.
 - **password**: Enter the password you assigned to the VM when you created the ARM template in the Azure portal. Use the "padlock" icon to hide and secure the password.
6. The first stage in this deployment is to create the VM you will use as the "golden image" for subsequent deployments. You create this within your Azure DevTestsLab instance using the task specially developed for this purpose. In the release pipeline, select + **Add tasks** and add an **Azure DevTest Labs Create**

VM task from the Deploy tab.

7. Configure the Azure DevTest Labs Create VM task as follows:



Azure DevTest Labs Tasks - Create the VM to use for subsequent deployments.

- **Azure RM Subscription:** Select a connection from the list under Available Azure Service Connections or create a more restricted permissions connection to your Azure subscription. For more details, see [Azure Resource Manager service connection](#).
- **Lab Name:** Select the name of the instance you created earlier.
- **Template Name:** Enter the full path and name of the template file you saved into your source code repository. You can use the built-in properties of Azure Pipelines to simplify the path, for example: `$(System.DefaultWorkingDirectory)/Contoso/ARMTemplates/CreateVMTemplate.json`.
- **Template Parameters:** Enter the parameters for the variables defined in the template. Use the names of the variables you defined in the stage, for example:
`-newVMName '$(vmName)' -userName '$(userName)' -password (ConvertTo-SecureString -String '$(password)' -AsPlainText -Force)`
- **Output Variables - Lab VM ID:** You will need the ID of the newly created VM in subsequent tasks. The default name of the stage variable that will automatically be populated with this ID is set in the Output Variables section. You can edit this if required, but remember to use the correct name in subsequent tasks. The Lab VM ID is in the form:
`/subscriptions/{subId}/resourceGroups/{rgName}/providers/Microsoft.DevTestLab/labs/{labName}/virtualMachines/{vmName}`

8. The next stage is to execute the script you created earlier to collect the details of the DevTest Labs VM. In the release pipeline, select + Add tasks and add an Azure PowerShell task from the Deploy tab. Configure the task as follows:



Deploy: Azure PowerShell - Execute the script to collect the details of the DevTest Labs VM.

- **Azure Connection Type:** `Azure Resource Manager`.
- **Azure RM Subscription:** Select a connection from the list under Available Azure Service Connections or create a more restricted permissions connection to your Azure subscription. For more details, see [Azure Resource Manager service connection](#).
- **Script Type:** `Script File`.
- **Script Path:** Enter the full path and name of the script you saved into your source code repository. You can use the built-in properties of Azure Pipelines to simplify the path, for example:
`$(System.DefaultWorkingDirectory)/Contoso/Scripts/GetLabVMParams.ps1`.
- **Script Arguments:** Enter as the script argument the name of the stage variable that was automatically populated with the ID of the lab VM by the previous task, for example:
`-labVmId '$(labVmId)' .|`

The script collects the values you will require and stores them in stage variables within the release pipeline so that you can easily refer to them in subsequent tasks.

9. Now you can deploy your app to the new DevTest Labs VM. The tasks you will typically use for this are **Azure File Copy and PowerShell on Target Machines**.

- The information about the VM you'll need for the parameters of these tasks is stored in three configuration variables named **labVmRgName**, **labVMIpAddress**, and **labVMFqdn** within the release pipeline.

- If you just want to experiment with creating a DevTest Labs VM and a custom image, without deploying an app to it, just skip this step.

10. The next stage is to create an image of the newly deployed VM in your Azure DevTest Labs instance. You can then use this image to create copies of the VM on demand, whenever you want to execute a dev task or run some tests. In the release pipeline, select + **Add tasks** and add an **Azure DevTest Labs Create Custom Image** task from the **Deploy** tab. Configure it as follows:



[Azure DevTest Labs Tasks](#) - Create an image of the VM.

- **Azure RM Subscription:** Select a connection from the list under **Available Azure Service Connections** or create a more restricted permissions connection to your Azure subscription. For more details, see [Azure Resource Manager service connection](#).
- **Lab Name:** Select the name of the instance you created earlier.
- **Custom Image Name:** Enter a name for the custom image you will create.
- **Description:** Optionally enter a description to make it easy to select the correct image later.
- **Source Lab VM - Source Lab VM ID:** If you changed the default name of the stage variable that was automatically populated with the ID of the lab VM by an earlier task, edit it here. The default is `$(labVMId)`.
- **Output Variables - Lab VM ID:** You will need the ID of the newly created image when you want to manage or delete it. The default name of the stage variable that will automatically be populated with this ID is set in the **Output Variables** section. You can edit this if required.

11. The final stage in this example is to delete the VM you deployed in your Azure DevTest Labs instance. In reality you will, of course, do this *after* you execute the dev tasks or run the tests you need on the deployed VM. In the release pipeline, select + **Add tasks** and add an **Azure DevTest Labs Delete VM** task from the **Deploy** tab. Configure it as follows:



[Azure DevTest Labs Tasks](#) - Delete the VM.

- **Azure RM Subscription:** Select a connection from the list under **Available Azure Service Connections** or create a more restricted permissions connection to your Azure subscription. For more details, see [Azure Resource Manager service connection](#).
- **Lab VM ID:** If you changed the default name of the stage variable that was automatically populated with the ID of the lab VM by an earlier task, edit it here. The default is `$(labVMId)`.

12. Enter a name for the release pipeline and save it.

13. Create a new release, select the latest build, and deploy it to the single stage in the pipeline.

14. At each stage, refresh the view of your DevTest Labs instance in the Azure portal to see the VM and image being created, and the VM being deleted again. You can now use the custom image to create VMs when required.

For more information, or if you have any suggestions for improvements to the extension, visit the [DevTest Labs feedback forum](#).

FAQ

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
- Get advice on [Stack Overflow](#), and feel free to post your questions, search for answers, or suggest a feature on our [Azure DevOps Developer Community](#). [Support](#) page.

Sign your mobile app

5/27/2021 • 9 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | [TFS 2017.2](#)

When developing an app for Android or Apple operating systems, you will eventually need to manage signing certificates, and in the case of Apple apps, [provisioning profiles](#). This article describes how to securely manage them for signing and provisioning your app.

TIP

Use a Microsoft-hosted Linux, macOS, or Windows build agent, or set up your own agent. See [Build and release agents](#).

This article covers:

- [Sign your Android app](#)
- [Sign your Apple iOS, macOS, tvOS, or watchOS app](#)

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Sign your Android app

Follow these steps to sign your Android app while keeping your signing certificate secure:

1. First, obtain a keystore file that contains your signing certificate. The [Android documentation](#) describes the process of generating a keystore file and its corresponding key.
2. Create your build pipeline from the Android or Xamarin.Android build template. Or, if you already have a build pipeline, add the [Android Signing](#) task after the task that builds your APK.
3. Find the Android Signing task's **Sign the APK** checkbox and enable it.
4. Next to the **Keystore file** field, click the settings icon and upload your keystore file to the [Secure Files library](#). During upload, your keystore will be encrypted and securely stored.
5. Once your keystore has been uploaded to the Secure Files library, select it in the **Keystore file** dropdown.
6. Go to the **Variables** tab and add the following variables. In their **Value** column, enter your **Keystore password**, **Key alias**, and **Key password**.
 - **keystore-password**: Password to the unencrypted keystore file. *Be sure to click the lock icon.* This will secure your password and obscure it in logs.
 - **key-alias**: The key alias for the signing certificate you generated.
 - **key-password**: The password for the key associated with the specified alias. *Again, be sure to click the lock icon.*

Name ↑	Value	Settable at queue time
system.collectionId	54d02617-2ebd-42b0-b1e2-257059c4c03d	
system.definitionId	< No definition id yet >	
system.teamProject	CanaryBuilds	
system.debug	true	<input checked="" type="checkbox"/>
keystore-password	*****	
key-alias	MyKeyAlias	
key-password	*****	<input type="checkbox"/> <input checked="" type="checkbox"/>

[+ Add](#)

7. Go back to the **Tasks** tab and reference the names of your newly-created variables in the signing options.

Signing Options ^

Sign the APK [i](#)

Keystore File * [i](#)

AndroidApp.keystore [▼](#) [↻](#) [⚙](#)

Keystore Password [i](#)
\$(keystore-password)

Alias [i](#)
\$(key-alias)

Key Password [i](#)
\$(key-password)

Save your build pipeline, and you are all set! Any build agent will now be able to securely sign your app without any certificate management on the build machine itself.

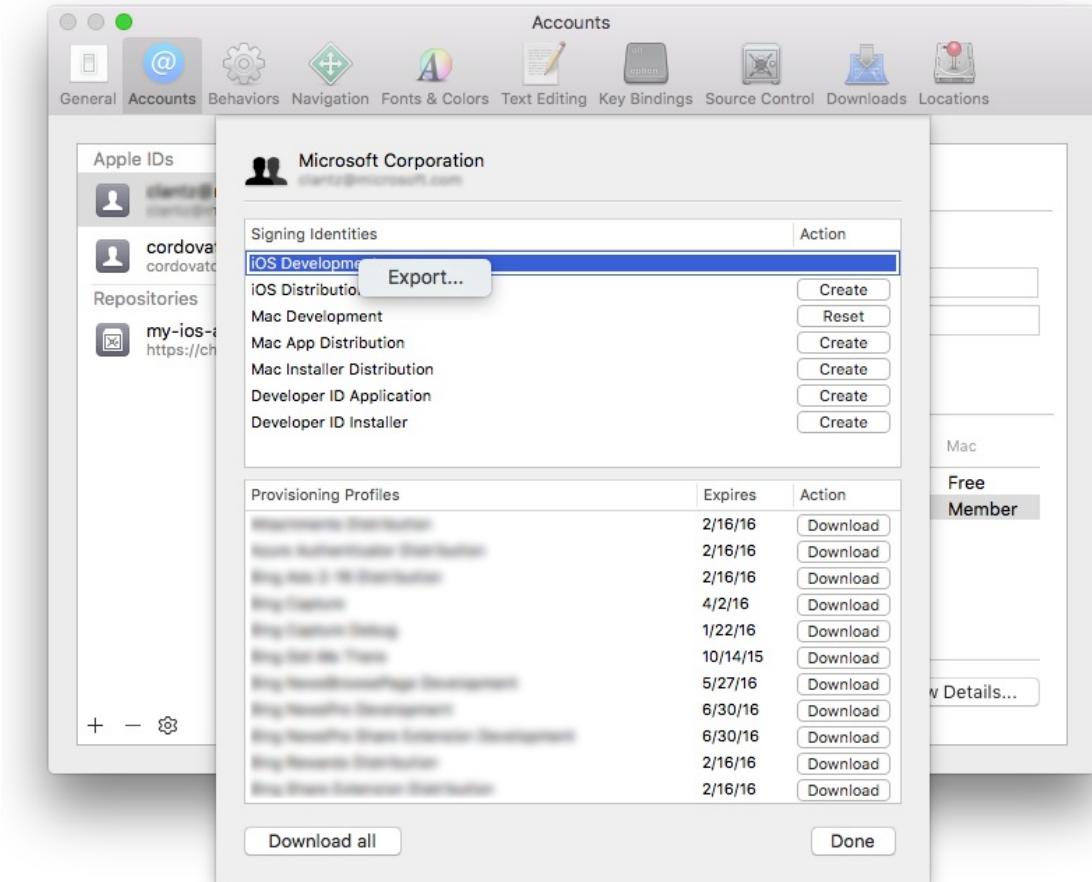
Sign your Apple iOS, macOS, tvOS, or watchOS app

For your Xcode or Xamarin.iOS build to sign and provision your app, it needs access to your P12 signing certificate and one or more provisioning profiles. The following sections explain how to obtain these files.

Obtain your P12 signing certificate

After creating your development or distribution signing certificate, export it to a `.p12` file using either Xcode or the Keychain Access app on macOS.

1. To export using Xcode 8 or lower, go to **Xcode > Preferences... > Accounts** and select your Apple Developer account.
2. Click **View Details...**, right-click on the signing identity you wish to export, and click **Export....**
3. Enter a filename and password. Take note of the password as you will need it later.

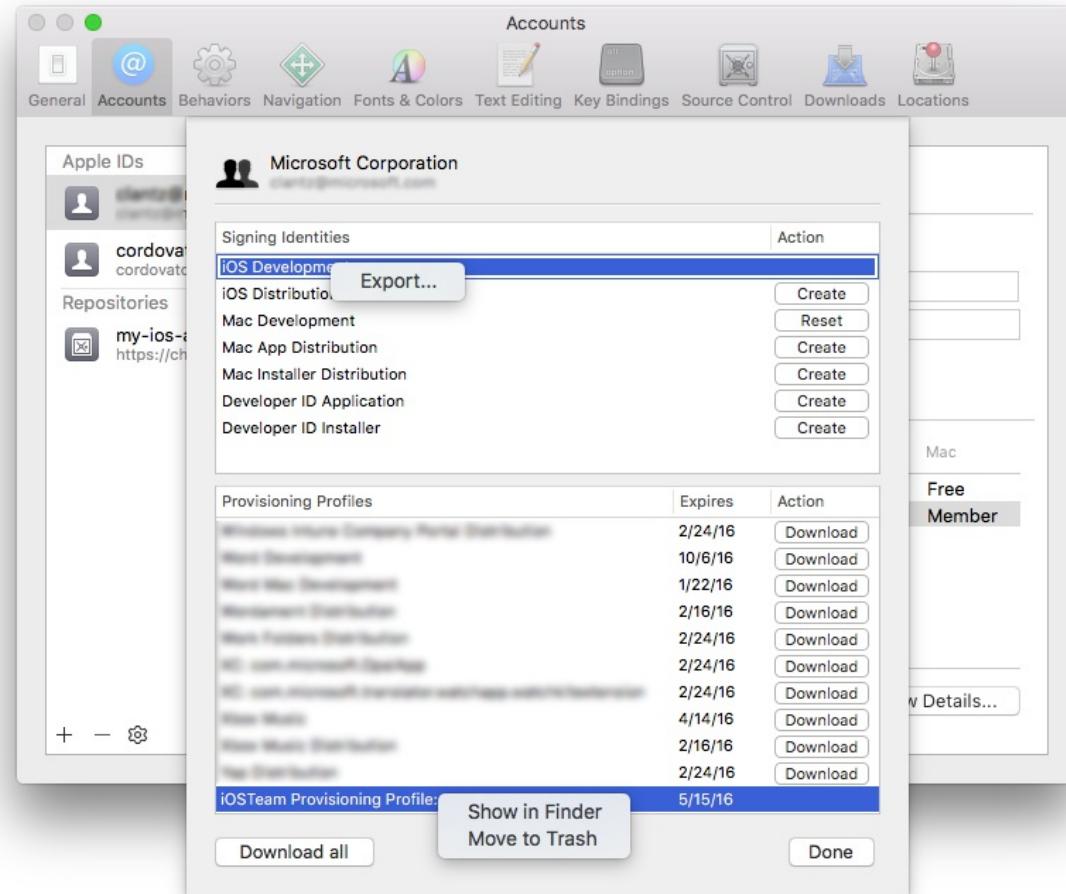


4. Alternatively, follow a similar process using the **Keychain Access** app on macOS or generate a signing certificate on Windows. Use the procedure [described in this article](#) if you prefer this method.

Obtain your provisioning profile

You can download your app provisioning profile from the Apple Developer portal, unless your app uses automatic signing. You can also use Xcode to access those that are installed on your Mac.

1. Using Xcode 8 or lower, go to **Xcode > Preferences... > Accounts** and select your Apple Developer account.
2. Right-click the provisioning profile you want to use and select **Show in Finder**.
3. Copy the highlighted file from Finder to another location and give it a descriptive filename.



Configure your build

There are two recommended ways for your build to access signing certificates and provisioning profiles for signing and provisioning your app:

1. Installing them during the build
2. Preinstalling them on a macOS build agent

Choose either of the tabs below for details.

- [Install them during the build](#)
- [Preinstall them on a macOS build agent](#)

Use this method when you do not have enduring access to the build agent, such as the [hosted macOS agents](#). The P12 certificate and provisioning profile are installed at the beginning of the build and removed when the build completes.

Install the P12 certificate during your build

Visual Editor

1. Add the [Install Apple Certificate](#) task to your build before the Xcode or Xamarin.iOS task.
2. Next to the **Certificate (P12)** field, click the settings icon and upload your P12 file to the [Secure Files library](#). During upload, your certificate will be encrypted and securely stored.
3. Once your certificate has been uploaded to the Secure Files library, select it in the **Certificate (P12)** dropdown.
4. Go to the **Variables** tab and add a variable named `P12password`. Set its value to the password of your certificate. *Be sure to click the lock icon*. This will secure your password and obscure it in logs.
5. Go back to the **Tasks** tab. In the [Install Apple Certificate](#) task's settings, reference your newly-created variable in the **Certificate (P12) password** field as: `$(P12password)`

Sample YAML

1. Upload your P12 file to the [Secure Files library](#). During upload, your certificate will be encrypted and

securely stored.

2. Go to the **Variables** tab and add a variable named `P12password`. Set its value to the password of your certificate. *Be sure to click the lock icon.* This will secure your password and obscure it in logs.
3. Add the [Install Apple Certificate](#) task to your YAML before the Xcode or Xamarin.iOS task:

```
- task: InstallAppleCertificate@2
  inputs:
    certSecureFile: 'my-secure-file.p12' # replace my-secure-file.p12 with the name of your P12 file.
    certPwd: '$(P12password)'
```

Install the provisioning profile during your build

Visual Editor

1. Add the [Install Apple Provisioning Profile](#) task to your build before the Xcode or Xamarin.iOS task.
2. For the **Provisioning profile location** option, choose **Secure Files** (in YAML, `secureFiles`).
3. Next to the **Provisioning profile** field, click the settings icon and upload your provisioning profile file to the [Secure Files library](#). During upload, your certificate will be encrypted and securely stored.
4. Once your certificate has been uploaded to the Secure Files library, select it in the **Provisioning profile** dropdown.
5. Enable the checkbox labeled **Remove profile after build**. This will ensure that the provisioning profile is not left on the agent machine.

Sample YAML

1. Upload your provisioning profile to the [Secure Files library](#). During upload, your certificate will be encrypted and securely stored.
2. Add the [Install Apple Provisioning Profile](#) task to your YAML before the Xcode or Xamarin.iOS task:

```
- task: InstallAppleProvisioningProfile@1
  inputs:
    provProfileSecureFile: 'my-provisioning-profile.mobileprovision' # replace my-provisioning-profile.mobileprovision with the name of your provisioning profile file.
```

[NOTE] Remove profile after build defaults to *true*.

Reference the files in your Xcode task

Visual Editor

1. Select the **Xcode** task.
2. For the **Signing style** option, choose **Manual signing**.
3. In the **Signing identity** field, enter `$(APPLE_CERTIFICATE_SIGNING_IDENTITY)`. This variable is automatically set by the [Install Apple Certificate](#) task for the certificate you selected.
4. In the **Provisioning profile UUID** field, enter `$(APPLE_PROV_PROFILE_UUID)`. This variable is automatically set by the [Install Apple Provisioning Profile](#) task for the provisioning profile you selected.

Sample YAML

```
- task: Xcode@5
  inputs:
    signingOption: 'manual'
    signingIdentity: '$(APPLE_CERTIFICATE_SIGNING_IDENTITY)'
    provisioningProfileUuid: '$(APPLE_PROV_PROFILE_UUID)'
```

Reference the files in your Xamarin.iOS task

Visual Editor

1. Select the **Xamarin.iOS** task.
2. For the **Override using** option, choose **Identifiers**.
3. In the **Signing identity** field, enter `$(APPLE_CERTIFICATE_SIGNING_IDENTITY)`. This variable is automatically set by the **Install Apple Certificate** task for the certificate you selected.
4. In the **Provisioning profile UUID** field, enter `$(APPLE_PROV_PROFILE_UUID)`. This variable is automatically set by the **Install Apple Provisioning Profile** task for the provisioning profile you selected.

Sample YAML

```
- task: XamariniOS@2
  inputs:
    solutionFile: '**/*.iOS.csproj'
    signingIdentity: '$(APPLE_CERTIFICATE_SIGNING_IDENTITY)'
    signingProvisioningProfileID: '$(APPLE_PROV_PROFILE_UUID)'
```

Save your build pipeline, and you are all set! The build agent will now be able to securely sign and provision your app.

FAQ

Do I need an agent?

You need at least one [agent](#) to run your build or release.

I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

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](#).

My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

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](#).

Sign your mobile app in your Xcode, Xamarin.iOS, Android, or Xamarin.Android build

5/27/2021 • 10 minutes to read • [Edit Online](#)

TFS 2017.2

This article highlights how to manage and secure profiles for your app. You'll need to manage signing certificates if you're developing an app for Android or iOS. For iOS, you'll use [provisioning profiles](#).

NOTE

For Microsoft Team Foundation Server (TFS) 2018 and previous versions. When building on macOS, you should either set up the [agent](#) as a launchd service (`./svc.sh install`) or run it as an interactive process (`./run.sh`).

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

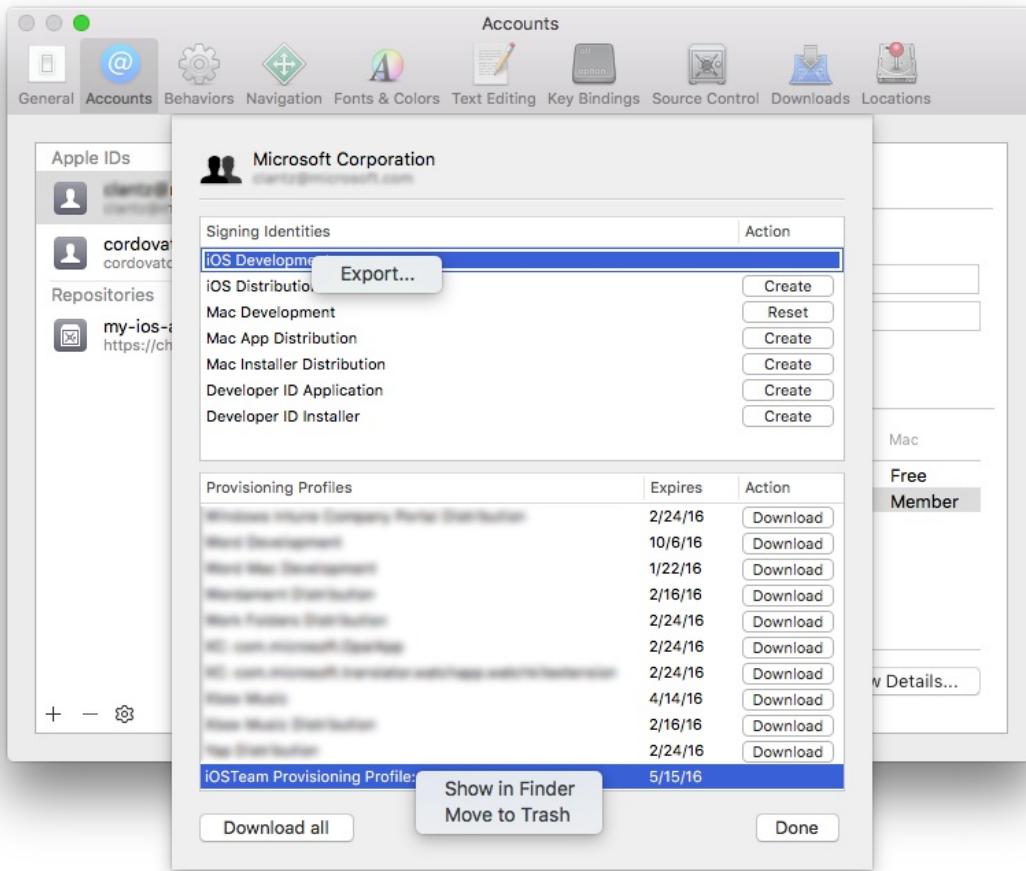
This article covers:

- [Using a certificate and mobile provisioning profile in your Xcode build](#)
- [Using preinstalled signing identities and mobile provisioning profiles in your Xcode or Xamarin.iOS build](#)
- [Building and signing Android and Xamarin.Android projects](#)

Using a certificate and mobile provisioning file in your Xcode build

You can use the Keychain Access to generate a certificate. To learn more about this option, [see iOS Signing](#).

1. Export your signing certificate to a .p12 file using Keychain Access or Xcode. A .p12 file contains the certificates Apple needs to build and publish apps.
 - a. Open **Keychain Access** on your computer.
 - b. In the **Keychain Access** menu, open **Certification Assistant > Request a Certificate From a Certificate Authority**.
 - c. Enter your email address and name. These values should match what you used for your Apple Developer account.
 - d. Save to disk. You can leave the **CA Email Address** field empty.
2. Next, get a copy of the provisioning profile you want to use.
 - a. You can download your mobile provisioning profile from the Apple Developer portal, unless your app uses automatic signing. You can also use Xcode to access one installed on your machine.
 - b. First go to the same screen you used to get the .p12 file above.
 - c. Next, right-click the provisioning profile you want and select **Show in Finder**.
 - d. Copy the file highlighted to another location and give it a descriptive filename.



3. At this point, you can opt to add these two files directly to source control or add an extra layer of security by encrypting them first. **See the next section for details on this step.**
4. Next, open your Xcode or Xamarin.iOS build pipeline and go to the **Variables** tab. Here, enter the password for the .p12 file:

- **P12_PWD:** Password to the .p12 file in its unencrypted form. *Be sure to select the "lock" icon.* Locking will secure your password and obscure it in all logs.

Name	Value	Allow at Queue Time
system.collectionId	85f58b25-5e4b-404f-82ce-93fc14939153	<input type="checkbox"/>
system.teamProject	tfvnext-test	<input type="checkbox"/>
system.definitionId	36	<input type="checkbox"/>
system.debug	true	<input type="checkbox"/>
Configuration	Debug	<input type="checkbox"/>
P12_PWD	*****	<input checked="" type="checkbox"/>
Add variable		

5. Finally, update your Xcode or Xamarin.iOS step with references to these two files. The build step will automatically determine the correct "signing identity" based on the contents of the .p12, create a temporary keychain with the .p12 in it and use that exclusively for the build. The build step will also install the mobile provisioning profile, determine the correct UUID based on the contents of the .mobileprovision profile, and then remove everything after the build.

Enter these values under the **Signing & Provisioning** category for the Xcode or Xamarin.iOS build task:

- **Override Using:** File Contents
- **P12 Certificate File:** Path to the .p12 file in the repository

- **P12 Password:** \$(P12_PWD)
- **Provisioning Profile File:** Path to the .mobileprovision file in the repository
- **Remove Profile After Build:** Checked if you want the provisioning profile to be removed from the system after the build. Only check this if you will have one agent that only runs this build, as it is installed in a global location and could be accessed by other builds.

The screenshot shows the 'Xcode build' configuration screen. It includes fields for Actions (set to 'build'), Configuration (set to '\$(Configuration)'), SDK (set to '\$(SDK)'), Workspace Path (set to '*.{xcproj, xcworkspace}'), Scheme (set to 'MyHealthClinic'), and Create App Package (checked). Below this, the 'Signing & Provisioning' section is expanded, showing settings for Override Using (radio button selected for 'File Contents'), P12 Certificate File (_build.p12), P12 Password (\$P12_PWD), Provisioning Profile File (_build.mobileprovision), and Remove Profile After Build (checked).

You are now all set! Any build agent that is running will now be able to build your app without any certificate management on the build machine itself. Repeat the process of adding different certificates and provisioning profiles to your source repository to enable separate dev, beta (ad hoc), and distribution builds.

In the next section, we'll cover how to add another layer of security by encrypting these files.

Optional: Using an encrypted .p12 and .mobileprovision file

You can add an extra layer of security by to your project by encrypting your .p12 and .mobileprovision files so that only those that have been given the appropriate credentials have permission to use them. The **Decrypt File** task makes this easy to set up in your build task.

1. First, encrypt your files.
 - a. Open the Terminal app on your Mac.
 - b. Change to the folder containing your .p12 file and .mobileprovision file.
 - c. Type the following:

```
openssl des3 -in iphonedev.p12 -out iphonedev.p12.enc
```

substituting des3 for any encryption cypher you'd like to use and the appropriate input and output file names.

 - d. Enter a passphrase to encrypt the certificate when prompted and note this as you will use it later.
2. Next, add these two files to source control (the encrypted .p12 and .mobileprovision files). This is secure particularly if you are using a private repository since a malicious user would need access to the repository, the encryption passphrase, and the P12 passphrase to be able to use your information without permission. Remove any unencrypted copies if present.
3. Now, set up your build pipeline to decrypt the files at build time.
 - a. Open your Xcode or Xamarin.iOS build pipeline and go to the **Variables** tab and enter the following:
 - **P12:** Path to your encrypted .p12 file in source control.
 - **P12_PWD:** Password to the unencrypted .p12 file. *Be sure to select the "lock" icon.* This will

secure your password and obscure it in all logs.

- **MOB_PROV:** Path to your encrypted mobile provisioning profile.
- **ENC_PWD:** The passphrase you used to encrypt the .p12 and .mobileprovision files. If you used two different passphrases, you'll need two variables. Again, *be sure to click the "lock" icon*.

Name	Value	Allow at Queue Time
system.collectionId	258eac23-f347-4e02-ba6f-a6e827241994	<input type="checkbox"/>
system.teamProject	My Health Clinic	<input type="checkbox"/>
system.definitionId	16	<input type="checkbox"/>
P12	Mobile_DevOps_-_Dev.enc	<input checked="" type="checkbox"/>
P12_PWD	*****	<input checked="" type="checkbox"/>
MOB_PROV	Mobile_DevOps_Dev.enc	<input checked="" type="checkbox"/>
ENC_PWD	*****	<input type="checkbox"/>
Configuration	Debug	<input checked="" type="checkbox"/>
SDK	iphoneos	<input checked="" type="checkbox"/>
system.debug	true	<input type="checkbox"/>
test_SDK	iphonesimulator	<input checked="" type="checkbox"/>

Add variable

- Under the **Build** tab in your build pipeline, add two **Decrypt File (OpenSSL)** steps and move the steps to the start of your build pipeline.
- Now, enter the proper information using the variables you created above to decrypt the two files to a static filename.

- **Cypher:** The cypher you specified while encrypting. (Ex: `des3`)
- **Encrypted File:** `$(P12)` for one step and `$(MOB_PROV)` for the other
- **Passphrase:** `$(ENC_PWD)`
- **Decrypted File Path:** `_build.p12` for one step and `_build.mobileprovision` for the other

Decrypt \$(P12)

Cypher: des3
Encrypted File: \$(P12)
Passphrase: \$(ENC_PWD)
Decrypted File Path: _build.p12

Advanced

Control Options

Enabled:
Continue on error:
Always run:

- Finally, update the Xcode Build step to reference the decrypted files.

- **Override Using:** File Contents
- **P12 Certificate File:** `_build.p12`
- **P12 Password:** `$(P12_PWD)`
- **Provisioning Profile File:** `_build.mobileprovision`
- **Remove Profile After Build:** Checked if you want the provisioning profile to be removed from the system after the build. Only check this if you will have one agent that only runs this build, as it is installed in a global location and could be accessed by other builds.

You are now all set! Any build agent that is running will now be able to securely build your app without any certificate management on the build machine itself. Repeat the process of adding different certificates and provisioning profiles to your source repository to enable separate dev, beta (ad hoc), and distribution builds.

Using preinstalled signing identities and mobile provisioning profiles in your Xcode build

You can also use a signing identity and provisioning profile you have installed on your Mac.

While you can just allow the build to attempt to automatch or use identities and profiles in your Xcode project, you can follow these steps to override the project and get specific about the exact identity you want to use.

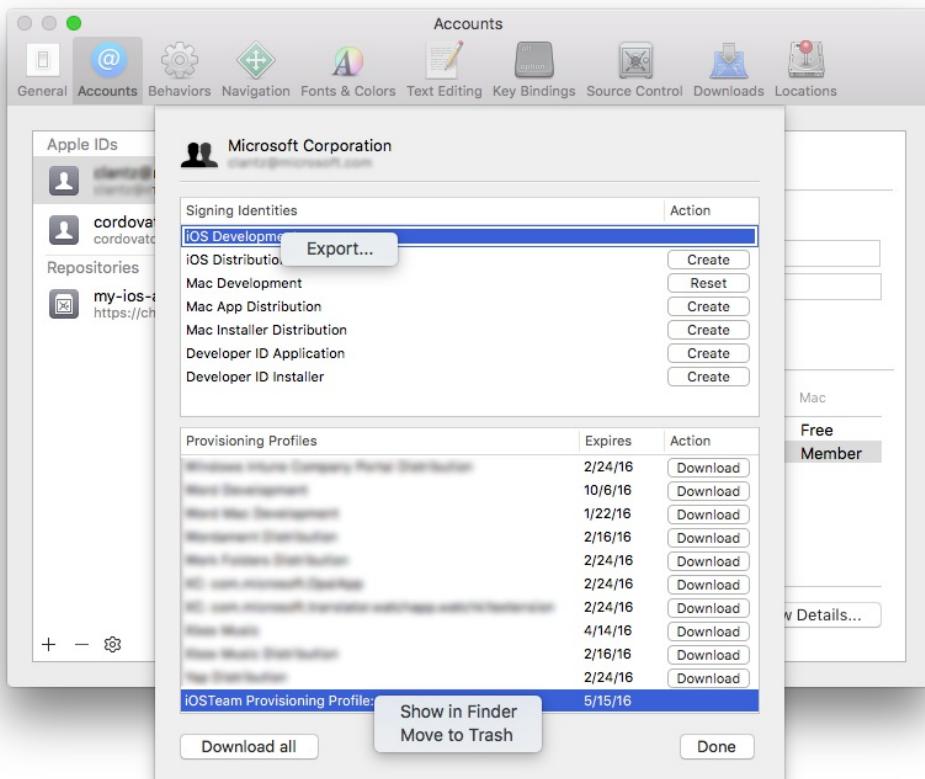
Follow these steps:

1. Determine the full name of the identity and the UUID of the provisioning profile you wish to use.
 - a. Find the full name of your signing identity by opening the Terminal app and typing:

```
security find-identity -v -p codesigning
```

You'll see a list of signing identities in the form "iPhone Developer/Distribution: Developer Name (ID)". If your identity is invalid, you will see something like (CSSMERR_TP_CERT_REVOKED) after the identity.

- b. Take note of the identity you want to use including the ID as this string is truly unique.
- c. Next, find the UUID for the provisioning profile you want to use.
 - a. Open Xcode and go to **Xcode > Preferences > Accounts** and select your Apple Developer account.
 - b. Select **View Details** and right-click the provisioning profile you want and select "Show in Finder".



- d. The name of the file that is highlighted is the UUID of your provisioning profile.
2. Now update your Xcode or Xamarin.iOS build step with references to these two identifiers. Enter the following values under the **Signing & Provisioning** category for the Xcode or Xamarin.iOS build task:
- **Override Using:** Identifiers
 - **Signing Identity:** Full signing identity you found using the security command above
 - **Provisioning Profile UUID:** UUID of the provisioning profile from the filename above
3. Next, if you are running the build agent as a launch agent, you will need to set up the build to unlock the default keychain.
- a. Go to the **Variables** tab and enter the following:
 - **KEYCHAIN_PWD:** Password to your default keychain. This is normally the password for the user that is starting the agent. *Be sure to click the "lock" icon.*
 - b. Update the Xcode step to unlock the keychain entering the following values under the **Signing & Provisioning** category for the Xcode or Xamarin.iOS build task:
 - **Unlock Default Keychain:** Checked
 - **Default Keychain Password:** `$(KEYCHAIN_PWD)`

Building and signing Android and Xamarin.Android projects

Managing Android signing is relatively simple. The [Android documentation](#) on the topic largely covers what you need to know to generate and use a keystore file containing your signing certificate.

However, here are a few extra tips that can help you get your app up and running quickly while still keeping your signing certificate secure.

Follow these steps:

1. **Windows only:** You may need to install openssl.exe. If you have the [Git command line tools](#) installed, openssl may already be in your path (Ex: C:\Program Files (x86)\Git\bin). If not, install the command-line

tools or download a binary distribution of OpenSSL for Windows from [one of the community mirrors](#) and add it to your path. You'll also need to do this for any Windows machines running the build agent.

2. Now, encrypt the keystore for your app.

- Open a terminal or command prompt and go to where your keystore is located.
- Type in the terminal:

```
openssl des3 -in release.keystore -out release.keystore.enc
`` substituting des3 for any encryption cypher you'd like to use and the appropriate input
and output file names.
```

- Enter a passphrase to encrypt the certificate when prompted and note this as you will use it later.

3. Next, add the encrypted keystore to source control.

4. Finally we'll update your build pipeline to decrypt and use the keystore.

- Open your Android or Xamarin.iOS build pipeline and go to the **Variables** tab. Enter these values:

- KEYSTORE**: Path to your encrypted keystore file in source control.
- KEYSTORE_PWD**: Password to the unencrypted keystore file. *Be sure to click the "lock" icon.* This will secure your password and obscure it in all logs.
- KEY**: The key alias for the signing certificate you generated.
- KEY_PWD**: The password for the key associated with the specified alias. *Again, be sure to select the "lock" icon.*
- ENC_PWD**: The passphrase you used to encrypt the keystore file. *Be sure to select the "lock" icon.*

List of predefined variables		
Name	Value	Allow at Queue Time
system.collectionId	85f58b25-5e4b-404f-82ce-93fc14939153	<input type="checkbox"/>
system.teamProject	tfs-vnext-test	<input type="checkbox"/>
system.definitionId	42	<input type="checkbox"/>
system.debug	true	<input type="checkbox"/> <input checked="" type="checkbox"/>
Configuration	Release	<input type="checkbox"/> <input checked="" type="checkbox"/>
KEYSTORE	release.keystore.enc	<input type="checkbox"/>
KEYSTORE_PWD	*****	<input type="checkbox"/> <input checked="" type="checkbox"/>
KEY	tfsvnext-test	<input type="checkbox"/> <input checked="" type="checkbox"/>
KEY_PWD	*****	<input type="checkbox"/> <input checked="" type="checkbox"/>
ENC_PWD	*****	<input type="checkbox"/> <input checked="" type="checkbox"/>

- Under the **Build** tab in your build pipeline, add a **Decrypt File (OpenSSL)** step and move this to the top of your build pipeline.

5. Now, enter the proper information using the variables you created above to decrypt the file to a static filename.

- Cypher**: The cypher you specified while encrypting. (Ex: des3)
- Encrypted File**: `$(KEYSTORE)`
- Passphrase**: `$(ENC_PWD)`
- Decrypted File Path**: `_build.keystore`

The screenshot shows a build pipeline step titled "Decrypt \$(KEYSTORE)". The configuration includes:

- Cypher: des3
- Encrypted File: \$(KEYSTORE)
- Passphrase: \$(ENC_PWD)
- Decrypted File Path: _build.keystore

Below the main configuration, there is an "Advanced" section.

6. Update the build step to use the values you entered.

- a. If you are using the **Gradle, Ant, or Maven** build tasks, add these values under **Options**:

The screenshot shows a build pipeline step titled "gradlew build". The configuration includes:

- Gradle Wrapper: gradlew
- Options: -Pkey.store=_build.keystore -Pkey.store.password=\$(KEYSTORE_PWD) -Pkey.alias=\$(KEY) -Pkey.alias.password=\$(KEY_PWD)
- Tasks: build

Below the main configuration, there is a "JUnit Test Results" section with options for publishing test results to VSO/TFS.

- b. If you are using the **Android Signing** task, add these values under **Jarsign Options**:

- Keystore File: _build.keystore
- Keystore Password: \$(KEYSTORE_PWD)
- Key Alias: \$(KEY)
- KEY Password: \$(KEY_PWD)

You are now all set! Any build agent will now be able to securely build your app without any certificate management on the build machine itself.

FAQ

Do I need an agent?

You need at least one [agent](#) to run your build or release.

I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

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](#).

My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

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](#).

Release pipelines

5/13/2021 • 11 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

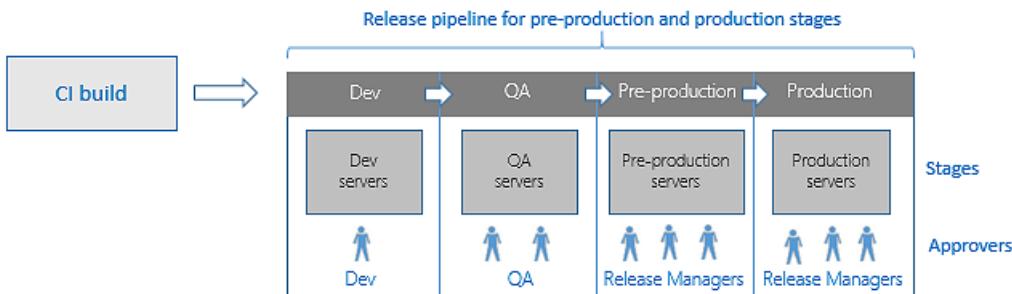
NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

NOTE

This article covers classic release pipelines. If you want to use YAML to author CI/CD pipelines, then see [Create your first pipeline](#).

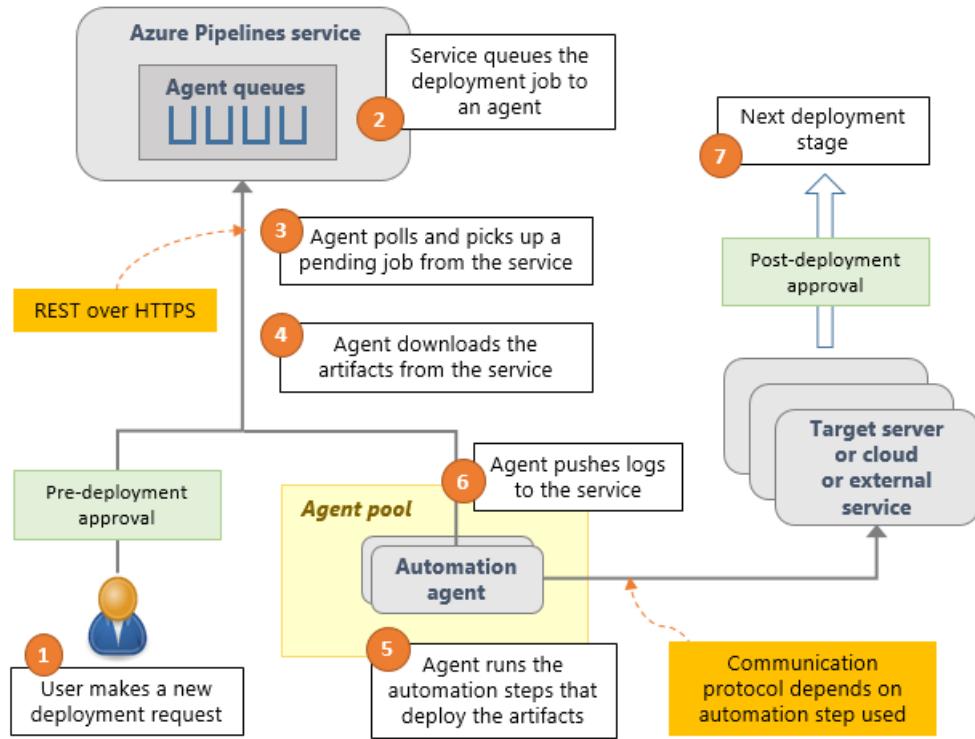
Release pipelines in Azure Pipelines help your team **continuously deliver** software to your customers at a faster pace and with lower risk. You can **fully automate** the testing and delivery of your software in multiple stages all the way to production. Or, set up semi-automated processes with **approvals** and **on-demand deployments**.



See [Releases in Azure Pipelines](#) to understand releases and deployments and watch the following video to see release pipelines in action.

How do release pipelines work?

Release pipelines store the data for your pipelines, stages, tasks, releases, and deployments in Azure Pipelines.



Azure Pipelines runs the following steps as part of every deployment:

1. **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's required, it sends out email notifications to the appropriate approvers.
2. **Queue deployment job:** Azure Pipelines schedules the deployment job on an available [automation agent](#). An agent is a piece of software that can run tasks in the deployment.
3. **Agent selection:** An automation agent picks up the job. The agents for release pipelines are exactly the same as the agents that run your builds in Azure Pipelines. A release pipeline can contain settings to select an appropriate agent at runtime.
4. **Download artifacts:** The agent downloads all the artifacts specified in that release, provided you haven't 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.
7. **Post-deployment approval:** When deployment to a stage is complete, Azure Pipelines checks if there's 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.

Release pipelines and build pipelines have separate UIs. The main differences in the pipelines are the support in release pipelines for different types of triggers, and the support for approvals and gates.

How do I use a release pipeline?

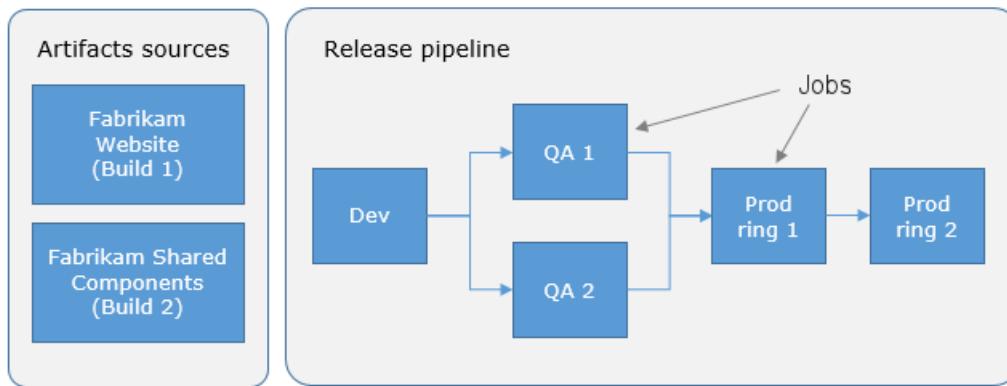
You start using Azure Pipelines releases by authoring a release pipeline for your application. To author a release pipeline, you must specify the [artifacts](#) that make up the application and the [release pipeline](#).

An [artifact](#) is a deployable component of your application. It's typically produced through a Continuous

Integration or a build pipeline. Azure Pipelines releases can deploy artifacts produced by a [wide range of artifact sources](#), such as Azure Pipelines build, Jenkins, or Team City.

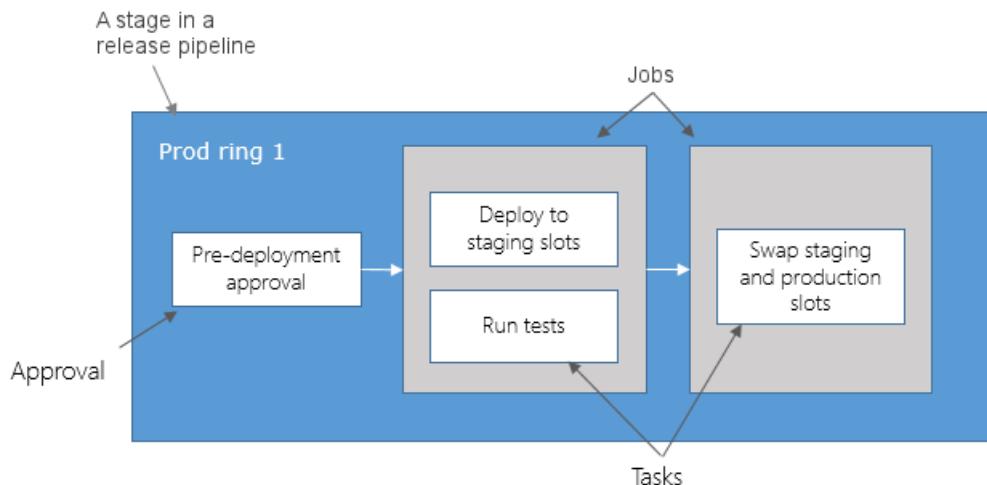
Define the **release pipeline** using [stages](#) and restrict deployments into or out of a stage using [approvals](#). Define the automation in each stage using [jobs](#) and [tasks](#). Use [variables](#) to generalize your automation and [triggers](#) to control when the deployments should be kicked off automatically.

See the following example of a release pipeline that can be modeled through a release pipeline:



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 Production ring 1 and then to Production ring 2. Each production ring represents multiple instances of the same website deployed at various locations around the world.

See the following example of how deployment automation can be modeled within a stage:



In this example, a [job](#) is used to deploy the app to websites across the world 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.

NOTE

TFS 2015: Jobs and fork/join deployments are not available in TFS 2015.

Next:

Check out the following articles to learn how to:

- [Create your first pipeline](#).

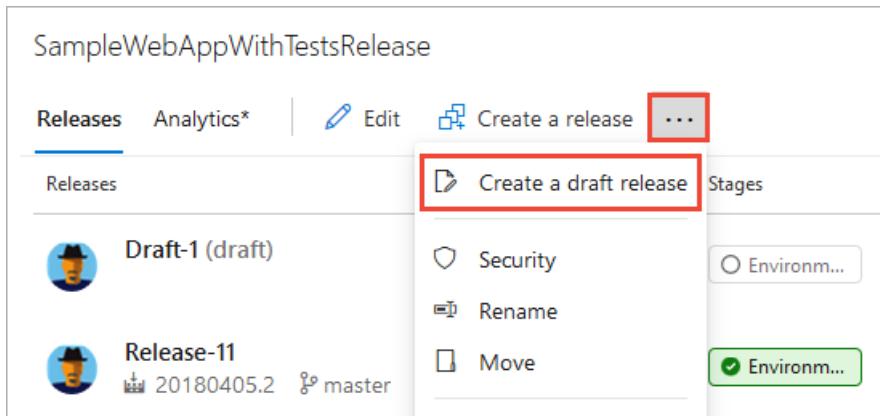
- Set up a multi-stage managed release pipeline.
- Manage deployments by using approvals and gates.

What is a draft release?

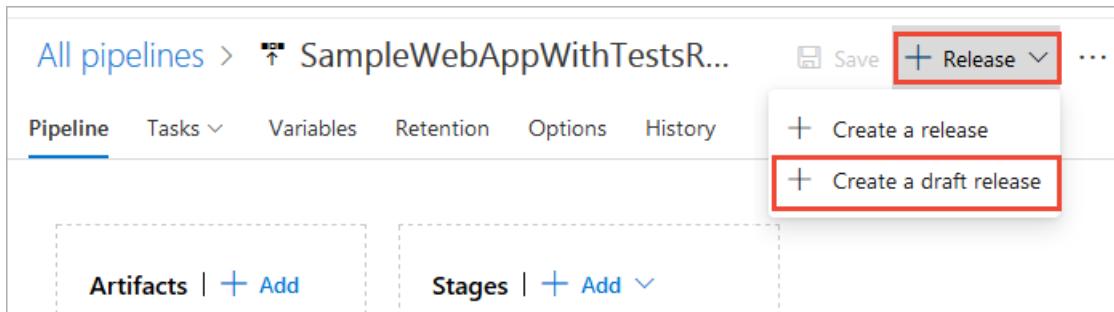
Draft releases are deprecated in Azure Pipelines because you can change variables while you're [creating the release](#).

Creating a draft release allows you to edit some settings for the release and tasks, depending on your role permissions before you start the deployment. The changes apply only to that release, and don't affect the settings of the original pipeline.

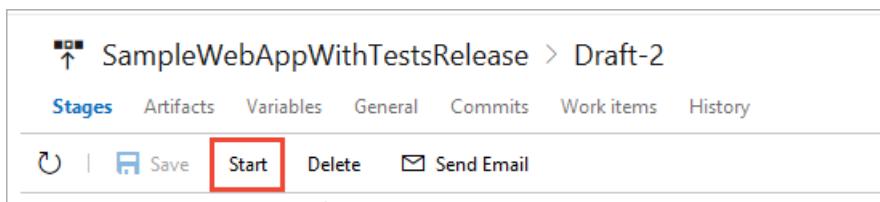
Create a draft release using the "..." ellipses link in the list of releases:



... or the **Release** drop-down in the pipeline definition page:



After you finish editing the draft release, choose **Start** from the draft release toolbar.



How do I specify variables I want to edit when a release is created?

In the **Variables** tab of a release pipeline, when you add new variables, set **Settable at release time** for the variables that you want to edit when a release gets created and queued.

All definitions > Anjani-RD

Pipeline Tasks Variables Retention Options History

Process variables

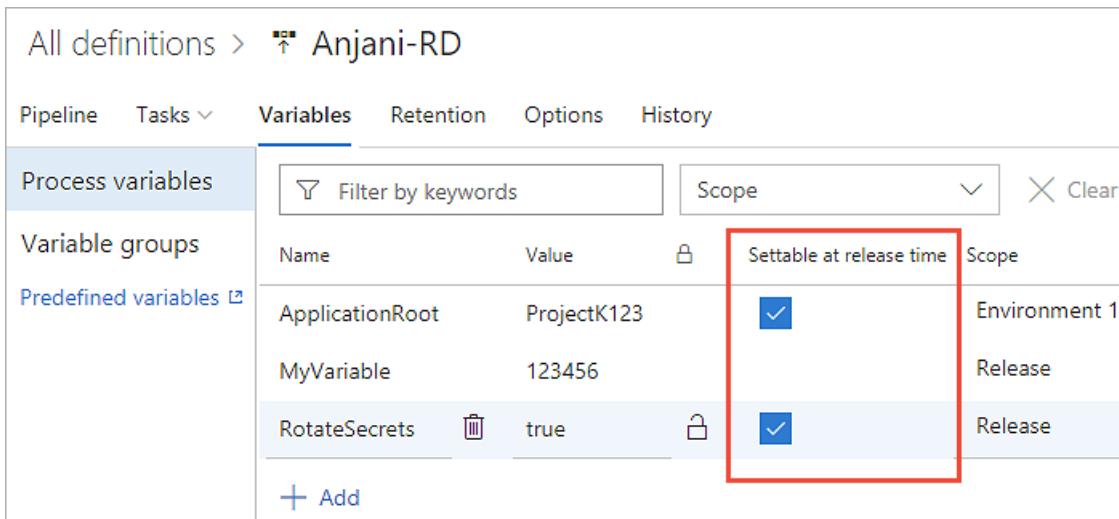
Variable groups

Predefined variables

Filter by keywords Scope Clear

Name	Value	Scope
ApplicationRoot	ProjectK123	Settable at release time Environment 1
MyVariable	123456	Release
RotateSecrets	true	Release

+ Add



Then, when you create a new release, you can edit the values for these variables.

Create new release

Anjani-RD

Pipeline ▲

Click on an environment to change its trigger from automated to manual.

Environment 1

Environments for trigger change from automated to manual. ⓘ

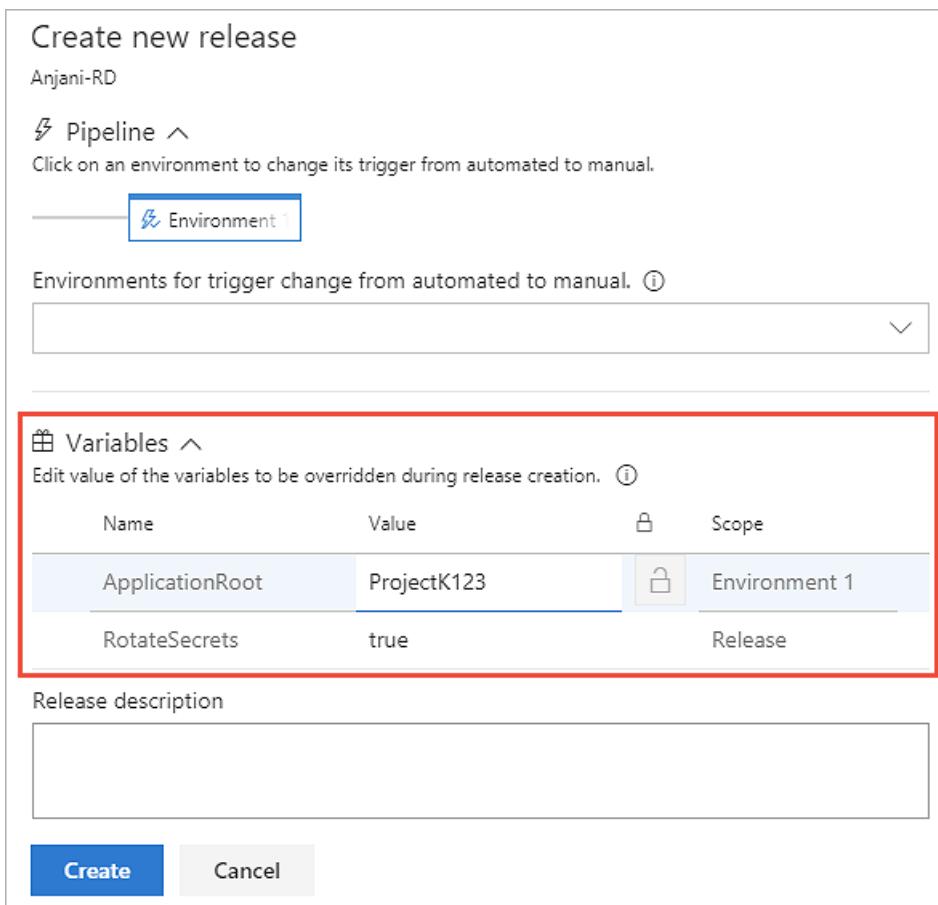
Variables ▲

Edit value of the variables to be overridden during release creation. ⓘ

Name	Value	Scope
ApplicationRoot	ProjectK123	Environment 1
RotateSecrets	true	Release

Release description

Create Cancel



How do I integrate and report release status?

The current status for a release can be reported back in the source repository. In the **Options** tab of a release pipeline, open the **Integrations** page.

All pipelines > SampleWebAppWithTestsRelease

Pipeline	Tasks ▾	Variables	Retention	Options	History						
<table border="1"><tr><td>General</td><td><input type="checkbox"/> Report deployment status to the repository host <small>(i)</small></td></tr><tr><td>Integrations</td><td><input type="checkbox"/> Report deployment status to Work <small>(i)</small></td></tr><tr><td></td><td><input type="checkbox"/> Enable the deployment status badge <small>(i)</small></td></tr></table>						General	<input type="checkbox"/> Report deployment status to the repository host <small>(i)</small>	Integrations	<input type="checkbox"/> Report deployment status to Work <small>(i)</small>		<input type="checkbox"/> Enable the deployment status badge <small>(i)</small>
General	<input type="checkbox"/> Report deployment status to the repository host <small>(i)</small>										
Integrations	<input type="checkbox"/> Report deployment status to Work <small>(i)</small>										
	<input type="checkbox"/> Enable the deployment status badge <small>(i)</small>										

Report deployment status to the repository host

If your sources are in an Azure Repos Git repository in your project, this option displays a badge on the Azure Repos pages. The badge indicates where the specific commit got deployed and whether the deployment is passing or failing. This option improves the traceability from code commit to deployment.

The deployment status is displayed in the following sections of Azure Repos:

- **Files**: Indicates the status of the latest deployment for the selected branch.
- **Commits**: Indicates the deployment status for each commit (requires the continuous integration (CD) trigger to be enabled for your release).
- **Branches**: Indicates the status of the latest deployment for each branch.

If a commit gets deployed to multiple release pipelines, with multiple stages, each has an entry in the badge with status that's shown for each stage. By default, when you create a release pipeline, deployment status is posted for all stages. However, you can selectively choose the stages for which deployment status should be displayed in the status badge (for example, show only the production stage). Your team members can select the status badge to view the latest deployment status for each of the selected stages of the release pipelines.

NOTE

If your source is not an Azure Repos Git repository, you cannot use Azure Pipelines to automatically publish the deployment status to your repository. However, you can still use the *Enable the Deployment status badge* option described as follows, to show deployment status within your version control system.

Report deployment status to Work

Select this option if you want to create links to all work items that represent associated changes to the source, when a release is complete.

Enable the deployment status badge

Select this option if you want to display the latest outcome of a stage deployment on an external website.

1. Select "Enable the deployment status badge".
2. Select the stages for which you want to display the outcome. By default, all the stages are selected.
3. Save your pipeline.
4. Copy the badge URL for the required stage to the clipboard.
5. Use this badge URL as a source of an image in an external website.

For example: ``

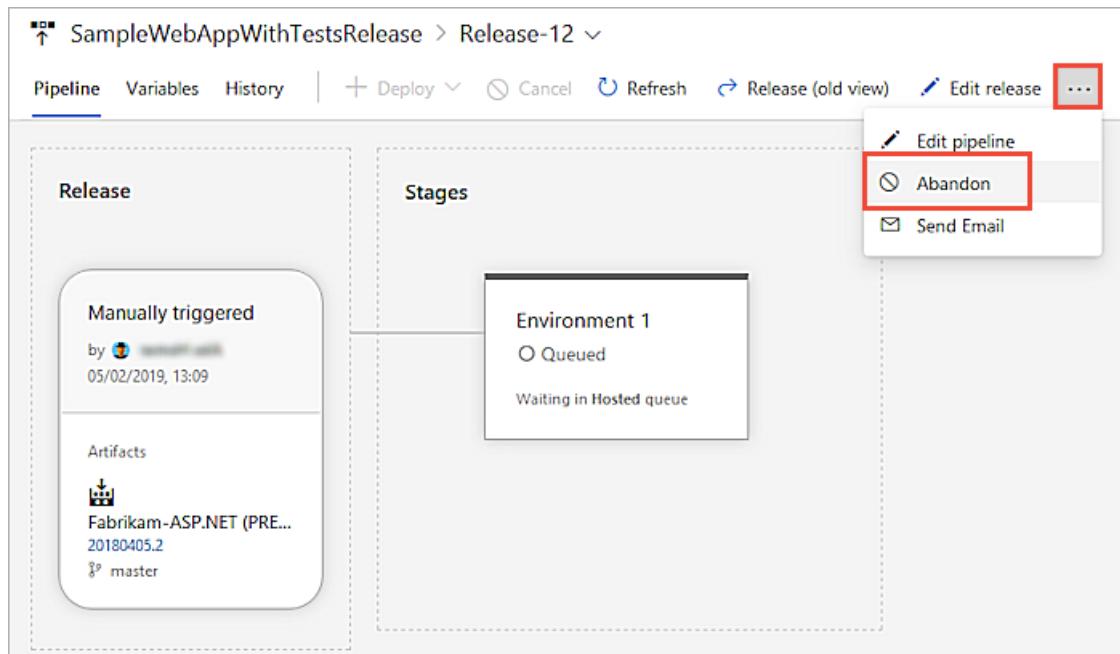
When should I edit a release instead of the pipeline that defines it?

You can edit the approvals, tasks, and variables of a previously deployed release. Do so instead of editing these values in the pipeline from which the release was created. However, these edits apply to only the release generated when you redeploy the artifacts. If you want your edits apply to all future releases and deployments, choose the option to edit the release pipeline instead.

When and why would I abandon a release?

After you create a [release](#), you can redeploy the artifacts to any stages that are defined in that release. This is useful if you want to do regular manual releases, or set up a continuous integration [stage trigger](#) that redeploys the artifacts using this release.

If you don't plan to reuse the release, or want to prevent it being used to redeploy the artifacts, you can abandon the release using the shortcut menu that opens from the ellipses (...) icon in the **Pipeline** view of the pipeline.



You can't abandon a release when a deployment is in progress, you must cancel the deployment first.

How do I send release summaries by email?

After a release is triggered and completed, you may want to email the summary to stakeholders. Use the [Send Email](#) option on the menu that opens from the ellipses (...) icon in the **Pipeline** view of the pipeline.

The screenshot shows a completed release named "Manually triggered" by a user on 2/13/2019 at 4:42 AM. The release consists of three stages: "Ring 0 - tfs-ppe" (Succeeded), "Rotate PPE Real" (Succeeded), and "Ring 0 - tfs-eus" (Succeeded). The "Artifacts" section shows a VSO.Release artifact from AzureDevOps_M147 2.18. A context menu is open in the top right corner, with the "Send Email" option highlighted in red.

In the **Send release summary mail** window, you can further customize the information sent in the email by selecting only certain sections of the release summary.

How do I manage the names for new releases?

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 in the **General** page.

When specifying the format mask, you can use the following pre-defined variables.

VARIABLE	DESCRIPTION
Rev: rr	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/dddd, y/yy/yyyy/yyyyy, 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, it's the number of the primary build .
Build.DefinitionName	The pipeline name of the build contained in the release. If a release has multiple builds, it's 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 main if the branch is <code>refs/heads/main</code> . For Team Foundation Version Control, this is of the form branch if the root server path for the workspace is <code>\$/teamproject/branch</code> . This variable is not set for Jenkins or other artifact sources.
Custom variable	The value of a global configuration property defined in the release pipeline. You can update the release name with custom variables using the Release logging commands

For example, the release name format

```
Release $(Rev:rrr) for build $(Build.BuildNumber) $(Build.DefinitionName)
```

creates releases with names such as **Release 002 for build 20170213.2 MySampleAppBuild**.

How do I specify the retention period for releases?

You can customize how long releases of this pipeline must be retained. For more information, see [release retention](#).

How do I use and manage release history?

Every time you save a release pipeline, Azure Pipelines keeps a copy of the changes. This copy allows you to compare the changes at a later point, especially when you're debugging a deployment failure.

How do I set it up to automatically link new work in a build?

When we establish traceability between work items and builds/releases, there are the following two aspects:

- List the work items that were newly built as part of a build. You can find this when you're looking at a build instance.
- List the builds that this work item was built in. You can find the list in the "Development" section in a work item form. The setting to "Automatically link new work in this build" has nothing to do with how we compute the first bulleted item. It only influences how we compute the second bulleted item.

The computation for the first bullet is as follows for a build: Let's say, for example, that you started a new build. Whatever the setting, we compute a list of new commits for the build. We do the following tasks:

- We find the commit c2 that is being built now.
- We find the commit c1 that was built in the last successful build of the same branch (`Build.SourceBranch`).
- We find all the commits between c1 and c2 (in the commit tree).

It could happen that there's no last known successful build on the same branch. For example, when you run a build for the first time on a branch, or when all the previous builds on a branch have been deleted (possibly through retention policies). The list could be long in these cases.

Once we have the list of commits, we enumerate all the work items associated with each of those commits. This is the list that you see in a build.

Get started now!

Complete the following steps:

1. [Set up a multi-stage managed release pipeline](#)
2. [Manage deployments by using approvals and gates](#)

Related articles

- [Deploy pull request builds using Azure Pipelines](#)
- [Stage templates in Azure Pipelines](#)

Deploy from multiple branches using Azure Pipelines

3/26/2021 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2019

Artifact filters can be used with release triggers to deploy from multiple branches. Applying the artifact filter to a specific branch enables deployment to a specific stage when all the conditions are met.

Prerequisites

- A Git repository to create the pipeline. If you don't have one, use the [pipelines-dotnet-core](#) sample app.
- A working build for your repository.

Set up a release pipeline

1. Select **Releases** under **Pipelines**. If this is your first release pipeline, select **New Pipeline**, otherwise select **Create a release**.
2. Select **Start with an empty job** when prompted to select a template.
3. Select **Add an artifact** and specify the project, the build pipeline, and the default version. Select **Add** when you are done.

Add an artifact

Source type

Build Azure Repos ... GitHub TFVC

5 more artifact types ▾

Project * ⓘ

FabrikamFiber

Source (build pipeline) * ⓘ

TestPipelines\FabrikamFiber-Cl

Default version * ⓘ

Latest

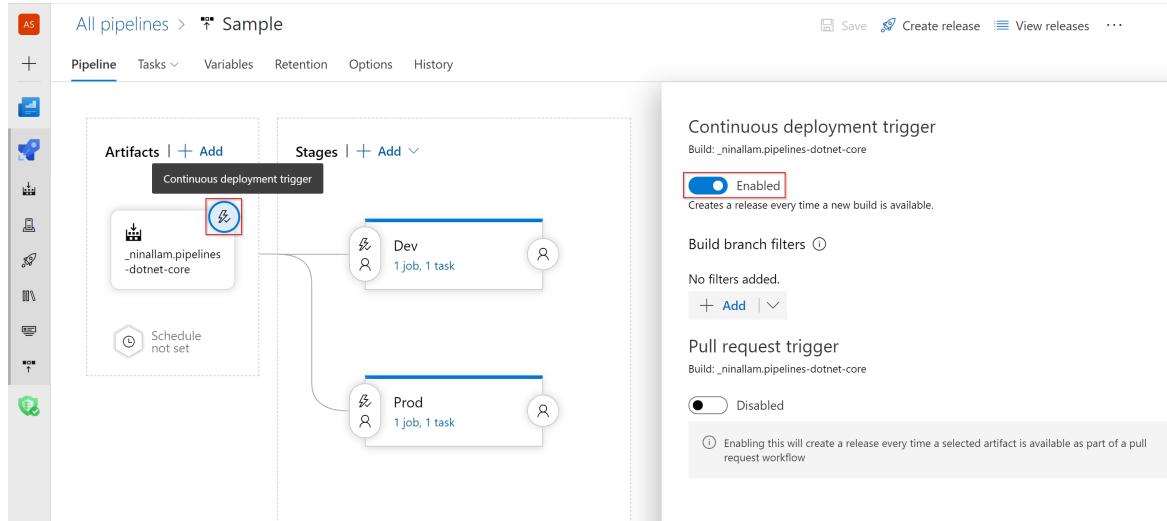
Source alias * ⓘ

_FabrikamFiber-Cl

ⓘ No version is available for TestPipelines\FabrikamFiber-Cl or the latest version has no artifacts to publish. Please check the source pipeline.

Add

4. Select the **Continuous deployment trigger** icon and enable the **Continuous deployment trigger** to create a release every time a new build is available.



5. Under **Stages**, select the stage and rename it to **Dev**. This stage will be triggered when a build artifact is published from the dev branch.

Stages | + Add ▾

Dev
1 job, 0 task

Properties

Stage name: Dev

Stage owner: Rami Bououni

6. Select the **Pre-deployment conditions** icon in the *Dev* stage and set the deployment trigger to **After release** to trigger a deployment to this stage every time a new release is created.

Stages | + Add ▾

Dev
1 job, 0 task

Pre-deployment conditions

Triggers

Select trigger: After release

Artifact filters: Disabled

Schedule: Disabled

Pull request deployment: Disabled

7. Enable the **Artifact filters**. Select **Add** and specify your artifact and build branch.

All pipelines > Sample

Pipeline Tasks Variables Retention Options History

Artifacts | + Add

_ninallam.pipelines-dotnet-core
Schedule not set

Stages | + Add ▾

Dev
1 job, 0 task

Prod
1 job, 1 task

Pre-deployment conditions

Triggers

Select trigger: After release

Artifact filters

+ Add

_ninallam.pipelines-dotnet-core

Type	Build branch	Build tags
Include	dev	

8. Under Stage, select Add then New stage to add a new stage. Select Start with an empty job when prompted to select a template, and rename the stage to Prod. This stage will be triggered when a build artifact is published from the main branch. Repeat the steps 6-8 and replace the **Build branch** for this stage to main.

The screenshot shows the Azure Pipelines interface for editing a pipeline named 'Sample'. On the left, there's a sidebar with various icons. The main area is divided into two sections: 'Artifacts' and 'Stages'. The 'Stages' section contains two stages: 'Dev' and 'Prod'. The 'Prod' stage is currently selected. To the right of the stages, a detailed configuration pane is open for the 'Prod' stage. It shows 'Pre-deployment conditions' set to 'Prod', 'Triggers' set to 'After release', and 'Artifact filters' set to 'Build branch: master'. The 'master' filter is highlighted with a red box.

9. Add to each stage all the relevant deployment tasks to your environment.

Now the next time you have a successful build, the pipeline will detect which branch triggered that build and trigger deployment to the appropriate stage only.

The screenshot shows the Azure Pipelines interface for viewing a release history. The release is named 'Release-5'. The 'Release' section shows a 'Continuous deployment' step with an artifact named '_ninallam.pipeline... 20200302.7'. The 'Stages' section shows two stages: 'Dev' and 'Prod'. The 'Dev' stage is listed as 'Succeeded' with a green checkmark and a timestamp 'on 3/2/2020, 3:38 PM'. The 'Prod' stage is listed as 'Not deployed' with a red warning icon and the message 'Artifact conditions not met'.

Related articles

- [Release triggers](#)
- [Build Artifacts](#)
- [Release artifacts and artifact sources](#)
- [Publish and download artifacts](#)

Deploy pull request Artifacts with classic release pipelines

6/4/2021 • 3 minutes to read • [Edit Online](#)

Azure DevOps Services | Azure DevOps Server 2020 | Azure DevOps Server 2019

Pull requests provide an effective way to review code changes before it is merged into the codebase. However, these changes can introduce issues that can be tricky to find without building and deploying your application to a specific environment. Pull request triggers enable you to set up a set of criteria that must be met before deploying your code. You can use pull request triggers with code hosted on Azure Repos or GitHub.

Setting up pull request deployments is a two step process:

1. Set up a pull request trigger.
2. Set up branch policies (in Azure Repos) or status checks (in GitHub) for your release pipeline.

With pull request triggers, anytime you raise a new pull request for the designated branch, a release is triggered automatically to start the deployment to the designated environments. The deployment status will then be displayed on the pull request page. Pull request triggers can help you maintain better code quality, release with higher confidence, and discover any issues early on in the development cycle.

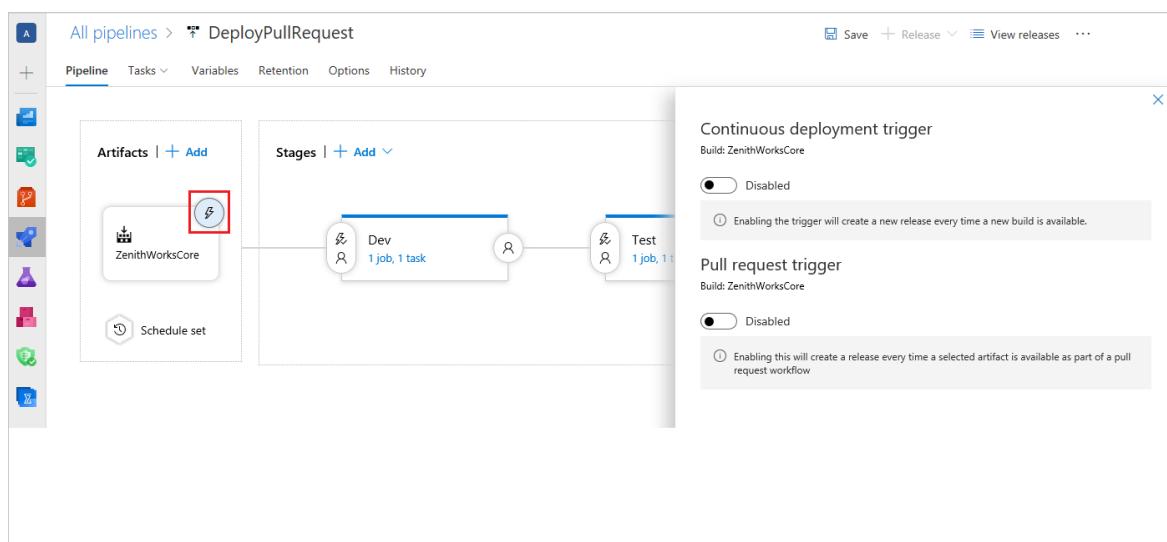
Prerequisites

- A GitHub/Azure repos repository to create the pipeline. If you don't have one, use the [pipelines-dotnet-core](#) sample app.
- A working build for your repository.
- A classic release pipeline. [Set up a release pipeline](#) for your project if you don't already have one.

Create a pull request trigger

A pull request trigger creates a release every time a new build Artifact is available. You can set up pull request triggers for both Azure Repos or GitHub repositories.

1. From within your project, Select **Pipelines > Releases**, and then select your release pipeline.
2. Select the **Continuous deployment trigger** icon in the **Artifacts** section



3. Under the **Pull request trigger** section, select the toggle button to enable it.

The screenshot shows the 'Pull request trigger' configuration page for a build named 'ZenithWorksCore'. At the top, there is a toggle switch labeled 'Enabled' which is turned on. Below the toggle, a description states: 'Creates a release every time a new version of the selected artifact is available as part of a pull request workflow'. Under the heading 'Target Branch Filters' (with an information icon), there are two tabs: 'Target Branch' and 'Build tags'. The 'Target Branch' tab is selected, showing a dropdown menu with 'main' selected. To the right of the dropdown is a delete icon. Below the dropdown is a blue '+ Add' button. A red box highlights the 'Enabled' toggle switch.

4. From the **Target Branch Filters**, select your **Target Branch** from the dropdown menu.

This screenshot is identical to the one above, showing the 'Pull request trigger' configuration page for 'ZenithWorksCore'. The 'Enabled' toggle is on, and the 'Target Branch' dropdown is set to 'main'. A red box highlights the 'main' option in the dropdown menu.

5. To deploy your application to a specific stage you need to explicitly opt-in that stage. The **Stages** section shows the stages that are enabled for pull request deployments.

The screenshot shows the 'Pull request trigger' configuration page for 'ZenithWorksCore'. It includes the 'Target Branch Filters' section with the 'main' branch selected. Below this is a 'Stages' section with an information icon. A yellow callout box contains the text: '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.' A red box highlights this message.

To opt-in a stage for pull request deployment, select the **Pre-deployment conditions** icon for your specific stage, and then select **Triggers > After release**. Finally select the **Pull request deployment** toggle button to enable it.

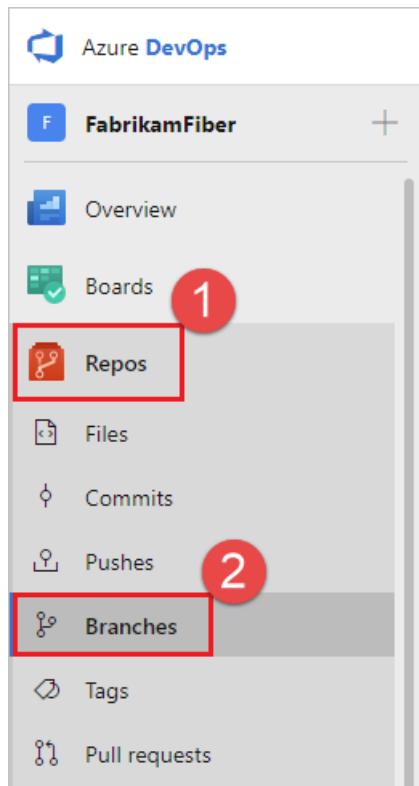
IMPORTANT

Enabling automatic pull request deployments for production stages is not recommended.

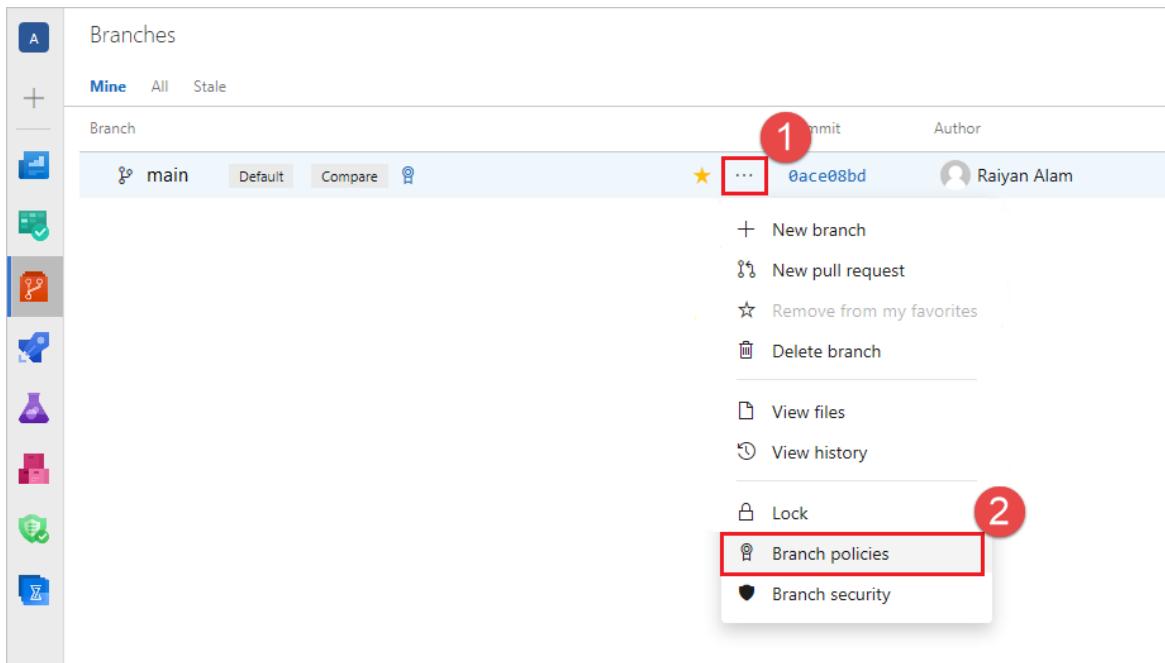
Set up branch policy for Azure Repos

You can use branch policies to implement a list of criteria that must be met for a pull request to be merged.

- From within your project, select **Repos > Branches** to access the list of branches for your repository.



- Select the context menu for your appropriate branch ..., then select **Branch policies**.



3. Select **Add status policy**, then select a **status to check** from the dropdown menu. The release definition should have run at least once with the pull request trigger switched on in order to get the status. Select the status corresponding to your release definition and then select **Save**.

Authorized organization	Requirement	Path filter	Reset conditions	Status name
Any organization	Required	No filter	Never	wip-checker
Any organization	Required	No filter	Never	VSTS-RM/DeployPullRequest
Any organization	Optional	No filter	Never	wip-checker-opt

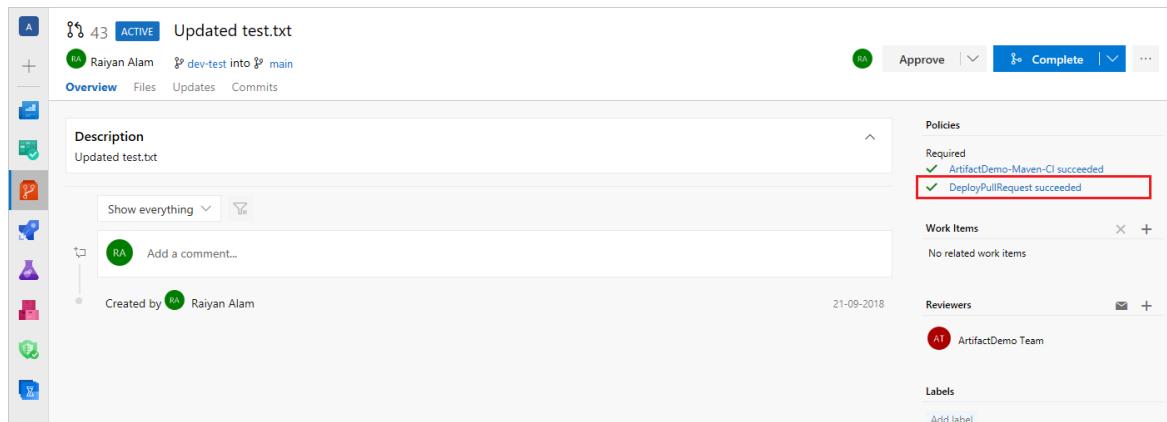
For more information on customizing your status policy, see [Configure a branch policy for an external service](#).

4. With the new status policy added, users won't be able to merge any changes to the target branch without a "succeeded" status is posted to the pull request.

Authorized organization	Requirement	Path filter	Reset conditions	Status name
Any organization	Required	No filter	Never	wip-checker
Any organization	Required	No filter	Never	VSTS-RM/DeployPullRequest
Any organization	Optional	No filter	Never	wip-checker-opt

5. You can view the status of your policies from the pull request Overview page. Depending on your policy settings, you can view the posted release status under the **Required**, **Optional**, or **Status** sections. The

release status gets updated every time the pipeline is triggered.



Set up status checks for GitHub repositories

Enabling status checks for a GitHub repository allow an administrator to choose which criteria must be met before a pull request is merged into the target branch. Check out the [Branch protection rule](#) GitHub article to learn how to enable status checks for your GitHub repository. The status checks will be posted on your pull request only after your release pipeline has run at least once with the pull request deployment condition Enabled.

A screenshot of the GitHub 'Branch protection rule' configuration page. On the left, there's a sidebar with options like 'Options', 'Collaborators', 'Branches' (which is selected and highlighted in orange), 'Webhooks', 'Integrations & services', and 'Deploy keys'. The main area is titled 'Branch protection rule' and contains sections for 'Apply rule to' (which applies to 1 branch), 'Rule settings', and 'Protect matching branches'. Under 'Rule settings', there are three checkboxes: 'Require pull request reviews before merging' (unchecked), 'Require status checks to pass before merging' (checked), and 'Require branches to be up to date before merging' (unchecked). The 'Require status checks to pass before merging' section includes a list of status checks found in the last week: 'Scan' (unchecked), 'GitHub-PR-Deploy' (unchecked and highlighted with a red border), and 'Github-CI-ArtifactDemo-Maven' (checked). A 'Required' button is located at the bottom right of this list.

You can view your status checks in your pull request under the **Conversation** tab.

Fixed typo in description #16

Edit

[Open](#) raiyanalam wants to merge 1 commit into main from raiyanalam-patch-9

Conversation 0

Commits 1

Checks 0

Files changed 1

+1 -1



raiyanalam commented 7 minutes ago

Owner

+ ...

No description provided.

Fixed typo in description

Verified ✓ e86293d

Add more commits by pushing to the raiyanalam-patch-9 branch on raiyanalam/java-repo.



All checks have passed

2 successful checks

Hide all checks

GitHub-PR-Deploy — GitHub-PR-Deploy succeeded

Details

Github-CI-ArtifactDemo-Maven — #20181003.3 succeeded

Required Details

This branch has no conflicts with the base branch

Merging can be performed automatically.

[Merge pull request](#)

You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

Reviewers



No reviews

Assignees



No one—assign yourself

Labels



None yet

Projects



None yet

Milestone



No milestone

Notifications

Unsubscribe

You're receiving notifications because you authored the thread.

Related articles

- [Release triggers](#)
- [Supported build source repositories](#)
- [Git branch policies](#)
- [Configure a branch policy for an external service](#)

Define your multi-stage continuous deployment (CD) pipeline

3/6/2021 • 6 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Azure Pipelines provide a highly configurable and manageable pipeline for releases to multiple stages such as development, staging, QA, and production. It also offers the opportunity to implement gates and approvals at each specific stage.

In this tutorial, you will learn about:

- Continuous deployment triggers
- Adding stages
- Adding pre-deployment approvals
- Creating releases and monitoring deployments

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.

Continuous deployment (CD) triggers

Enabling continuous deployment trigger will instruct the pipeline to automatically create a new release every time a new build is available.

1. In **Azure Pipelines**, open the **Releases** tab. Select your release pipeline select **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. Below it, there are tabs for 'Overview', 'Releases', and 'Deleted'. A navigation bar at the top includes icons for 'Release', 'Title', and 'Environments'.

2. Select 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 selected in the 'SampleApp - 1' pipeline editor. In the 'Artifacts' section, a 'DotNetSample-ASP.NET Core-Cl' artifact is listed with its 'Continuous deployment trigger' icon highlighted with a red box. In the 'Stages' section, a stage named 'Stage 1' is shown with its 'Pre-deployment conditions' icon highlighted with a red box. The right panel displays the 'Continuous deployment trigger' configuration, which is currently enabled and set to create a release every time a new build is available. It also shows build branch filters and type settings.

3. Select 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 in the 'Pipeline' tab. Stage 1 is selected and its 'Pre-deployment conditions' icon is highlighted with a red box. The right panel shows the 'Pre-deployment conditions' configuration for Stage 1. Under 'Triggers', the 'After release' option is selected. Other options like 'Manual only' and schedule settings are also visible.

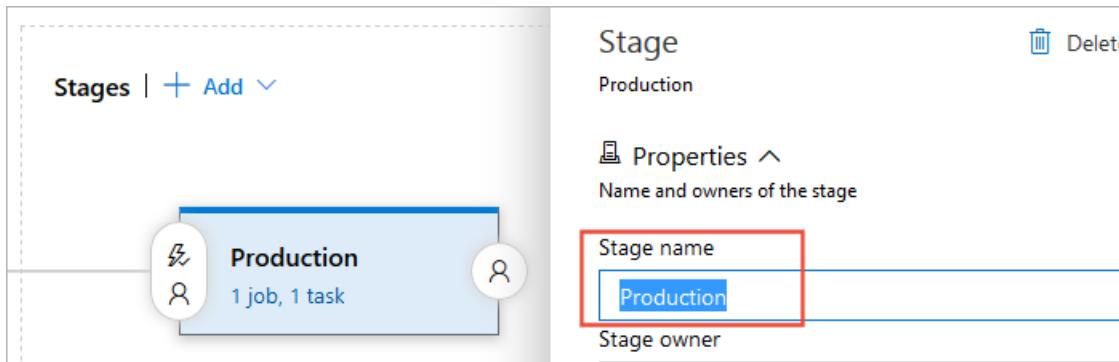
You can also set up [Release triggers](#), [Stage triggers](#) or [schedule deployments](#).

Add stages

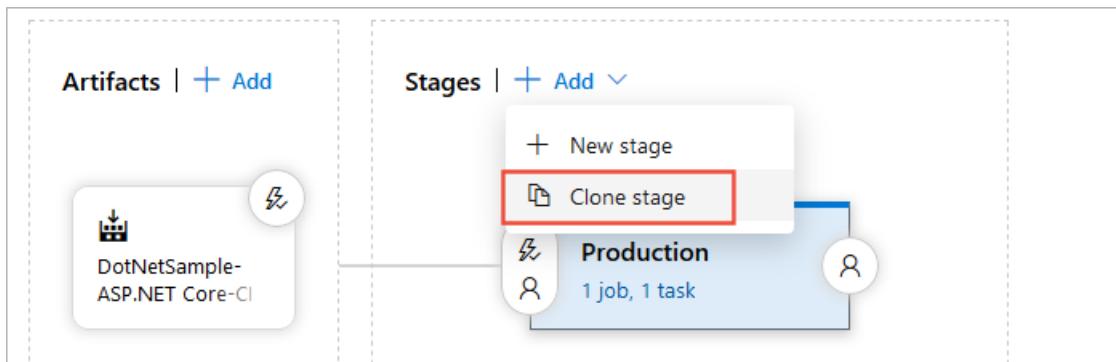
In this section, we will add two new stages to our release pipeline: QA and production (Two Azure App Services

websites in this example). This is a typical scenario where you would deploy initially to a test or staging server, and then to a live or production server. Each **stage** represents one deployment target.

1. Select the **Pipeline** tab in your release pipeline and select the existing stage. Change the name of your stage to **Production**.

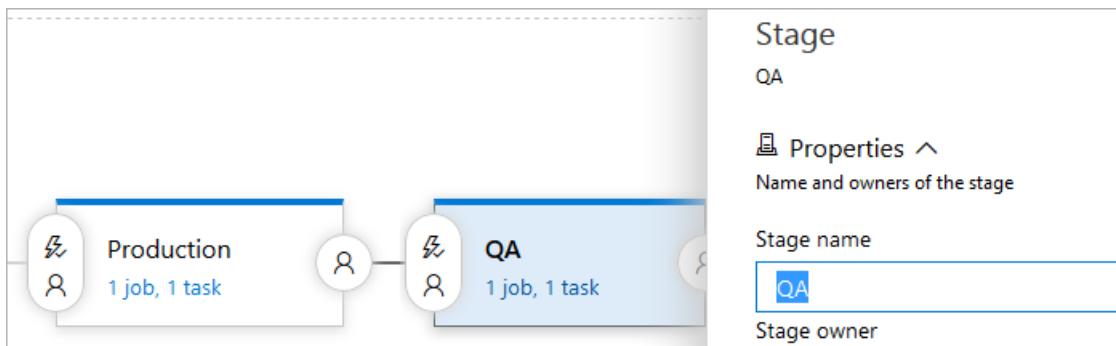


2. Select the **+ Add** drop-down list and choose **Clone stage** (the clone option is available only when an existing stage is selected).

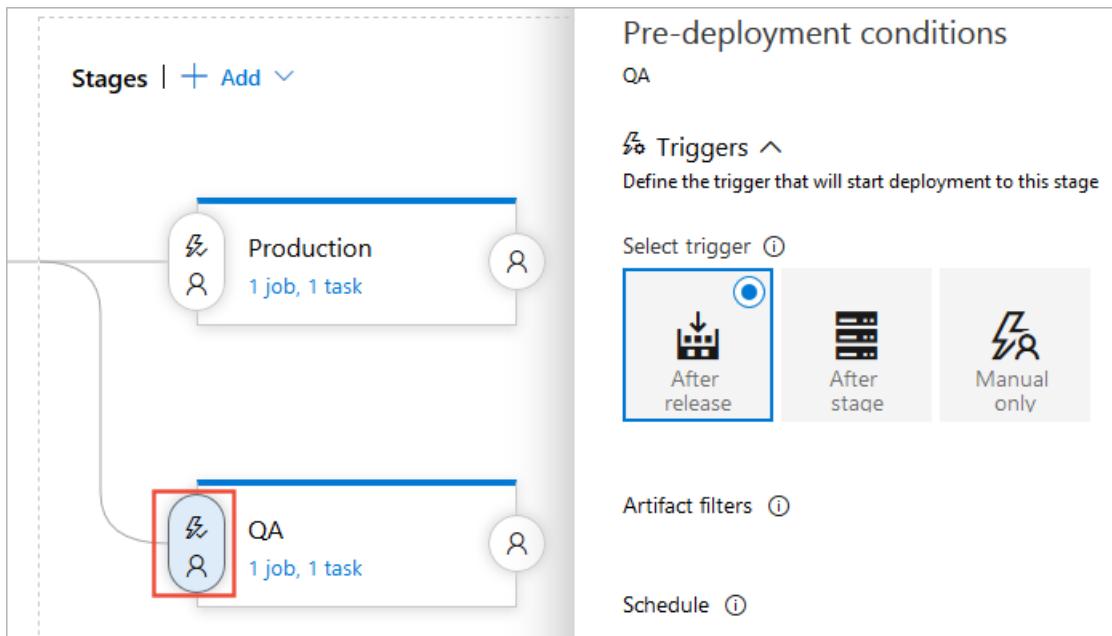


Typically, you want to use the same deployment methods with a test and a production stage so that you can be sure your deployed apps will behave the same way. Cloning an existing stage is a good way to ensure you have the same settings for both. You then just need to change the deployment targets.

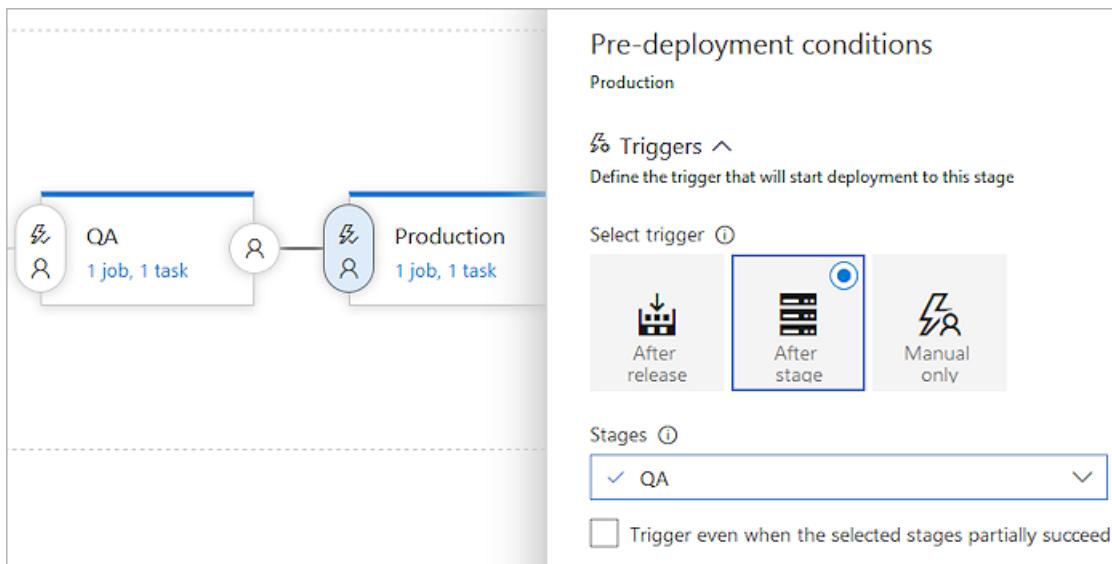
3. Your cloned stage will have the name **Copy of Production**. Select it and change the name to **QA**.



4. To reorganize the stages in the pipeline, select the **Pre-deployment conditions** icon in your **QA** stage and set the trigger to **After release**. The pipeline diagram will then show the two stages in parallel.



5. Select the **Pre-deployment conditions** icon in your **Production** stage and set the trigger to **After stage**, then select **QA** in the **Stages** drop-down list. The pipeline diagram will now indicate that the two stages will execute in the correct order.



NOTE

You can set up your deployment to start when a deployment to the previous stage is *partially* successful. This means that the deployment will continue even if a specific non-critical task have failed. This is usually used in a fork and join deployments that deploy to different stages in parallel.

6. Select the **Tasks** drop-down list and select the **QA** stage.

The screenshot shows the 'Tasks' tab of a pipeline named 'SampleApp - 1'. The pipeline has two stages: 'Production' and 'QA'. The 'QA' stage is currently selected, indicated by a red box around its name. The stage configuration panel on the right shows the following settings:

- Stage name:** QA
- Parameters:** Azure subscription: Visual Studio Enterprise (Scope: Scoped to subscription 'Visual Studio Enterprise')
- App type:** Web App
- App service name:** MySampleApp (highlighted with a red box)

7. Depending on the tasks that you are using, change the settings so that this stage deploys to your "QA" target. In our example, we will be using **Deploy Azure App Service** task as shown below.

The screenshot shows the same pipeline configuration as the previous one, but with a different setting for the 'App service name' in the QA stage. The 'App service name' field now contains 'MySampleApp-QAstage' (highlighted with a red box), indicating a specific target for this stage.

Add Pre-deployment approvals

The release pipeline we previously modified deploys to QA and production. If the deployment to QA fails, then deployment to production won't trigger. It is recommended to always verify if your app is working properly in QA or test stage before deploying to production. Adding approvals will ensure all the criteria are met before deploying to the next stage. To add approvals to your pipeline follow the steps below:

1. Select the **Pipeline** tab, **Pre-deployment conditions** icon then **Pre-deployment approvers**.

Pre-deployment conditions

Production

Triggers ▾ Define the trigger that will start deployment to this stage

Pre-deployment approvals ^ Enabled Select the users who can approve or reject deployments to this stage

2. In the **Approvers** text box, enter the user(s) that will be responsible for approving the deployment. It is also recommended to uncheck the **The user requesting a release or deployment should not approve it** check box.

Pre-deployment conditions

Production

Triggers ▾ Define the trigger that will start deployment to this stage

Pre-deployment approvals ^ Enabled Select the users who can approve or reject deployments to this stage

Approvers ⓘ Search users and groups for approvers

Timeout ⓘ 30 Days

Approval policies

The user requesting a release or deployment should not approve it (unchecked)

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 selecting the "user" icon at the right side of the stage in the pipeline diagram. For more information, see [Releases gates and approvals](#).

3. Select **Save**.

SampleApp - 1

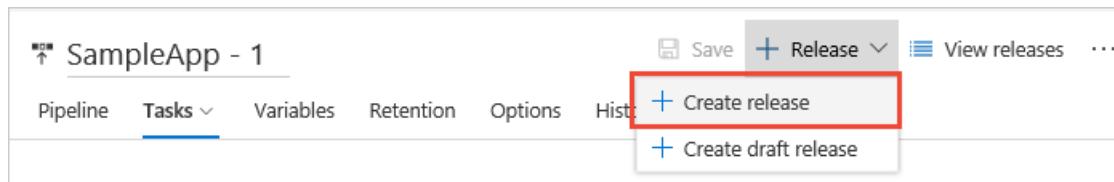
Save + Release View releases ...

Pipeline Tasks Variables Retention Options History

Create a release

Now that the release pipeline setup is complete, it's time to start the deployment. To do this, we will manually create a new release. Usually a release is created automatically when a new build artifact is available. However, in this scenario we will create it manually.

1. Select the **Release** drop-down list and choose **Create release**.



2. Enter a description for your release, check that the correct artifacts are selected, and then select **Create**.

The dialog box is titled "Create new release" for "SampleApp - 1". It contains sections for "Pipeline" (QA to Production), "Artifacts" (Source alias: SampleApp - 1, Version: 20170815.1), and "Release description" (New manual release for Sample App 1). At the bottom are "Create" and "Cancel" buttons.

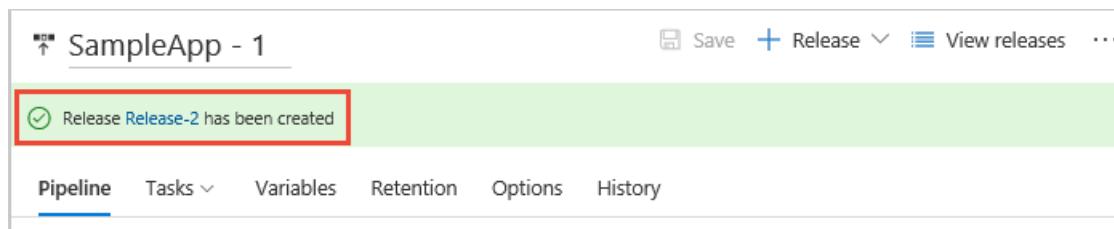
Pipeline
Click on an environment to change its trigger from automated to manual.
QA → Production

Artifacts
Select the version for the artifact sources for this release
Source alias: SampleApp - 1 Version: 20170815.1

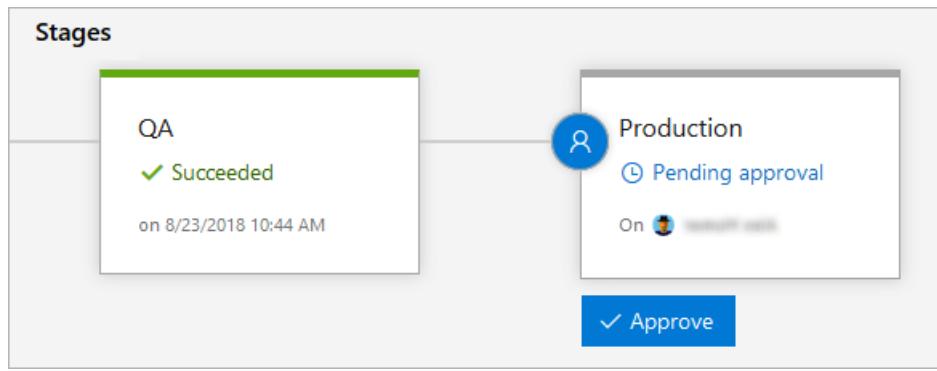
Release description
New manual release for Sample App 1

Create **Cancel**

3. A banner will appear indicating that a new release has been created. Select the release link to see more details.



4. The release summary page will show the status of the deployment to each stage.



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.

The screenshot shows a web interface for managing releases. At the top, it says "SampleApp - 1 | Edit" and has tabs for "Overview", "Releases" (which is selected), and "Deleted". Below the tabs is a toolbar with icons for refresh, add, and release status. The main area lists releases with columns for lock, title, and environments. One release, "Release-2", is highlighted. A tooltip appears over this release, displaying the text "Production: Pending pre-deployment approval".

5. Select the *pending_approval* icon to open the approval window panel. Enter a brief comment, and select **Approve**.

The screenshot shows the "Production" approval window. At the top, it says "Production" and "Pre-deployment conditions • Pending approval". Below that, there are tabs for "Approvers" (which is selected) and "View logs". Under the "Approvers" tab, it says "Approval pending for 3 minutes" and "Waiting for all approvers to approve in sequence.". There is a list of approvers, one of whom is shown with a profile picture and the status "Pending for 3 minutes". Below the approver list is a "Comment" section with a text input field containing "Checked and OK to release". At the bottom, there are "Approve" and "Reject" buttons.

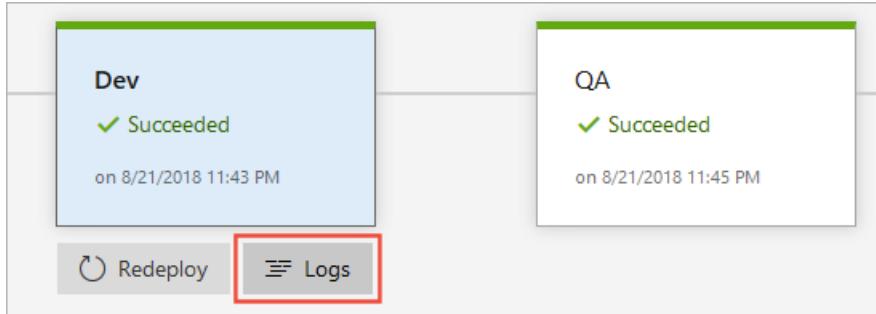
NOTE

You can schedule deployment at a later date, for example during non-peak hours. You can also reassign approval to a different user. Release administrators can access and override all approval decisions.

Monitor and track deployments

Deployment logs help you monitor and debug the release of your application. To check the logs of our deployment follow the steps below:

1. In the release summary, hover over a stage and select Logs.



During deployment, you can still access the logs page to see the live logs of every task.

2. Select any task to see the logs for that specific task. This makes it easier to trace and debug deployment issues. You can also download individual task logs, or a zip of all the log files.

The screenshot shows the 'Logs' page for the 'Dev' stage of a release named 'PartsUnlimitedE2E > Release-6'. The stage status is 'Succeeded'. The page includes navigation links for Pipeline, Tasks, Variables, Logs (which is selected), Tests, Deploy, Refresh, and Download all logs. On the left, a sidebar shows the deployment process with a single task 'Dev' marked as 'Succeeded'. The main area displays a list of tasks under the 'Dev' stage, all of which have a green checkmark and the status 'succeeded'. One task, 'Initialize Agent', has a red box drawn around it. Another red box highlights the 'Download all logs' button at the top right of the list.

3. If you need additional information to debug your deployment, you can [run the release in debug mode](#).

Next step

[Use approvals and gates to control your deployment](#)

Building a Continuous Integration and Continuous Deployment pipeline with DSC

11/2/2020 • 11 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

This example demonstrates how to build a Continuous Integration/Continuous Deployment (CI/CD) pipeline by using PowerShell, DSC, and Pester.

After the pipeline is built and configured, you can use it to fully deploy, configure and test a DNS server and associated host records. This process simulates the first part of a pipeline that would be used in a development environment.

An automated CI/CD pipeline helps you update software faster and more reliably, ensuring that all code is tested, and that a current build of your code is available at all times.

Prerequisites

To use this example, you should be familiar with the following:

- CI-CD concepts. A good reference can be found at [The Release Pipeline Model](#).
- [Git](#) source control
- The [Pester](#) testing framework
- [Desired State Configuration\(DSC\)](#)

What you will need

To build and run this example, you will need an environment with several computers and/or virtual machines.

Client

This is the computer where you'll do all of the work setting up and running the example. The client computer must be a Windows computer with the following installed:

- [Git](#)
- a local git repo cloned from https://github.com/PowerShell/Demo_CI
- a text editor, such as [Visual Studio Code](#)

Azure DevOps Subscription

An Azure DevOps organization. If you don't have one, you can [create one for free](#). (An Azure DevOps organization is different from your GitHub organization. Give them the same name if you want alignment between them.)

TFSSrv

The computer that hosts the TFS server where you will define your build and release. This computer must have

Team Foundation Server 2017 installed.

BuildAgent

The computer that runs the Windows build agent that builds the project. This computer must have a Windows build agent installed and running. See [Deploy an agent on Windows](#) for instructions on how to install and run a Windows build agent.

You also need to install both the `xDnsServer` and `xNetworking` DSC modules on this computer.

TestAgent1

This is the computer that is configured as a DNS server by the DSC configuration in this example. The computer must be running [Windows Server 2016](#).

TestAgent2

This is the computer that hosts the website this example configures. The computer must be running [Windows Server 2016](#).

Add the code to a repository

We'll start out by creating a Git repository, and importing the code from your local repository on the client computer. If you have not already cloned the Demo_CI repository to your client computer, do so now by running the following git command:

```
git clone https://github.com/PowerShell/Demo_CI
```

1. On your client computer, navigate to your TFS server in a web browser.
2. [Create a new team project](#) named Demo_CI.

Make sure that **Version control** is set to **Git**.

3. On your client computer, add a remote to the repository you just created in TFS with the following command:

```
git remote add tfs <YourTFSRepoURL>
```

Where `<YourTFSRepoURL>` is the clone URL to the TFS repository you created in the previous step.

If you don't know where to find this URL, see [Clone an existing Git repo](#).

4. Push the code from your local repository to your TFS repository with the following command:

```
git push tfs --all
```

5. The TFS repository will be populated with the Demo_CI code.

1. Navigate to your Azure DevOps subscription in a web browser.
2. [Create a new team project](#) named Demo_CI. Make sure that **Version control** is set to **Git**.
3. On your client computer, add a remote to the repository you just created with the following command:

```
git remote add devops <YourDevOpsRepoURL>
```

Where `<YourDevOpsRepoURL>` is the clone URL to the Azure DevOps repository you created in the previous step.

If you don't know where to find this URL, see [Clone an existing Git repo](#).

4. Push the code from your local repository to your TFS repository with the following command:

```
git push devops --all
```

5. The Azure DevOps repository will be populated with the Demo_CI code.

NOTE

This example uses the code in the `ci-cd-example` branch of the Git repo. Be sure to specify this branch as the default branch in your project, and for the CI/CD triggers you create.

Understanding the code

Before we create the build and deployment pipelines, let's look at some of the code to understand what is going on. On your client computer, open your favorite text editor and navigate to the root of your Demo_CI Git repository.

The DSC configuration

Open the file `DNSServer.ps1` (from the root of the local Demo_CI repository, `./InfraDNS/Configs/DNSServer.ps1`).

This file contains the DSC configuration that sets up the DNS server. Here it is in its entirety:

```

configuration DNSServer
{
    Import-DscResource -module 'xDnsServer','xNetworking', 'PSDesiredStateConfiguration'

    Node $AllNodes.Where{$_.Role -eq 'DNSServer'}.NodeName
    {
        WindowsFeature DNS
        {
            Ensure  = 'Present'
            Name    = 'DNS'
        }

        xDnsServerPrimaryZone $Node.zone
        {
            Ensure      = 'Present'
            Name       = $Node.Zone
            DependsOn = '[WindowsFeature]DNS'
        }

        foreach ($ARec in $Node.ARecords.keys) {
            xDnsRecord $ARec
            {
                Ensure      = 'Present'
                Name       = $ARec
                Zone       = $Node.Zone
                Type       = 'ARecord'
                Target     = $Node.ARecords[$ARec]
                DependsOn = '[WindowsFeature]DNS'
            }
        }

        foreach ($CName in $Node.CNameRecords.keys) {
            xDnsRecord $CName
            {
                Ensure      = 'Present'
                Name       = $CName
                Zone       = $Node.Zone
                Type       = 'CName'
                Target     = $Node.CNameRecords[$CName]
                DependsOn = '[WindowsFeature]DNS'
            }
        }
    }
}

```

Notice the `Node` statement:

```
Node $AllNodes.Where{$_.Role -eq 'DNSServer'}.NodeName
```

This finds any nodes that were defined as having a role of `DNSServer` in the [configuration data](#), which is created by the `DevEnv.ps1` script.

You can read more about the `Where` method in [about_arrays](#)

Using configuration data to define nodes is important when doing CI because node information will likely change between environments, and using configuration data allows you to easily make changes to node information without changing the configuration code.

In the first resource block, the configuration calls the `WindowsFeature` to ensure that the DNS feature is enabled. The resource blocks that follow call resources from the `xDnsServer` module to configure the primary zone and DNS records.

Notice that the two `xDnsRecord` blocks are wrapped in `foreach` loops that iterate through arrays in the configuration data. Again, the configuration data is created by the `DevEnv.ps1` script, which we'll look at next.

Configuration data

The `DevEnv.ps1` file (from the root of the local Demo_CI repository, `./InfraDNS/DevEnv.ps1`) specifies the environment-specific configuration data in a hashtable, and then passes that hashtable to a call to the `New-DscConfigurationDataDocument` function, which is defined in `DscPipelineTools.psm1` (`./Assets/DscPipelineTools/DscPipelineTools.psm1`).

The `DevEnv.ps1` file:

```
param(
    [parameter(Mandatory=$true)]
    [string]
    $OutputPath
)

Import-Module $PSScriptRoot\..\Assets\DscPipelineTools\DscPipelineTools.psd1 -Force

# Define Unit Test Environment
$DevEnvironment = @{
    Name              = 'DevEnv';
    Roles = @(
        @{
            Role      = 'DNSServer';
            VMName   = 'TestAgent1';
            Zone     = 'Contoso.com';
            ARecords = @{'TFSSrv1'=
                '10.0.0.10'; 'Client'='10.0.0.15'; 'BuildAgent'='10.0.0.30'; 'TestAgent1'='10.0.0.40'; 'TestAgent2'='10.0.0.50'}
        ;
            CNameRecords = @{'DNS' = 'TestAgent1.contoso.com'};
        }
    )
}

return New-DscConfigurationDataDocument -RawEnvData $DevEnvironment -OutputPath $OutputPath
```

The `New-DscConfigurationDataDocument` function (defined in `\Assets\DscPipelineTools\DscPipelineTools.psm1`) programmatically creates a configuration data document from the hashtable (node data) and array (non-node data) that are passed as the `RawEnvData` and `otherEnvData` parameters.

In our case, only the `RawEnvData` parameter is used.

The psake build script

The `psake` build script defined in `Build.ps1` (from the root of the Demo_CI repository, `./InfraDNS/Build.ps1`) defines tasks that are part of the build. It also defines which other tasks each task depends on. When invoked, the psake script ensures that the specified task (or the task named `Default` if none is specified) runs, and that all dependencies also run (this is recursive, so that dependencies of dependencies run, and so on).

In this example, the `Default` task is defined as:

```
Task Default -depends UnitTests
```

The `Default` task has no implementation itself, but has a dependency on the `CompileConfigs` task. The resulting chain of task dependencies ensures that all tasks in the build script are run.

In this example, the psake script is invoked by a call to `Invoke-PSake` in the `Initiate.ps1` file (located at the root of the Demo_CI repository):

```

param(
    [parameter()]
    [ValidateSet('Build','Deploy')]
    [string]
    $fileName
)

##$Error.Clear()

Invoke-PSake $PSScriptRoot\InfraDNS\$fileName.ps1

<#if($Error.count)
{
    Throw "$fileName script failed. Check logs for failure details."
}
#>

```

When we create the build definition for our example, we will supply our psake script file as the `fileName` parameter for this script.

The build script defines the following tasks:

GenerateEnvironmentFiles

Runs `DevEnv.ps1`, which generates the configuration data file.

InstallModules

Installs the modules required by the configuration `DNSServer.ps1`.

ScriptAnalysis

Calls the [PSScriptAnalyzer](#).

UnitTests

Runs the [Pester](#) unit tests.

CompileConfigs

Compiles the configuration (`DNSServer.ps1`) into a MOF file, using the configuration data generated by the `GenerateEnvironmentFiles` task.

Clean

Creates the folders used for the example, and removes any test results, configuration data files, and modules from previous runs.

The psake deploy script

The [psake](#) deployment script defined in `Deploy.ps1` (from the root of the [Demo_CI](#) repository, `./InfraDNS/Deploy.ps1`) defines tasks that deploy and run the configuration.

`Deploy.ps1` defines the following tasks:

DeployModules

Starts a PowerShell session on `TestAgent1` and installs the modules containing the DSC resources required for the configuration.

DeployConfigs

Calls the [Start-DscConfiguration](#) cmdlet to run the configuration on `TestAgent1`.

IntegrationTests

Runs the [Pester](#) integration tests.

AcceptanceTests

Runs the [Pester](#) acceptance tests.

Clean

Removes any modules installed in previous runs, and ensures that the test result folder exists.

Test scripts

Acceptance, Integration, and Unit tests are defined in scripts in the `Tests` folder (from the root of the Demo_CI repository, `./InfraDNS/Tests`), each in files named `DNSServer.tests.ps1` in their respective folders.

The test scripts use [Pester](#) and [PoshSpec](#) syntax.

Unit tests

The unit tests test the DSC configurations themselves to ensure that the configurations will do what is expected when they run. The unit test script uses [Pester](#).

Integration tests

The integration tests test the configuration of the system to ensure that when integrated with other components, the system is configured as expected. These tests run on the target node after it has been configured with DSC.

The integration test script uses a mixture of [Pester](#) and [PoshSpec](#) syntax.

Acceptance tests

Acceptance tests test the system to ensure that it behaves as expected. For example, it tests to ensure a web page returns the right information when queried. These tests run remotely from the target node in order to test real world scenarios. The integration test script uses a mixture of [Pester](#) and [PoshSpec](#) syntax.

Define the build

Now that we've uploaded our code to a repo and looked at what it does, let's define our build.

Here, we'll cover only the build steps that you'll add to the build. For instructions on how to create a build definition in Azure DevOps, see [Create and queue a build definition](#).

Create a new build definition (select the **Starter Pipeline** template) named "InfraDNS". Add the following steps to your build definition:

- PowerShell
- Publish Test Results
- Copy Files
- Publish Artifact

After adding these build steps, edit the properties of each step as follows:

PowerShell

1. Set the **targetType** property to `File Path`.
2. Set the **filePath** property to `initiate.ps1`.
3. Add `-fileName build` to the **Arguments** property.

This build step runs the `initiate.ps1` file, which calls the psake build script.

Publish Test Results

1. Set **TestResultsFormat** to `NUnit`
2. Set **TestResultsFiles** to `InfraDNS/Tests/Results/*.xml`
3. Set **TestRunTitle** to `Unit`.
4. Make sure **Control Options Enabled** and **Always run** are both selected.

This build step runs the unit tests in the Pester script we looked at earlier, and stores the results in the `InfraDNS/Tests/Results/*.xml` folder.

Copy Files

1. Add each of the following lines to **Contents**:

```
initiate.ps1
**\deploy.ps1
**\Acceptance\
**\Integration\*
```

2. Set **TargetFolder** to `$(Build.ArtifactStagingDirectory)\`

This step copies the build and test scripts to the staging directory so that they can be published as build artifacts by the next step.

Publish Artifact

1. Set **TargetPath** to `$(Build.ArtifactStagingDirectory)\`
2. Set **ArtifactName** to `Deploy`
3. Set **Enabled** to `true`.

Enable continuous integration

Now we'll set up a trigger that causes the project to build any time a change is checked in to the `ci-cd-example` branch of the git repository.

1. In TFS, click the **Build & Release** tab
2. Select the `DNS Infra` build definition, and click **Edit**
3. Click the **Triggers** tab
4. Select **Continuous integration (CI)**, and select `refs/heads/ci-cd-example` in the branch drop-down list
5. Click **Save** and then **OK**

Now any change in the git repository triggers an automated build.

Create the release definition

Let's create a release definition so that the project is deployed to the development environment with every code check-in.

To do this, add a new release definition associated with the `InfraDNS` build definition you created previously. Be sure to select **Continuous deployment** so that a new release will be triggered any time a new build is completed. ([What are release pipelines?](#)) and configure it as follows:

Add the following steps to the release definition:

- PowerShell
- Publish Test Results
- Publish Test Results

Edit the steps as follows:

PowerShell

1. Set the **TargetPath** field to `$(Build.DefinitionName)\Deploy\initiate.ps1"`
2. Set the **Arguments** field to `-fileName Deploy`

First Publish Test Results

1. Select `NUnit` for the **TestResultsFormat** field
2. Set the **TestResultsFiles** field to `$(Build.DefinitionName)\Deploy\InfraDNS\Tests\Results\Integration*.xml`

3. Set the **TestRunTitle** to `Integration`
4. Set **Condition** to `succeededOrFailed()`

Second Publish Test Results

1. Select `NUnit` for the **TestResultsFormat** field
2. Set the **TestResultsFiles** field to `$(Build.DefinitionName)\Deploy\InfraDNS\Tests\Results\Acceptance*.xml`
3. Set the **TestRunTitle** to `Acceptance`
4. Set **Condition** to `succeededOrFailed()`

Verify your results

Now, any time you push changes in the `ci-cd-example` branch, a new build will start. If the build completes successfully, a new deployment is triggered.

You can check the result of the deployment by opening a browser on the client machine and navigating to www.contoso.com.

Next steps

This example configures the DNS server `TestAgent1` so that the URL `www.contoso.com` resolves to `TestAgent2`, but it does not actually deploy a website. The skeleton for doing so is provided in the repo under the `WebApp` folder. You can use the stubs provided to create psake scripts, Pester tests, and DSC configurations to deploy your own website.

Stage templates in Azure Pipelines

3/6/2021 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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 is 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.

NOTE

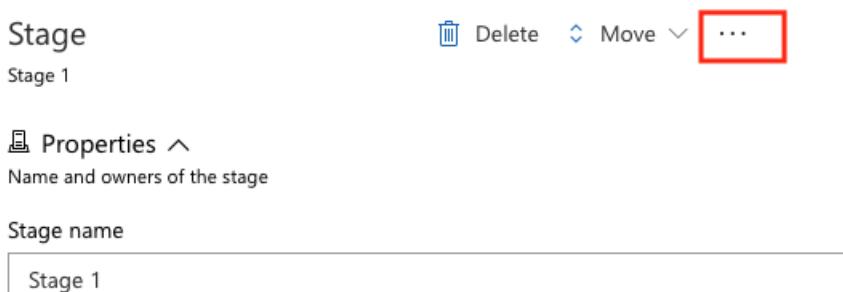
Templates do not have any additional security capability. There is no way to restrict the use of a template to specific users. All templates, pre-defined and custom, are available for use by all users who have permission to create release pipelines.

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.

Save a template

You can save a stage template from within a classic release pipeline.

1. Open and edit your release pipeline.
2. Open the stage you want to export.
3. Select the three dots and choose **Save as template**.



4. Name and save your template.

Save stage as a template X

Name *

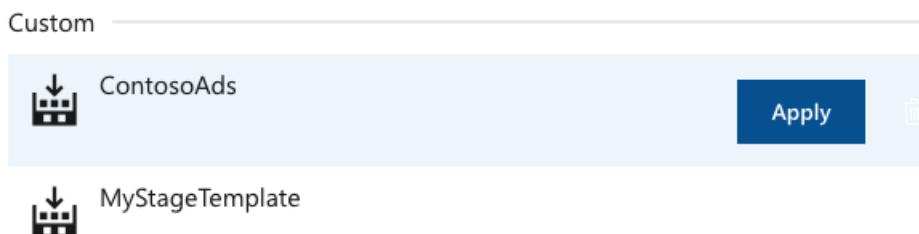
Description

(i) Any secret variable in the stage won't be saved as part of the template.

Ok Cancel

Use a template

1. From the release pipeline, select **Add** to add a stage.
2. Choose **New stage**.
3. Scroll to the end of the **Select a template** window to see your custom templates.
4. Select **Apply** to use your custom template.



FAQ

- **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.

- **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 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 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 a 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
- Get advice on [Stack Overflow](#), and feel free to post your questions, search for answers, or suggest a feature on our [Azure DevOps Developer Community](#). [Support](#) page.

View release progress and test summary

6/17/2021 • 2 minutes to read • [Edit Online](#)

Azure Pipelines provides a quick and easy way to check the status of your deployment and test results right from your pipeline definition page. The user interface provides a live update of deployment progress and easy access to logs for more details.

Visualize release progress

The user interface provides a visual representation of the current status of the release pipeline, making it easier to understand what's happening and surfaces appropriate details and actions at different stages of the deployment.

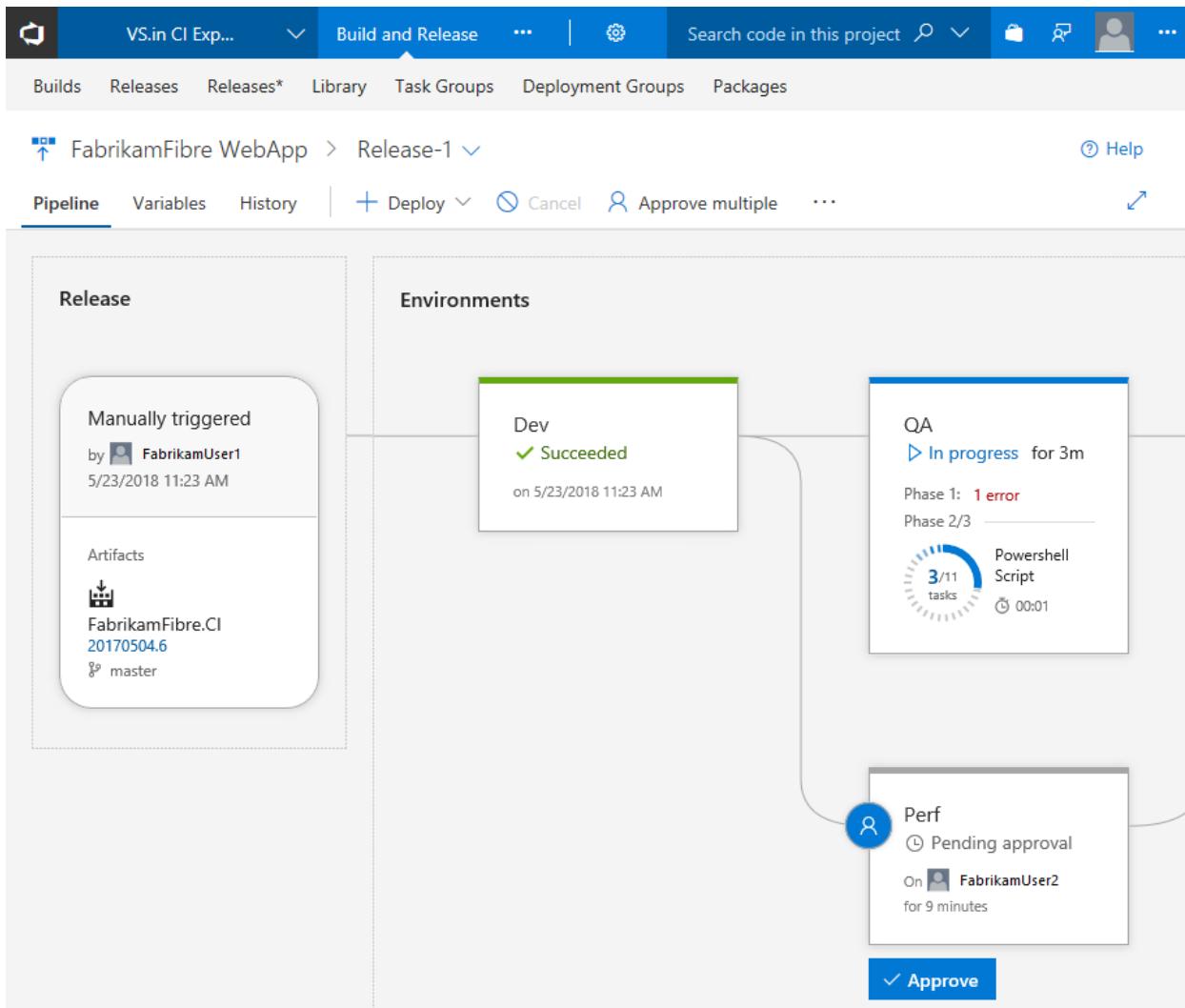
The screenshot shows the Azure Pipelines interface for a 'New release pipeline > Release-3'. The top navigation bar includes Pipeline, Variables, History, Deploy, Cancel, Refresh, Edit, and more. The main area is divided into two sections: 'Release' and 'Stages'.

Release: Shows a summary of the release trigger: 'Manually triggered by FabrikamUser on 4/14/2021, 11:17 AM'. It also indicates 'Artifacts' status: 'No artifacts'.

Stages: Shows a single stage named 'Stage 1' which is currently 'In progress'. The stage details include 'Deploy Azure App Service' and a progress indicator showing '2/2 tasks' completed in '00:02'.

Visualize release progress

The release progress view provides live updates of deployment progress and one-click access to further details. The user interface provides a visual representation of the current status of the release pipeline, making it easier to understand what's happening and surfaces appropriate details and actions at different stages of the deployment.



Pre-deployment and post-deployment

If pre-deployment or post-deployment conditions have been set for an environment, it is indicated on the environment with the presence of the approvals and gates. The progress of approvals and gates show up in the status of the environment as well. You can take action or view further details by clicking on the environment's condition icon displayed on the right or left side of the environment.

FabrikamFibre WebApp > Release-3

Pipeline Variables History + Deploy Cancel Approve multiple ...

Environments

Dev
Pending approval
On FabrikamUser1

✓ Approve

Dev

Pre-deployment conditions • Pending approval

Approvers | View logs

Approval pending Timeout in 720h
Waiting for all approvers to approve in any order.

FabrikamUser2 Pending | Override

FabrikamUser1 Pending | Reassign

Comment

Defer deployment for later

Approve Reject

You can also view the deployment gates status and a quick action buttons to view logs and debug issues if needed.

SmartHotel 360 - CD > Release-1312

Pipeline Variables History + Deploy Cancel ...

Environments

Dev ✓ Succeeded on 5/23/2018 12:39 PM

Update Staging
Processing gates

1/1 gates failed | 2m to next sample

Update Staging

Pre-deployment conditions • Processing gates

Gates | View logs

1/1 gates failed in latest sample at 12:47 PM Timeout in 48h
Waiting for all gates to succeed within the same sample interval

Deployment gates \ Samples 12:37 PM 12:42 PM 12:47 PM Next in 2m

Wait on blocking bugs Executes a work item query and che... X X X (empty)

Commits and work items

With each new release, you can see the list of associated commits and work items for each environment separately by clicking on the environment. If the list is long, use filters to find a commit or work item you're interested in.

The screenshot shows the TFS Pipeline interface. On the left, a summary card for 'Ring 0' indicates it has succeeded on 5/18/2018 at 11:19 AM. To the right, a detailed view of the 'Commits' tab for the 'VS.Release.CI' environment is shown. It lists 40 commits from 'FabrikamUser' with messages related to issue fixes and cleanup. A filter bar at the top of this section allows users to search by author ('FabrikamUser') or clear filters.

Commit	Message
1b8fa915	Merged PR 344152: Fix issue with duplicate identity being in...
cf8866c4	Minor cleanup in ThreadPropertiesConverter.
5714cf49	Move FailOnTrace condition to ThreadPropertiesConverter tests.
8a578f2f	Fix issues with inconsistent case in Guids used in thread prope...
ee03d36e	Add tests to verify whether or not a duplicate identity can is b...

Deployment progress and logs

The environments show live updates for in-progress deployments, including how many phases and tasks are complete and the running time. Clicking the environment status opens a view containing the logs, with focus on the what's currently active.

The screenshot shows the TFS Logs view for a deployment step named 'Run on agent'. The status is 'Failed'. The log details the execution of tasks: 'Initialize Agent' succeeded (0:07), 'Initialize Job' succeeded (0:04), 'Download artifact - FabrikamFibre.Cl' succeeded (0:02), 'Stop App' succeeded (0:02), and 'Deploy Azure App Service' failed with an error message: 'Error: Resource 'webappp' doesn't exist. Resource should exist before deployment.' (0:02). The final task, 'Restart App', was skipped.

Task	Status	Time
Initialize Agent	succeeded	0:07
Initialize Job	succeeded	0:04
Download artifact - FabrikamFibre.Cl	succeeded	0:02
Stop App	succeeded	0:02
Deploy Azure App Service	1 error	0:02
Error: Resource 'webappp' doesn't exist. Resource should exist before deployment.		
Restart App	skipped	

You can also click into the logs to enter a focused view for each of the tasks.

The screenshot shows the Azure DevOps build logs interface. At the top, there are navigation links: 'Deploy Azure App Service' (with a red X), 'Previous task' (with a left arrow), 'Next task' (with a right arrow), 'Download log' (with a downward arrow), and a close button ('X'). Below this, there's a sidebar with sections for 'Build' (selected), 'Deployment' (Failed), and 'Logs'. The main area displays deployment logs:

```

1 2018-05-23T19:21:49.1042979Z ##[section]Starting: Deploy Azure App Service
2 2018-05-23T19:21:49.1070989Z =====
3 2018-05-23T19:21:49.1071220Z Task       : Azure App Service Deploy
4 2018-05-23T19:21:49.1071670Z Description : Update Azure WebApp Services On Windows, Web App On Lin
5 2018-05-23T19:21:49.1071986Z Version    : 3.3.47
6 2018-05-23T19:21:49.1072135Z Author     : Microsoft Corporation
7 2018-05-23T19:21:49.1072324Z Help      : [More Information](https://aka.ms/azurermwebdeployreadm)
8 2018-05-23T19:21:49.1072523Z =====
9 2018-05-23T19:21:50.1220008Z Got connection details for Azure App Service:'webappp'
10 2018-05-23T19:21:51.1698382Z ##[error]Error: Resource 'webappp' doesn't exist. Resource should exist.
11 2018-05-23T19:21:51.1721952Z ##[section]Finishing: Deploy Azure App Service
12

```

View build results

The build results view provides insights into the multiple components that make up your build. The user interface gives a snapshot view of the status of your current and previous builds. You can also view your logs, issues, commits, and all the work items related to your build.

The screenshot shows the VSO.CI build results page for build VSO.CI_20180611.44. At the top, there's a green circular icon with a checkmark, the build name, and a progress bar. Below this, there are tabs for 'Logs', 'Timeline' (selected), 'Code coverage*', 'Tests', 'Build Targets', and 'WhiteSource Bolt Build Report'. There's also a 'Edit pipeline' button and a three-dot menu.

Progression

Build process succeeded
0 error(s) / 0 warning(s)

Queued 3 hours ago
Ran for 21:33

Associated changes
3 commit(s)

- Merged PR 352974: null ref in task definition validator
Eric authored ⌂ 5fcaa92 3 hours ago
- Merged PR 352576: Fix SSH V2 URL handling for codex
Congyi authored ⌂ 880e507 3 hours ago
- Merged PR 352852: fix is empty issues
Chen authored ⌂ d9c22da 3 hours ago

Linked work items

- Null reference exception in Microsoft.TeamFoundation.DistributedTask.Core.Server.Plugins.TaskDefinitionExtensionPublishHandler.UpdateExt...
Bug 1239861 · Assigned to Prabhu
- Null reference exception in Microsoft.TeamFoundation.DistributedTask.Core.Server.Plugins.TaskDefinitionExtensionPublishHandler.UpdateExt...
Bug 1239861 · Assigned to Prabhu

Review test summary

During test execution, a test might spawn multiple instances of tests that contribute to the overall outcome. A few examples include: tests that rerun due to failures, tests composed of an ordered combination of other tests (for example, ordered test), or tests having different instances based on provided input parameter (data-driven tests). Since these tests are related, they need to be reported together with the overall outcome derived based on the individual test outcomes. The summary page provides insights on the result of the test runs related to your pipeline.

Summary ▾

4 Run(s) Completed (4 Passed, 0 Failed, 0 Not impacted, 0 Others)

1063

Total tests



1063 • Passed
Passed on rerun (2)
0 • Failed
0 • Others

100%

Pass percentage

4

Run

[Bug](#) [Link](#)[Test run](#) [Column Options](#)[Filter by test name](#)

Container

[Clear](#)

Test

▼ TFS.SelfHost.CodeDev (2/1306)

- ✓ ✓ ExportWorkItemTypeDefinitionFuzzTest
 - ✓ Attempt# 1 - ExportWorkItemTypeDefinitionFuzzTest
 - ✗ Attempt# 0 - ExportWorkItemTypeDefinitionFuzzTest
- ✓ ✓ UpdateWorkItemTypeDefinitionFuzzTest
 - ✓ Attempt# 1 - UpdateWorkItemTypeDefinitionFuzzTest
 - ✗ Attempt# 0 - UpdateWorkItemTypeDefinitionFuzzTest

UpdateWorkItemTy

✓ Passed 2 days ago on R

[Debug](#) [Work items](#)

Error message ▾

```
Test method MS.TF.Test.W
System.Exception: Microso
at Microsoft.VisualStudio.T
at MS.TF.Test.WIT.REST.T
at Common.TestFramework
at Common.TestFramework
```

Stack trace ▾

```
at Common.TestFramew
at Common.TestFramew
at Common.TestFramew
at Common.TestFramew
at MS.TF.Test.WIT.REST.T
```

NOTE

Metrics in the test summary section (e.g. Total tests, Passed, etc.), are computed using the root level of the hierarchy rather than each individual iteration of the tests.

Related articles

- [Release pipelines overview](#)
- [Classic release pipelines](#)
- [Stage templates in Azure Pipelines](#)

Deploy a web app to an nginx web server on a Linux Virtual Machine

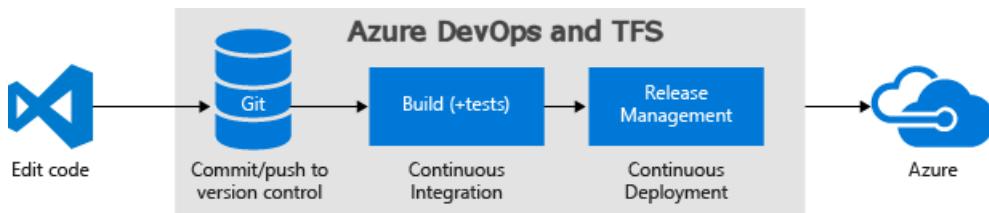
3/6/2021 • 6 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018

NOTE

If you want to deploy your application to a Linux virtual machine using YAML, see [Deploy to a Linux virtual machine](#).

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 and higher. 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. Set up a CI build pipeline based on the runtime you want to use.

- [Java](#)
- [JavaScript](#)

If you already have an app in GitHub that you want to deploy, you can create a pipeline for that code.

If you are a new user, fork this repo in GitHub:

```
https://github.com/spring-guides/gs-spring-boot-docker.git
```

Follow additional steps mentioned in [Build your Java app with Maven](#) for creating a build to deploy to Linux.

Prerequisites for the Linux VM

The deployment scripts used in the above sample repositories have been tested on Ubuntu 16.04, and we recommend you use the same version of Linux VM for this quickstart. Follow the additional steps described below based on the runtime stack used for the app.

- [Java](#)
- [JavaScript](#)

- For deploying Java Spring Boot and Spring Cloud based apps, create a Linux VM in Azure using [this template](#), which provides a fully supported OpenJDK-based runtime.
- For deploying Java servlets on Tomcat server, create a Linux VM with Java 8 using [this Azure template](#) and [configure Tomcat 9.x as a service](#).
- For deploying Java EE based app, use an Azure template to create a [Linux VM + Java + WebSphere 9.x](#) or a [Linux VM + Java + WebLogic 12.x](#) or a [Linux VM +Java](#) + WildFly/JBoss 14

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 Prerequisites for installing the build and release agent onto a Ubuntu 16.04 VM. Prerequisites 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, 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.

The screenshot shows the 'Summary' tab of a build named 'PartsUnlimitedE2E' from '20180821.1'. It includes a large green circle with a white checkmark icon. Below it, there's a repository summary: 'PartsUnlimited · master · 93555b9 : Clean up markdown in README.' and 'Retained by release'. There are buttons for 'Add a tag' and 'Edit'. At the bottom, there are tabs: 'Logs', 'Summary' (which is selected and underlined), 'Code coverage*', 'Tests', 'Artifacts', 'Release' (which is highlighted with a red box), and 'Edit'.

- Open the **Releases** tab of **Azure Pipelines**, open the **+** drop-down in the list of release pipelines, and choose **Create release pipeline**.

The screenshot shows the 'Releases' tab of the Azure Pipelines interface. On the left, there's a search bar and a dropdown menu with options: 'Create a release pipeline' (which is highlighted with a red box), 'Import release pipeline', and 'All release pipelines'. The main area shows a list titled 'All release pipelines' with tabs for 'Overview', 'Releases' (which is selected), 'Analytics*', and 'Deleted'. There are also filters for 'Title' and a refresh button.

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.

The screenshot shows the 'Pipeline' tab for a new release pipeline. On the left, there are sections for 'Artifacts' (with a '+ Add' button highlighted with a red box) and 'Stages'. On the right, there's a modal window titled 'Add an artifact' with the following fields:

- Source type:** Build (highlighted with a blue box)
- Project:** PartsUnlimited
- Source (build pipeline):** PartsUnlimitedE2E (highlighted with a blue box)
- Default version:** Latest
- Source alias:** _PartsUnlimitedE2E

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 **main** branch.

The screenshot shows the 'Continuous deployment trigger' configuration for a release pipeline named '_PartsUnlimitedE2E'. It includes sections for 'Artifacts' (with a red box around the 'PartsUnlimitedE2E' artifact), 'Stages' (with a red box around the 'Schedule not set' icon), and 'Build branch filters' (with a red box around the 'Type' dropdown set to 'Build branch'). A note at the bottom states: '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 release pipeline. It lists a single 'Agent job' under 'Stage 1'. A red box highlights the 'Remove' button next to the job. 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 release pipeline. The 'Stage 1' row has a red box around the '...' button. A dropdown menu is open, showing options: 'Add an agent job', 'Add a deployment group job' (which is highlighted with a red box), and 'Add an agentless job'. A note at the top of the stage row says: 'The stage 'Stage 1' should have at least one job.'

7. For the **Deployment Group**, select the deployment group you created earlier such as **myNginx**.

All pipelines > **New release pipeline**

Pipeline Tasks Variables Retention Options History

Stage 1
Deployment process

Deployment group job + ⋮

Deployment group job **Deployment group job** **Remove**

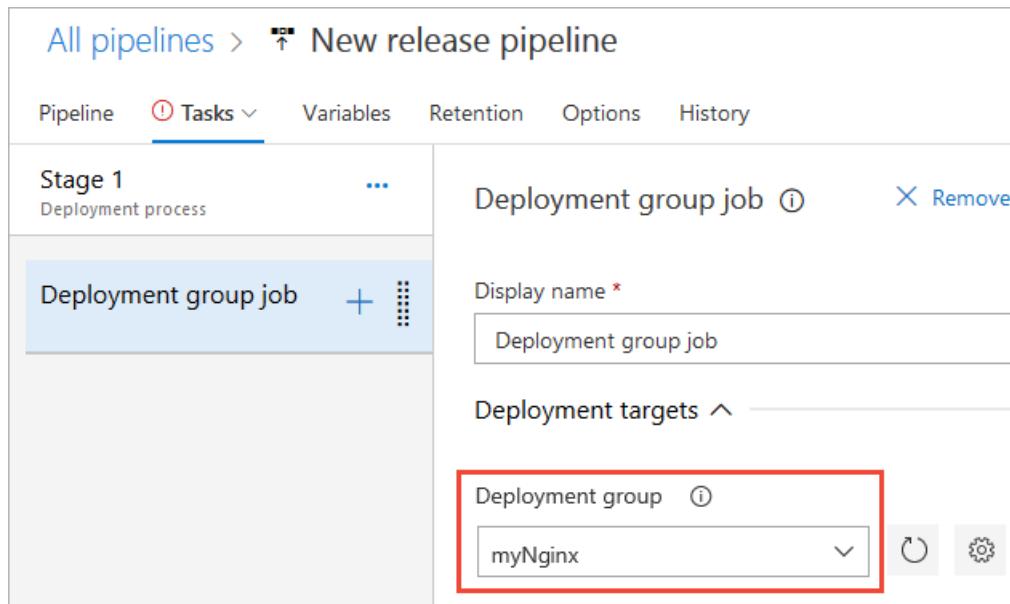
Display name *

Deployment group job

Deployment targets ^

Deployment group ⓘ

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 Bash task.

Pipeline **Tasks** Variables Retention Options History

Stage 1
Deployment process

Deployment group job **+ ⋮**

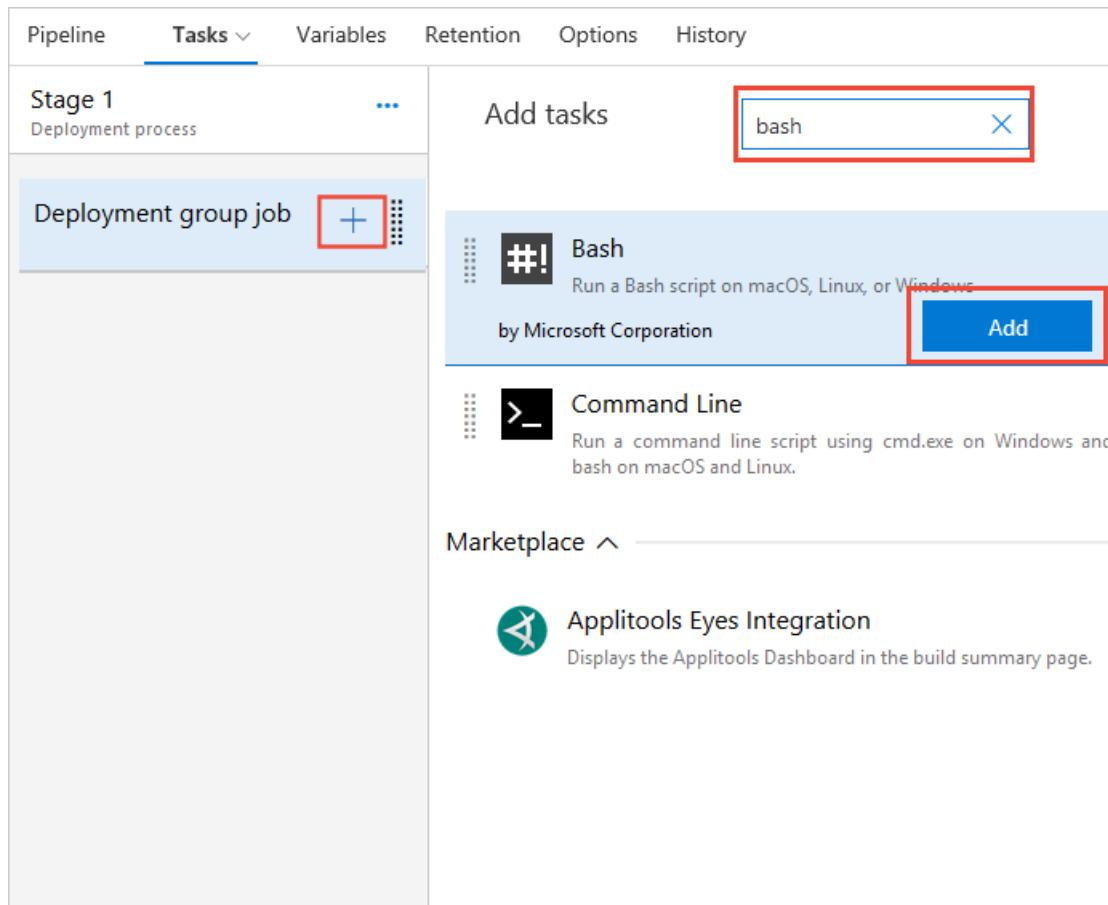
Add tasks **bash** X

Bash
Run a Bash script on macOS, Linux, or Windows
by Microsoft Corporation **Add**

Command Line
Run a command line script using cmd.exe on Windows and bash on macOS and Linux.

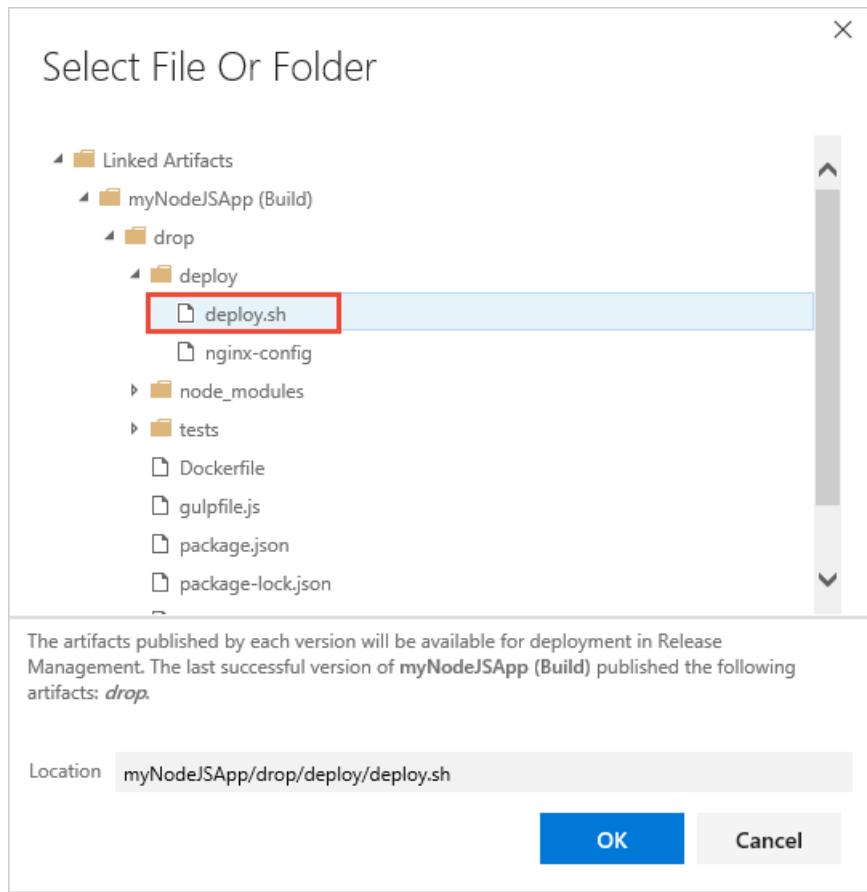
Marketplace ^

Applitools Eyes Integration
Displays the Applitools Dashboard in the build summary page.



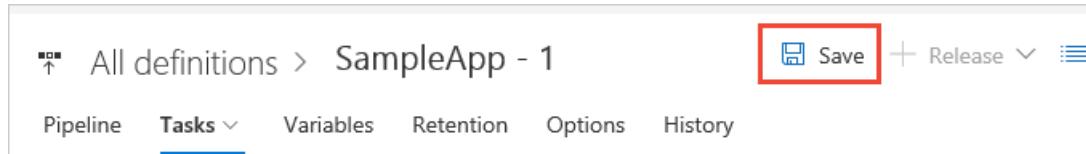
9. In the properties of the **Bash** task, use the **Browse** button for the **Script Path** to select the path to the **deploy.sh** script in the build artifact. For example, when you use the **nodejs-sample** repository to build your app, the location of the script is

```
$(System.DefaultWorkingDirectory)/nodejs-sample/drop/deploy/deploy.sh .
```



See a [sample deploy.sh file](#) for a Node.js web app.

10. 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

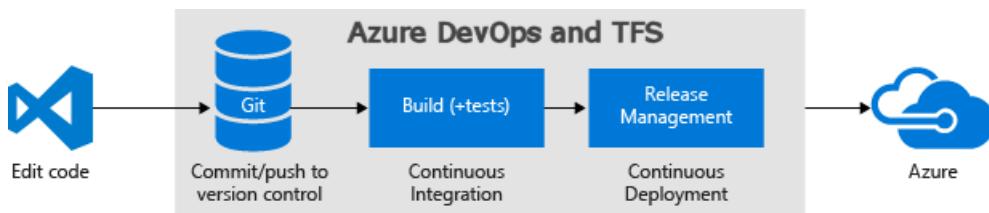
- [Dynamically create and remove a deployment group](#)
- [Apply stage-specific configurations](#)
- [Perform a safe rolling deployment](#)
- [Deploy a database with your app](#)

Deploy to a Windows Virtual Machine

3/6/2021 • 4 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018

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://dotnet.microsoft.com/permalink/dotnetcore-current-windows-runtime-bundle-installer -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
Stop-Service was -Force
Start-Service w3svc

```

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 **main** 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](#)

Deploy your Web Deploy package to IIS servers using WinRM

3/6/2021 • 7 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

A simpler way to deploy web applications to IIS servers is by using [deployment groups](#) instead of WinRM. However, deployment groups are not available in version of TFS earlier than TFS 2018.

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 latest successful build of DotNetSample-ASP.NET Core-C

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 **main** branch.

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 FQDN.
- **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.

FAQ

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

- Get advice on [Stack Overflow](#), and feel free to post your questions, search for answers, or suggest a feature on our [Azure DevOps Developer Community Support](#) page.

How To: Extend your deployments to IIS Deployment Groups

2/26/2020 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018

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, `<connectionStringKey> = <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 the 'SampleApp - 1' pipeline configuration. On the left, there's a sidebar with 'Production' selected under 'Deployment process'. A red box highlights the 'Deployment group job' section, which contains a 'Display name' field set to 'Deployment group job'. Below it is an 'IIS Manage IISWebsite' task. The main area shows a 'Deployment group job' configuration with a red box around its settings. It includes fields for 'Deployment targets' (set to 'Deployment group'), 'Required tags' (empty), and 'Targets to deploy to in parallel'. Under 'Targets to deploy to in parallel', 'Multiple' is selected (radio button checked) and 'One target at a time' is unselected. A slider for 'Maximum number of targets in parallel' is set to 50% (0 targets). Other settings include 'Timeout' (0) and 'Deployment job cancel timeout in minutes' (1). The top right of the page has 'Save' and 'Release' buttons.

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.
- **Required 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.
- **Required tags:** `database`

Then add a SQL Server Database Deploy task to this job.

Deploy with System Center Virtual Machine Manager

11/2/2020 • 8 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | 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.

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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 SCVM 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](#)

Deploy to VMware vCenter Server

2/26/2020 • 5 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

You can automatically provision virtual machines in a VMware environment and deploy to those virtual machines after every successful build.

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 Azure DevOps Server connects to vCenter. 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.

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 Azure

7/1/2021 • 2 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [Azure DevOps Server 2020](#) | [Azure DevOps Server 2019](#) | [TFS 2018 - TFS 2015](#)

Azure Pipelines combines continuous integration (CI) and continuous delivery (CD) to test and build your code and ship it to any target. While you do not have to use Azure services with Pipelines, Pipelines can help you take advantage of Azure. You can use Pipelines to integrate your CI/CD process with most Azure services.

To learn more about selecting an Azure service for hosting your application code, see [Choose an Azure compute service for your application](#).

Service	Description	Resources
Azure Web App Service / App Service for Containers	A managed service for hosting web apps, mobile app back ends, RESTful APIs, or automated business processes	Deploy an Azure Web App , Deploy an Azure Web App Container , Deploy to Azure App Service using Visual Studio Code , Azure App Service Deploy task , Azure App Service Manage task , Azure App Service Settings task
Azure Static Web Apps	Automatically builds and deploys full stack web apps to Azure from a code repository	Tutorial: Publish Azure Static Web Apps with Azure DevOps
Azure Functions	Serverless solution for hosting applications	Deploy an Azure Function , Azure Function App task , Azure Function App for Containers task
Azure Databases (SQL, MySQL)	Managed database solutions	Deploy to Azure SQL Database , Azure SQL Database Deployment task , Azure Database for MySQL Deployment task
Azure DevOps Starter	Azure portal tool for quickly setting up a CI/CD pipeline	Overview of DevOps Starter
Azure App Configuration	Service to centrally manage application settings and feature flags	Push settings to App Configuration with Azure Pipelines , Pull settings to App Configuration with Azure Pipelines
Azure Blob Storage	Object storage solution for storing large amounts of unstructured data in the cloud	Azure File Copy task
Azure Kubernetes Service	Managed Kubernetes service for running containerized applications	Build and deploy to Azure Kubernetes Service
Azure Machine Learning	Cloud-based environment for working with machine learning models	Prepare data, train, deploy, and monitor machine learning models with Azure Pipelines

SERVICE	DESCRIPTION	RESOURCES
Azure Resource Manager	Deployment and management service for Azure. Azure Resource Manager templates (ARM templates) are a way of quickly deploying infrastructure as code.	Integrate ARM templates with Azure Pipelines , Quickstart: Use an ARM template to deploy a Linux web app to Azure
Azure Container Registry	Managed, private Docker registry service	Build and push Docker images to Azure Container Registry
Azure Service Fabric	Distributed systems platform that can run in many environments, including Azure or on premises	Tutorial: Deploy an application with CI/CD to a Service Fabric cluster , Service Fabric Application Deployment task
Azure IoT Edge	Managed service built on Azure IoT Hub.	Continuous integration and continuous deployment to Azure IoT Edge devices , Create a CI/CD pipeline for IoT Edge with Azure DevOps Starter
Azure Key Vault	Managed service for storing secret data.	Use Azure Key Vault secrets in Azure Pipelines , Azure Key Vault task

Connect to Microsoft Azure

3/18/2021 • 7 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

To deploy your app to an Azure resource (to an app service or to a virtual machine), you need an Azure Resource Manager service connection.

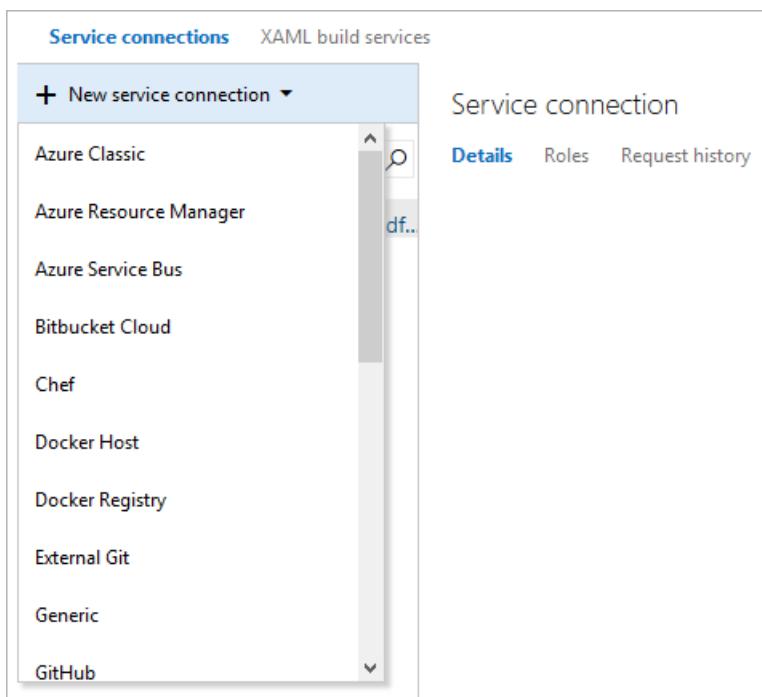
For other types of connection, and general information about creating and using connections, see [Service connections for builds and releases](#).

Create an Azure Resource Manager service connection using automated security

We recommend this simple approach if:

- You're signed in as the owner of the Azure Pipelines organization and the Azure subscription.
- You don't need to further limit the permissions for Azure resources accessed through the service connection.
- You're not connecting to [Azure Stack](#) or an [Azure Government Cloud](#).
- You're not connecting from Azure DevOps Server 2019 or earlier versions of TFS

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. Specify the following parameters.

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 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're using the classic editor, select the connection name you assigned in the **Azure subscription** setting of your pipeline.
- If you're using YAML, copy the connection name into your code as the `azureSubscription` value.

5. To deploy to a specific Azure resource, the task will need additional data about that resource.

- If you're using the classic editor, select data you need. For example, the App service name.
- If you're using YAML, then go to the resource in the Azure portal, and then copy the data into your code. For example, to deploy a web app, you would copy the name of the App Service into the `WebAppName` value.

NOTE

When you follow this approach, Azure DevOps *connects with Azure Active Directory (Azure AD) and creates an app registration with a secret that's valid for two years*. When the service connection is close to two years old, Azure AD displays this prompt: **A certificate or secret is expiring soon. Create a new one.** In this scenario, you must refresh the service connection.

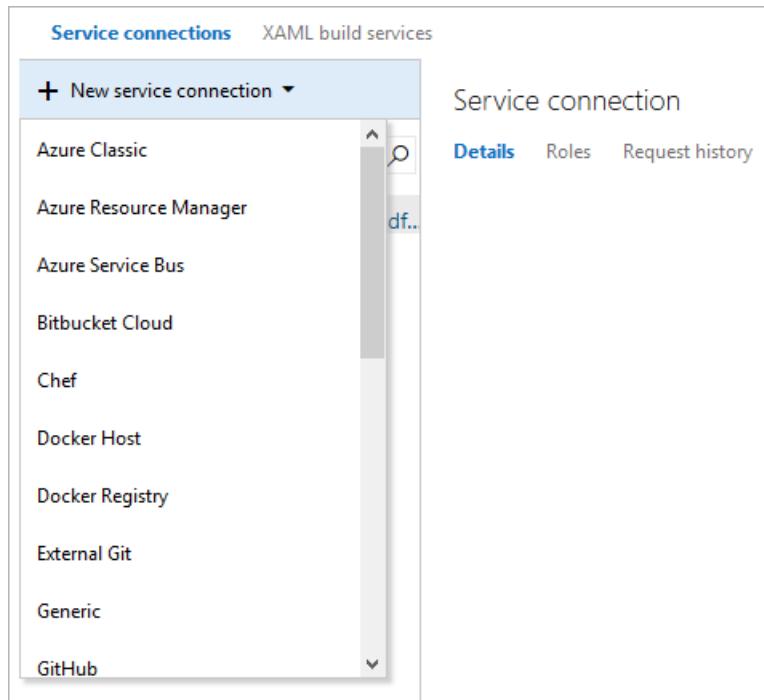
To refresh a service connection, in the Azure DevOps portal, edit the connection and select **Verify**. After you save the edit, the service connection is valid for another two years.

See also: [Troubleshoot Azure Resource Manager service connection](#).

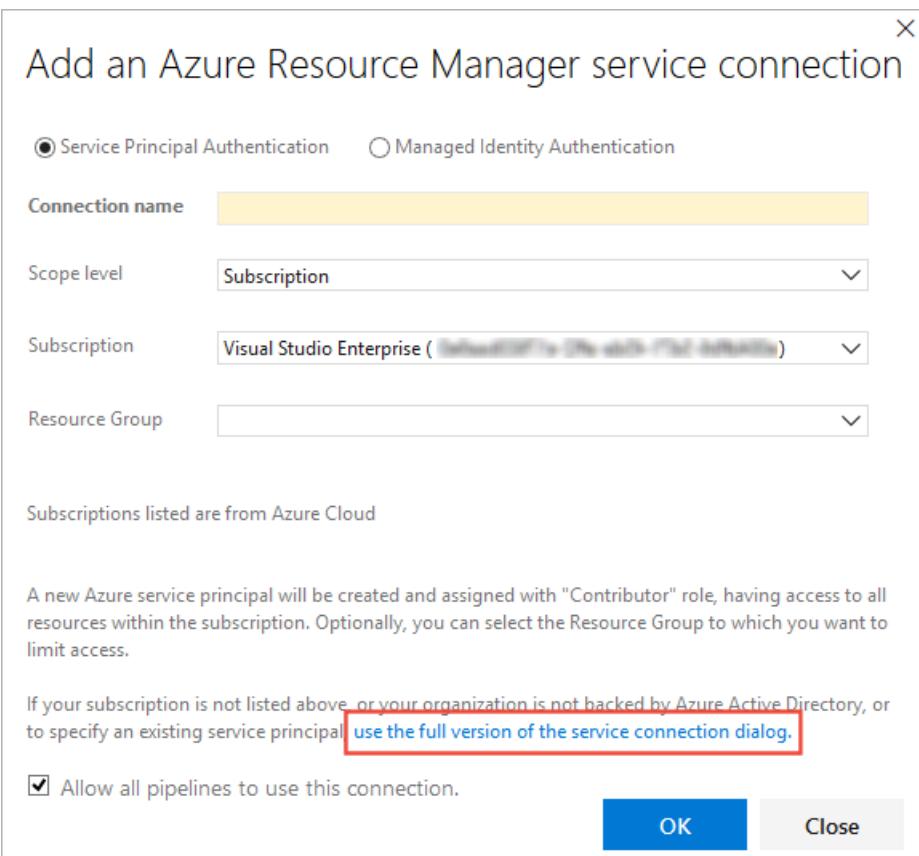
If you have problems using this approach (such as no subscriptions being shown in the drop-down list), or if you want to further limit users' permissions, you can instead use a [service principal](#) or a [VM with a managed service identity](#).

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 a service principal that can access resources](#)
 - [Use Azure PowerShell to create an Azure service principal with a certificate](#)
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**.



4. Choose Service Principal (manual) option and enter the Service Principal details.



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. Enter the information about your service principal into the Azure subscription dialog textboxes:
 - Subscription ID
 - Subscription name
 - Service principal ID
 - Either the service principal client key or, if you have selected **Certificate**, enter the contents of both the certificate and private key sections of the *.pem file.
 - Tenant IDYou can obtain this information if you don't have it to hand by downloading and running [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. Choose **Verify connection** to validate the settings you entered.
11. 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.
12. 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](#).

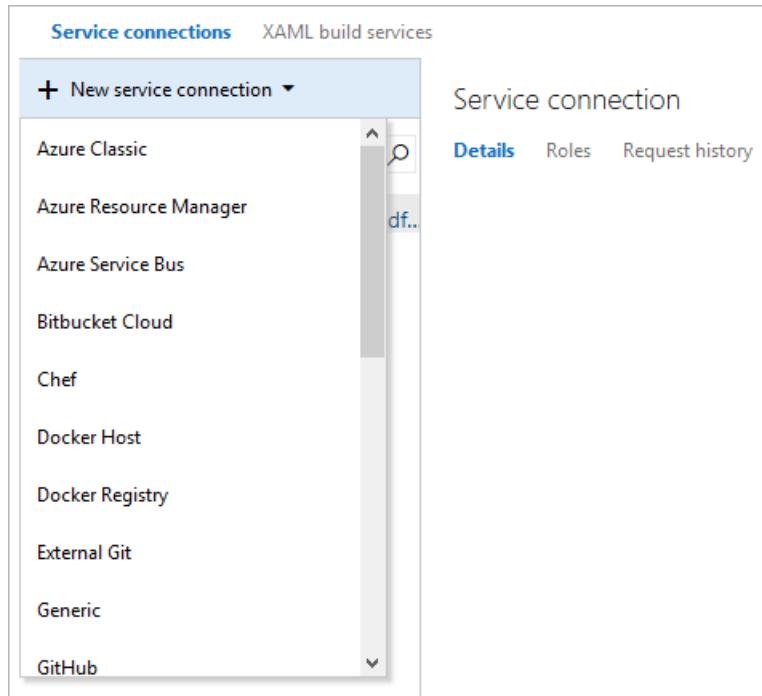
Create an Azure Resource Manager service connection to a VM with a managed service identity

NOTE

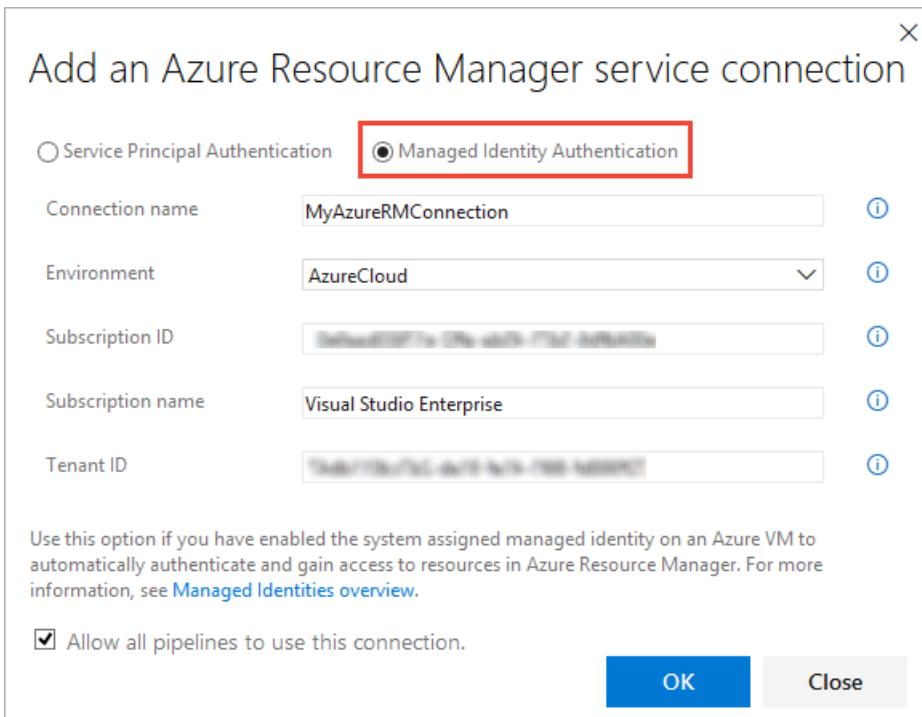
You are required to use a self-hosted agent on an Azure VM in order to use managed service identity

You can configure Azure Virtual Machines (VM)-based agents with an [Azure Managed Service Identity](#) in Azure Active Directory (Azure AD). This lets you use the system assigned identity (Service Principal) to grant the Azure VM-based agents access to any Azure resource that supports Azure AD, such as Key Vault, instead of persisting credentials in Azure DevOps for the 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. Select the [Managed Identity Authentication](#) option.



4. Enter a user-friendly **Connection name** to use when referring to this service connection.
5. Select the **Environment** name (such as Azure Cloud, Azure Stack, or an Azure Government Cloud).
6. Enter the values for your subscription into these fields of the connection dialog:
 - Subscription ID
 - Subscription name
 - Tenant ID
7. 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.
8. Ensure that the VM (agent) has the appropriate permissions. For example, if your code needs to call Azure Resource Manager, assign the VM the appropriate role using Role-Based Access Control (RBAC) in Azure AD. For more details, see [How can I use managed identities for Azure resources?](#) and [Use Role-Based Access Control to manage access to your Azure subscription resources](#).

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
- Get advice on [Stack Overflow](#), and feel free to post your questions, search for answers, or suggest a feature on our [Azure DevOps Developer Community](#). [Support](#) page.

Deploy an Azure Web App (Linux)

11/2/2020 • 5 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

You can automatically deploy your web app to an Azure App Service Linux on every successful build.

NOTE

This guidance applies to Azure DevOps Services.

Before you begin

Based on the desired runtime, [import](#) (into Azure DevOps) or fork (into GitHub) the following repository.

- [.NET Core](#)
- [Java](#)
- [Nodejs](#)

If you already have an app in GitHub that you want to deploy, you can try creating a pipeline for that code.

However, if you are a new user, then you might get a better start by using our sample code. In that case, fork this repo in GitHub:

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

Build your app

- [YAML](#)
- [Classic](#)
- [.NET Core](#)
- [Java](#)
- [Nodejs](#)

Follow the [Build, test, and deploy .NET Core apps](#) till [Create your first pipeline](#) section to set up the build pipeline. When you're done, you'll have a YAML pipeline to build, test, and publish the source as an artifact.

We advise new users to use Classic Editor and not use YAML pipelines to deploy from Azure DevOps Services. If you're an experienced pipeline user and already have a YAML pipeline to build your .NET Core app, then you might find the examples below useful.

YAML pipelines aren't available on TFS.

Now you're ready to read through the rest of this topic to learn some of the more common configurations to customize the deployment of the Azure Web App.

Azure service connection

The [Azure Web App](#) 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 Azure DevOps Server to Azure.

- [YAML](#)
- [Classic](#)

You must supply an Azure service connection to the `AzureWebApp` task. Add the following YAML snippet to your existing `azure-pipelines.yaml` file. Make sure you add the service connection details in the variables section as shown below.

```
variables:  
  ## Add this under variables section in the pipeline  
  azureSubscription: <Name of the Azure subscription>  
  appName: <Name of the Web App>  
  
## Add the below snippet at the end of your pipeline  
- task: AzureWebApp@1  
  displayName: 'Azure Web App Deploy'  
  inputs:  
    azureSubscription: $(azureSubscription)  
    appType: webAppLinux  
    appName: $(appName)  
    package: $(System.ArtifactsDirectory)/**/*.zip
```

YAML pipelines aren't available on TFS.

Deploy with Azure Web App

- [YAML](#)
- [Classic](#)

The simplest way to deploy to an Azure Web App is to use the [Azure Web App](#) task.

To deploy to an Azure Web App, add the following snippet at the end of your `azure-pipelines.yml` file:

```

trigger:
- main

variables:
# Azure service connection established during pipeline creation
azureSubscription: <Name of your Azure subscription>
appName: <Name of the web app>
# Agent VM image name
vmImageName: 'ubuntu-latest'

- task: AzureWebApp@1 # Add this at the end of your file
inputs:
  azureSubscription: <Azure service connection>
  appType: webAppLinux
  appName: $(appName)
  package: $(System.ArtifactsDirectory)/**/*.zip

```

The snippet assumes that the build steps in your YAML file build and publishes the source as an artifact. The **Azure Web App Deploy** task will pull the artifact corresponding to the **BuildId** from the **Source type** specified, and then deploys the artifact to the Linux App Service.

YAML pipelines aren't available on TFS.

Deploy to a slot

- [YAML](#)
- [Classic](#)

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 YAML snippet shows how to deploy to a staging slot, and then swap to a production slot:

```

- task: AzureWebApp@1
inputs:
  azureSubscription: '<Azure service connection>'
  appType: webAppLinux
  appName: '<name of web app>'
  deployToSlotOrASE: true
  resourceGroupName: '<name of resource group>'
  slotName: staging

- task: AzureAppServiceManage@0
inputs:
  azureSubscription: '<Azure service connection>'
  appType: webAppLinux
  WebAppName: '<name of web app>'
  ResourceGroupName: '<name of resource group>'
  SourceSlot: staging
  SwapWithProduction: true

```

YAML pipelines aren't available on TFS.

Deploy an Azure Web App Container

11/2/2020 • 5 minutes to read • [Edit Online](#)

You can automatically deploy your web app to an [Azure Web App for Linux Containers](#) after every successful build.

Before you begin

Based on the desired runtime, [import](#) (into Azure DevOps) or fork (into GitHub) the following repository.

- [.NET Core](#)
- [Java](#)
- [Nodejs](#)

If you already have an app in GitHub that you want to deploy, you can try creating a pipeline for that code.

However, if you are a new user, then you might get a better start by using our sample code. In that case, fork this repo in GitHub:

```
https://github.com/MicrosoftDocs/pipelines-dotnet-core-docker
```

Build your app

- [YAML](#)
- [Classic](#)
- [.NET Core](#)
- [Java](#)
- [Nodejs](#)

Follow the [Build, test, and push Docker container apps](#) till [push an image](#) section to set up the build pipeline. When you're done, you'll have a YAML pipeline to build, test, and push the image to container registry.

We aren't yet advising new users to use YAML pipelines to deploy from Azure DevOps Server 2019. If you're an experienced pipeline user and already have a YAML pipeline to build your .NET Core app, then you might find the examples below useful.

YAML pipelines aren't available on TFS.

Now that the build pipeline is in place, you will learn a few more common configurations to customize the deployment of the Azure Container Web App.

Azure service connection

The **Azure WebApp Container** 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 Azure DevOps Server to Azure.

- [YAML](#)
- [Classic](#)

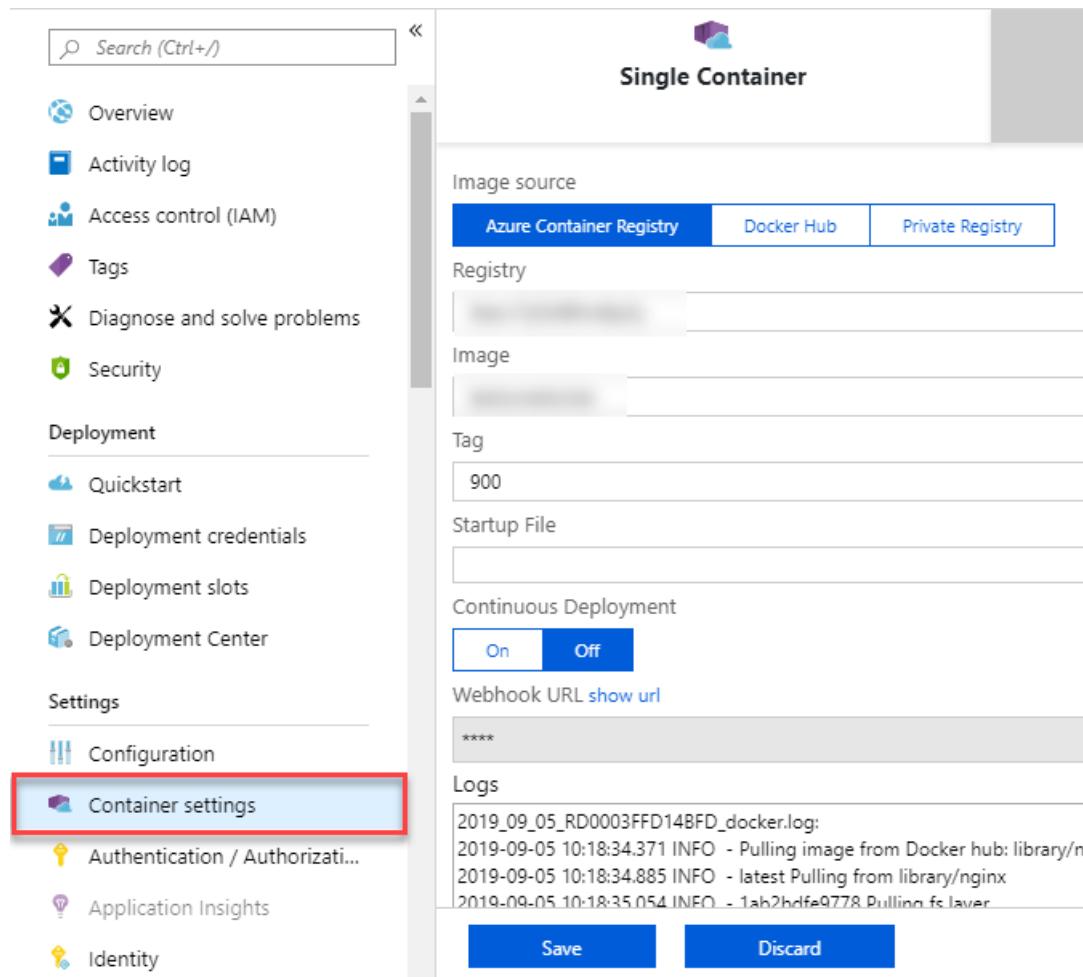
You must supply an Azure service connection to the `AzureWebAppContainer` task. Add the following YAML snippet to your existing `azure-pipelines.yaml` file. Make sure you add the service connection details in the variables section as shown below-

```
variables:  
    ## Add this under variables section in the pipeline  
    azureSubscription: <Name of the Azure subscription>  
    appName: <Name of the Web App>  
    containerRegistry: <Name of the Azure container registry>  
  
    ## Add the below snippet at the end of your pipeline  
    - task: AzureWebAppContainer@1  
        displayName: 'Azure Web App on Container Deploy'  
        inputs:  
            azureSubscription: $(azureSubscription)  
            appName: $(appName)  
            containers: $(containerRegistry)}/${{imageRepository}}:${{tag}}
```

YAML pipelines aren't available on TFS.

Configure registry credentials in web app

App Service needs information about your registry and image to pull the private image. In the [Azure portal](#), go to **Container settings** from the web app and update the **Image source**, **Registry** and save.



The screenshot shows the Azure portal's Container settings page for a "Single Container". The left sidebar lists various settings like Overview, Activity log, and Deployment. The "Container settings" link is highlighted with a red box. The main pane shows the "Image source" configuration, with the "Azure Container Registry" tab selected. It displays fields for Registry, Image, and Tag, along with other deployment-related options like Continuous Deployment and Webhook URL. The logs section at the bottom shows a recent log entry from a Docker container.

Deploy with Azure Web App for Container

- [YAML](#)
- [Classic](#)

The simplest way to deploy to an Azure Web App Container is to use the [Azure Web App for Containers](#) task.

To deploy to an Azure Web App container, add the following snippet at the end of your `azure-pipelines.yml` file:

```
trigger:
- main

variables:
# Container registry service connection established during pipeline creation
imageRepository: <Name of your image repository>
containerRegistry: <Name of the Azure container registry>
dockerfilePath: '$(Build.SourcesDirectory)/Dockerfile'
tag: '$(Build.BuildId)'

# Agent VM image name
vmImageName: 'ubuntu-latest'

- task: AzureWebAppContainer@1 # Add this at the end of your file
inputs:
  azureSubscription: '<Azure service connection>'
  appName: '<Name of the container web app>'
  containers: $(containerRegistry)}/${imageRepository}:$(tag)
```

The snippet assumes that the build steps in your YAML file build and push the docker image to your Azure container registry. The **Azure Web App on Container** task will pull the appropriate docker image corresponding to the BuildId from the repository specified, and then deploys the image to the Linux App Service.

YAML pipelines aren't available on TFS.

Deploy to a slot

- [YAML](#)
- [Classic](#)

You can configure the Azure Web App container to have multiple slots. Slots allow you to safely deploy your app and test it before making it available to your customers.

The following YAML snippet shows how to deploy to a staging slot, and then swap to a production slot:

```
- task: AzureWebAppContainer@1
inputs:
  azureSubscription: '<Azure service connection>'
  appName: '<Name of the web app>'
  containers: $(containerRegistry)}/${imageRepository}:$(tag)
  deployToSlotOrASE: true
  resourceGroupName: '<Name of the resource group>'
  slotName: staging

- task: AzureAppServiceManage@0
inputs:
  azureSubscription: '<Azure service connection>'
  WebAppName: '<name of web app>'
  ResourceGroupName: '<name of resource group>'
  SourceSlot: staging
  SwapWithProduction: true
```

YAML pipelines aren't available on TFS.

Deploy an Azure Web App

11/2/2020 • 11 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

You can automatically deploy your web app to an Azure App Service web app after every successful build.

NOTE

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

Build your app

- [YAML](#)
- [Classic](#)

Follow the guidance in [Create your first pipeline](#) and use the .NET Core sample offered there before you use this topic. When you're done, you'll have a YAML pipeline to build, test, and publish the source as an artifact.

We aren't yet advising new users to use YAML pipelines to deploy from Azure DevOps Server 2019. If you're an experienced pipeline user and already have a YAML pipeline to build your .NET Core app, then you might find the examples below useful.

YAML pipelines aren't available on TFS.

Now you're ready to read through the rest of this topic to learn some of the more common changes that people make to customize an Azure Web App deployment.

Azure Web App Deploy task

- [YAML](#)
- [Classic](#)

The simplest way to deploy to an Azure Web App is to use the [Azure Web App Deploy](#) (`AzureWebApp`) 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: AzureWebApp@1
  inputs:
    azureSubscription: '<Azure service connection>'
    appName: '<Name of web app>'
    package: $(System.DefaultWorkingDirectory)/**/*.zip
```

- **azureSubscription**: your Azure subscription.
- **appName**: the name of your existing app service.
- **package**: the file path to the package or a folder containing your app service contents. Wildcards are supported.

The snippet assumes that the build steps in your YAML file produce the zip archive in the `$(System.DefaultWorkingDirectory)` 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) to a Linux Webapp:

```
- task: AzureWebApp@1
  inputs:
    azureSubscription: '<Azure service connection>'
    appType: webAppLinux
    appName: '<Name of web app>'
    package: '$(System.DefaultWorkingDirectory)/**/*.war'
```

- **azureSubscription**: your Azure subscription.
- **appType**: your Web App type.
- **appName**: the name of your existing app service.
- **package**: the file path to the package or a folder containing your app service contents. Wildcards are supported.

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 `$(System.DefaultWorkingDirectory)` instead, change the last line in the snippet to `$(System.DefaultWorkingDirectory)/**/*.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. This snippet also generates a Web.config file during deployment if the application does not have one and starts the iisnode handler on the Azure Web App:

```
- task: AzureWebApp@1
  inputs:
    azureSubscription: '<Azure service connections>'
    appName: '<Name of web app>'
    package: '$(System.DefaultWorkingDirectory)'
    customWebConfig: '-Handler iisnode -NodeStartFile server.js -appType node'
```

- **azureSubscription**: your Azure subscription.
- **appName**: the name of your existing app service.
- **package**: the file path to the package or a folder containing your app service contents. Wildcards are supported.
- **customWebConfig**: generate web.config parameters for Python, Node.js, Go and Java apps. A standard `web.config` file will be generated and deployed to Azure App Service if the application does not have one.

For information on Azure service connections, see the [following section](#).

YAML pipelines aren't available on TFS.

Azure service connection

All the built-in Azure tasks require an Azure service connection as an input. The Azure service connection stores the credentials to connect from Azure Pipelines or TFS to Azure.

- [YAML](#)
- [Classic](#)

You must supply an Azure service connection to the `AzureWebApp` task. The Azure service connection stores the credentials to connect from Azure Pipelines to Azure. See [Create an Azure service connection](#).

YAML pipelines aren't available on TFS.

Deploy to a virtual application

- [YAML](#)
- [Classic](#)

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 `VirtualApplication` property of the `AzureRmWebAppDeployment` task:

```
- task: AzureRmWebAppDeployment@4
  inputs:
    VirtualApplication: '<name of virtual application>'
```

- **VirtualApplication**: the name of the Virtual Application that has been configured in the Azure portal. See [Configure an App Service app in the Azure portal](#) for more details.

YAML pipelines aren't available on TFS.

Deploy to a slot

- [YAML](#)
- [Classic](#)

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: AzureWebApp@1
  inputs:
    azureSubscription: '<Azure service connection>'
    appName: '<name of web app>'
    slotName: staging

- task: AzureAppServiceManage@0
  inputs:
    azureSubscription: '<Azure service connection>'
    WebAppName: '<name of web app>'
    ResourceGroupName: '<name of resource group>'
    SourceSlot: staging
```

YAML pipelines aren't available on TFS.

Deploy to multiple web apps

- [YAML](#)
- [Classic](#)

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:
    # publish an artifact called drop
    - task: PublishBuildArtifacts@1
      inputs:
        artifactName: drop

    # deploy to Azure Web App staging
    - task: AzureWebApp@1
      inputs:
        azureSubscription: '<test stage Azure service connection>'
        appName: '<name of test stage web app>'

- job: deploy
  pool:
    vmImage: 'ubuntu-16.04'
  dependsOn: buildandtest
  condition: succeeded()
  steps:
    # download the artifact drop from the previous job
    - task: DownloadBuildArtifacts@0
      inputs:
        artifactName: drop

    # deploy to Azure Web App production
    - task: AzureWebApp@1
      inputs:
        azureSubscription: '<prod Azure service connection>'
        appName: '<name of prod web app>'
```

YAML pipelines aren't available on TFS.

Configuration changes

For most language stacks, [app settings](#) and [connection strings](#) can be set as environment variables at runtime. App settings can also be resolved from Key Vault using [Key Vault references](#).

For ASP.NET and ASP.NET Core developers, setting app settings in App Service are like setting them in `Web.config`. You might want to apply a specific configuration for your web app target before deploying to it. This is 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.

Azure App Service Deploy task allows users to modify configuration settings in configuration files (`*.config` files) inside web packages and XML parameters files (`parameters.xml`), based on the stage name specified.

NOTE

File transforms and variable substitution are also supported by the separate [File Transform task](#) for use in Azure Pipelines. You can use the File Transform task to apply file transformations and variable substitutions on any configuration and parameters files.

- [YAML](#)
- [Classic](#)

The following snippet shows an example of variable substitution:

```
jobs:  
- job: test  
  variables:  
    connectionString: <test-stage connection string>  
  steps:  
- task: AzureRmWebAppDeployment@4  
  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@4  
  inputs:  
    azureSubscription: '<Prod stage Azure service connection>'  
    WebAppName: '<name of prod stage web app>'  
    enableXmlVariableSubstitution: true
```

YAML pipelines aren't available on TFS.

Deploying conditionally

You can choose to deploy only certain builds to your Azure Web App.

- [YAML](#)
- [Classic](#)

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 main branch:

```
- task: AzureWebApp@1  
  condition: and(succeeded(), eq(variables['Build.SourceBranch'], 'refs/heads/main'))  
  inputs:  
    azureSubscription: '<Azure service connection>'  
    appName: '<name of web app>'
```

To learn more about conditions, see [Specify conditions](#).

YAML pipelines aren't 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 Web App 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](#).

One thing worth checking before deploying is the Azure App Service access restrictions list. This list can include IP addresses or Azure Virtual Network subnets. When there are one or more entries, there is then an implicit "deny all" that exists at the end of the list. To modify the access restriction rules to your app, see [Adding and editing access restriction rules in Azure portal](#). You can also [modify/restrict access to your source control management \(scm\) site](#).

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.

You can use the [File Transform task](#) to apply file transformations and variable substitutions on any configuration and parameters files.

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 Azure App Service using Visual Studio Code

3/15/2021 • 6 minutes to read • [Edit Online](#)

This tutorial walks you through setting up a CI/CD pipeline for deploying Node.js application to Azure App Service using [Deploy to Azure](#) extension.

Prerequisites

- An Azure account. If you don't have one, you can [create for free](#).
- You need [Visual Studio Code](#) installed along with the [Node.js](#) and [npm](#) the [Node.js package manager](#) and the below extensions:
- You need [Azure Account extension](#) and [Deploy to Azure extension](#)
- A GitHub account, where you can create a repository. If you don't have one, you can [create one for free](#).

IMPORTANT

Ensure that you have all the prerequisites installed and configured. In VS Code, you should see your Azure email address in the Status Bar.

Create Node.js application

Create a Node.js application that can be deployed to the Cloud. This tutorial uses an application generator to quickly scaffold the application from a terminal.

TIP

If you have already completed the [Node.js](#) tutorial, you can skip ahead to [Setup CI/CD Pipeline](#).

Install the Express Generator

[Express](#) is a popular framework for building and running Node.js applications. You can scaffold (create) a new Express application using the [Express Generator](#) tool. The Express Generator is shipped as an npm module and installed by using the npm command-line tool `npm`.

TIP

To test that you've got `npm` correctly installed on your computer, type `npm --help` from a terminal and you should see the usage documentation.

Install the Express Generator by running the following from a terminal:

```
npm install -g express-generator
```

The `-g` switch installs the Express Generator globally on your machine so you can run it from anywhere.

Scaffold a new application

We can now scaffold a new Express application called `myExpressApp` by running:

```
express myExpressApp --view pug --git
```

This creates a new folder called `myExpressApp` with the contents of your application. The `--view pug` parameters tell the generator to use the `pug` template engine (formerly known as `jade`).

To install all of the application's dependencies (again shipped as `npm` modules), go to the new folder and execute `npm install`:

```
cd myExpressApp  
npm install
```

At this point, we should test that our application runs. The generated Express application has a `package.json` file, that includes a start script to run `node ./bin/www`. This will start the Node.js application running.

Run the application

1. From a terminal in the Express application folder, run:

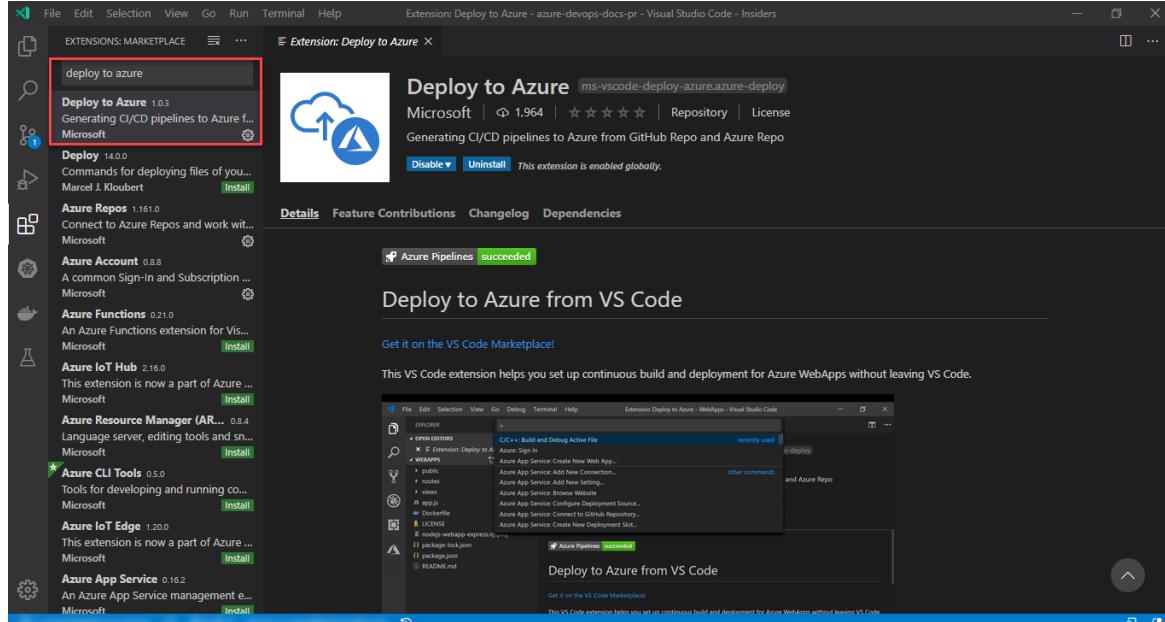
```
npm start
```

The Node.js web server will start and you can browse to `http://localhost:3000` to see the running application.

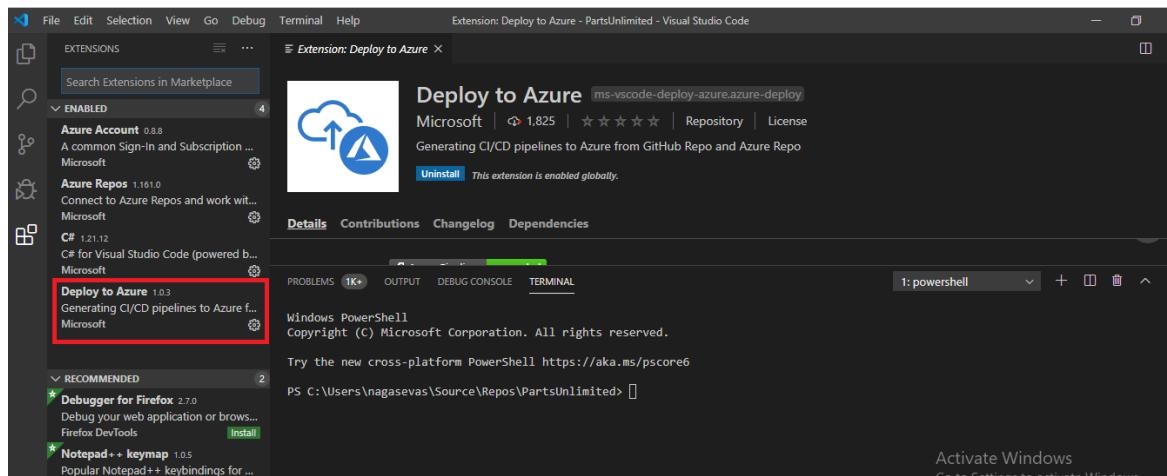
2. Follow [this link](#) to push this project to GitHub using the command line.
3. Open your application folder in VS Code and get ready to deploy to Azure.

Install the extension

1. Bring up the **Extensions** view by clicking on the Extensions icon in the Activity Bar on the side of VS Code or the **View: Extensions** command (`Ctrl+Shift+X`).
2. Search for **Deploy to Azure** extension and install.



3. After the installation is complete, the extension will be located in enabled extension space.



Setup CI/CD Pipeline

Now you can deploy to Azure App Services, Azure Function App and AKS using VS code. This VS Code extension helps you set up continuous build and deployment for Azure App Services without leaving VS Code.

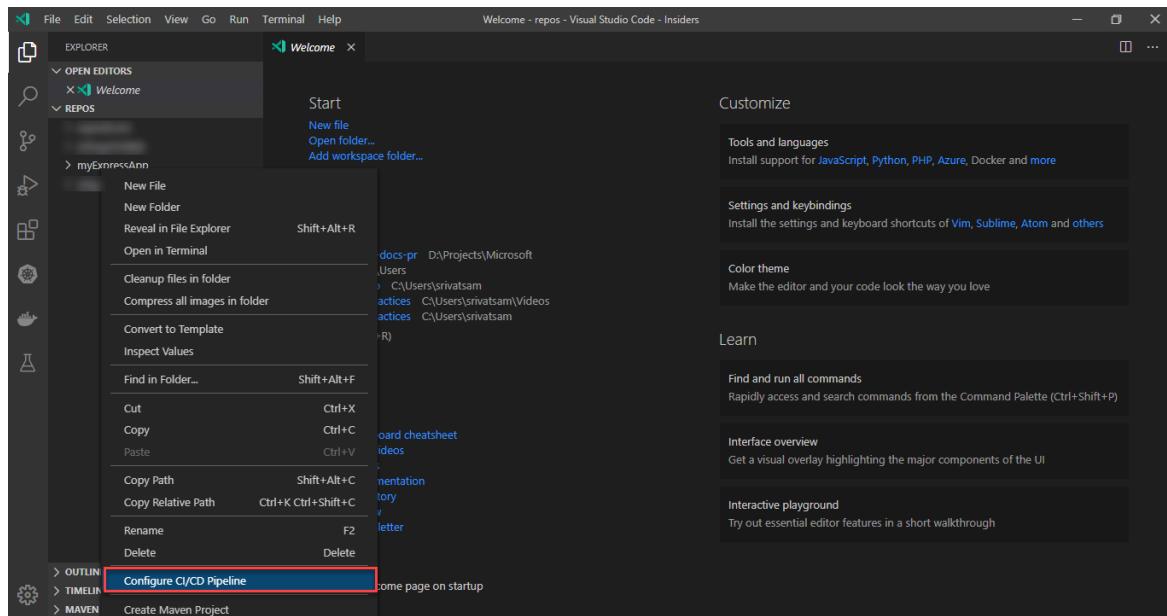
To use this service, you need to install the extension on VS Code. You can browse and install extensions from within VS Code.

Combination of workflows

We support GitHub Actions and Azure Pipelines for GitHub & Azure Repos correspondingly. We also allow you to create Azure Pipelines if you still manage the code in GitHub.

GitHub + GitHub Actions

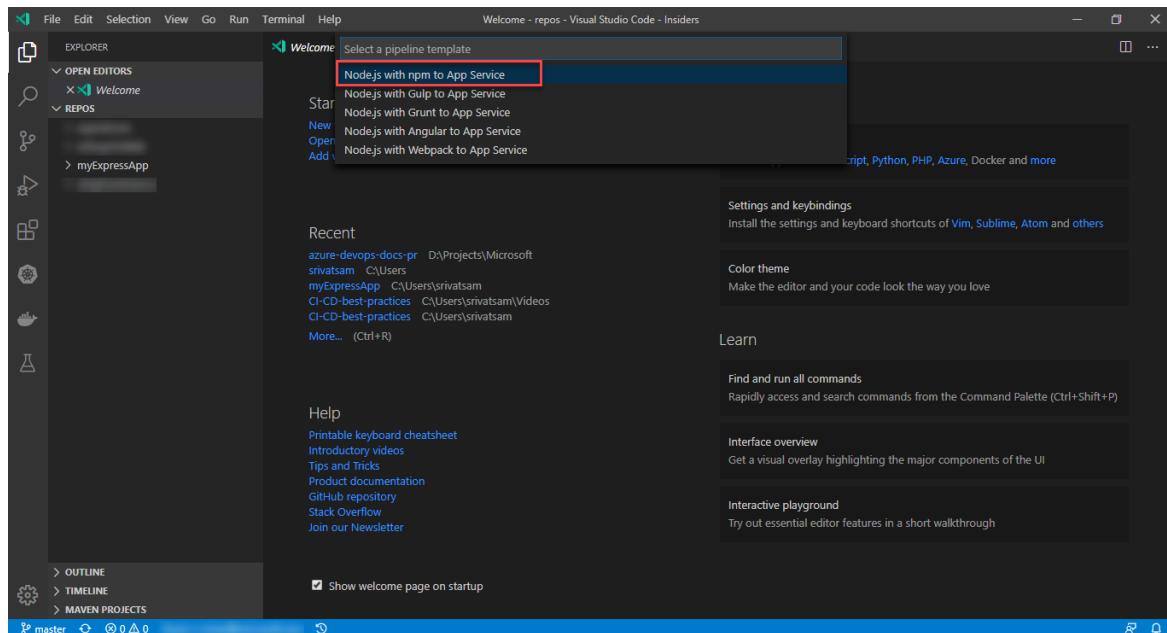
1. To set up a pipeline, choose `Deploy to Azure: Configure CI/CD Pipeline` from the command palette (Ctrl/Cmd + Shift + P) or right-click on the file explorer.



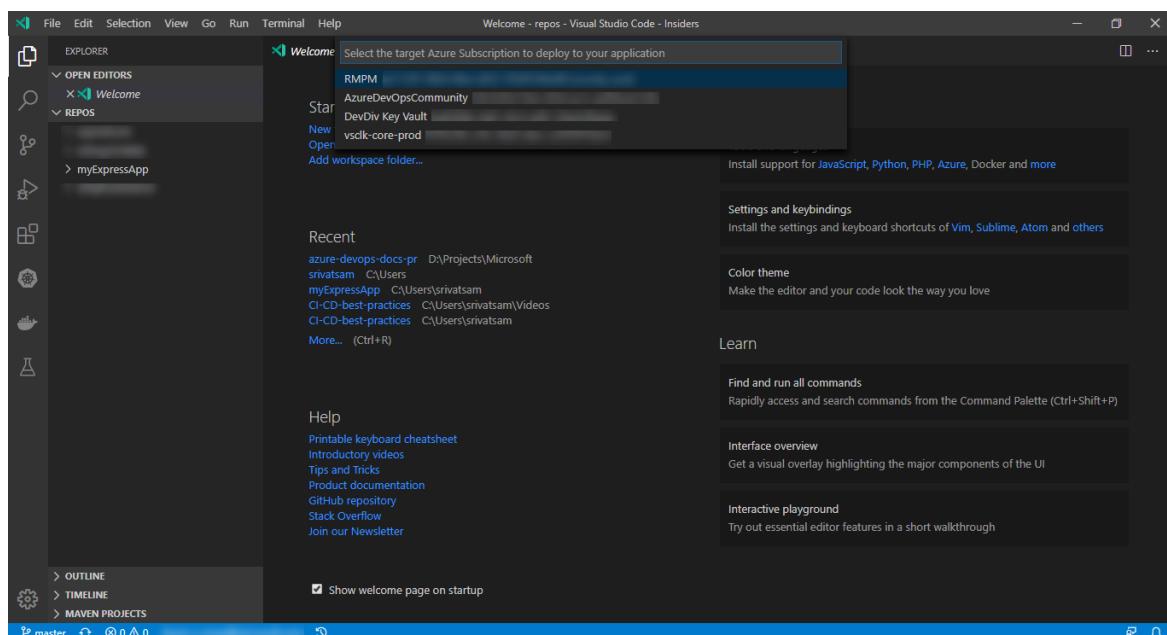
NOTE

If the code is not opened in the workspace, it will ask for folder location. Similarly, if the code in the workspace has more than one folder, it will ask for folder.

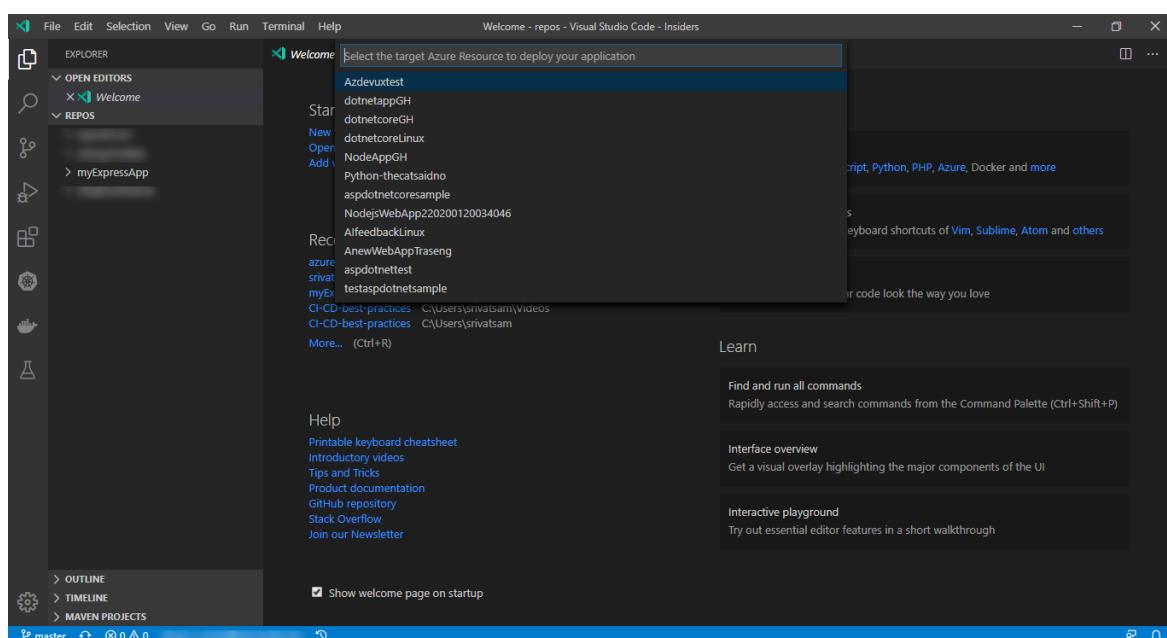
2. Select a pipeline template you want to create from the list. Since we're targeting `Node.js`, select `Node.js with npm to App Service.`



3. Select the target Azure Subscription to deploy your application.



4. Select the target Azure resource to deploy your application.



5. Enter GitHub personal access token (PAT), required to populate secrets that are used in GitHub workflows.

Set the scope to `repo` and `admin:repo_hook`.

Select scopes

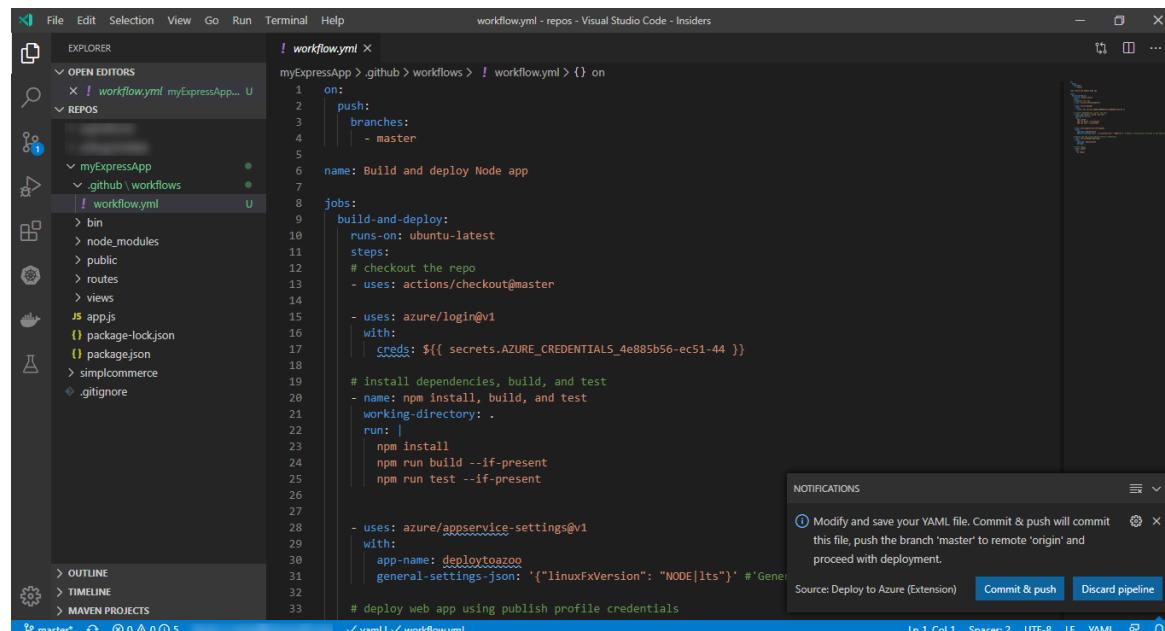
Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> <code>repo:status</code>	Access commit status
<input checked="" type="checkbox"/> <code>repo_deployment</code>	Access deployment status
<input checked="" type="checkbox"/> <code>public_repo</code>	Access public repositories
<input checked="" type="checkbox"/> <code>repo:invite</code>	Access repository invitations
<input type="checkbox"/> <code>write:packages</code>	Upload & delete packages in github package registry
<input type="checkbox"/> <code>read:packages</code>	Download packages from github package registry
<input type="checkbox"/> <code>admin:org</code>	Full control of orgs and teams, read and write org projects
<input type="checkbox"/> <code>write:org</code>	Read and write org and team membership, read and write org projects
<input type="checkbox"/> <code>read:org</code>	Read org and team membership, read org projects
<input type="checkbox"/> <code>admin:public_key</code>	Full control of user public keys
<input type="checkbox"/> <code>write:public_key</code>	Write user public keys
<input type="checkbox"/> <code>read:public_key</code>	Read user public keys
<input checked="" type="checkbox"/> <code>admin:repo_hook</code>	Full control of repository hooks
<input checked="" type="checkbox"/> <code>write:repo_hook</code>	Write repository hooks
<input checked="" type="checkbox"/> <code>read:repo_hook</code>	Read repository hooks

TIP

If the code is in Azure Repos, you need different permissions.

6. The configuration of GitHub workflow or Azure Pipeline happens based on the extension setting. The guided workflow will generate a starter YAML file defining the build and deploy process. **Commit & push** the YAML file to proceed with the deployment.



```
on:
  push:
    branches:
      - master
  name: Build and deploy Node app

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
      # checkout the repo
      - uses: actions/checkout@master

      - uses: azure/login@v1
        with:
          creds: ${{ secrets.AZURE_CREDENTIALS_4e885b56-ec51-44 }}

      # install dependencies, build, and test
      - name: npm install, build, and test
        working-directory:
          run:
            npm install
            npm run build --if-present
            npm run test --if-present
      - uses: azure/appservice-settings@v1
        with:
          app-name: deploytoazoo
          general-settings-json: '{"linuxFxVersion": "NODE|lts"}' # General settings for the app
      # deploy web app using publish profile credentials
```

TIP

You can customize the pipeline using all the features offered by [Azure Pipelines](#) and [GitHub Actions](#).

7. Navigate to your GitHub repo to see the actions in progress.

The screenshot shows a GitHub repository page for 'iamsrivatsa / deploytoazure'. A workflow named 'Build and deploy Node app' has been triggered. The logs for this workflow show a series of steps: 'Set up job' (8s), 'Run actions/checkout@master' (1s), 'Run azure/login@v1' (12s), 'npm install, build, and test' (9s), 'Run azure/appservice-settings@v1' (5s), 'Run azure/webapps-deploy@v1' (26s), 'logout' (5s), and 'Post actions/checkout@master'. All steps are marked with a green checkmark, indicating success.

8. Navigate to your site running in Azure using the Web App URL `http://{web_app_name}.azurewebsites.net`, and verify its contents.

GitHub + Azure Pipelines

IMPORTANT

To setup CI/CD in Azure Pipelines for Github Repository, you need to enable `Use Azure Pipelines for GitHub` in the extension.

To open your user and workspace settings, use the following VS Code menu command:

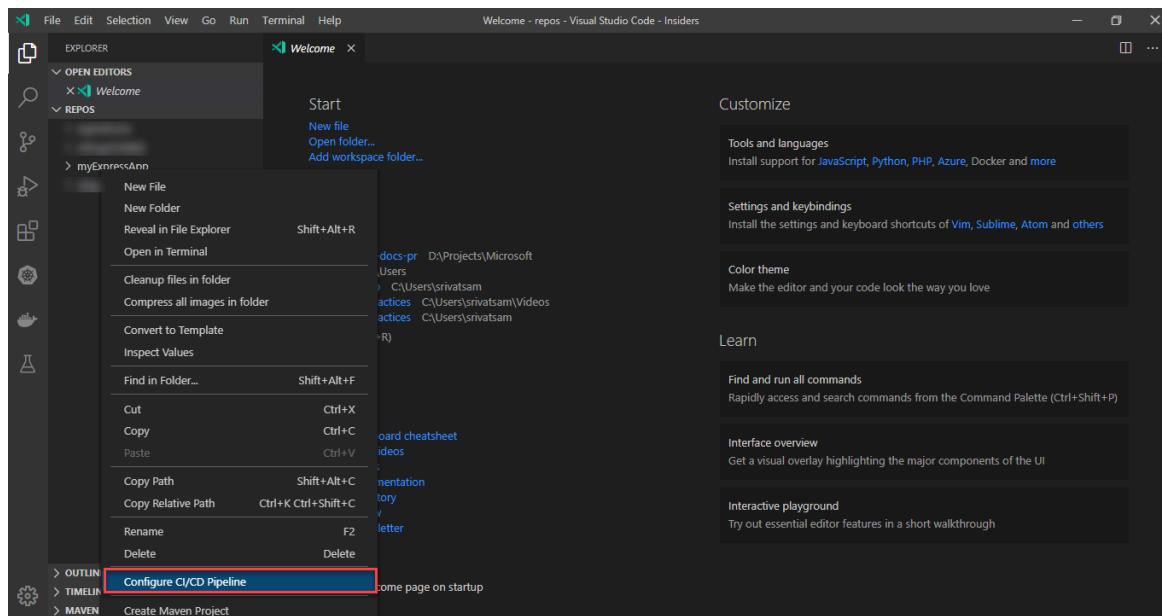
- On Windows/Linux - **File > Preferences > Settings**
- On macOS - **Code > Preferences > Settings**

You can also open the Settings editor from the Command Palette (`Ctrl+Shift+P`) with Preferences: Open Settings or use the keyboard shortcut (`Ctrl+,`).

When you open the settings editor, you can search and discover settings you are looking for. Search for the name `deployToAzure.UseAzurePipelinesForGitHub` and enable as shown below.

The screenshot shows the VS Code Settings editor. The search bar at the top contains the text `deployToAzure.UseAzurePipelinesForGitHub`. Below the search bar, there are two tabs: 'User' and 'Workspace'. Under 'User', there is one entry: 'Deploy To Azure: Use Azure Pipelines For Github (Also modified in: User)'. This entry has a checked checkbox next to it, and the entire row is highlighted with a red box. The status bar at the bottom indicates the current branch is 'master'.

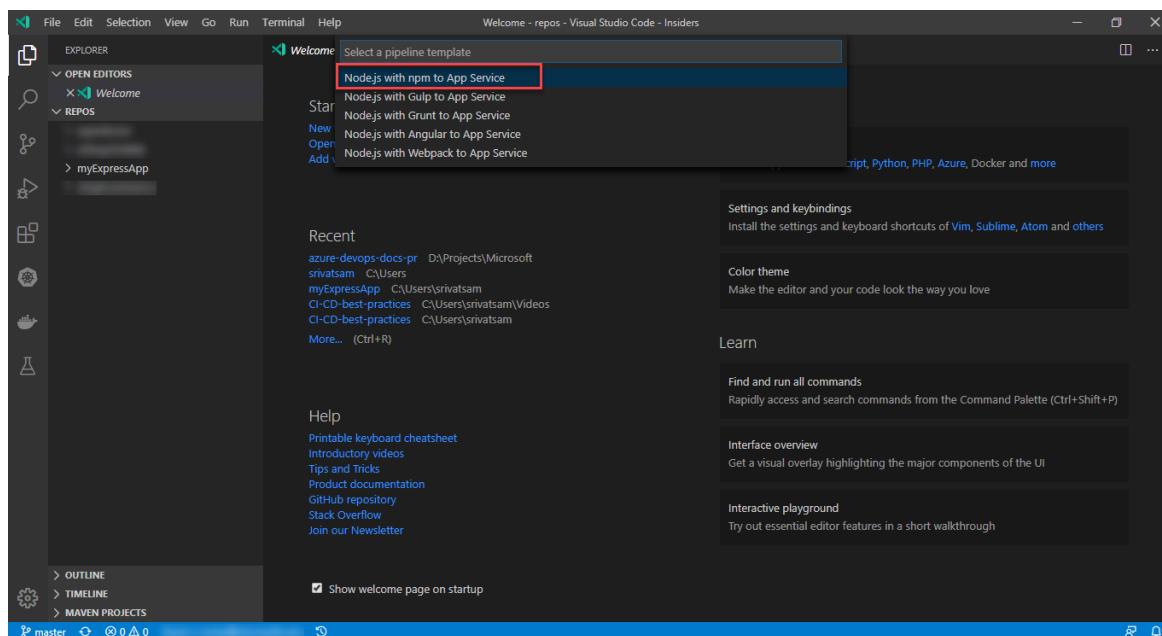
- To set up a pipeline, choose `Deploy to Azure: Configure CI/CD Pipeline` from the command palette (Ctrl/Cmd + Shift + P) or right-click on the file explorer.



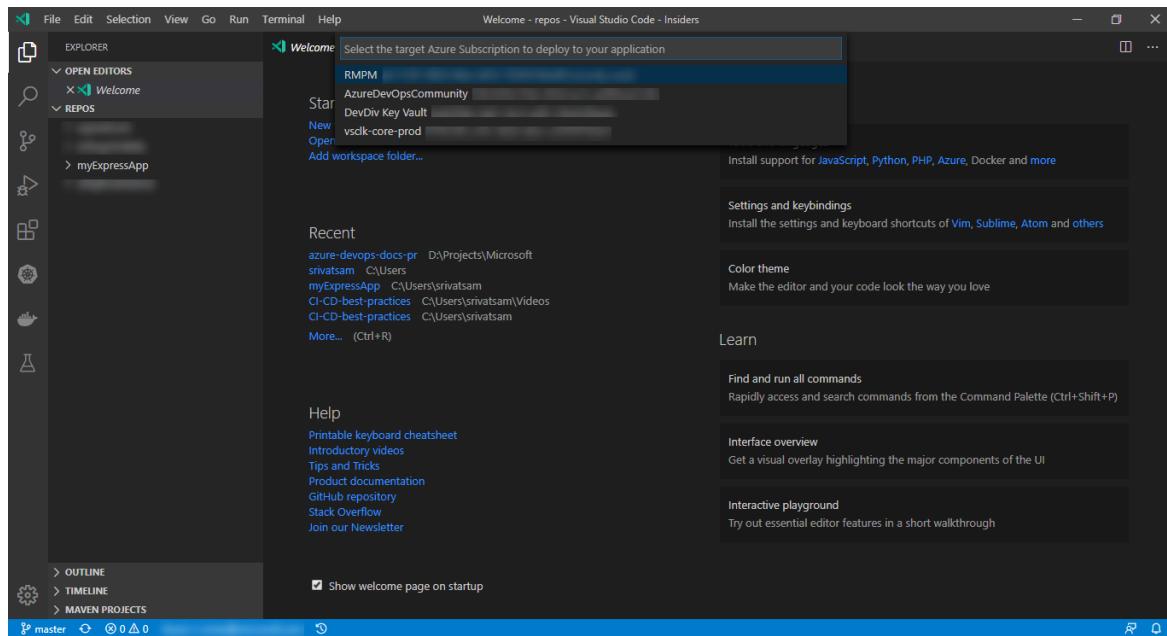
NOTE

If the code is not opened in the workspace, it will ask for folder location. Similarly, if the code in the workspace has more than one folder, it will ask for folder.

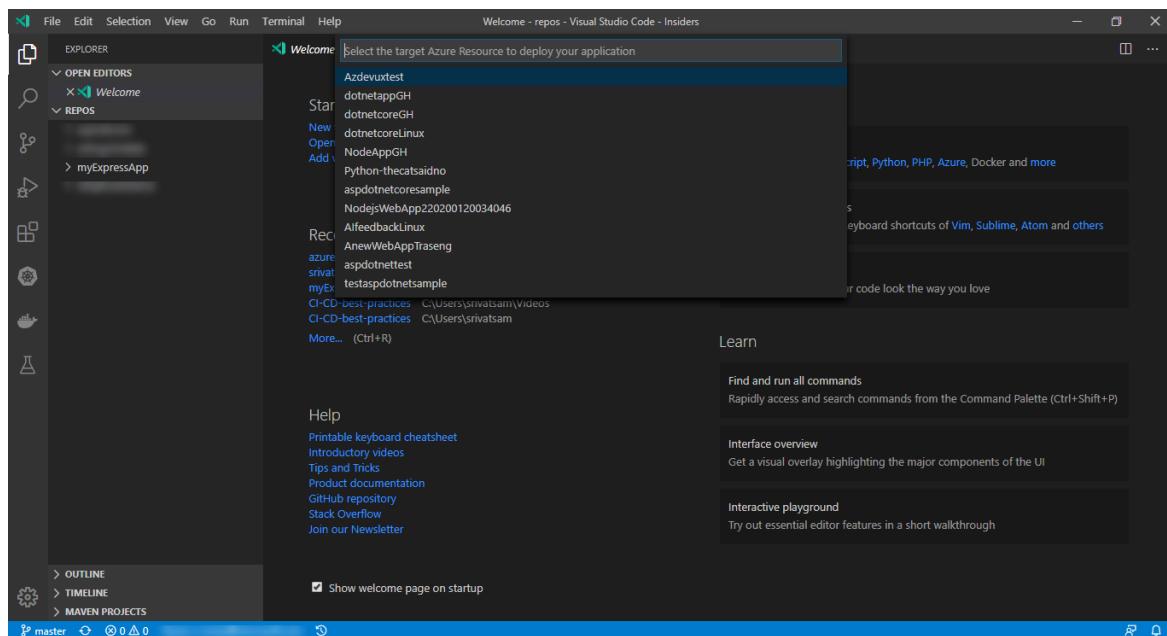
- Select a pipeline template you want to create from the list. Since we're targeting `Node.js`, select `Node.js with npm to App Service`.



- Select the target Azure Subscription to deploy your application.

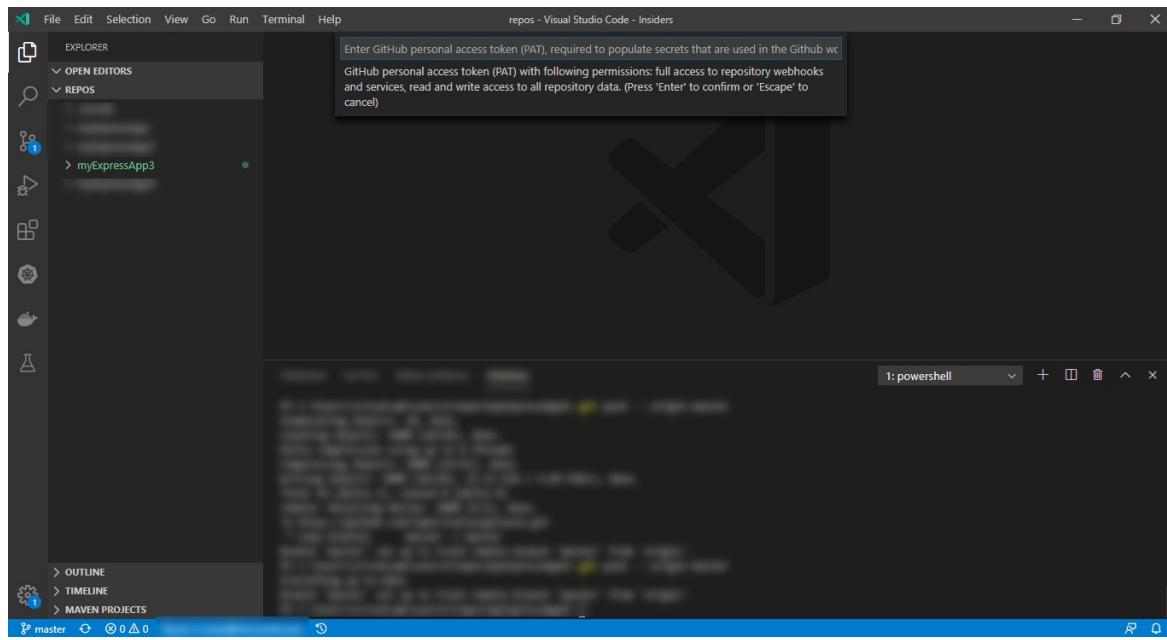


4. Select the target Azure resource to deploy your application.



5. Enter GitHub personal access token (PAT), required to populate secrets that are used in GitHub workflows.

Set the scope to `repo` and `admin:repo_hook`.

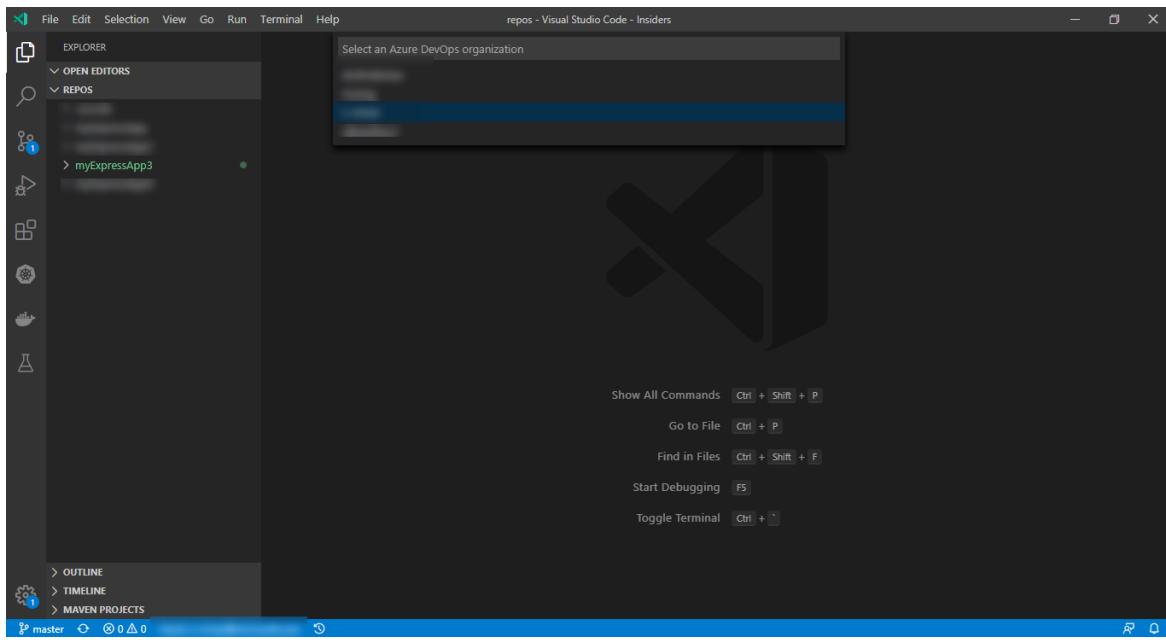


Select scopes

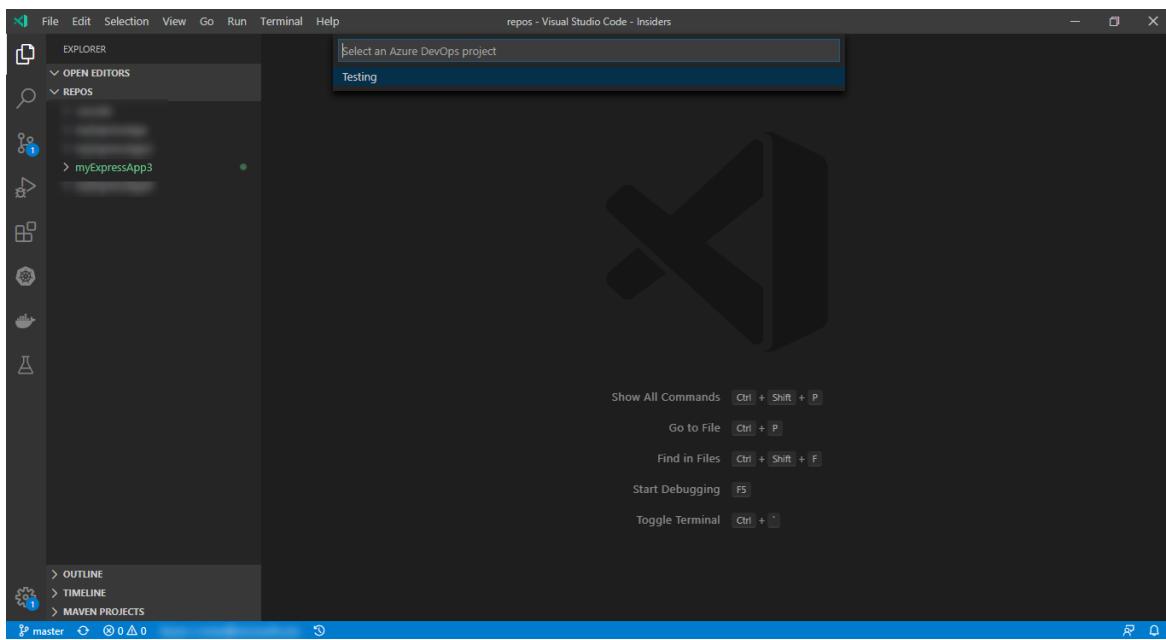
Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input type="checkbox"/> write:packages	Upload & delete packages in github package registry
<input type="checkbox"/> read:packages	Download packages from github package registry
<input type="checkbox"/> admin:org	Full control of orgs and teams, read and write org projects
<input type="checkbox"/> write:org	Read and write org and team membership, read and write org projects
<input type="checkbox"/> read:org	Read org and team membership, read org projects
<input type="checkbox"/> admin:public_key	Full control of user public keys
<input type="checkbox"/> write:public_key	Write user public keys
<input type="checkbox"/> read:public_key	Read user public keys
<input checked="" type="checkbox"/> admin:repo_hook	Full control of repository hooks
<input checked="" type="checkbox"/> write:repo_hook	Write repository hooks
<input checked="" type="checkbox"/> read:repo_hook	Read repository hooks

6. Select an Azure DevOps organization.



7. Select an Azure DevOps project.



8. The configuration of GitHub workflow or Azure Pipeline happens based on the extension setting. The guided workflow will generate a starter YAML file defining the build and deploy process. **Commit & push** the YAML file to proceed with the deployment.

```

myExpressApp4 > ! azure-pipelines.yml
1 # Node.js App on Linux Web App
2 # Build a Node.js app and deploy it to Azure as a Linux web app.
3 # Add steps that analyze code, save build artifacts, deploy, and more:
4 # https://docs.microsoft.com/azure/devops/pipelines/languages/javascript
5
6 trigger:
7 - master
8
9 variables:
10 # Azure Resource Manager connection
11 azureSubscription: '...'
12
13 # Web app name
14 webAppName: 'node111'
15
16 # Working Directory
17 workingDirectory: '.'
18
19 stages:
20 - stage: Build
21   displayName: Build stage
22   jobs:
23     - job: BuildJob
24       displayName: Build
25       pool:
26         vmImage: 'ubuntu-latest'
27
28       steps:
29         - task: NodeTool@0
30           inputs:
31             versionSpec: '10.x'
32             displayName: 'Install Node.js'
33

```

TIP

You can customize the pipeline using all the features offered by [Azure Pipelines](#) and [GitHub Actions](#).

9. Navigate to your Azure DevOps project to see the pipeline in progress.

#20200318.1 Added Azure Pipelines YAML definition.

Summary WhiteSource Bolt Build Report

Manually run by [redacted]

Repository and version

Time started and elapsed

Related

Tests and coverage

Stages Jobs

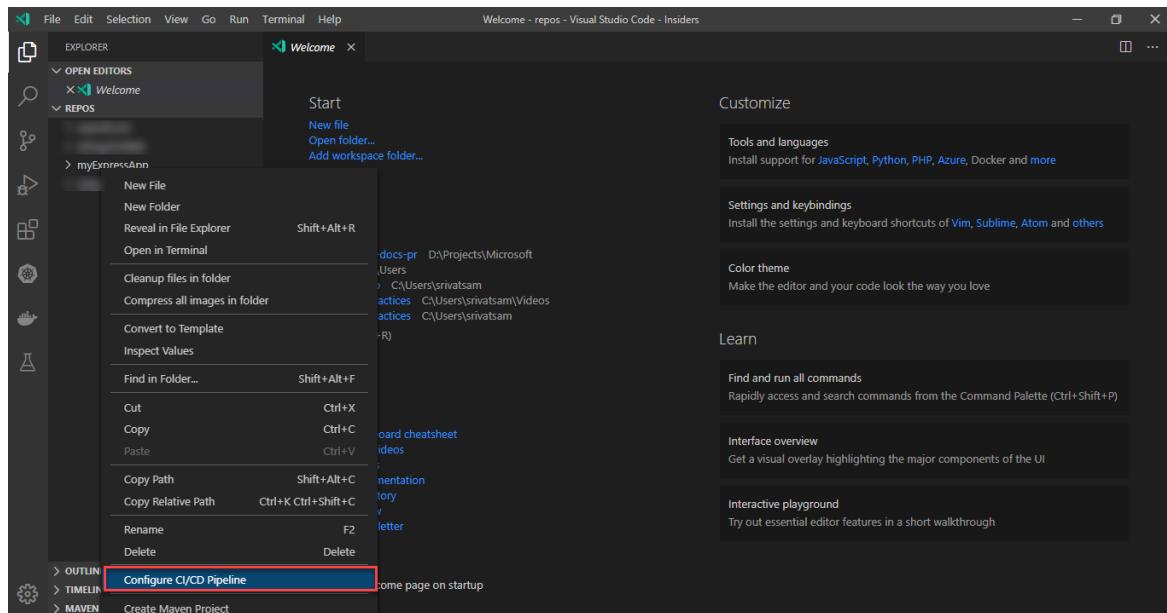
Build stage Deploy stage

Not started Not started

10. Navigate to your site running in Azure using the Web App URL `http://{web_app_name}.azurewebsites.net`, and verify its contents.

Azure Repos + Azure Pipelines

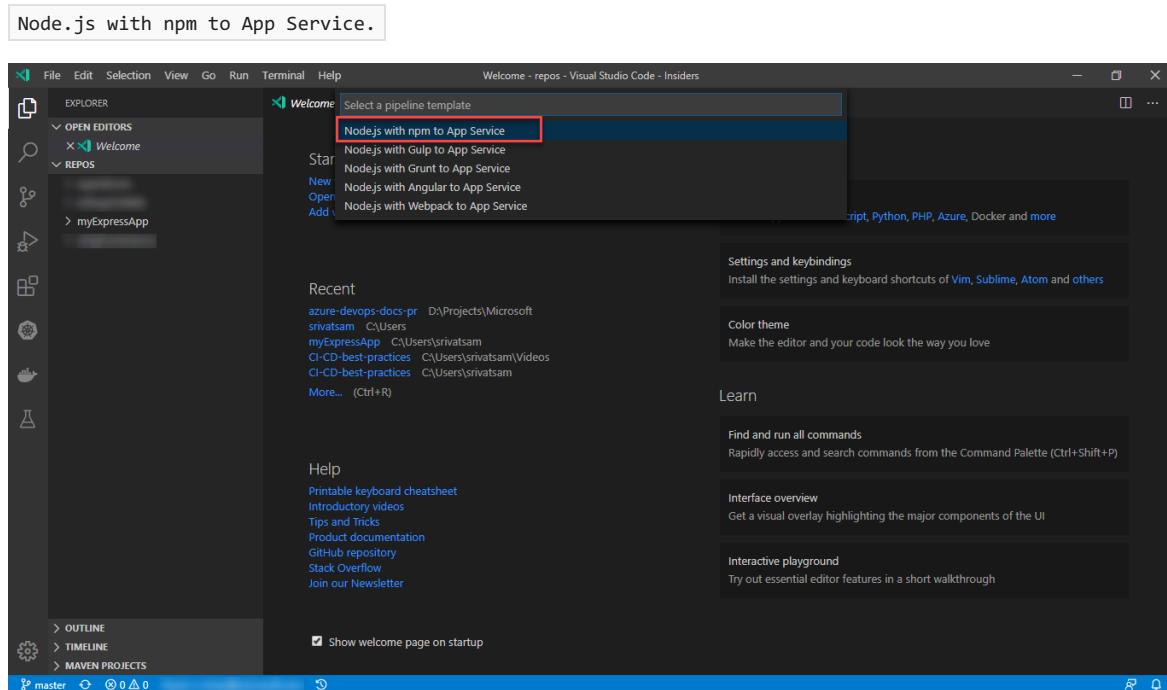
1. To set up a pipeline, choose `Deploy to Azure: Configure CI/CD Pipeline` from the command palette (Ctrl/Cmd + Shift + P) or right-click on the file explorer.



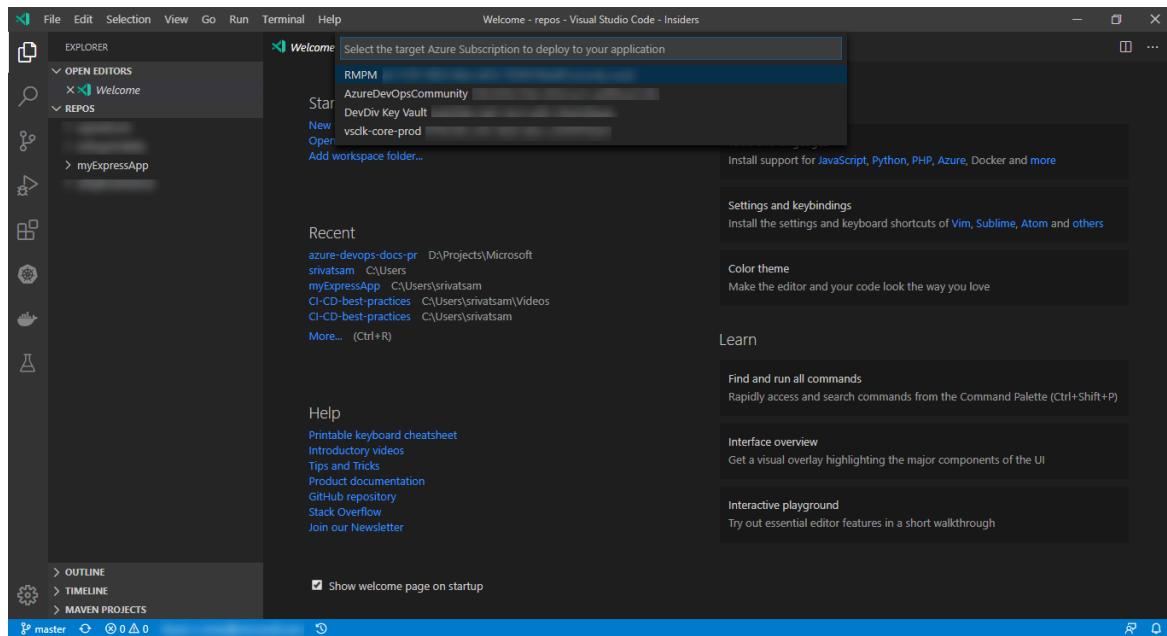
NOTE

If the code is not opened in the workspace, it will ask for folder location. Similarly, if the code in the workspace has more than one folder, it will ask for folder.

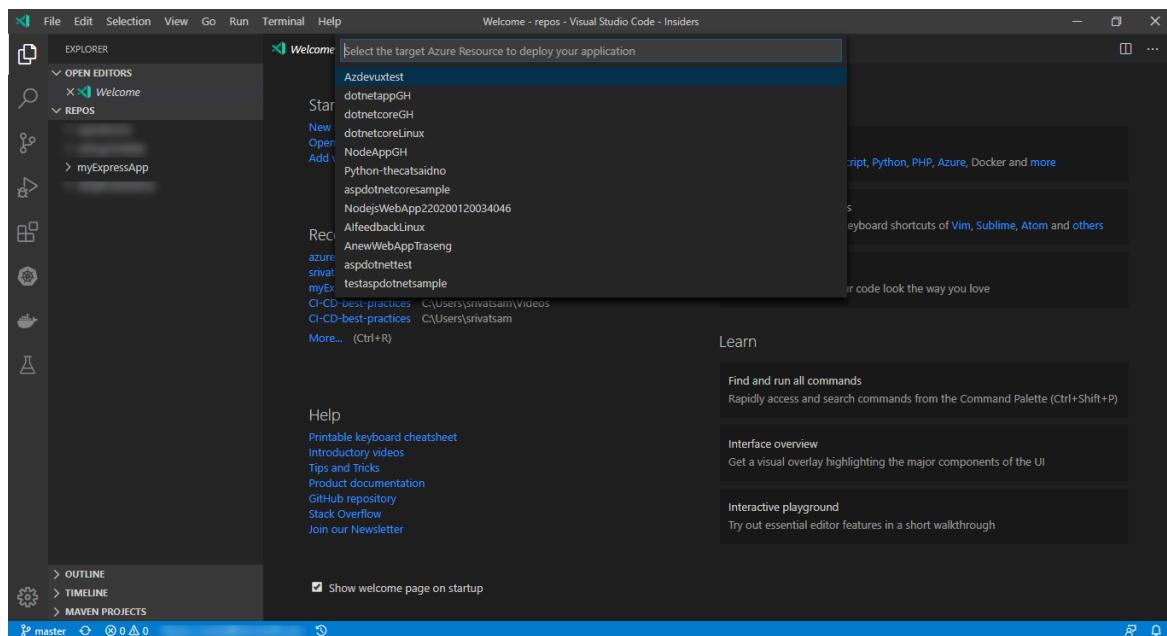
2. Select a pipeline template you want to create from the list. Since we're targeting `Node.js`, select



3. Select the target Azure Subscription to deploy your application.



4. Select the target Azure resource to deploy your application.



5. The configuration of GitHub workflow or Azure Pipeline happens based on the extension setting. The guided workflow will generate a starter YAML file defining the build and deploy process. **Commit & push** the YAML file to proceed with the deployment.

```

myExpressApp5 > ! azure-pipelines.yml ...
1 # Node.js App on Linux Web App
2 # Build a Node.js app and deploy it to Azure as a Linux web app.
3 # Add steps that analyze code, save build artifacts, deploy, and more:
4 # https://docs.microsoft.com/azure/devops/pipelines/languages/javascript
5
6 trigger:
7 - master
8
9 variables:
10 # Azure Resource Manager connection
11 azureSubscription: [REDACTED]
12
13 # Web app name
14 webAppName: 'spinode'
15
16 # Working Directory
17 workingDirectory: '.'
18
19 stages:
20 - stage: Build
21   displayName: Build stage
22   jobs:
23     - job: BuildJob
24       displayName: Build
25       pool:
26         vmImage: 'ubuntu-latest'
27
28       steps:
29         - task: NodeTool@0
30           inputs:
31             versionSpec: '10.x'
32             displayName: 'Install Node.js'
33

```

Source: Deploy to Azure (Extension) Commit & push Discard pipeline

TIP

You can customize the pipeline using all the features offered by [Azure Pipelines](#) and [GitHub Actions](#).

6. Navigate to your Azure DevOps project to see the pipeline in progress.

#20200318.1 Added Azure Pipelines YAML definition.

Summary WhiteSource Bolt Build Report

Manually run by [REDACTED]

Repository and version
vscode
master 29ff617

Time started and elapsed
Just now

Related
0 work items
Get started

Tests and coverage
0 artifacts

Stages Jobs

Build stage Deploy stage

Not started Not started

7. Navigate to your site running in Azure using the Web App URL http://{web_app_name}.azurewebsites.net, and verify its contents.

Troubleshoot

Learn how to troubleshoot common issues.

Issue: Failed to deploy web package to App Service. Conflict (CODE: 409)

Resolution: Restart App Service, and then deploy the package to App Service again.

Next steps

Try the workflow with a Docker file in a repo.

Help and support

- See our [troubleshooting](#) page
- Get advice on [Stack Overflow](#), and feel free to post your questions, search for answers, or suggest a feature on our [Azure DevOps Developer Community](#). [Support](#) page.

Deploy a web app to Azure App Services

2/26/2020 • 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](#)

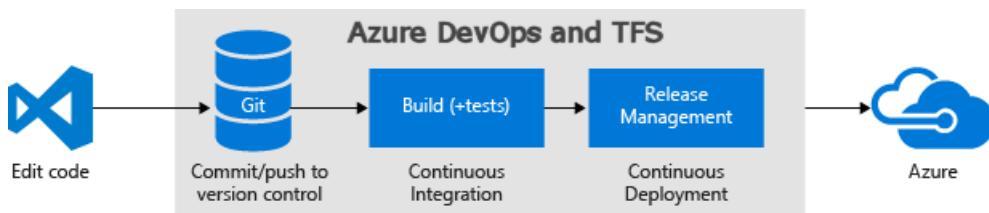
Deploy to Azure Web App for Containers

4/27/2021 • 3 minutes to read • [Edit Online](#)

Azure Pipelines

With Azure Web App for Containers, you can easily deploy and run container-based web apps on Windows and Linux. In this quickstart, we will use Azure Pipelines to build and deploy our sample application to Web App for Containers App service.

With Azure Pipelines, you can implement a CI/CD workflow to automatically generate build Artifacts and trigger deployment to specific environments.



Prerequisites

- Azure DevOps account. Create a free [Azure DevOps account](#) if you don't have one already.
- GitHub account. Create a free [GitHub account](#) if you don't have one already.
- Azure subscription. Create a free [Azure account](#) if you don't have one already.
- [Create an Azure container registry](#) if you don't have one already.

Get the code

[Fork](#) or [clone](#) the sample app to follow along with this tutorial.

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

```
https://github.com/spring-guides/gs-spring-boot-docker.git
```

Build and publish a Docker image to an Azure Container Registry

To complete this section successfully, you must have an [Azure Container Registry](#). Refer to the prerequisites section for details.

1. Sign in to your Azure DevOps organization and navigate to your project.
2. Select **Pipelines**, and then **New Pipeline**.
3. Select **GitHub** when prompted for the location of your source code, and then select your repository.
4. Select the **Docker: build and push an image to Azure Container Registry** pipeline template.

The screenshot shows the 'Configure your pipeline' step in the Azure DevOps interface. On the left, there's a sidebar with various project management and development options like Overview, Boards, Repos, Pipelines, and Artifacts. The 'Pipelines' section is currently selected. The main area is titled 'Configure your pipeline' and lists several pipeline tasks. One task, 'Docker' (Build and push an image to Azure Container Registry), is highlighted with a red box.

- Docker
- Deploy to Azure Kubernetes Service
- Deploy to Kubernetes - Review app with Azure DevSpaces
- Node.js
- Node.js Express Web App to Linux on Azure
- Node.js with Vue
- Node.js with webpack
- Node.js with React
- Node.js React Web App to Linux on Azure
- Node.js with Angular
- Starter pipeline
- Existing Azure Pipelines YAML file

5. Select your Azure Subscription, and then select **Continue**.
6. Select your **Container registry** from the drop-down menu, and then select **Validate and configure**.

This screenshot shows the configuration details for the 'Docker' task. It includes fields for 'Container registry' (set to 'sampleappcontinerregistry'), 'Image Name' (set to 'javascriptdocker'), and 'Dockerfile' (set to '\$(Build.SourcesDirectory)/app/Dockerfile'). At the bottom, there are 'Back' and 'Validate and configure' buttons, with 'Validate and configure' being highlighted by a red box.

Container registry
sampleappcontinerregistry

Image Name
javascriptdocker

Dockerfile
\$(Build.SourcesDirectory)/app/Dockerfile

Back **Validate and configure**

7. Review the pipeline YAML template, and then select **Save and run** to build and publish the Docker image to your Azure Container Registry.

```

trigger:
- master

resources:
- repo: self

variables:
  # Container registry service connection established during pipeline creation
  dockerRegistryServiceConnection: '{{ containerRegistryConnection.Id }}'
  imageRepository: 'javascriptdocker'
  containerRegistry: 'sampleappcontinerregistry.azurecr.io'
  dockerfilePath: '$(Build.SourcesDirectory)/app/Dockerfile'
  tag: '$(Build.BuildId)'

  # Agent VM image name
  vmImageName: 'ubuntu-latest'

stages:
- stage: Build
  displayName: Build and push stage
  jobs:
  - job: Build
    displayName: Build
    pool:
      vmImage: $(vmImageName)
    steps:
    - task: Docker@2
      displayName: Build and push an image to container registry
      inputs:
        command: buildAndPush
        repository: $(imageRepository)
        dockerfile: $(dockerfilePath)
        containerRegistry: $(dockerRegistryServiceConnection)
        tags: |
          $(tag)

```

8. To view the published docker image after your pipeline run has been completed, navigate to your container registry in Azure portal, and then select **Repositories**.

The screenshot shows the Azure Container Registry interface. On the left, there's a sidebar with various icons and a list of services. Under 'Services', the 'Repositories' option is selected and highlighted with a red box. In the main content area, there's a search bar labeled 'Search (Ctrl+ /)' and a 'Refresh' button. Below that is a 'Repositories' section with a search bar 'Search to filter repositories ...'. A repository named 'javascriptdocker' is listed and also highlighted with a red box.

9. To deploy your image from the container registry, you must enable the admin user account. Navigate to your container registry in Azure portal, and select **Access keys**. Next, select the toggle button to **Enable Admin user**.

The screenshot shows the 'Access keys' page for the 'sampleappcontinerregistry' container registry. The left sidebar has a 'Settings' section with 'Access keys' highlighted with a red box. The main area contains fields for 'Registry name' (set to 'sampleappcontinerregistry'), 'Login server' (set to 'sampleappcontinerregistry.azurecr.io'), and 'Admin user'. The 'Admin user' field includes a toggle switch that is set to 'Enabled' and is also highlighted with a red box. There are also fields for 'Username' and 'Password'.

Create an Azure Web App for Containers

1. Sign into Azure at <https://portal.azure.com>.
2. In the Azure portal, choose **Create a resource > Containers**, and then choose **Web App for Containers**.

Microsoft Azure (Preview) Search resources, services, and docs (G+/)

Create a resource

Home > Create a resource

Get started

Recently created

Categories

AI + Machine Learning

Analytics

Blockchain

Compute

Containers

Databases

Developer Tools

DevOps

Identity

Integration

Internet of Things

IT & Management Tools

Media

Migration

Mixed Reality

Monitoring & Diagnostics

Networking

Security

Software as a Service (SaaS)

Storage

Web

Search services and marketplace

Featured See all

Container Instances
Create | Docs | MS Learn

Container Registry
Create | Docs | MS Learn

Kubernetes Service
Create | Docs | MS Learn

Web App for Containers
Create | Docs | MS Learn

DC/OS on Azure (preview)
Create | Docs

WordPress Container Image (preview)
Create | Learn more

Node Exporter Container Image (preview)
Create | Learn more

Bitnami Elasticsearch Cluster (preview)
Create | Learn more

3. Enter a name for your new web app, and select or create a new Resource Group. Select **Linux** for the **Operating System**.

Create Web App

...

App Service web Apps lets you quickly build, deploy, and scale enterprise-grade web, mobile, and API apps running on any platform. Meet rigorous performance, scalability, security and compliance requirements while using a fully managed platform to perform infrastructure maintenance. [Learn more](#)

Project Details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * [?](#)

Your Azure Subscription [▼](#)

Resource Group * [?](#)

(New) sample-web-app-rg [▼](#)

[Create new](#)

Instance Details

Name * [?](#)

sample-docker-web-app [✓](#)

.azurewebsites.net

Publish * [?](#)

Code Docker Container

Operating System * [?](#)

Linux Windows

Region * [?](#)

Central US [▼](#)

[Not finding your App Service Plan? Try a different region.](#)

App Service Plan

App Service plan pricing tier determines the location, features, cost and compute resources associated with your app.
[Learn more](#) [?](#)

Linux Plan (Central US) * [?](#)

(New) ASP-samplewebapprg-843d [▼](#)

[Create new](#)

Sku and size * [?](#)

Free F1

1 GB memory

[Change size](#)

[Review + create](#)

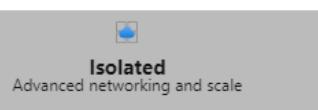
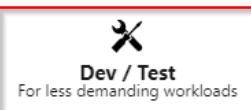
[< Previous](#)

[Next : Deployment \(Preview\) >](#)

4. In the **SKU and Size** section, select **Change** to specify the pricing tier. Select the **Dev/Test** plan, and then choose the **F1 Free plan**. Select **Apply** when you are done.

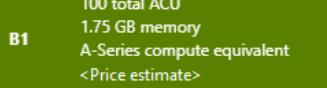
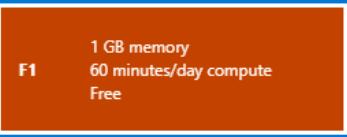
Spec Picker

X



i The first Basic (B1) core for Linux is free for the first 30 days!

Recommended pricing tiers



▼ See additional options

Included hardware

Every instance of your App Service plan will include the following hardware configuration:

	Memory Memory available to run applications deployed and running in the App Service plan.
	Storage 1 GB disk storage shared by all apps deployed in the App Service plan.

Apply

5. Select **Review and create**. Review your configuration, and select **Create** when you are done.

Create a release pipeline

1. From within your project, select **Pipelines** then **Release**.
2. Select **New pipeline** to create a new release pipeline.
3. Select the **Azure App Service deployment** template

Select a template
Or start with an [Empty job](#)

Search

Featured



Azure App Service deployment

Deploy your application to Azure App Service. Choose from Web App on Windows, Linux, containers, Function Apps, or WebJobs.



Deploy a Java app to Azure App Service

Deploy a Java application to an Azure Web App.



Deploy a Node.js app to Azure App Service

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



Deploy a PHP app to Azure App Service and Azure Database for MySQL

Deploy a PHP application to an Azure Web App and database to Azure Database for MySQL.



Deploy a Python app to Azure App Service and Azure database for MySQL

Deploy a Python Django, Bottle, or Flask application to an Azure Web App and database to Azure Database for MySQL.



Deploy to a Kubernetes cluster

Deploy, configure, update your containerized applications to a Kubernetes cluster.



IIS website and SQL database deployment

Deployment Group: Deploy ASP.NET or ASP.NET Core web applications to an IIS Website and SQL database on physical or virtual machines (VM).

4. Select Tasks, then **Unlink all** in stage 1 to unlink all the pipeline parameters.

The screenshot shows the Azure DevOps Pipeline editor interface. On the left, there's a sidebar with various icons for different task types. The main area shows a pipeline with one stage named "Stage 1". The "Tasks" tab is selected, indicated by a red box around the tab name. Inside the stage, there are two tasks: "Run on agent" and "Deploy Azure App Service". The "Deploy Azure App Service" task has a red box around its "Parameters" section, which contains a link labeled "Unlink all". Below the tasks, there are fields for "Stage name" (set to "Stage 1"), "Parameters" (with a note about required settings), "App type" (set to "Web App on Windows"), and "App service name" (with a note about required settings). At the top right, there are "Save" and "Create release" buttons.

5. Select the **Deploy Azure App Service** task, and fill out the required fields. Select **Save** when you are done.

Pipeline Tasks Variables Retention Options History

Stage 1 Deployment process

Run on agent +

Deploy Azure App Service Azure App Service deploy

Azure App Service deploy

Task version 4.*

Display name * Deploy Azure App Service

Connection type * Azure Resource Manager

Azure subscription * Manage Your Azure Subscription

App Service type * Web App for Containers (Linux)

App Service name * sample-docker-web-app

Deploy to Slot or App Service Environment

Registry or Namespace * sampleappcontinerregistry.azurecr.io

Image * javascriptdocker

Tag \$(Build.BuildId)

Startup command

6. Select **Create release**, and then choose **Stage 1** from the dropdown menu. Select **Create** when you are done.

Create a new release

New release pipeline

Pipeline ^

Click on a stage to change its trigger from automated to manual.

Stage 1

Stages for a trigger change from automated to manual.

Stage 1

Release description

Create **Cancel**

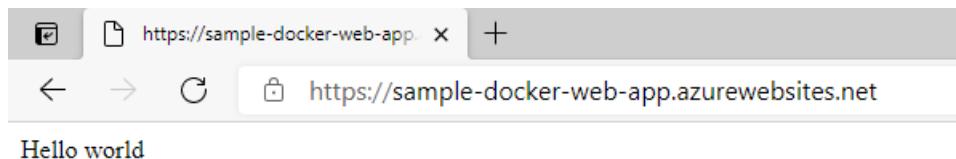
7. Hover over **Stage 1** in your pipeline, and select **Deploy** to queue and start the deployment.

The screenshot shows the Azure DevOps Release pipeline interface. On the left, there's a 'Release' summary card with 'Manually triggered' status by 'Rami Bououni' on '4/9/2021, 2:43 PM'. Below it, an 'Artifacts' section shows 'No artifacts'. To the right, the 'Stages' section lists 'Stage 1' which is currently 'Not deployed'. There are 'Deploy' and 'Logs' buttons below the stage. A red box highlights the 'Deploy' button. On the far right, the 'Stage 1' details pane is open under 'Overview', stating 'To be deployed (Deploying for the first time)' for 'Release-1'. It also notes 'No artifact is associated with this release' and has a 'Comment' field and a 'Deploy' button.

8. Your pipeline logs should look similar to the screenshot below.

```
15 2021-04-12T18:32:00.4640649Z Successfully updated deployment History at https://sample-docker-web-app.scm.azurewebsites.net/api/deployments/21618252317907
16 2021-04-12T18:32:00.7900417Z App Service Application URL: http://sample-docker-web-app.azurewebsites.net
17 2021-04-12T18:32:00.8051842Z ##[section]Finishing: Deploy - sample-app-service
```

9. Navigate to your newly deployed web app to verify your deployment.



Next steps

- [Set up multi-stage release](#)
- [Stage templates in Azure Pipelines](#)
- [Deploy from multiple branches using Azure Pipelines](#)

Deploy a Function App Container

6/30/2021 • 5 minutes to read • [Edit Online](#)

You can automatically deploy an Azure Functions function app using a custom container after every successful build.

Azure Functions is a serverless solution that allows you to write less code, maintain less infrastructure, and save on costs. You can use Azure Pipelines to deploy to Azure Functions and quickly get your code up-and-running.

Prerequisites

- A GitHub account, where you can create a repository. If you don't have one, you can [create one for free](#).
- An Azure DevOps organization. If you don't have one, you can [create one for free](#). (An Azure DevOps organization is different from your GitHub organization. Give them the same name if you want alignment between them.)

If your team already has one, then make sure you're an administrator of the Azure DevOps project that you want to use.

- An ability to run pipelines on Microsoft-hosted agents. You can either purchase a [parallel job](#) or you can request a free tier. To request a free tier, follow the instructions in [this article](#). Please note that it may take us 2-3 business days to grant the free tier.

Create your Function App for Containers

If you already have an app in GitHub that you want to deploy, you can try creating a pipeline for that code.

To use sample code instead, fork this GitHub repo:

```
https://github.com/azoooinmyluggage/GHFunctionAppContainer
```

Build your app

- [YAML](#)
- [Classic](#)

Follow the [Build, test, and push Docker container apps](#) to set up the build pipeline. When you're done, you'll have a YAML pipeline to build, test, and push the image to container registry.

We aren't yet advising new users to use YAML pipelines to deploy from Azure DevOps Server 2019. If you're an experienced pipeline user and already have a YAML pipeline to build your .NET Core app, then you might find the examples below useful.

YAML pipelines aren't available on TFS.

Now that the build pipeline is in place, you will learn a few more common configurations to customize the deployment of the Azure Function App Container.

Add a service connection

The [Azure Function App on Container 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 Azure DevOps Server to Azure.

- [YAML](#)
- [Classic](#)

You must supply an Azure service connection to the `AzureFunctionAppContainer` task. Add the following YAML snippet to your existing `azure-pipelines.yaml` file. Make sure you add the service connection details in the variables section as shown below.

```
variables:  
## Add this under variables section in the pipeline  
azureSubscription: <Name of the Azure subscription>  
appName: <Name of the function App>  
containerRegistry: <Name of the Azure container registry>
```

YAML pipelines aren't available on TFS.

Configure registry credentials in Function App

App Service needs information about your registry and image to pull the private image. In the [Azure portal](#), go to your **Function App --> Platform features --> All settings**. Select **Container settings** from the app service and update the **Image source**, **Registry** and save.

Search (Ctrl+ /)

Single Container

Image source

Azure Container Registry Docker Hub Private Registry

Registry

Image

Tag

4915

Startup File

Continuous Deployment

On Off

Webhook URL [show url](#)

Logs

2019_09_18_RD0004FFD128EA_docker.log:
2019-09-18 06:01:01.688 INFO - Pulling image: [redacted]
2019-09-18 06:01:02.110 INFO - 4915 Pulling from containerfapp

Save Discard

Deploy with the Azure Function App for Container task

- [YAML](#)
- [Classic](#)

The simplest way to deploy to an Azure Function App Container is to use the [Azure Function App on Container Deploy](#) task.

To deploy to an Azure Function App container, add the following snippet at the end of your `azure-pipelines.yml` file:

```
trigger:
- main

variables:
# Container registry service connection established during pipeline creation
dockerRegistryServiceConnection: <Docker registry service connection>
imageRepository: <Name of your image repository>
containerRegistry: <Name of the Azure container registry>
dockerfilePath: '$(Build.SourcesDirectory)/Dockerfile'
tag: '$(Build.BuildId)'

# Agent VM image name
vmImageName: 'ubuntu-latest'

- task: AzureFunctionAppContainer@1 # Add this at the end of your file
  inputs:
    azureSubscription: '<Azure service connection>'
    appName: '<Name of the function app>'
    imageName: $(containerRegistry)}/${imageRepository}:$(tag)
```

The snippet assumes that the build steps in your YAML file build and push the docker image to your Azure container registry. The **Azure Function App on Container Deploy** task will pull the appropriate docker image corresponding to the BuildId from the repository specified, and then deploys the image to the Azure Function App Container.

YAML pipelines aren't available on TFS.

Deploy to a slot

- [YAML](#)
- [Classic](#)

You can configure the Azure Function App container to have multiple slots. Slots allow you to safely deploy your app and test it before making it available to your customers.

The following YAML snippet shows how to deploy to a staging slot, and then swap to a production slot:

```
- task: AzureFunctionAppContainer@1
  inputs:
    azureSubscription: <Azure service connection>
    appName: <Name of the function app>
    imageName: $(containerRegistry)/${imageRepository}:$(tag)
    deployToSlotOrASE: true
    resourceGroupName: <Name of the resource group>
    slotName: staging

- task: AzureAppServiceManage@0
  inputs:
    azureSubscription: <Azure service connection>
    WebAppName: <name of the function app>
    ResourceGroupName: <name of resource group>
    SourceSlot: staging
    SwapWithProduction: true
```

YAML pipelines aren't available on TFS.

Deploy an Azure Function

6/30/2021 • 5 minutes to read • [Edit Online](#)

Azure Functions is a serverless solution that allows you to write less code, maintain less infrastructure, and save on costs. You can use Azure Pipelines to deploy to Azure Functions and quickly get your code up-and-running.

Learn how to deploy to a Linux function app.

Prerequisites

- A GitHub account, where you can create a repository. If you don't have one, you can [create one for free](#).
- An Azure DevOps organization. If you don't have one, you can [create one for free](#). (An Azure DevOps organization is different from your GitHub organization. Give them the same name if you want alignment between them.)
If your team already has one, then make sure you're an administrator of the Azure DevOps project that you want to use.
- An ability to run pipelines on Microsoft-hosted agents. You can either purchase a [parallel job](#) or you can request a free tier. To request a free tier, follow the instructions in [this article](#). Please note that it may take us 2-3 business days to grant the free tier.

Create your Linux function app

This is a step-by-step guide to using Azure Pipelines with Azure Functions for Windows.

- [.NET Core](#)
- [Java](#)
- [Nodejs](#)

If you already have an app in GitHub that you want to deploy, you can try creating a pipeline for that code.

To use sample code instead, fork this GitHub repo:

```
https://github.com/microsoft/devops-project-samples/tree/master/dotnet/aspnetcore/functionApp
```

Build your app

- [YAML](#)
- [Classic](#)

Follow the guidance in [Create your first pipeline](#) to set up the build pipeline. The CI steps will be similar to any Nodejs or .NET Core apps. When you're done, you'll have a YAML pipeline to build, test, and publish the source as an artifact.

We aren't yet advising new users to use YAML pipelines to deploy from Azure DevOps Server 2019. If you're an experienced pipeline user and already have a YAML pipeline to build your java function app, then you might find the examples below useful.

YAML pipelines aren't available on TFS.

Now you're ready to read through the rest of this topic to learn some of the more common configurations to customize the deployment of an Azure Function App.

Azure service connection

The [Azure Function App 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 Azure DevOps Server to Azure.

- [YAML](#)
- [Classic](#)

You must supply an Azure service connection to the `AzureFunctionApp` task. Add the following YAML snippet to your existing `azure-pipelines.yaml` file. Make sure you add the service connection details in the variables section as shown below-

```
variables:  
## Add this under variables section in the pipeline  
azureSubscription: <Name of the Azure subscription>  
appName: <Name of the Function App>
```

The snippet assumes that the build steps in your YAML file build and publishes the source as an artifact. The [Azure Function App Deploy](#) task will pull the artifact corresponding to the BuildId from the **Source type** specified, and then deploys the artifact to the Azure Function App Service.

YAML pipelines aren't available on TFS.

Deploy with Azure Function App

- [YAML](#)
- [Classic](#)

The simplest way to deploy to an Azure Function is to use the [Azure Function App Deploy](#) task.

To deploy to Azure Function, add the following snippet at the end of your `azure-pipelines.yml` file:

```
trigger:  
- main  
  
variables:  
# Azure service connection established during pipeline creation  
azureSubscription: <Name of your Azure subscription>  
appName: <Name of the Function app>  
# Agent VM image name  
vmImageName: 'ubuntu-latest'  
  
- task: AzureFunctionApp@1 # Add this at the end of your file  
inputs:  
azureSubscription: <Azure service connection>  
appType: functionAppLinux  
appName: $(appName)  
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.

YAML pipelines aren't available on TFS.

Deploy to a slot

- [YAML](#)
- [Classic](#)

You can configure the Azure Function 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 YAML snippet shows how to deploy to a staging slot, and then swap to a production slot:

```
- task: AzureFunctionApp@1
  inputs:
    azureSubscription: <Azure service connection>
    appType: functionAppLinux
    appName: <Name of the Function app>
    package: $(System.ArtifactsDirectory)/**/*.zip
    deployToSlotOrASE: true
    resourceGroupName: <Name of the resource group>
    slotName: staging

- task: AzureAppServiceManage@0
  inputs:
    azureSubscription: <Azure service connection>
    WebAppName: <name of the Function app>
    ResourceGroupName: <name of resource group>
    SourceSlot: staging
    SwapWithProduction: true
```

YAML pipelines aren't available on TFS.

Deploy an Azure Function (Windows)

6/30/2021 • 5 minutes to read • [Edit Online](#)

Azure Functions is a serverless solution that allows you to write less code, maintain less infrastructure, and save on costs. You can use Azure Pipelines to deploy to Azure Functions and quickly get your code up-and-running.

You can automatically deploy your Azure Function after every successful build.

Prerequisites

- A GitHub account, where you can create a repository. If you don't have one, you can [create one for free](#).
- An Azure DevOps organization. If you don't have one, you can [create one for free](#). (An Azure DevOps organization is different from your GitHub organization. Give them the same name if you want alignment between them.)
If your team already has one, then make sure you're an administrator of the Azure DevOps project that you want to use.
- An ability to run pipelines on Microsoft-hosted agents. You can either purchase a [parallel job](#) or you can request a free tier. To request a free tier, follow the instructions in [this article](#). Please note that it may take us 2-3 business days to grant the free tier.

Create your Windows function App

This is a step-by-step guide to using Azure Pipelines with Azure Functions for Windows.

- [.NET Core](#)
- [Java](#)
- [Nodejs](#)

Fork this repo in GitHub:

```
https://github.com/microsoft/devops-project-samples/tree/master/dotnet/aspnetcore/functionApp
```

Build your app

- [YAML](#)
- [Classic](#)

1. Sign in to your Azure DevOps organization and navigate to your project.
2. In your project, navigate to the [Pipelines](#) page. Then choose the action to create a new pipeline.
3. Walk through the steps of the wizard by first selecting **GitHub** as the location of your source code.
4. You might be redirected to GitHub to sign in. If so, enter your GitHub credentials.
5. When the list of repositories appears, select your desired sample app repository.
6. Azure Pipelines will analyze your repository and recommend a template. Select **Save and run**, then select **Commit directly to the main branch**, and then choose **Save and run** again.
7. A new run is started. Wait for the run to finish.

We aren't yet advising new users to use YAML pipelines to deploy from Azure DevOps Server 2019. If you're an

experienced pipeline user and already have a YAML pipeline to build your java function app, then you might find the examples below useful.

YAML pipelines aren't available on TFS.

Now you're ready to read through the rest of this topic to learn some of the more common configurations to customize the deployment of an Azure Function App.

Azure service connection

The [Azure Function App Deploy task](#) requires an Azure service connection as an input. The Azure service connection stores the credentials to connect from Azure Pipelines or Azure DevOps Server to Azure.

- [YAML](#)
- [Classic](#)

You must supply an Azure service connection to the AzureFunctionApp task. Add the following YAML snippet to your existing `azure-pipelines.yaml` file. Make sure you add the service connection details in the variables section as shown below.

```
variables:  
    ## Add this under variables section in the pipeline  
    azureSubscription: <Name of the Azure subscription>  
    appName: <Name of the Function App>  
  
    ## Add the below snippet at the end of your pipeline  
    - task: AzureFunctionApp@1  
        displayName: Azure Function App Deploy  
        inputs:  
            azureSubscription: $(azureSubscription)  
            appType: functionApp  
            appName: $(appName)  
            package: $(System.ArtifactsDirectory)/**/*.zip
```

The snippet assumes that the build steps in your YAML file build and publishes the source as an artifact. The [Azure Function App Deploy task](#) will pull the artifact corresponding to the BuildId from the **Source type** specified, and then deploys the artifact to the Azure Function App Service.

YAML pipelines aren't available on TFS.

Deploy with Azure Function App

- [YAML](#)
- [Classic](#)

The simplest way to deploy to an Azure Function is to use the [Azure Function App Deploy task](#).

To deploy to Azure Function, add the following snippet at the end of your `azure-pipelines.yml` file:

```

trigger:
- main

variables:
# Azure service connection established during pipeline creation
azureSubscription: <Name of your Azure subscription>
appName: <Name of the Function app>
# Agent VM image name
vmImageName: 'ubuntu-latest'

- task: AzureFunctionApp@1 # Add this at the end of your file
inputs:
  azureSubscription: <Azure service connection>
  appType: functionApp
  appName: $(appName)
  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.

YAML pipelines aren't available on TFS.

Deploy to a slot

- [YAML](#)
- [Classic](#)

You can configure the Azure Function 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 YAML snippet shows how to deploy to a staging slot, and then swap to a production slot:

```

- task: AzureFunctionApp@1
inputs:
  azureSubscription: <Azure service connection>
  appType: functionApp
  appName: <Name of the Function app>
  package: $(System.ArtifactsDirectory)/**/*.zip
  deployToSlotOrASE: true
  resourceGroupName: <Name of the resource group>
  slotName: staging

- task: AzureAppServiceManage@0
inputs:
  azureSubscription: <Azure service connection>
  WebAppName: <name of the Function app>
  ResourceGroupName: <name of resource group>
  SourceSlot: staging
  SwapWithProduction: true

```

YAML pipelines aren't available on TFS.

Quickstart: Use an ARM template to deploy a Linux web app to Azure

5/27/2021 • 4 minutes to read • [Edit Online](#)

Get started with [Azure Resource Manager templates \(ARM templates\)](#) by deploying a Linux web app with MySQL. ARM templates give you a way to save your configuration in code. Using an ARM template is an example of infrastructure as code and a good DevOps practice.

An [ARM template](#) is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project. The template uses declarative syntax. In declarative syntax, you describe your intended deployment without writing the sequence of programming commands to create the deployment.

Prerequisites

Before you begin, you need:

- An Azure account with an active subscription. [Create an account for free](#).
- An active Azure DevOps organization. [Sign up for Azure Pipelines](#).

Get the code

Fork this repo in GitHub:

```
https://github.com/Azure/azure-quickstart-templates/
```

Review the template

The template used in this quickstart is from [Azure Quickstart Templates](#).

The template defines several resources:

- [Microsoft.Web/serverfarms](#)
- [Microsoft.Web/sites](#)
- [Microsoft.DBforMySQL/servers](#)
- [Microsoft.DBforMySQL/servers/firewallrules](#)
- [Microsoft.DBforMySQL/servers/databases](#)

Create your pipeline and deploy your template

1. Sign in to your Azure DevOps organization and navigate to your project. [Create a project](#) if you do not already have one.
2. Go to **Pipelines**, and then select **Create Pipeline**.
3. Select **GitHub** as the location of your source code.

NOTE

You may be redirected to GitHub to sign in. If so, enter your GitHub credentials.

4. When the list of repositories appears, select `yourusername/azure-quickstart-templates/`.

NOTE

You may be redirected to GitHub to install the Azure Pipelines app. If so, select **Approve and install**.

5. When the Configure tab appears, select `Starter pipeline`.

6. Replace the content of your pipeline with this code:

```
trigger:  
- none  
  
pool:  
vmImage: 'ubuntu-latest'
```

7. Create three variables: `siteName`, `administratorLogin`, and `administratorLoginPassword`.

`administratorLoginPassword` needs to be a secret variable.

- Select **Variables**.
- Use the `+` sign to add three variables. When you create `administratorLoginPassword`, select **Keep this value secret**.
- Click **Save** when you're done.

VARIABLE	VALUE	SECRET?
siteName	<code>mytestsite</code>	No
adminUser	<code>fabrikam</code>	No
adminPass	<code>Fqdn:5362!</code>	Yes

8. Map the secret variable `$(adminPass)` so that it is available in your Azure Resource Group Deployment task. At the top of your YAML file, map `$(adminPass)` to `$(ARM_PASS)`.

```
variables:  
ARM_PASS: $(adminPass)  
  
trigger:  
- none  
  
pool:  
vmImage: 'ubuntu-latest'
```

9. Add the Copy Files task to the YAML file. You will use the `101-webapp-linux-managed-mysql` project. For more information, see [Build a Web app on Linux with Azure database for MySQL](#) repo for more details.

```

variables:
  ARM_PASS: $(adminPass)

trigger:
- none

pool:
  vmImage: 'ubuntu-latest'

steps:
- task: CopyFiles@2
  inputs:
    SourceFolder: '101-webapp-linux-managed-mysql'
    Contents: '**'
    TargetFolder: '$(Build.ArtifactStagingDirectory)'

```

10. Add and configure the Azure Resource Group Deployment task.

The task references both the artifact you built with the Copy Files task and your pipeline variables. Set these values when configuring your task.

- **Deployment scope (deploymentScope)**: Set the deployment scope to `Resource Group`. You can target your deployment to a management group, an Azure subscription, or a resource group.
- **Azure Resource Manager connection (azureResourceManagerConnection)**: Select your Azure Resource Manager service connection. To configure new service connection, select the Azure subscription from the list and click **Authorize**. See [Connect to Microsoft Azure](#) for more details
- **Subscription (subscriptionId)**: Select the subscription where the deployment should go.
- **Action (action)**: Set to `Create or update resource group` to create a new resource group or to update an existing one.
- **Resource group**: Set to `ARMPipelinesLAMP-rg` to name your new resource group. If this is an existing resource group, it will be updated.
- **Location(location)**: Location for deploying the resource group. Set to your closest location (for example, West US). If the resource group already exists in your subscription, this value will be ignored.
- **Template location (templateLocation)**: Set to `Linked artifact`. This is location of your template and the parameters files.
- **Template (cmsFile)**: Set to `$(Build.ArtifactStagingDirectory)/azuredeploy.json`. This is the path to the ARM template.
- **Template parameters (cmsParametersFile)**: Set to `$(Build.ArtifactStagingDirectory)/azuredeploy.parameters.json`. This is the path to the parameters file for your ARM template.
- **Override template parameters (overrideParameters)**: Set to `-siteName $(siteName) -administratorLogin $(adminUser) -administratorLoginPassword $(ARM_PASS)` to use the variables you created earlier. These values will replace the parameters set in your template parameters file.
- **Deployment mode (deploymentMode)**: The way resources should be deployed. Set to `Incremental`. Incremental keeps resources that are not in the ARM template and is faster than `Complete`. `Validate` mode lets you find problems with the template before deploying.

```

variables:
  ARM_PASS: $(adminPass)

trigger:
- none

pool:
  vmImage: 'ubuntu-latest'

steps:
- task: CopyFiles@2
  inputs:
    SourceFolder: '101-webapp-linux-managed-mysql'
    Contents: '**'
    TargetFolder: '$(Build.ArtifactStagingDirectory)'

- task: AzureResourceManagerTemplateDeployment@3
  inputs:
    deploymentScope: 'Resource Group'
    azureResourceManagerConnection: '<your-resource-manager-connection>'
    subscriptionId: '<your-subscription-id>'
    action: 'Create Or Update Resource Group'
    resourceGroupName: 'ARMPipelinesLAMP-rg'
    location: '<your-closest-location>'
    templateLocation: 'Linked artifact'
    csmFile: '$(Build.ArtifactStagingDirectory)/azuredeploy.json'
    csmParametersFile: '$(Build.ArtifactStagingDirectory)/azuredeploy.parameters.json'
    overrideParameters: '-siteName $(siteName) -administratorLogin $(adminUser) -administratorLoginPassword $(ARM_PASS)'
    deploymentMode: 'Incremental'

```

- Click **Save and run** to deploy your template. The pipeline job will be launched and after few minutes, depending on your agent, the job status should indicate **Success**.

Review deployed resources

- Verify that the resources deployed. Go to the **ARMPipelinesLAMP-rg** resource group in the Azure portal and verify that you see App Service, App Service Plan, and Azure Database for MySQL server resources.

<input type="checkbox"/> Name ↑↓	Type ↑↓
<input type="checkbox"/> armpipelinetestsit	App Service
<input type="checkbox"/> hpn-jgc4v36g4nflm	App Service plan
<input type="checkbox"/> mysql-jgc4v36g4nflm	Azure Database for MySQL server

You can also verify the resources using Azure CLI.

```
az resource list --resource-group ARMPipelinesLAMP-rg --output table
```

- Go to your new site. If you set **siteName** to **armpipelinetestsit**, the site is located at <https://armpipelinetestsit.azurewebsites.net/>.

Clean up resources

You can also use an ARM template to delete resources. Change the **action** value in your **Azure Resource Group Deployment** task to **DeleteRG**. You can also remove the inputs for **templateLocation**, **csmFile**, **csmParametersFile**, **overrideParameters**, and **deploymentMode**.

```
variables:
  ARM_PASS: $(adminPass)

trigger:
- none

pool:
  vmImage: 'ubuntu-latest'

steps:
- task: CopyFiles@2
  inputs:
    SourceFolder: '101-webapp-linux-managed-mysql'
    Contents: '**'
    TargetFolder: '$(Build.ArtifactStagingDirectory)'

- task: AzureResourceManagerTemplateDeployment@3
  inputs:
    deploymentScope: 'Resource Group'
    azureResourceManagerConnection: '<your-resource-manager-connection>'
    subscriptionId: '<your-subscription-id>'
    action: 'DeleteRG'
    resourceGroupName: 'ARMPipelinesLAMP-rg'
    location: ''<your-closest-location>'
```

Next steps

[Create your first ARM template](#)

CD of an Azure virtual machine using a Resource Manager template

11/2/2020 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

In just a few steps, you can provision Azure virtual machines (VMs) using [Resource Manager \(RM\) templates](#). Managing the pipelines for virtual machines in this way is considered **Infrastructure as code** and is a good DevOps practice.

Prerequisites

Before you begin, you need a CI build that creates your Azure RM template. To set up CI, see:

- [Build an Azure virtual machine using an Azure RM template](#)

Define and test your CD release pipeline

Carry out the following steps to deploy the Azure Resource Group.

1. Open the **Releases** tab of **Azure Pipelines** and choose the "+" icon to create a new release pipeline.
2. In the **Create release pipeline** dialog, select the **Empty** template and choose **Next**.
3. In the next page, select the build pipeline you created earlier and choose **Create**. This creates a new release pipeline with one default stage.
4. In the new release pipeline, select **+ Add tasks** and add an **Azure Resource Group Deployment** task. Optionally edit the name to help identify the task, such as **Provision Windows 2012 R2 VM**.
5. Configure the **Azure Resource Group Deployment** task as follows:



[Deploy: Azure Resource Group Deployment](#) - Deploy files to an Azure Resource Group.

- **Azure Subscription:** Select a connection from the list under **Available Azure Service Connections** or create a more restricted permissions connection to your Azure subscription. For more details, see [Azure Resource Manager service connection](#).
- **Action:** `Create or Update Resource Group`
- **Resource Group:** The name for a new resource group, or an existing resource group name.
- **Template location:** The path of the Resource Manager template; for example:
`$(System.DefaultWorkingDirectory)\ASPNet4.CI\drop\HelloWorldARM\Templates\WindowsVirtualMachine.json`
- **Template Parameters:** The path of the Resource Manager template parameters file; for example:
`$(System.DefaultWorkingDirectory)\ASPNet4.CI\drop\HelloWorldARM\Templates\WindowsVirtualMachine.parameters.json`

- **Override Template Parameters:** A list of values for the parameters in the template; for example:

```
-adminUsername $(vmuser) -adminPassword (ConvertTo-SecureString -String $(vmpassword) -AsPlainText -Force) -dnsNameForPublicIP $(dns)'
```

Use the ... button to open the parameters editor dialog.

- **Enable Deployment Prerequisites:** Checked.
- **Output - Resource Group:** The name of the Resource Group output from the task as a value that can be used as an input to further deployment tasks.

Checking the **Enable Deployment Prerequisites** checkbox configures WinRM on the virtual machine and enables execution of remote PowerShell scripts, which may be required to deploy an application. Also notice the use of **ConvertTo-SecureString** to specify the value for **adminPassword**. You must do this because **adminPassword** is defined as a **SecureString** type in the Resource Manager template file.

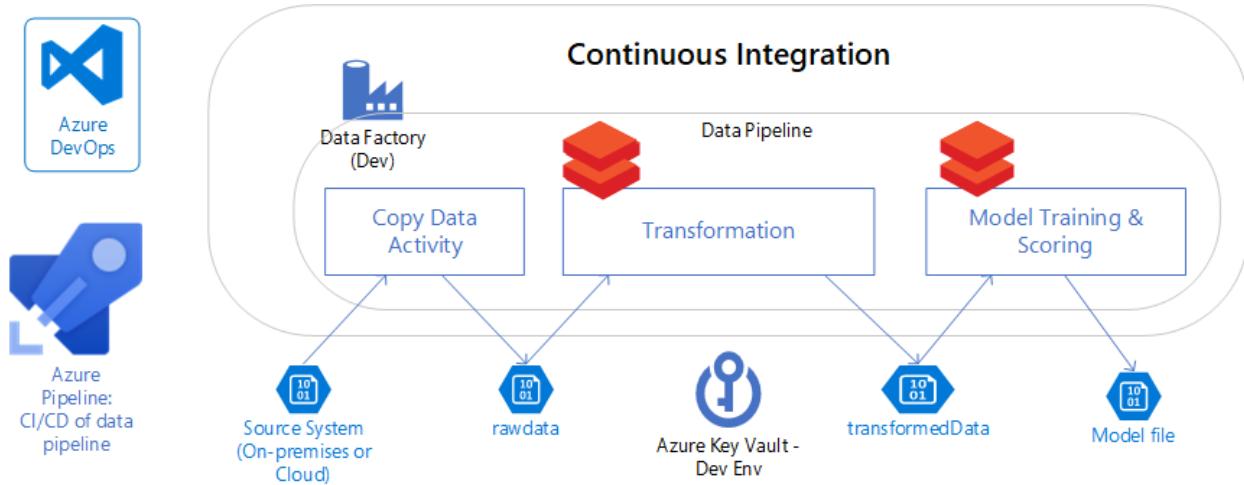
6. If you used **variables** in the parameters of the **Azure Resource Group Deployment** task, such as **vmuser**, **vmpassword**, and **dns**, set the values for them in the stage configuration variables. Encrypt the value of **vmpassword** by selecting the "padlock" icon.
7. Enter a name for the release pipeline and save it.
8. Create a new release, select the latest build, and deploy it to the single stage.

Why data pipelines?

11/2/2020 • 2 minutes to read • [Edit Online](#)

You can use data pipelines to:

- Ingest data from various data sources
- Process and transform the data
- Save the processed data to a staging location for others to consume



Data pipelines in the enterprise can evolve into more complicated scenarios with multiple source systems and supporting various downstream applications.

Data pipelines provide:

- Consistency: Data pipelines transform data into a consistent format for users to consume
- Error reduction: Automated data pipelines eliminate human errors when manipulating data
- Efficiency: Data professionals save time spent on data processing transformation. Saving time allows them to focus on their core job function - getting the insight out of the data and helping business make better decisions

What is CI/CD?

Continuous integration and continuous delivery (CI/CD) is a software development approach where all developers work together on a shared repository of code – and as changes are made, there are automated build processes for detecting code issues. The outcome is a faster development life cycle and a lower error rate.

What is a CICD/ data pipeline and why does it matter for data science?

The building of machine learning models is similar to traditional software development in the sense that the data scientist needs to write code to train and score machine learning models.

Unlike traditional software development where the product is based on code, data science machine learning models are based on both the code (algorithm, hyper parameters) and the data used to train the model. That's why most data scientists will tell you that they spend 80% of the time doing data preparation, cleaning and feature engineering.

To complicate the matter even further – to ensure the quality of the machine learning models, techniques such as A/B testing is used – where there could be multiple machine learning models being used concurrently. There is usually one control model and one or more treatment models for comparison – so that the model performance can be compared and maintained. Having multiple models adds another layer of complexity for the CI/CD of machine learning models.

Having a CI/CD data pipeline is crucial for the data science team to deliver the machine learning models to the business in a timely and quality manner.

Next steps

[Build a data pipeline with Azure](#)

Build a data pipeline by using Azure Data Factory, DevOps, and machine learning

6/14/2021 • 8 minutes to read • [Edit Online](#)

Get started building a data pipeline with data ingestion, data transformation, and model training.

Learn how to grab data from a CSV (comma-separated values) file and save the data to Azure Blob Storage. Transform the data and save it to a staging area. Then train a machine learning model by using the transformed data. Write the model to blob storage as a Python [pickle file](#).

Prerequisites

Before you begin, you need:

- An Azure account that has an active subscription. [Create an account for free](#).
- An active Azure DevOps organization. [Sign up for Azure Pipelines](#).
- Data from [sample.csv](#).
- Access to the [data pipeline solution](#) in GitHub.
- [DevOps for Azure Databricks](#).

Provision Azure resources

1. Sign in to the [Azure portal](#).
2. From the menu, select the **Cloud Shell** button. When you're prompted, select the **Bash** experience.



NOTE

You'll need an Azure Storage resource to persist any files that you create in Azure Cloud Shell. When you first open Cloud Shell, you're prompted to create a resource group, storage account, and Azure Files share. This setup is automatically used for all future Cloud Shell sessions.

Select an Azure region

A *region* is one or more Azure datacenters within a geographic location. East US, West US, and North Europe are examples of regions. Every Azure resource, including an App Service instance, is assigned a region.

To make commands easier to run, start by selecting a default region. After you specify the default region, later commands use that region unless you specify a different region.

1. In Cloud Shell, run the following `az account list-locations` command to list the regions that are available from your Azure subscription.

```
az account list-locations \
--query "[].{Name: name, DisplayName: displayName}" \
--output table
```

2. From the `Name` column in the output, choose a region that's close to you. For example, choose `eastasia` or `westus2`.

3. Run `az config` to set your default region. In the following example, replace `<REGION>` with the name of the region you chose.

```
az config set defaults.location=<REGION>
```

The following example sets `westus2` as the default region.

```
az config set defaults.location=westus2
```

Create Bash variables

1. In Cloud Shell, generate a random number. You'll use this number to create globally unique names for certain services in the next step.

```
resourceSuffix=$RANDOM
```

2. Create globally unique names for your storage account and key vault. The following commands use double quotation marks, which instruct Bash to interpolate the variables by using the inline syntax.

```
storageName="datacid${resourceSuffix}"
keyVault="keyvault${resourceSuffix}"
```

3. Create one more Bash variable to store the names and the region of your resource group. In the following example, replace `<REGION>` with the region that you chose for the default region.

```
rgName='data-pipeline-cicd-rg'
region='<REGION>'
```

4. Create variable names for your Azure Data Factory and Azure Databricks instances.

```
datafactorydev='data-factory-cicd-dev'
datafactorytest='data-factory-cicd-test'

databricksname='databricks-cicd-ws'
```

Create Azure resources

1. Run the following `az group create` command to create a resource group by using `rgName`.

```
az group create --name $rgName
```

2. Run the following `az storage account create` command to create a new storage account.

```
az storage account create \
--name $storageName \
--resource-group $rgName \
--sku Standard_RAGRS \
--kind StorageV2
```

3. Run the following `az storage container create` command to create two containers, `rawdata` and `prepareddata`.

```
az storage container create -n rawdata --account-name $storageName  
az storage container create -n prepareddata --account-name $storageName
```

4. Run the following `az keyvault create` command to create a new key vault.

```
az keyvault create \  
--name $keyVault \  
--resource-group $rgName
```

5. [Create a new data factory](#) by using the portal UI or the Azure CLI:

- Name: `data-factory-cicd-dev`
- Version: `v2`
- Resource group: `data-pipeline-cicd-rg`
- Location: Your closest location
- Clear the selection for **Enable GIT**.

- a. Add the Azure Data Factory extension.

```
az extension add --name datafactory
```

- b. Run the following `az datafactory factory create` command to create a new data factory.

```
az datafactory factory create \  
--name data-factory-cicd-dev \  
--resource-group $rgName
```

- c. Copy the subscription ID. Your data factory will use this ID later.

6. [Create a second data factory](#) by using the portal UI or the Azure CLI. You'll use this data factory for testing.

- Name: `data-factory-cicd-test`
- Version: `v2`
- Resource group: `data-pipeline-cicd-rg`
- Location: Your closest location
- Clear the selection for **Enable GIT**.

- a. Run the following `az datafactory factory create` command to create a new data factory for testing.

```
az datafactory factory create \  
--name data-factory-cicd-test \  
--resource-group $rgName
```

- b. Copy the subscription ID. Your data factory will use this ID later.

7. [Add a new Azure Databricks service](#):

- Resource group: `data-pipeline-cicd-rg`
- Workspace name: `databricks-cicd-ws`

- Location: Your closest location
- Add the Azure Databricks extension if it's not already installed.

```
az extension add --name databricks
```

- Run the following `az databricks workspace create` command to create a new workspace.

```
az databricks workspace create \
--resource-group $rgName \
--name databricks-cicd-ws \
--location eastus2 \
--sku trial
```

- Copy the subscription ID. Your Databricks service will use this ID later.

Upload data to your storage container

- In the Azure portal, open your storage account in the `data-pipeline-cicd-rg` resource group.
- Go to **Blob Service > Containers**.
- Open the `prepareddata` container.
- Upload the [sample.csv](#) file.

Set up Key Vault

You'll use Azure Key Vault to store all connection information for your Azure services.

Create a Databricks personal access token

- In the Azure portal, go Databricks and then open your workspace.
- In the Azure Databricks UI, [create and copy a personal access token](#).

Copy the account key and connection string for your storage account

- Go to your storage account.
- Open **Access keys**.
- Copy the first key and connection string.

Save values to Key Vault

- Create three secrets:
 - databricks-token: `your-databricks-pat`
 - StorageKey: `your-storage-key`
 - StorageConnectionString: `your-storage-connection`
 - Run the following `az keyvault secret set` command to add secrets to your key vault.
- ```
az keyvault secret set --vault-name "$keyVault" --name "databricks-token" --value "your-databricks-
pat"
az keyvault secret set --vault-name "$keyVault" --name "StorageKey" --value "your-storage-key"
az keyvault secret set --vault-name "$keyVault" --name "StorageConnectionString" --value "your-storage-
connection"
```

## Import the data pipeline solution

- Sign in to your Azure DevOps organization and then go to your project.

2. Go to **Repos** and then import your forked version of the [GitHub repository](#). For more information, see [Import a Git repo into your project](#).

## Add an Azure Resource Manager service connection

1. Create an Azure Resource Manager [service connection](#).
2. Select **Service Principal (automatic)**.
3. Choose the **data-pipeline-cicd-rg** resource group.
4. Name the service connection `azure_rm_connection`.
5. Select **Grant access permission to all pipelines**.

## Add pipeline variables

1. [Create a new variable group](#) named `datapipline-vg`.
2. Add the Azure DevOps extension if it isn't already installed.

```
az extension add --name azure-devops
```

3. Sign in to your [Azure DevOps organization](#).

```
az devops login --org https://dev.azure.com/<yourorganizationname>
```

```
az pipelines variable-group create --name datapipline-vg -p <yourazuredevopsprojectname> --variables \
\
 "LOCATION=$region" \
 "RESOURCE_GROUP=$rgName" \
 "DATA_FACTORY_NAME=$datafactorydev" \
 "DATA_FACTORY_DEV_NAME=$datafactorydev" \
 "DATA_FACTORY_TEST_NAME=$datafactorytest" \
 "ADF_PIPELINE_NAME=DataPipeline" \
 "DATABRICKS_NAME=$databricksname" \
 "AZURE_RM_CONNECTION=azure_rm_connection" \
 "DATABRICKS_URL=<URL copied from Databricks in Azure portal>" \
 "STORAGE_ACCOUNT_NAME=$storageName" \
 "STORAGE_CONTAINER_NAME=rawdata"
```

4. Create a second variable group named `keys-vg`. This group will pull data variables from Key Vault.
5. Select **Link secrets from an Azure key vault as variables**. For more information, see [Link secrets from an Azure key vault](#).
6. Authorize the Azure subscription.
7. Choose all of the available secrets to add as variables (`databricks-token`, `StorageConnectionString`, `StorageKey`).

## Configure Azure Databricks and Azure Data Factory

Follow the steps in the next sections to set up Azure Databricks and Azure Data Factory.

### Create testscope in Azure Databricks

1. In the Azure portal, go to **Key vault** > **Properties**.
2. Copy the **DNS Name** and **Resource ID**.
3. In your Azure Databricks workspace, [create a secret scope](#) named `testscope`.

## Add a new cluster in Azure Databricks

1. In the Azure Databricks workspace, go to **Clusters**.
2. Select **Create Cluster**.
3. Name and save your new cluster.
4. Select your new cluster name.
5. In the URL string, copy the content between `/clusters/` and `/configuration`. For example, in the string `clusters/0306-152107-daft561/configuration`, you would copy `0306-152107-daft561`.
6. Save this string to use later.

## Set up your code repository in Azure Data Factory

1. In Azure Data Factory, go to **Author & Monitor**. For more information, see [Create a data factory](#).
2. Select **Set up code repository** and then connect your repo.
  - Repository type: Azure DevOps Git
  - Azure DevOps organization: Your active account
  - Project name: Your Azure DevOps data pipeline project
  - Git repository name: **Use existing**.
    - Select the **main** branch for collaboration.
    - Set `/azure-data-pipeline/factorydata` as the root folder.
  - Branch to import resource into: Select **Use existing** and **main**.

## Link Azure Data Factory to your key vault

1. In the Azure portal UI, open the key vault.
2. Select **Access policies**.
3. Select **Add Access Policy**.
4. For **Configure from template**, select **Key & Secret Management**.
5. In **Select principal**, search for the name of your development data factory and add it.
6. Select **Add** to add your access policies.
7. Repeat these steps to add an access policy for the test data factory.

## Update the key vault linked service in Azure Data Factory

1. Go to **Manage > Linked services**.
2. Update the Azure key vault to connect to your subscription.

## Update the storage linked service in Azure Data Factory

1. Go to **Manage > Linked services**.
2. Update the Azure Blob Storage value to connect to your subscription.

## Update the Azure Databricks linked service in Azure Data Factory

1. Go to **Manage > Linked services**.
2. Update the Azure Databricks value to connect to your subscription.
3. For the **Existing Cluster ID**, enter the cluster value you saved earlier.

## Test and publish the data factory

1. In Azure Data Factory, go to **Edit**.
2. Open `DataPipeline`.
3. Select **Variables**.
4. Verify that the `storage_account_name` refers to your storage account in the Azure portal. Update the default value if necessary. Save your changes.
5. Select **Validate** to verify `DataPipeline`.
6. Select **Publish** to publish data-factory assets to the `adf_publish` branch of your repository.

## Run the CI/CD pipeline

Follow these steps to run the continuous integration and continuous delivery (CI/CD) pipeline:

1. Go to the **Pipelines** page. Then choose the action to create a new pipeline.
2. Select **Azure Repos Git** as the location of your source code.
3. When the list of repositories appears, select your repository.
4. As you set up your pipeline, select **Existing Azure Pipelines YAML file**. Choose the YAML file: `/azure-data-pipeline/data_pipeline_ci_cd.yml`.
5. Run the pipeline. If your pipeline hasn't been run before, you might need to give permission to access a resource during the run.

## Clean up resources

If you're not going to continue to use this application, delete your data pipeline by following these steps:

1. Delete the `data-pipeline-cicd-rg` resource group.
2. Delete your Azure DevOps project.

## Next steps

[Learn more about data in Azure Data Factory](#)

# Prepare data, train, deploy, and monitor machine learning models with Azure Pipelines

4/27/2021 • 7 minutes to read • [Edit Online](#)

## Azure Pipelines

You can use a pipeline to automate the machine learning lifecycle. Some of the operations you can automate are:

- Data preparation (extract, transform, load operations)
- Training machine learning models with on-demand scale-out and scale-up
- Deployment of machine learning models as public or private web services
- Monitoring deployed machine learning models (such as for performance or data-drift analysis)

This article will teach you how to create an Azure Pipeline that builds and deploys a machine learning model as a web service.

## Prerequisites

Before you read this topic, you should understand [how the Azure Machine Learning service works](#).

Follow the steps in [Azure Machine Learning quickstart: portal](#) to create a workspace.

## Get the code

Fork this repo in GitHub:

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

This sample includes a file *diabetes-train-and-deploy.yml* in the directory *pipelines/*.

## Sign in to Azure Pipelines

Sign in to [Azure Pipelines](#). After you sign in, your browser goes to <https://dev.azure.com/my-organization-name> and displays your Azure DevOps dashboard.

Within your selected organization, create a *project*. If you don't have any projects in your organization, you see a **Create a project to get started** screen. Otherwise, select the **Create Project** button in the upper-right corner of the dashboard.

## Create the pipeline

You can use 1 of the following approach to create a new pipeline.

- [YAML](#)
- [Classic](#)

1. Sign in to your Azure DevOps organization and navigate to your project.
2. Go to **Pipelines**, and then select **New Pipeline**.
3. Walk through the steps of the wizard by first selecting **GitHub** as the location of your source code.

4. You might be redirected to GitHub to sign in. If so, enter your GitHub credentials.
5. When the list of repositories appears, select your repository.
6. You might be redirected to GitHub to install the Azure Pipelines app. If so, select **Approve & install**.

When your new pipeline appears:

1. Replace `myresourcegroup` with the name of the Azure resource group that contains your Azure Machine Learning service workspace.
2. Replace `myworkspace` with the name of your Azure Machine Learning service workspace.
3. When you're ready, select **Save and run**.
4. You're prompted to commit your changes to the `diabetes-train-and-deploy.yml` file in your repository. After you're happy with the message, select **Save and run** again.

If you want to watch your pipeline in action, select the build job.

You now have a YAML pipeline in your repository that's ready to train your model!

## Azure Machine Learning service automation

There are two primary ways to use automation with the Azure Machine Learning service:

- The [Machine Learning CLI](#) is an extension to the Azure CLI. It provides commands for working with the Azure Machine Learning service.
- The [Azure Machine Learning SDK](#) is Python package that provides programmatic access to the Azure Machine Learning service.
  - The Python SDK includes [automated machine learning](#) to help automating the time consuming, iterative tasks of machine learning model development.

The example with this document uses the Machine Learning CLI.

## Planning

Before you use Azure Pipelines to automate model training and deployment, you must understand the files needed by the model and what indicates a "good" trained model.

### Machine learning files

In most cases, your data science team will provide the files and resources needed to train the machine learning model. In the example project, data scientists would provide these files:

- **Training script** (`train.py`): The training script contains logic specific to the model that you're training.
- **Scoring file** (`score.py`): When the model is deployed as a web service, the scoring file receives data from clients and scores it against the model. The output is then returned to the client.
- **RunConfig settings** (`sklearn.runconfig`): Defines how the training script is run on the compute target that is used for training.
- **Training environment** (`myenv.yml`): Defines the packages needed to run the training script.
- **Deployment environment** (`deploymentConfig.yml`): Defines the resources and compute needed for the deployment environment.
- **Deployment environment** (`inferenceConfig.yml`): Defines the packages needed to run and score the model in the deployment environment.

Some of these files are directly used when developing a model. For example, the `train.py` and `score.py` files. However the data scientist may be programmatically creating the run configuration and environment settings. If

so, they can create the `.runconfig` and training environment files, by using `RunConfiguration.save()`. Or, default run configuration files can be created for all compute targets already in the workspace by running the following command:

```
az ml folder attach --experiment-name myexp -w myws -g mygroup
```

The files created by this command are stored in the `.azureml` directory.

### Determine the best model

The example pipeline deploys the trained model without doing any performance checks. In a production scenario, you may want to log metrics so that you can determine the "best" model.

For example, you have a model that is already deployed and has an accuracy of 90. You train a new model based on new checkins to the repo, and the accuracy is only 80, so you don't want to deploy it. You can use a metric such as this to build automation logic, as you can directly rank different models. In other cases, you may have several metrics that are used to indicate the "best" model. In this case, choosing the best model requires human judgment.

Depending on what "best" looks like for your scenario, you may need to create a [release pipeline](#) where someone must inspect the metrics to determine if the model should be deployed.

To log metrics during training, use the [Run](#) class.

## Azure CLI Deploy task

The [Azure CLI Deploy task](#) is used to run Azure CLI commands. In the example, it installs the Azure Machine Learning CLI extension and then uses individual CLI commands to train and deploy the model.

## Azure Service Connection

The [Azure CLI Deploy task](#) requires an Azure service connection. The Azure service connection stores the credentials needed to connect from Azure Pipelines to Azure.

The name of the connection used by the example is `azmldemows`.

To create a service connection, see [Create an Azure service connection](#).

## Machine Learning CLI

The following Azure Machine Learning service CLI commands are used in the example for this document:

| COMMAND                                 | PURPOSE                                                                                 |
|-----------------------------------------|-----------------------------------------------------------------------------------------|
| <code>az ml folder attach</code>        | Associates the files in the project with your Azure Machine Learning service workspace. |
| <code>az ml computetarget create</code> | Creates a compute target that is used to train the model.                               |
| <code>az ml experiment list</code>      | Lists experiments for your workspace.                                                   |
| <code>az ml run submit-script</code>    | Submits the model for training.                                                         |
| <code>az ml model register</code>       | Registers a trained model with your workspace.                                          |
| <code>az ml model deploy</code>         | Deploys the model as a web service.                                                     |

| COMMAND                    | PURPOSE                                 |
|----------------------------|-----------------------------------------|
| az ml service list         | Lists deployed services.                |
| az ml service delete       | Deletes a deployed service.             |
| az ml pipeline list        | Lists Azure Machine Learning pipelines. |
| az ml computetarget delete | Deletes a compute target.               |

For more information on these commands, see the [CLI extension reference](#).

## Next steps

Learn how you can further integrate machine learning into your pipelines with the [Machine Learning extension](#).

For more examples of using Azure Pipelines with Azure Machine Learning service, see the following repos:

- [MLOps \(CLI focused\)](#)
- [MLOps \(Python focused\)](#)

# Azure SQL database deployment

4/20/2021 • 8 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

You can automatically deploy your database updates to Azure SQL database after every successful build.

## 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](#)
- [Classic](#)

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>'
```

YAML pipelines aren't available in TFS.

See also [authentication information when using the Azure SQL Database Deployment task](#).

## 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 [main]
GO
IF NOT EXISTS (SELECT name FROM main.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.

## ARM

```
[CmdletBinding(DefaultParameterSetName = 'None')]
param
(
 [String] [Parameter(Mandatory = $true)] $ServerName,
 [String] [Parameter(Mandatory = $true)] $ResourceGroup,
 [String] $AzureFirewallName = "AzureWebAppFirewall"
)
$agentIP = (New-Object net.webclient).downloadstring("https://api.ipify.org")
New-AzSqlServerFirewallRule -ResourceGroupName $ResourceGroup -ServerName $ServerName -FirewallRuleName
$AzureFirewallName -StartIPAddress $agentIP -EndIPAddress $agentIP
```

## Classic

```
[CmdletBinding(DefaultParameterSetName = 'None')]
param
(
 [String] [Parameter(Mandatory = $true)] $ServerName,
 [String] [Parameter(Mandatory = $true)] $ResourceGroupName,
 [String] $AzureFirewallName = "AzureWebAppFirewall"
)

$ErrorActionPreference = 'Stop'

function New-AzureSqlServerFirewallRule {
 $agentIP = (New-Object net.webclient).downloadstring("https://api.ipify.org")
 New-AzureSqlDatabaseServerFirewallRule -StartIPAddress $agentIP -EndIPAddress $agentIP -FirewallRuleName
 $AzureFirewallName -ServerName $ServerName -ResourceGroupName $ResourceGroupName
}

function Update-AzureSqlServerFirewallRule{
 $agentIP= (New-Object net.webclient).downloadstring("https://api.ipify.org")
 Set-AzureSqlDatabaseServerFirewallRule -StartIPAddress $agentIP -EndIPAddress $agentIP -FirewallRuleName
 $AzureFirewallName -ServerName $ServerName -ResourceGroupName $ResourceGroupName
}

If ((Get-AzureSqlDatabaseServerFirewallRule -ServerName $ServerName -FirewallRuleName $AzureFirewallName -
ResourceGroupName $ResourceGroupName -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.

## ARM

```
[CmdletBinding(DefaultParameterSetName = 'None')]
param
(
 [String] [Parameter(Mandatory = $true)] $ServerName,
 [String] [Parameter(Mandatory = $true)] $ResourceGroup,
 [String] $AzureFirewallName = "AzureWebAppFirewall"
)
Remove-AzSqlServerFirewallRule -ServerName $ServerName -FirewallRuleName $AzureFirewallName -
ResourceGroupName $ResourceGroup
```

## Classic

```
[CmdletBinding(DefaultParameterSetName = 'None')]
param
(
 [String] [Parameter(Mandatory = $true)] $ServerName,
 [String] [Parameter(Mandatory = $true)] $ResourceGroupName,
 [String] $AzureFirewallName = "AzureWebAppFirewall"
)

$ErrorActionPreference = 'Stop'

If ((Get-AzureSqlDatabaseServerFirewallRule -ServerName $ServerName -FirewallRuleName $AzureFirewallName -ResourceGroupName $ResourceGroupName -ErrorAction SilentlyContinue))
{
 Remove-AzureSqlDatabaseServerFirewallRule -FirewallRuleName $AzureFirewallName -ServerName $ServerName -ResourceGroupName $ResourceGroupName
}
```

- [YAML](#)
- [Classic](#)

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 pipelines aren't available 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](#)
- [Classic](#)

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 main branch.

```

- task: SqlAzureDacpacDeployment@1
 condition: and(succeeded(), eq(variables['Build.SourceBranch'], 'refs/heads/main'))
 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 pipelines aren't available in 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 /?
```

# How-To: CI/CD with App Service and Azure Cosmos DB

5/4/2021 • 4 minutes to read • [Edit Online](#)

## Azure Pipelines

Create a continuous integration (CI) and continuous delivery (CD) pipeline for Azure Cosmos 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.

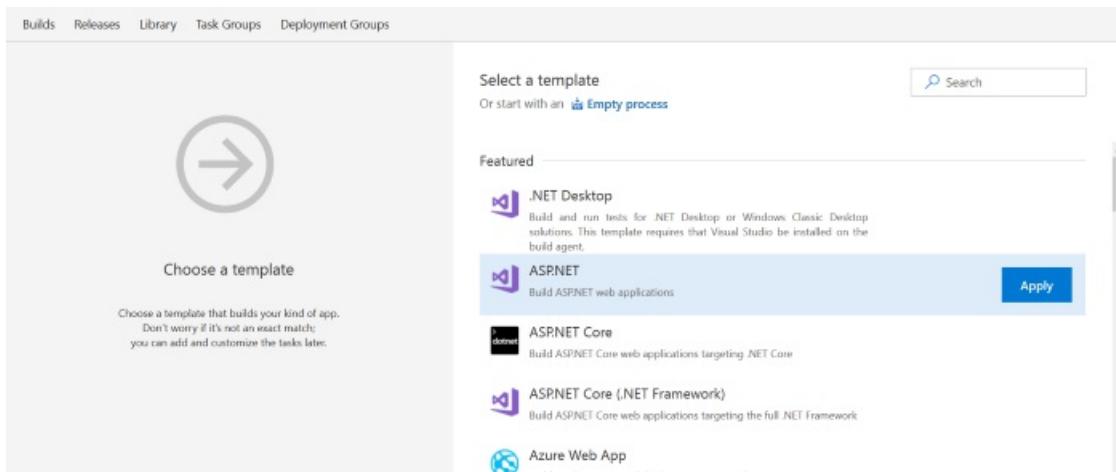
## Set up CI for your App

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**.



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

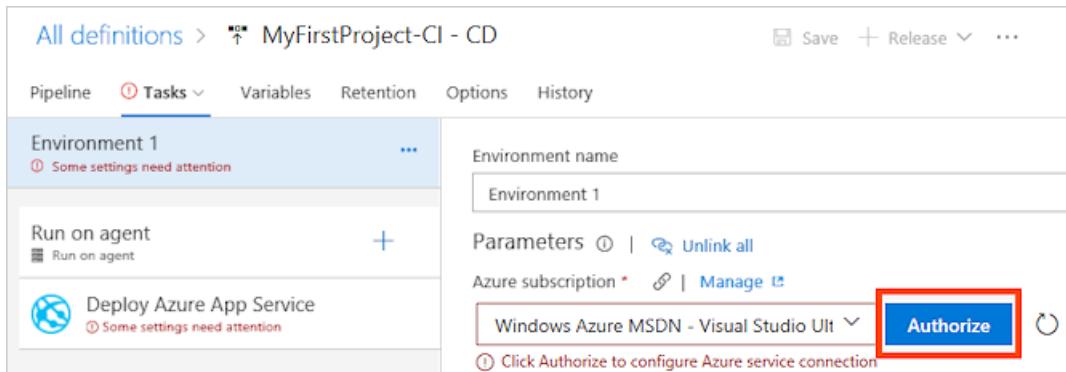
The CI for the sample app produces the artifacts needed for deployment to Azure. Follow the steps below to create a release pipeline which uses the CI artifacts for deploying a Cosmos DB instance.

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**.



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.

The screenshot shows the Azure portal's 'Keys' page for a Cosmos DB account. The 'contoso1 - Keys' section is visible. The 'Read-only Keys' tab is selected. The 'URI' field contains the URL 'https://contoso1.documents.azure.com:443/'. The 'PRIMARY KEY' field contains the placeholder '<primary key>' and features a padlock icon, indicating it is a secret. The 'SECONDARY KEY' field contains the placeholder '<secondary key>'. Below these fields are 'PRIMARY CONNECTION STRING' and 'SECONDARY CONNECTION STRING' fields, each containing a connection string with an account key. The 'Keys' link in the left sidebar is highlighted with a red box.

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 main 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](#)

# Tutorial: Deploy a Java app to a virtual machine scale set

4/9/2021 • 4 minutes to read • [Edit Online](#)

A [virtual machine scale set](#) lets you deploy and manage identical, autoscaling virtual machines.

VMs are created as needed in a scale set. You define rules to control how and when VMs are added or removed from the scale set. These rules can be triggered based on metrics such as CPU load, memory usage, or network traffic.

In this tutorial, you build a Java app and deploy it to a virtual machine scale set. You learn how to:

- Create a virtual machine scale set
- Build a machine image
- Deploy a custom image to a virtual machine scale set

## Prerequisites

Before you begin, you need:

- An Azure account with an active subscription. [Create an account for free](#).
- An active Azure DevOps organization. [Sign up for Azure Pipelines](#).
- The [Azure VM Image Builder DevOps task](#) installed for your DevOps organization.
- A forked GitHub repo with the example project. Fork the [pipelines-vmss repository](#).

## Set up your Java pipeline

1. Sign in to your Azure DevOps organization and navigate to your project.
2. Go to **Pipelines**, and then select **New Pipeline**.
3. Walk through the steps of the wizard by first selecting **GitHub** as the location of your source code.
4. You might be redirected to GitHub to sign in. If so, enter your GitHub credentials.
5. When the list of repositories appears, select your repository.
6. You might be redirected to GitHub to install the Azure Pipelines app. If so, select **Approve & install**.

When the **Configure** tab appears, select **Maven**.

## Customize the pipeline

1. When your new pipeline appears, take a look at the YAML to see what it does. When you're ready, select **Save and run**.

**New pipeline**

## Review your pipeline YAML

**Save and run**

### azure-pipelines.yml

2. You are prompted to commit a new `azure-pipelines.yml` file to your repository. After you're happy with the message, select **Save and run** again.

If you want to watch your pipeline in action, select the build job.

You just created and ran a pipeline that we automatically created for you, because your code appeared to be a good match for the [Maven](#) template.

You now have a working YAML pipeline (`azure-pipelines.yml`) in your repository that's ready for you to customize!

3. When you're ready to make changes to your pipeline, select it in the **Pipelines** page, and then **Edit the `azure-pipelines.yml` file**.

#### Add the Copy Files and Publish Build Artifact tasks

1. Update your pipeline to include the `CopyFiles@2` task. This will create an artifact that you can deploy to your virtual machine scale set.

```
trigger: none

pool:
 vmImage: 'ubuntu-latest'

steps:
- task: Maven@3
 inputs:
 mavenPomFile: 'pom.xml'
 mavenOptions: '-Xmx3072m'
 javaHomeOption: 'JDKVersion'
 jdkVersionOption: '1.8'
 jdkArchitectureOption: 'x64'
 publishJUnitResults: true
 testResultsFiles: '**/surefire-reports/TEST-*.xml'
 goals: 'package'

- task: CopyFiles@2
 displayName: 'Copy File to: $(TargetFolder)'
 inputs:
 SourceFolder: '$(Build.SourcesDirectory)'
 Contents: |
 **/*.sh
 **/*.war
 **/*jar-with-dependencies.jar
 TargetFolder: '$(System.DefaultWorkingDirectory)/pipeline-artifacts/'
 flattenFolders: true
```

## Create a custom image and upload it to Azure

You'll need a resource group, storage account, and shared image gallery for your custom image.

1. Create a resource group with `az group create`. This example creates a resource group named `myVMSSResourceGroup` in the `eastus2` location:

```
az group create --name myVMSSResourceGroup --location eastus2
```

2. Create a new storage account. This example creates a storage account, `vmsstorageaccount`.

```
az storage account create \
--name vmsstorageaccount \
--resource-group myVMSSResourceGroup \
--location eastus2 \
--sku Standard_LRS
```

3. Create a [shared image gallery](#).

```
az sig create --resource-group myVMSSResourceGroup --gallery-name myVMSSGallery
```

4. Create a new image gallery in the `myVMSSGallery` resource. See [Create an Azure Shared Image Gallery using the portal](#) to learn more about working with image galleries.

```
az sig create --resource-group myVMSSResourceGroup --gallery-name myVMSSGallery
```

5. Create an image definition. Copy the `id` of the new image that looks like

```
/subscriptions/<SUBSCRIPTION_ID>/resourceGroups/<RESOURCE GROUP>/providers/Microsoft.Compute/galleries/myVMSSGallery/images/MyImage
```

```
az sig image-definition create -g myVMSSResourceGroup --gallery-name myVMSSGallery --gallery-image-definition MyImage --publisher GreatPublisher --offer GreatOffer --sku GreatSku --os-type linux
```

## Create a managed identity

1. Create a [managed identity](#) in your resources group.

```
az identity create -g myVMSSResourceGroup -n myVMSSIdentity
```

2. From the output, copy the `id`. The `id` will look like

```
/subscriptions/<SUBSCRIPTION_ID>/resourcegroups/<RESOURCE GROUP>/providers/Microsoft.ManagedIdentity/userAssignedIdentities/<USER ASSIGNED IDENTITY NAME>
```

3. Open your image gallery in the gallery and assign `myVMSSIdentity` the Contributor role. Follow [these steps to add a role assignment](#).

## Create the custom image

To create a custom image, you can use the [Azure VM Image Builder DevOps Task](#).

1. Add the `AzureImageBuilderTask@1` task to your YAML file. Replace the values for `<SUBSCRIPTION_ID>`, `<RESOURCE GROUP>`, `<USER ASSIGNED IDENTITY NAME>` with your own. Make sure to verify that the `galleryImageId`, `managedIdentity` and `storageAccountName` values are accurate.

```

- task: AzureImageBuilderTask@1
 displayName: 'Azure VM Image Builder Task'
 inputs:
 managedIdentity: '/subscriptions/<SUBSCRIPTION ID>/resourcegroups/<RESOURCE GROUP>/providers/Microsoft.ManagedIdentity/userAssignedIdentities/<USER ASSIGNED IDENTITY NAME>'
 imageSource: 'marketplace'
 packagePath: '${System.DefaultWorkingDirectory}/pipeline-artifacts'
 inlineScript: |
 sudo mkdir /lib/buildArtifacts
 sudo cp "/tmp/pipeline-artifacts.tar.gz" /lib/buildArtifacts/.
 cd /lib/buildArtifacts.
 sudo tar -zxf pipeline-artifacts.tar.gz
 sudo sh install.sh
 storageAccountName: 'vmssstorageaccount2'
 distributeType: 'sig'
 galleryImageId: '/subscriptions/<SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP>/providers/Microsoft.Compute/galleries/myVMSSGallery/images/MyImage/versions/0.0.$(Build.BuildId)'
 replicationRegions: 'eastus2'
 ibSubscription: '<SUBSCRIPTION ID>'
 ibAzureResourceGroup: 'myVMSSResourceGroup'
 ibLocation: 'eastus2'

```

2. Run the pipeline to generate your first image. You may need to [authorize resources](#) during the pipeline run.
3. Go to the your new image in the Azure portal and open [Overview](#). Select [Create VMSS](#) to create a new virtual machine scale set from the new image. Set [Virtual machine scale set name](#) to `vmssScaleSet`. See [Create a virtual machine scale set in the Azure portal](#) to learn more about creating virtual machine scale sets in the Azure portal.

## Deploy updates to the virtual machine scale set

Add an Azure CLI task to your pipeline to deploy updates to the scale set. Add the task at the end of the pipeline. Replace `<SUBSCRIPTION ID>` with your subscription ID.

```

- task: AzureCLI@2
 inputs:
 azureSubscription: ``YOUR_SUBSCRIPTION_ID`` #Authorize and in the task editor
 ScriptType: 'pscore'
 scriptLocation: 'inlineScript'
 Inline: 'az vmss update --resource-group myVMSSResourceGroup --name vmssScaleSet --set
virtualMachineProfile.storageProfile.imageReference.id=/subscriptions/<SUBSCRIPTION ID>/resourceGroups/myVMSSResourceGroup/providers/Microsoft.Compute/galleries/myVMSSGallery/images/MyImage/versions/0.0.$(Build.BuildId)'

```

## Clean up resources

Go to the Azure portal and delete your resource group, `myVMSSResourceGroup`.

## Next steps

[Learn more about virtual machine scale sets](#)

# Deploy apps to Azure Government Cloud

2/26/2020 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2017 | TFS 2018

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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
- Get advice on [Stack Overflow](#), and feel free to post your questions, search for answers, or suggest a feature on our [Azure DevOps Developer Community Support](#) page.

# Deploy to Azure Stack Hub App Service using Azure Pipelines

3/17/2021 • 5 minutes to read • [Edit Online](#)

This article walks you through setting up a CI/CD pipeline for deploying an application to app services in an Azure Stack Hub instance using Azure Pipelines.

In this article you can learn to create or validate:

- Azure Stack Hub service principle (SPN) credentials for the pipeline.
- A web app in your Azure Stack Hub instance.
- A service connection to your Azure Stack Hub instance.
- A repo with your app code to deploy to your app

## Prerequisites

- Access to [Azure Stack Hub](#) instance with the App Service RP enabled.
- An [Azure DevOps solution](#) associated with your Azure Stack Hub tenant.

## Create or validate your SPN

An SPN provides role-based credentials so that processes outside of Azure can connect to and interact with resources. You will need an SPN with contributor access and the attributes specified in these instructions to use with your Azure DevOps pipeline.

As a user of Azure Stack Hub you do not have the permission to create the SPN. You will need to request this principle from your cloud operator. The instructions are being provided here so you can create the SPN if you are a cloud operator, or you can validate the SPN if you are a developer using an SPN in your workflow provided by a cloud operator.

The cloud operator will need to create the SPN using Azure CLI.

The following code snippets are written for a Windows machine using the PowerShell prompt with [Azure CLI for Azure Stack Hub](#). If you are using CLI on a Linux machine and bash, either remove the line extension or replace them with a `\`.

1. Prepare the values of the following parameters used to create the SPN:

| PARAMETER                                   | EXAMPLE                                                    | DESCRIPTION                               |
|---------------------------------------------|------------------------------------------------------------|-------------------------------------------|
| endpoint-resource-manager                   | "https://management.orlando.azurestack.corp.microsoft.com" | The resource management endpoint.         |
| suffix-storage-endpoint                     | "orlando.azurestack.corp.microsoft.com"                    | The endpoint suffix for storage accounts. |
| suffix-keyvault-dns                         | ".vault.orlando.azurestack.corp.microsoft.com"             | The Key Vault service dns suffix.         |
| endpoint-active-directory-graph-resource-id | "https://graph.windows.net/"                               | The Active Directory resource ID.         |

| PARAMETER               | EXAMPLE              | DESCRIPTION                                                          |
|-------------------------|----------------------|----------------------------------------------------------------------|
| endpoint-sql-management | https://notsupported | The sql server management endpoint. Set this to https://notsupported |
| profile                 | 2019-03-01-hybrid    | Profile to use for this cloud.                                       |

2. Open your command-line tool such as Windows PowerShell or Bash and sign in. Use the following command:

```
az login
```

3. Use the `register` command for a new environment or the `update` command if you are using an existing environment. Use the following command.

```
az cloud register `
 -n "AzureStackUser" `
 --endpoint-resource-manager "https://management.<local>.<FQDN>" `
 --suffix-storage-endpoint ".<local>.<FQDN>" `
 --suffix-keyvault-dns ".vault.<local>.<FQDN>" `
 --endpoint-active-directory-graph-resource-id "https://graph.windows.net/" `
 --endpoint-sql-management https://notsupported `
 --profile 2019-03-01-hybrid
```

4. Get your subscription ID and resource group that you want to use for the SPN.

5. Create the SPN with the following command with the subscription ID and resource group:

```
az ad sp create-for-rbac --name "myApp" --role contributor `
 --scopes /subscriptions/{subscription-id}/resourceGroups/{resource-group} `
 --sdk-auth
```

If you do not have cloud operator privileges, you can also sign in with the SPN provided to you by your cloud operator. You will need the client ID, the secret, and your tenant ID. With these values, you can use the following Azure CLI commands to create the JSON object that contains the values you will need to create your service connection.

```
az login --service-principal -u "<client-id>" -p "<secret>" --tenant "<tenant-ID>" --allow-no-
 subscriptions
az account show --sdk-auth
```

6. Check the resulting JSON object. You will use the JSON object to create your service connection. The JSON object should have the following attributes:

```
{
 "clientId": <Application ID for the SPN>,
 "clientSecret": <Client secret for the SPN>,
 "subscriptionId": <Subscription ID for the SPN>,
 "tenantId": <Tenant ID for the SPN>,
 "activeDirectoryEndpointUrl": "https://login.microsoftonline.com/",
 "resourceManagerEndpointUrl": "https://management.<FQDN>",
 "activeDirectoryGraphResourceId": "https://graph.windows.net/",
 "sqlManagementEndpointUrl": "https://notsupported",
 "galleryEndpointUrl": "https://providers.<FQDN>:30016/",
 "managementEndpointUrl": "https://management.<FQDN>"
}
```

## Create the web app target

1. Sign in to your Azure Stack Hub portal.
2. Select **Create a resource > Web + Mobile > Web App**.
3. Select your **Subscription**.
4. Create or select a **Resource Group**.
5. Type the **Name** of your app. The name of the app will appear in the URL for your app, for example,  
`yourappname.appservice.<region>.<FQDN>`
6. Select the **Runtime stack** for your app. The runtime must match the code you plan to use for your web app.
7. Select the **Operating System (OS)** that will host your runtime and app.
8. Select or type the **Region** for your Azure Stack Hub instance.
9. Select the plan based on your Azure Stack Hub instance, region, and app OS.
10. Select **Review + Create**.
11. Review your web app. Select **Create**.
12. Select **Go to resource**.
13. Make note of your app name. You will add the name to the yml document that defines your pipeline in your repository.

## Create an Azure DevOps instance

You will need an Azure DevOps instance associated with your Azure Stack Hub instance.

## Create a service connection

Create a service connection. You will need the values from your SPN and the name of your Azure Stack Hub subscription.

1. Sign in to [Azure DevOps](#).
2. Select **Service connections > New service connection**.
3. Select **Azure Resource Manager** and select **Next**.
4. Select **Service principal (manual)**.

**New Azure service connection**

Azure Resource Manager using service principal (manual)

**Environment**

Azure Stack

**Server Url**

**Scope Level**

Subscription  
 Management Group  
 Machine Learning Workspace

**Subscription Id**

Subscription Id from the [publish settings file](#)

**Subscription Name**

Subscription Name from the [publish settings file](#)

**Authentication**

**Service Principal Id**

Client Id for connecting to the endpoint. Refer to [Azure Service Principal link](#) on how to create Azure Service Principal.

**Credential**

Service principal key       Certificate

**Service principal key**

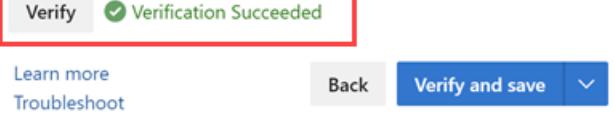
Service Principal Key for connecting to the endpoint. Refer to [Azure Service Principal link](#) on how to create Azure Service Principal. Ignore this field if the authentication type is spnCertificate.

**Tenant ID**

Tenant Id for connecting to the endpoint. Refer to [Azure Service Principal link](#) on how to create Azure Service Principal.

**Verify** **Verification Succeeded**

**Learn more** **Troubleshoot** **Back** **Verify and save**



5. Select Azure Stack for Environment.
6. Add the value for `managementEndpointUrl` for the Server URL.
7. Select Subscription for Scope Level.
8. Add `subscriptionId` and type the name.
9. Add the name of your subscription for Subscription Name.
10. Add `clientId` for Service Principal Id.
11. Select Service principal key.
12. Add `clientSecret` for the Service principal key.

13. Add `tenantId` for Tenant ID.

14. Select Verify.

15. Add the name of the connection and make a note of it. You will add the name of the connection to the yaml document that defines your pipeline in your repository.

## Create your repository and add pipeline

1. If you have not added your web app code to the repository, add it now.

2. Open the repository. Select the repo and select **Browse**.

3. Select **Pipelines**

4. Select **New pipeline**.

5. Select **Azure Repos Git**.

6. Select your repository.

7. Select **Starter pipeline**.

8. Navigate back to the repo and open the `azure-pipelines.yml`.

9. Add the following yaml:

```
Starter pipeline
Start with a minimal pipeline that you can customize to build and deploy your code.
Add steps that build, run tests, deploy, and more:
https://aka.ms/yaml
trigger:
- master
variables:
 azureSubscription: '<your connection name>'
 VSTS_ARM_REST_IGNORE_SSL_ERRORS: true
steps:
- task: AzureWebApp@1
 displayName: Azure Web App Deploy
 inputs:
 azureSubscription: $(azureSubscription)
 appName: <your-app-name>
 package: '$(System.DefaultWorkingDirectory)'
```

### NOTE

To ignore SSL errors, set a variable named `VSTS_ARM_REST_IGNORE_SSL_ERRORS` to the value `true` in the build or release pipeline, as in this example.

10. Update the `azureSubscription` value with the name of your service connection.

11. Update the `appName` with your app name. You are now ready to deploy.

## Notes about using Azure tasks with Azure Stack Hub

The following Azure tasks are validated with Azure Stack Hub:

- [Azure PowerShell](#)
- [Azure File Copy](#)
- [Azure Resource Group Deployment](#)

- [Azure App Service Deploy](#)
- [Azure App Service Manage](#)
- [Azure SQL Database Deployment](#)

## Next steps

- [Deploy an Azure Web App](#)
- [Troubleshoot Azure Resource Manager service connections](#)
- [Azure Stack Hub User Documentation](#)

# Build and push Docker images to Azure Container Registry

6/4/2021 • 2 minutes to read • [Edit Online](#)

In this step-by-step tutorial, you'll learn how to set up a continuous integration pipeline to build a containerized application. New pull requests trigger the pipeline to build and publish Docker images to Azure Container Registry.

## Prerequisites

- A GitHub account. [Create a free GitHub account](#), if you don't already have one.
- An Azure DevOps organization and a project. [Create a new organization](#) and/or a [new project](#), if you don't already have one.
- An Azure account. Sign up for a [free Azure account](#), if you don't already have one.

## Get the code

Fork or clone the *pipeline-javascript-docker* sample application:

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

## Create a container registry

1. Sign in to [Azure](#), and then select the **Azure Cloud Shell** button in the upper-right corner.



2. Run the following commands to create a resource group and an Azure Container Registry using the Azure CLI

```
Create a resource group
az group create --name myapp-rg --location eastus

Create a container registry
az acr create --resource-group myapp-rg --name myContainerRegistry --sku Basic
```

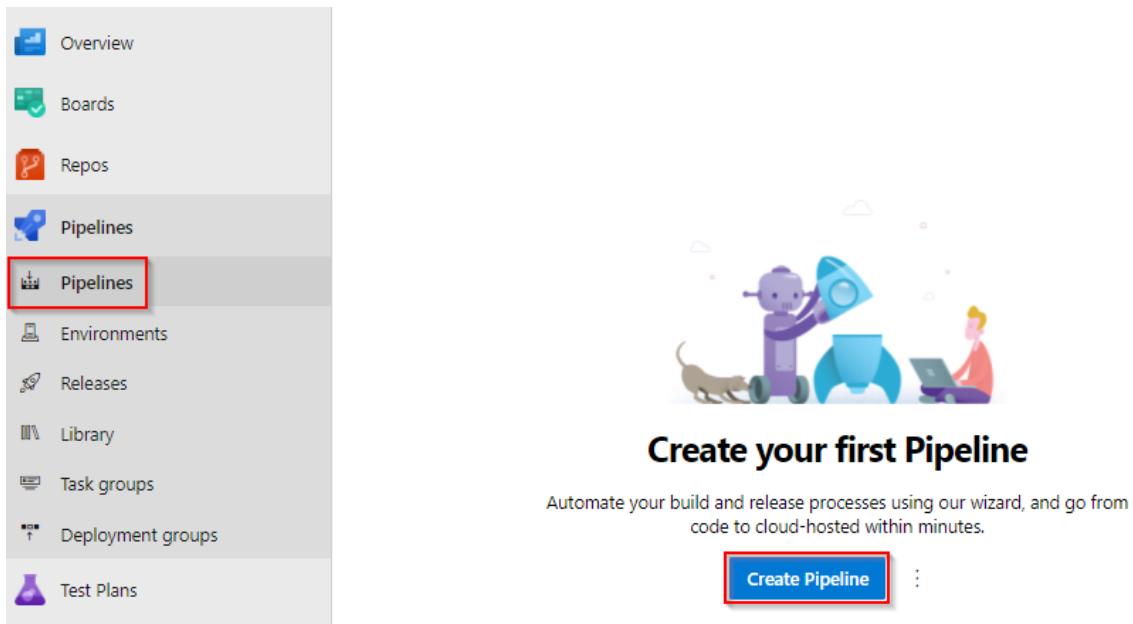
You can also use the Azure portal web UI to create your Azure Container Registry. See the [Create a container registry](#) for details.

### IMPORTANT

You must enable the admin user account in order for you to deploy a Docker image from an Azure Container Registry. See [Container registry authentication](#) for more details.

## Create the pipeline

1. Sign in to your Azure DevOps organization and navigate to your project.
2. Select **Pipelines**, and then select **New Pipeline** to create a new pipeline.



3. Select **GitHub YAML**, and then select **Authorize Azure Pipelines** to provide the appropriate permissions to access your repository.
4. You might be asked to sign in to GitHub. If so, enter your GitHub credentials, and then select your repository from the list of repositories.
5. From the **Configure** tab, select the **Docker - Build and push an image to Azure Container Registry** task.

✓ Connect

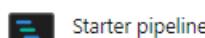
✓ Select

Configure

Review

New pipeline

## Configure your pipeline



Starter pipeline

Start with a minimal pipeline that you can customize to build and deploy your code.



Existing Azure Pipelines YAML file

Select an Azure Pipelines YAML file in any branch of the repository.



.NET Core Function App to Windows on Azure

Build a .NET Core function app and deploy it to Azure as a Windows function App.



.NET Desktop

Build and run tests for .NET Desktop or Windows classic desktop solutions.



Android

Build your Android project with Gradle.



Ant

Build your Java projects and run tests with Apache Ant.



ASP.NET

Build and test ASP.NET projects.



ASP.NET Core

Build and test ASP.NET Core projects targeting .NET Core.



ASP.NET Core (.NET Framework)

Build and test ASP.NET Core projects targeting the full .NET Framework.



C/C++ with GCC

Build your C/C++ project with GCC using make.



Deploy to Azure Kubernetes Service

Build and push image to Azure Container Registry; Deploy to Azure Kubernetes Service



Deploy to Kubernetes - Review app with Azure DevSpaces

Build and push image to Azure Container Registry; Deploy to Azure Kuberentes Services and setup Review App



Docker

Build a Docker image



Docker

Build and push an image to Azure Container Registry

6. Select your **Azure Subscription**, and then select your **Container registry** from the dropdown menu.

7. Provide an **Image Name** to your container image, and then select **Validate and configure**.

As Azure Pipelines creates your pipeline, it:

- Creates a *Docker registry service connection* to enable your pipeline to push images to your container registry.
- Generates an *azure-pipelines.yml* file, which defines your pipeline.

8. Review your pipeline YAML and select **Save and run** when you are ready.

The screenshot shows the 'Review' step of a pipeline creation process. At the top, there are four tabs: 'Connect', 'Select', 'Configure', and 'Review'. Below the tabs, it says 'New pipeline' and 'Review your pipeline YAML'. On the right, there are buttons for 'Variables' and 'Save and run'. A red box highlights the 'Save and run' button. The main area shows a code editor with the following YAML:

```
1 # Docker
2 # Build and push an image to Azure Container Registry
3 # https://docs.microsoft.com/azure/devops/pipelines/languages/docker
4
5 trigger:
6 - main
7
```

9. Add a **Commit message**, and then select **Save and run** to commit your changes and run your pipeline.
10. As your pipeline runs, select the build job to watch your pipeline in action.

The screenshot shows a 'Jobs' list. It has columns for 'Name', 'Status', and 'Duration'. There is one entry: 'Build' with a status of 'Queued'. A red arrow points to the 'Build' name.

| Name  | Status | Duration |
|-------|--------|----------|
| Build | Queued |          |

## How we build your pipeline

The pipeline that we just created in the previous section was generated from the *Docker container template* YAML. The build stage uses the Docker task `Docker@2` to build and push your Docker image to the container registry.

```
- stage: Build
 displayName: Build and push stage
 jobs:
 - job: Build
 displayName: Build job
 pool:
 vmImage: $(vmImageName)
 steps:
 - task: Docker@2
 displayName: Build and push an image to container registry
 inputs:
 command: buildAndPush
 repository: $(imageRepository)
 dockerfile: $(dockerfilePath)
 containerRegistry: $(dockerRegistryServiceConnection)
 tags: |
 $(tag)
```

## Clean up resources

If you are not going to continue to use this application, you can delete the resources you created in this tutorial to avoid incurring ongoing charges. Run the following to delete your resource group.

```
az group delete --name myapp-rg
```

## Related articles

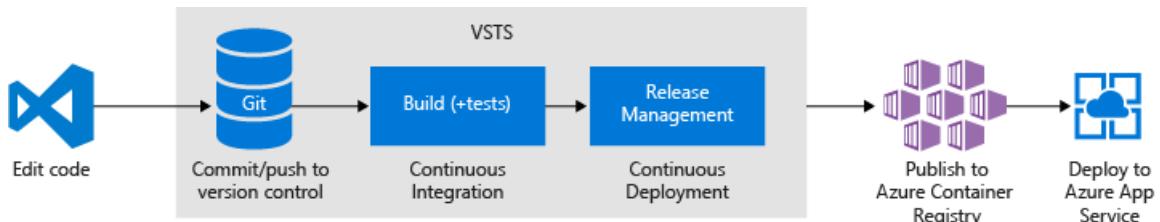
- [Deploy containerized ASP.NET Core apps](#)
- [Deploy to Azure Web App for Containers \(Classic\)](#)
- [Docker Content Trust](#)

# Deploy your ASP.NET Core app as a container

2/26/2020 • 4 minutes to read • [Edit Online](#)

## Azure Pipelines

Azure Pipelines provides a highly customizable continuous integration (CI) and continuous deployment (CD) pipeline for your ASP.NET Core apps. This quickstart shows how to set up CI and CD to deploy an ASP.NET Core application to Linux App Service using Docker and [Azure Container Registry](#).



With your CI/CD processes in place, you'll push a change into your team's git repo and the results will automatically show up on your site.

The screenshot shows a demo website for "dotnetcore\_sample". The top navigation bar includes links for "Home", "About", and "Contact". The main content area features a heading "Demo of CI/CD!!" above a large blue banner. The banner contains the text "ASP.NET Core" and "Windows", "Linux", "OSX" separated by vertical lines. Below this, there are arrows pointing left and right, and the text "Learn how to build ASP.NET apps that can run anywhere." with a "Learn More" button. At the bottom of the banner, there are three small circles. Below the banner, the page is divided into four sections: "Application uses", "How to", "Overview", and "Run & Deploy", each with a list of links.

| Application uses                                                                                                                                                     | How to                                                                                                                                                                                              | Overview                                                                                                                                                                                                    | Run & Deploy                                                                                                                                              |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"><li>Sample pages using ASP.NET Core MVC</li><li>Bower for managing client-side libraries</li><li>Theming using Bootstrap</li></ul> | <ul style="list-style-type: none"><li>Add a Controller and View</li><li>Manage User Secrets using Secret Manager.</li><li>Use logging to log a message.</li><li>Add packages using NuGet.</li></ul> | <ul style="list-style-type: none"><li>Conceptual overview of what is ASP.NET Core</li><li>Fundamentals of ASP.NET Core such as Startup and middleware.</li><li>Working with Data</li><li>Security</li></ul> | <ul style="list-style-type: none"><li>Run your app</li><li>Run tools such as EF migrations and more</li><li>Publish to Microsoft Azure Web Apps</li></ul> |

## Prerequisites

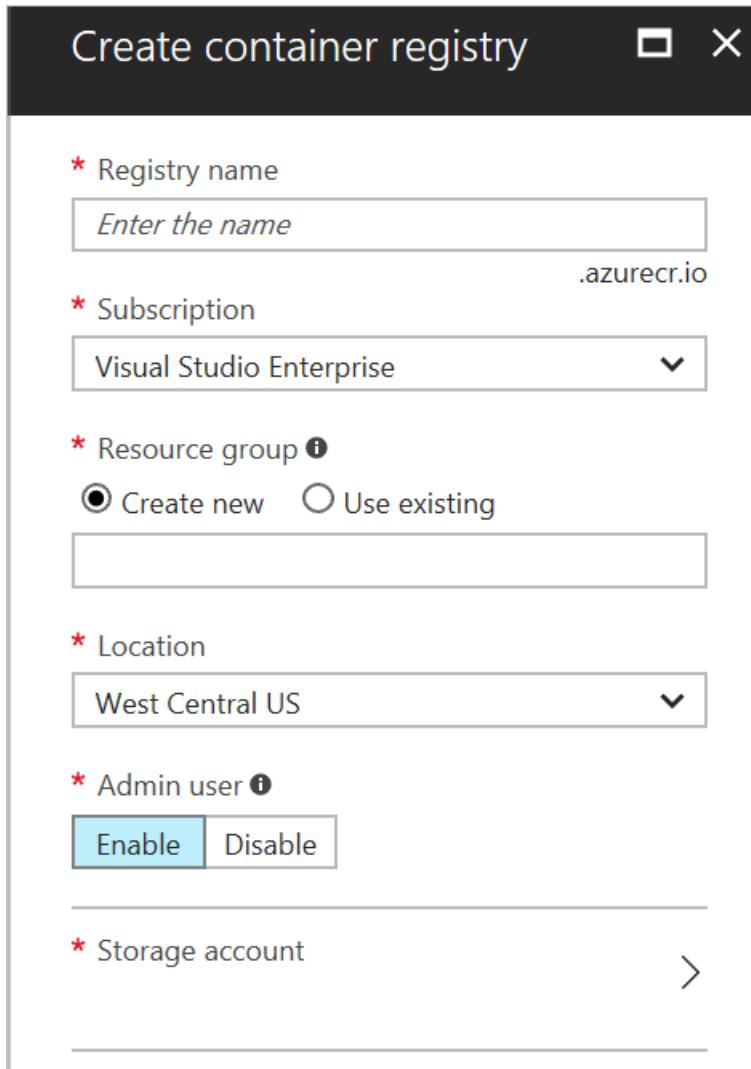
- You need an Azure DevOps organization. If you don't have one, you can [create one for free](#). If your team already has one, then make sure you're an administrator of the Azure DevOps project that you want to use. (An Azure DevOps organization is different from your GitHub organization. Give them the same name if you want alignment between them.)
- [Visual Studio 2017](#)
- [Continuous Delivery Tools for Visual Studio](#)

## Create an Azure Container Registry

You can use [Azure Container Registry](#) to host the Docker image that is published by the CI pipeline. Follow the

steps below to create and configure a registry. In later steps, you use Azure Pipelines to deploy the image to an Azure Web App for Containers.

1. Sign into Azure at <https://portal.azure.com>.
2. In the Azure Portal, choose **New, Containers**, then choose **Azure Container Registry**.
3. Enter a **Registry name**, **Resource Group**, and select a **Location**.



4. For **Admin user**, choose **Enable** and then choose **Create**.
5. Wait for the Azure Container Registry deployment to finish.

## Import code for sample app into Azure Repos

Azure Repos is a full-featured Git server for hosting your team's source code. You'll import code for a sample ASP.NET Core app into an Azure Repos Git repository. This is the app that you will configure CI/CD for.

1. In Azure Repos, on the **Code** page for your Azure DevOps project, select the option to **Import repository**.

The screenshot shows the GitHub interface with the 'Code' tab selected. In the top navigation bar, the 'Code' tab is highlighted with a red box. Below it, the 'MyFirstProject' dropdown menu is open, showing options like 'Files', 'History', 'Branches', 'Tags', and 'Pull Requests'. A red box highlights the 'Import repository' button in the sidebar, which is located under the 'All repositories' section. Other visible buttons include 'New repository', 'Manage repositories', and 'Clone in Visual Studio'.

2. In the **Import a Git repository** dialog box, paste the following URL into the **Clone URL** textbox.

```
https://github.com/MicrosoftDocs/pipelines-dotnet-core
```

3. Click **Import** to copy the sample code into your Git repo.

## Clone the solution

Use the following steps to clone the sample solution and open it in Visual Studio 2017. This allows you to work with the solution in a local development environment. In upcoming steps, you use Visual Studio 2017 to configure continuous integration and continuous delivery for the web app.

1. Ensure the import of the code completes successfully. In the top right-hand corner of your browser Select **clone**. Select **Clone in Visual Studio**.

The screenshot shows the 'Clone' dialog box from GitHub. At the top, there is a large 'Clone' button with a blue icon. Below it, there are several actions: 'Open in Visual Studio' (with a Microsoft icon), 'Edit' (with a pencil icon), 'Rename' (with a folder icon), a trash can icon, a download arrow icon, and a more options icon. The background of the dialog is white, and the overall design is clean and modern.

## Clone repo

Clone git repo using command line or IDE

Command line

HTTPS SSH

<https://devopstutorial.visualstudio.com/DevOpsDemos/>



**Generate Git credentials**

IDE

Clone in Visual Studio



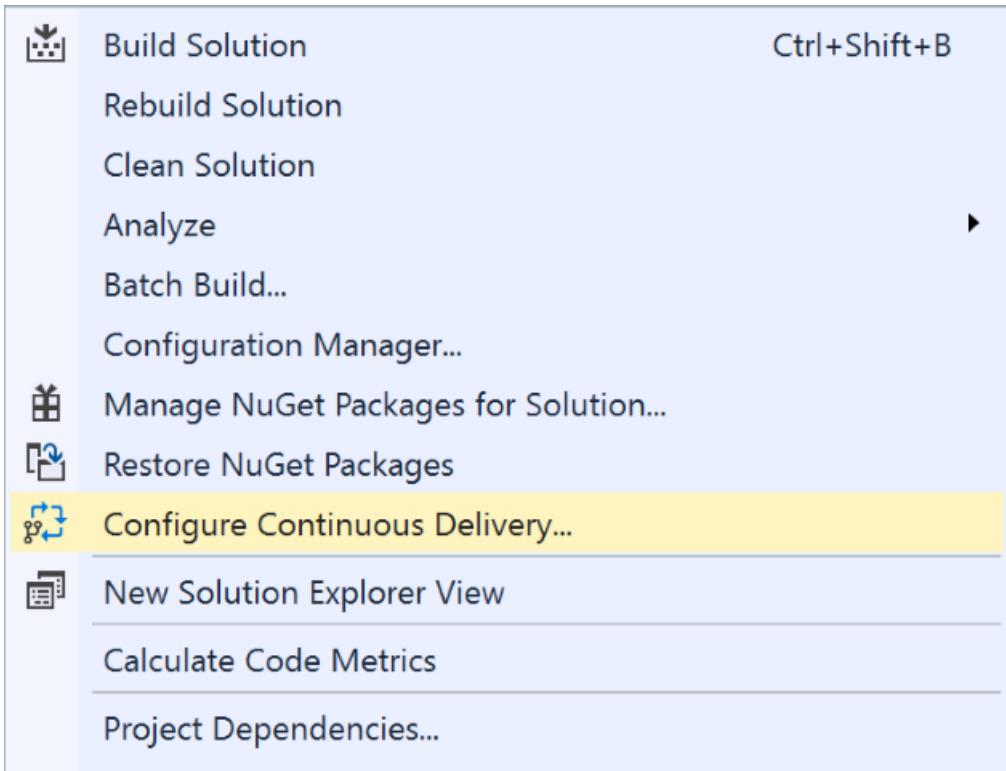
*(i) Having problems authenticating in Git? Be sure to get the latest version of [Git for Windows](#) or our plugins for [IntelliJ](#), [Eclipse](#), [Android Studio](#) or [Windows command line](#).*

2. Visual Studio 2017 will launch and prompt for credentials.
3. After authenticating, choose a directory and Select **clone** to finish cloning the code to your local environment.

## Install Continuous Delivery (CD) Tools for Visual Studio 2017

Continuous Delivery (CD) Tools for Visual Studio 2017 helps automate the creation of Azure Pipelines to Azure release pipelines. Optionally, you can extend the generated pipeline to further enhance your CI/CD workflows.

1. In Visual Studio 2017 Select **Tools** then **Select Extensions and Updates**.
2. In the dialogue, Select **Online** and search for **Continuous Delivery Tools for Visual Studio**.
3. Select **Download**, and once it finishes **close** Visual Studio to finish the install process.
4. **Restart** Visual Studio and **open** your solution.
5. In solution explorer **Right Select** your solution and verify you see a **Configure Continuous Delivery** icon.



## Configure and execute Continuous Delivery (CD) for your app

Visual Studio 2017 can automate the creation of a CI/CD pipeline. The CD tooling quickly creates various services in Azure and Azure Pipelines. This automation allows you to enable CI/CD to Azure for your apps by creating full build and release pipelines in Azure Pipelines.

1. In Visual Studio, right Select your solution and Select **Configure Continuous Delivery**.

### Configure Continuous Delivery

**Select a Repository Branch**

Repository: [dotnetcoredockerenabled](#)

Branch:

**Select Target Azure Resources**

Subscription:

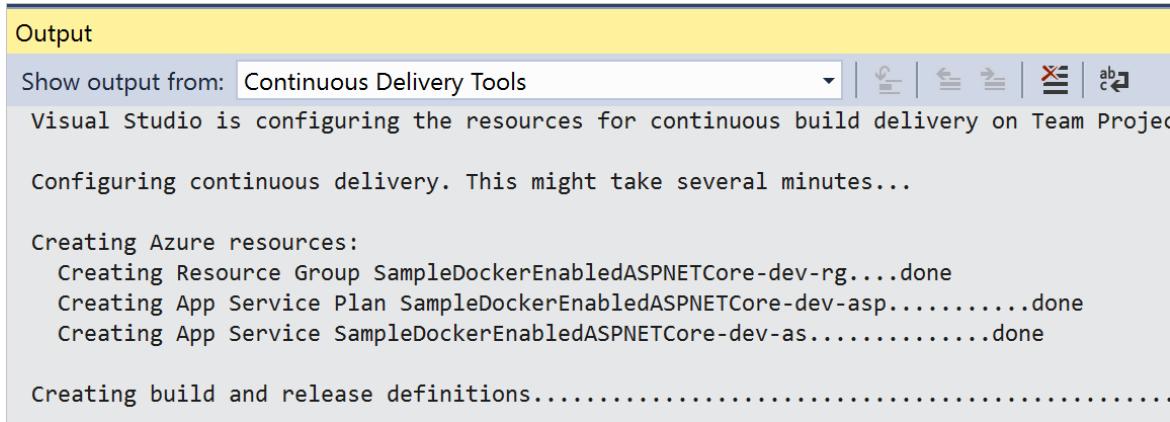
Host Type:

App Service:  [Edit...](#)

Container Registry:

2. Choose App Service (Linux) as your Host Type.

3. Choose the **Container Registry** you created earlier.
4. Accept the default App Service name or Select **edit** to choose a new name.
5. Select **ok**, and monitor the output window for results.
6. **Azure resources** including the App Service will be created. **Azure Pipelines resources** such as a build pipeline, and a release pipeline will be created in the same project that contains your source code. You can monitor the configuration progress in the **Output** window in Visual Studio.



The screenshot shows the Visual Studio Output window with a yellow header bar labeled "Output". Below the header, there is a dropdown menu set to "Continuous Delivery Tools". The main area displays deployment logs:

```
Show output from: Continuous Delivery Tools
Visual Studio is configuring the resources for continuous build delivery on Team Project
Configuring continuous delivery. This might take several minutes...
Creating Azure resources:
 Creating Resource Group SampleDockerEnabledASPNETCore-dev-rg.....done
 Creating App Service Plan SampleDockerEnabledASPNETCore-dev-asp.....done
 Creating App Service SampleDockerEnabledASPNETCore-dev-as.....done

Creating build and release definitions.....
```

7. A **build** and **release** will then execute. **Navigate** to the organization and monitor the builds and releases until they complete. This may take several minutes.
8. When the deployment finishes, verify that your changes are live in your web browser: <http://your-app-name.azurewebsites.net>.
9. Make changes to the web app code and push them to Azure Pipelines. A new build and release will execute and deploy the changes.

You're now ready to collaborate with a team on an ASP.NET Core app with a CI/CD pipeline that automatically deploys your latest work to your Azure App Service.

## Next Steps

You've just put your own CI/CD processes in place. You can modify these build and release pipelines to meet the needs of your team. To learn more see this tutorial:

- [Customize CD pipeline](#)

# Build and deploy to Azure Kubernetes Service

3/8/2021 • 6 minutes to read • [Edit Online](#)

## Azure Pipelines

Azure Kubernetes Service manages your hosted Kubernetes environment, making it quicker and easier for you to deploy and manage containerized applications. This service also eliminates the burden of ongoing operations and maintenance by provisioning, upgrading, and scaling resources on demand, without taking your applications offline.

In this step-by-step guide, you'll learn how to create a pipeline that continuously builds and deploys your app. Every time you change your code in a repository that contains a Dockerfile, the images are pushed to your Azure Container Registry, and the manifests are then deployed to your Azure Kubernetes Service cluster.

## Prerequisites

To ensure that your Azure DevOps project has the authorization required to access your Azure subscription, [create an Azure Resource Manager service connection](#). The service connection is required when you create a pipeline in the project to deploy to Azure Kubernetes Service. Otherwise, the drop-down lists for **Cluster** and **Container Registry** are empty.

- A GitHub account, where you can create a repository. If you don't have one, you can [create one for free](#).
- An Azure DevOps organization. If you don't have one, you can [create one for free](#). (An Azure DevOps organization is different from your GitHub organization. Give them the same name if you want alignment between them.)

If your team already has one, then make sure you're an administrator of the Azure DevOps project that you want to use.

- An ability to run pipelines on Microsoft-hosted agents. You can either purchase a [parallel job](#) or you can request a free tier. To request a free tier, follow the instructions in [this article](#). Please note that it may take us 2-3 business days to grant the free tier.
- An Azure account. If you don't have one, you can [create one for free](#).

### TIP

If you're new at this, the easiest way to get started is to use the same email address as the owner of both the Azure Pipelines organization and the Azure subscription.

## Get the code

Fork the following repository containing a sample application and a Dockerfile:

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

## Create the Azure resources

Sign in to the [Azure Portal](#), and then select the [Cloud Shell](#) button in the upper-right corner.

## Create a container registry

```
Create a resource group
az group create --name myapp-rg --location eastus

Create a container registry
az acr create --resource-group myapp-rg --name myContainerRegistry --sku Basic

Create a Kubernetes cluster
az aks create \
 --resource-group myapp-rg \
 --name myapp \
 --node-count 1 \
 --enable-addons monitoring \
 --generate-ssh-keys \
 --kubernetes-version 1.16.10
```

## Sign in to Azure Pipelines

Sign in to [Azure Pipelines](#). After you sign in, your browser goes to <https://dev.azure.com/my-organization-name> and displays your Azure DevOps dashboard.

Within your selected organization, create a *project*. If you don't have any projects in your organization, you see a **Create a project to get started** screen. Otherwise, select the **Create Project** button in the upper-right corner of the dashboard.

## Create the pipeline

### Connect and select repository

1. Sign in to your Azure DevOps organization and navigate to your project.
2. Go to **Pipelines**, and then select **New Pipeline**.
3. Walk through the steps of the wizard by first selecting **GitHub** as the location of your source code.
4. You might be redirected to GitHub to sign in. If so, enter your GitHub credentials.
5. When the list of repositories appears, select your repository.
6. You might be redirected to GitHub to install the Azure Pipelines app. If so, select **Approve & install**.

When the **Configure** tab appears, select **Deploy to Azure Kubernetes Service**.

1. If you are prompted, select the subscription in which you created your registry and cluster.
2. Select the `myapp` cluster.
3. For **Namespace**, select **Existing**, and then select **default**.
4. Select the name of your container registry.
5. You can leave the image name and the service port set to the defaults.
6. Set the **Enable Review App for Pull Requests** checkbox for [review app](#) related configuration to be included in the pipeline YAML auto-generated in subsequent steps.
7. Select **Validate and configure**.

As Azure Pipelines creates your pipeline, it:

- Creates a *Docker registry service connection* to enable your pipeline to push images into your

container registry.

- Creates an *environment* and a Kubernetes resource within the environment. For an RBAC enabled cluster, the created Kubernetes resource implicitly creates ServiceAccount and RoleBinding objects in the cluster so that the created ServiceAccount can't perform operations outside the chosen namespace.
  - Generates an *azure-pipelines.yml* file, which defines your pipeline.
  - Generates Kubernetes manifest files. These files are generated by hydrating the [deployment.yml](#) and [service.yml](#) templates based on selections you made above.
8. When your new pipeline appears, review the YAML to see what it does. For more information, see [how we build your pipeline](#) below. When you're ready, select **Save and run**.
9. The commit that will create your new pipeline appears. You can see the generated files mentioned above. Select **Save and run**.
10. If you want, change the **Commit message** to something like *Add pipeline to our repository*. When you're ready, select **Save and run** to commit the new pipeline into your repo, and then begin the first run of your new pipeline!

## See the pipeline run, and your app deployed

As your pipeline runs, watch as your build stage, and then your deployment stage, go from blue (running) to green (completed). You can select the stages and jobs to watch your pipeline in action.

### NOTE

If you're using a Microsoft-hosted agent, you must add the IP range of the Microsoft-hosted agent to your firewall. Get the weekly list of IP ranges from the [weekly JSON file](#), which is published every Wednesday. The new IP ranges become effective the following Monday. For more information, see [Microsoft-hosted agents](#). To find the IP ranges that are required for your Azure DevOps organization, learn how to [identify the possible IP ranges for Microsoft-hosted agents](#).

After the pipeline run is finished, explore what happened and then go see your app deployed. From the pipeline summary:

1. Select the **Environments** tab.
2. Select **View environment**.
3. Select the instance if your app for the namespace you deployed to. If you stuck to the defaults we mentioned above, then it will be the **myapp** app in the **default** namespace.
4. Select the **Services** tab.
5. Select and copy the external IP address to your clipboard.
6. Open a new browser tab or window and enter <IP address>:8080.

If you're building our sample app, then *Hello world* appears in your browser.

## How we build your pipeline

When you finished selecting options and then proceeded to validate and configure the pipeline (see above) Azure Pipelines created a pipeline for you, using the *Deploy to Azure Kubernetes Service* template.

The build stage uses the [Docker task](#) to build and push the image to the Azure Container Registry.

```
- stage: Build
displayName: Build stage
jobs:
- job: Build
 displayName: Build job
 pool:
 vmImage: $(vmImageName)
 steps:
- task: Docker@2
 displayName: Build and push an image to container registry
 inputs:
 command: buildAndPush
 repository: $(imageRepository)
 dockerfile: $(dockerfilePath)
 containerRegistry: $(dockerRegistryServiceConnection)
 tags: |
 $(tag)

- task: PublishPipelineArtifact@1
 inputs:
 artifactName: 'manifests'
 path: 'manifests'
```

The deployment job uses the *Kubernetes manifest task* to create the `imagePullSecret` required by Kubernetes cluster nodes to pull from the Azure Container Registry resource. Manifest files are then used by the Kubernetes manifest task to deploy to the Kubernetes cluster.

```
- stage: Deploy
displayName: Deploy stage
dependsOn: Build
jobs:
- deployment: Deploy
 displayName: Deploy job
 pool:
 vmImage: $(vmImageName)
 environment: 'azoooinmyluggagepipelinesjavascriptdocker.aksnamespace'
 strategy:
 runOnce:
 deploy:
 steps:
 - task: DownloadPipelineArtifact@2
 inputs:
 artifactName: 'manifests'
 downloadPath: '$(System.ArtifactsDirectory)/manifests'

 - task: KubernetesManifest@0
 displayName: Create imagePullSecret
 inputs:
 action: createSecret
 secretName: $(imagePullSecret)
 namespace: $(k8sNamespace)
 dockerRegistryEndpoint: $(dockerRegistryServiceConnection)

 - task: KubernetesManifest@0
 displayName: Deploy to Kubernetes cluster
 inputs:
 action: deploy
 namespace: $(k8sNamespace)
 manifests: |
 $(System.ArtifactsDirectory)/manifests/deployment.yml
 $(System.ArtifactsDirectory)/manifests/service.yml
 imagePullSecrets: |
 $(imagePullSecret)
 containers: |
 $(containerRegistry)}/${{imageRepository}}:${{tag}}
```

## Clean up resources

Whenever you're done with the resources you created above, you can use the following command to delete them:

```
az group delete --name myapp-rg
```

Type `y` when prompted.

```
az group delete --name MC_myapp-rg_myapp_eastus
```

Type `y` when prompted.

## Learn more

We invite you to learn more about:

- The services:
  - [Azure Kubernetes Service](#)
  - [Azure Container Registry](#)

- The template used to create your pipeline: [Deploy to existing Kubernetes cluster template](#)
- Some of the tasks used in your pipeline, and how you can customize them:
  - [Docker task](#)
  - [Kubernetes manifest task](#)
- Some of the key concepts for this kind of pipeline:
  - [Environments](#)
  - [Deployment jobs](#)
  - [Stages](#)
  - [Docker registry service connections](#) (the method your pipeline uses to connect to the service)

# Deploy a Docker container app to Azure Kubernetes Service

11/2/2020 • 7 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.

## Get the code

If you want some sample code that works with this guidance, [import](#) (into Azure DevOps), or fork (into GitHub), the following repository, based on the desired runtime.

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

If you already have an app in GitHub that you want to deploy, you can create a pipeline for that code.

If you are a new user, fork this repo in GitHub:

```
https://github.com/spring-guides/gs-spring-boot-docker.git
```

## Define your CI build process

Set up a CI pipeline for [building an image](#) and [pushing it to a container registry](#).

## Prerequisites

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

## Create an AKS cluster 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. Alternatively, if 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 `$(System.DefaultWorkingDirectory)/**/*.tgz`
  - **Release Name:** Enter a name for your release; for example `azureddevops`
  - **Recreate Pods:** Tick this checkbox if there is a configuration change during the release and you want to replace a running pod with the new configuration.
  - **Reset Values:** Tick this checkbox if you want the values built into the chart to override all values provided by the task.
  - **Force:** Tick this checkbox if, should conflicts occur, you want to upgrade and rollback to delete, recreate the resource, and reinstall the full release. This is useful in scenarios where applying patches can fail (for example, for services because the cluster IP address is immutable).
  - **Arguments:** Enter the Helm command arguments and their values; for this example  
`--set image.repository=$(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](#)

# Tutorial: Deploy to Kubernetes on Azure Container Service (AKS) with Jenkins CI and Azure Pipelines CD

11/2/2020 • 16 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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 main 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 **main** 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

```

- Select **Commit** and choose to commit these changes to the **main** 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.

- 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.
- Open your Jenkins account and select **Credentials**, **System**, and then choose **Add Domain**.
- Enter a **name**, **description**, and then select **Add hostname**.
- 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.
- Select **Ok**. You will see a message about empty credentials. Select the link for **adding some credentials**.
- 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:

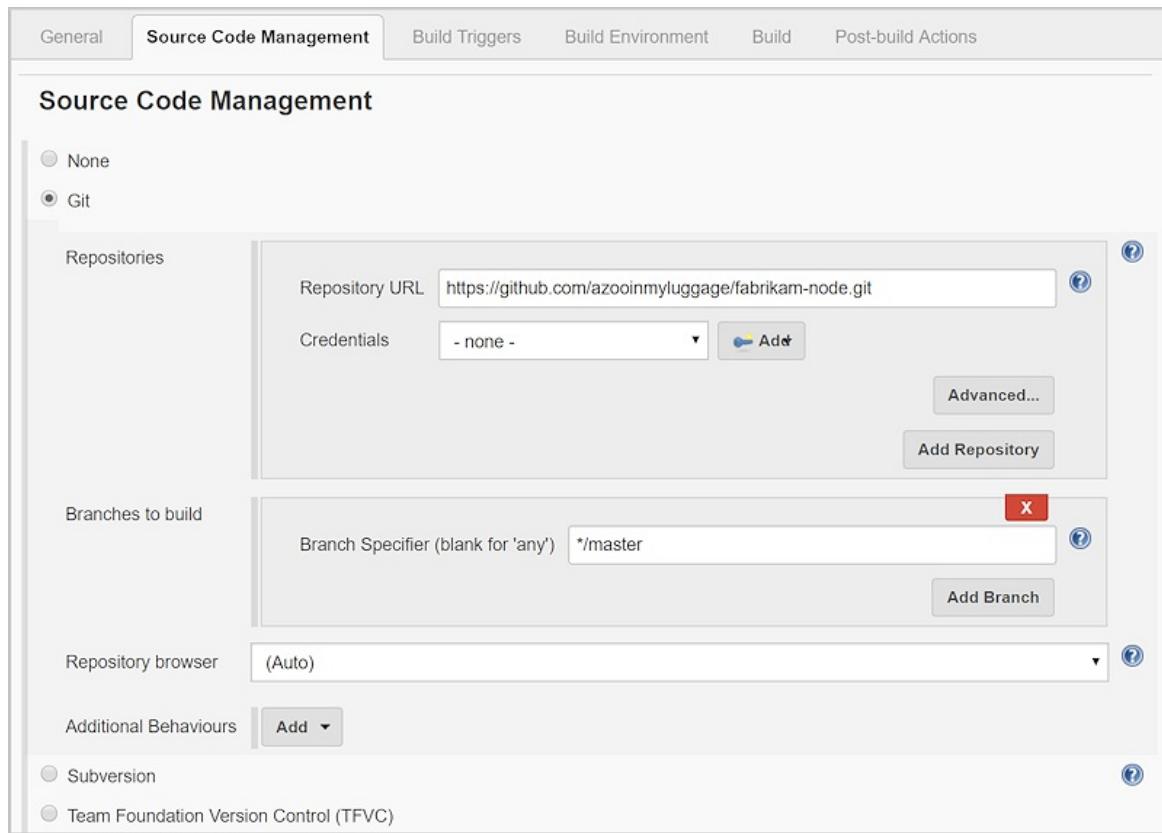
```
<server>
 <id>yourACRName.azurecr.io</id>
 <username>yourACRName</username>
 <password>e1M3Y3Z9vuUtusVx12+GoQvuAV3wsQFZ</password>
</server>
```

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)**.

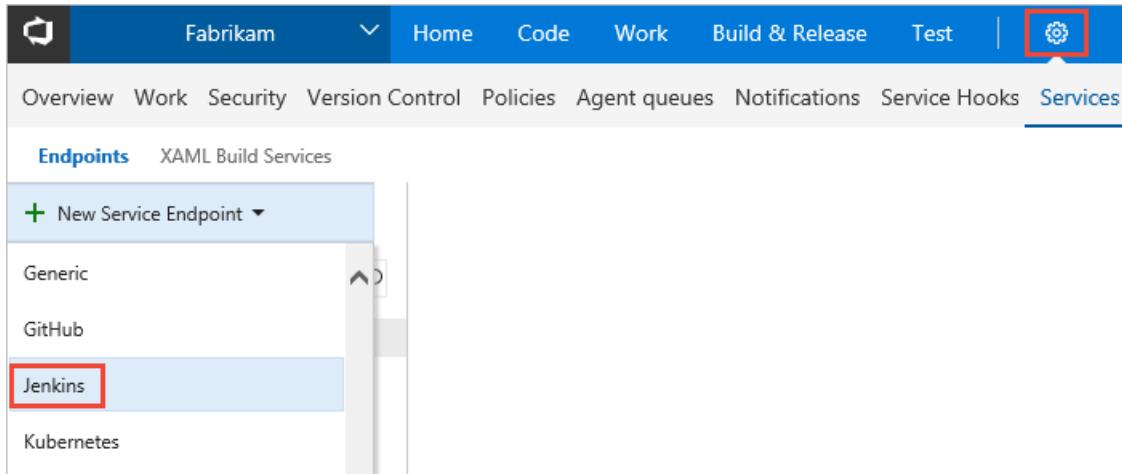


4. Select the **Credentials** drop down and choose the credential you created earlier. You should successfully authenticate to your Azure Repos repository and not receive errors before continuing. If you see errors, you likely have an issue with your credentials and Azure DevOps PAT.
5. Select the **Build 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 <https://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**.



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.

1. From the **Service Hooks** page in Azure Pipelines, Select the **+** to add a new service and choose **Jenkins**. Select **next**.
2. Choose **Code pushed** for the type of event trigger. Select your **code repository** that contains the sample application you imported earlier. Select **Next**.
3. Enter the URL of your Jenkins server. An example is below.  
`http://{YourJenkinsURL}.westcentralus.cloudapp.azure.com`
4. Enter the **user name and password** for your Jenkins account.
5. Select the Jenkins build you created earlier.
6. Choose **Test** to check that the information is correct.
7. Choose **Finish** to create the service connection.
8. 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.

1. 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.)
2. Search for the **Release Management Utility Tasks** extension provided by **Microsoft DevLabs**. The extension includes the **Tokenizer** utility.
3. Select the **Get it free** button.
4. Select your Azure DevOps organization and select the **Install** button to install the extension.
5. 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

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**.

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.

1. Next to the **1 job, 0 stages** link, select the **+** on the **Agent Job** to add a task to the job.

2. 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.

3. 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.yml** 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 main 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 main 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 main.
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 --no-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](#)

# Automatically deploy to IoT Edge devices

7/2/2021 • 7 minutes to read • [Edit Online](#)

In this tutorial, we will learn how to build an Azure Internet of Things (IoT) solution, push the created module images to Azure Container Registry (ACR), create a deployment manifest, and then deploy the modules to targeted IoT Edge devices.

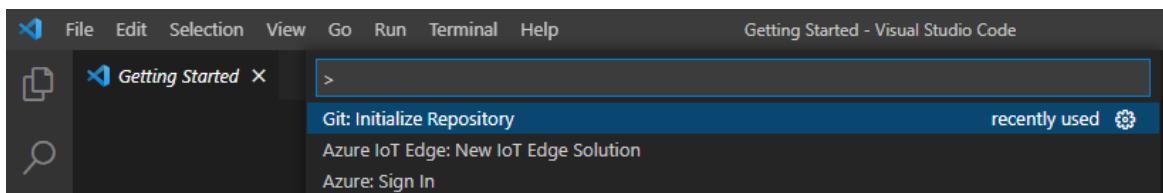
## Prerequisites

- An Azure DevOps organization and a project. [Create a new organization](#) and/or a [new project](#), if you don't already have one.
- An Azure account. Sign up for a [free Azure account](#), if you don't already have one.
- [Azure IoT tools](#) for Visual Studio Code.
- [Docker CE](#).
- An Azure Container Registry. Create an [Azure Container Registry](#), if you don't already have one.

## Create an IoT Edge project

The following steps creates an IoT Edge module project based on .NET Core SDK by using Visual Studio Code and Azure IoT tools extension.

1. In the Visual Studio Code, select **View**, and then select **Command Palette** to open the VS Code command palette.



2. In the command palette, enter and run the **Azure: Sign in** command and follow the instructions to sign in to your Azure account.
3. In the command palette, enter and run the **Azure IoT Edge: New IoT Edge solution** command and follow the prompts to create your solution.

FIELD	VALUES
Select Folder	Choose the local path on your machine where you want to create the solution files
Provide a solution name	Enter a descriptive name for your solution or accept the default EdgeSolution
Select module template	Select C# Module
Provide a module name	Provide a name for your module: E.g. CSharpModule

FIELD	VALUES
Provide Docker image repository for the module	An image repository includes the name of your container registry and the name of your container image. Your container image is pre-populated from the name that you have provided in the last step. Replace the <b>localhost:5000</b> with the login server value from your Azure container registry. You can retrieve the login server from the Overview page of your container registry in Azure portal.

Visual Studio code should load your IoT Edge solution workspace now. The solution workspace contains five top-level components:

- **modules** - contains the code for your module as well as the Dockerfiles to build your Docker image.
- **.env** - to store your container registry credentials
- **deployment.template.json** - a JSON file to build the deployment version of your modules
- **deployment.debug.template.json** - a JSON file to build the debug version of your modules
- **.vscode** and **.gitignore** - The workspace settings file and the gitignore to un-track files

#### NOTE

If you didn't specify a container registry when you created your solution and accepted the default localhost:5000 value, you won't have a .env file.

## Add registry credentials

The **.env** environment file stores the credentials for your container registry and shares them with the IoT Edge runtime. The runtime needs these credentials to pull your private images onto the IoT Edge device.

1. In the Visual Studio Code select the **.env** file to open it.
2. Update the fields with the user name and password values that you copied from your Azure container registry.
3. Save the file when you are done.

## Build your IoT Edge solution

Now that our IoT Edge solution is set up, let's build it now and generate our Docker image.

1. From the Visual Studio Code explorer, right-click on the **deployment.template.json** file and then select **Build IoT Edge solution**.
2. Upon successful build, you should see an image with the following format **registryname.azurecr.io/csharpmodule:0.0.1-amd64**. in your local Docker images.

## Push the code to Azure Repo

If your workspace isn't under Git source control, you can easily create a Git repository with the **Initialize Repository** command.

1. In the Visual Studio Code, select **View**, and then select **Command Palette** to open the VS Code command palette.

2. Enter and run the **Git: Initialize Repository** command from the Command Palette. Running Initialize Repository will create the necessary Git repository metadata files and show your workspace files as untracked changes ready to be staged.
3. Select **View > Terminal** to open the terminal window.
4. To push, pull, or sync your code, you must first set up a Git origin. To do so, run the following commands and enter your organization name, project name, and repository name in the placeholders.

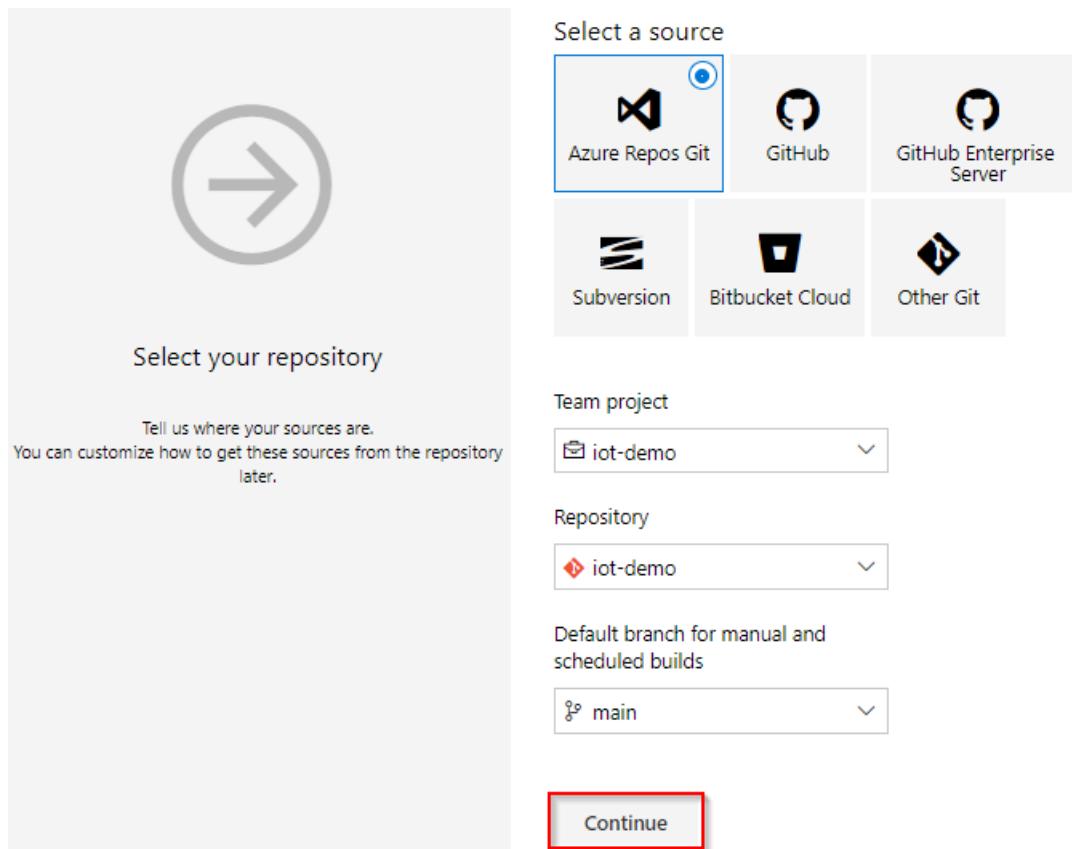
```
git remote add origin https://dev.azure.com/<org-name>/<project-name>/_git/<repo-name>
git push -u origin --all
```

Your code should be published now to your repository.

## Create a pipeline

You can use Azure Pipelines to build your 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.

1. From within your project, select **Pipelines > Pipelines**. Select **Create pipeline** to create a new one.
2. The default option for build pipelines involves using YAML to define the process. For this lab, select **use the classic editor**.
3. Select the **Source** and then select the **Team project**, **Repository**, and **Default branch**. Select **Continue** when you are done.



4. Select **Empty job**.

5. From the **Agent job 1**, select the **ubuntu-20.04** Agent Specification from the drop down.

6. Select **+** and search for **ARM template Deployment** task. Select **Add** and fill out the required fields as shown below. Select **Save & queue** when you are done.

FIELD	VALUES
Azure subscription	(Required) Name of Azure Resource Manager service connection
Action	(Required) Action to perform. Leave the default value as is
Resource group	(Required) Provide the name of a resource group
Location	(Required) Provide the location for deploying the resource group
Template location	(Required) Set the template location to URL of the file

FIELD	VALUES
Template link	(Required) <a href="https://raw.githubusercontent.com/Azure-Samples/devops-iot-scripts/12d60bd513ead7c94aa1669e505083beaf8a480/arm-acr.json">https://raw.githubusercontent.com/Azure-Samples/devops-iot-scripts/12d60bd513ead7c94aa1669e505083beaf8a480/arm-acr.json</a>
Override template parameters	-registryName YOUR_REGISTRY_NAME - registrySku Basic -registryLocation YOUR_REGISTRY_LOCATION

#### TIP

You must use quotation marks when overriding the template parameters to avoid getting the "One of the deployment parameters has an empty key..." error. Example: `-registryName "iotDemoContainerRegistry" -registrySku "Standard" -registryLocation "West US"`

- Select your pipeline, and then select **Edit** to edit your pipeline. Select **+** in the **Agent job 1**, and then search for the **Azure IoT Edge** task. Select **Add** to add the step to build the module images.
- Select **+** one more time, and search for the **Azure IoT Edge** task. Select **Add** and configure the task as shown below -

FIELD	VALUES
Action	Select an Azure IoT Edge action to <b>Push module images</b>
Container registry type	Select the Container registry type <b>Azure Container Registry</b>

FIELD	VALUES
Azure subscription	Select the Azure Resource Manager subscription for the deployment
Azure Container Registry	Select an Azure Container Registry from the dropdown which was created in the step 5

9. Select + and search for **Publish Build Artifacts** task. Select **Add** and then set the **Path to publish to** `$(Build.ArtifactStagingDirectory)/deployment.amd64.json`.
10. Select **Save & queue** when you are done.

## Create a release pipeline

Now that we built a Docker image and pushed it to Azure Container Registry, we must first create an IoT hub and an IoT Edge device in that hub. We will then deploy the sample module, and provision a virtual machine to run as our IoT Edge device.

1. [Create an IoT hub using the Azure portal](#)
2. [Register an IoT Edge device in IoT Hub](#)
3. From within your project, navigate to the **Pipelines > Releases**.
4. Select **New pipeline** to create a new release pipeline.
5. Select **Empty job** to start with an empty pipeline.
6. Select **Tasks** and then select + in the **Agent job**. Search for the **ARM template deployment** task. and select **Add**. Configure the task as shown below:

FIELD	VALUES
Azure subscription	(Required) Name of Azure Resource Manager service connection
Action	(Required) Action to perform. Leave the default value as is
Resource group	(Required) Provide the name of a resource group
Location	(Required) Provide the location for deploying the resource group
Template location	(Required) Set the template location to <b>URL of the file</b>
Template link	(Required) <a href="https://raw.githubusercontent.com/Azure-Samples/devops-iot-scripts/12d60bd513ead7c94aa1669e505083beaef8a480/arm-iothub.json">https://raw.githubusercontent.com/Azure-Samples/devops-iot-scripts/12d60bd513ead7c94aa1669e505083beaef8a480/arm-iothub.json</a>
Override template parameters	<code>-iotHubName "Your_IoT_HubName" -iotHubSku "F1"</code>

7. Select + and search for the **Azure IoT Edge** task. Select **Add** and configure the task as shown below. This step will Deploy the module images to IoT Edge device.

FIELD	VALUES
Action	Select an Azure IoT Edge action to <b>Deploy to IoT Edge devices</b>
Deployment file	<code>\$(System.DefaultWorkingDirectory)/*.json</code>
Azure subscription contains IoT Hub	Select an Azure subscription that contains IoT Hub
IoT Hub name	Select the IoT Hub
Choose single/multiple device	Select Single Device
IoT Edge device ID	Input the IoT Edge device ID

8. Select **+** and search for **Azure Resource Group Deployment** task. Select **Add** and configure the task as shown below.

FIELD	VALUES
Azure subscription	(Required) Name of <a href="#">Azure Resource Manager service connection</a>
Action	(Required) Action to perform. Leave the default value as is
Resource group	(Required) Provide the name of a resource group
Location	(Required) Provide the location for deploying the resource group
Template location	(Required) Set the template location to <a href="#">URL of the file</a>
Template link	(Required) <a href="https://raw.githubusercontent.com/Azure-Samples/devops-iot-scripts/12d60bd513ead7c94aa1669e505083beaf8a480/arm-linux-vm.json">https://raw.githubusercontent.com/Azure-Samples/devops-iot-scripts/12d60bd513ead7c94aa1669e505083beaf8a480/arm-linux-vm.json</a>
Override template parameters	<code>-edgeDeviceConnectionString \$(CS_OUTPUT) -virtualMachineName "YOUR_VM_NAME" -adminUsername "devops" -adminPassword "\$(vmPassword)" -appInsightsLocation "" -virtualMachineSize "Standard_A0" -location "YOUR_LOCATION"</code>

9. Once the release is complete, go to IoT hub in the Azure portal to view more information about your latest deployment.

## Related article

- [Deploy a web app to Azure App Services](#)
- [Deploy to Azure Web App for Containers](#)
- [Deploy a Docker container app to Azure Kubernetes Service](#)

# Check policy compliance with gates

4/20/2021 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

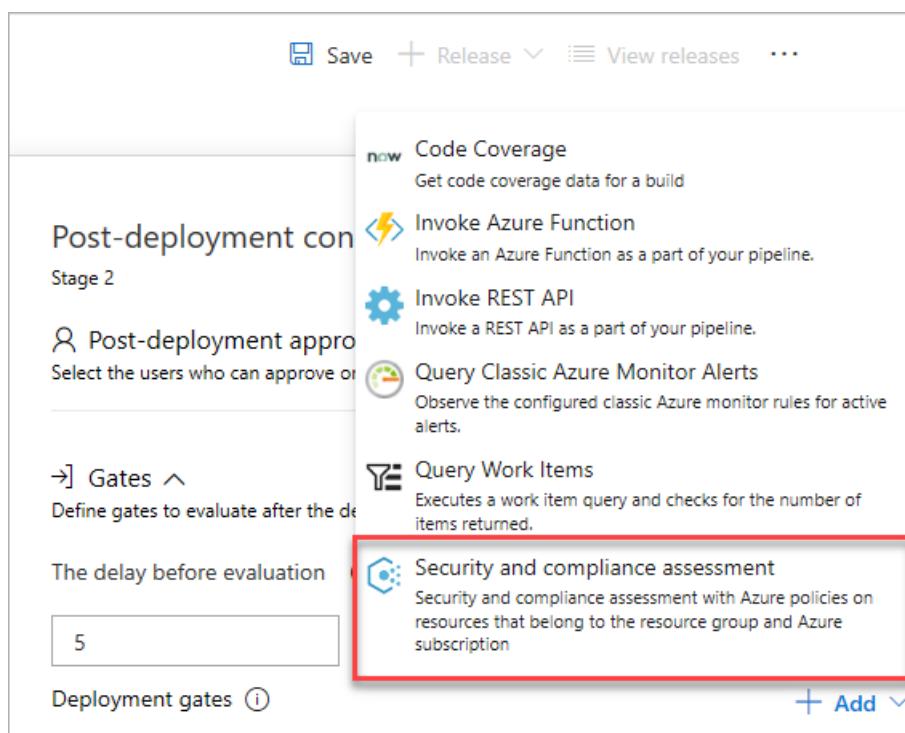
Azure Policy helps you manage and prevent IT issues by using policy definitions that enforce rules and effects for your resources. When you use Azure Policy, resources stay compliant with your corporate standards and service level agreements. Policies can be applied to an entire subscription, a management group, or a resource group.

This tutorial guides you in enforcing compliance policies on your resources before and after deployment during the release process through Azure Pipelines.

For more information, see [What is Azure Policy?](#) and [Create and manage policies to enforce compliance](#).

## Prepare

1. Create an [Azure Policy](#) in the Azure portal. There are several [pre-defined sample policies](#) that can be applied to a management group, subscription, and resource group.
2. In Azure DevOps create a release pipeline that contains at least one stage, or open an existing release pipeline.
3. Add a pre- or post-deployment condition that includes the **Security and compliance assessment** task as a gate. [More details](#).



## Validate for any violation(s) during a release

1. Navigate to your team project in Azure DevOps.
2. In the **Pipelines** section, open the **Releases** page and create a new release.
3. Choose the **In progress** link in the release view to open the live logs page.

4. When the release is in progress and attempts to perform an action disallowed by the defined policy, the deployment is marked as **Failed**. The error message contains a link to view the policy violations.

The screenshot shows the 'Stages' section of the Azure Pipelines interface. A stage named 'Dev servers' is highlighted with a red border and has a red 'Failed' status indicator. Below it, a deployment task is listed: 'Deploy Azure App Service t...' on 18/09/2018, 15:04. To the right, the 'Dev servers' summary page is shown, which also indicates a 'Failed' status. It displays a deployment log entry: 'Deployment failed at 19:00, 18/8/17' with '3 issues (2 errors)'. One of the errors is highlighted with a red box and points to a link: 'Request disallowed by Azure Policy. Click here for more information on Azure Portal'.

5. 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' logs for a specific job. The logs list several successful steps: 'Initialize Agent', 'Initialize Job', 'Download Artifacts', and 'Download artifact - html-website-static-CI'. However, a step titled 'Azure Deployment>Create Or Update Resource Group action on aloka' failed with 4 errors. The error details are expanded, showing a policy violation message: 'The template deployment failed because of policy violation. Please see details for more information.' followed by a detailed JSON log entry about a 'RequestDisallowedByPolicy' error.

6. When the policy compliance gate passes the release, a **Succeeded** status is displayed.

The screenshot shows the 'Stages' section of the Azure Pipelines interface. A stage named 'Dev' is shown with a green 'Succeeded' status indicator. Below it, a deployment task is listed: 'Dev' on 3/18/2019, 3:26 PM. To the right, the 'Gates' tab of the 'Dev' summary page is shown, indicating that all gates succeeded at 3/18/2019, 3:26 PM. A table below shows the results of a 'Deployment gates \ samples' check, with one row for 'Azure Policy compliance assessment' where the status is marked as 'Succeeded' with a green checkmark.

7. Choose the successful deployment to view the detailed logs.

The screenshot shows the Azure DevOps Pipelines interface. On the left, there's a sidebar with various icons for different pipeline stages like 'Deployment process', 'Run on agent', and 'Post-deployment gates'. The main area displays a pipeline run titled 'Azure.Policy > Release'. The tasks listed are 'Deployment process' (Succeeded), 'Run on agent' (Succeeded), and 'Post-deployment gates' (Succeeded). The 'Post-deployment gates' task is currently selected. To the right of the tasks, the logs for the 'Azure Policy compliance assessment' task are displayed. The logs show the execution of an expression to check if a URL is valid ('isUrl') and then evaluate the response code ('statuscode'). The output indicates that the URL is valid and the response code is OK.

```
1 | Parsing expression: <isUrl(variables['locationUrl'])>
2 | isUrl
3 | (
4 | ...variables
5 | ...
6 | ...
7 | 'locationUrl'
8 | ...
9 |)
10 | Evaluating: isUrl(variables['locationUrl'])
11 | Evaluating isUrl:
12 | ..Evaluating indexer:
13 | Evaluating variables:
14 | => Object
15 | Evaluating String:
16 | => 'locationUrl'
17 | ...=> "https://management.azure.com/subscriptions/afc112">
18 | => True
19 | Expanded: isUrl('https://management.azure.com/subscriptions/afc112')
20 | Result: True
21 |
22 | 2019-03-18T09:56:26.210673Z GET https://management.azure.com/subscriptions/afc112
23 | Response Code: OK
24 | Response: {
25 | "status": "Succeeded"
26 | }
27 | Parsing expression: <and(eq(response['statuscode'], '00'))>
28 | and
29 | (
30 | ...
31 | ...
32 | ...
33 | response
34 | ['statuscode']
```

## Help and support

- See our [troubleshooting](#) page
- Get advice on [Stack Overflow](#), and feel free to post your questions, search for answers, or suggest a feature on our [Azure DevOps Developer Community](#). [Support](#) page.

# Quickstart: Build an image

4/20/2021 • 4 minutes to read • [Edit Online](#)

## Azure Pipelines

Get started with container images by using Azure Pipelines to build an image. All you need to build an image is a Dockerfile in your repository.

You can build both Linux and Windows containers depending on what agent you use in your pipeline. Once you build an image, you can then push it to Azure Container Registry, Docker Hub, and Google Container registry. See [push an image](#) to learn more about pushing images to container registries.

## Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- A GitHub account. If you don't have one, sign up for [free](#).
- A GitHub repository with a Dockerfile. If you do not have a repository to use, fork this repository that contains a sample application and a Dockerfile:

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

## Build a Linux or Windows image

1. Sign in to your Azure DevOps organization and navigate to your project.
2. Go to **Pipelines**, and then select **New Pipeline**.
3. Select **GitHub** as the location of your source code and select your repository.

### NOTE

You might be redirected to GitHub to sign in. If so, enter your GitHub credentials. You might be redirected to GitHub to install the Azure Pipelines app. If so, select **Approve and install**.

4. Select **Starter pipeline**. Replace the contents of azure-pipelines.yml with this code. If you are building a Linux app, use `ubuntu-1604` for your `vmImage`. You can use `windows-latest` for your `vmImage` for Windows.

```

trigger:
- main

pool:
 vmImage: 'ubuntu-1604' # set to windows-latest or another Windows vmImage for Windows
builds

variables:
 imageName: 'pipelines-javascript-docker'

steps:
- task: Docker@2
 displayName: Build an image
 inputs:
 repository: $(imageName)
 command: build
 Dockerfile: app/Dockerfile

```

Windows container images can be built using either Microsoft hosted Windows agents or Windows platform based self-hosted agents (all Microsoft hosted Windows platform-based agents are shipped with Moby engine and client needed for Docker builds). Linux container images can be built using Microsoft hosted Ubuntu-16.04 agents or Linux platform based self-hosted agents. Learn more about the Windows and Linux agent options available with [Microsoft hosted agents](#).

#### **NOTE**

Currently the Microsoft hosted MacOS agents can't be used to build container images as the Moby engine needed for building the images is not pre-installed on these agents.

5. Select **Save and run**. You'll see a prompt to add a commit message when adding `azure-pipelines.yml` to your repository. Edit the message and then select **Save and run** again to see the pipeline in action.

#### **TIP**

Learn more about how to push the image to [Azure Container Registry](#) or [push it other container registries](#) such as Google Container Registry or Docker Hub. Learn more about the [Docker task](#) used in the above sample. Instead of using the recommended Docker task, it is also possible to invoke docker commands directly using a [command line task\(script\)](#)

## Clean up resources

If you're not going to continue to use this application, delete your pipeline and code repository.

## FAQ

### **What pre-cached images are available on hosted agents?**

Some commonly used images are pre-cached on Microsoft-hosted agents to avoid long time intervals spent in pulling these images from the container registry for every job. The list of pre-cached images is available in the [release notes of azure-pipelines-image-generation](#) repository.

### **How do I set the BuildKit variable for my docker builds?**

[BuildKit](#) introduces build improvements in the areas of performance, storage management, feature functionality, and security. To enable BuildKit based docker builds, set the DOCKER\_BUILDKIT variable.

```
trigger:
- main

pool:
 vmImage: 'ubuntu-latest'

variables:
 imageName: 'pipelines-javascript-docker'
 DOCKER_BUILDKIT: 1

steps:
- task: Docker@2
 displayName: Build an image
 inputs:
 repository: $(imageName)
 command: build
 Dockerfile: app/Dockerfile
```

#### NOTE

BuildKit is not currently supported on Windows hosts.

### How can I use a self-hosted agent?

You need to have Docker installed on your self-hosted machine before creating a container image. Add the [Docker installer task](#) in your pipeline before to the [Docker task](#) that builds your image.

### How can I create a script-based Docker build instead of using the docker task?

You can use the command `build` (or any other Docker command).

```
docker build -f Dockerfile -t foobar.azurecr.io/hello:world .
```

This command creates an equivalent image to one built with the Docker task. The Docker task itself internally calls the Docker binary on a script, and also stitches together a few more commands to provide a few more benefits. Learn more in the [Docker task documentation](#).

### Is reutilizing layer caching during builds possible on Azure Pipelines?

In the current design of Microsoft-hosted agents, every job is dispatched to a newly provisioned virtual machine (based on the image generated from [azure-pipelines-image-generation](#) repository templates). These virtual machines are cleaned up after the job reaches completion, not persisted and thus not reusable for subsequent jobs. The ephemeral nature of virtual machines prevents the reuse of cached Docker layers.

However, you can cache Docker layers with self-hosted agents because the ephemeral lifespan problem is not applicable for these agents.

### How to build Linux container images for architectures other than x64?

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 or ARM), you can use a machine emulator like [QEMU](#). The following steps illustrate how to create an ARM container image by using QEMU:

1. Author your Dockerfile with a base image that matches the target architecture:

```
FROM arm64v8/alpine:latest
```

2. Run the following script in your job before you build the image:

```
register QEMU binary - this can be done by running the following image
docker run --rm --privileged multiarch/qemu-user-static --reset -p yes
build your image
```

For more information, see [qemu-user-static](#) on GitHub.

### How to run tests and publish test results for containerized applications?

For different options on testing containerized applications and publishing the resulting test results, check out [Publish Test Results task](#).

## Next steps

Advance to the next article to learn how to push an image to a container registry.

[Push an image](#)

# Push an image

3/6/2021 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Azure Pipelines can be used to push images to container registries such as Azure Container Registry (ACR), Docker Hub, Google Container Registries, and others.

## Push step in pipeline

The following YAML snippet showcases the usage of [Docker registry service connection](#) along with a Docker task to login and push to a container registry. Instances of Docker registry service connection serve as secure options for storing credentials needed to login to the container registry before pushing the image. These service connections can be directly referenced in Docker task to login to the registry without the need to add a script task for docker login and setting up of secret variables for username and password.

```
- task: Docker@2
 displayName: Push image
 inputs:
 containerRegistry: |
 $(dockerHub)
 repository: $(imageName)
 command: push
 tags: |
 test1
 test2
```

## Azure Container Registry

Under Azure Container Registry option of [Docker registry service connection](#), the subscription (associated with the AAD identity of the user signed into Azure DevOps) and container registry within the subscription can be chosen to create the service connection. These service connections can be subsequently referenced from a pipeline task as shown in the YAML snippet above.

For creating a new pipeline for a repository containing Dockerfile, the [Build and Push to Azure Container Registry document](#) describes the Docker template automatically recommended by Azure Pipelines upon detecting of Dockerfile in the repository. The Azure subscription and Azure Container Registry inputs provided for template configuration are used by Azure Pipelines to automatically create the Docker registry service connection and even construct a functioning build and push pipeline by referencing the created service connection.

## Docker Hub

Choose the Docker Hub option under [Docker registry service connection](#) and provide the username and password required for verifying and creating the service connection.

## Google Container Registry

The following steps walk through the creation of Docker registry service connection associated with Google Container Registry:

1. Open your project in the GCP Console and then open Cloud Shell

2. To save time typing your project ID and Compute Engine zone options, set default configuration values by running the following commands:

```
gcloud config set project [PROJECT_NAME]
gcloud config set compute/zone [ZONE]
```

3. Replace `[PROJECT_NAME]` with the name of your GCP project and replace `[ZONE]` with the name of the zone that you're going to use for creating resources. If you're unsure about which zone to pick, use `us-central1-a`. For example:

```
gcloud config set project azure-pipelines-test-project-12345
gcloud config set compute/zone us-central1-a
```

4. Enable the Container Registry API for your project:

```
gcloud services enable containerregistry.googleapis.com
```

5. Create a service account for Azure Pipelines to publish Docker images:

```
gcloud iam service-accounts create azure-pipelines-publisher --display-name "Azure Pipelines Publisher"
```

6. Assign the Storage Admin IAM role to the service account:

```
PROJECT_NUMBER=$(gcloud projects describe \
$(gcloud config get-value core/project) \
--format='value(projectNumber)')

AZURE_PIPELINES_PUBLISHER=$(gcloud iam service-accounts list \
--filter="displayName:Azure Pipelines Publisher" \
--format='value(email)')

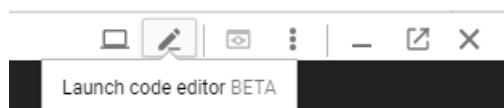
gcloud projects add-iam-policy-binding \
$(gcloud config get-value core/project) \
--member serviceAccount:$AZURE_PIPELINES_PUBLISHER \
--role roles/storage.admin
```

7. Generate a service account key:

```
gcloud iam service-accounts keys create \
azure-pipelines-publisher.json --iam-account $AZURE_PIPELINES_PUBLISHER

tr -d '\n' < azure-pipelines-publisher.json > azure-pipelines-publisher-oneline.json
```

Launch Code Editor by clicking the button in the upper-right corner of Cloud Shell:



8. Open the file named `azure-pipelines-publisher-oneline.json`. You'll need the content of this file in one of the following steps:
9. In your Azure DevOps organization, select **Project settings** and then select **Pipelines -> Service connections**.

10. Click **New service connection** and choose **Docker Registry**

11. In the dialog, enter values for the following fields:

- **Docker Registry:** `https://gcr.io/[PROJECT-ID]`, where `[PROJECT-ID]` is the name of your GCP project.
- **Docker ID:** `_json_key`
- **Docker Password:** Paste the contents of `azure-pipelines-publisher-oneline.json`
- **Service connection name:** `gcrServiceConnection`

12. Click **Save** to create the service connection

# Docker Content Trust

6/22/2021 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Docker Content Trust (DCT) provides the ability to use digital signatures for data sent to and received from remote Docker registries. These signatures allow client-side or runtime verification of the integrity and publisher of specific image tags.

### NOTE

A prerequisite for signing an image is a Docker Registry with a Notary server attached (Such as the [Docker Hub](#) or [Azure Container Registry](#))

## Signing images in Azure Pipelines

### Prerequisites on development machine

1. Use Docker trust's built in generator or manually generate delegation key pair. If the [built-in generator](#) is used, the delegation private key is imported into the local Docker trust store. Else, the private key will need to be manually imported into the local Docker trust store. See [Manually Generating Keys](#) for details.
2. Using the delegation key generated from the step above, upload the first key to a delegation and [initiate the repository](#)

### TIP

To view the list of local Delegation keys, use the Notary CLI to run the following command: `$ notary key list`.

### Set up pipeline for signing images

1. Fetch the delegation private key, which is present in the local Docker trust store of your development machine used earlier, and add the same as a [secure file](#) in Pipelines.
2. [Authorize this secure file](#) for use in all pipelines.
3. The service principal associated with `containerRegistryServiceConnection` must have the `AcrlImageSigner` role in the target container registry.
4. Create a pipeline based on the following YAML snippet:

```

pool:
 vmImage: 'Ubuntu 16.04'

variables:
 system.debug: true
 containerRegistryServiceConnection: serviceConnectionName
 imageRepository: foobar/content-trust
 tag: test

steps:
- task: Docker@2
 inputs:
 command: login
 containerRegistry: $(containerRegistryServiceConnection)

- task: DownloadSecureFile@1
 name: privateKey
 inputs:
 secureFile: cc8f3c6f998bee63fefaaabc5a2202eab06867b83f491813326481f56a95466f.key
- script: |
 mkdir -p $(DOCKER_CONFIG)/trust/private
 cp $(privateKey.secureFilePath) $(DOCKER_CONFIG)/trust/private

- task: Docker@2
 inputs:
 command: build
 Dockerfile: '**/Dockerfile'
 containerRegistry: $(containerRegistryServiceConnection)
 repository: $(imageRepository)
 tags: |
 $(tag)
 arguments: '--disable-content-trust=false'

- task: Docker@2
 inputs:
 command: push
 containerRegistry: $(containerRegistryServiceConnection)
 repository: $(imageRepository)
 tags: |
 $(tag)
 arguments: '--disable-content-trust=false'
 env:
 DOCKER_CONTENT_TRUST_REPOSITORY_PASSPHRASE: $(DOCKER_CONTENT_TRUST_REPOSITORY_PASSPHRASE)

```

In the previous example, the `DOCKER_CONFIG` variable is set by the login action done by the Docker task. We recommend that you set up `DOCKER_CONTENT_TRUST_REPOSITORY_PASSPHRASE` as a [secret variable](#) for your pipeline. The alternative approach of using a pipeline variable in YAML exposes the passphrase in plain text. We only need the repository passphrase in this example because the repository has been initiated already (prerequisites).

# Deploy to Kubernetes

2/26/2020 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Azure Pipelines can be used to deploy to Kubernetes clusters offered by multiple cloud providers. This document contains the concepts associated with setting up deployments for any Kubernetes cluster.

While it is possible to use script for loading kubeconfig files onto the agent from a remote location or secure files and then use kubectl for performing the deployments, the KubernetesManifest task and Kubernetes service connection can be used to do this in a simpler and more secure way.

## KubernetesManifest task

[KubernetesManifest task](#) has the added benefits of being able to check for object stability before marking a task as success/failure, perform artifact substitution, add pipeline traceability-related annotations onto deployed objects, simplify creation and referencing of imagePullSecrets, bake manifests using Helm or kustomization.yaml or Docker compose files, and aid in deployment strategy rollouts.

## Kubernetes resource in environments

[Kubernetes resource in environments](#) provides a secure way of specifying the credential required to connect to a Kubernetes cluster for performing deployments.

### Resource creation

In the **Azure Kubernetes Service** provider option, once the subscription, cluster and namespace inputs are provided, in addition to fetching and securely storing the required credentials, for an RBAC enabled cluster [ServiceAccount](#) and [RoleBinding](#) objects are created such that the ServiceAccount is able to perform actions only on the chosen namespace.

The **Generic provider** (reusing existing ServiceAccount) option can be used to configure a connection to any cloud provider's cluster (AKS/EKS/GKE/OpenShift/etc.).

## Example

```

jobs:
- deployment:
 displayName: Deploy to AKS
 pool:
 vmImage: ubuntu-latest
 environment: contoso.aksnamespace
 strategy:
 runOnce:
 deploy:
 steps:
 - checkout: self
 - task: KubernetesManifest@0
 displayName: Create secret
 inputs:
 action: createSecret
 namespace: aksnamespace
 secretType: dockerRegistry
 secretName: foo-acr-secret
 dockerRegistryEndpoint: fooACR

 - task: KubernetesManifest@0
 displayName: Create secret
 inputs:
 action: createSecret
 namespace: aksnamespace
 secretType: dockerRegistry
 secretName: bar-acr-secret
 dockerRegistryEndpoint: barACR

 - task: KubernetesManifest@0
 displayName: Deploy
 inputs:
 action: deploy
 namespace: aksnamespace
 manifests: manifests/deployment.yml|manifests/service.yml
 containers: |
 foo.azurecr.io/demo:${tagVariable1}
 bar.azurecr.io/demo:${tagVariable2}
 imagePullSecrets: |
 foo-acr-secret
 bar-acr-secret

```

Note that to allow image pull from private registries, prior to the `deploy` action, the `createSecret` action is used along with instances of [Docker registry service connection](#) to create imagePullSecrets that are subsequently referenced in the step corresponding to `deploy` action.

#### TIP

- If setting up an end-to-end CI-CD pipeline from scratch for a repository containing a Dockerfile, checkout the [Deploy to Azure Kubernetes template](#), which constructs an end-to-end YAML pipeline along with creation of an [environment](#) and [Kubernetes resource](#) to help visualize these deployments.
- While YAML based pipeline currently supports triggers on a single Git repository, if triggers are required for manifest files stored in another Git repository or if triggers are required for Azure Container Registry or Docker Hub, usage of release pipelines instead of a YAML based pipeline is recommended for doing the Kubernetes deployments.

## Alternatives

Instead of using the `KubernetesManifest` task for deployment, one can also use the following alternatives:

- [Kubectl task](#)

- kubectl invocation on script. For example: `script: kubectl apply -f manifest.yml`

# Bake manifests

2/26/2020 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Bake action of the [Kubernetes manifest task](#) is useful for turning templates into manifests with the help of a template engine. The bake action of Kubernetes manifest task is intended to provide visibility into the transformation between the input templates and the end manifest files that are used in the deployments. [Helm 2](#), [kustomize](#), and [kompose](#) are supported as templating options under the bake action.

The baked manifest files are intended to be consumed downstream (subsequent task) where these manifest files are used as inputs for the deploy action of the Kubernetes manifest task.

## Helm 2 example

```
- deployment:
 displayName: Bake and deploy to AKS
 pool:
 vmImage: ubuntu-latest
 environment: contoso.aksnamespace
 strategy:
 runOnce:
 deploy:
 steps:
 - checkout: self
 - task: KubernetesManifest@0
 name: bake
 displayName: Bake K8s manifests from Helm chart
 inputs:
 action: bake
 renderType: helm2
 helmChart: charts/sample
 overrides: 'image.repository:nginx'

 - task: KubernetesManifest@0
 displayName: Deploy K8s manifests
 inputs:
 kubernetesServiceConnection: k8sSC1
 manifests: $(bake.manifestsBundle)
 containers: |
 nginx: 1.7.9
```

### NOTE

Instead of transforming the Helm charts into manifest files in the template as shown above, if one intends to use Helm for directly managing releases and rollbacks, checkout the [Package and Deploy Helm Charts task](#).

## Kustomize example

```
steps:
- task: KubernetesManifest@0
 name: bake
 displayName: Bake K8s manifests from kustomization path
 inputs:
 action: bake
 renderType: kustomize
 kustomizationPath: folderContainingKustomizationFile

- task: KubernetesManifest@0
 displayName: Deploy K8s manifests
 inputs:
 kubernetesServiceConnection: k8sSC1
 manifests: $(bake.manifestsBundle)
```

## Kompose example

```
steps:
- task: KubernetesManifest@0
 name: bake
 displayName: Bake K8s manifests from Docker Compose
 inputs:
 action: bake
 renderType: kompose
 dockerComposeFile: docker-compose.yaml

- task: KubernetesManifest@0
 displayName: Deploy K8s manifests
 inputs:
 kubernetesServiceConnection: k8sSC1
 manifests: $(bake.manifestsBundle)
```

# Multi-cloud Kubernetes deployments

3/20/2021 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

With Kubernetes having a standard interface and running the same way on all cloud providers, Azure Pipelines can be used for deploying to Azure Kubernetes Service (AKS), Google Kubernetes Engine (GKE), Amazon Elastic Kubernetes Service (EKS), or clusters from any other cloud providers. This document contains information on how to connect to each of these clusters, and how to perform parallel deployments to multiple clouds.

## Setup environment and Kubernetes resources

Kubernetes resources belonging to environments can be targeted from deployment jobs to enable pipeline traceability and ability to diagnose resource health.

### NOTE

Deployments to Kubernetes clusters are possible using regular jobs as well, but the benefits of pipeline traceability and ability to diagnose resource health are not available in this option.

To set up multi-cloud deployment, [create an environment](#) and subsequently add Kubernetes resources associated with namespaces of Kubernetes clusters. Follow the steps under the linked sections based on the cloud provider of your Kubernetes cluster -

- [Azure Kubernetes Service](#)
- [Generic provider using existing service account](#) (For GKE/EKS/...)

### TIP

The generic provider approach based on existing service account works with clusters from any cloud provider, including Azure. The incremental benefit of using the Azure Kubernetes Service option instead is that it involves creation of new ServiceAccount and RoleBinding objects (instead of reusing an existing ServiceAccount) so that the newly created RoleBinding object limits the operations of the ServiceAccount to the chosen namespace only.

## Parallel deployments to multiple clouds

The following YAML snippet showcases how to perform parallel deployments to clusters from multiple clouds. In this example, deployments are done to resources corresponding to namespaces from AKS, GKE, EKS, and OpenShift clusters. These four namespaces are associated with Kubernetes resources under the 'contoso' environment.

```
trigger:
- master

jobs:
- deployment:
 displayName: Deploy to AKS
 pool:
 vmImage: ubuntu-latest
 environment: contoso.aksnamespace
 strategy:
 runOnce:
```

```
deploy:
 steps:
 - checkout: self
 - task: KubernetesManifest@0
 displayName: Deploy to Kubernetes cluster
 inputs:
 action: deploy
 namespace: aksnamespace
 manifests: manifests/*
- deployment:
 displayName: Deploy to GKE
 pool:
 vmImage: ubuntu-latest
 environment: contoso.gkenamespace
 strategy:
 runOnce:
 deploy:
 steps:
 - checkout: self
 - task: KubernetesManifest@0
 displayName: Deploy to Kubernetes cluster
 inputs:
 action: deploy
 namespace: gkenamespace
 manifests: manifests/*
- deployment:
 displayName: Deploy to EKS
 pool:
 vmImage: ubuntu-latest
 environment: contoso.eksnamespace
 strategy:
 runOnce:
 deploy:
 steps:
 - checkout: self
 - task: KubernetesManifest@0
 displayName: Deploy to Kubernetes cluster
 inputs:
 action: deploy
 namespace: eksnamespace
 manifests: manifests/*
- deployment:
 displayName: Deploy to OpenShift
 pool:
 vmImage: ubuntu-latest
 environment: contoso.openshiftnamespace
 strategy:
 runOnce:
 deploy:
 steps:
 - checkout: self
 - task: KubernetesManifest@0
 displayName: Deploy to Kubernetes cluster
 inputs:
 action: deploy
 namespace: openshiftnamespace
 manifests: manifests/*
- deployment:
 displayName: Deploy to DigitalOcean
 pool:
 vmImage: ubuntu-latest
 environment: contoso.digitaloceannamespace
 strategy:
 runOnce:
 deploy:
 steps:
 - checkout: self
 - task: KubernetesManifest@0
 displayName: Deploy to Kubernetes cluster
```

```
displayName: deploy to Kubernetes Cluster
inputs:
 action: deploy
 namespace: digitaloceannamespace
 manifests: manifests/*
```

#### NOTE

When using the service account option, [ensure that a RoleBinding exists](#), which grants permissions in the [edit ClusterRole](#) to the desired service account. This is needed so that the service account can be used by Azure Pipelines for creating objects in the chosen namespace.

# Deployment strategies for Kubernetes in Azure Pipelines

2/26/2020 • 3 minutes to read • [Edit Online](#)

## Azure Pipelines

[Kubernetes manifest task](#) currently supports canary deployment strategy. This document explains the guidelines and best practices around the usage of this task for setting up canary deployments to Kubernetes.

## Overview of canary deployment strategy

Canary deployment strategy involves partially deploying the new changes in such a way that the new changes coexist with the current deployments before performing a full rollout. In this phase, usually a final check and comparison of the two versions is done on the basis of application health checks and performance monitoring using metrics originating from both the versions. If the canary is found to be at least at par or better than the currently deployed version, complete rollout of the new changes is initiated. If the canary is found to be performing worse than the currently deployed versions, the new changes are rejected, and a complete rollout, which could lead to regression, is thus avoided.

## Canary deployments for Kubernetes

The label selector relationship between pods and services in Kubernetes allow for setting up deployments in such a way that a single service routes requests to both the stable and the canary variants. Kubernetes manifest task utilizes this to facilitate canary deployments in the following way

- If the task is provided the inputs of `action: deploy` and `strategy: canary`, for each workload (Deployment, ReplicaSet, Pod, ...) defined in the input manifest files, a `-baseline` and `-canary` variant of the deployment are created. For example
  - Assume there exists a deployment named `sampleapp` in the input manifest file and that after completion of run number 22 of the pipeline, the stable variant of this deployment named `sampleapp` is deployed in the cluster
  - In the subsequent run (in this case run number 23), Kubernetes manifest task with `action: deploy` and `strategy: canary` would result in creation of `sampleapp-baseline` and `sampleapp-canary` deployments whose number of replicas are determined by the product of `percentage` task input with the value of the desired number of replicas for the final stable variant of `sampleapp` as per the input manifest files
  - Excluding the number of replicas, the baseline version has the same configuration as the stable variant while the canary version has the new changes that are being introduced by the current run (in this case, run number 23)
- If a manual intervention is set up in the pipeline after the above mentioned step, it would allow for an opportunity to pause the pipeline so that the pipeline admin can evaluate key metrics for the baseline and canary versions and take the decision on whether the canary changes are safe and good enough for a complete rollout.
- `action: promote` and `strategy: canary` or `action: reject` and `strategy: canary` inputs of the Kubernetes manifest tasks can be used to promote or reject the canary changes respectively. Note that in either cases, at the end of this step, only the stable variant of the workloads declared in the input manifest files will be remain deployed in the cluster, while the ephemeral baseline and canary versions are cleaned up.

#### NOTE

The percentage input mentioned above doesn't result in percentage traffic split, but rather refers to the percentage used for calculating the number of replicas for baseline and canary variants. A service mesh's service discovery and load balancing capabilities would help in achieving a true request based traffic split. Support for canary deployments utilizing [Service Mesh Interface](#) is currently being worked on and will be added to the Kubernetes manifest task soon.

## Compare canary against baseline and not against stable variant

While it is possible to compare the canary deployment against the current production deployment, it is better to instead compare the canary against an equivalent baseline. As the baseline anyways uses the same version and configuration as the currently deployed version, canary and baseline are better candidates for comparison as they are identical with respect to the following aspects:

- Same time of deployment
- Same size of deployment
- Same type and amount of traffic This minimizes the effects of factors such as cache warmup time, the heap size, and so on, in the canary variant, which could have adversely impacted the analysis.

## End-to-end example

An [end-to-end example](#) of setting up build and release pipelines to perform canary deployments on Kubernetes clusters for each change made to application code is available under the how-to guides. This example also demonstrates the usage of Prometheus for comparing the baseline and canary metrics when the pipeline is paused using a manual intervention task.

# Canary deployment strategy for Kubernetes deployments

11/2/2020 • 13 minutes to read • [Edit Online](#)

## Azure Pipelines

Canary deployment strategy involves deploying new versions of an application next to stable production versions to see how the canary version compares against the baseline before promoting or rejecting the deployment. This step-by-step guide covers usage of [Kubernetes manifest task's](#) canary strategy support for setting up canary deployments for Kubernetes and the associated workflow in terms of instrumenting code and using the same for comparing baseline and canary before taking a manual judgment on promotion/rejection of the canary.

## Prerequisites

- A repository in Azure Container Registry or Docker Hub (Azure Container Registry, Google Container Registry, Docker Hub) with push privileges.
- Any Kubernetes cluster (Azure Kubernetes Service, Google Kubernetes Engine, Amazon Elastic Kubernetes Service).

## Sample code

Fork the following repository on GitHub -

```
https://github.com/MicrosoftDocs/azure-pipelines-canary-k8s
```

Here's a brief overview of the files in the repository that are used during the course of this guide -

- ./app:
  - app.py - Simple Flask based web server instrumented using [Prometheus instrumentation library for Python applications](#). A custom counter is set up for the number of 'good' and 'bad' responses given out based on the value of `success_rate` variable.
  - Dockerfile - Used for building the image with each change made to app.py. With each change made to app.py, build pipeline (CI) is triggered and the image gets built and pushed to the container registry.
- ./manifests:
  - deployment.yml - Contains specification of the `sampleapp` Deployment workload corresponding to the image published earlier. This manifest file is used not just for the stable version of Deployment object, but for deriving the -baseline and -canary variants of the workloads as well.
  - service.yml - Creates `sampleapp` service for routing requests to the pods spun up by the Deployments (stable, baseline, and canary) mentioned above.
- ./misc
  - service-monitor.yml - Used for setup of a [ServiceMonitor](#) object to set up Prometheus metric scraping.
  - fortio-deploy.yml - Used for setup of fortio deployment that is subsequently used as a load-testing tool to send a stream of requests to the `sampleapp` service deployed earlier. With `sampleapp` service's selector being applicable for all the three pods resulting from the Deployment objects that get created during the course of this how-to guide - `sampleapp`, `sampleapp-baseline` and `sampleapp-canary`, the stream of requests sent to `sampleapp` get routed to pods under all these three deployments.

#### NOTE

While [Prometheus](#) is used for code instrumentation and monitoring in this how-to guide, any equivalent solution like [Azure Application Insights](#) can be used as an alternative as well.

## Install prometheus-operator

Use the following command from your development machine (with kubectl and Helm installed and context set to the cluster you want to deploy against) to install Prometheus on your cluster. [Grafana](#), which is used later in this how-to guide for visualizing the baseline and canary metrics on dashboards, is installed as part of this Helm chart -

```
helm install --name sampleapp stable/prometheus-operator
```

## Create service connections

- Navigate to **Project settings** -> **Pipelines** -> **Service connections**.
- Create a [Docker registry service connection](#) associated with your container registry. Name it **azure-pipelines-canary-k8s**.
- Create a [Kubernetes service connection](#) for the Kubernetes cluster and namespace you want to deploy to. Name it **azure-pipelines-canary-k8s**.

## Setup continuous integration

1. Navigate to **Pipelines** -> **New pipeline** and select your repository.
2. Upon reaching **Configure** tab, choose **Starter pipeline**
3. In **Review** tab, replace the contents of the pipeline YAML with the following snippet -

```
trigger:
- master

pool:
 vmImage: Ubuntu-16.04

variables:
 imageName: azure-pipelines-canary-k8s

steps:
- task: Docker@2
 displayName: Build and push image
 inputs:
 containerRegistry: dockerRegistryServiceConnectionName #replace with name of your Docker registry
 service connection
 repository: $(imageName)
 command: buildAndPush
 Dockerfile: app/Dockerfile
 tags: |
 $(Build.BuildId)
```

If the Docker registry service connection created by you was associated with `foobar.azurecr.io`, then the image is to `foobar.azurecr.io/azure-pipelines-canary-k8s:$(Build.BuildId)` based on the above configuration.

## Edit manifest file

In manifests/deployment.yaml, replace `<foobar>` with your container registry's URL. For example after replacement, the image field should look something like `contosodemo.azurecr.io/azure-pipelines-canary-k8s`.

## Setup continuous deployment

### Deploy canary stage

- [YAML](#)
- [Classic](#)

1. Navigate to Pipelines -> Environments -> New environment
2. Configure the new environment as follows -
  - **Name:** akscanary
  - **Resource:** choose Kubernetes
3. Click on Next and now configure your Kubernetes resource as follows -
  - **Provider:** Azure Kubernetes Service
  - **Azure subscription:** Choose the subscription that holds your kubernetes cluster
  - **Cluster:** Choose your cluster
  - **Namespace:** Create a new namespace with the name **canarydemo**
4. Click on Validate and Create
5. Navigate to Pipelines -> Select the pipeline you just created -> Edit
6. Change the step you created previously to now use a Stage. And add two additional steps to copy the manifests and mics directories as artifacts for use by consecutive stages. You might also want to move a couple of values to variables for easier usage later in your pipeline. Your complete YAML should now look like this:

```

trigger:
- master

pool:
 vmImage: Ubuntu-16.04

variables:
 imageName: azure-pipelines-canary-k8s
 dockerRegistryServiceConnection: dockerRegistryServiceConnectionName #replace with name of your
Docker registry service connection
 imageRepository: 'azure-pipelines-canary-k8s'
 containerRegistry: containerRegistry #replace with the name of your container registry, Should be
in the format foobar.azurecr.io
 tag: '$(Build.BuildId)'

stages:
- stage: Build
 displayName: Build stage
 jobs:
- job: Build
 displayName: Build
 pool:
 vmImage: Ubuntu-16.04
 steps:
- task: Docker@2
 displayName: Build and push image
 inputs:
 containerRegistry: $(dockerRegistryServiceConnection)
 repository: $(imageName)
 command: buildAndPush
 Dockerfile: app/Dockerfile
 tags: |
 $(tag)

- upload: manifests
 artifact: manifests

- upload: misc
 artifact: misc

```

7. Add an additional stage at the bottom of your YAML file to deploy the canary version.

```

- stage: DeployCanary
 displayName: Deploy canary
 dependsOn: Build
 condition: succeeded()

 jobs:
 - deployment: Deploycanary
 displayName: Deploy canary
 pool:
 vmImage: Ubuntu-16.04
 environment: 'akscanary.canarydemo'
 strategy:
 runOnce:
 deploy:
 steps:
 - task: KubernetesManifest@0
 displayName: Create imagePullSecret
 inputs:
 action: createSecret
 secretName: azure-pipelines-canary-k8s
 dockerRegistryEndpoint: azure-pipelines-canary-k8s

 - task: KubernetesManifest@0
 displayName: Deploy to Kubernetes cluster
 inputs:
 action: 'deploy'
 strategy: 'canary'
 percentage: '25'
 manifests: |
 $(Pipeline.Workspace)/manifests/deployment.yml
 $(Pipeline.Workspace)/manifests/service.yml
 containers: '$(containerRegistry)$(imageRepository):$(tag)'
 imagePullSecrets: azure-pipelines-canary-k8s

 - task: KubernetesManifest@0
 displayName: Deploy Forbio and ServiceMonitor
 inputs:
 action: 'deploy'
 manifests: |
 $(Pipeline.Workspace)/misc/*

```

- Save your pipeline by committing directly to the main branch. This commit should already run your pipeline successfully.

#### **Manual intervention for promoting or rejecting canary**

- [YAML](#)
- [Classic](#)

1. Navigate to Pipelines -> Environments -> New environment
2. Configure the new environment as follows -
  - **Name:** akspromote
  - **Resource:** choose Kubernetes
3. Click on Next and now configure your Kubernetes resource as follows -
  - **Provider:** Azure Kubernetes Service
  - **Azure subscription:** Choose the subscription that holds your kubernetes cluster
  - **Cluster:** Choose your cluster
  - **Namespace:** Choose the namespace canarydemo namespace you created earlier
4. Click on Validate and Create

5. Select your new akspromote environment from the list of environments.
6. Click on the button with the **three dots** in the top right -> **Approvals and checks** -> **Approvals**
7. Configure your approval as follows -
  - **Approvers:** Add your own user account
  - **Advanced:** Make sure the **Allow approvers to approve their own runs** checkbox is checked.
8. Click on **Create**
9. Navigate to **Pipelines** -> Select the pipeline you just created -> **Edit**
10. Add an additional stage PromoteRejectCanary at the end of your YAML file to promote the changes.

```

- stage: PromoteRejectCanary
 displayName: Promote or Reject canary
 dependsOn: DeployCanary
 condition: succeeded()

jobs:
- deployment: PromoteCanary
 displayName: Promote Canary
 pool:
 vmImage: Ubuntu-16.04
 environment: 'aks(promote.canarydemo)'
 strategy:
 runOnce:
 deploy:
 steps:
 - task: KubernetesManifest@0
 displayName: promote canary
 inputs:
 action: 'promote'
 strategy: 'canary'
 manifests: '$(Pipeline.Workspace)/manifests/*'
 containers: '$(containerRegistry)$(imageRepository):$(tag)'
 imagePullSecrets: '$(imagePullSecret)'
```

11. Add an additional stage RejectCanary at the end of your YAML file to roll back the changes.

```

- stage: RejectCanary
 displayName: Reject canary
 dependsOn: PromoteRejectCanary
 condition: failed()

jobs:
- deployment: RejectCanary
 displayName: Reject Canary
 pool:
 vmImage: Ubuntu-16.04
 environment: 'akscanary.canarydemo'
 strategy:
 runOnce:
 deploy:
 steps:
 - task: KubernetesManifest@0
 displayName: reject canary
 inputs:
 action: 'reject'
 strategy: 'canary'
 manifests: '$(Pipeline.Workspace)/manifests/*'
```

12. Save your YAML pipeline by clicking on **Save** and commit it directly to the main branch.

## Deploy a stable version

- [YAML](#)
- [Classic](#)

Currently for the first run of the pipeline, the stable version of the workloads and their baseline/canary version do not exist in the cluster. To deploy the stable version -

1. In `app/app.py`, change `success_rate = 5` to `success_rate = 10`. This change triggers the pipeline leading to build and push of the image to container registry. It will also trigger the DeployCanary stage.
2. Given you have configured an approval on the `aks(promote)` environment, the release will wait before executing that stage.
3. In the summary of the run click on **Review** and next click on **Approve** in the subsequent fly-out. This will result in the stable version of the workloads (`sampleapp` deployment in `manifests/deployment.yaml`) being deployed to the namespace

## Initiate canary workflow

Once the above release has been completed, the stable version of workload `sampleapp` now exists in the cluster.

To understand how baseline and canaries are created for comparison purposes with every subsequent deployment, perform the following changes to the simulation application -

1. In `app/app.py`, change `success_rate = 10` to `success_rate = 20`

The above change triggers build pipeline resulting in build and push of image to the container registry, which in turn triggers the release pipeline and the commencement of **Deploy canary** stage.

## Simulate requests

On your development machine, run the following commands and keep it running to send a constant stream of requests at the `sampleapp` service. `sampleapp` service routes the requests to the pods spun by stable `sampleapp` deployment and the pods spun up by `sampleapp-baseline` and `sampleapp-canary` deployments as the selector specified for `sampleapp` is applicable for all these pods.

```
FORTIO_POD=$(kubectl get pod | grep fortio | awk '{ print $1 }')
kubectl exec -it $FORTIO_POD -c fortio /usr/bin/fortio -- load -allow-initial-errors -t 0
http://sampleapp:8080/
```

## Setup Grafana dashboard

1. Run the following port forwarding command on your local development machine to be able to access Grafana -

```
kubectl port-forward svc/sampleapp-grafana 3000:80
```

2. In a browser, open the following URL -

```
http://localhost:3000/login
```

3. When prompted for login credentials, unless the `adminPassword` value was overridden during `prometheus-operator` Helm chart installation, use the following values -

- username: admin

- password: prom-operator
- In the left navigation menu, choose + -> **Dashboard** -> **Graph**
  - Click anywhere on the newly added panel and type  to edit the panel.
  - In the **Metrics** tab, enter the following query -

```
rate(requests_total{pod=~"sampleapp-.*", custom_status="good"}[1m])
```
  - In the **General** tab, change the name of this panel to **All sampleapp pods**
  - In the overview bar at the top of the page, change the duration range to **Last 5 minutes** or **Last 15 minutes**.
  - Click on the save icon in the overview bar to save this panel.
  - While the above panel visualizes success rate metrics from all the variants - stable (from  deployment), baseline (from  deployment) and canary (from  deployment), you can visualize just the baseline and canary metrics by adding another panel with the following configuration -

- General tab -> Title: sampleapp baseline and canary
- Metrics tab -> query to be used:

```
rate(requests_total{pod=~"sampleapp-baseline-.*|sampleapp-canary-.*", custom_status="good"}[1m])
```

#### NOTE

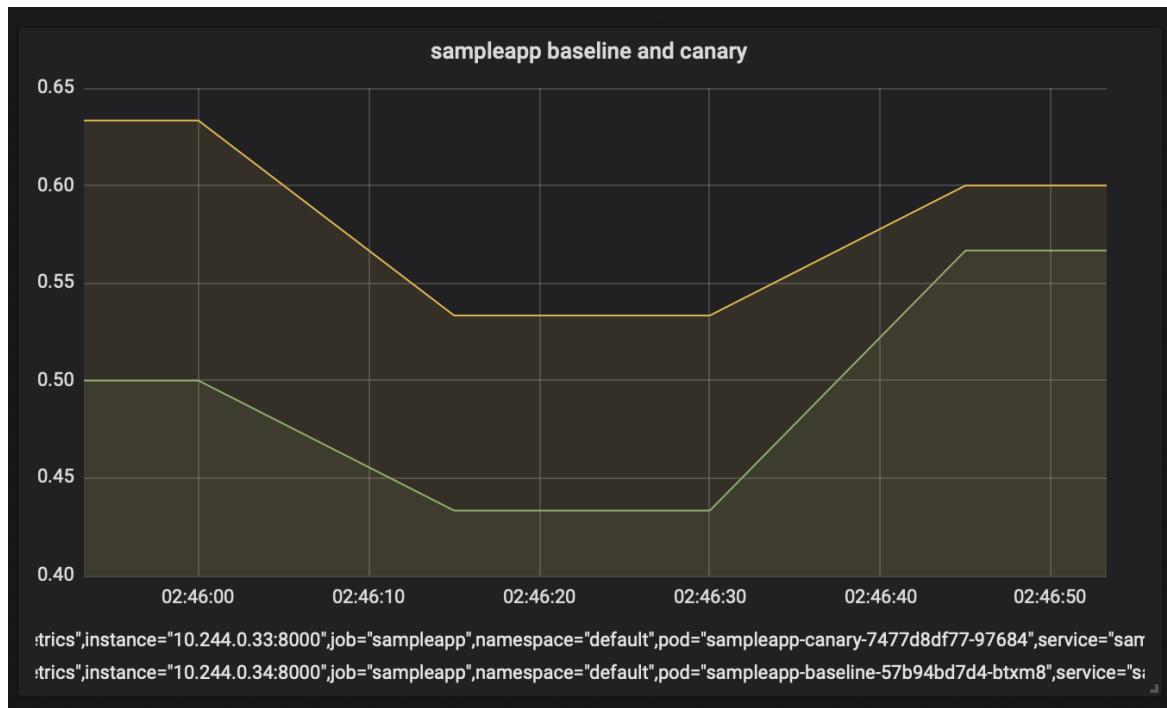
Note that the panel for baseline and canary metrics will only have metrics available for comparison when the **Deploy canary** stage has successfully completed and the **Promote/reject canary** stage is waiting on manual intervention.

#### TIP

Setup [annotations for Grafana dashboards](#) to visually depict stage completion events for **Deploy canary** and **Promote/reject canary** so that you know when to start comparing baseline with canary and when the promotion/rejection of canary has completed respectively.

## Compare baseline and canary

- At this point, with **Deploy canary** stage having successfully completed (based on the change of  from '10' to '20') and with the **Promote/reject canary** stage is waiting on manual intervention, one can compare the success rate (as determined by `custom_status=good`) of baseline and canary variants in the Grafana dashboard. It should look similar to the below image -



2. Based on the observation that the success rate is higher for canary, promote the canary by clicking on **Resume** in the manual intervention task

# Artifacts in Azure Pipelines overview

3/31/2021 • 2 minutes to read • [Edit Online](#)

Azure Artifacts enable developers to consume and publish different types of packages to Artifacts feeds and public registries such as NuGet.org and npmjs.com. You can use Azure Artifacts in conjunction with Azure Pipelines to deploy packages, publish build artifacts, or integrate files between your pipeline stages to build, test, or deploy your application.

## Supported artifact types

The following table describes supported artifact types in Azure Pipelines.

SUPPORTED ARTIFACT TYPE	DESCRIPTION
<a href="#">Build artifacts</a>	The files produced by a build such as .dll, .exe, or .PDB files.
<a href="#">Maven</a>	Publish Maven packages to Azure Artifacts feeds or Maven central repository.
<a href="#">npm</a>	Publish npm packages to Azure Artifacts feeds or npm registry.
<a href="#">NuGet</a>	Publish NuGet packages to Azure Artifacts feeds or NuGet public repository.
<a href="#">PyPI</a>	Publish Python packages to Azure Artifacts feeds or PyPI registry.
<a href="#">Universal packages</a>	Publish Universal Packages to Azure Artifacts feeds.
<a href="#">Symbols</a>	Symbol files contain debugging information about the compiled executables. You can publish symbols to a symbol server in Azure Artifacts to debug your application. Symbol servers enable debuggers to automatically retrieve the correct symbol files without knowing the specific product, package, or build information.

## How do I publish and consume artifacts?

Each 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 Maven. Other artifact types offer more generic file storage, such as pipeline artifacts and Universal Packages.

- [NuGet](#)
  - [Npm](#)
  - [Python](#)
  - [Maven](#)
- 
- [Publish a NuGet package using the command line](#)
  - [Publish to NuGet feeds \(YAML/Classic\)](#)

- Consume NuGet packages

## Next steps

[Publish and download artifacts in Azure Pipelines](#) [Releases in Azure Pipelines](#) [Build Artifacts](#)

# Publish and download artifacts in Azure Pipelines

3/23/2021 • 8 minutes to read • [Edit Online](#)

## Azure Pipelines

Pipeline artifacts provide a way to share files between stages in a pipeline or between different pipelines. They are typically the output of a build process that needs to be consumed by another job or be deployed. Artifacts are associated with the run they were produced in and remain available after the run has completed.

### NOTE

Both `PublishPipelineArtifact@1` and `DownloadPipelineArtifact@2` require a minimum agent version of 2.153.1

## Publish artifacts

You can publish your artifacts using YAML, the classic web UI, or Azure CLI

- [YAML](#)
- [YAML \(task\)](#)
- [Classic](#)
- [Azure CLI](#)

```
steps:
- publish: $(System.DefaultWorkingDirectory)/bin/WebApp
 artifact: WebApp
```

### NOTE

The `publish` keyword is a shortcut for the [Publish Pipeline Artifact task](#).

Things to keep in mind:

- Although artifact name is optional, it is a good practice to specify a name that accurately reflects the contents of the artifact.
- The path of the file or folder to publish is required. It can be an absolute or a relative path to `$(System.DefaultWorkingDirectory)`.
- If you plan to consume the artifact from a job running on a different operating system or file system, you must ensure all file paths are valid for the target environment. For example, a file name containing a `\` or `*` character will typically fail to download on Windows.

### NOTE

You will not be billed for storage of Pipeline Artifacts, Build Artifacts, and Pipeline Caching. For more information, see [Which artifacts count toward my total billed storage](#).

### Caution

Deleting a build that published Artifacts to a file share will result in the deletion of all Artifacts in that UNC path.

## Use `.artifactignore`

`.artifactignore` uses the identical file-globbing syntax of `.gitignore` (with very few limitations) to provide a version-controlled way to specify which files should not be included when publishing artifacts. For more information, see [.`artifactignore` usage].

Example: ignore all files except `.exe` files:

```
**/*
!*.*.exe
```

### NOTE

`.artifactignore` follows the same syntax as `.gitignore` with some minor limitations. The plus sign character `+` is not supported in URL paths as well as some of the builds semantic versioning metadata (`+ suffix`) in some packages types such as Maven.

## Download artifacts

You can download artifacts using YAML, the classic web UI, or Azure CLI.

- [YAML](#)
- [YAML \(task\)](#)
- [Classic](#)
- [Azure CLI](#)

```
steps:
- download: current
 artifact: WebApp
```

- **current**: download artifacts produced by the current pipeline run. Options: current, specific.

### NOTE

The `download` keyword is a shortcut to the [Download Pipeline Artifact](#) task.

Things to keep in mind:

- By default, files are downloaded to `$(Pipeline.Workspace)`. If an artifact name was not specified, a subdirectory will be created for each downloaded artifact.
- File matching patterns can be used to limit which files are downloaded.
- To download artifacts from a different pipeline or from earlier stages in your pipeline see, [Download Pipeline Artifacts](#) task.

### Artifacts selection

A single download step can download one or more artifacts. To download multiple artifacts, leave the *artifact name* field empty and use file matching patterns to limit which files will be downloaded. `**` is the default file matching pattern (all files in all artifacts).

#### Single artifact

When an artifact name is specified:

1. Only files for that specific artifact are downloaded. If the artifact does not exist, the task will fail.
2. File matching patterns are evaluated relative to the root of the artifact. For example, the pattern `*.jar` matches all files with a `.jar` extension at the root of the artifact.

The following example illustrates how to download all `*.js` from an artifact `WebApp`:

- [YAML](#)
- [YAML \(task\)](#)
- [Classic](#)
- [Azure CLI](#)

```
steps:
- download: current
 artifact: WebApp
 patterns: '**/*.js'
```

#### Multiple artifacts

When no artifact name is specified:

1. Multiple artifacts can be downloaded and the task does not fail if no files are found.
2. A subdirectory is created for each artifact.
3. File matching patterns should assume the first segment of the pattern is (or matches) an artifact name.  
For example, `WebApp/**` matches all files from the `WebApp` artifact. The pattern `*/*.dll` matches all files with a `.dll` extension at the root of each artifact.

The following example illustrates how to download all `.zip` files from all artifacts:

- [YAML](#)
- [YAML \(task\)](#)
- [Classic](#)
- [Azure CLI](#)

```
steps:
- download: current
 patterns: '**/*.zip'
```

## Artifacts in release and deployment jobs

Artifacts are only downloaded automatically in deployment jobs. The download artifact task will be auto injected only when using the `deploy` lifecycle hook in your deployment. To stop artifacts from being downloaded automatically, add a `download` step and set its value to none. In a regular build job, you need to explicitly use the `download` step keyword or the [Download Pipeline Artifact](#) task. See [lifecycle hooks](#) to learn more about the other types of hooks.

```
steps:
- download: none
```

#### NOTE

Deleting or overwriting Pipeline Artifacts is not currently supported. The recommended workflow if you want to re-run a failed pipeline job is to include the job ID in the artifact name. `$(system.JobId)` is the appropriate variable for this purpose. See [System variables](#) to learn more about predefined variables.

## Use Artifacts across stages

If you want to be able to access your artifact across different stages in your pipeline, you can use output variables to pass it to the next stage in your YAML. See [Dependency expressions](#) for more details.

```
trigger: none
pool:
 vmImage: 'ubuntu-latest'
stages:
- stage: A
 jobs:
 - job: A1
 steps:
 - script: echo ##vso[task.setvariable variable=shouldrun;isOutput=true]true
 name: printvar
- stage: B
 dependsOn: A
 jobs:
 - job: B1
 condition: in(stageDependencies.A.A1.result, 'Succeeded', 'SucceededWithIssues', 'Skipped')
 steps:
 - script: echo hello from Job B1
 - job: B2
 condition: eq(stageDependencies.A.A1.outputs['printvar.shouldrun'], 'true')
 steps:
 - script: echo hello from Job B2
```

## Migrate from build artifacts

Pipeline artifacts are the next generation of build artifacts and are the recommended way to work with artifacts. Artifacts published using the **Publish Build Artifacts** task can continue to be downloaded using **Download Build Artifacts**, but we recommend using the latest **Download Pipeline Artifact** task instead.

When migrating from build artifacts to pipeline artifacts:

1. For build artifacts, it's common to copy files to `$(Build.ArtifactStagingDirectory)` and then use the **Publish Build Artifacts** task to publish this folder. With the **Publish Pipeline Artifact** task, you can just publish directly from the path containing the files.
2. By default, the **Download Pipeline Artifact** task downloads files to `$(Pipeline.Workspace)`. This is the default and recommended path for all types of artifacts.
3. File matching patterns for the **Download Build Artifacts** task are expected to start with (or match) the artifact name, regardless if a specific artifact was specified or not. In the **Download Pipeline Artifact** task, patterns should not include the artifact name when an artifact name has already been specified. For more information, see [single artifact selection](#).

#### TIP

For more information on billing and usage tiers, check out the [Azure DevOps pricing tool](#).

## FAQ

- **Can this task publish artifacts to a shared folder or a network path?**

Not currently, but this feature is planned.

- **What are build artifacts?**

Build artifacts are the files generated by your build. See [Build Artifacts](#) to learn more about how to publish and consume your build artifacts.

## Related articles

- [Build artifacts](#)
- [Releases in Azure Pipelines](#)
- [Release artifacts and artifact sources](#)
- [How to mitigate risk when using private package feeds](#)

# Artifacts in Azure Pipelines

3/23/2021 • 6 minutes to read • [Edit Online](#)

## NOTE

We recommend upgrading from **build artifacts** (`PublishBuildArtifacts@1` and `DownloadBuildArtifacts@0`) to **pipeline artifacts** (`PublishPipelineArtifact@1` and `DownloadPipelineArtifact@2`) for faster output storage speeds.

Azure Artifacts is a service that enables teams to use feeds and upstream sources to manage their dependencies. You can use Azure Pipelines to publish and consume different types of artifacts as part of your CI/CD workflow.

## How do I publish artifacts?

Artifacts can be published at any stage of your pipeline. You can use YAML or the classic Azure DevOps editor to publish your packages.

### Publish a text file

- [YAML](#)
- [Classic](#)

```
- 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
```

- **pathToPublish**: the path of your artifact. This can be an absolute or a relative path. Wildcards are not supported.
- **artifactName**: the name of your artifact.

## NOTE

Make sure you are not using one of the reserved folder names when publishing your artifact. See [Application Folders](#) for more details.

YAML is not supported in TFS.

### Publish two sets of artifacts

- [YAML](#)
- [Classic](#)

```

- 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

```

- **pathToPublish**: the path of your artifact. This can be an absolute or a relative path. Wildcards are not supported.
- **artifactName**: the name of your artifact.

YAML is not supported in TFS.

## Example: Assemble C++ artifacts into one location and publish as an artifact

- [YAML](#)
- [Classic](#)

```

- 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)'
 targetFolder: '$(Build.ArtifactStagingDirectory)'
- task: PublishBuildArtifacts@1
 inputs:
 pathToPublish: '$(Build.ArtifactStagingDirectory)'
 artifactName: drop

```

- **sourceFolder**: the folder that contains the files you want to copy. If you leave this value empty, copying will be done from the root folder of your repo (`$(Build.SourcesDirectory)`).
- **contents**: location(s) of the file(s) that will be copied to the destination folder.
- **targetFolder**: destination folder.
- **pathToPublish**: the folder or file path to publish. It can be an absolute or a relative path, and wildcards are not supported.
- **artifactName**: the name of the artifact that you want to create.

### NOTE

You cannot use **Bin**, **App\_Data** and other folder names reserved by IIS as an artifact name because this content is not served in response to Web requests. Please see [ASP.NET Web Project Folder Structure](#) for more details.

YAML is not supported in TFS.

## How do I consume artifacts?

You can consume your artifacts in different ways: you can use it in your release pipeline, pass it between your

pipeline jobs, download it directly from your pipeline and even download it from feeds and upstream sources.

## Consume artifacts in release pipelines

You can download artifacts produced by either a build pipeline (created in a classic editor) or a YAML pipeline (created through a YAML file) in a release pipeline and deploy them to the target of your choice.

## Consume an artifact in the next job of your pipeline

You can consume an artifact produced by one job in a subsequent job of the pipeline, even when that job is in a different stage (YAML pipelines). This can be useful to test your artifact.

### Download to debug

You can download an artifact directly from a pipeline for use in debugging.

- [YAML](#)
- [Classic](#)

```
- 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)'
```

- **buildType**: specify which build artifacts will be downloaded: `current` (the default value) or from a specific build.
- **downloadType**: choose whether to download a single artifact or all artifacts of a specific build.
- **artifactName**: the name of the artifact that will be downloaded.
- **downloadPath**: path on the agent machine where the artifacts will be downloaded.

YAML is not supported in TFS.

#### NOTE

In case you are using a `deployment` task, you can then reference your build artifacts by using `$(Agent.BuildDirectory)` variable. See [Agent variables](#) for more information on how to use predefined variables.

## 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 to [drop to a UNC file share](#).
- Use forward slashes in file path arguments so that they work for all agents. Backslashes don't work for macOS and Linux agents.
- Build artifacts are stored on a Windows filesystem, which causes all UNIX permissions to be lost, including the execution bit. You might need to restore the correct UNIX permissions after downloading your artifacts from Azure Pipelines or TFS.
- On Azure Pipelines and some versions of TFS, two different [variables](#) 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.
- Deleting a build that published Artifacts to a file share will result in the deletion of all Artifacts in that UNC path.
- 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](#) You can 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, select the **Summary** tab and see your artifact in the **Build artifacts published** section.

The screenshot shows the 'Summary' tab for a completed build. At the top, it displays the build number (#20200213.2) and a green checkmark indicating success. Below this, it shows the trigger (ramiMSFT), repository information (ramiMSFT/mslearn-tailspin-spacegame-web), and build duration (1m 19s). On the right side, there's a 'Tests and coverage' section with a link to 'Get started'. A red box highlights the '1 published' link under the 'Related' section, which typically links to the artifacts page.

When the build is done, if you watched it run, select the name of the completed build and then select the **Artifacts** tab to see your artifact.

The screenshot shows the 'Artifacts' tab for build 1688. It indicates the build succeeded and ran for 9 seconds. The 'Artifacts' tab is highlighted with a red box. Below the tabs, there's a list with a single item: 'drop'. There are 'Download' and 'Explore' buttons at the bottom.

From here, you can explore or download the artifacts.

You can also use Azure Pipelines to deploy your app by using the artifacts that you've published. See [Artifacts in Azure Pipelines releases](#).

## Publish from TFS to a 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. 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 by using [variables](#). For example: `\my\share\$(Build.DefinitionName)\$(Build.BuildNumber)`.

## Publish artifacts from TFS 2015 RTM

If you're using TFS 2015 RTM, the steps in the preceding examples are not available. Instead, you copy and publish your artifacts by using a single task: [Build: Publish Build Artifacts](#).

## Related articles

- [Publish and download artifacts in Azure Pipelines](#)
- [Define your multi-stage classic pipeline](#)
- [How to mitigate risk when using private package feeds](#)

# Releases in Azure Pipelines

4/14/2021 • 2 minutes to read • [Edit Online](#)

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

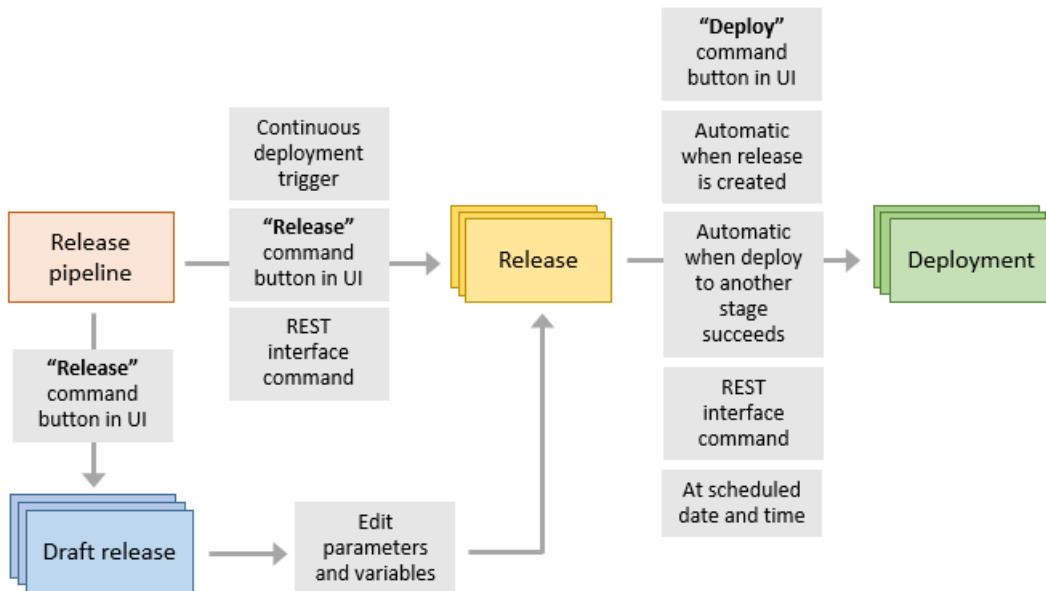
## NOTE

This topic covers classic release pipelines. If you want to create your pipelines using YAML, see [Customize your pipeline](#).

A **release** is a construct that holds a versioned set of artifacts specified in a CI/CD pipeline. It includes a snapshot of all the information required to carry out all the tasks and actions in the release pipeline, such as stages, tasks, policies such as triggers and approvers, and deployment options. There can be multiple releases from one release 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 can include running automated tests, deploying build artifacts, 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. To redeploy a release, simply navigate to the release you want to deploy and select **deploy**.

The following diagram shows the relationship between release, release pipelines, and deployments.



Create release pipelines

Releases can be created in several ways:

1. By using a [deployment trigger](#) to create a release every time a new build artifact is available.

The screenshot shows the 'Continuous deployment trigger' configuration for a build pipeline named 'DotNetSample-ASP.NET Core-Cl'. A red box highlights the 'Enabled' toggle switch, which is turned on. Below it, a note says 'Creates a release every time a new build is available.' Under 'Build branch filters', there is a table with one row showing 'Include' and 'The build pipeline's default bra...'. A 'Branch filter' dropdown menu is open, showing 'The build pipeline's default branch'.

2. By using the [Create release](#) button from within your [Pipelines > Releases](#) to manually create a release pipeline.

The screenshot shows the 'Releases' page for the 'FabrikamFiber' project. A red box highlights the 'Create release' button, which is located in the top right corner of the main content area. The page lists three previous releases: 'Release-5' (4/25/20), 'Release-4' (4/25/20), and 'Release-3' (4/25/20). The sidebar on the left contains icons for different pipeline types: TFS, New, All pipelines, Pipeline, Test, and Analytics.

3. By using the [REST API](#) to create a release definition.

## Q&A

Q: Why my deployment did not get triggered?

A: Creating a release pipeline does not necessarily mean that it will automatically/immediately start a deployment. Below are few reasons why this might happen:

- Defined [deployment triggers](#) forcing the deployment to pause. This can happen with scheduled triggers or when a delay is imposed until deployment to another stage is complete.
- Defined [queuing policies](#) dictating the order of execution and when releases are queued for deployment.
- [Pre-deployment approvals or gates](#) for a specific stage preventing deployment until all the defined conditions are met.

## Related articles

- [Release deployment control using approvals.](#)
- [Release deployment control using gates.](#)
- [Release triggers.](#)
- [Release artifacts and artifact sources.](#)
- [Add stages, dependencies, & conditions.](#)

# Release artifacts and artifact sources

3/6/2021 • 21 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## NOTE

This topic covers classic release pipelines. To understand artifacts in YAML pipelines, see [artifacts](#).

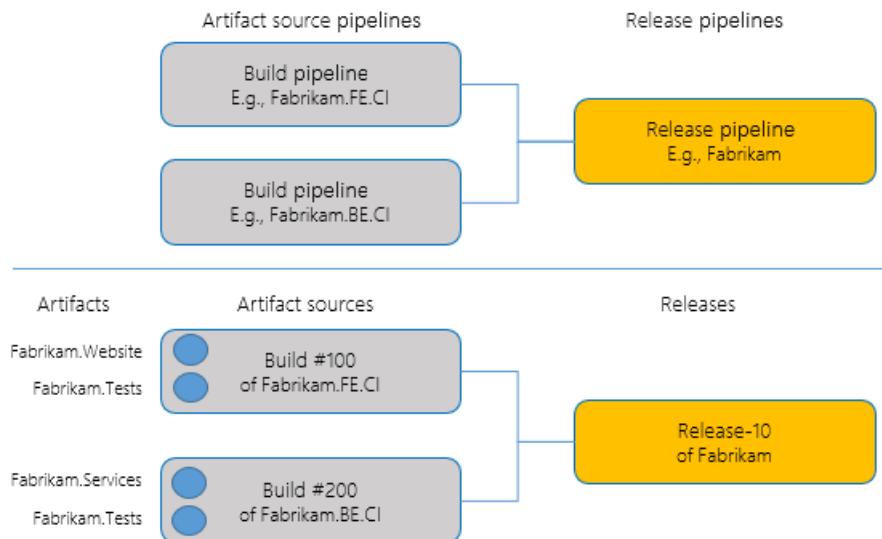
A release is a collection of artifacts in your DevOps CI/CD processes. An **artifact** is a deployable component of your application. Azure Pipelines can deploy artifacts that are produced by a [wide range of artifact sources](#), and stored in different types of artifact repositories.

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 information, 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 information, 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 information, 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 Azure Artifacts 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 Refresh

**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

**Mark primary**

#### IMPORTANT

The Artifacts **Default version** drop down list items depends on the **repository type** of the linked build definition.

- The following options are supported by all the repository types: **Specify at the time of release creation**, **Specific version**, and **Latest**.
- Latest from a specific branch with tags** and **Latest from the build pipeline default branch with tags** options are supported by the following repository types: **TfsGit**, **GitHub**, **Bitbucket**, and **GitHubEnterprise**.
- Latest from the build pipeline default branch with tags** is not supported by **XAML** build definitions.

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 \(NuGet, Maven, npm, Python, and Universal Packages\)](#)
- [External or on-premises TFS](#)
- [TeamCity](#)
- [Other sources](#)

## Artifact 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 <a href="#">Continuous Deployment</a> 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 <a href="#">artifact variables</a> 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 Other 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 <a href="#">skip the download</a> of artifacts.
Deployment section in build	The build summary includes a <b>Deployment</b> section, which lists all the stages to which the build was deployed.

By default, the releases execute in with a collection level Job authorization scope. That means releases can access

resources in all projects in the organization (or collection for Azure DevOps Server). This is useful when linking build artifacts from other projects. You can enable **Limit job authorization scope to current project for release pipelines** in project settings to restrict access to artifacts for releases in a project.

To set job authorization scope for the organization:

- Navigate to your organization settings page in the Azure DevOps user interface.
- Select Settings under Pipelines.
- Turn on the toggle *Limit job authorization scope to current project for release pipelines* to limit the scope to current project. This is the recommended setting, as it enhances security for your pipelines.

To set job authorization scope for a specific project:

- Navigate to your project settings page in the Azure DevOps user interface.
- Select Settings under Pipelines.
- Turn on the toggle Limit job authorization scope to current project to limit the scope to project. This is the recommended setting, as it enhances security for your pipelines.

**NOTE**

If the scope is set to project at the organization level, you cannot change the scope in each project.

All jobs in releases run with the job authorization scope set to collection. In other words, these jobs have access to resources in all projects in your project collection.

## Artifact sources - 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 checkout 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 <a href="#">Triggers</a> .
Artifact variables	A number of <a href="#">artifact variables</a> 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 <a href="#">skip the download</a> of artifacts.

By default, the releases execute in with a collection level Job authorization scope. That means releases can access all repositories in the organization (or collection for Azure DevOps Server). You can enable **Limit job authorization scope to current project for release pipelines** in project settings to restrict access to artifacts for releases in a project.

## Artifact sources - Jenkins

To consume Jenkins artifacts, you must create a service connection with credentials to connect to your Jenkins server. For more information, 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 <a href="#">Triggers</a> .
Artifact variables	A number of <a href="#">artifact variables</a> 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 <a href="#">skip the download</a> 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](#).

## Artifact sources - 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 information, 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 <a href="#">Triggers</a> .
Artifact variables	A number of <a href="#">artifact variables</a> 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 <a href="#">skip the download</a> of artifacts.

**NOTE**

In the case of continuous deployment from multiple artifact sources (multiple registries/repositories), it isn't possible to map artifact sources to trigger particular stages. A release will be created anytime there is a push to any of the artifact sources. If you wish to map an artifact source to trigger a specific stage, the recommended way is to decompose the release pipeline into multiple release pipelines.

## Artifact sources - Azure Artifacts

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 <a href="#">Triggers</a> .
Artifact variables	A number of <a href="#">artifact variables</a> 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 <a href="#">skip the download</a> of artifacts.

## Handling Maven Snapshots

When obtaining Maven artifacts and the artifact is a snapshot build, multiple versions of that snapshot may be downloaded at once (e.g. `myApplication-2.1.0.BUILD-20190920.220048-3.jar`, `myApplication-2.1.0.BUILD-20190820.221046-2.jar`, `myApplication-2.1.0.BUILD-20190820.220331-1.jar`). You will likely need to add additional automation to keep only the latest artifact prior to subsequent deployment steps. This can be accomplished with the following PowerShell snippet:

```
Remove all copies of the artifact except the one with the lexicographically highest value.
Get-Item "myApplication*.jar" | Sort-Object -Descending Name | Select-Object -SkipIndex 0 | Remove-Item
```

For more information, see the [Azure Artifacts](#) overview.

## Artifact sources - 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.

### TIP

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
---------	------------------------------------

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 <a href="#">artifact variables</a> are supported for external TFS sources.
Work items and commits	Azure Pipelines cannot show work items or commits for external TFS sources.
Artifact download	By default, External TFS artifacts are downloaded to the agent. You can configure an option in the stage to <a href="#">skip the download</a> of artifacts.

#### NOTE

Azure Pipelines may not be able to contact an on-premises TFS server in case it's within your enterprise network. In that 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 include those variables in 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, enter the **Build ID** (not the build number) of the desired build in the appropriate field, or select the **Latest** build.

## Artifact sources - 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 <a href="#">artifact variables</a> are supported for builds from TeamCity.
Work items and commits	Azure Pipelines cannot show work items or commits for TeamCity builds.

FEATURE	BEHAVIOR WITH TEAMCITY SOURCES
Artifact download	By default, TeamCity builds are downloaded to the agent. You can configure an option in the stage to <a href="#">skip the download</a> 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.

## Artifact sources - Custom artifacts

In addition to built-in artifact sources, Azure Artifacts supports integrating any custom artifact source with the artifact extensibility model. You can plug in any custom artifact source, and Azure DevOps will provide a first-class user experience and seamless integration.

For more information, see [Azure DevOps artifact extensibility model](#).

## Artifact sources - 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 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.

## 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 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.

Azure Pipelines currently 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' interface for a build named 'HotelFinder-Cl'. At the top, there's a header with the build name, 'Latest' (which is underlined), and 'Selected 1/3 artifact(s)'. Below the header, a message says 'Following are the artifacts published in the latest version of the build.' There's a checkbox labeled 'Select all artifacts'. Underneath, there are three artifact items: 'TestFile' (unchecked), 'SeleniumTests' (unchecked), and 'Web' (checked). Below the artifact list, there are three sections for different users: 'thelisland' (Latest, Selected all artifacts), 'Fabrikam' (Specify at the time of release creation, Selected all artifacts), and another user whose profile picture is not visible.

## 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](#).

## Additional information

- [Code repo sources in Azure Pipelines](#)
- [Jenkins artifacts in Azure Pipelines](#)
- [TeamCity extension for Continuous Integration](#)
- [External TFS extension for Release Management](#)

## Related topics

- [Release pipelines](#)
- [Stages](#)

## Help and support

- See our [troubleshooting](#) page
- Get advice on [Stack Overflow](#), and feel free to post your questions, search for answers, or suggest a feature on our [Azure DevOps Developer Community](#). [Support](#) page.

# Set up Azure Pipelines and Maven

3/6/2021 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

This guide covers the basics of using Azure Pipelines to work with Maven artifacts in Azure Artifacts feeds.

## Set up your feed

1. Select **Build and Release**.
2. Select **Packages**.
3. With your feed selected, select **Connect to feed**.
4. Select **Maven**.

Connect to feed

The screenshot shows the 'Connect to feed' dialog in Azure Pipelines. On the left, there are icons for NuGet, npm, Maven (which is highlighted with a red box), and Gradle. The right side has two main sections: 'Consume artifacts using Maven' and 'Publish artifacts using Maven'. In the 'Consume artifacts using Maven' section, there is a button labeled 'Generate Maven credentials' with a red box around it. Below it, there is sample XML code for pom.xml. In the 'Publish artifacts using Maven' section, there is a command line input field containing 'mvn deploy'.

5. Click on "Generate Maven Credentials"

6. Create a local file named `settings.xml` from the following template and then paste the generated XML, replacing the comment:

## IMPORTANT

Do not commit this file into your repository.

```

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
 https://maven.apache.org/xsd/settings-1.0.0.xsd">
 <servers>
 <!-- Paste the <server> snippet generated by Azure DevOps here -->
 </servers>
</settings>

```

7. Below the `settings.xml` snippet in the generated credentials dialog, there is a snippet to be added to the `<repositories>` section of your project's `pom.xml`. Add that snippet. If you intend to use Maven to publish to Artifacts, add the snippet to the `<distributionManagement>` section of the POM file as well. Commit and push this change.

8. Upload `settings.xml` created in step 3 as a **Secure File** into the pipeline's library.

9. Add tasks to your pipeline to [download the secure file](#) and to copy it to the `(~/m2)` directory. The latter can be accomplished with the following PowerShell script, where `settingsxml` is the reference name of the "Download secure file" task:

```

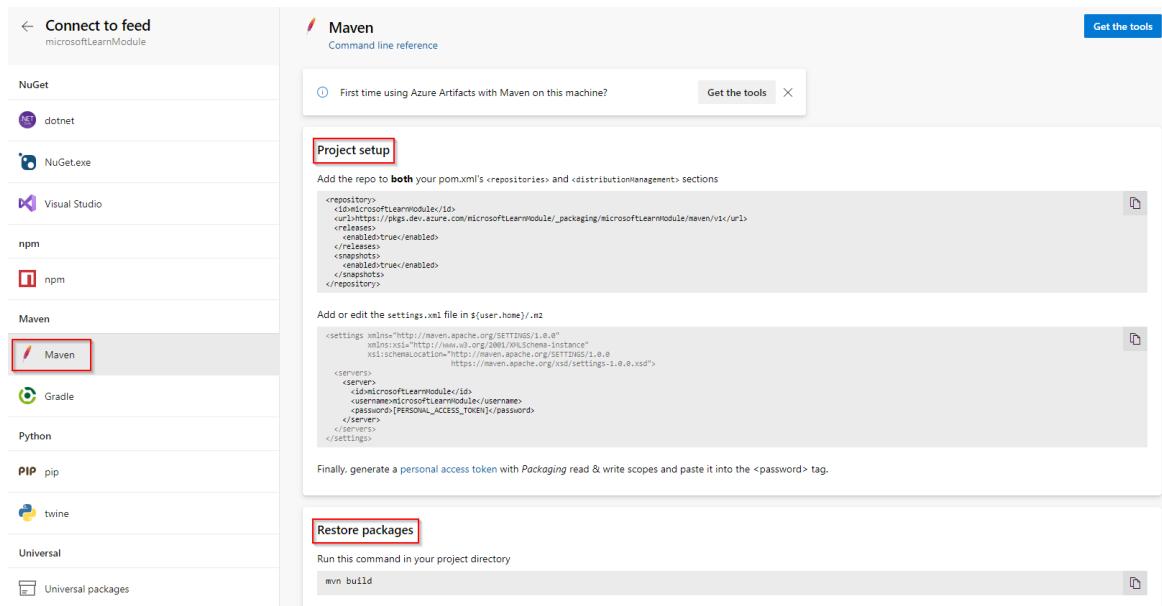
New-Item -Type Directory -Force "${HOME}/.m2"
Copy-Item -Force "$(settingsxml.secureFilePath)" "${HOME}/.m2/settings.xml"

```

1. Navigate to **Artifacts**.

2. With your feed selected, select **Connect to feed**.

3. Select **Maven**.



4. Set up your project by following these steps:

- Add the repo to **both** your `pom.xml`'s `<repositories>` and `<distributionManagement>` sections. Replace the `[ORGANIZATION_NAME]` placeholder with your own organization.

```
<repository>
 <id>[ORGANIZATION_NAME]</id>

 <url>https://pkgs.dev.azure.com/[ORGANIZATION_NAME]/_packaging/[ORGANIZATION_NAME]/maven/v1</u
rl>
 <releases>
 <enabled>true</enabled>
 </releases>
 <snapshots>
 <enabled>true</enabled>
 </snapshots>
</repository>
```

- b. Add or edit the settings.xml file in \${user.home}/.m2. Replace the [ORGANIZATION\_NAME] placeholder with your own organization.

```
<server>
 <id>[ORGANIZATION_NAME]</id>
 <username>[ORGANIZATION_NAME]</username>
 <password>[PERSONAL_ACCESS_TOKEN]</password>
</server>
```

- c. Generate a [Personal Access Token](#) with Packaging read & write scopes and paste it into the <password> tag.

#### IMPORTANT

In order to automatically authenticate Maven feeds from Azure Artifacts, you must have the mavenAuthenticateFeed argument set to true in your Maven task. See [Maven build task](#) for more information.

## Restore your package

Run this command in your project directory to restore your package.

```
mvn build
```

## Publish your package

Run this command in your project directory to publish your package.

```
mvn deploy
```

# Publish npm packages (YAML/Classic)

3/6/2021 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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/>.
- [YAML](#)
- [Classic](#)

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.

```
- task: Npm@1
 inputs:
 command: publish
 publishRegistry: useFeed
 publishFeed: projectName/feedName
```

- **useFeed**: this option allows the use of an Azure Artifacts feed in the same organization as the build.
- **feedName**: the name of the feed you want to publish to.
- **projectName** the name of your project.

## NOTE

All new feeds that were created through the classic user interface are project scoped feeds. You must include the project name in the `publishFeed` parameter: `publishFeed: '<projectName>/<feedName>'`. See [Project-scoped feeds vs. Organization-scoped feeds](#) to learn about the difference between the two types.

To publish to an external npm registry, you must first create a service connection to point to that feed. You can do this by going to **Project settings**, selecting **Services**, and then creating a **New service connection**. Select the **npm** option for the service connection. Fill in the registry URL and the credentials to connect to the registry. See [Service connections](#) to learn more about how to create, manage, secure, and use a service connection.

To publish a package to an 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>'
```

- **publishEndpoint:** This argument is required when `publishRegistry == UseExternalRegistry`. Copy and paste the name of the service connection you created earlier.

For a list of other options, see the [npm task](#) to install and publish your npm packages, or run an npm command.

YAML is not supported in TFS.

#### NOTE

Ensure that your working folder has an `.npmrc` file with a `registry=` line, as described in the [Connect to feed](#) screen in your feed.

The build does not support using the `publishConfig` property to specify the `registry` to which you're publishing. The build will fail, potentially with unrelated authentication errors, if you include the `publishConfig` property in your `package.json` configuration file.

## FAQ

- **Where can I learn about the Azure Pipelines and TFS Package Management service?**

Check out the [Azure Artifacts landing page](#) for details about Artifacts in Azure Pipelines.

- **How to publish packages to my feed from the command line?**

See [Publish your package to an npm feed using the CLI](#) for more information.

- **How to create a token that lasts longer than 90 days?**

See [Set up your client's npmrc](#) for more information on how to set up authentication to Azure Artifacts feeds.

- **Do you recommend using scopes or upstream sources?**

We recommend using upstream sources because it gives you the most flexibility to use a combination of scoped- and non-scoped packages in your feed, as well as scoped- and non-scoped packages from npmjs.com.

See [Use npm scopes](#) and [Use packages from npmjs.com](#) for more details.

# Publish to NuGet feeds (YAML/Classic)

3/6/2021 • 8 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

You can publish NuGet packages from your build to NuGet feeds using the Pipeline tasks as well as the Classic user interface. 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.

## Create a NuGet package

There are various 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 and [publish your packages](#). Otherwise, add a [NuGet task](#):

- [YAML](#)
- [Classic](#)

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. The following list describes some of the key ones. The [task documentation](#) describes the rest.

- **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 `$(BuildConfiguration)` unless you want to always build either `Debug` or `Release` packages, or unless you have a custom build configuration.
- **packDestination**: The default is `$(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 three numeric components,

`Major.Minor.Patch`.

When you fix a bug, you increment the patch (`1.0.0` to `1.0.1`). When you release a new backward-compatible feature, you increment the minor version and reset the patch version to 0 (`1.4.17` to `1.5.0`). When you make a backward-incompatible change, you increment the major version and reset the minor and patch versions to 0 (`2.6.5` to `3.0.0`).

In addition to `Major.Minor.Patch`, Semantic Versioning provides for a prerelease label. Prerelease labels are a hyphen (-) 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 continuous integration (CI), you can use Semantic Versioning with prerelease labels. You can use the **NuGet** task for this purpose. It supports the following formats:

- Use the same versioning scheme for your builds and packages, if 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).$(rev:.r)`, where `Major` and `Minor` are two variables defined in the build pipeline. This format will automatically increment the build number and the package version with a new patch number. It will keep the major and minor versions constant, until you change them manually in the build pipeline.
  - `$(Major).$(Minor).$(Patch).$(date:yyyyMMdd)`, where `Major`, `Minor`, and `Patch` are variables defined in the build pipeline. This format will create a new prerelease label for the build and package while keeping the major, minor, and patch versions constant.
- Use a version that's 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](#)
- [Classic](#)

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.

Although 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

after packages are produced, they're [immutable](#). They can't be updated or replaced.

When you're producing a package in a 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 toward that release. Although none of the following solutions are ideal, you can use one of these depending on your preference:

- After 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 might uncover new issues.
- Publish only packages that you want to release. In this case, you won't use a prerelease label for every build. Instead, you'll reuse the same package version for all packages. Because you do not publish packages from every build, you do not cause a conflict.

#### NOTE

Please note that `DotNetCore` and `DotNetStandard` packages should be packaged with the `DotNetCoreCLI@2` task to avoid `System.InvalidCastException`s. See the [.NET Core CLI task](#) for more details.

```
task: DotNetCoreCLI@2
displayName: 'dotnet pack $(buildConfiguration)'
inputs:
 command: pack
 versioningScheme: byPrereleaseNumber
 majorVersion: '$(Major)'
 minorVersion: '$(Minor)'
 patchVersion: '$(Patch)'
```

## Publish your packages

In the previous section, you learned how to create a package with every build. When you're ready to share the changes to your package with your users, you can publish it.

- [YAML](#)
- [Classic](#)

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: NuGetAuthenticate@0
 displayName: 'NuGet Authenticate'
- task: NuGetCommand@2
 displayName: 'NuGet push'
 inputs:
 command: push
 publishVstsFeed: '< projectName >/< feed >'
 allowPackageConflicts: true
```

#### NOTE

Artifact feeds that were created through the classic user interface are project scoped feeds. You must include the project name in the `publishVstsFeed` parameter: `publishVstsFeed: '< projectName >/< feed >'`. See [Project-scoped feeds vs. Organization-scoped feeds](#) to learn about the difference between the two types.

To publish to an external NuGet feed, you must first create a service connection to point to that feed. You can do this by going to **Project settings**, selecting **Service connections**, and then creating a **New service connection**. Select the **NuGet** option for the service connection. To connect to the feed, fill in the feed URL and the API key or token.

To publish a package to a NuGet feed, add the following snippet to your `azure-pipelines.yml` file.

```
- task: NuGetAuthenticate@0
 inputs:
 nuGetServiceConnections: '<Name of the NuGet service connection>'
- task: NuGetCommand@2
 inputs:
 command: push
 nuGetFeedType: external
 versioningScheme: byEnvVar
 versionEnvVar: <VersionVariableName>
```

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](#).

## FAQ

### 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

6/10/2021 • 2 minutes to read • [Edit Online](#)

You can publish Python packages produced by your build to:

- Azure Artifacts
- Other repositories such as <https://pypi.org/>

To publish Python packages produced by your build, you'll use `twine`, a widely used tool for publishing Python packages. This guide covers how to do the following in your pipeline:

1. Install `twine` on your build agent
2. Authenticate `twine` with your Azure Artifacts feeds
3. Use a custom task that invokes `twine` to publish your Python packages

## Install twine

First, you'll need to run `pip install twine` to ensure the build agent has `twine` installed.

- [YAML](#)
- [Classic](#)

```
- script: 'pip install twine'
```

Check out the [script YAML task reference](#) for the schema for this command.

## 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](#)
- [Classic](#)

To authenticate with `twine`, add the following snippet to your `azure-pipelines.yml` file.

The example below will enable you to authenticate to a list of Azure Artifacts feeds as well as a list of service connections from external organizations. If you need to authenticate to a single feed, you must replace the following arguments: `artifactFeeds` and `externalFeeds` with `artifactFeed` and `externalFeed` and specify your feed name accordingly.

```
- task: TwineAuthenticate@1
 inputs:
 artifactFeeds: 'feed_name1, feed_name2'
 externalFeeds: 'feed_name1, feed_name2'
```

- **artifactFeeds**: a list of Azure Artifacts feeds within your organization. If you only have one Azure Artifacts feed, use **artifactFeed** (singular) instead.
- **externalFeeds**: a list of [service connections](#) from external organizations including PyPI or feeds in other

organizations in Azure DevOps.

**TIP**

The authentication credentials written into the `PYPIRC_PATH` environment variable supersede those in your .ini and .conf files.

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.

## Publish Python packages to Azure Artifacts feeds

After you've set up authentication with the `TwineAuthenticate@1` task, you can now use `twine` to publish your Python packages to an Azure Artifacts feed.

```
- script: |
 pip install wheel
 pip install twine

- script: |
 python setup.py bdist_wheel

- task: TwineAuthenticate@1
 displayName: 'Twine Authenticate'
 inputs:
 artifactFeed: projectName/feedName

- script: |
 python -m twine upload -r feedName --config-file $(PYPIRC_PATH) dist/*.whl
```

**WARNING**

We strongly recommend NOT checking any credentials or tokens into source control.

# Publish symbols for debugging

11/2/2020 • 4 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## 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

To publish symbols to the symbol server in Azure Artifacts, include the [Index Sources and Publish Symbols](#) task in your build pipeline. Configure the task as follows:

- For **Version**, select `2.\*`.
- For **Version**, select `1.\*`.
- For **Symbol Server Type**, select **Symbol Server in this organization/collection (requires Azure Artifacts)**.
- 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 preceding 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 by 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 it as follows:

- For **Version**, select 2.\\*.
- For **Symbol Server Type**, select **File share**.
  - When you select **File share** as **Symbol Server Type**, you get the **Compress Symbols** option. This option compresses 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 still need to use the **Index Sources and Publish Symbols** task to publish symbols. For portable PDBs, the build does the indexing, however you should [use SourceLink](#) to index the symbols as part of the build. Note that Azure Artifacts doesn't presently support ingesting NuGet symbol packages and so the task is used to publish the generated PDB files into the symbols service directly.

## 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 development machine

In Visual Studio, you might need to enable the following two options in **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 might be required if the collection URL has changed. When you're 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 (SRCSRVRTRG) where the debugger can find the retrieved source file.

The mapping information might look something like the following:

```

SRCRV: variables -----
TFS_EXTRACT_TARGET=%targ%\%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%
SRCRV: 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 preceding example contains two sections: the variables section and the source files section. The information in the variables section can be overridden. The variables can use other variables, and can use 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\srvcsrv.ini`. Set the content of the .ini file to override the variables. For example:

```
[variables]
TFS_COLLECTION=http://DIFFERENT_SERVER:8080/tfs/DifferentCollection
```

#### IMPORTANT

If you want to delete symbols that were published using the [Index Sources & Publish Symbols](#) task, you must first remove the build that generated those symbols. This can be accomplished by [using retention policies to clean up your build](#) or by [manually deleting the run](#). For more information about debugging your app, see [Debug with symbols in Visual Studio](#), and [Debug with symbols in WinDbg](#).

## FAQ

**Q: What's the retention policy for the symbols stored in the Azure Pipelines symbol server?**

A: Symbols have the same retention as the build. When you delete a build, you also delete the symbols that the build produced.

**Q: Can I use source indexing on a portable .pdb file created from a .NET Core assembly?**

A: No, source indexing is currently not enabled for portable .pdb files because SourceLink doesn't support authenticated source repositories. The workaround at the moment is to configure the build to generate full .pdb files.

**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

5/7/2021 • 5 minutes to read • [Edit Online](#)

## Azure Pipelines

Using Universal Packages you can pack any number of files of any type and share it with your team. Using the Universal Package task, you can pack, publish, and download packages of various sizes up to 4 TB. Each package will be uniquely identified with a name and a version number. Packages can be published and consumed to and from Artifacts feeds by using the Azure CLI or Azure Pipelines.

### NOTE

Universal Packages are only available for Azure DevOps Services.

## Prepare a Universal Package

By default, the Universal Packages task uses the `$(Build.ArtifactStagingDirectory)` as the publish directory. To prepare your Universal Package for publishing, place the files you want to publish in that directory. You can also use the [Copy Files](#) utility task to copy those files to the publish directory.

## Publish a Universal Package

- [YAML](#)
- [Classic](#)

To publish a Universal Package to your Artifacts feed, add the following task to your pipeline's yaml file.

```
Publish a Universal Package
- task: UniversalPackages@0
 displayName: Universal Publish
 inputs:
 command: publish
 publishDirectory: '$(Build.ArtifactStagingDirectory)'
 vstsFeedPublish: '<projectName>/<feedName>'
 vstsFeedPackagePublish: '<Package name>'
 packagePublishDescription: '<Package description>'
```

ARGUMENT	DESCRIPTION
publishDirectory	Location of the files to be published.
vstsFeedPublish	The project and feed name to publish to.
vstsFeedPackagePublish	The package name. Must be lower case. Use only letters, numbers, and dashes.
packagePublishDescription	Description of the content of the package.

To publish packages to an Azure Artifacts feed from your pipeline, you must add the **Project Collection Build Service** identity as a **Contributor** from the feed's settings. See [Adding users/groups permissions to a feed](#) for details.

To publish to an external feed, you must first create a service connection to point to that feed. see [Manage service connection](#) for details.

## Package versioning

Universal Packages follow the semantic versioning spec and are identified by their names and version numbers. Semantic version numbers have three numeric components, Major, Minor, and Patch: `Major.Minor.Patch`.

When you release a new backward-compatible feature, you increment the minor version and reset the patch version to 0 (`1.4.17` to `1.5.0`), and when you make a backward-incompatible change, you increment the major version and reset the minor and patch versions to 0 (`2.6.5` to `3.0.0`). The patch version number should be incremented in the case of fixing a bug (`1.0.0` to `1.0.1`).

The Universal Packages task automatically selects the next major, minor, or patch version for you when you publish a new package.

- [YAML](#)
- [Classic](#)

To enable versioning for your package, add a `versionOption` input to your YAML file. The options for publishing a new package version are: `major`, `minor`, `patch`, or `custom`.

Selecting `custom` enables you to manually specify your package version. The other options will get the latest package version from your feed and increment the chosen version segment by 1. So if you have a *testPackage 1.0.0*, and select the *major* option, your new package will be *testPackage 2.0.0*. If you select the *minor* option, your package version will be *1.1.0*, and if you select the *patch* option, your package version will be *1.0.1*.

One thing to keep in mind is that if you select the `custom` option, you must also provide a `versionPublish` as follows.

```
- task: UniversalPackages@0
 displayName: Universal Publish
 inputs:
 command: publish
 publishDirectory: '$(Build.ArtifactStagingDirectory)'
 vstsFeedPublish: '<projectName>/<feedName>'
 vstsFeedPackagePublish: '<Package name>'
 versionOption: custom
 versionPublish: '<Package version>'
 packagePublishDescription: '<Package description>'
```

ARGUMENT	DESCRIPTION
<code>publishDirectory</code>	Location of the files to be published.
<code>vstsFeedPublish</code>	The project and feed name to publish to.
<code>vstsFeedPackagePublish</code>	The package name.
<code>versionOption</code>	Select a version increment strategy. Options: <code>major</code> , <code>minor</code> , <code>patch</code> , <code>custom</code>

ARGUMENT	DESCRIPTION
versionPublish	The custom package version
packagePublishDescription	Description of the content of the package.

## Download a Universal Package

- [YAML](#)
- [Classic](#)

To download a Universal Package from a feed in your organization, use the Universal package task with the `download` command input as follows:

```
steps:
- task: UniversalPackages@0
 displayName: 'Universal download'
 inputs:
 command: download
 vstsFeed: '<projectName>/<feedName>'
 vstsFeedPackage: '<packageName>'
 vstsPackageVersion: '<packageVersion>'
 downloadDirectory: '$(Build.SourcesDirectory)\someFolder'
```

ARGUMENT	DESCRIPTION
vstsFeed	The Artifacts feed hosting the package to be downloaded.
vstsFeedPackage	Name of the package to be downloaded.
vstsPackageVersion	Version of the package to be downloaded.
downloadDirectory	The package destination folder. Default value: \$(System.DefaultWorkingDirectory).

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	Set the value to <code>external</code> when downloading from an external source.

ARGUMENT	DESCRIPTION
externalFeedCredentials	Name of the service connection pointing to the external feed. See <a href="#">service connections</a> for details.
feedDownloadExternal	Name of the external feed.
packageDownloadExternal	The package name to be downloaded.
versionDownloadExternal	The version of the package to be downloaded.

#### NOTE

You can use wildcards in the string you pass to the download command to download the latest version of a Universal Package. See [Universal Packages quickstart](#) for more details.

## Next Steps

- [Publish and download artifacts in Azure Pipelines](#)
- [Release artifacts and artifact sources](#)

# Restore NuGet packages in Azure Pipelines

5/18/2021 • 5 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

NuGet package restore allows you to have all your project's dependencies available without having to store them in source control. This allows for a cleaner development environment and smaller repository size. You can restore your NuGet packages using the NuGet restore build task, the NuGet CLI, or the .NET Core CLI. This article will show you how to restore your NuGet packages using both YAML and the classic Azure Pipelines.

## Prerequisites

- [Set up your solution](#) to consume packages from Azure Artifacts feed.
- [Created your first pipeline](#) for your repository.
- [Set up the build identity permissions](#) for your feed.

## Restore packages with NuGet restore build task

To build a solution that relies on NuGet packages from Azure Artifacts feeds, we will want to add the **NuGet** build task to our pipeline.

1. Navigate to your build pipeline and select **Edit**.
2. Under **Tasks**, **Agent job**, select the plus sign "+" to add a new task. Search for **NuGet** task and add it to your agent job.
3. Fill out the following information:
  - **Display name:** NuGet restore.
  - **Command:** restore.
  - **Path to solution, packages.config, or project.json:** The path to the solution, packages.config, or project.json file that references the packages to be restored.
4. If you've checked in a [NuGet.config](#), select **Feeds in my NuGet.config** and specify the file from your repository. If you're using a single Azure Artifacts feed, select the **Feed(s) I select here** option and select your feed from the dropdown.
5. Check the **Use packages from NuGet.org** option if you want to include NuGet.org in the generated NuGet.config.
6. Select **Save & queue**.

The screenshot shows the Azure DevOps Pipeline editor. On the left, a sidebar lists various tasks: Get sources, Agent job 1, Use NuGet 4.4.1, NuGet restore (which is currently selected and highlighted in blue), Build solution, Test Assemblies, Publish symbols path, and Publish Artifact. The main pane displays the configuration for the selected 'NuGet restore' task. At the top right are links for 'Link settings', 'View YAML', and 'Remove'. Below that, the 'Task version' is set to '2.\*'. The 'Display name' is 'NuGet restore'. The 'Command' dropdown contains 'restore'. The 'Path to solution, packages.config, or project.json' field is set to '\*\*\\*.sln'. Under 'Feeds and authentication', the 'Feed(s) I select here' radio button is selected. There is also an option to 'Use packages from this Azure Artifacts/TFS feed'. At the bottom, there are sections for 'Advanced', 'Control Options', and 'Output Variables'.

## Restore your NuGet packages with the NuGet CLI

In order for your project to be set up properly, your `nuget.config` must be in the same folder as your `.csproj` or `.sln` file. The `nuget.config` file you check-in also 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="FabrikamFiber"
value="https://pkgs.dev.azure.com/microsoftLearnModule/_packaging/FabrikamFiber/nuget/v3/index.json" />
 <!-- also get packages from the NuGet Gallery -->
 <add key="nuget.org" value="https://api.nuget.org/v3/index.json" protocolVersion="3" />
 </packageSources>
</configuration>
```

To restore your NuGet packages run the following command in your project directory:

```
nuget.exe restore
```

## Restore NuGet packages with the .NET Core CLI task

To restore your package using YAML and the [.NET Core CLI task](#), use the following example:

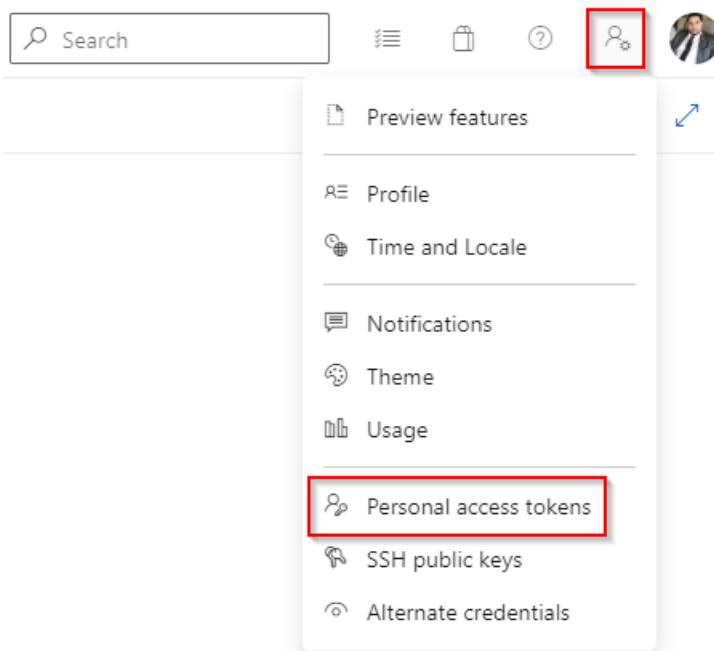
```
- task: DotNetCoreCLI@2
 displayName: dotnet restore
 inputs:
 command: restore
 projects: '**/*.csproj'
 feedsToUse: 'select'
 vstsFeed: '<projectName>/<feedName>'
 includeNuGetOrg: true
```

- `command` : The dotnet command to run. Options: `build` , `push` , `pack` , `restore` , `run` , `test` , and `custom` .
- `projects` : The path to the csproj file(s) to use. You can use wildcards (e.g. `**/*.csproj` for all `.csproj` files in all subfolders).
- `feedsToUse` : You can either choose to select a feed or commit a NuGet.config file to your source code repository and set its path using `nugetConfigPath` . Options: `select` , `config` .
- `vstsFeed` : This argument is required when `feedsToUse` == `Select` . Value format: `<projectName>/<feedName>` .
- `includeNuGetOrg` : Use packages from NuGet.org.

## Restore NuGet packages from feeds in a different organization

If your NuGet.config contains feeds in a different Azure DevOps organization than the one running the build, you'll need to set up credentials for those feeds manually.

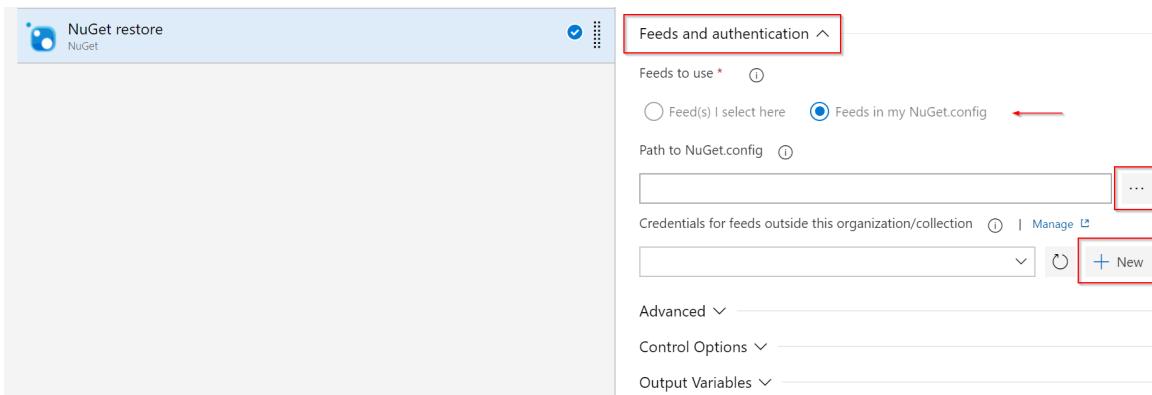
1. Select an account (either a service account (recommended) or a user account) that has access to the remote feed.
2. In your browser, open a Private mode, Incognito mode, or a similar mode window and navigate to the Azure DevOps organization that hosts the feed. Sign in with the credentials mentioned in step 1, select **User settings** then **Personal Access Tokens**.



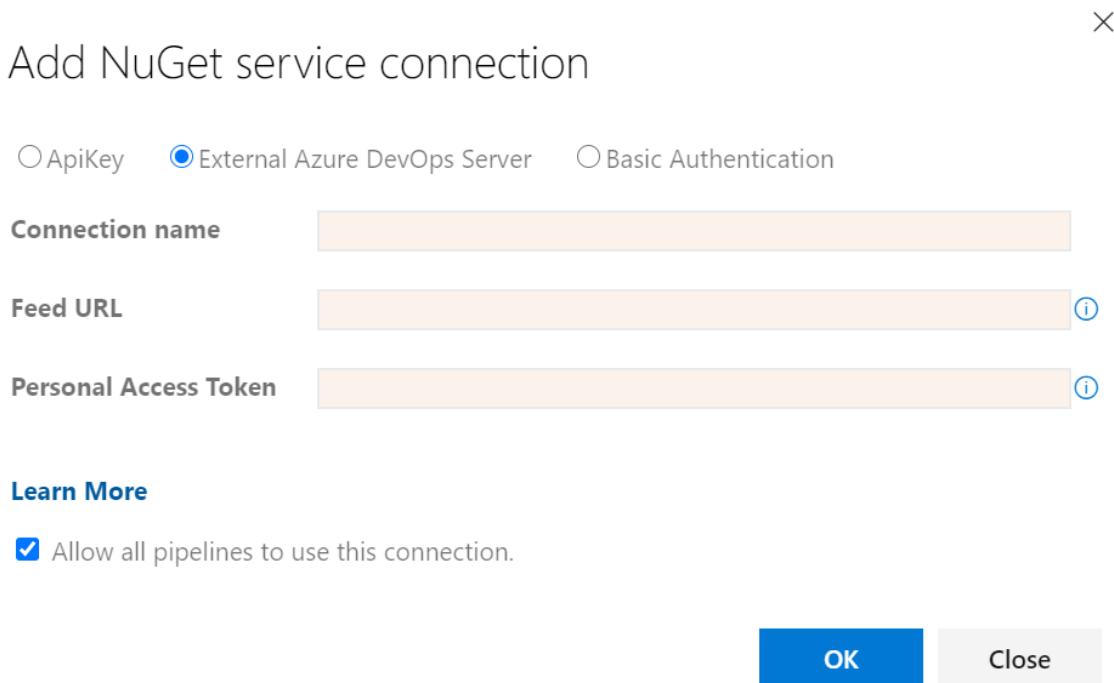
3. Create your PAT with the **Packaging (read)** scope and keep it handy.
4. In the Azure DevOps organization that contains the build, edit the build's NuGet step and ensure you're using version 2 or greater of the task, using the version selector.



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**, select the **+ New**.



8. In the service connection dialog that appears, select the **External Azure DevOps Server** option and enter a connection name, the feed URL (make sure it matches what's in your NuGet.config) and the PAT you created in step 3.



9. Save & queue a new build.

## FAQ

### Why can't my build restore NuGet 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](https://nuget.org).

The screenshot shows the Azure DevOps interface for a build definition named 'FabrikamFiber-Cl'. The 'Tasks' tab is selected. On the left, a list of tasks includes 'Get sources', 'Use NuGet 4.3.0' (which is currently selected), and 'NuGet restore'. A '+' button is available to add more tasks. To the right, the 'Use NuGet 4.3.0' task is detailed. It has a 'Version' dropdown set to '0.\*'. The 'Display name' is 'Use NuGet 4.3.0'. The 'Version of NuGet.exe to install' is set to '4.3.0'. There is also a checked checkbox for 'Always download the latest matching version'. A 'Control Options' section is partially visible.

#### 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.

## Related articles

- [Publish to NuGet feeds \(YAML/Classic\)](#)
- [Publish and consume build artifacts](#)
- [How to mitigate risk when using private package feeds](#)

# Use Jenkins to restore and publish packages

11/2/2020 • 4 minutes to read • [Edit Online](#)

Azure Artifacts | TFS 2018 | TFS 2017

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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:

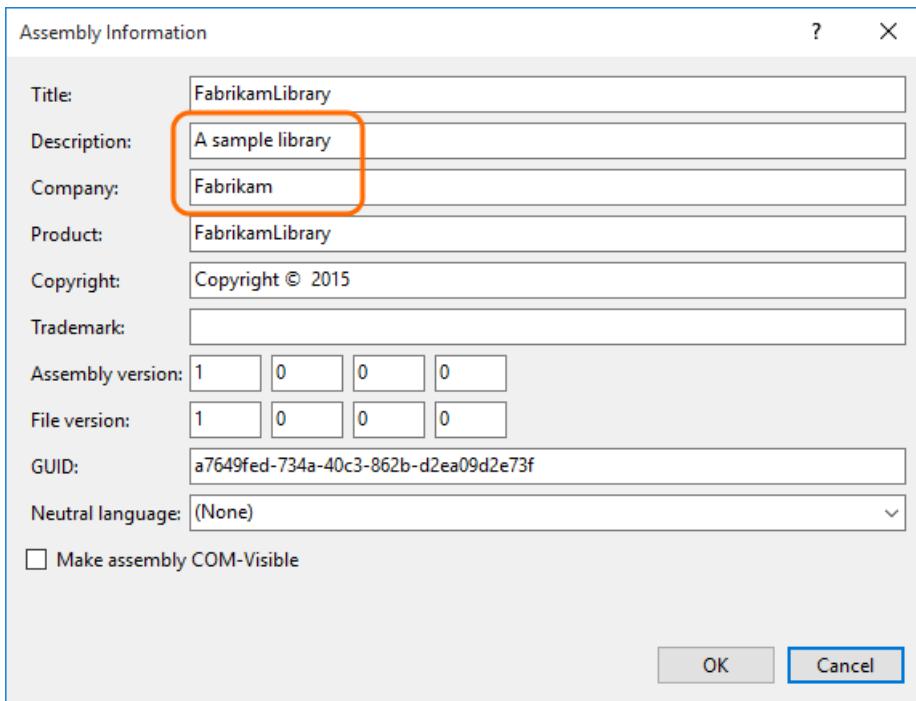
- [MSBuild 1.24](#)
- [Git 2.4.0](#)
- [Git Client 1.19.0](#)
- [Credentials Binding plugin 1.6](#)

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="https://www.myget.org/F/vss-package-management/api/v2" />
6 <add key="MyGreatFeed" value="https://fabrikam.pkgs.visualstudio.com/DefaultCollection/_apis/packaging/MyGreatFeed/nuget/index.json" />
7 </packageSources>
8 <activePackageSource>
9 <add key="All" value="(Aggregate source)" />
10 </activePackageSource>
11 </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  
Copy from

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

Add Cancel Advanced...

#### 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)



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

Pass build variables as properties

Advanced...

Delete

3

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\nuget sources update -Name "MyGreatFeed" -UserName "%FEEDUSER%"
-Password "%FEEDPASS%"
.tools\VSS.NuGet\nuget 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.

# About pipeline resources

6/7/2021 • 5 minutes to read • [Edit Online](#)

A [resource](#) is anything used by a pipeline that lives outside the pipeline. Resources are:

- a way to share something such as a secure file or password across pipelines
- a tool for enhancing security through access checks and other restrictions
- a way to improve traceability for your pipeline and make it easier to troubleshoot [environments](#)

Resources are a way to share information across pipelines. Good examples of using resources for sharing are variable groups, secure files, and service connections. In all these cases, you're using a resource as a way for a pipeline to access and consume something.

Resources are also useful for restricting access and making sure your valuable information is shared with the fewest number of authorized users or pipelines. For example, you can limit a service connection to only run on one pipeline. You could also make sure that a repository can only be accessed from a pipeline after a manual approach check.

You can also use resources for better traceability. When you use resources with environments, you can trace the lifecycle of your resources. For example, you'll be able to see the number of the last run that was deployed to an [environment](#).

## Share resources across pipelines

You can share resources across pipelines by configuring resources within the pipelines UI and then referencing those resources in a task. You can also access some shared resources with the [resources](#) [YAML pipeline syntax](#).

Examples of sharing resources with the pipelines UI include [secure files](#), [variable groups](#), and [service connections](#). With the [resources](#) syntax, examples include accessing pipelines themselves, repositories, and packages.

The way you use a resource in a pipeline depends on the type of pipeline and the type of resource.

- [YAML](#)
- [Classic](#)

For YAML pipelines:

- Service connections and secure files are directly used as inputs to tasks. They do not have to be pre-declared.
- Variable groups use the [group](#) syntax.
- Pipelines and repositories use the [resources](#) syntax.

For example, to use variable groups in a pipeline, you'll first add your variables at [Pipelines > Library](#). Then, you can reference the variable group in your YAML pipeline with the [variables](#) syntax.

```
variables:
- group: my-variable-group
```

To call a second pipeline from your pipeline with the [resources](#) syntax, reference [pipelines](#).

```

resources:
 pipelines:
 - pipeline: SmartHotel-resource # identifier for the resource (used in pipeline resource variables)
 source: SmartHotel-CI # name of the pipeline that produces an artifact

```

## Use resources to enhance security

You can enhance your pipeline's security with resources by identifying how that resource gets consumed, and how to prevent an unauthorized access.

For YAML pipelines only, one way to restrict resources is by setting them as [protected or open](#). When a resource is protected, you can apply approvals and checks to it and limit access to specific users and pipelines. Protected resources include service connections, agent pools, environments, repositories, variable groups, and secure files.

RESOURCE	HOW IS IT CONSUMED?	HOW DO YOU PREVENT AN UNINTENDED PIPELINE FROM USING THIS?
<a href="#">service connections</a>	Consumed by tasks in a YAML file that use the service connection as an input.	Protected with checks and pipeline permissions. Checks and pipeline permissions are controlled by service connection users. A resource owner can control which pipelines can access an service connection. You can also use pipeline permissions to restrict access to a particular pipeline.
<a href="#">variable groups</a>	A special syntax exists for using variable groups in a pipeline or in a job. A variable group gets added like a service connection.	Protected with checks and pipeline permissions. Checks and pipeline permissions are controlled by variable group users. A resource owner can control which pipelines can access a variable group. You can also use pipeline permissions to restrict access to a particular pipeline.
<a href="#">secure files</a>	Secure files are consumed by tasks (example: <a href="#">Download Secure File task</a> ).	Protected with checks and pipeline permissions. Checks and pipeline permissions are controlled by secure files users. A resource owner can control which pipelines can access secure files. You can also use pipeline permissions to restrict access to a particular pipeline.
<a href="#">agent pools</a>	There is a special syntax to use an agent pool to run a job.	Protected with checks and pipeline permissions. Checks and pipeline permissions are controlled by agent pool users. A resource owner can control which pipelines can access an agent pool. You can also use pipeline permissions to restrict access to a particular pipeline.
<a href="#">environments</a>	There's a special syntax to use an environment in a YAML.	Protected with checks and pipeline permissions that are controlled by environment users. You can also use pipeline permissions to restrict access to a particular environment.

RESOURCE	HOW IS IT CONSUMED?	HOW DO YOU PREVENT AN UNINTENDED PIPELINE FROM USING THIS?
<a href="#">repositories</a>	A script can clone a repository if the job access token has access to the repo.	Protected with checks and pipeline permissions controlled by repository contributors. A repository owner can restrict ownership.
<a href="#">artifacts</a> , work items, <a href="#">pipelines</a>	<a href="#">Pipeline artifacts</a> are resources, but <a href="#">Azure Artifacts</a> are not. A script can download artifacts if the job access token has access to the feed. A pipeline artifact can be declared as a resource in the resources section – primarily for the intent of triggering the pipeline when a new artifact is available, or to consume that artifact in the pipeline.	Artifacts and work items are not protected. Checks and pipeline permissions for feeds are not supported.
<a href="#">containers</a> , <a href="#">packages</a> , <a href="#">webhooks</a>	These live outside the Azure DevOps ecosystem and access is controlled with service connections. There is a special syntax for using all three types in YAML pipelines.	Protected with checks and pipeline permissions controlled by service connection users.

## Use resources for traceability

- [YAML](#)
- [Classic](#)

For YAML pipelines, you can combine resources with environments to better track resource use. [Environments](#) are collections of resources using in pipelines.

Environments support two resource types:

- [Kubernetes](#)
- [Virtual machines](#)

# Resources in YAML

5/17/2021 • 24 minutes to read • [Edit Online](#)

## Azure Pipelines

A resource is anything used by a pipeline that lives outside the pipeline. Pipeline resources include:

- CI/CD pipelines that produce artifacts (Azure Pipelines, Jenkins, etc.)
- code repositories (Azure Repos Git repos, GitHub, GitHub Enterprise, Bitbucket Cloud)
- container image registries (Azure Container Registry, Docker Hub, etc.)
- package feeds (GitHub packages)

## Why resources?

Resources are defined at one place and can be consumed anywhere in your pipeline. Resources provide you the full traceability of the services consumed in your pipeline including the version, artifacts, associated commits, and work-items. You can fully automate your DevOps workflow by subscribing to trigger events on your resources.

Resources in YAML represent sources of types pipelines, builds, repositories, containers, and packages.

## Schema

```
resources:
 pipelines: [pipeline]
 builds: [build]
 repositories: [repository]
 containers: [container]
 packages: [package]
 webhooks: [webhook]
```

## Variables

When a resource triggers a pipeline, the following variables are set:

```
resources.triggeringAlias
resources.triggeringCategory
```

These values will be blank if a resource does not trigger a pipeline run. The variable `Build.Reason` must be `ResourceTrigger` for these values to get set.

## Resources: `pipelines`

If you have an Azure Pipeline that produces artifacts, you can consume the artifacts by defining a `pipelines` resource. `pipelines` is a dedicated resource only for Azure Pipelines. You can also set triggers on pipeline resource for your CD workflows.

In your resource definition, `pipeline` is a unique value that you can use to reference the pipeline resource later on. `source` is the name of the pipeline that produces an artifact.

For an alternative way to download pipelines, see tasks in [Pipeline Artifacts](#).

- [Schema](#)

- Example

```
resources: # types: pipelines | builds | repositories | containers | packages
pipelines:
- pipeline: string # identifier for the resource used in pipeline resource variables
 project: string # project for the source; optional for current project
 source: string # name of the pipeline that produces an artifact
 version: string # the pipeline run number to pick the artifact, defaults to latest pipeline successful
across all stages; Used only for manual or scheduled triggers
 branch: string # branch to pick the artifact, optional; defaults to all branches; Used only for manual
or scheduled triggers
 tags: [string] # list of tags required on the pipeline to pickup default artifacts, optional; Used
only for manual or scheduled triggers
 trigger: # triggers are not enabled by default unless you add trigger section to the resource
 branches: # branch conditions to filter the events, optional; Defaults to all branches.
 include: [string] # branches to consider the trigger events, optional; Defaults to all branches.
 exclude: [string] # branches to discard the trigger events, optional; Defaults to none.
 tags: [string] # list of tags to evaluate for trigger event, optional
 stages: [string] # list of stages to evaluate for trigger event, optional
```

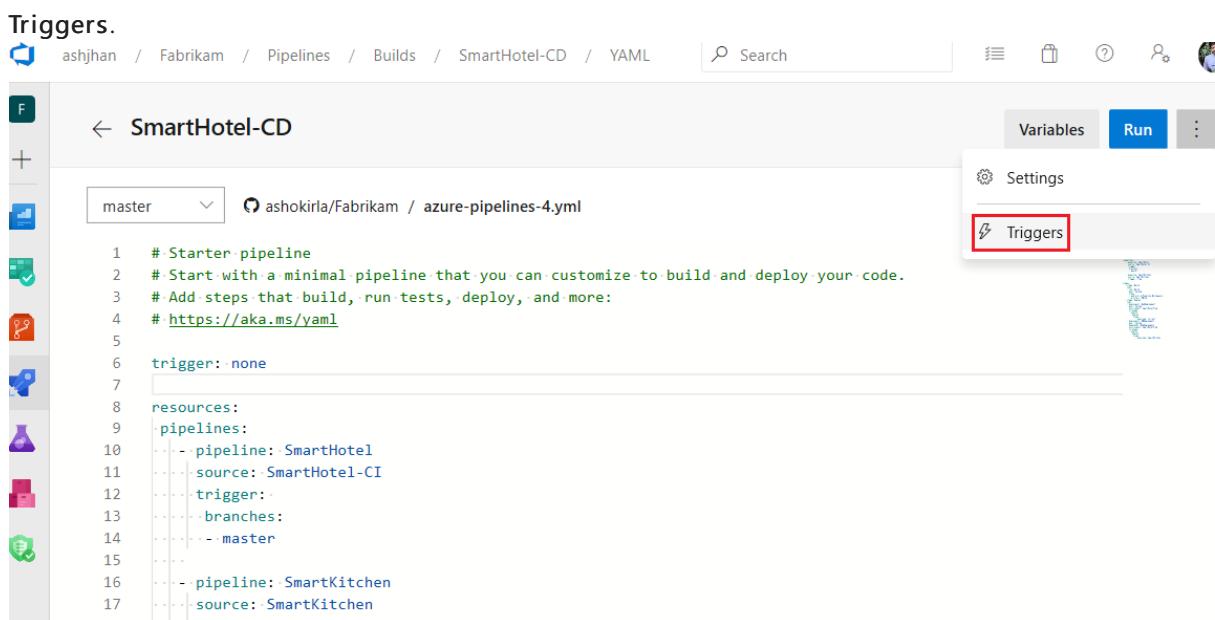
### IMPORTANT

When you define a resource trigger, if its pipeline resource is from the same repo as the current pipeline, triggering follows the same branch and commit on which the event is raised. But if the pipeline resource is from a different repo, the current pipeline is triggered on the default branch.

### Default branch for triggers

Triggers for resources are created based on the default branch configuration of your YAML, which is master. However, if you want to configure resource triggers from a different branch, you need to change the default branch for the pipeline.

1. Go to the edit view of the pipeline and click on the overflow menu on the top-right corner and choose Triggers.



2. Now select 'YAML' tab and go to 'Get sources'.
3. Now you can set the default branch for your pipeline.

The screenshot shows the Azure DevOps Pipelines interface for a project named 'SmartHotel-CD'. The pipeline is defined in YAML. The 'Get sources' step is selected, configured to pull from the 'ashokirla/Fabrikam' repository on GitHub, specifically the 'releases/M145' branch. The 'GitHub' source option is highlighted. The pipeline also includes steps for 'Build pipeline' and 'Deploy to App Service'.

## Evaluation of artifact version

The pipeline version (CI build run) that gets picked in your pipeline run is controlled by how your pipeline run is triggered.

In case your pipeline run is created by manual trigger or by scheduled trigger, the default version, branch, and tags are used to evaluate the version of CI pipeline version.

- If you provide a build version (number), that version runs.
- If you provide a branch, the latest version from the given branch runs.
- If you provide a list of tags, the latest run that has all the matching tags runs.
- If you provide a branch and list of tags, the latest run from the branch provided and that has the matching tags runs.
- If you don't provide anything, the latest version across all the branches runs.

```
resources:
 pipelines:
 - pipeline: MyCIALias
 project: Fabrikam
 source: Farbrikam-CI
 branch: master ### This branch input cannot have wild cards. It is used for evaluating default
 version when pipeline is triggered manually or scheduled.
 tags: ### These tags are used for resolving default version when the pipeline is triggered
 manually or scheduled
 - Production ### Tags are AND'ed
 - PreProduction
```

In case your pipeline is triggered automatically, the CI pipeline version will be picked based on the trigger event. The default version info provided irrelevant.

- If you provide branches, a new pipeline will be triggered whenever a CI run is successfully completed that matches to the branches that are included.
- If you provide tags, a new pipeline will be triggered whenever a CI run is successfully completed that matches all the tags mentioned.
- If you provide stages, new pipeline will be triggered whenever a CI run has all the stages mentioned are completed successfully.
- If you provide branches, tags and stages together, a new pipeline run is triggered whenever a CI run matches all the conditions.

- If you don't provide anything and just say `trigger: true`, a new pipeline run is triggered whenever a CI run is successfully completed.
- If you don't provide any trigger for the resource, no pipeline run will be triggered. Triggers are disabled by default unless you specifically enable them.

```
resources:
 pipelines:
 - pipeline: SmartHotel
 project: DevOpsProject
 source: SmartHotel-CI
 trigger:
 branches:
 include:
 - releases/*
 - master
 exclude:
 - topic/*
 tags:
 - Verified
 - Signed
 stages:
 - Production
 - PreProduction
```

#### `download` for pipelines

All artifacts from the current pipeline and from all `pipeline` resources are automatically downloaded and made available at the beginning of each `deployment` job. You can override this behavior. For more information, see [Pipeline Artifacts](#). Regular 'job' artifacts are not automatically downloaded. Use `download` explicitly when needed.

- [Schema](#)
- [Example](#)

```
steps:
 - download: [current | pipeline resource identifier | none] # disable automatic download if "none"
 artifact: string ## artifact name, optional; downloads all the available artifacts if not specified
 patterns: string # patterns representing files to include; optional
```

Artifacts from the `pipeline` resource are downloaded to

`$(PIPELINE.WORKSPACE)/<pipeline-identifier>/<artifact-identifier>` folder.

#### Pipeline resource variables

In each run, the metadata for a pipeline resource is available to all jobs in the form of below [predefined variables](#). The `<Alias>` is the identifier that you gave for your pipeline resource. Pipeline resources variables are only available at runtime.

- [Schema](#)
- [Example](#)

```
resources.pipeline.<Alias>.projectID
resources.pipeline.<Alias>.pipelineName
resources.pipeline.<Alias>.pipelineID
resources.pipeline.<Alias>.runName
resources.pipeline.<Alias>.runID
resources.pipeline.<Alias>.runURI
resources.pipeline.<Alias>.sourceBranch
resources.pipeline.<Alias>.sourceCommit
resources.pipeline.<Alias>.sourceProvider
resources.pipeline.<Alias>.requestedFor
resources.pipeline.<Alias>.requestedForID
```

## Resources: `builds`

If you have any external CI build system that produces artifacts, you can consume artifacts with a `builds` resource. A `builds` resource can be any external CI systems like Jenkins, TeamCity, CircleCI etc.

- [Schema](#)
- [Example](#)

```
resources: # types: pipelines | builds | repositories | containers | packages
builds:
- build: string # identifier for the build resource
 type: string # the type of your build service like Jenkins, circleCI etc.
 connection: string # service connection for your build service.
 source: string # source definition of the build
 version: string # the build number to pick the artifact, defaults to Latest successful build
 trigger: boolean # Triggers are not enabled by default and should be explicitly set
```

`builds` is an extensible category. You can write an extension to consume artifacts from your `builds` service (CircleCI, TeamCity etc.) and introduce a new type of service as part of `builds`. Jenkins is a type of resource in `builds`.

### IMPORTANT

Triggers are only supported for hosted Jenkins where Azure DevOps has line of sight with Jenkins server.

### `downloadBuild` for `builds`

You can consume artifacts from the `build` resource as part of your jobs using `downloadBuild` task. Based on the type of `build` resource defined (Jenkins, TeamCity etc.), this task automatically resolves to the corresponding download task for the service during the run time.

Artifacts from the `build` resource are downloaded to `$(PIPELINE.WORKSPACE)/<build-identifier>/` folder.

### IMPORTANT

`build` resource artifacts are not automatically downloaded in your jobs/deploy-jobs. You need to explicitly add `downloadBuild` task for consuming the artifacts.

- [Schema](#)
- [Example](#)

```

- downloadBuild: string # identifier for the resource from which to download artifacts
 artifact: string # artifact to download; if left blank, downloads all artifacts associated with the
resource provided
 patterns: string | [string] # a minimatch path or list of [minimatch paths](tasks/file-matching-
patterns.md) to download; if blank, the entire artifact is downloaded

```

## Resources: repositories

If your pipeline has [templates in another repository](#), or if you want to use [multi-repo checkout](#) with a repository that requires a service connection, you must let the system know about that repository. The `repository` keyword lets you specify an external repository.

- [Schema](#)
- [Example](#)

```

resources:
 repositories:
 - repository: string # identifier (A-Z, a-z, 0-9, and underscore)
 type: enum # see the following "Type" topic
 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 types that aren't Azure Repos)
 trigger: # CI trigger for this repository, no CI trigger if skipped (only works for Azure Repos)
 branches:
 include: [string] # branch names which will trigger a build
 exclude: [string] # branch names which will not
 tags:
 include: [string] # tag names which will trigger a build
 exclude: [string] # tag 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

```

### Type

Pipelines support the following values for the repository type: `git`, `github`, `githubenterprise`, and `bitbucket`. The `git` type refers to Azure Repos Git repos.

- If you specify `type: git`, the `name` value refers to another repository in the same project. An example is `name: otherRepo`. To refer to a repo in another project within the same organization, prefix the name with that project's name. An example is `name: OtherProject/otherRepo`.
- If you specify `type: github`, the `name` value is the full name of the GitHub repo and includes the user or organization. An example is `name: Microsoft/vscode`. GitHub repos require a [GitHub service connection](#) for authorization.
- If you specify `type: githubenterprise`, the `name` value is the full name of the GitHub Enterprise repo and includes the user or organization. An example is `name: Microsoft/vscode`. GitHub Enterprise repos require a [GitHub Enterprise service connection](#) for authorization.
- If you specify `type: bitbucket`, the `name` value is the full name of the Bitbucket Cloud repo and includes the user or organization. An example is `name: MyBitbucket/vscode`. Bitbucket Cloud repos require a [Bitbucket Cloud service connection](#) for authorization.

#### `checkout` your repository

Use `checkout` keyword to consume your repos defined as part of `repository` resource.

## Schema

```
steps:
- checkout: string # identifier for your repository resource
clean: boolean # if true, execute `execute git clean -ffdx && git reset --hard HEAD` before fetching
fetchDepth: number # the depth of commits to ask Git to fetch; defaults to no limit
lfs: boolean # whether to download Git-LFS files; defaults to false
submodules: true | recursive # set to 'true' for a single level of submodules or 'recursive' to get
submodules of submodules; defaults to not checking out submodules
path: string # path to check out source code, relative to the agent's build directory (e.g. _work\1);
defaults to a directory called `s`
persistCredentials: boolean # if 'true', leave the OAuth token in the Git config after the initial fetch;
defaults to false
```

Repos from the `repository` resource are not automatically synced in your jobs. Use `checkout` to fetch your repos as part of your jobs.

For more information, see [Check out multiple repositories in your pipeline](#).

## Resources: `containers`

If you need to consume a container image as part of your CI/CD pipeline, you can achieve it using `containers`. A container resource can be a public or private Docker Registry, or Azure Container Registry.

If you need to consume images from Docker registry as part of your pipeline, you can define a generic container resource (not `type` keyword required).

- [Schema](#)
- [Example](#)

```
resources:
containers:
- container: string # identifier (A-Z, a-z, 0-9, and underscore)
image: string # container image name
options: string # arguments to pass to container at startup
endpoint: string # reference to a service connection for the private registry
env: { string: string } # list of environment variables to add
ports: [string] # ports to expose on the container
volumes: [string] # volumes to mount on the container
mapDockerSocket: bool # whether to map in the Docker daemon socket; defaults to true
mountReadOnly: # volumes to mount read-only - all default to false
externals: boolean # components required to talk to the agent
tasks: boolean # tasks required by the job
tools: boolean # installable tools like Python and Ruby
work: boolean # the work directory
```

A generic container resource can be used as an image consumed as part of your job or it can also be used for [Container jobs](#).

You can use a first class container resource type for Azure Container Registry (ACR) to consume your ACR images. This resources type can be used as part of your jobs and also to enable automatic pipeline triggers.

- [Schema](#)
- [Example](#)

```

resources: # types: pipelines | repositories | containers | builds | packages
 containers:
 - container: string # identifier for the container resource
 type: string # type of the registry like ACR, GCR etc.
 azureSubscription: string # Azure subscription (ARM service connection) for container registry;
 resourceGroup: string # resource group for your ACR
 registry: string # registry for container images
 repository: string # name of the container image repository in ACR
 trigger: # Triggers are not enabled by default and need to be set explicitly
 tags:
 include: [string] # image tags to consider the trigger events, optional; defaults to any new tag
 exclude: [string] # image tags on discard the trigger events, optional; defaults to none

```

### Container resource variables

Once you define a container as resource, container image metadata is passed to the pipeline in the form of variables. Information like image, registry, and connection details are made accessible across all the jobs to be used in your container deploy tasks.

- [Schema](#)
- [Example](#)

```

resources.container.<Alias>.type
resources.container.<Alias>.registry
resources.container.<Alias>.repository
resources.container.<Alias>.tag
resources.container.<Alias>.digest
resources.container.<Alias>.URI
resources.container.<Alias>.location

```

Note: location variable is only applicable for `ACR` type of container resources.

## Resources: `packages`

You can consume NuGet and npm GitHub packages as a resource in YAML pipelines.

When specifying `package` resources, set the package as NuGet or npm. You can also enable automated pipeline triggers when a new package version gets released.

To use GitHub packages, you will need to use PAT-based authentication and create a GitHub service connection that uses PAT.

By default, packages will not be automatically downloaded into jobs. To download, use `getPackage`.

- [Schema](#)
- [Example](#)

```

resources:
 packages:
 - package: myPackageAlias # alias for the package resource
 type: Npm # type of the package NuGet/npm
 connection: GitHubConnectionName # Github service connection with the PAT type
 name: nugetTest/nodeapp # <Repository>/<Name of the package>
 version: 1.0.1 # Version of the package to consume; Optional; Defaults to latest
 trigger: true # To enable automated triggers (true/false); Optional; Defaults to no triggers

```

## Resources: `webhooks`

With other resources (such as pipelines, containers, build, and packages) you can consume artifacts and enable automated triggers. However, you could not automate your deployment process based on other external events or services. `webhooks` resource enables you to integrate your pipeline with any external service and automate the workflow. You can subscribe to any external events through its webhooks (GitHub, GitHub Enterprise, Nexus, Artifactory, etc.) and trigger your pipelines.

Here are the steps to configure the webhook triggers:

1. Set up a webhook on your external service. When creating your webhook, you need to provide the following info:
  - Request Url -  
`https://dev.azure.com/<ADO Organization>/_apis/public/distributedtask/webhooks/<WebHook Name>?api-version=6.0-preview`
  - Secret - This is optional. If you need to secure your JSON payload, provide the **Secret** value
2. Create a new "Incoming Webhook" service connection. This is a newly introduced Service Connection Type that will allow you to define three important pieces of information:
  - **Webhook Name:** The name of the webhook should match webhook created in your external service.
  - **HTTP Header** - The name of the HTTP header in the request that contains the payload's HMAC-SHA1 hash value for request verification. For example, in the case of the GitHub, the request header will be "**X-Hub-Signature**"
  - **Secret** - The secret is used to verify the payload's HMAC-SHA1 hash used for verification of the incoming request (this is optional). If you have used a secret in creating your webhook, you will need to provide the same secret key

# Edit service connection



## Authentication

WebHook Name

Name of the WebHook

Secret (optional)

Optional secret for the webhook. WebHook service will use this secret to calculate the payload checksum

Http Header (optional)

Http header name on which checksum will be sent

## Details

Service connection name

Description (optional)

[Learn more](#)[Troubleshoot](#)[Cancel](#)[Save](#)

3. A new resource type called `webhooks` is introduced in YAML pipelines. For subscribing to a webhook event, you need to define a webhook resource in your pipeline and point it to the Incoming webhook service connection. You can also define additional filters on the webhook resource based on the JSON payload data to further customize the triggers for each pipeline, and you can consume the payload data in the form of variables in your jobs.

4. Whenever a webhook event is received by the Incoming Webhook service connection, a new run will be triggered for all the pipelines subscribed to the webhook event. You can consume the JSON payload data in your jobs using the format \${{ parameters.}}}

- [Schema](#)
- [Example](#)

```
resources:
 webhooks:
 - webhook: MyWebhookTriggerAlias ### Webhook alias
 connection: IncomingWebhookConnection ### Incoming webhook service connection
 filters:
 - path: JSONParameterPath ### List of JSON parameters to filter; Parameters are AND'ed
 value: JSONParameterExpectedValue ### JSON path in the payload
 ### Expected value in the path provided
```

Webhooks are a great way to automate your workflow based on any external webhook event that is not supported by first class resources like pipelines, builds, containers, and packages. Also, for on-premise services where Azure DevOps doesn't have visibility into the process, you can configure webhooks on the service and to trigger your pipelines automatically.

## Manual version picker for resources in the create run dialogue

When you manually trigger a CD YAML pipeline, we automatically evaluate the default version for the resources defined in the pipeline based on the inputs provided. However, you can choose to pick a different version from resource version picker in create run dialogue.

1. In the create run pane, you can see resources section.
2. Clicking on it shows the list of resources consumed in this pipeline.
3. You can select each of the resource and pick a specific version from the list of versions available. Resource version picker is supported for pipeline, build, repository, container, and package resources.

The screenshot shows the Azure DevOps interface for the 'myAppCD' pipeline. On the left, there's a list of pipeline runs with descriptions like '#20200430.20 Update azure-pipelines-8.yml for Azure Pipelines'. On the right, a modal dialog titled 'myApp CI' is open, listing pipeline runs with their status (Completed) and the date they were run. One run is selected: '#20200430.12'.

Pipeline run	Completed
#20200430.13	Yesterday
<b>#20200430.12</b>	Yesterday
#20200430.11	Yesterday
#20200430.10	Yesterday
#20200430.9	Yesterday
#20200430.8	Yesterday
#20200430.7	Yesterday
#20200430.6	Yesterday

For pipeline resources, you can see all the available runs across all branches. You can search them based on the pipeline number or branch. And you can pick a run that is successful, failed or in-progress run. This flexibility is given to ensure you can run your CD pipeline if you are sure your CI pipeline produced all the artifacts you need and you don't need to wait for the CI run is complete or rerun due to some unrelated stage in the CI run failed. However, when we evaluate default version for scheduled triggers or if you don't use manual version picker, we only consider successfully completed CI runs.

For resources where you can't fetch available versions (like GitHub packages), we will show a text box as part of

version picker so that user can provide the version to be picked in the run.

## Troubleshooting authorization for a YAML pipeline

Resources must be authorized before they can be used. A resource owner controls the users and pipelines that can access that resource. The pipeline must be authorized to use the resource. There are multiple ways to accomplish this.

- Navigate to the administration experience of the resource. For example, variable groups and secure files are managed in the **Library** page under **Pipelines**. Agent pools and service connections are managed in **Project settings**. Here you can authorize **all pipelines** to be able to access that resource. This is convenient if you do not have a need to restrict access to a resource - for example, test resources.
- When you create a pipeline for the first time, all the resources that are referenced in the YAML file are automatically authorized for use by the pipeline, provided that you are a member of the **User** role for that resource. So, resources that are referenced in the YAML file at pipeline creation time are automatically authorized.
- When you make changes to the YAML file and add additional resources (assuming that these not authorized for use in all pipelines as explained above), then the build fails with a resource authorization error that is similar to the following:

```
Could not find a <resource> with name <resource-name>. The <resource> does not exist or has not been authorized for use.
```

In this case, you will see an option to authorize the resources on the failed build. If you are a member of the **User** role for the resource, you can select this option. Once the resources are authorized, you can start a new build.

- If you continue to have problems authorizing resources, verify that the [agent pool security roles](#) for your project are correct.

## Set approval checks for resources

You can manually control when a resource runs with approval checks and templates. With the [required template approval check](#), you can require that any pipeline using a resource or environment also extends from a specific YAML template. Setting a required template approval enhances security. You can make sure that your resource only gets used under specific conditions with a template. Learn more about how to [enhance pipeline security](#) with templates and resources.

## Traceability

We provide full traceability for any resource consumed at a pipeline or deployment-job level.

### Pipeline traceability

For every pipeline run, we show the info about the

1. The resource that has triggered the pipeline (if it is triggered by a resource).

The screenshot shows the Azure Pipelines pipeline run summary for a run titled "#20200102.3 Update azure-pipelines-4.yml for Azure Pipelines" on the SmartHotel-CD project. The run was triggered by "SmartHotel/20200102.3". The "Summary" tab is selected, showing details such as the trigger, duration (32s), tests (Get started), changes (1 commit), work items (-), and artifacts (2 consumed). The "Environments" tab is also visible.

2. Version of the resource and the artifacts consumed.

The screenshot shows the 'Artifacts' page with the 'Consumed' tab selected. It displays a single artifact entry for 'SmartKitchen / # 20200102.3'. The entry includes the name 'SmartKitchen / # 20200102.3' and a 'View details' button. Below the main entry, there is a section titled 'Accessories' with a checkbox icon.

3. Commits associated with each resource.

The screenshot shows the 'Changes' view for the 'SmartKitchen / #20200102.3' pipeline. It lists two commits:

Commit	SHA
Update azure-pipelines-4.yml for Azure Pipelines ashokirla committed on 46m ago	2d98a3315
Update azure-pipelines-4.yml for Azure Pipelines ashokirla committed on 52m ago	b1dde2df3

Below this, another section for 'SmartKitchen / #20200102.3' shows one commit:

Commit	SHA
Update azure-pipelines.yml for Azure Pipelines ashokirla committed on 50m ago	0466067c8

4. Work-items for each resource.

### Environment traceability

Whenever a pipeline deploys to an environment, you can see a list of resources that are consumed in the environments view. This view includes resources consumed as part of the deployment-jobs and their associated commits and work-items.

The screenshot shows the 'Deployment by 20200102.10' view. It lists two work-items:

Work-item	Commit	SHA
SmartKitchen / #20200102.5	Update azure-pipelines.yml for Azure Pipelines ashokirla committed on 5m ago	50f8fc08a
current / #20200102.10	Update azure-pipelines-4.yml for Azure Pipelines ashokirla committed on 5m ago	37f337e5b

## Showing associated CD pipelines info in CI pipelines

To provide end to end traceability, user should be able to track which CD pipelines are consuming a giving CI pipeline. You can see the list of CD YAML pipelines runs where a CI pipeline run is consumed through `pipeline` resource. In your CI pipeline run view, if it is consumed by other pipeline(s), you will see a 'Associated pipelines' tab where you can find all the pipeline runs that consume your pipeline and artifacts from it.

The screenshot shows a pipeline run details page. At the top, there's a summary bar with a green checkmark icon, the run ID '#20200428.4 Set up CI with Azure Pipelines', the project name 'Proj1', and status 'Retained'. Below this are two tags: 'tag5' and 'tag6'. The main content area has tabs: 'Summary', 'Code Coverage', and 'Associated pipelines', with 'Associated pipelines' being the active tab. It displays a table with two rows of data:

Run	Start time
Update azure-pipelines-4.yml for Azure Pipelines #20200429.1 on Proj1CD	Wed at 9:34 PM
Update azure-pipelines.yml for Azure Pipelines #20200428.7 on CD Pipeline	Tue at 5:32 PM

## YAML resource trigger issues support and traceability

It can be confusing when pipeline triggers fail to execute. To help better understand this, we've added a new menu item in the pipeline definition page called **Trigger Issues** where you can learn why triggers are not executing. To access this page, open your pipeline history. Select **Trigger Issues** in the menu. The **Trigger Issues** option will only appear for non-repository resources.

The screenshot shows a dropdown menu from a pipeline history card. The menu items are:

- View
- Run pipeline** (highlighted in blue)
- : (three dots)
- Manage security
- Status badge
- Trigger issues** (highlighted with a red box)
- Clone
- Export to JSON
- Export to YAML
- Scheduled runs
- Request signing approval

At the bottom of the menu, there are two small status indicators: 'yesterday' and '<1s'.

Resource triggers can fail to execute for two reasons.

- If the source of the service connection provided is invalid, or if there are any syntax errors in the trigger, the trigger will not be configured at all. These are surfaced as errors.
- If trigger conditions are not matched, the trigger will not execute. Whenever this occurs, a warning will be surfaced so you can understand why the conditions were not matched.

A

Trigger issues

Issues	YAML Commit
<span style="color: red;">X</span> Configuring the trigger failed, edit and save the pipeline again MyPipeline1	217bf782
<span style="color: orange;">!</span> Tags set for the trigger didn't match the pipeline MyPipeline2/#20200529.3	c31ed8fd
<span style="color: orange;">!</span> Tags set for the trigger didn't match the pipeline MyPipeline2/#20200529.4	c31ed8fd
<span style="color: orange;">!</span> Tags set for the trigger didn't match the pipeline MyPipeline2/#20200529.5	c31ed8fd
<span style="color: orange;">!</span> Tags set for the trigger didn't match the pipeline MyPipeline2/#20200529.1	4fb226f

## FAQ

### Why should I use pipelines resources instead of the download shortcut?

Using a `pipelines` resource is a first class way to consume artifacts from a CI pipeline and also configure automated triggers. It gives you full visibility into the process by displaying the version consumed, artifacts, commits, and work-items. When you define a pipeline resource, the associated artifacts are automatically downloaded in deployment jobs.

You can choose to download the artifacts in build jobs or to override the download behavior in deployment jobs with `download`. The `download` task internally uses the [Download Pipeline Artifacts task](#).

### Why should I use resources instead of the Download Pipeline Artifacts task?

When you use the [Download Pipeline Artifacts task](#) directly, you miss traceability and triggers. At the same time, there are times when it makes sense to use the Download Pipeline Artifacts task directly. For example, you might have a script task stored in a different template and the script task requires artifacts from a build to be downloaded. Or, you may not know if someone using a template will add a pipeline resource. To avoid dependencies, you can use the Download Pipeline Artifacts task to pass all the build info to a task.

# Protect a repository resource

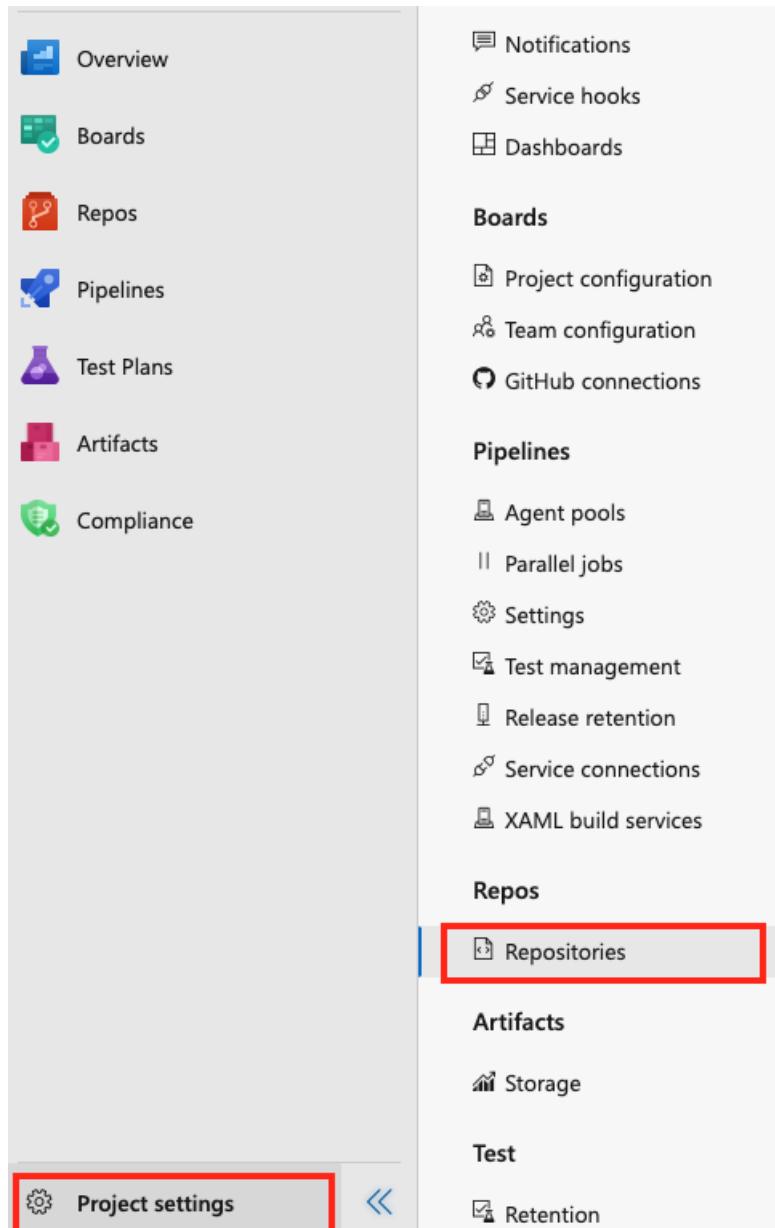
5/19/2021 • 2 minutes to read • [Edit Online](#)

You can add protection to your [repository resource](#) with checks and pipeline permissions. When you add protection, you'll be better able to restrict repository ownership and editing privileges.

## Add a repository resource check

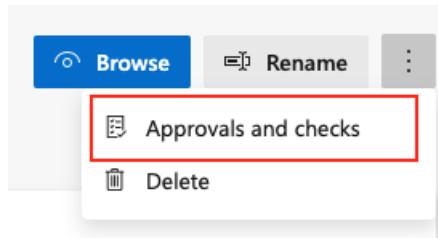
Repository owners can control when a stage in a pipeline gets to use their repository with [checks and approvals](#). You can manage your checks and approvals for repositories from the [Project settings](#). To set up a check:

1. Go to [Project settings > Repositories](#).



2. Select the repository you want to modify.

3. From , select [Approvals and checks](#).



4. Add a check to set how your repository resource can be used. For example, you can add a manual approver each time a pipeline requests the repository. Learn more about [approvals and checks](#).

## Add pipeline permissions to a repository resource

You can also set a repository to only be used on specific pipelines. Restricting a repository to specific pipelines will prevent an unauthorized pipeline in your project from using your repository. To set up repository pipeline permissions:

1. Go to **Project settings > Repositories**.

The screenshot shows the 'Project settings' interface on the left, with the 'Repositories' option highlighted by a red box. The main pane displays a hierarchical menu:

- Notifications**
- Service hooks**
- Dashboards**
- Boards**
  - Project configuration**
  - Team configuration**
  - GitHub connections**
- Pipelines**
  - Agent pools**
  - Parallel jobs**
  - Settings**
  - Test management**
  - Release retention**
  - Service connections**
  - XAML build services**
- Repos**
  - Repositories** (highlighted with a red box)
- Artifacts**
- Storage**
- Test**
- Retention**

2. Select the repository you want to modify.
3. Open the **Security** tab.

4. Go to Pipeline permissions.

The screenshot shows a user interface for managing pipeline permissions. At the top, there are tabs for 'Settings', 'Policies', and 'Security'. The 'Security' tab is selected, indicated by a blue underline. Below the tabs, the title 'Pipeline permissions' is displayed. To the right of the title are two small buttons: a '+' sign and a three-dot menu icon. In the center of the page is a large circular icon containing a lock symbol. Below this icon, the text 'No permitted pipelines' is shown in bold. Underneath this, a smaller message reads: 'Only YAML pipelines defined in this Repository will be permitted to use it.' and 'Designer pipelines can use this resource.' A 'Learn more' link is also present. The overall background is light gray.

5. Select + .

6. Select your repository to add. You'll see the repository listed now.

# Add & use variable groups

5/21/2021 • 19 minutes to read • [Edit Online](#)

Use a variable group to store values that you want to control and make available across multiple pipelines. You can also use variable groups to store secrets and other values that might need to be [passed into a YAML pipeline](#). Variable groups are defined and managed in the [Library](#) page under [Pipelines](#).

Variables groups are a [protected resource](#). You can add approvals and checks to them and set pipeline permissions.

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## 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

- [YAML](#)
- [Classic](#)
- [Azure DevOps CLI](#)

Variable groups can't be created in YAML, but they can be used as described in [Use a variable group](#).

## Use a variable group

- [YAML](#)
- [Classic](#)
- [Azure DevOps CLI](#)

To use a variable from a variable group, you need to add a reference to the group in your YAML file:

```
variables:
- group: my-variable-group
```

Thereafter variables from the variable group can be used in your YAML file.

If you use both variables and variable groups, you'll have to use `name / value` syntax for the individual (non-grouped) variables:

```
variables:
- group: my-variable-group
- name: my-bare-variable
 value: 'value of my-bare-variable'
```

To reference a variable group, you can use macro syntax or a runtime expression. In this example, the group `my-variable-group` has a variable named `myhello`.

```
variables:
- group: my-variable-group
- name: my-passed-variable
 value: ${variables.myhello} # uses runtime expression

steps:
- script: echo $(myhello) # uses macro syntax
- script: echo $(my-passed-variable)
```

You can reference multiple variable groups in the same pipeline. If multiple variable groups include the same variable, the variable group included last in your YAML file will set the variable's value.

```
variables:
- group: my-first-variable-group
- group: my-second-variable-group
```

You can also reference a variable group in a template. In the template `variables.yml`, the group `my-variable-group` is referenced. The variable group includes a variable named `myhello`.

```
variables.yml
variables:
- group: my-variable-group
```

In this pipeline, the variable `$(myhello)` from the variable group `my-variable-group` included in `variables.yml` is referenced.

```
azure-pipeline.yml
stages:
- stage: MyStage
 variables:
 - template: variables.yml
 jobs:
 - job: Test
 steps:
 - script: echo $(myhello)
```

To work with variable groups, you must authorize the group. This is a security feature: if you only had to name the variable group in YAML, then anyone who can push code to your repository could extract the contents of secrets in the variable group. To do this, or if you encounter a resource authorization error in your build, use one of the following techniques:

- If you want to authorize any pipeline to use the variable group, which may be a suitable option if you do not have any secrets in the group, go to Azure Pipelines, open the **Library** page, choose **Variable groups**, select the variable group in question, and enable the setting **Allow access to all pipelines**.
- If you want to authorize a variable group for a specific pipeline, open the pipeline by selecting **Edit** and queue a build manually. You will see a resource authorization error and a "Authorize resources" action on

the error. Choose this action to explicitly add the pipeline as an authorized user of the variable group.

#### NOTE

If you added a variable group to a pipeline and did not get a resource authorization error in your build when you expected one, turn off the **Allow access to all pipelines** setting described above.

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. For more information, see [secrets](#)

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.

## Manage a variable group

Using the Azure DevOps CLI, you can list the variable groups for the pipeline runs in your project and show details for each one. You can also delete variable groups if you no longer need them.

[List variable groups](#) | [Show details for a variable group](#) | [Delete a variable group](#)

### List variable groups

You can list the variable groups in your project with the [az pipelines variable-group list](#) command. To get started, see [Get started with Azure DevOps CLI](#).

```
az pipelines variable-group list [--action {manage, none, use}]
 [--continuation-token]
 [--group-name]
 [--org]
 [--project]
 [--query-order {Asc, Desc}]
 [--top]
```

#### Optional parameters

- **action**: Specifies the action that can be performed on the variable groups. Accepted values are *manage*, *none* and *use*.
- **continuation-token**: Lists the variable groups after a continuation token is provided.
- **group-name**: Name of the variable group. Wildcards are accepted, such as `new-var*`.
- **org**: Azure DevOps organization URL. You can configure the default organization using  
`az devops configure -d organization=ORG_URL`. Required if not configured as default or picked up using  
`git config`. Example: `--org https://dev.azure.com/MyOrganizationName/`.
- **project**: Name or ID of the project. You can configure the default project using  
`az devops configure -d project=NAME_OR_ID`. Required if not configured as default or picked up using  
`git config`.
- **query-order**: Lists the results in either ascending or descending (the default) order. Accepted values are *Asc* and *Desc*.
- **top**: Number of variable groups to list.

#### Example

The following command lists the top 3 variable groups in ascending order and returns the results in table format.

```
az pipelines variable-group list --top 3 --query-order Asc --output table

ID Name Type Number of Variables
--- -----
1 myvariables Vsts 2
2 newvariables Vsts 4
3 new-app-variables Vsts 3
```

## Show details for a variable group

You can display the details of a variable group in your project with the [az pipelines variable-group show](#) command. To get started, see [Get started with Azure DevOps CLI](#).

```
az pipelines variable-group show --group-id
 [--org]
 [--project]
```

### Parameters

- **group-id**: Required. ID of the variable group. To find the variable group ID, see [List variable groups](#).
- **org**: Azure DevOps organization URL. You can configure the default organization using  
`az devops configure -d organization=ORG_URL`. Required if not configured as default or picked up using  
`git config`. Example: `--org https://dev.azure.com/MyOrganizationName/`.
- **project**: Name or ID of the project. You can configure the default project using  
`az devops configure -d project=NAME_OR_ID`. Required if not configured as default or picked up using  
`git config`.

### Example

The following command shows details for the variable group with the ID 4 and returns the results in YAML format.

```
az pipelines variable-group show --group-id 4 --output yaml

authorized: false
description: Variables for my new app
id: 4
name: MyNewAppVariables
providerData: null
type: Vsts
variables:
 app-location:
 isSecret: null
 value: Head_Office
 app-name:
 isSecret: null
 value: Fabrikam
```

## Delete a variable group

You can delete a variable group in your project with the [az pipelines variable-group delete](#) command. To get started, see [Get started with Azure DevOps CLI](#).

```
az pipelines variable-group delete --group-id
 [--org]
 [--project]
 [--yes]
```

#### Parameters

- **group-id**: Required. ID of the variable group. To find the variable group ID, see [List variable groups](#).
- **org**: Azure DevOps organization URL. You can configure the default organization using  
`az devops configure -d organization=ORG_URL`. Required if not configured as default or picked up using  
`git config`. Example: `--org https://dev.azure.com/MyOrganizationName/`.
- **project**: Name or ID of the project. You can configure the default project using  
`az devops configure -d project=NAME_OR_ID`. Required if not configured as default or picked up using  
`git config`.
- **yes**: Optional. Doesn't prompt for confirmation.

#### Example

The following command deletes the variable group with the ID 1 and doesn't prompt for confirmation.

```
az pipelines variable-group delete --group-id 1 --yes

Deleted variable group successfully.
```

## Manage variables in a variable group

Using the Azure DevOps CLI, you can add and delete variables from a variable group in a pipeline run. You can also list the variables in the variable group and make updates to them as needed.

[Add variables to a variable group](#) | [List variables in a variable group](#) | [Update variables in a variable group](#) |  
[Delete variables from a variable group](#)

### Add variables to a variable group

You can add a variable to a variable group with the `az pipelines variable-group variable create` command. To get started, see [Get started with Azure DevOps CLI](#).

```
az pipelines variable-group variable create --group-id
 --name
 [--org]
 [--project]
 [--secret {false, true}]
 [--value]
```

#### Parameters

- **group-id**: Required. ID of the variable group. To find the variable group ID, see [List variable groups](#).
- **name**: Required. Name of the variable you are adding.
- **org**: Azure DevOps organization URL. You can configure the default organization using  
`az devops configure -d organization=ORG_URL`. Required if not configured as default or picked up using  
`git config`. Example: `--org https://dev.azure.com/MyOrganizationName/`.
- **project**: Name or ID of the project. You can configure the default project using  
`az devops configure -d project=NAME_OR_ID`. Required if not configured as default or picked up using  
`git config`.
- **secret**: Optional. Indicates whether the variable's value is a secret. Accepted values are *false* and *true*.
- **value**: Required for non secret variable. Value of the variable. For secret variables, if **value** parameter is not

provided, it is picked from environment variable prefixed with `AZURE_DEVOPS_EXT_PIPELINE_VAR_` or user is prompted to enter it via standard input. For example, a variable named **MySecret** can be input using the environment variable `AZURE_DEVOPS_EXT_PIPELINE_VAR_MySecret`.

#### Example

The following command creates a variable in the variable group with ID of 4. The new variable is named **requires-login** and has a value of **True**, and the result is shown in table format.

```
az pipelines variable-group variable create --group-id 4 --name requires-login --value True --output table

Name Is Secret Value

requires-login False True
```

#### List variables in a variable group

You can list the variables in a variable group with the `az pipelines variable-group variable list` command. To get started, see [Get started with Azure DevOps CLI](#).

```
az pipelines variable-group variable list --group-id
[--org]
[--project]
```

#### Parameters

- **group-id**: Required. ID of the variable group. To find the variable group ID, see [List variable groups](#).
- **org**: Azure DevOps organization URL. You can configure the default organization using  
`az devops configure -d organization=ORG_URL`. Required if not configured as default or picked up using  
`git config`. Example: `--org https://dev.azure.com/MyOrganizationName/`.
- **project**: Name or ID of the project. You can configure the default project using  
`az devops configure -d project=NAME_OR_ID`. Required if not configured as default or picked up using  
`git config`.

#### Example

The following command lists all of the variables in the variable group with ID of 4 and shows the result in table format.

```
az pipelines variable-group variable list --group-id 4 --output table

Name Is Secret Value

app-location False Head_Office
app-name False Fabrikam
requires-login False True
```

#### Update variables in a variable group

You can update a variable in a variable group with the `az pipelines variable-group variable update` command. To get started, see [Get started with Azure DevOps CLI](#).

```
az pipelines variable-group variable update --group-id
 --name
 [--new-name]
 [--org]
 [--project]
 [--prompt-value {false, true}]
 [--secret {false, true}]
 [--value]
```

#### Parameters

- **group-id**: Required. ID of the variable group. To find the variable group ID, see [List variable groups](#).
- **name**: Required. Name of the variable you are adding.
- **new-name**: Optional. Specify to change the name of the variable.
- **org**: Azure DevOps organization URL. You can configure the default organization using  
`az devops configure -d organization=ORG_URL`. Required if not configured as default or picked up using  
`git config`. Example: `--org https://dev.azure.com/MyOrganizationName/`.
- **project**: Name or ID of the project. You can configure the default project using  
`az devops configure -d project=NAME_OR_ID`. Required if not configured as default or picked up using  
`git config`.
- **prompt-value**: Set to **true** to update the value of a secret variable using environment variable or prompt via standard input. Accepted values are *false* and *true*.
- **secret**: Optional. Indicates whether the variable's value is kept secret. Accepted values are *false* and *true*.
- **value**: Updates the value of the variable. For secret variables, use the **prompt-value** parameter to be prompted to enter it via standard input. For non-interactive consoles, it can be picked from environment variable prefixed with `AZURE_DEVOPS_EXT_PIPELINE_VAR_`. For example, a variable named **MySecret** can be input using the environment variable `AZURE_DEVOPS_EXT_PIPELINE_VAR_MySecret`.

#### Example

The following command updates the **requires-login** variable with the new value **False** in the variable group with ID of 4. It specifies that the variable is a **secret** and shows the result in YAML format. Notice that the output shows the value as **null** instead of **False** since it is a secret value (hidden).

```
az pipelines variable-group variable update --group-id 4 --name requires-login --value False --secret true -
--output yaml

requires-login:
 isSecret: true
 value: null
```

## Delete variables from a variable group

You can delete a variable from a variable group with the `az pipelines variable-group variable delete` command. To get started, see [Get started with Azure DevOps CLI](#).

```
az pipelines variable-group variable delete --group-id
 --name
 [--org]
 [--project]
 [--yes]
```

#### Parameters

- **group-id**: Required. ID of the variable group. To find the variable group ID, see [List variable groups](#).
- **name**: Required. Name of the variable you are deleting.
- **org**: Azure DevOps organization URL. You can configure the default organization using

```
az devops configure -d organization=ORG_URL . Required if not configured as default or picked up using
git config . Example: --org https://dev.azure.com/MyOrganizationName/ .
```

- **project:** Name or ID of the project. You can configure the default project using  
`az devops configure -d project=NAME_OR_ID` . Required if not configured as default or picked up using  
`git config` .
- **yes:** Optional. Doesn't prompt for confirmation.

#### Example

The following command deletes the **requires-login** variable from the variable group with ID of 4 and prompts for confirmation.

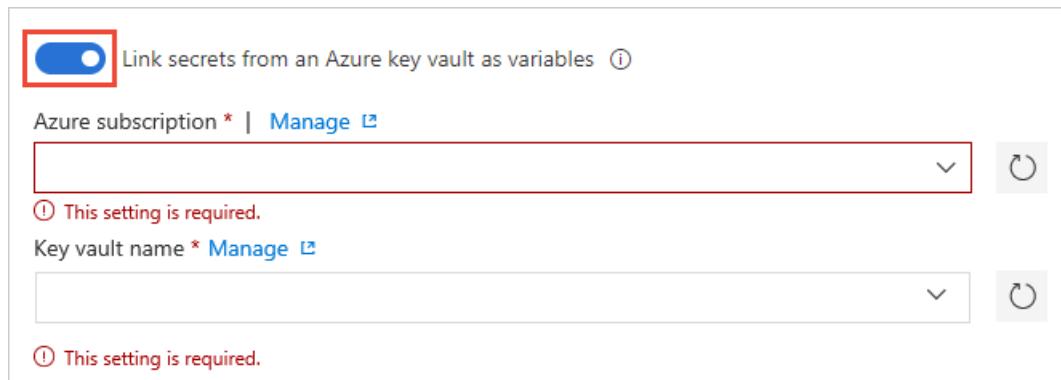
```
az pipelines variable-group variable delete --group-id 4 --name requires-login

Are you sure you want to delete this variable? (y/n): y
Deleted variable 'requires-login' successfully.
```

## Link secrets from an Azure key vault

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](#).



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 run.

- 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 pipelines 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 pipelines 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 supported.

## Expansion of variables in a group

- [YAML](#)
- [Classic](#)
- [Azure DevOps CLI](#)

When you set a variable in a group and use it in a YAML file, it has the same precedence as any other variable defined within the YAML file. For more information about precedence of variables, see the topic on [variables](#).

YAML is not supported in TFS.

# Use secure files

6/2/2021 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Secure Files are a way to store files that you can share across pipelines. You can 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 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 when you consume them from a task. Secure files are a [protected resource](#). You can add approvals and checks to them and set pipeline permissions. Secure files also can use the [Library security model](#).

There's a size limit of 10 MB for each secure file.

## Add a secure file

You can add Secure Files from **Pipelines > Library**.

1. Go to **Library** and select the **Secure files** tab.

The screenshot shows the Azure Pipelines Library interface. At the top, there are tabs for "Variable groups" and "Secure files". The "Secure files" tab is highlighted with a blue underline. Below the tabs, there is a button labeled "+ Secure file" with a blue plus sign icon. The main area displays a list of secure files, with the first item being "+ Secure file". At the bottom of the screen, there is a navigation bar with several items: "Secure file" (selected), "Save", "Security" (highlighted with a red box), "Pipeline permissions", "Approvals and checks" (highlighted with a red box), and "Help".

2. Select **+** to upload a new secure file. You can browse to upload or drag and drop your file. Once your file is uploaded, you'll be able to delete it but you will not be able to replace it.
3. Add permissions to your file.
  - a. You can apply security role restrictions for all files from the **Security** tab at **Pipelines > Library**.
  - b. To add permissions for an individual file, in the file's edit view select **Pipeline permissions** to set per-pipeline permissions. Or, select **Security** to set security roles. You can also **Approvals and Checks** for the file. Learn more about [approvals and checks](#).

## Consume a secure file in a pipeline

Use the [Download Secure File](#) Utility task to consume secure files in a pipeline.

This YAML pipeline example downloads a secure certificate file and installs it in a Linux environment.

```
- task: DownloadSecureFile@1
 name: caCertificate
 displayName: 'Download CA certificate'
 inputs:
 secureFile: 'myCACertificate.pem'

- script: |
 echo Installing $(caCertificate.secureFilePath) to the trusted CA directory...
 sudo chown root:root $(caCertificate.secureFilePath)
 sudo chmod a+r $(caCertificate.secureFilePath)
 sudo ln -s -t /etc/ssl/certs/ $(caCertificate.secureFilePath)
```

## FAQ

### 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](#).

### How do I authorize a secure file for use in a specific pipeline?

1. In Azure Pipelines, select the **Library** tab.
2. Select the **Secure files** tab at the top.
3. Select the secure file you want to authorize.
4. Select the **Pipeline permissions** button.
5. In the open dialog, you are able to review and modify the access for each available pipeline.

### Why do I see an `Invalid Resource` error when downloading a secure file with Azure DevOps Server/TFS on-premises?

Make sure [IIS Basic Authentication](#) is disabled on the TFS or Azure DevOps Server.

### How are secure files secured?

Secure files, variable groups, and service connections are all secured in the same way in Azure DevOps. They are also all [protected resources](#).

Secrets are encrypted and stored in the database. The keys to decrypt secrets are stored in Azure Key Vault. The keys are specific to each scale unit. So, two regions do not share the same keys. The keys are also rotated with every deployment of Azure DevOps.

The rights to retrieve secure keys are only given to the Azure DevOps service principals and (on special occasions) on-demand to diagnose problems. The secure storage does not have any certifications.

[Azure Key Vault](#) is another, more secure option for securing sensitive information. If you decide to use Azure Key Vault, you can [use it with variable groups](#).

# Service connections

5/26/2021 • 27 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Service connections enable you to connect to external and remote services to execute tasks in a job. 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.

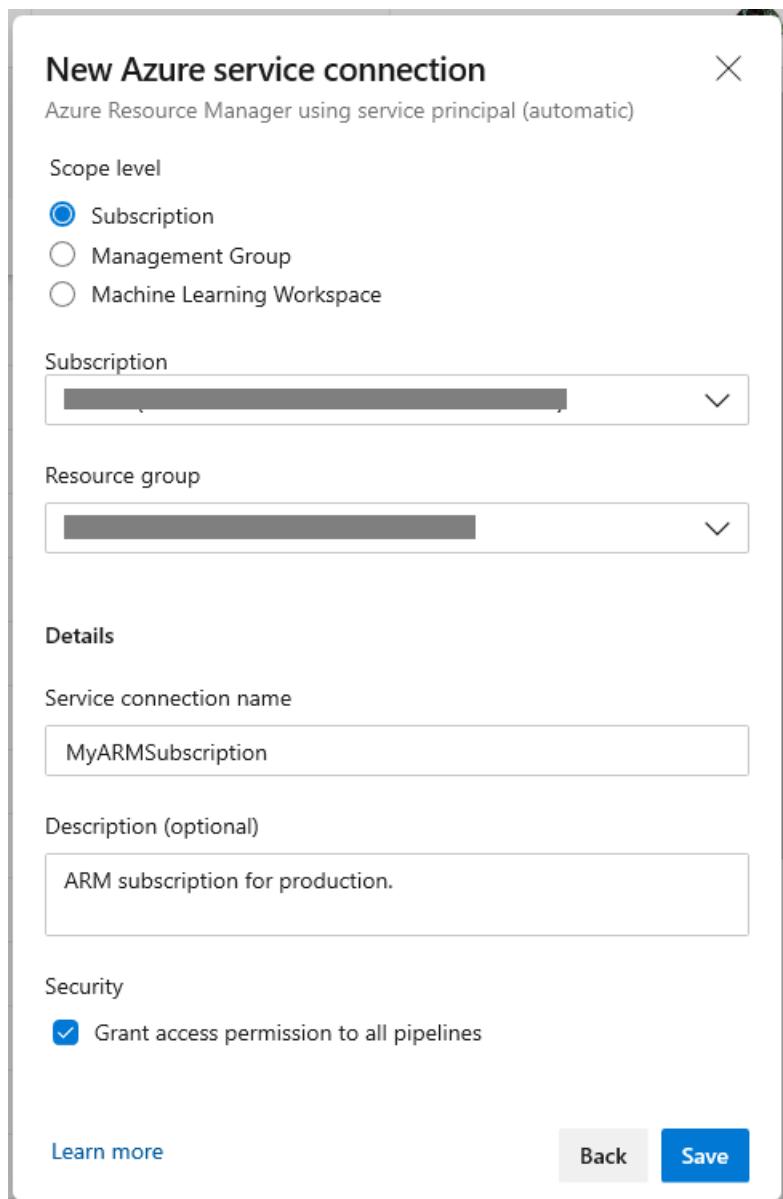
It's possible to define service connections in Azure Pipelines that are available for use in all your tasks. For example, you can create a service connection for your Azure subscription and use this service connection name in an Azure Web Site Deployment task in a release pipeline.

You define and manage service connections from the Admin settings of your project:

- Azure DevOps: [https://dev.azure.com/{organization}/{project}/\\_settings/adminservices](https://dev.azure.com/{organization}/{project}/_settings/adminservices)
- TFS: [https://{tfsserver}/{collection}/{project}/\\_admin/\\_services](https://{tfsserver}/{collection}/{project}/_admin/_services)

## 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](#).
4. Decide if you want the service connection to be accessible for any pipeline by setting the **Allow all pipelines to use this connection** option. This option allows pipelines defined in YAML, which are not automatically authorized for service connections, to use this service connection. See [Use a service connection](#).
5. Choose **OK** to create the connection. For example, this is the default **Azure Resource Manager** connection dialog:



#### NOTE

The connection dialog may appear different for the different types of service connections, and have different parameters. See the list of parameters in [Common service connection types](#) for each service connection type.

## Manage a service connection

1. In Azure DevOps, open the **Service connections** page from the [project settings page](#). Or, in TFS, open the **Services** page from the "settings" icon in the top menu bar.
2. Select the service connection you want to manage.
3. You will land in the **Overview** tab of the service connection where you can see the details of the service connection i.e. type, creator, authentication type (like Token, Username/Password or OAuth etc.).

The screenshot shows the Azure DevOps interface for managing service connections. On the left, a sidebar lists various project settings like Boards, Pipelines, and Service connections\*. The main area is titled 'MyARMSsubscription' and shows the 'Overview' tab selected. It displays details about the service connection, including its type (Azure Resource Manager using service principal authentication), creator (a user icon), and description ('ARM subscription for Production').

4. Next to the overview tab, you can see **Usage history** that shows the list of pipelines using the service connection.

The screenshot shows the same service connection page, but the 'Usage history' tab is now selected. This tab lists all the pipeline runs that have used this service connection, including their names (e.g., Release-94, Release-93) and dates (e.g., Oct 7, 2019, Oct 4, 2019).

Run	Date
Release-94	Oct 7, 2019
SCTestRelease	
Release-94	Oct 7, 2019
SCTestRelease	
Release-93	Oct 4, 2019
SCTestRelease	
Release-93	Oct 4, 2019
SCTestRelease	
Release-92	Oct 3, 2019
SCTestRelease	
Release-92	Oct 3, 2019
SCTestRelease	
Release-91	Oct 2, 2019
SCTestRelease	
Release-91	Oct 2, 2019
SCTestRelease	
Release-91	Oct 2, 2019
SCTestRelease	

5. To update the service connection, click on **Edit** at the top right corner of the page.

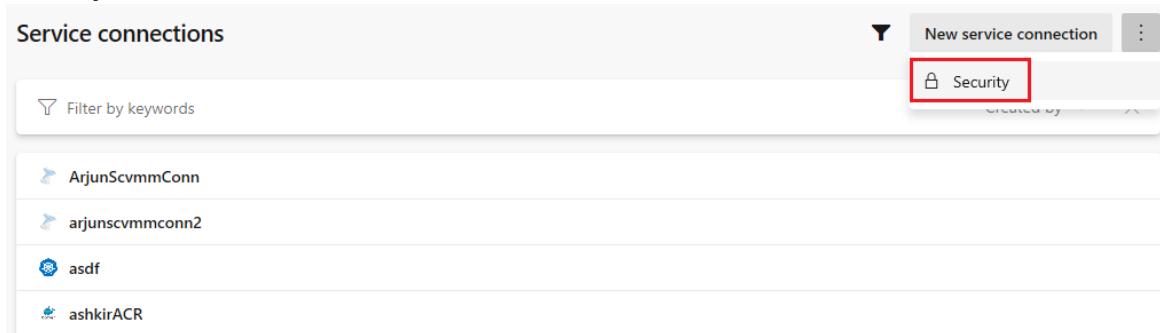
6. **Approvals and checks**, **Security** and **Delete** are part of the more options at the top right corner.

The screenshot shows the 'Edit' button from the previous screen, which has triggered a context menu. The menu includes three items: 'Approvals and checks', 'Security', and 'Delete'. The 'Delete' option is highlighted with a red box.

# Secure a service connection

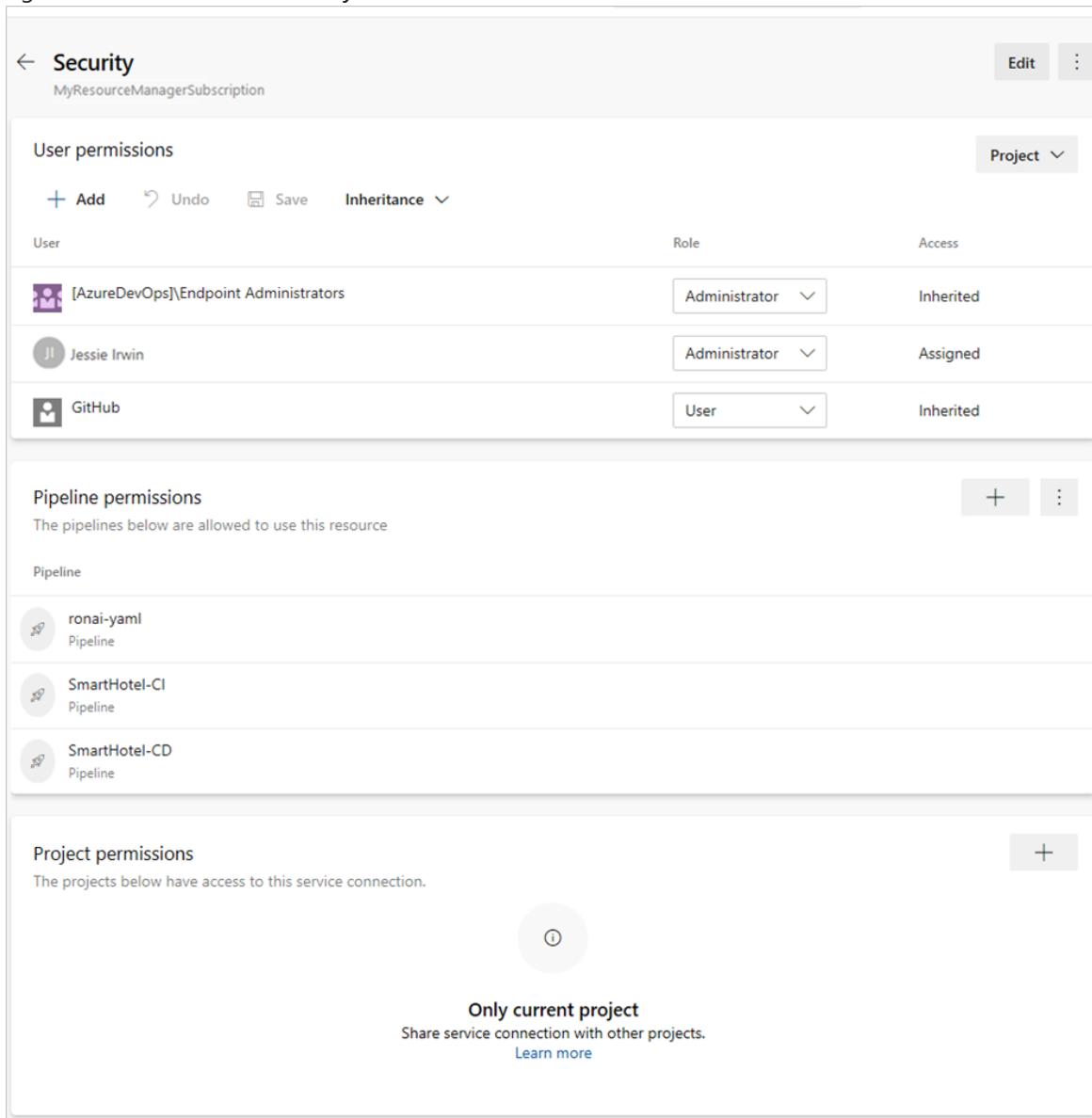
To manage the security for a connection:

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.
2. To manage user permissions at hub level, go to the more options at the top-right corner and choose **Security**.



The screenshot shows the 'Service connections' page. At the top right, there is a 'More options' icon (three dots) and a 'Security' button, which is highlighted with a red box. Below the header is a search bar labeled 'Filter by keywords'. The main area lists four service connections: 'ArjunScvmmConn', 'arjunsrvmmconn2', 'asdf', and 'ashkirACR'. Each entry has a small icon and a delete button.

3. To manage security for a service connection, open the service connection and go to more options at top-right corner and choose **Security**.



The screenshot shows the 'Security' settings for a service connection named 'MyResourceManagerSubscription'.  
**User permissions:** This section shows three entries: '[AzureDevOps]\Endpoint Administrators' (Role: Administrator, Access: Inherited), 'Jessie Irwin' (Role: Administrator, Access: Assigned), and 'GitHub' (Role: User, Access: Inherited).  
**Pipeline permissions:** This section lists three pipelines: 'ronai-yaml', 'SmartHotel-CI', and 'SmartHotel-CD'.  
**Project permissions:** This section states 'Only current project' and 'Share service connection with other projects.' with a 'Learn more' link.

Service connection is a critical resource for various workflows in Azure DevOps like Classic Build and Release pipelines, YAML pipelines, KeyVault Variable groups etc. Based on the usage patterns, service connection security

is divided into three categories in the service connections new UI.

- User permissions
- Pipeline permissions
- Project permissions

### User permissions

You can control who can create, view, use, and manage the service connection with user permissions. There are four roles to manage each of these actions: Creator, Reader, User, and Administrator. In the service connections tab, you can set the hub level permissions. Hub level permissions are inherited. You can also override the roles for each service connection in the service connections tab.

ROLE ON A SERVICE CONNECTION	PURPOSE
Creator	Members of this role can create the service connection in the project. Contributors are added as members by default
Reader	Members of this role can view the service connection.
User	Members of this role can use the service connection when authoring build or release pipelines or authorize yaml pipelines.
Administrator	In addition to using the service connection, members of this role can manage membership of all other roles for the service connection in the project. Project administrators are added as members by default

Previously, two special groups, Endpoint Creators and Endpoint Administrator groups were used to control who can create and manage service connections. Now, as part of service connection new UI, we are moving to pure RBAC model i.e. using roles. For backward compatibility, in the existing projects, Endpoint Administrators group is added as Administrator role and Endpoint creators group is assigned with creator role which ensures there is no change in the behavior for existing service connections.

#### NOTE

This change is applicable only in Azure DevOps Services where new UI is available. Azure DevOps Server 2019 and older versions still follow the previous security model.

Along with the new service connections UI, we are introducing **Sharing of service connections across projects**. With this feature, service connections now become an organization level object however scoped to current project by default. In User permissions section, you can see **Project** and **Organization** level permissions. And the functionalities of administrator role is split between the two levels.

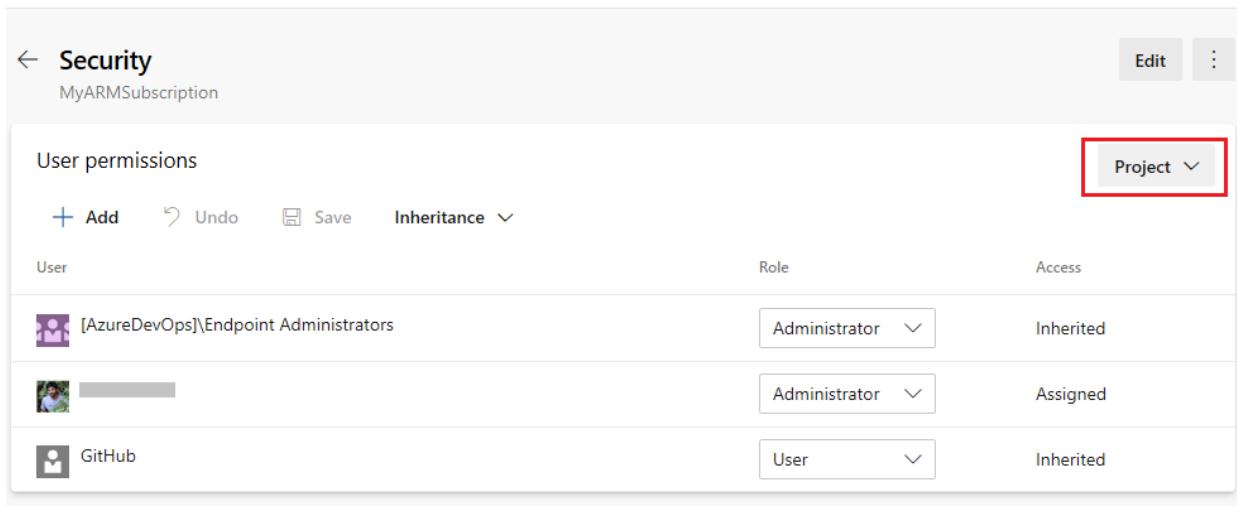
#### Project level permissions

The project level permissions are the user permissions with reader, user, creator and administrator roles, as explained above, within the project scope. You have inheritance and you can set the roles at the hub level as well as for each service connection.

The project-level administrator have limited administrative capabilities as below:

- A project-level administrator can manage other users and roles at project scope.
- A project-level administrator can rename a service connection, update description and enable/disable "Allow pipeline access" flag.
- A project-level administrator can delete a service connection which removes the existence of service

connection from the project.



The screenshot shows the 'User permissions' section of the security settings for a service connection named 'MyARMSubscription'. The 'Project' dropdown is highlighted with a red box. The table lists three users: '[AzureDevOps]\Endpoint Administrators' (Role: Administrator, Access: Inherited), a user with a profile picture (Role: Administrator, Access: Assigned), and 'GitHub' (Role: User, Access: Inherited).

User	Role	Access
[AzureDevOps]\Endpoint Administrators	Administrator	Inherited
[Redacted]	Administrator	Assigned
GitHub	User	Inherited

The user that created the service connection is automatically added to the project level Administrator role for that service connection. And users/groups assigned administrator role at hub level are inherited if the inheritance is turned on.

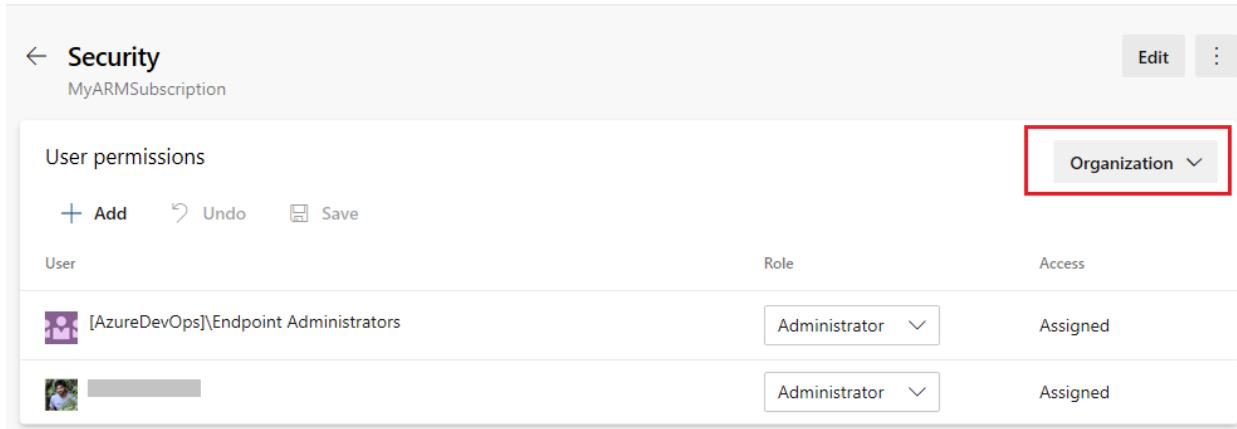
#### Organization level permissions

Organization level permissions are introduced along with cross project sharing feature. Any permissions set at this level are reflected across all the projects where the service connection is shared.

There is no inheritance for organization level permissions. Today we only have administrator role at organization level.

The organization-level administrator has all the administrative capabilities that include:

- A organization-level administrator can manage organization level users.
- A organization-level administrator can edit all the fields of a service connection.
- A organization-level administrator can share/un-share a service connection with other projects.



The screenshot shows the 'User permissions' section of the security settings for a service connection named 'MyARMSubscription'. The 'Organization' dropdown is highlighted with a red box. The table lists two users: '[AzureDevOps]\Endpoint Administrators' (Role: Administrator, Access: Assigned) and a user with a profile picture (Role: Administrator, Access: Assigned).

User	Role	Access
[AzureDevOps]\Endpoint Administrators	Administrator	Assigned
[Redacted]	Administrator	Assigned

The user that created the service connection is automatically added as a organization level Administrator role for that service connection. In all the existing service connections, for backward compatibility, all the connection administrators are made organization-level administrators to ensure there is no change in the behavior.

#### Pipeline permissions

Pipeline permissions control which YAML pipelines are authorized to use this service connection. This is interlinked with 'Allow pipeline access' checkbox you find in service connection creation dialogue.

You can either choose to open access for all pipelines to consume this service connection from the more options at top right corner of the **Pipeline permissions** section in security tab of a service connection.

Or you can choose to lock down the service connection and only allow selected YAML pipelines to consume this service connection. If any other YAML pipeline refers to this service connection, an authorization request is raised which has to be approved by the connection administrators.

The screenshot shows the 'Pipeline permissions' section. It lists three pipelines that are allowed to use the service connection: 'ronai-yaml Pipeline', 'SmartHotel-CI Pipeline', and 'SmartHotel-CD Pipeline'. There are buttons for '+', '...', and 'Open access'.

### Project permissions - Cross project sharing of service connections

Project permissions control which projects can use this service connection. By default, service connections are not shared with any other projects.

- Only the organization-level administrators from **User permissions** can share the service connection with other projects.
- The user who is sharing the service connection with a project should have atleast create service connection permission in the target project.
- The user who shares the service connection with a project becomes the project-level administrator for that service connection and the project-level inheritance is turned on in the target project.
- The service connection name is appended with the project name and it can be renamed in the target project scope.
- Organization level administrator can un-share a service connection from any shared project.

The screenshot shows the 'Project permissions' section. It lists two projects that have access to the service connection: 'APexplorer' and 'zzz\_Azure DevOps'. There is a '+' button.

#### NOTE

Project permissions feature is dependent on the new service connections UI and once we enable this feature, the old service connections UI is no longer usable.

## Use a service connection

After the new service connection is created:

- [YAML](#)
- [Classic](#)

Copy the connection name into your code as the `azureSubscription` (or the equivalent connection name)

value.

The screenshot shows a file browser interface with a left sidebar containing project files and a right panel showing the contents of the selected file, .vsts-ci.yml. The file contains YAML configuration for a build pipeline. Two sections of the code are highlighted with red boxes:

```
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'
```

Next you must authorize the service connection. To do this, or if you encounter a resource authorization error in your build, use one of the following techniques:

- If you want to authorize any pipeline to use the service connection, go to Azure Pipelines, open the Settings page, select Service connections, and enable the setting **Allow all pipelines to use this connection** option for the connection.
- If you want to authorize a service connection for a specific pipeline, open the pipeline by selecting **Edit** and queue a build manually. You will see a resource authorization error and an "Authorize resources" action on the error. Choose this action to explicitly add the pipeline as an authorized user of the service connection.

You can also create your own [custom service connections](#).

#### NOTE

Service connection cannot be specified by variable

## Common service connection types

Azure Pipelines and TFS support various service connection types by default, including:

- [Azure Classic](#)
- [Azure Repos/Team Foundation Server](#)
- [Azure Resource Manager](#)
- [Azure Service Bus](#)
- [Bitbucket Cloud](#)
- [Chef](#)
- [Docker Host](#)
- [Docker Registry](#)
- [External Git](#)
- [Generic](#)
- [GitHub](#)

- [GitHub Enterprise Server](#)
- [Jenkins](#)
- [Kubernetes](#)
- [Maven](#)
- [npm](#)
- [NuGet](#)
- [Python package download](#)
- [Python package upload](#)
- [Service Fabric](#)
- [SSH](#)
- [Subversion](#)
- [Visual Studio App Center](#)

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 <b>Credentials</b> or <b>Certificate based</b> .
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 <b>azureSubscription</b> or the equivalent subscription name value in the script.
Environment	Required. Select <b>Azure Cloud</b> , <b>Azure Stack</b> , or one of the pre-defined <b>Azure Government Clouds</b> 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 <a href="#">publish settings XML file</a> 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 Repos/Team Foundation Server

Defines and secures a connection to another Azure DevOps organization.

PARAMETER	DESCRIPTION
(authentication)	Select <b>Basic</b> or <b>Token Based</b> 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 <b>azureSubscription</b> or the equivalent subscription name value in the script.
Connection URL	Required. The URL of the TFS or the other Azure DevOps organization.
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. <a href="#">Learn more</a> .

Use the **Verify connection** link to validate your connection information.

See also [Authenticate access with personal access tokens for Azure DevOps](#).

## Azure Resource Manager service connection

Defines and secures a connection to a Microsoft Azure subscription using Service Principal Authentication (SPA) or an Azure Managed Service Identity. The dialog offers two main 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. Service principals are valid for two years.

For more information, see [Connect to Microsoft Azure](#)

### 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 <b>azureSubscription</b> or the equivalent subscription name value in the script.
Service Bus ConnectionString	The URL of your Azure Service Bus instance. <a href="#">More information</a> .
Service Bus Queue Name	The name of an existing Azure Service Bus queue.

## Bitbucket Cloud service connection

Defines a connection to Bitbucket Cloud. You can use OAuth with **Grant authorization** or a username and password with **Basic Authentication**. For pipelines to keep working, your repository access must remain active.

### Grand authorization

PARAMETER	DESCRIPTION
OAuth Configuration	Required. OAuth connection to Bitbucket.

### Basic authentication

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 <b>azureSubscription</b> 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.

## 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 <b>azureSubscription</b> or the equivalent subscription name value in the script.
Server URL	Required. The URL of the Chef automation server.

PARAMETER	DESCRIPTION
Node Name (Username)	Required. The name of the node to connect to. Typically this is your username.
Client Key	Required. The key specified in the Chef .pem file.

## 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 <b>azureSubscription</b> 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](#).

## Docker Registry service connection

Defines a connection to a container registry.

### Azure Container Registry

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task inputs.
Azure subscription	Required. The Azure subscription containing the container registry to be used for service connection creation.
Azure Container Registry	Required. The Azure Container Registry to be used for creation of service connection.

### Docker Hub or Others

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task inputs.
Docker Registry	Required. The URL of the Docker registry.

PARAMETER	DESCRIPTION
Docker ID	Required. The identifier of the Docker account user.
Password	Required. The password for the account user identified above. (Docker Hub requires a PAT instead of a password.)
Email	Optional. An email address to receive notifications.

### External Git service connection

Defines and secures a connection to a Git repository server. There is a specific service connection for [GitHub](#) and [GitHub Enterprise Server](#) 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 <b>azureSubscription</b> 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](#).

### 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 <b>azureSubscription</b> or the equivalent subscription name value in the script.
Server URL	Required. The URL of the service.
User name	Optional. The username to connect to the service.
Password/Token Key	Optional. The password or access token for the specified username.

### 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 Server](#) connections.

PARAMETER	DESCRIPTION
Choose authorization	Required. Either <b>Grant authorization</b> or <b>Personal access token</b> . See notes below.
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 <b>azureSubscription</b> or the equivalent subscription name value in the script.

#### 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:

1. Open your profile from your organization name at the right of the Azure Pipelines page heading.
2. At the top of the left column, under **DETAILS**, choose **Security**.
3. In the **Security** tab, in the right column, choose **Personal access tokens**.
4. Choose the **Add** link and enter the information required to create the token.

Also see [Artifact sources](#).

#### GitHub Enterprise Server 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 <b>Personal access token</b> , <b>Username and Password</b> , or <b>OAuth2</b> . 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 <b>azureSubscription</b> 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 <b>agent</b> .
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 configuration	Required for OAuth2 authorization. The OAuth configuration specified in your account.
GitHub Enterprise Server configuration URL	The URL is fetched from OAuth configuration.

#### 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:

1. Open your profile from your account name at the right of the Azure Pipelines page heading.
2. At the top of the left column, under **DETAILS**, choose **Security**.
3. In the **Security** tab, in the right column, choose **Personal access tokens**.
4. 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 <b>azureSubscription</b> 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 <a href="#">agent</a> .
User name	Required. The username to connect to the service.
Password	Required. The password for the specified username.

Also see [Azure Pipelines Integration with Jenkins](#) and [Artifact sources](#).

### Kubernetes service connection

Defines a connection to a Kubernetes cluster.

#### Azure subscription option

PARAMETER	DESCRIPTION

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task inputs.
Azure subscription	Required. The Azure subscription containing the cluster to be used for service connection creation.
Cluster	Name of the Azure Kubernetes Service cluster.
Namespace	Namespace within the cluster.

For an RBAC enabled cluster, a ServiceAccount is created in the chosen namespace along with RoleBinding object so that the created ServiceAccount is able to perform actions only on the chosen namespace.

For an RBAC disabled cluster, a ServiceAccount is created in the chosen namespace. But the created ServiceAccount has cluster-wide privileges (across namespaces).

#### NOTE

This option lists all the subscriptions the service connection creator has access to *across different Azure tenants*. If you are unable to see subscriptions from other Azure tenants, please check your AAD permissions in those tenants.

### Service account option

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task inputs.
Server URL	Required. Cluster's API server URL.
Secret	Secret associated with the service account to be used for deployment

The following command can be used to fetch Server URL -

```
kubectl config view --minify -o 'jsonpath={.clusters[0].cluster.server}'
```

For fetching Secret object required to connect and authenticate with the cluster, the following sequence of commands needs to be run -

```
kubectl get serviceAccounts <service-account-name> -n <namespace> -o 'jsonpath={.secrets[*].name}'
```

The above command fetches the name of the secret associated with a ServiceAccount. The output of the above command is to be substituted in the following command for fetching Secret object -

```
kubectl get secret <service-account-secret-name> -n <namespace> -o json
```

Copy and paste the Secret object fetched in YAML form into the Secret text-field.

#### NOTE

When using the service account option, [ensure that a RoleBinding exists](#), which grants permissions in the [edit ClusterRole](#) to the desired service account. This is needed so that the service account can be used by Azure Pipelines for creating objects in the chosen namespace.

## Kubeconfig option

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task inputs.
Kubeconfig	Required. Contents of the kubeconfig file
Context	Context within the kubeconfig file that is to be used for identifying the cluster

## Maven service connection

Defines and secures a connection to a Maven 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 <code>azureSubscription</code> or the equivalent subscription name value in the script.
Registry URL	Required. The URL of the Maven repository.
Registry ID	Required. This is the ID of the server that matches the <code>id</code> element of the repository/mirror that Maven tries to connect to.
Username	Required when connection type is <b>Username and Password</b> . The username for authentication.
Password	Required when connection type is <b>Username and Password</b> . The password for the username.
Personal Access Token	Required when connection type is <b>Authentication Token</b> . The token to use to authenticate with the service. <a href="#">Learn more</a> .

## npm service connection

Defines and secures a connection to an npm server.

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 <b>azureSubscription</b> or the equivalent subscription name value in the script.
Registry URL	Required. The URL of the npm server.
Username	Required when connection type is <b>Username and Password</b> . The username for authentication.
Password	Required when connection type is <b>Username and Password</b> . The password for the username.
Personal Access Token	Required when connection type is <b>External Azure Pipelines</b> . The token to use to authenticate with the service. <a href="#">Learn more</a> .

## 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 <b>azureSubscription</b> or the equivalent subscription name value in the script.
Feed URL	Required. The URL of the NuGet server.
ApiKey	Required when connection type is <b>ApiKey</b> . The authentication key.
Personal Access Token	Required when connection type is <b>External Azure Pipelines</b> . The token to use to authenticate with the service. <a href="#">Learn more</a> .
Username	Required when connection type is <b>Basic authentication</b> . The username for authentication.
Password	Required when connection type is <b>Basic authentication</b> . The password for the username.

To configure NuGet to authenticate with Azure Artifacts and other NuGet repositories, See [NuGet Authenticate](#).

## Python package download service connection

Defines and secures a connection to a Python repository for downloading Python packages.

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 <b>azureSubscription</b> 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 <b>Authentication Token</b> . The token to use to authenticate with the service. <a href="#">Learn more</a> .
Username	Required when connection type is <b>Username and Password</b> . The username for authentication.
Password	Required when connection type is <b>Username and Password</b> . The password for the username.

## 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 <b>azureSubscription</b> 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.
Personal Access Token	Required when connection type is <b>Authentication Token</b> . The token to use to authenticate with the service. <a href="#">Learn more</a> .
Username	Required when connection type is <b>Username and Password</b> . The username for authentication.
Password	Required when connection type is <b>Username and Password</b> . The password for the username.

## Service Fabric service connection

Defines and secures a connection to a Service Fabric cluster.

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 <b>azureSubscription</b> 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 <b>Certificate based</b> or <b>Azure Active Directory</b> .
Client Certificate	Required when connection type is <b>Certificate based</b> .
Password	Required when connection type is <b>Certificate based</b> . The certificate password.
Username	Required when connection type is <b>Azure Active Directory</b> . The username for authentication.
Password	Required when connection type is <b>Azure Active Directory</b> . The password for the username.
Use Windows security	Required when connection type is <b>Others</b> .
Cluster SPN	Required when connection type is <b>Others</b> and using Windows security.

## SSH service connection

Defines and secures a connection to a remote host using Secure Shell (SSH).

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 <b>azureSubscription</b> 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.

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 <b>azureSubscription</b> 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.

## 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 <b>azureSubscription</b> or the equivalent subscription name value in the script.
API Token	Required. The token to use to authenticate with the service. <a href="#">Learn more</a> .

## 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 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
- Get advice on [Stack Overflow](#), and feel free to post your questions, search for answers, or suggest a feature on our [Azure DevOps Developer Community](#). [Support](#) page.

# Azure Pipelines agents

6/11/2021 • 25 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

To build your code or deploy your software using Azure Pipelines, you need at least one agent. As you add more code and people, you'll eventually need more.

When your pipeline runs, the system begins one or more jobs. An agent is computing infrastructure with installed agent software that runs one job at a time.

Jobs can be run [directly on the host machine of the agent](#) or [in a container](#).

## Microsoft-hosted agents

If your pipelines are in Azure Pipelines, then you've got a convenient option to run your jobs 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 for each job in the pipeline. The virtual machine is discarded after one job. Microsoft-hosted agents can run jobs [directly on the VM](#) or [in a container](#).

Azure Pipelines provides a pre-defined agent pool named **Azure Pipelines** with Microsoft-hosted agents.

For many teams this is the simplest way to run your jobs. 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.

[Learn more about Microsoft-hosted agents.](#)

## Self-hosted agents

An agent that you set up and manage on your own to run 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. Also, machine-level caches and configuration persist from run to run, which can boost speed.

## NOTE

Although multiple agents can be installed per machine, we strongly suggest to only install one agent per machine. Installing two or more agents may adversely affect performance and the result of your pipelines.

**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 Docker container. For more information about installing a self-hosted agent, see:

- [macOS agent](#)
- [Linux agent](#) (x64, ARM, RHEL6)
- [Windows agent](#) (x64, x86)
- [Docker agent](#)

You can install the agent on Linux, macOS, or Windows machines. For more information about installing a self-hosted agent, see:

- [macOS agent](#)
- [Red Hat agent](#)
- [Ubuntu 14.04 agent](#)
- [Ubuntu 16.04 agent](#)
- [Windows agent v1](#)

**NOTE**

On macOS, you need to clear the special attribute on the download archive to prevent Gatekeeper protection from displaying for each assembly in the tar file when `./config.sh` is run. The following command clears the extended attribute on the file:

```
xattr -c vsts-agent-osx-x64-V.v.v.tar.gz ## replace V.v.v with the version in the filename downloaded.
then unpack the gzip tar file normally:
tar xvfz vsts-agent-osx-x64-V.v.v.tar.gz
```

After you've installed the agent on a machine, you can install any other software on that machine as required by your jobs.

**NOTE**

Agents are widely backward compatible. Any version of the agent should be compatible with any Azure DevOps version as long as Azure DevOps isn't demanding a higher version of the agent.

We only support the most recent version of the agent since that is the only version guaranteed to have all up-to-date patches and bug fixes.

## Azure virtual machine scale set agents

Azure virtual machine scale set agents are a form of self-hosted agents that can be auto-scaled to meet your demands. This elasticity reduces your need to run dedicated agents all the time. Unlike Microsoft-hosted agents, you have flexibility over the size and the image of machines on which agents run.

You specify a virtual machine scale set, a number of agents to keep on standby, a maximum number of virtual machines in the scale set, and Azure Pipelines manages the scaling of your agents for you.

For more information, see [Azure virtual machine scale set agents](#).

## Parallel jobs

**Parallel jobs** represents the number of jobs you can run at the same time in your organization. If your organization has a single parallel job, you can run a single job at a time in your organization, with any additional concurrent jobs being queued until the first job completes. To run two jobs at the same time, you need two parallel jobs. In Azure Pipelines, you can run parallel jobs on Microsoft-hosted infrastructure or on your own (self-hosted) infrastructure.

Microsoft provides a free tier of service by default in every organization that includes at least one parallel job. Depending on the number of concurrent pipelines you need to run, you might need more parallel jobs to use multiple Microsoft-hosted or self-hosted agents at the same time. For more information on parallel jobs and different free tiers of service, see [Parallel jobs in Azure Pipelines](#).

You might need more parallel jobs to use multiple agents at the same time:

- [Parallel jobs in TFS](#)

### IMPORTANT

Starting with Azure DevOps Server 2019, you do not have to pay for self-hosted concurrent jobs in releases. You are only limited by the number of agents that you have.

## Capabilities

Every self-hosted 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.

### NOTE

Storing environment variables as capabilities means that when an agent runs, the stored capability values are used to set the environment variables. Also, any changes to environment variables that are made while the agent is running won't be picked up and used by any task. If you have sensitive environment variables that change and you don't want them to be stored as capabilities, you can have them ignored by setting the `VSO_AGENT_IGNORE` environment variable, with a comma-delimited list of variables to ignore. For example, `PATH` is a critical variable that you might want to ignore if you're installing software.

When you author a pipeline, 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 jobs to specific agents.

### NOTE

Demands and capabilities are designed for use with self-hosted agents so that jobs can be matched with an agent that meets the requirements of the job. When using Microsoft-hosted agents, you select an image for the agent that matches the requirements of the job, so although it is possible to add capabilities to a Microsoft-hosted agent, you don't need to use capabilities with Microsoft-hosted agents.

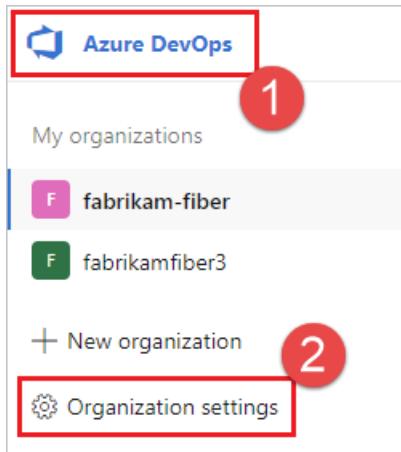
## Configure agent capabilities

- Browser
- Azure DevOps CLI

You can view the details of an agent, including its version and system capabilities, and manage its user capabilities, by navigating to **Agent pools** and selecting the **Capabilities** tab for the desired agent.

1. In your web browser, navigate to Agent pools:

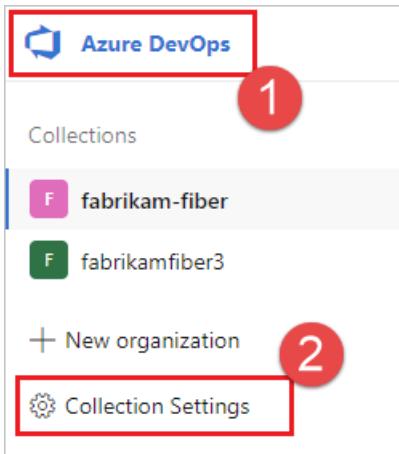
1. Choose **Azure DevOps, Organization settings**.



2. Choose **Agent pools**.

The screenshot shows the 'Agent pools' page in Azure DevOps. The left sidebar has links for Security, Policies, Permissions, Boards, Process, Pipelines, Agent pools (which is highlighted with a red box and a red circle containing the number '1'), Settings, Deployment pools, and Parallel jobs. The main area is titled 'Agent pools' and lists four items: 'Azure Pipelines' (with a cloud icon), 'Default' (with a monitor icon), 'Test' (with a monitor icon), and 'Test2' (with a monitor icon). A red circle containing the number '2' is placed over the 'Agent pools' title.

1. Choose **Azure DevOps, Collection settings**.



2. Choose Agent pools.

Azure DevOps

FabrikamFiber / Organization Settings / Agent pools

Security

- Policies
- Permissions

Boards

- Process

Pipelines

Agent pools

Settings

Deployment pools

Parallel jobs

## Agent pools

Name	Type
Azure Pipelines	Azure Pipelines
Default	Mobile
Test	Mobile
Test2	Mobile

1. Choose Azure DevOps, Collection settings.

Azure DevOps

Collections

- fabrikam-fiber
- fabrikamfiber3

+ New organization

Collection Settings

2. Choose Agent pools.

Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards  
Process

Pipelines

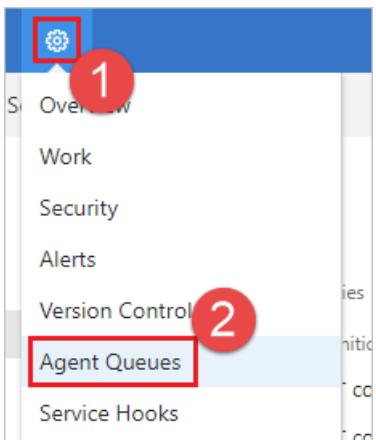
Agent pools **1**

Settings  
Deployment pools  
Parallel jobs

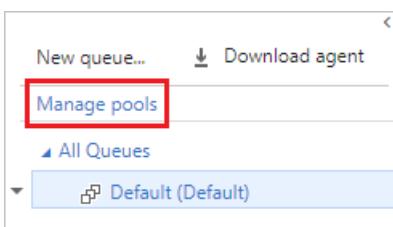
## Agent pools

Name
Azure Pipelines
Default
Test
Test2

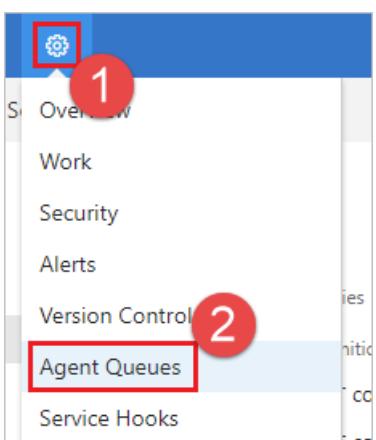
1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



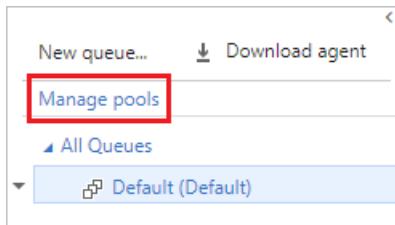
2. Choose **Manage pools**.



1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



2. Choose **Manage pools**.



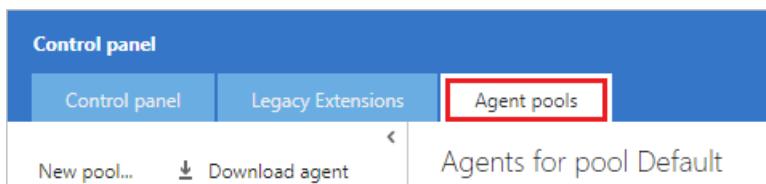
1. Navigate to your project and choose **Manage project** (gear icon).



2. Choose **Control panel**.



3. Select **Agent pools**.



2. Navigate to the capabilities tab:

1. From the **Agent pools** tab, select the desired agent pool.

A screenshot of the Azure DevOps interface. On the left, a sidebar shows 'Agent pools' highlighted with a red box and a red number '1'. The main area is titled 'Agent pools' and lists three items: 'Azure Pipelines' (highlighted with a red box and a red number '2'), 'Default' (highlighted with a red box), and 'Test'. The 'Default' pool has a red circle with the number '2' above it.

2. Select **Agents** and choose the desired agent.

The screenshot shows the 'Default' pool settings. The 'Agents' tab is selected, indicated by a red circle labeled '1'. Below it, the first agent entry, '160724-AT14-SP3', is highlighted with a red box labeled '2'. The status of this agent is 'Offline'.

3. Choose the **Capabilities** tab.

The screenshot shows the '160724-AT14-SP3' agent details. The 'Capabilities' tab is selected, indicated by a red box. The 'User-defined capabilities' section has a '+ Add new capability' button. The 'System capabilities' section lists the following:

Name	Value
Agent.Name	160724-AT14-SP3
Agent.Version	2.107.0
Agent.ComputerName	160724-AT14-SP3

**NOTE**

Microsoft-hosted agents don't display system capabilities. For a list of software installed on Microsoft-hosted agents, see [Use a Microsoft-hosted agent](#).

1. From the **Agent pools** tab, select the desired pool.

Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards

Process

Pipelines  
1

Agent pools  
2

Settings

Deployment pools

Parallel jobs

## Agent pools

Name
Azure Pipelines
Default
Test
Test2

2. Select **Agents** and choose the desired agent.

Default

Jobs Agents Details Security Settings  
1

Name

160724-AT14-SP3 ● Offline
20160317-W10 ● Offline

2

3. Choose the **Capabilities** tab.

160724-AT14-SP3

Jobs Capabilities

User-defined capabilities

No user-defined capabilities

Add a new capability

System capabilities

Search by keyword

Name	Value
Agent.Name	160724-AT14-SP3
Agent.Version	2.107.0
Agent.ComputerName	160724-AT14-SP3

1. From the Agent pools tab, select the desired pool.

Azure DevOps / FabrikamFiber / Organization Settings / Agent pools

Security

Policies

Permissions

Boards

Process

Pipelines

Agent pools

Settings

Deployment pools

Parallel jobs

1

2

Name
Azure Pipelines
Default
Test
Test2

2. Select Agents and choose the desired agent.

Default

Jobs Agents Details Security Settings

Name
160724-AT14-SP3 ● Offline
20160317-W10 ● Offline

3. Choose the **Capabilities** tab.

160724-AT14-SP3

Jobs Capabilities

User-defined capabilities

No user-defined capabilities

Add a new capability

+

System capabilities

Search by keyword

Name	Value
Agent.Name	160724-AT14-SP3
Agent.Version	2.107.0
Agent.ComputerName	160724-AT14-SP3

Select the desired agent, and choose the **Capabilities** tab.

Agents for pool Default

Agents Roles

Enabled	Name	Current Status	X
<input checked="" type="checkbox"/>	20160317-W10	Idle	X
<input checked="" type="checkbox"/>	DEV15-TFS-RTM	Idle	X
<input checked="" type="checkbox"/>	DEV15VS	Idle	X

Requests Capabilities

USER CAPABILITIES

Shows information about user-defined capabilities supported by this host

+ Add capability

Save changes Undo changes

SYSTEM CAPABILITIES

Shows information about the capabilities provided by this host

AegisRoot	C:\AegisTools
Agent.ComputerName	20160317-W10

Select the desired agent, and choose the **Capabilities** tab.

From the **Agent pools** tab, select the desired agent, and choose the **Capabilities** tab.

3. To register a new capability with the agent, choose **Add a new capability**.

#### TIP

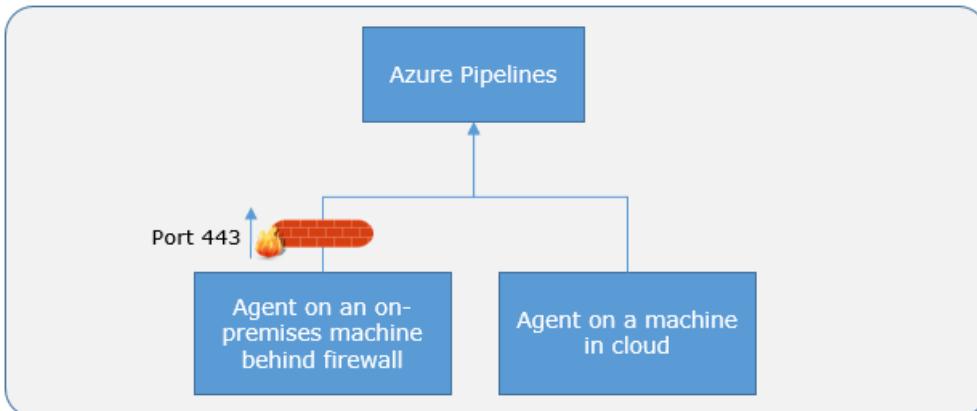
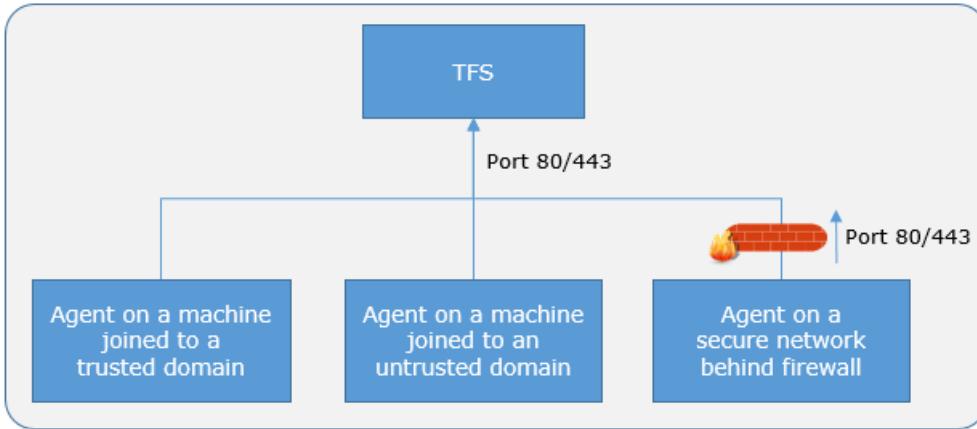
After you install new software on a self-hosted agent, you must restart the agent for the new capability to show up. For more information, see [Restart Windows agent](#), [Restart Linux agent](#), and [Restart Mac agent](#).

## 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 it 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. The agent listens to see if a new job request has been posted for it in the job queue in Azure Pipelines/TFS using an HTTP long poll. 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 (for example, source code) or modify resources (for example, upload test results) on Azure Pipelines or TFS within that job.
3. After 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 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.

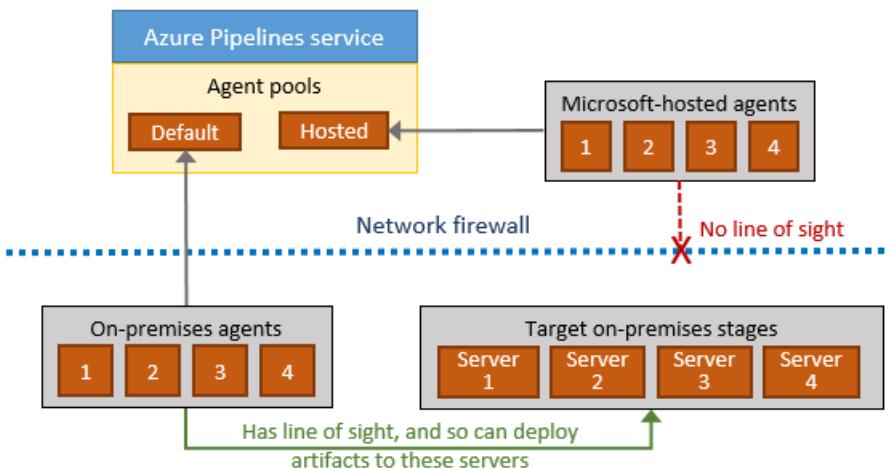
## Communication to deploy to target servers

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.

### NOTE

If your Azure resources are running in an Azure Virtual Network, you can get the [Agent IP ranges](#) where Microsoft-hosted agents are deployed so you can configure the firewall rules for your Azure VNet to allow access by the agent.

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 using the following method:

Your agent can authenticate to Azure DevOps Server 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. The PAT must have **Agent Pools (read, manage)** scope (for a **deployment group** agent, the PAT must have **Deployment group (read, manage)** scope), and while a single PAT can be used for registering multiple agents, the PAT is used only at the time of registering the agent, and not for subsequent [communication](#). For more information, see the [Authenticate with a personal access token \(PAT\)](#) section in the [Windows, Linux, or macOS](#) self-hosted agents articles.

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.

The screenshot shows the IIS Manager interface. In the left pane, under 'Connections', the 'Sites' section is expanded, showing 'Default Web Site', 'Sps DevFabric Web', 'Team Foundation Server', and 'tfs'. The 'tfs' site is selected and highlighted with a red box. In the center pane, the 'Authentication' configuration page is displayed. A table lists five authentication providers: 'Anonymous Authentication' (Enabled), 'ASP.NET Impersonation' (Disabled), 'Basic Authentication' (Disabled), 'Forms Authentication' (Disabled), and 'Windows Authentication' (Enabled). The 'Windows Authentication' row is also highlighted with a red box. On the right side, there's an 'Actions' panel with options like 'Disable', 'Advanced Settings...', and 'Providers...', with 'Providers...' also highlighted with a red box.

This screenshot shows the 'Providers' configuration dialog box overlaid on the IIS Manager interface. The dialog has two main sections: 'Enabled Providers' and 'Available Providers'. In the 'Enabled Providers' section, 'Negotiate' and 'NTLM' are listed, with 'NTLM' highlighted with a red box. Below this, instructions say: 'Select a provider from the list of available providers and click Add to add it to the enabled providers.' In the 'Available Providers' section, there is a dropdown menu and an 'Add' button. At the bottom of the dialog are 'OK' and 'Cancel' buttons. The background of the IIS Manager interface shows the same tree structure as the previous screenshot, with the 'tfs' site selected.

## 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 screenshot shows the IIS Manager interface with the 'Authentication' section selected for the 'tfs' site. The 'Windows Authentication' provider is highlighted with a red box.

Name	Status	Response Type
Anonymous Authentication	Enabled	
ASP.NET Impersonation	Disabled	
Basic Authentication	Disabled	HTTP 401 Challenge
Forms Authentication	Disabled	HTTP 302 Login/Redirect
<b>Windows Authentication</b>	<b>Enabled</b>	<b>HTTP 401 Challenge</b>

The screenshot shows the 'Providers' dialog box for the 'tfs' site. The 'Enabled Providers' list contains 'Negotiate' and 'NTLM', which are highlighted with a red box.

Enabled Providers:

- Negotiate
- NTLM

Select a provider from the list of available providers and click Add to add it to the enabled providers.

Available Providers:

- [List of providers]

## 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. Sign in 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 self-hosted agent as either a service or an interactive process. 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 account

Whether you run an agent as a service or interactively, you can choose which computer 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.

On Windows, you should consider using a service account such as Network Service or Local Service. These accounts have restricted permissions and their passwords don't expire, meaning the agent requires less management over time.

## Agent version and upgrades

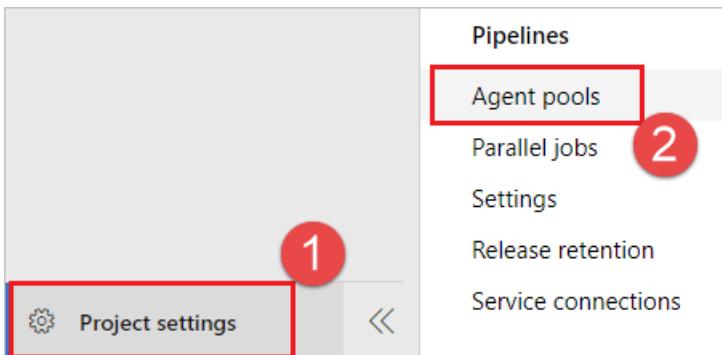
We update the agent software every few weeks in Azure Pipelines. We indicate the agent version in the format `{major}.{minor}`. For instance, if the agent version is `2.1`, then the major version is 2 and the minor version is 1.

Microsoft-hosted agents are always kept up-to-date. If the newer version of the agent is only different in *minor* version, self-hosted agents can usually be updated automatically (configure this setting in **Agent pools**, select your agent, **Settings** - the default is enabled) by Azure Pipelines. An upgrade is requested when a platform feature or one of the tasks used in the pipeline requires a newer version of the agent.

If you run a self-hosted agent interactively, or if there is a newer *major* version of the agent available, then you may have to manually upgrade the agents. You can do this easily from the **Agent pools** tab under your organization. Your pipelines won't run until they can target a compatible agent.

### To update self-hosted agents

1. Navigate to **Project settings**, **Agent pools**.



2. Select your agent pool and choose **Update all agents**.

A screenshot of the 'FabrikamPool' agent pool details page. The 'Jobs' tab is selected. A red box highlights the 'Update all agents' button. Other buttons include 'New agent' and a three-dot menu. The table below shows two jobs:

Name	Project	Agent	Queued	Wait time	Duration
Job 1847784 20200326.1 FabrikamAgents	FabrikamAgents		Yesterday at 1...	<1s	17s
Job 1847770 20200326.1 FabrikamAgents	FabrikamAgents		Yesterday at 1...	2m 53s	13m 10s

You can also update agents individually by choosing **Update agent** from the ... menu.

A screenshot of the 'Agent version' dropdown menu. It shows two entries: '2.163.1' (Enabled) and '2.163.1' (Disabled). A red box highlights the 'Update agent' option in the context menu that appears when clicking the three-dot menu icon next to the first entry.

3. Select **Update** to confirm the update.

A screenshot of the 'Confirm Update' dialog box. It asks if you want to update all agents in the 'FabrikamPool'. It states that running jobs will complete before the update begins and queued jobs will be delayed until the update is complete. Buttons at the bottom are 'Cancel' and 'Update'.

4. An update request is queued for each agent in the pool, that runs when any currently running jobs

complete. Upgrading typically only takes a few moments - long enough to download the latest version of the agent software (approximately 200 MB), unzip it, and restart the agent with the new version. You can monitor the status of your agents on the **Agents** tab.

We update the agent software with every update in Azure DevOps Server and 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 your Azure DevOps Server or TFS server has a newer version of the agent, and that newer agent is only different in *minor* version, it can usually be automatically upgraded. An upgrade is requested when a platform feature or one of the tasks used in the pipeline requires a newer version of the agent. Starting with Azure DevOps Server 2019, you don't have to wait for a new server release. You can [upload a new version of the agent to your application tier](#), and that version will be offered as an upgrade.

If you run the agent interactively, or if there is a newer *major* version of the agent available, then you may have to manually upgrade the agents. You can do this easily from the **Agent pools** tab under your project collection. Your pipelines won't run until they can target a compatible agent.

You can view the version of an agent by navigating to **Agent pools** and selecting the **Capabilities** tab for the desired agent, as described in [Configure agent capabilities](#).

#### NOTE

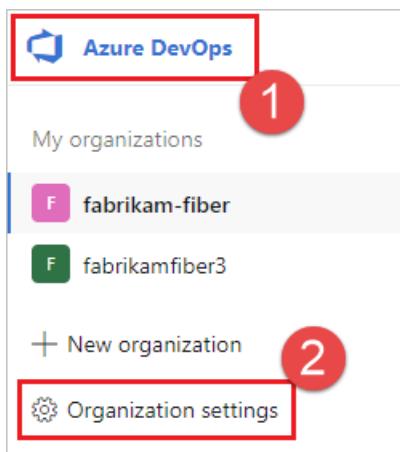
For servers with no internet access, manually copy the agent zip file to  
`C:\ProgramData\Microsoft\Azure DevOps\Agents\` to use as a local file.

## FAQ

### How do I make sure I have the latest v2 agent version?

1. Navigate to the **Agent pools** tab:

1. Choose **Azure DevOps, Organization settings**.



2. Choose **Agent pools**.

Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards  
Process

Pipelines

Agent pools **1**

Settings  
Deployment pools  
Parallel jobs

**Agent pools**

Name
Azure Pipelines
Default
Test
Test2

1. Choose Azure DevOps, Collection settings.

Azure DevOps **1**

Collections  
fabrikam-fiber **2**  
fabrikamfiber3  
+ New organization  
Collection Settings

2. Choose Agent pools.

Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards  
Process

Pipelines

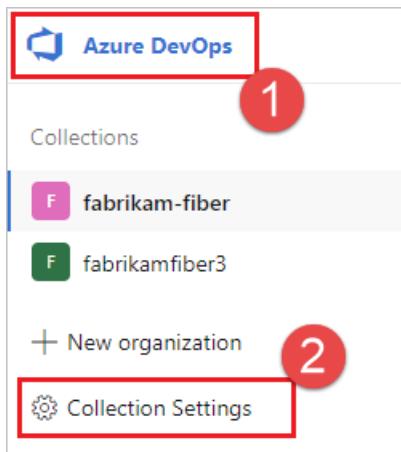
Agent pools **1**

Settings  
Deployment pools  
Parallel jobs

**Agent pools**

Name
Azure Pipelines
Default
Test
Test2

1. Choose Azure DevOps, Collection settings.



2. Choose Agent pools.

A screenshot of the Azure DevOps portal. The top navigation bar shows 'FabrikamFiber / Organization Settings / Agent pools'. On the left, there's a sidebar with the following items:

- Security
- Policies
- Permissions
- Boards
- Process
- Pipelines
- Agent pools (highlighted with a red box)
- Settings
- Deployment pools
- Parallel jobs

The main area is titled 'Agent pools' and lists four entries:

Name
Azure Pipelines
Default
Test
Test2

1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.

A screenshot of the Azure DevOps project settings menu. The top bar has a gear icon (highlighted with a red box) and the text 'Settings Overview'. The menu items are:

- Work
- Security
- Alerts
- Version Control
- Agent Queues (highlighted with a red box)
- Service Hooks

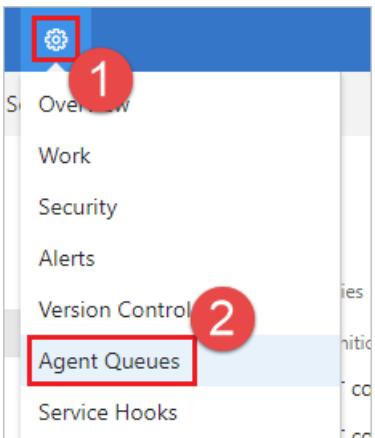
Red numbers 1 and 2 are overlaid on the screen: 1 is above the gear icon, and 2 is above the Agent Queues item.

2. Choose **Manage pools**.

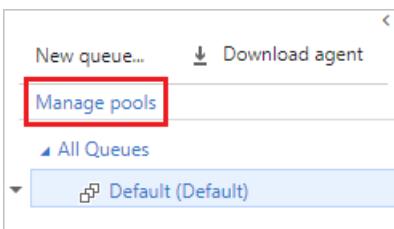
A screenshot of the 'Agent Queues' page. At the top are buttons for 'New queue...' and 'Download agent'. Below them is a list:

- Manage pools (highlighted with a red box)
- All Queues
- Default (Default)

1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



2. Choose **Manage pools**.



1. Navigate to your project and choose **Manage project** (gear icon).



2. Choose **Control panel**.



3. Select **Agent pools**.



2. Click the pool that contains the agent.

3. Make sure the agent is enabled.

4. Navigate to the capabilities tab:

1. From the **Agent pools** tab, select the desired agent pool.

Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions  
Boards  
Process  
Pipelines  
**Agent pools** (1)  
Settings  
Deployment pools  
Parallel jobs

## Agent pools

Name
Azure Pipelines (2)
<b>Default</b> (2)
Test
Test2

2. Select **Agents** and choose the desired agent.

**Default**

Jobs Agents Details Security Settings (1)

Name
160724-AT14-SP3 (2)
● Offline
20160317-W10
● Offline

3. Choose the **Capabilities** tab.

160724-AT14-SP3

Jobs Capabilities

User-defined capabilities

No user-defined capabilities

Add a new capability

System capabilities

Search by keyword

Name	Value
Agent.Name	160724-AT14-SP3
Agent.Version	2.107.0
Agent.ComputerName	160724-AT14-SP3

**NOTE**

Microsoft-hosted agents don't display system capabilities. For a list of software installed on Microsoft-hosted agents, see [Use a Microsoft-hosted agent](#).

- From the Agent pools tab, select the desired pool.

Azure DevOps / Organization Settings / Agent pools

Agent pools

Name
Azure Pipelines
<b>Default</b>
Test
Test2

1

2

- Select Agents and choose the desired agent.

The screenshot shows the 'Agents' tab selected in the top navigation bar. Below the tabs, a table lists two agents. The first agent, '160724-AT14-SP3', is highlighted with a red box and a red circle labeled '2'. It is listed as 'Offline'. The second agent, '20160317-W10', is also listed as 'Offline'.

Name	Status
160724-AT14-SP3	Offline
20160317-W10	Offline

3. Choose the **Capabilities** tab.

The screenshot shows the 'Capabilities' tab selected for the agent '160724-AT14-SP3'. The 'User-defined capabilities' section is currently empty, showing a placeholder message 'No user-defined capabilities' and a 'Add a new capability' button. The 'System capabilities' section displays a table with the following data:

Name	Value
Agent.Name	160724-AT14-SP3
Agent.Version	2.107.0
Agent.ComputerName	160724-AT14-SP3

1. From the **Agent pools** tab, select the desired pool.

Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards

Process

Pipelines  
1

Agent pools  
2

Settings

Deployment pools

Parallel jobs

## Agent pools

Name
Azure Pipelines
Default
Test
Test2

2. Select **Agents** and choose the desired agent.

Default

Jobs Agents Details Security Settings  
1

Name

160724-AT14-SP3 ● Offline
20160317-W10 ● Offline

2

3. Choose the **Capabilities** tab.

160724-AT14-SP3

Jobs Capabilities

User-defined capabilities



No user-defined capabilities

Add a new capability

System capabilities

Search by keyword

Name	Value
Agent.Name	160724-AT14-SP3
Agent.Version	2.107.0
Agent.ComputerName	160724-AT14-SP3

Select the desired agent, and choose the **Capabilities** tab.

Agents for pool Default

Agents Roles

Enabled	Name	Current Status	X
<input checked="" type="checkbox"/>	20160317-W10	Idle	X
<input checked="" type="checkbox"/>	DEV15-TFS-RTM	Idle	X
<input checked="" type="checkbox"/>	DEV15VS	Idle	X

1 Requests Capabilities 2

**USER CAPABILITIES**  
Shows information about user-defined capabilities supported by this host  
[+ Add capability](#)

[Save changes](#) [Undo changes](#)

**SYSTEM CAPABILITIES**  
Shows information about the capabilities provided by this host

AegisRoot	C:\AegisTools
Agent.ComputerName	20160317-W10

Select the desired agent, and choose the **Capabilities** tab.

Agents for pool Default

Agents Roles

Enabled	Name	Current Status	X
<input checked="" type="checkbox"/>	20160317-W10	Idle	X
<input checked="" type="checkbox"/>	DEV15-TFS-RTM	Idle	X
<input checked="" type="checkbox"/>	DEV15VS	Idle	X

1 Requests Capabilities 2

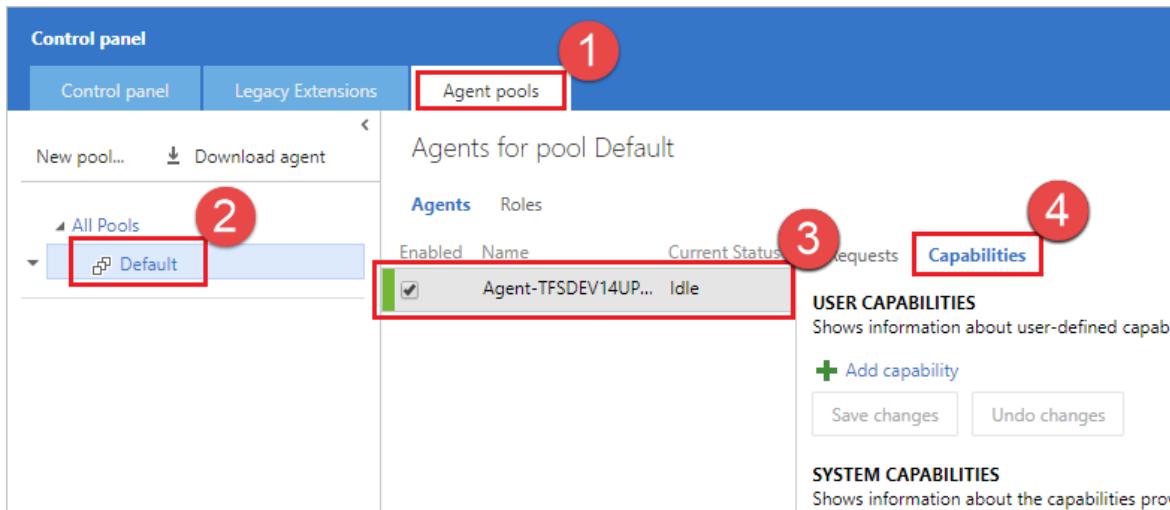
**USER CAPABILITIES**  
Shows information about user-defined capabilities supported by this host  
[+ Add capability](#)

[Save changes](#) [Undo changes](#)

**SYSTEM CAPABILITIES**  
Shows information about the capabilities provided by this host

AegisRoot	C:\AegisTools
Agent.ComputerName	20160317-W10

From the Agent pools tab, select the desired agent, and choose the Capabilities tab.



5. 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.
6. Each agent automatically updates itself when it runs a task that requires a newer version of the agent. If you want to manually update some agents, right-click the pool, and select **Update all agents**.

#### Can I update my v2 agents that are part of an Azure DevOps Server pool?

Yes. Beginning with Azure DevOps Server 2019, you can configure your server to look for the agent package files on a local disk. This configuration will override the default version that came with the server at the time of its release. This scenario also applies when the server doesn't 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 by using a method of your choice (such as USB drive, Network transfer, and so on). Place the agent files under the `%ProgramData%\Microsoft\Azure DevOps\Agents` folder.
3. You're all set! Your Azure DevOps Server will now use the local files whenever the agents are 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**.

#### 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, if you define a 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 many shared resources. For example, you could try it for agents that run releases that mostly orchestrate deployments and don't do much 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 much 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.

### What's the behavior of agents when the pipeline jobs are cancelled?

For Microsoft-hosted agents, the agent is torn down and returned to the Azure Pipelines pool.

For self-hosted agents:

When a pipeline is cancelled, the agent sends a sequence of commands to the process executing the current step. The first command is sent with a timeout of 7.5 seconds. If the process has not terminated, a second command is sent with a timeout of 2.5 seconds. If the process has not terminated, the agent issues a command to kill the process. If the process does not honor the two initial termination requests, it will be killed. From the initial request to termination takes approximately 10 seconds.

The commands issued to the process are different based on the agent operating system.

- macOS and Linux - The commands sent are SIGINT, followed by SIGTERM, followed by SIGKILL.
- Windows - The commands sent to the process are Ctrl+C, followed by Ctrl+Break, followed by Process.Kill.

#### NOTE

Azure Pipelines Agent is open source on [GitHub](#).

## Learn more

For more information about agents, see the following modules from the [Build applications with Azure DevOps](#) learning path.

- [Choose a Microsoft-hosted or self-hosted build agent](#)
- [Host your own build agent in Azure Pipelines](#)

# Agent pools

3/26/2021 • 18 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

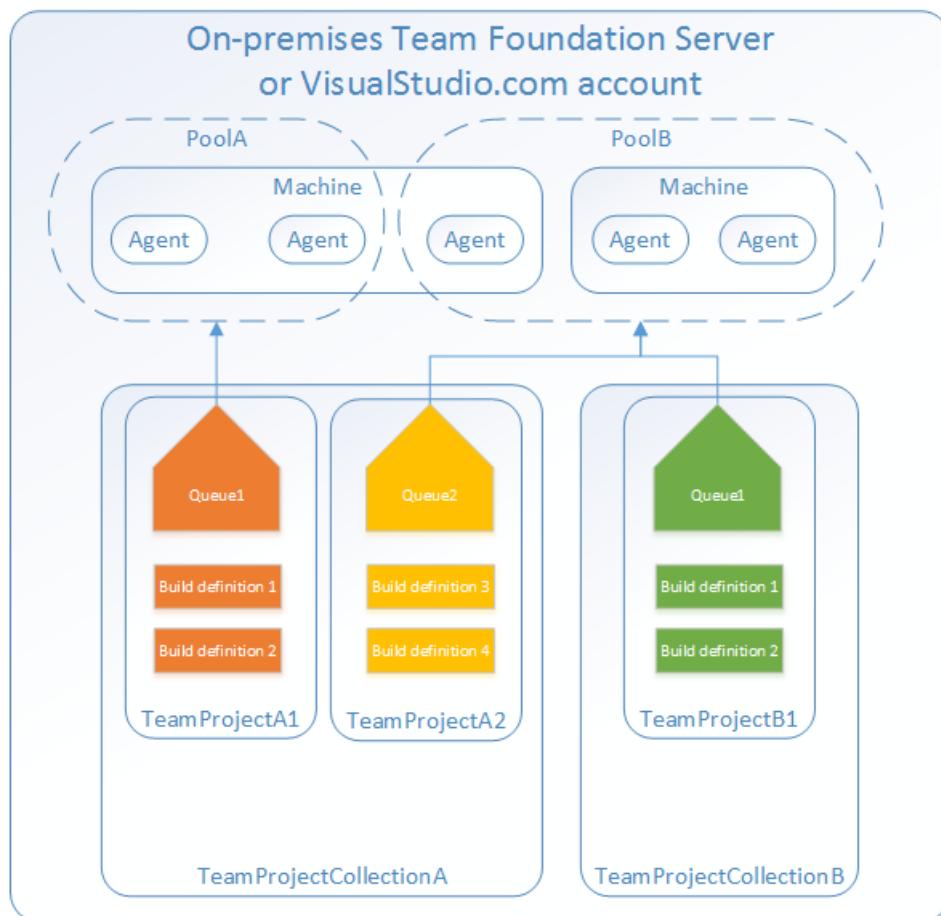
## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

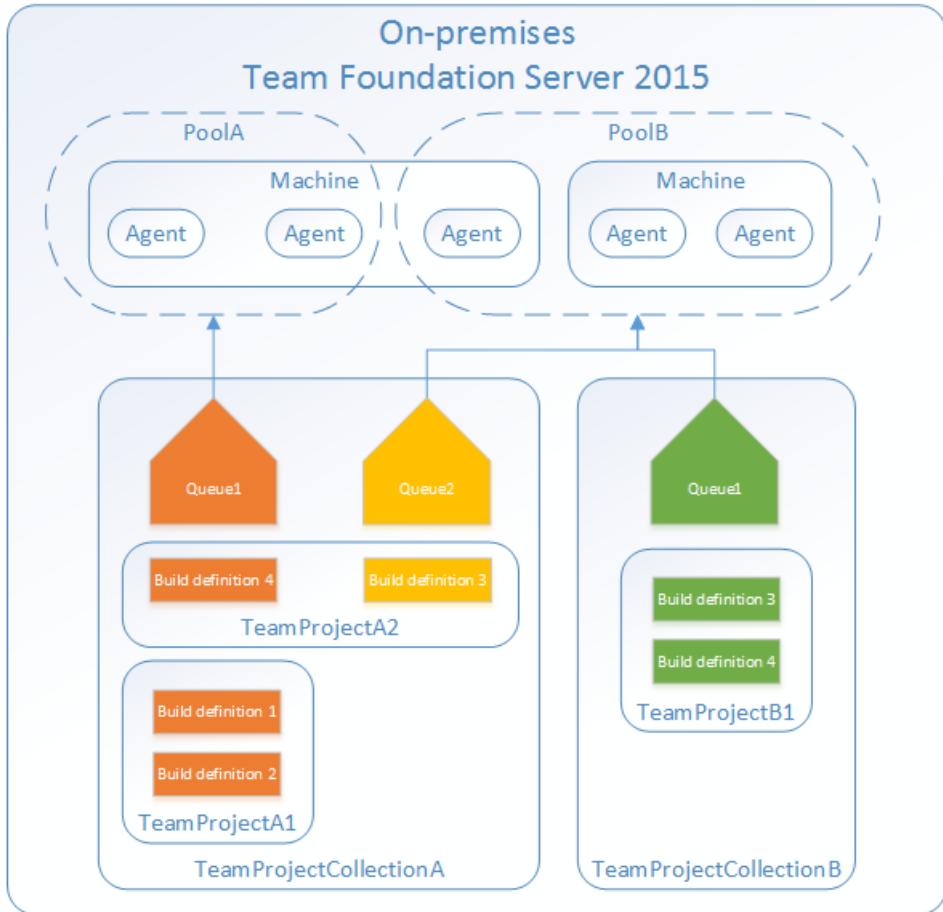
Instead of managing each **agent** individually, you organize agents into **agent pools**. In TFS, pools are scoped to the entire server; so you can share an agent pool across project collections and projects.

An **agent queue** provides access to an **agent pool** within a project. When you create a build or release pipeline, you specify which queue it uses. Queues are scoped to your project in TFS 2017 and newer, 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 an agent queue pointing to the same agent pool. While multiple queues across projects can use the same agent pool, multiple queues within a project cannot use the same agent pool. Also, each agent queue can use only one agent pool.



Agent pools are scoped to project collections.



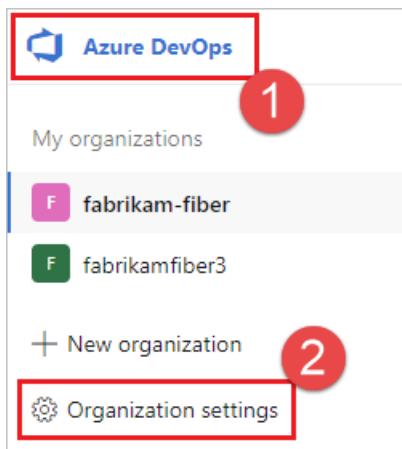
Instead of managing each **agent** individually, you organize agents into **agent pools**. In Azure Pipelines, pools are scoped to the entire organization; so you can share the agent machines across projects. In Azure DevOps Server, agent pools are scoped to the entire server; so you can share the agent machines across projects and collections.

When you configure an agent, it is registered with a single pool, and when you create a pipeline, you specify which pool the pipeline uses. When you run the pipeline, it runs on an agent from that pool that meets the **demands** of the pipeline.

You create and manage agent pools from the agent pools tab in admin settings.

If you are an organization administrator, you create and manage agent pools from the agent pools tab in admin settings.

1. Choose **Azure DevOps, Organization settings**.



2. Choose **Agent pools**.

 Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards  
Process

Pipelines

Agent pools  

Settings  
Deployment pools  
Parallel jobs

## Agent pools

Name
Azure Pipelines Azure Pipelines
Default
Test
Test2

1. Choose Azure DevOps, Collection settings.

 Azure DevOps 1

Collections

fabrikam-fiber 2

fabrikamfiber3

+ New organization

Collection Settings 2

2. Choose Agent pools.

 Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards  
Process

Pipelines

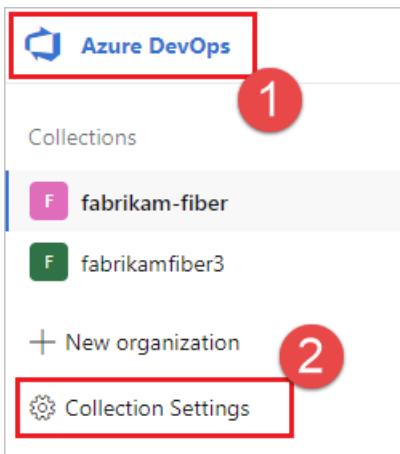
Agent pools  

Settings  
Deployment pools  
Parallel jobs

## Agent pools

Name
Azure Pipelines Azure Pipelines
Default
Test
Test2

1. Choose Azure DevOps, Collection settings.



2. Choose Agent pools.

FabrikamFiber / Organization Settings / Agent pools

Agent pools

Name
Azure Pipelines Azure Pipelines
Default
Test
Test2

1. Navigate to your project and choose Settings (gear icon) > Agent Queues.

S... Overview Work Security Alerts Version Control Agent Queues Service Hooks

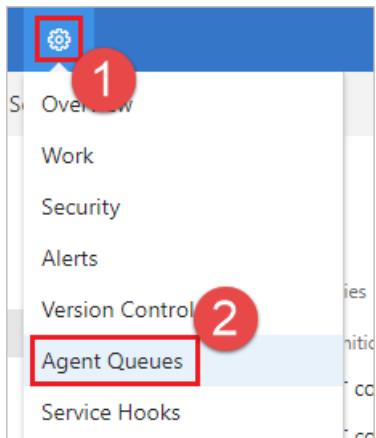
2. Choose Manage pools.

New queue... Download agent

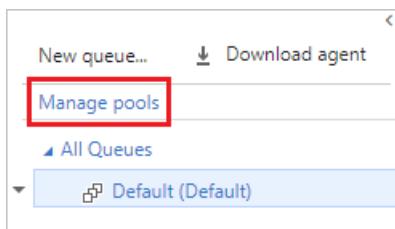
Manage pools

All Queues Default (Default)

1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



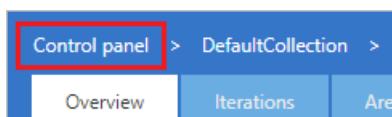
2. Choose **Manage pools**.



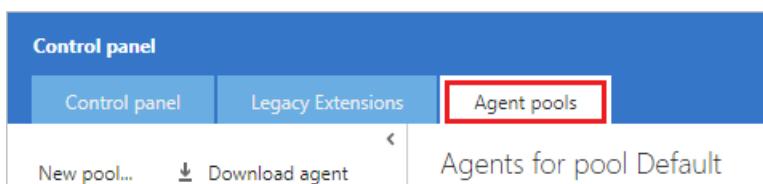
1. Navigate to your project and choose **Manage project** (gear icon).



2. Choose **Control panel**.



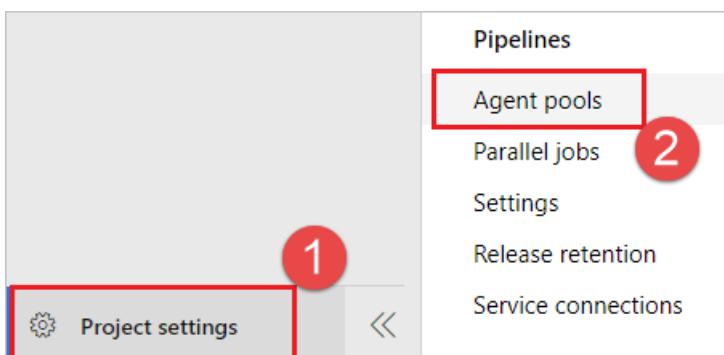
3. Select **Agent pools**.



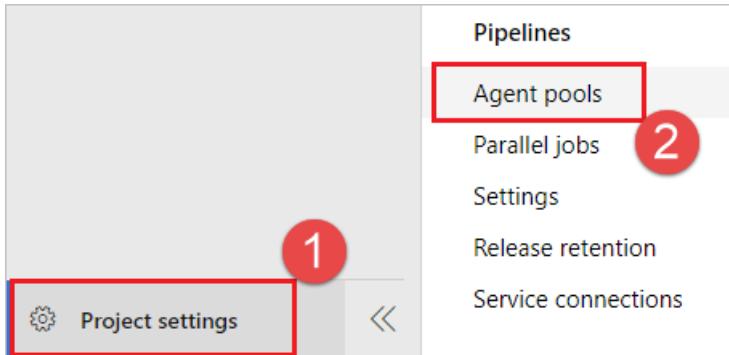
You create and manage agent queues from the agent queues tab in project settings.

If you are a project team member, you create and manage agent queues from the agent pools tab in project settings.

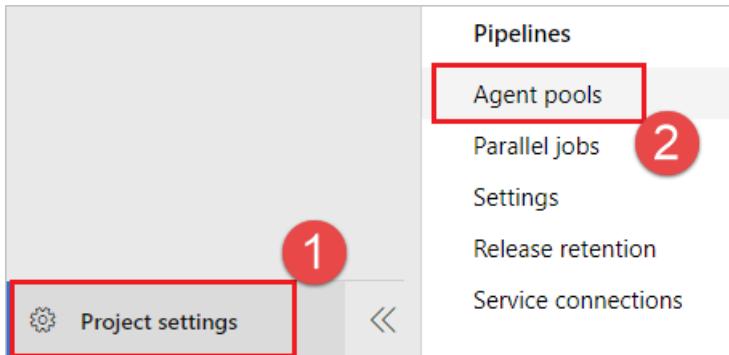
Navigate to your project and choose **Project settings**, **Agent pools**.



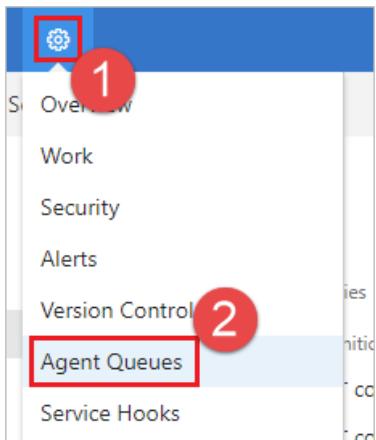
Navigate to your project and choose **Project settings**, **Agent pools**.



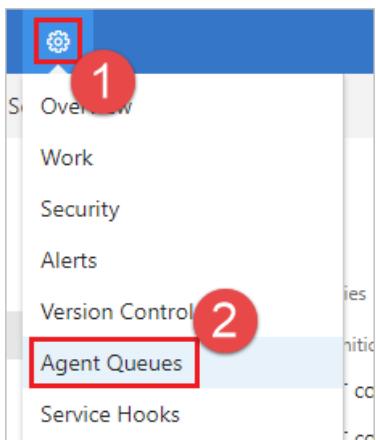
Navigate to your project and choose **Project settings**, **Agent pools**.



Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



1. Navigate to your project and choose **Manage project** (gear icon).



2. Choose Control panel.



3. Select the desired project collection, and choose **View the collection administration page**.

A screenshot of the 'Collection administration' page for 'DefaultCollection'. The page has a blue header with tabs: 'Control panel' (selected), 'Legacy Extensions', and 'Agent pools'. Below the header, there's a section titled 'TEAM FOUNDATION' with a search bar and a list of collections. The 'DefaultCollection' entry is highlighted with a red box and a red number '1' in a circle to its right. To the right of the list, there are details about the collection: 'Collection name: DefaultCollection', 'Status: Online', and 'Team projects: 1'. At the bottom, there's a section titled 'Administration tasks' with two links: 'Manage collection security and group membership' and 'View the collection administration page', both of which are highlighted with a red box and a red number '2' in a circle to their left.

a. Select Agent Queues (For TFS 2015, Select Build and then Queues).

A screenshot of the 'Agent queues' page for 'DefaultCollection'. The page has a blue header with tabs: 'Control panel &gt; DefaultCollection', 'Overview', 'Security', 'Build' (selected), 'Agent queues' (highlighted with a red box and a red number '1' in a circle to its right), and 'Extensions\*'. On the left, there's a sidebar with 'New queue...', 'Download agent', 'Manage pools', and 'All Queues' sections. The 'Default (Default)' queue is selected and highlighted with a red box and a red number '2' in a circle to its right. On the right, there's a table titled 'Agents for pool Default' showing one agent: 'Agent-TFSDEV14UP...' with 'Idle' status.

## Default agent pools

The following agent pools are provided by default:

- **Default** pool: Use it to register [self-hosted agents](#) that you've set up.
- **Azure Pipelines** hosted pool with various Windows, Linux, and macOS images. For a complete list of the available images and their installed software, see [Microsoft-hosted agents](#).

## NOTE

The Azure Pipelines hosted pool replaces the previous hosted pools that had names that mapped to the corresponding images. Any jobs you had in the previous hosted pools are automatically redirected to the correct image in the new Azure Pipelines hosted pool. In some circumstances, you may still see the old pool names, but behind the scenes the hosted jobs are run using the Azure Pipelines pool. For more information, see the [Single hosted pool release notes from the July 1 2019 - Sprint 154 release notes](#).

By default, all contributors in a project are members of the **User** role on hosted pools. This allows every contributor in a project to author and run pipelines using Microsoft-hosted agents.

## Choosing a pool and agent in your pipeline

- [YAML](#)
- [Classic](#)

To choose a Microsoft-hosted agent from the Azure Pipelines pool in your Azure DevOps Services YAML pipeline, specify the name of the image, using the **YAML VM Image Label** from [this table](#).

```
pool:
 vmImage: ubuntu-16.04
```

To use a private pool with no demands:

```
pool: MyPool
```

For more information, see the [YAML schema](#) for **pools**.

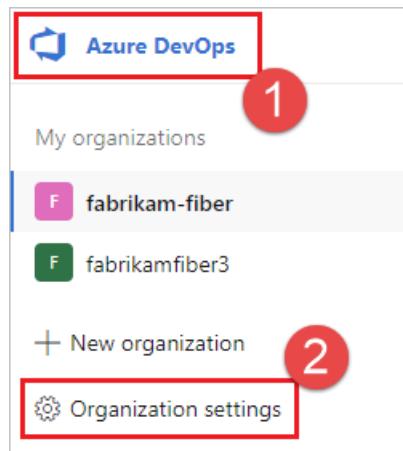
## Managing pools and queues

- [Browser](#)
- [Azure DevOps CLI](#)

You create and manage agent pools from the agent pools tab in admin settings.

If you are an organization administrator, you create and manage agent pools from the agent pools tab in admin settings.

### 1. Choose **Azure DevOps, Organization settings**.



### 2. Choose **Agent pools**.

 Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards  
Process

Pipelines

Agent pools  

Settings  
Deployment pools  
Parallel jobs

## Agent pools

Name
Azure Pipelines Azure Pipelines
Default
Test
Test2

1. Choose Azure DevOps, Collection settings.

 Azure DevOps 1

Collections

fabrikam-fiber 2

fabrikamfiber3

+ New organization

Collection Settings 2

2. Choose Agent pools.

 Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards  
Process

Pipelines

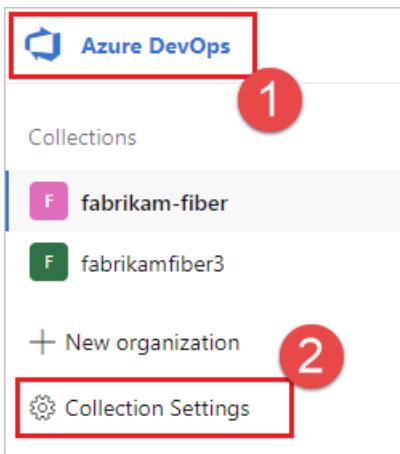
Agent pools  

Settings  
Deployment pools  
Parallel jobs

## Agent pools

Name
Azure Pipelines Azure Pipelines
Default
Test
Test2

1. Choose Azure DevOps, Collection settings.



2. Choose Agent pools.

FabrikamFiber / Organization Settings / Agent pools

Agent pools

Name
Azure Pipelines Azure Pipelines
Default
Test
Test2

1. Navigate to your project and choose Settings (gear icon) > Agent Queues.

S... Overview Work Security Alerts Version Control Agent Queues Service Hooks

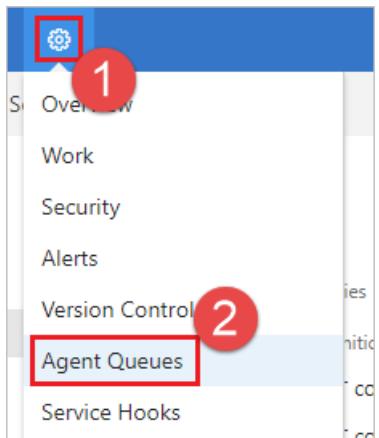
2. Choose Manage pools.

New queue... Download agent

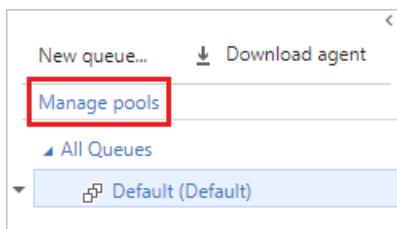
Manage pools

All Queues Default (Default)

1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



2. Choose **Manage pools**.



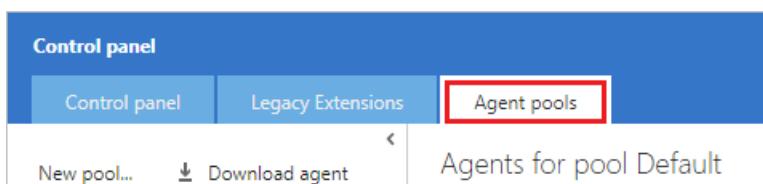
1. Navigate to your project and choose **Manage project** (gear icon).



2. Choose **Control panel**.



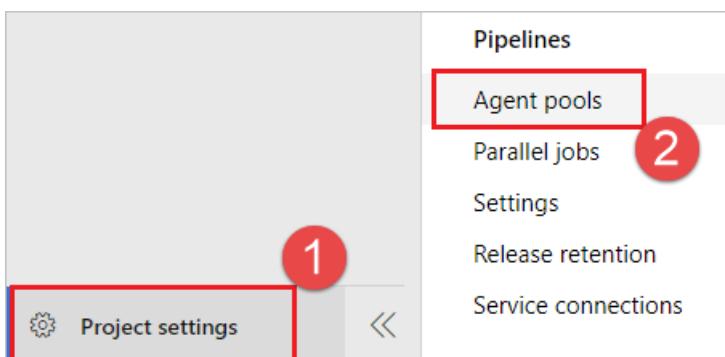
3. Select **Agent pools**.



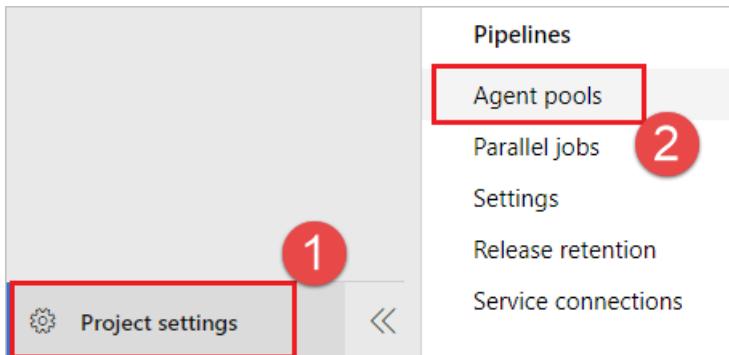
You create and manage agent queues from the agent queues tab in project settings.

If you are a project team member, you create and manage agent queues from the agent pools tab in project settings.

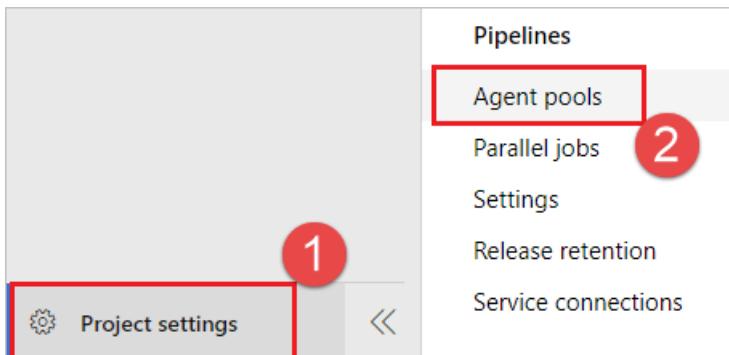
Navigate to your project and choose **Project settings**, **Agent pools**.



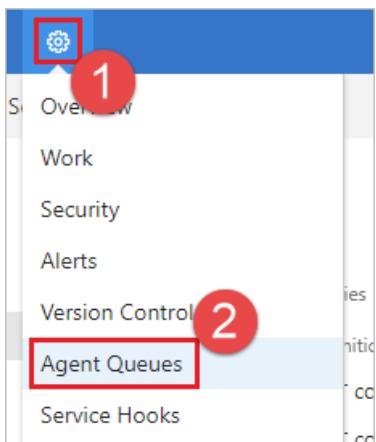
Navigate to your project and choose **Project settings**, **Agent pools**.



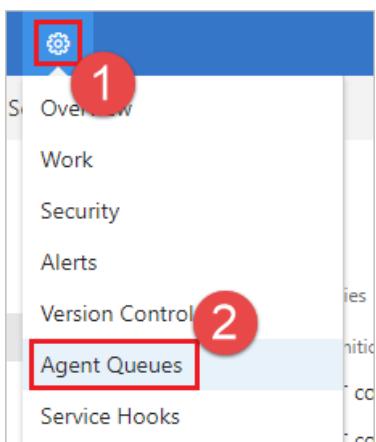
Navigate to your project and choose **Project settings**, **Agent pools**.



Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



1. Navigate to your project and choose **Manage project** (gear icon).



2. Choose Control panel.



3. Select the desired project collection, and choose **View the collection administration page**.

A screenshot of the "Collection administration" page for the "DefaultCollection". The left sidebar shows a search bar and a list of collections, with "DefaultCollection" selected. The main pane displays collection details: "Collection name: DefaultCollection", "Status: Online", and "Team projects: 1". Under "Administration tasks", there are two links: "Manage collection security and group membership" and "View the collection administration page". A red circle with the number "1" is over the collection name, and another red circle with the number "2" is over the "View the collection administration page" link.

a. Select Agent Queues (For TFS 2015, Select Build and then Queues).

A screenshot of the "Agent queues" page for the "DefaultCollection". The top navigation bar has tabs: "Overview", "Security", "Build", "Agent queues" (highlighted in red), and "Extensions\*". The left sidebar shows "New queue...", "Download agent", "Manage pools", and "All Queues" (with "Default (Default)" selected). The main pane shows "Agents for pool Default" with a table of agents. A red circle with the number "1" is over the "Agent queues" tab, and another red circle with the number "2" is over the "Default (Default)" queue in the sidebar.

Pools are used to run jobs. Learn about [specifying pools for jobs](#).

If you've got a lot of self-hosted 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 self-hosted 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've the permissions to create pools in your project by selecting **Security** on the agent pools page in your project settings. You must have **Administrator** role to be able to create new pools. Next, select **Add pool** and select the option to create a **new** pool at the organization level. 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 agent pools** with the **Administrator** role by navigating to agent pools page in your organization settings. Next create a **New agent pool** and select the option to **Auto-provision corresponding agent pools in all projects** while creating the pool. This setting ensures all projects have access to this agent pool. 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, navigate to the settings for one of the projects, add an agent pool, and select the option to create a **new** pool at the organization level. Next, go to each of the other projects, and create a pool in each of them while selecting the option to **Use an existing agent pool from the organization**. Finally, [install](#) and configure agents to be part of the shared agent pool.
- 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.

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 AGENT POOL IN ORGANIZATION SETTINGS	PURPOSE
Reader	Members of this role can view the agent pool as well as agents. You typically use this to add operators that are responsible for monitoring the agents and their health.
Service Account	Members of this role can use the organization agent pool to create a project agent pool in a project. If you follow the guidelines above for creating new project agent pools, you typically do not have to add any members here.

ROLE ON AN AGENT POOL IN ORGANIZATION SETTINGS	PURPOSE
Administrator	In addition to all the above permissions, members of this role can register or unregister agents from the organization agent pool. They can also refer to the organization agent pool when creating a project agent pool in a project. Finally, they can also manage membership for all roles of the organization agent pool. The user that created the organization agent pool is automatically added to the Administrator role for that pool.

The **All agent pools** node in the Agent Pools tab is used to control the security of *all* organization agent pools. Role memberships for individual organization agent pools are automatically inherited from those of the 'All agent pools' node. When using TFS or Azure DevOps Server, by default, TFS and Azure DevOps Server administrators are also administrators of the 'All agent pools' node.

Roles are also defined on each project agent pool, and memberships in these roles govern what operations you can perform on an agent pool at the project level.

ROLE ON AN AGENT POOL IN PROJECT SETTINGS	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 pipelines.
Administrator	In addition to all the above operations, members of this role can manage membership for all roles of the project agent pool. The user that created the pool is automatically added to the Administrator role for that pool.

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

The **Security** action 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 what you define here. 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.

## FAQ

### If I don't schedule a maintenance window, when will the agents run maintenance?

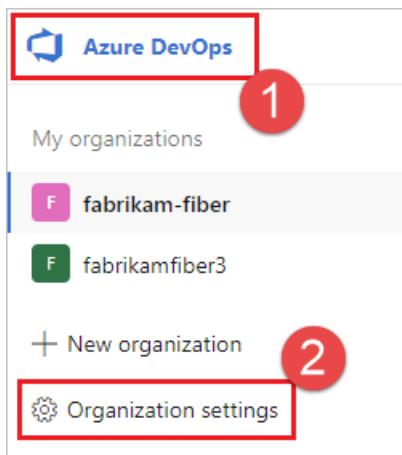
If no window is scheduled, then the agents in that pool will not run the maintenance job.

### What is a maintenance job?

You can configure agent pools to periodically clean up stale working directories and repositories. This should reduce the potential for the agents to run out of disk space. Maintenance jobs are configured at the project collection or organization level in agent pool settings.

To configure maintenance job settings:

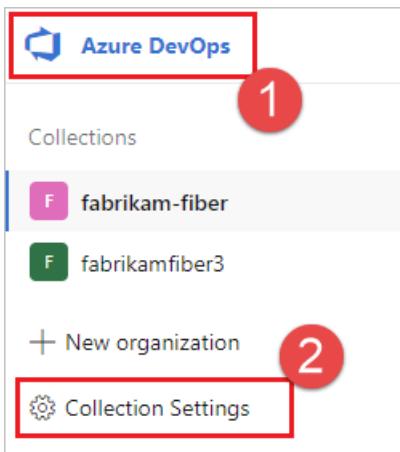
1. Choose Azure DevOps, Organization settings.



2. Choose Agent pools.

A screenshot of the Azure DevOps Agent pools settings page. The top navigation bar shows 'FabrikamFiber / Organization Settings / Agent pools'. On the left is a sidebar with various options: Security, Policies, Permissions, Boards, Process, Pipelines, Agent pools (which is highlighted with a red box and labeled '2'), Settings, Deployment pools, and Parallel jobs. The main content area is titled 'Agent pools' and lists four entries: 'Azure Pipelines' (with a cloud icon), 'Default' (with a monitor icon), 'Test' (with a monitor icon), and 'Test2' (with a monitor icon).

1. Choose Azure DevOps, Collection settings.



2. Choose Agent pools.

The screenshot shows the 'Agent pools' page in the Azure DevOps portal. The left sidebar has 'Agent pools' selected, highlighted with a red box and a red circle containing the number '1'. The main area displays a list of agent pools:

Name
Azure Pipelines Azure Pipelines
Default
Test
Test2

The 'Default' pool is highlighted with a red box and a red circle containing the number '2'.

1. Choose Azure DevOps, Collection settings.

The screenshot shows the 'Collections' page in the Azure DevOps portal. At the top, the 'Azure DevOps' header is highlighted with a red box and a red circle containing the number '1'. Below it, there are two collections: 'fabrikam-fiber' (purple icon) and 'fabrikamfiber3' (green icon). A red box highlights the 'Collection Settings' button at the bottom, and a red circle containing the number '2' is positioned next to it.

2. Choose Agent pools.

Screenshot of the Azure DevOps Organization Settings page showing the Agent pools section.

The left sidebar shows the following navigation items:

- Security
- Policies
- Permissions
- Boards
- Process
- Pipelines
- Agent pools** (highlighted with a red box)
- Settings
- Deployment pools
- Parallel jobs

The main content area is titled "Agent pools" and displays a list of agent pools:

Name
Azure Pipelines
Default
Test
Test2

1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.

Screenshot of the Project Settings page showing the Agent Queues section.

The left sidebar shows the following navigation items:

- Overview (highlighted with a red box)
- Work
- Security
- Alerts
- Version Control
- Agent Queues** (highlighted with a red box)
- Service Hooks

2. Choose **Manage pools**.

Screenshot of the Agent Queues page showing the Manage pools section.

The top navigation bar includes:

- New queue...
- Download agent

The main content area shows:

- Manage pools** (highlighted with a red box)
- All Queues
- Default (Default)

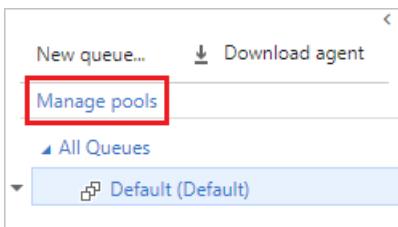
1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.

Screenshot of the Project Settings page showing the Agent Queues section.

The left sidebar shows the following navigation items:

- Overview (highlighted with a red box)
- Work
- Security
- Alerts
- Version Control
- Agent Queues** (highlighted with a red box)
- Service Hooks

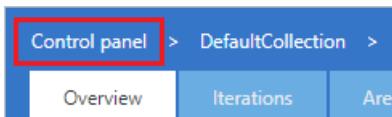
2. Choose **Manage pools**.



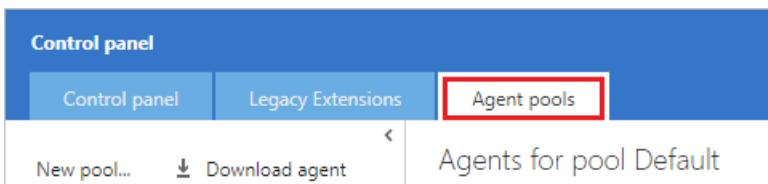
1. Navigate to your project and choose **Manage project** (gear icon).



2. Choose **Control panel**.



3. Select **Agent pools**.



Choose the desired pool and choose **Settings** to configure maintenance job settings for that agent pool.

**IMPORTANT**

You must have the [Manage build queues](#) permission to configure maintenance job settings. If you don't see the **Settings** tab or the **Maintenance History** tab, you don't have that permission, which is granted by default to the **Administrator** role. For more information, see [Security of agent pools](#).

fabrikam-tailspin / Settings / Agent pools / FabrikamPool

Search

Jobs Agents Details Security **Settings** Maintenance History

### Maintenance job

Enable agent maintenance job

#### Settings

Maintenance job timeout (minutes):  
60

Maximum percentage of agents running maintenance concurrently:  
25

Number of maintenance job records to keep:  
10

Days to keep unused working directories (0 indicates forever):  
30

#### Scheduling

Start maintenance job at this time: 01 : 00 (UTC) Coordinated Universal Time

Run maintenance job on these days:  
 Mon  Tues  Wed  Thurs  Fri  Sat  Sun

1

2

3

ting Projects Favorites Work items Pull requests ...

Overview Settings Security Process Build **Agent Pools** Notifications Usage

New pool... **Agent Pools**

All Pools Default Hosted

Agents Roles **Settings** Maintenance history

**Maintenance setting**

Enable agent maintenance job  
Maintenance job timeout: 60 minutes  
Max percentage of agents running maintenance: 25 percent ⓘ  
Number of maintenance job records to keep: 10  
 Delete unused working directory after 30 days  
Schedule pool maintenance:  
24h time: 00 : 00 (UTC-12:00) International Date Line West  
Fri  Mon  Tue  Wed  Thu  Fri  Sat

Save changes Undo changes

Configure your desired settings and choose **Save**.

Select **Maintenance History** to see the maintenance job history for the current agent pool. You can download and review logs to see the cleaning steps and actions taken.

The screenshot shows the 'FabrikamPool' agent pool page. At the top, there are tabs for 'Jobs', 'Agents', 'Details', 'Security', 'Settings', and 'Maintenance History'. The 'Maintenance History' tab is selected and highlighted with a red box. Below the tabs is a table with columns: ID, Errors and war..., Queued, Started, Duration, Target agents, and Logs. A single job named 'Job 1' is listed. The 'Logs' column for this job also has a red box around it, and a link 'Download logs...' is visible.

ID	Errors and war...	Queued	Started	Duration	Target agents	Logs
Job 1	0 errors, 0 war...	Today at 6:04...	Today at 6:04...	52s	3 agents	<a href="#">Download logs...</a>

The maintenance is done per agent pool, not per machine; so if you have multiple agent pools on a single machine, you may still run into disk space issues.

**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 Azure Pipelines pool provides 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](#)

# Historical graph for agent pools (Preview)

3/6/2021 • 4 minutes to read • [Edit Online](#)

The pool consumption report enables you to view jobs running in your agent pools graphed with agent pool job concurrency over a span of up to 30 days. You can use this information to help decide whether your jobs aren't running because of concurrency limits. If you have many jobs queued or running jobs at the concurrency or online agents limit, you may wish to [purchase additional parallel jobs](#) or provision more self-hosted agents.

## Prerequisites

Historical graph for agent pools is in preview. To enable or disable this preview feature, navigate to [Preview features](#), find **Historical graph for agent pools**, and choose the desired setting.

Historical graph for agent pools  On

Turn on the new historical graph for pools feature. [Learn more](#)

## Pool consumption report

The pool consumption report is part of the **Analytics** tab for an agent pool and contains the following charts, depending on the agent pool type.

CHART TYPE	DESCRIPTION	AGENT POOL TYPE
Public hosted concurrency	Displays concurrency, queued jobs, and running jobs for public projects	Microsoft-hosted
Private hosted concurrency	Displays concurrency, queued jobs, and running jobs for private projects	Microsoft-hosted
Agent usage	Displays online agents, queued jobs, and running jobs for self-hosted agents	Scale set agent and self-hosted
Private self-hosted concurrency	Displays concurrency, queued jobs, and running jobs for private self-hosted projects	Scale set agent and self-hosted

The charts in the pool consumption report graph the following data points:

- **Concurrency** - The number of parallel jobs in the organization that apply to the project type (public or private) and agent pool type (Microsoft-hosted or self-hosted). For more information, see [Configure and pay for parallel jobs](#).
- **Online agents** - The number of agents online in a [self-hosted agent pool](#) or a [scale set agent pool](#).
- **Queued jobs** - The number of jobs queued and waiting for an agent.
- **Running jobs** - The number of running jobs.

Pool data is aggregated at a granularity of 10 minutes, and the number of running jobs is plotted based on the maximum number of running jobs for the specified interval of time. Because multiple short-running jobs may complete within the 10 minute timeline, the count of running jobs may sometimes be higher than the

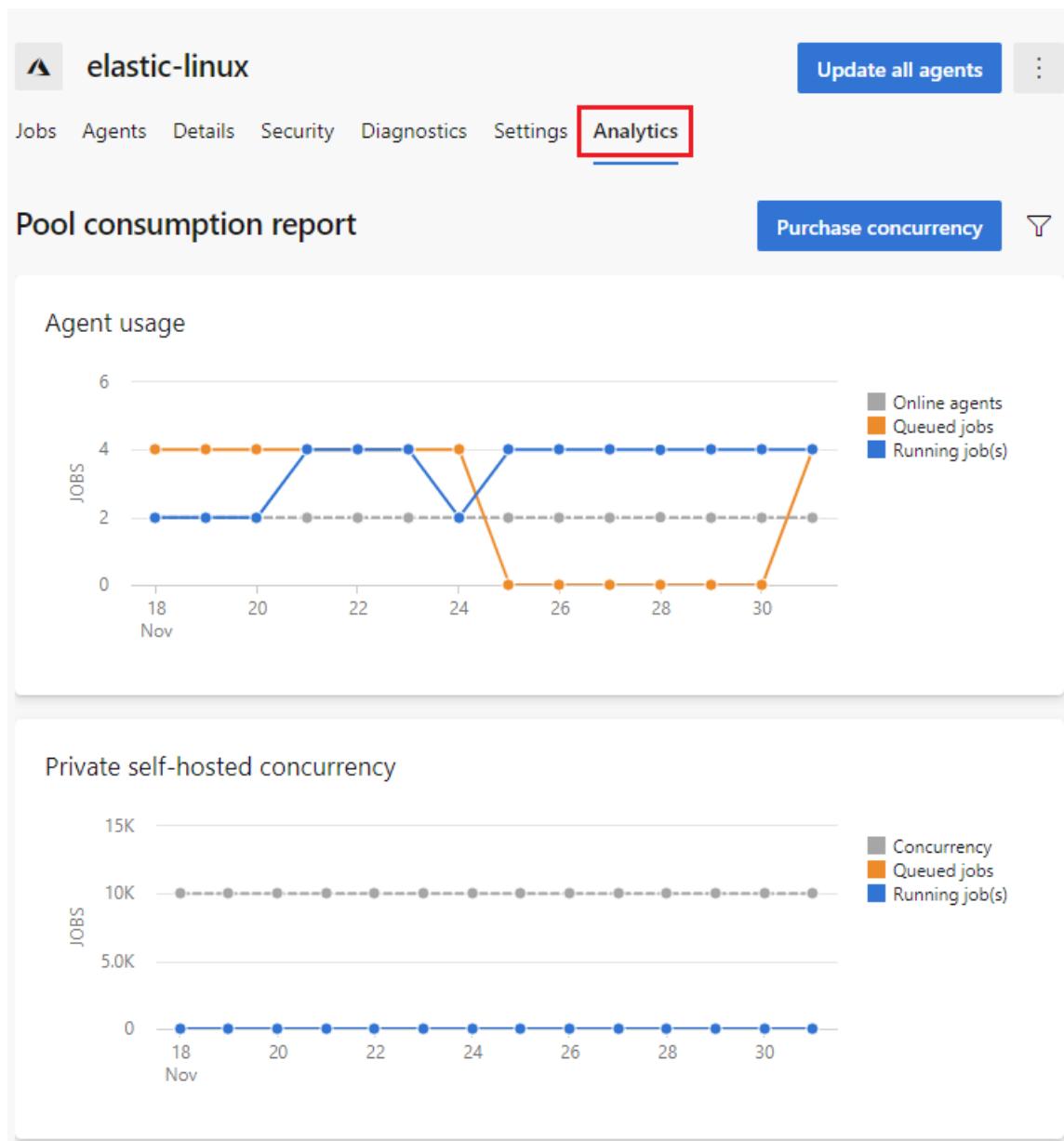
concurrency or online agents during that same period.

## Report scope

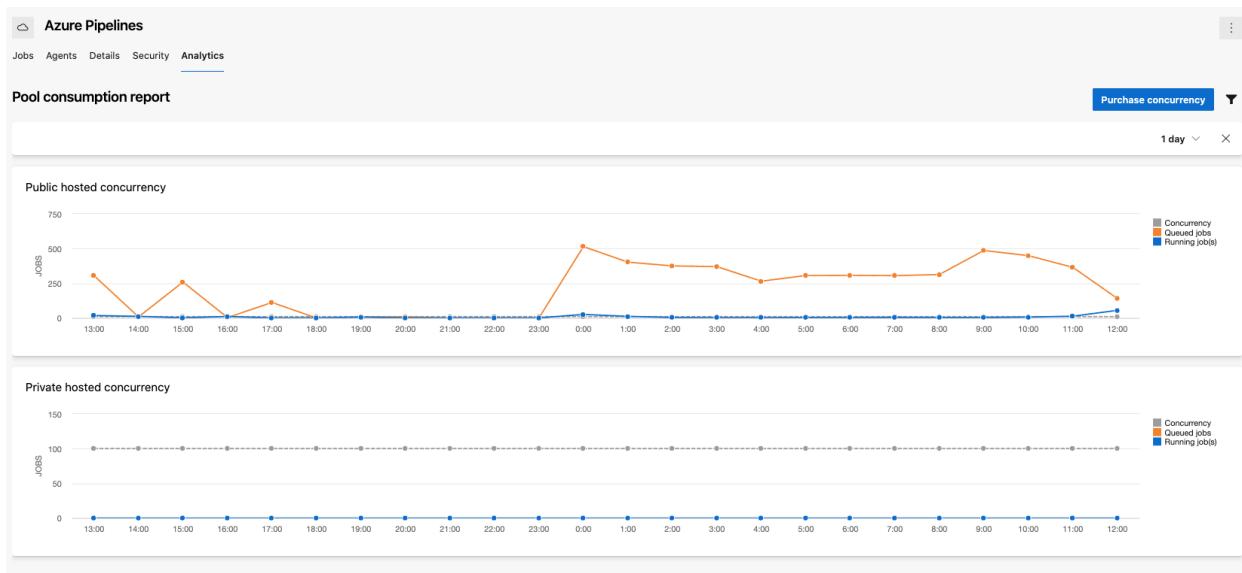
The pool consumption report can be displayed at organization scope, or project scope. At the organization level, the chart is plotted using data from pipelines across any project within the organization that have run jobs in that pool. At the project level, the chart is plotted using data from pipelines in that particular project that have run jobs in that pool.

- To view the pool consumption report at the organization level, choose **Organization settings, Pipelines, Agent pools**.
- To view the pool consumption report at the project level, navigate to the desired project and choose **Project settings, Pipelines, Agent pools**.

From the **Agent pools** view, choose the desired pool, and view the **Analytics** tab. The following example shows the pool consumption report for a self-hosted agent pool.

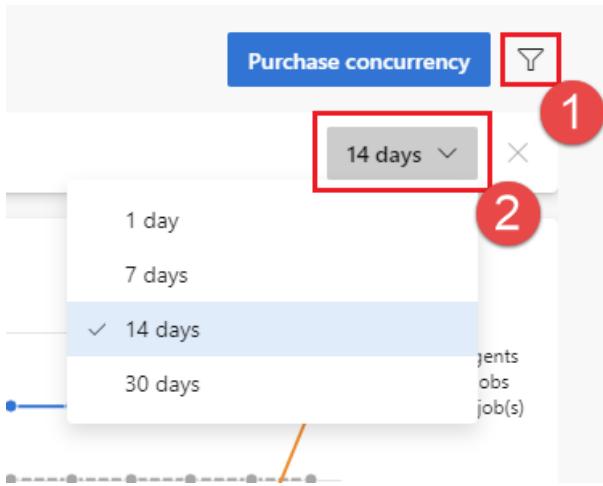


This example shows the usage graphs for the Azure Pipelines Microsoft-hosted agent pool.

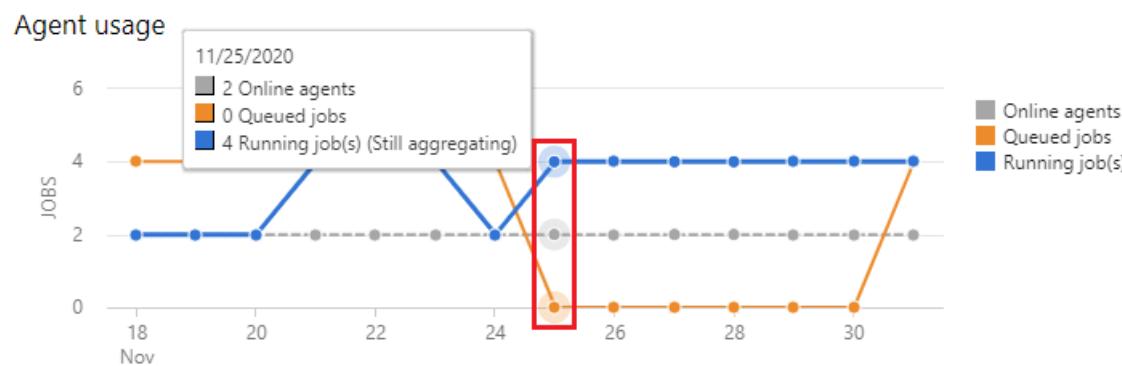


## Filtering

To adjust the timeline of the graph, choose **Filter** , select the interval drop-down, and choose the desired interval.



For the 1 day interval, you can view data per hour, and for the other intervals you can view it per day. Pool data is aggregated at a granularity of 10 minutes, and the number of running jobs is plotted based on the maximum number of running jobs for the specified interval of time. In this example there are two online agents, but in some areas there are four running jobs due to the way the pool data is aggregated.



## FAQ

### Where does the pool consumption report get the data it displays?

The pool consumption report uses the [Azure DevOps Analytics service](#) and the [TaskAgentRequestSnapshots](#)

endpoint. You can query this endpoint using the following URL prefix:

```
https://analytics.dev.azure.com/{organization}/{project_id}/_odata/v4.0-preview/TaskAgentRequestSnapshots .
```

You can retrieve the `project_id` for your project by navigating to the following URL:

```
https://dev.azure.com/{organization}/_apis/projects?api-version=5.0-preview.3 .
```

The following example shows a sample query and response.

```
{
 "@odata.context": "https://analytics.dev.azure.com/{org}/{project_id}/_odata/v4.0-
 preview/$metadata#TaskAgentRequestSnapshots",
 "vsts.warnings@odata.type": "#Collection(String)",
 "@vsts.warnings": [
 "VS403507: The specified query does not include a $select or $apply clause which is recommended for all
 queries. Details on recommended query patterns are available here: https://go.microsoft.com/fwlink/?
 linkid=861060."
],
 "value": [
 {
 "SamplingDateSK": 20201117,
 "SamplingHour": 13,
 "SamplingTime": "2020-11-17T13:10:00-08:00",
 "QueuedDate": "2020-11-17T13:07:26.22-08:00",
 "QueuedDateSK": 20201117,
 "StartedDate": "2020-11-17T15:02:23.7398429-08:00",
 "StartedDateSK": 20201117,
 "FinishedDate": "2020-11-17T15:13:49.89-08:00",
 "FinishedDateSK": 20201117,
 "QueueDurationSeconds": 6897.519,
 "ProjectSK": "...",
 "PipelineSK": 5141,
 "RequestId": 6313,
 "PoolId": 28,
 "PipelineType": "Build",
 "IsHosted": true,
 "IsRunning": false,
 "IsQueued": true
 },
 ...
]
}
```

For more information on query options, see [Query guidelines for Analytics with OData](#).

#### NOTE

The `TaskAgentRequestSnapshots` endpoint is in preview and not yet documented but you can view information about the data returned by navigating to the endpoint URL:

```
https://analytics.dev.azure.com/{org}/{project_id}/_odata/v4.0-preview/TaskAgentRequestSnapshots .
```

### Why are there more running jobs than there are agents or concurrency?

Pool data is aggregated at a granularity of 10 minutes, and the number of running jobs is plotted based on the maximum number of running jobs for the specified interval of time. Each running job is counted separately, and if multiple jobs complete during the 10 minute interval they contribute to the total count of running jobs for that interval.

### What is the difference between viewing the graphs in Project settings vs Organization settings?

The scope of the data in the graph is determined based on whether the chart is accessed through **Project settings** or **Organization settings**. At the organization level, the chart is plotted using data from pipelines across any project within the organization that have run jobs in that pool. At the project level, the chart is plotted using data from pipelines in that particular project that have run jobs in that pool.

# Microsoft-hosted agents

4/20/2021 • 18 minutes to read • [Edit Online](#)

## Azure Pipelines

Microsoft-hosted agents are only available with Azure DevOps Services, which is hosted in the cloud. You cannot use Microsoft-hosted agents or the Azure Pipelines agent pool with on-premises TFS or Azure DevOps Server. With these on-premises versions, you must use [self-hosted agents](#).

### IMPORTANT

To view the content available for your platform, make sure that you select the correct version of this article from the version selector which is located above the table of contents. Feature support differs depending on whether you are working from Azure DevOps Services or an on-premises version of Azure DevOps Server, renamed from Team Foundation Server (TFS).

To learn which on-premises version you are using, see [What platform/version am I using?](#)

If your pipelines are in Azure Pipelines, then you've got a convenient option to run your jobs 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 for each job in the pipeline. The virtual machine is discarded after one job. Microsoft-hosted agents can run jobs [directly on the VM](#) or [in a container](#).

Azure Pipelines provides a pre-defined agent pool named **Azure Pipelines** with Microsoft-hosted agents.

For many teams this is the simplest way to run your jobs. 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.

## Software

The **Azure Pipelines** agent pool offers several virtual machine images to choose from, each including a broad range of tools and software.

IMAGE	CLASSIC EDITOR AGENT SPECIFICATION	YAML VM IMAGE LABEL	INCLUDED SOFTWARE
Windows Server 2019 with Visual Studio 2019	<i>windows-2019</i>	<code>windows-latest</code> OR <code>windows-2019</code>	<a href="#">Link</a>
Windows Server 2016 with Visual Studio 2017	<i>vs2017-win2016</i>	<code>vs2017-win2016</code>	<a href="#">Link</a>
Ubuntu 20.04	<i>ubuntu-20.04</i>	<code>ubuntu-latest</code> OR <code>ubuntu-20.04</code>	<a href="#">Link</a>

IMAGE	CLASSIC EDITOR AGENT SPECIFICATION	YAML VM IMAGE LABEL	INCLUDED SOFTWARE
Ubuntu 18.04	<i>ubuntu-18.04</i>	<code>ubuntu-18.04</code>	<a href="#">Link</a>
Ubuntu 16.04	<i>ubuntu-16.04</i>	<code>ubuntu-16.04</code>	<a href="#">Link</a>
macOS X Mojave 10.14	<i>macOS-10.14</i>	<code>macOS-10.14</code>	<a href="#">Link</a>
macOS X Catalina 10.15	<i>macOS-10.15</i>	<code>macOS-latest</code> OR <code>macOS-10.15</code>	<a href="#">Link</a>

You can see the installed software for each hosted agent by choosing the **Included Software** link in the table. When using macOS images, you can manually select from tool versions. [See below.](#)

#### NOTE

In March 2020, we removed the following Azure Pipelines hosted images:

- Windows Server 2012R2 with Visual Studio 2015 ( `vs2015-win2012r2` )
- macOS X High Sierra 10.13 ( `macOS-10.13` )
- Windows Server Core 1803 - ( `win1803` )

Customers are encouraged to migrate to `vs2017-win2016`, `macOS-10.14`, or a [self-hosted agent](#) respectively.

For more information and instructions on how to update your pipelines that use those images, see [Removing older images in Azure Pipelines hosted pools](#).

#### NOTE

The Azure Pipelines hosted pool replaces the previous hosted pools that had names that mapped to the corresponding images. Any jobs you had in the previous hosted pools are automatically redirected to the correct image in the new Azure Pipelines hosted pool. In some circumstances, you may still see the old pool names, but behind the scenes the hosted jobs are run using the Azure Pipelines pool. For more information about this update, see the [Single hosted pool](#) release notes from the [July 1 2019 - Sprint 154 release notes](#).

#### IMPORTANT

To request additional software to be installed on Microsoft-hosted agents, don't create a feedback request on this document or open a support ticket. Instead, open an issue on our [repository](#), where we manage the scripts to generate various images.

## Use a Microsoft-hosted agent

- [YAML](#)
- [Classic](#)

In YAML pipelines, if you do not specify a pool, pipelines will default to the Azure Pipelines agent pool. You simply need to specify which virtual machine image you want to use.

```
jobs:
- job: Linux
 pool:
 vmImage: 'ubuntu-latest'
 steps:
 - script: echo hello from Linux
- job: macOS
 pool:
 vmImage: 'macOS-latest'
 steps:
 - script: echo hello from macOS
- job: Windows
 pool:
 vmImage: 'windows-latest'
 steps:
 - script: echo hello from Windows
```

#### NOTE

The specification of a pool can be done at multiple levels in a YAML file. If you notice that your pipeline is not running on the expected image, make sure that you verify the pool specification at the pipeline, stage, and job levels.

### 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 hard-code the drive letter or folder that contains the repository. The precise layout of the hosted agents is subject to change without warning.

## Hardware

Microsoft-hosted agents that run Windows and Linux images are provisioned on Azure general purpose virtual machines [Standard\\_DS2\\_v2](#). These virtual machines are colocated in the same geography as your Azure DevOps organization.

Agents that run macOS images are provisioned on Mac pros. These agents always run in the US irrespective of the location of your Azure DevOps organization. If data sovereignty is important to you and if your organization is not in the US, then you should not use macOS images. [Learn more](#).

All of these machines have 10 GB of free disk space available for your pipelines to run. This free space is consumed when your pipeline checks out source code, downloads packages, pulls docker images, or generates intermediate files.

#### IMPORTANT

We cannot honor requests to increase disk space on Microsoft-hosted agents, or to provision more powerful machines. If the specifications of Microsoft-hosted agents do not meet your needs, then you should consider [self-hosted agents](#) or [scale set agents](#).

## Networking

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. Because Azure DevOps uses the Azure global network, IP ranges vary over time. We publish a [weekly JSON file](#) listing IP ranges for Azure datacenters, broken out by region. This file is updated weekly with new planned IP ranges. The new IP ranges become effective the following week. We recommend that you check back frequently

(at least once every week) 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. The IP address ranges for the hosted agents are listed in the weekly file under `AzureCloud.<region>`, such as `AzureCloud.westus` for the West US region.

Your hosted agents run in the same [Azure geography](#) as your organization. Each geography contains one or more regions. While your agent may run in the same region as your organization, it is not guaranteed to do so. To obtain the complete list of possible IP ranges for your agent, you must use the IP ranges from all of the regions that are contained in your geography. For example, if your organization is located in the **United States** geography, you must use the IP ranges for all of the regions in that geography.

To determine your geography, navigate to

[https://dev.azure.com/<your\\_organization>/\\_settings/organizationOverview](https://dev.azure.com/<your_organization>/_settings/organizationOverview), get your region, and find the associated geography from the [Azure geography](#) table. Once you have identified your geography, use the IP ranges from the [weekly file](#) for all regions in that geography.

#### IMPORTANT

You cannot use private connections such as [ExpressRoute](#) or VPN to connect Microsoft-hosted agents to your corporate network. The traffic between Microsoft-hosted agents and your servers will be over public network.

### To identify the possible IP ranges for Microsoft-hosted agents

1. Identify the [region for your organization](#) in [Organization settings](#).
2. Identify the [Azure Geography](#) for your organization's region.
3. Map the names of the regions in your geography to the format used in the weekly file, following the format of `AzureCloud.<region>`, such as `AzureCloud.westus`. You can map the names of the regions from the [Azure Geography](#) list to the format used in the weekly file by reviewing the region names passed to the constructor of the regions defined in the [source code for the Region class](#), from the [Azure Management Libraries for .NET](#).

#### NOTE

Since there is no API in the [Azure Management Libraries for .NET](#) to list the regions for a geography, you must list them manually as shown in the following example.

4. Retrieve the IP addresses for all regions in your geography from the [weekly file](#). If your region is **Brazil South** or **West Europe**, you must include additional IP ranges based on your fallback geography, as described in the following note.

#### NOTE

Due to capacity restrictions, some organizations in the **Brazil South** or **West Europe** regions may occasionally see their hosted agents located outside their expected geography. In these cases, in addition to including the IP ranges as described in the previous section, additional IP ranges must be included for the regions in the capacity fallback geography.

If your organization is in the **Brazil South** region, your capacity fallback geography is **United States**.

If your organization is in the **West Europe** region, the capacity fallback geography is **France**.

Our Mac IP ranges are not included in the Azure IPs above, though we are investigating options to publish these in the future.

#### Example

In the following example, the hosted agent IP address ranges for an organization in the West US region are

retrieved from the weekly file. Since the West US region is in the United States geography, the IP addresses for all regions in the United States geography are included. In this example, the IP addresses are written to the console.

```
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;

namespace WeeklyFileIPRanges
{
 class Program
 {
 // Path to the locally saved weekly file
 const string weeklyFilePath = @"C:\MyPath\ServiceTags_Public_20200504.json";

 static void Main(string[] args)
 {
 // United States geography has the following regions:
 // Central US, East US 2, East US, North Central US,
 // South Central US, West Central US, West US, West US 2
 List<string> USGeographyRegions = new List<string>
 {
 "centralus",
 "eastus",
 "eastus2",
 "northcentralus",
 "southcentralus",
 "westcentralus",
 "westus",
 "westus2"
 };

 // Load the weekly file
 JObject weeklyFile = JObject.Parse(File.ReadAllText(weeklyFilePath));
 JArray values = (JArray)weeklyFile["values"];

 foreach (string region in USGeographyRegions)
 {
 string azureCloudRegion = $"AzureCloud.{region}";
 Console.WriteLine(azureCloudRegion);

 var ipList =
 from v in values
 where (string)v["name"] == azureCloudRegion
 select v["properties"]["addressPrefixes"];

 foreach (var ip in ipList.Children())
 {
 Console.WriteLine(ip);
 }
 }
 }
 }
}
```

## Service tags

Microsoft-hosted agents can't be listed by service tags. If you're trying to grant hosted agents access to your resources, you'll need to follow the IP range allow listing method.

## Security

Microsoft-hosted agents run on secure Azure platform. However, you must be aware of the following security

considerations.

- Although Microsoft-hosted agents run on Azure public network, they are not assigned public IP addresses. So, external entities cannot target Microsoft-hosted agents.
- Microsoft-hosted agents are run in individual VMs, which are re-imaged after each run. Each agent is dedicated to a single organization, and each VM hosts only a single agent.
- There are several benefits to running your pipeline on Microsoft-hosted agents, from a security perspective. If you run untrusted code in your pipeline, such as contributions from forks, it is safer to run the pipeline on Microsoft-hosted agents than on self-hosted agents that reside in your corporate network.
- When a pipeline needs to access your corporate resources behind a firewall, you have to allow the IP address range for the Azure geography. This may increase your exposure as the range of IP addresses is rather large and since machines in this range can belong to other customers as well. The best way to prevent this is to avoid the need to access internal resources.
- Hosted images do not conform to [CIS hardening benchmarks](#). To use CIS-hardened images, you must create either self-hosted agents or scale-set agents.

## 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 10 GB of storage for your source and build outputs.
- Provide a free tier:
  - 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. [Contact us](#) to get your free tier limits increased.
  - Private project: One free parallel job that can run for up to 60 minutes each time, until you've used 1,800 minutes (30 hours) per month. 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](#).
- Run on Microsoft Azure general purpose virtual machines [Standard\\_DS2\\_v2](#)
- Run as an administrator on Windows and a passwordless sudo user on Linux
- (Linux only) Run steps in a [cgroup](#) that offers 6 GB of physical memory and 13 GB of total memory

Microsoft-hosted agents do not offer:

- The ability to remotely connect.
- The ability to [drop artifacts to a UNC file share](#).
- The ability to join machines directly to your corporate network.
- The ability to get bigger or more powerful build machines.
- The ability to pre-load custom software. You can install software during a pipeline run, such as through [tool installer tasks](#) or in a script.
- Potential performance advantages that you might get by using self-hosted agents that might start and run builds faster. [Learn more](#)
- The ability to run [XAML builds](#).

If Microsoft-hosted agents don't meet your needs, then you can deploy your own [self-hosted agents](#) or use [scale set agents](#).

## FAQ

### How can I see what software is included in an image?

You can see the installed software for each hosted agent by choosing the [Included Software](#) link in the [Use a](#)

[Microsoft-hosted agent](#) table.

## How does Microsoft choose the software and versions to put on the image?

More information about the versions of software included on the images can be found at [Guidelines for what's installed](#).

## When are the images updated?

Images are typically updated weekly. You can check the [status badges](#) which are in the format `20200113.x` where the first part indicates the date the image was updated.

## What can I do if software I need is removed or replaced with a newer version?

You can let us know by filing a GitHub issue by choosing the [Included Software](#) links in the [Use a Microsoft-hosted agent](#) table.

You can also use a self-hosted agent that includes the exact versions of software that you need. For more information, see [Self-hosted agents](#).

## What if I need a bigger machine with more processing power, memory, or disk space?

We can't increase the memory, processing power, or disk space for Microsoft-hosted agents, but you can use [self-hosted agents](#) or [scale set agents](#) hosted on machines with your desired specifications.

## I can't select a Microsoft-hosted agent and I can't queue my build or deployment. What should I do?

Microsoft-hosted agents are only available in Azure Pipelines and not in TFS or Azure DevOps Server.

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](#).

## My pipelines running on Microsoft-hosted agents take more time to complete. How can I speed them up?

If your pipeline has recently become slower, review our [status page](#) for any outages. We could be having issues with our service. Or else, review any changes that you made in your application code or pipeline. Your repository size during check-out might have increased, you may be uploading larger artifacts, or you may be running more tests.

If you are just setting up a pipeline and are comparing the performance of Microsoft-hosted agents to your local machine or a self-hosted agent, then note the [specifications](#) of the hardware that we use to run your jobs. We are unable to provide you with bigger or powerful machines. You can consider using [self-hosted agents](#) or [scale set agents](#) if this performance is not acceptable.

## I need more agents. What can I do?

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 additional minutes or parallel jobs for your open-source project, contact [support](#). If you need additional minutes or parallel jobs for your private project, then you can [buy more](#).

## My pipeline succeeds on self-hosted agent, but fails on Microsoft-hosted agents. What should I do?

Your self-hosted agent probably has all the right dependencies installed on it, whereas the same dependencies, tools, and software are not installed on Microsoft-hosted agents. First, carefully review the list of software that is installed on Microsoft-hosted agents by following the link to [Included software](#) in the table above. Then, compare that with the software installed on your self-hosted agent. In some cases, Microsoft-hosted agents may have the tools that you need (for example, Visual Studio), but all of the necessary optional components may not have been installed. If you find differences, then you have two options:

- You can create a new issue on the [repository](#), where we track requests for additional software. Contacting support will not help you with setting up new software on Microsoft-hosted agents.

- You can use [self-hosted agents](#) or [scale set agents](#). With these agents, you are fully in control of the images that are used to run your pipelines.

### **My build succeeds on my local machine, but fails on Microsoft-hosted agents. What should I do?**

Your local machine probably has all the right dependencies installed on it, whereas the same dependencies, tools, and software are not installed on Microsoft-hosted agents. First, carefully review the list of software that is installed on Microsoft-hosted agents by following the link to **Included software** in the table above. Then, compare that with the software installed on your local machine. In some cases, Microsoft-hosted agents may have the tools that you need (e.g., Visual Studio), but all of the necessary optional components may not have been installed. If you find differences, then you have two options:

- You can create a new issue on the [repository](#), where we track requests for additional software. This is your best bet for getting new software installed. Contacting support will not help you with setting up new software on Microsoft-hosted agents.
- You can use [self-hosted agents](#) or [scale set agents](#). With these agents, you are fully in control of the images that are used to run your pipelines.

### **My pipeline fails with the error: "no space left on device".**

Microsoft-hosted agents only have 10 GB of disk space available for running your job. This space is consumed when you check out source code, when you download packages, when you download docker images, or when you produce intermediate files. Unfortunately, we cannot increase the free space available on Microsoft-hosted images. You can restructure your pipeline so that it can fit into this space. Or, you can consider using [self-hosted agents](#) or [scale set agents](#).

### **My pipeline running on Microsoft-hosted agents requires access to servers on our corporate network? How do we get a list of IP addresses to allow in our firewall?**

See the section [Agent IP ranges](#)

### **Our pipeline running on Microsoft-hosted agents is unable to resolve the name of a server on our corporate network. How can we fix this?**

If you refer to the server by its DNS name, then make sure that your server is publicly accessible on the Internet through its DNS name. If you refer to your server by its IP address, make sure that the IP address is publicly accessible on the Internet. In both cases, ensure that any firewall in between the agents and your corporate network has the [agent IP ranges](#) allowed.

### **I'm getting an SAS IP authorization error from an Azure Storage account**

If you get an SAS error code, it is most likely because the IP address ranges from the Microsoft-hosted agents aren't permitted due to your Azure Storage rules. There are a few workarounds:

1. [Manage the IP network rules for your Azure Storage account](#) and add the [IP address ranges for your hosted agents](#).
2. In your pipeline, use [Azure CLI to update the network ruleset for your Azure Storage account](#) right before you access storage, and then restore the previous ruleset.
3. Use [self-hosted agents](#) or [Scale set agents](#).

### **How can I manually select versions of tools on the Hosted macOS agent?**

#### **Xamarin**

Hosted macOS agent stores Xamarin SDK versions and the associated Mono versions as a set of symlinks to Xamarin SDK locations that are available by a single bundle symlink.

To manually select a Xamarin SDK version to use on the **Hosted macOS** agent, execute the following bash command before your Xamarin build task as a part of your build, specifying the symlink to Xamarin versions bundle that you need.

```
/bin/bash -c "sudo $AGENT_HOMEDIRECTORY/scripts/select-xamarin-sdk.sh <symlink>"
```

The list of all available Xamarin SDK versions and symlinks can be found in the agents documentation:

- [macOS 10.14](#)
- [macOS 10.15](#)

This command does not select the Mono version beyond the Xamarin SDK. To manually select a Mono version, see instructions below.

In case you are using a non-default version of Xcode for building your Xamarin.iOS or Xamarin.Mac apps, you should additionally execute this command line:

```
/bin/bash -c "echo '##vso[task.setvariable variable=MD_APPLE_SDK_ROOT;]$(xcodeRoot);sudo xcode-select --switch $(xcodeRoot)/Contents/Developer'"
```

where `$(xcodeRoot)` = `/Applications/Xcode_12.4.app`

Xcode versions on the **Hosted macOS** agent pool can be found [here](#).

#### 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 12.4 as needed:

```
/bin/bash -c "sudo xcode-select -s /Applications/Xcode_12.4.app/Contents/Developer"
```

Xcode versions on the **Hosted macOS** agent pool can be found [here](#).

This command does not work for Xamarin apps. To manually select an Xcode version for building Xamarin apps, see instructions above.

#### Mono

To manually select a Mono version to use on the **Hosted macOS** agent pool, execute this script in each job of your build before your Mono build task, specifying the symlink with the required Mono version (list of all available symlinks can be found in the [Xamarin section](#) above):

```
SYMLINK=<symlink>
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"
```

## Videos

# Self-hosted Linux agents

5/7/2021 • 22 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

To run your jobs, 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 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 [Azure Pipelines agents](#).

## Check prerequisites

The agent is based on .NET Core 3.1. You can run this agent on several Linux distributions. We support the following subset of .NET Core supported distributions:

- x64
  - CentOS 7, 6 (see note 1)
  - Debian 9
  - Fedora 30, 29
  - Linux Mint 18, 17
  - openSUSE 42.3 or later
  - Oracle Linux 7
  - Red Hat Enterprise Linux 7, 6 (see note 1)
  - SUSE Enterprise Linux 12 SP2 or later
  - Ubuntu 18.04, 16.04
- ARM32 (see note 2)
  - Debian 9
  - Ubuntu 18.04

## NOTE

Note 1: RHEL 6 and CentOS 6 require installing the specialized `rhel.6-x64` version of the agent.

#### NOTE

Note 2: ARM instruction set [ARMv7](#) or above is required. Run `uname -a` to see your Linux distro's instruction set.

Regardless of your platform, you will need to install Git 2.9.0 or higher. We strongly recommend installing the latest version of Git.

If you'll be using TFVC, you will also need the [Oracle Java JDK 1.6](#) or higher. (The Oracle JRE and OpenJDK are not sufficient for this purpose.)

The agent installer knows how to check for other dependencies. You can install those dependencies on supported Linux platforms by running `./bin/installdependencies.sh` in the agent directory.

**TFS 2018 RTM and older:** The shipped agent is based on CoreCLR 1.0. We recommend that, if able, you should upgrade to a later agent version (2.125.0 or higher). See [Azure Pipelines agent prereqs](#) for more about what's required to run a newer agent.

If you must stay on the older agent, 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.

You should run agent setup manually the first time. After you get a feel for how agents work, or if you want to automate setting up many agents, consider using [unattended config](#).

## 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 your Team Foundation Server web portal (

`https://{{your-server}}:8080/tfs/` ).

1. Sign in with the user account you plan to use in your Azure DevOps Server web portal (

`https://{{your-server}}/DefaultCollection/` ).

1. Sign in with the user account you plan to use in your Azure DevOps organization (

`https://dev.azure.com/{{your_organization}}` ).

2. From your home page, open your profile. Go to your security details.



3. [Create a personal access token](#).



2. From your home page, open your user settings, and then select **Personal access tokens**.



3. Create a personal access 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.

Select **Show all scopes** at the bottom of the **Create a new personal access token window** window to see the complete list of scopes.

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

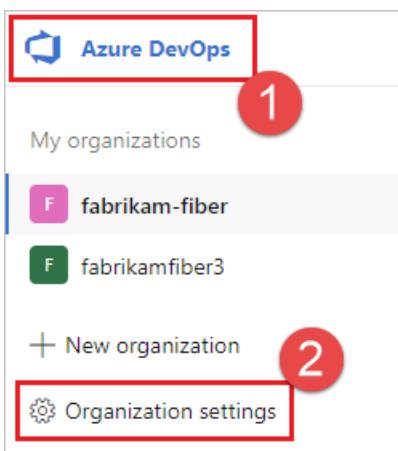
Make sure the user account that you're going to use has permission to register the agent.

Is the user an Azure DevOps organization owner or TFS or Azure DevOps 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 Azure DevOps Server or TFS server:

1. Choose **Azure DevOps, Organization settings**.



2. Choose **Agent pools**.

Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards  
Process

Pipelines

Agent pools **1**

Settings  
Deployment pools  
Parallel jobs

**Agent pools**

Name
Azure Pipelines
Default
Test
Test2

1. Choose Azure DevOps, Collection settings.

Azure DevOps **1**

Collections  
fabrikam-fiber **2**  
fabrikamfiber3  
+ New organization

Collection Settings

2. Choose Agent pools.

Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards  
Process

Pipelines

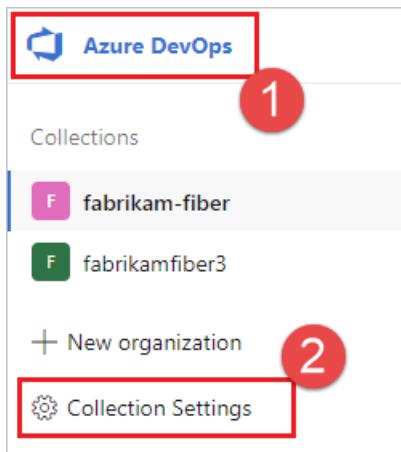
Agent pools **1**

Settings  
Deployment pools  
Parallel jobs

**Agent pools**

Name
Azure Pipelines
Default
Test
Test2

1. Choose Azure DevOps, Collection settings.



2. Choose Agent pools.

A screenshot of the 'Agent pools' section in the 'Organization Settings' of the Azure DevOps portal. The left sidebar shows the following navigation:

- Security
- Policies
- Permissions
- Boards
- Process
- Pipelines
- Agent pools (highlighted with a red box)
- Settings
- Deployment pools
- Parallel jobs

The main area is titled 'Agent pools' and lists four entries:

Name
Azure Pipelines
Default
Test
Test2

1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.

A screenshot of the 'Agent Queues' section in the 'Settings' menu of a project. The menu items are:

- Overview (highlighted with a red box)
- Work
- Security
- Alerts
- Version Control
- Agent Queues (highlighted with a red box)
- Service Hooks

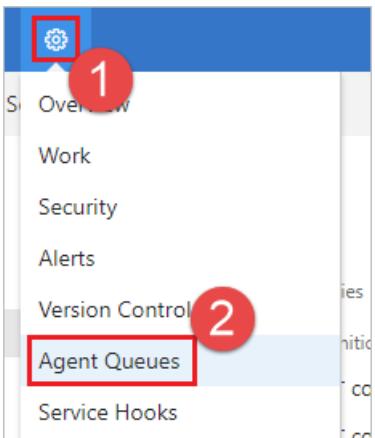
Red numbers 1 and 2 are overlaid on the screen: 1 is at the top right of the 'Overview' item, and 2 is at the bottom right of the 'Agent Queues' item.

2. Choose **Manage pools**.

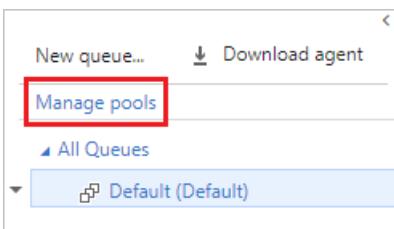
A screenshot of the 'Manage pools' section in the 'Agent Queues' menu. The menu items are:

- New queue...
- Download agent
- Manage pools (highlighted with a red box)
- All Queues
- Default (Default)

1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



2. Choose **Manage pools**.



1. Navigate to your project and choose **Manage project** (gear icon).



2. Choose **Control panel**.



3. Select **Agent pools**.



2. Select the pool on the right side of the page and then click **Security**.

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 or Azure DevOps 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 or Azure DevOps 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**.

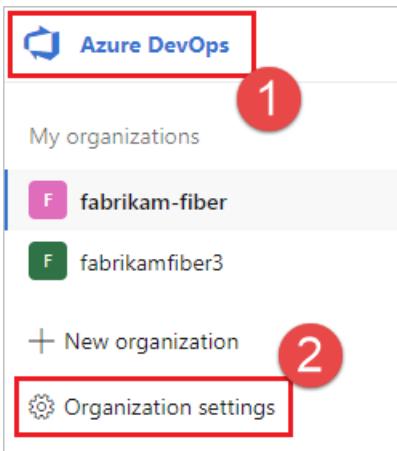
#### NOTE

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 or Azure DevOps 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:
  1. Choose **Azure DevOps, Organization settings**.

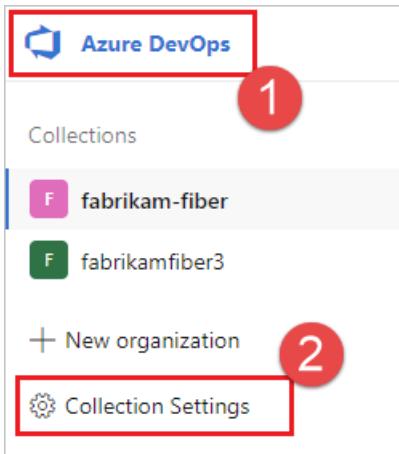


2. Choose **Agent pools**.

The screenshot shows the 'Agent pools' page within the 'Organization Settings' of 'FabrikamFiber'. The left sidebar has a red box around the 'Agent pools' button. The main area displays a table of agent pools with columns for Name and Status. The table contains four rows: 'Azure Pipelines' (Status: 'Active'), 'Default' (Status: 'Active'), 'Test' (Status: 'Active'), and 'Test2' (Status: 'Active').

Name	Status
Azure Pipelines	Active
Default	Active
Test	Active
Test2	Active

1. Choose **Azure DevOps, Collection settings**.



2. Choose Agent pools.

Azure DevOps

FabrikamFiber / Organization Settings / Agent pools

Security

Policies

Permissions

Boards

Process

Pipelines

Agent pools

Settings

Deployment pools

Parallel jobs

Agent pools

Name	Type
Azure Pipelines	Azure Pipelines
Default	Hosted
Test	Hosted
Test2	Hosted

1. Choose Azure DevOps, Collection settings.

Azure DevOps

Collections

fabrikam-fiber

fabrikamfiber3

+ New organization

Collection Settings

2. Choose Agent pools.

Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards  
Process

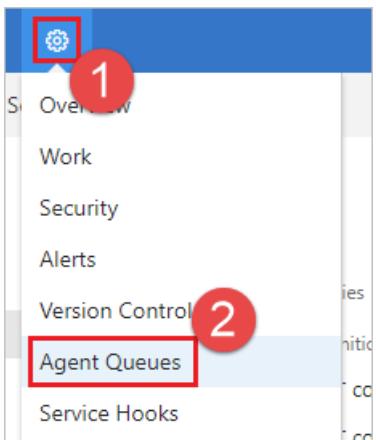
Pipelines

Agent pools **(Red Box)**  
Settings  
Deployment pools  
Parallel jobs

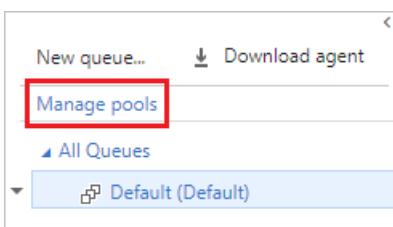
Agent pools

Name
Azure Pipelines
Default
Test
Test2

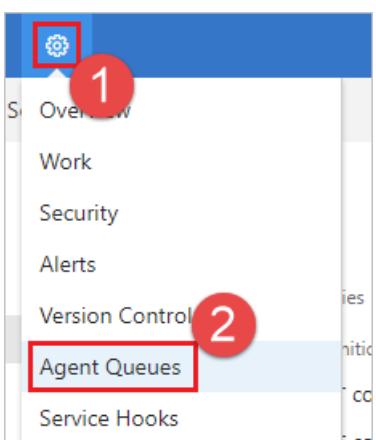
1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



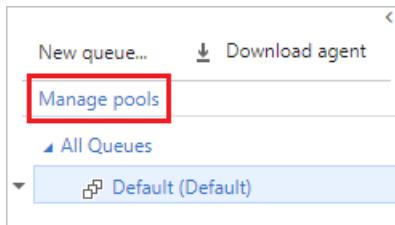
2. Choose **Manage pools**.



1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



2. Choose **Manage pools**.



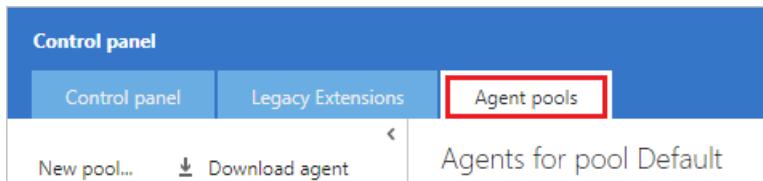
1. Navigate to your project and choose **Manage project** (gear icon).



2. Choose **Control panel**.



3. Select **Agent pools**.



3. Select the **Default** pool, select the **Agents** tab, and choose **New agent**.

4. On the **Get the agent** dialog box, click **Linux**.

5. On the left pane, select the specific flavor. We offer x64 or ARM for most Linux distributions. We also offer a specific build for Red Hat Enterprise Linux 6.

6. On the right pane, click the **Download** button.

7. Follow the instructions on the page.

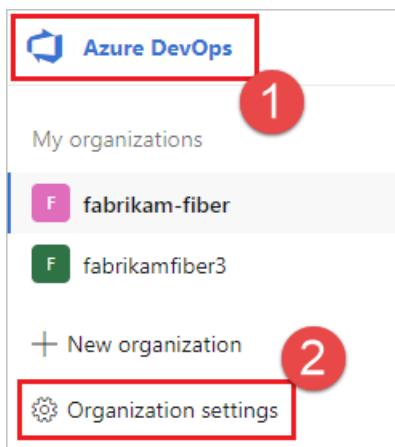
8. Unpack the agent into the directory of your choice. `cd` to that directory and run `./config.sh`.

## Azure DevOps Server 2019 and Azure DevOps Server 2020

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 DevOps Server 2019, and navigate to the **Agent pools** tab:

1. Choose **Azure DevOps, Organization settings**.



2. Choose Agent pools.

The screenshot shows the 'Agent pools' section of the Azure DevOps Organization Settings. On the left, a sidebar lists various settings: Security, Policies, Permissions, Boards, Process, Pipelines, Agent pools (which is highlighted with a red box), Settings, Deployment pools, and Parallel jobs. The main area displays a table with columns for Name and Description. It contains four entries: 'Azure Pipelines' (Description: Azure Pipelines), 'Default' (Description: Default agent pool), 'Test' (Description: Test agent pool), and 'Test2' (Description: Test2 agent pool).

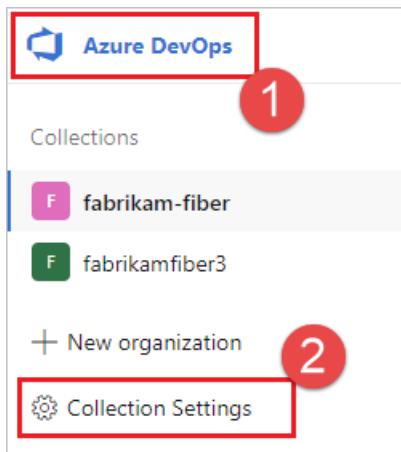
1. Choose Azure DevOps, Collection settings.

The screenshot shows the 'Collections' page of Azure DevOps. A red box highlights the 'Azure DevOps' header. Below it, there's a list of collections: 'fabrikam-fiber' (highlighted with a red box) and 'fabrikamfiber3'. At the bottom, there are buttons for '+ New organization' and 'Collection Settings'. A red circle with the number '1' is on the 'Azure DevOps' header, and a red circle with the number '2' is on the 'Collection Settings' button.

2. Choose Agent pools.

This screenshot is identical to the one in step 2 of the previous section, showing the 'Agent pools' section of the Azure DevOps Organization Settings. The sidebar on the left has 'Agent pools' highlighted with a red box. The main area shows a table with four agent pool entries: 'Azure Pipelines', 'Default', 'Test', and 'Test2'.

1. Choose Azure DevOps, Collection settings.



2. Choose Agent pools.

Azure DevOps / FabrikamFiber / Organization Settings / Agent pools

Agent pools

Name
Azure Pipelines
Default
Test
Test2

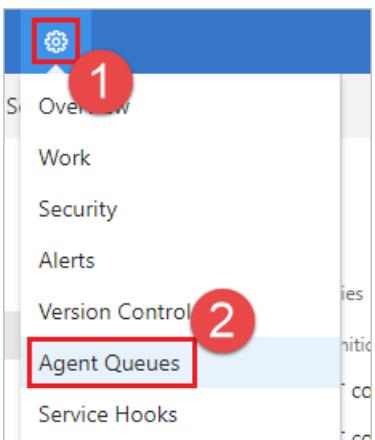
1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.

Settings Overview Work Security Alerts Version Control Agent Queues Service Hooks

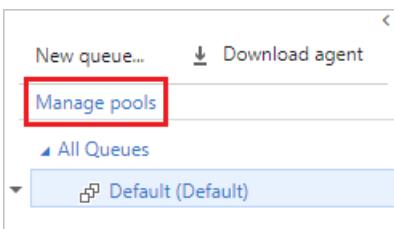
2. Choose **Manage pools**.

New queue... Download agent  
Manage pools  
All Queues  
Default (Default)

1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



2. Choose **Manage pools**.



1. Navigate to your project and choose **Manage project** (gear icon).



2. Choose **Control panel**.



3. Select **Agent pools**.



3. Click **Download agent**.

4. On the **Get agent** dialog box, click **Linux**.

5. On the left pane, select the specific flavor. We offer x64 or ARM for most Linux distributions. We also offer a specific build for Red Hat Enterprise Linux 6.

6. On the right pane, click the **Download** button.

7. Follow the instructions on the page.

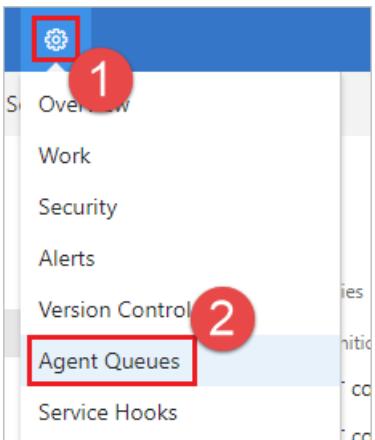
8. Unpack the agent into the directory of your choice. `cd` to that directory and run `./config.sh`.

## TFS 2017 and TFS 2018

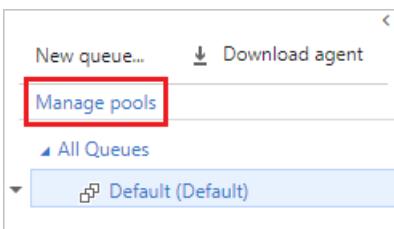
1. Log on to the machine using the account for which you've prepared permissions as explained above.

2. In your web browser, sign in to TFS, and navigate to the **Agent pools** tab:

- a. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



- b. Choose **Manage pools**.



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}`

Azure DevOps Server 2019: `https://{your_server}/DefaultCollection`

TFS 2017 and newer: `https://{your_server}/tfs`

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 or Azure DevOps Server

#### 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 or Azure DevOps Server 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 or Azure DevOps Server 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 Azure DevOps Server or TFS instance and the agent machine are not in a trusted domain. PAT authentication is handled by your Azure DevOps Server or 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 Azure DevOps Server and the 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 restart the agent, press **Ctrl+C** and then run `run.sh` to restart it.

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 once

For agents configured to run interactively, you can choose to have the agent accept only one job. To run in this configuration:

```
./run.sh --once
```

Agents in this mode will accept only one job and then spin down gracefully (useful for running in [Docker](#) on a service like Azure Container Instances).

## 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 an example `./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. We encourage you to review, and if needed, update the script before running it.

Some important caveats:

- If you run your agent as a service, you cannot run the agent service as `root` user.
- Users running [SELinux](#) have reported difficulties with the provided `svc.sh` script. Refer to [this agent issue](#) as a starting point. SELinux is not an officially supported configuration.

### 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 [username]
```

This command creates a service file that points to `./runsvc.sh`. This script sets up the environment (more details below) and starts the agents host. If `[username]` parameter is not specified then the username is taken from the `$SUDO_USER` environment variable which is set by sudo command. This variable is always equal to the name of the user who invoked the `sudo` command.

### 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):

```
./env.sh
sudo ./svc.sh stop
sudo ./svc.sh start
```

The snapshot of the environment variables is stored in `.env` file (`PATH` is stored in `.path`) under agent root directory, you can also change these files 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 or Azure DevOps Server:** the name of your on-premises 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.

## Use a cgroup to avoid agent failure

It's important to avoid situations in which the agent fails or become unusable because otherwise the agent can't stream pipeline logs or report pipeline status back to the server. You can mitigate the risk of this kind of problem being caused by high memory pressure by using cgroups and a lower `oom_score_adj`. After you've done this, Linux reclaims system memory from pipeline job processes before reclaiming memory from the agent process. [Learn how to configure cgroups and OOM score](#).

## 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.

To configure an agent, it must know the URL to your organization or collection and credentials of someone authorized to set up agents. All other responses are optional. Any command-line parameter can be specified using an environment variable instead: put its name in upper case and prepend `VSTS_AGENT_INPUT_`. For example, `VSTS_AGENT_INPUT_PASSWORD` instead of specifying `--password`.

### Required options

- `--unattended` - agent setup will not prompt for information, and all settings must be provided on the command line
- `--url <url>` - URL of the server. For example: <https://dev.azure.com/myorganization> or `http://my-azure-devops-server:8080/tfs`
- `--auth <type>` - authentication type. Valid values are:
  - `pat` (Personal access token)
  - `negotiate` (Kerberos or NTLM)
  - `alt` (Basic authentication)
  - `integrated` (Windows default credentials)

## Authentication options

- If you chose `--auth pat`:
  - `--token <token>` - specifies your personal access token
- If you chose `--auth negotiate` or `--auth alt`:
  - `--userName <userName>` - specifies a Windows username in the format `domain\userName` or `userName@domain.com`
  - `--password <password>` - specifies a password

## Pool and agent names

- `--pool <pool>` - pool name for the agent to join
- `--agent <agent>` - agent name
- `--replace` - replace the agent in a pool. If another agent is listening by the same name, it will start failing with a conflict

## Agent setup

- `--work <workDirectory>` - work directory where job data is stored. Defaults to `_work` under the root of the agent directory. The work directory is owned by a given agent and should not share between multiple agents.
- `--acceptTeeEula` - accept the Team Explorer Everywhere End User License Agreement (macOS and Linux only)
- `--disableloguploads` - don't stream or send console log output to the server. Instead, you may retrieve them from the agent host's filesystem after the job completes.

## Windows-only startup

- `--runAsService` - configure the agent to run as a Windows service (requires administrator permission)
- `--runAsAutoLogon` - configure auto-logon and run the agent on startup (requires administrator permission)
- `--windowsLogonAccount <account>` - used with `--runAsService` or `--runAsAutoLogon` to specify the Windows user name in the format `domain\userName` or `userName@domain.com`
- `--windowsLogonPassword <password>` - used with `--runAsService` or `--runAsAutoLogon` to specify Windows logon password
- `--overwriteAutoLogon` - used with `--runAsAutoLogon` to overwrite the existing auto logon on the machine
- `--noRestart` - used with `--runAsAutoLogon` to stop the host from restarting after agent configuration completes

## Deployment group only

- `--deploymentGroup` - configure the agent as a deployment group agent
- `--deploymentGroupName <name>` - used with `--deploymentGroup` to specify the deployment group for the agent to join
- `-- projectName <name>` - used with `--deploymentGroup` to set the project name
- `--addDeploymentGroupTags` - used with `--deploymentGroup` to indicate that deployment group tags should be added
- `--deploymentGroupTags <tags>` - used with `--addDeploymentGroupTags` to specify the comma separated list of tags for the deployment group agent - for example "web, db"

## Environments only

- `--addvirtualmachineresourcetags` - used to indicate that environment resource tags should be added
- `--virtualmachineresourcetags <tags>` - used with `--addvirtualmachineresourcetags` to specify the comma separated list of tags for the environment resource agent - for example "web, db"

`./config.sh --help` always lists the latest required and optional responses.

# Diagnostics

If you're having trouble with your self-hosted agent, you can try running diagnostics. After configuring the agent:

```
./run.sh --diagnostics
```

This will run through a diagnostic suite that may help you troubleshoot the problem. The diagnostics feature is available starting with agent version 2.165.0.

## 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

Capabilities include all environment variables and the values that are set when the agent runs. If any of these values change while the agent is running, the agent must be restarted to pick up the new values. 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.

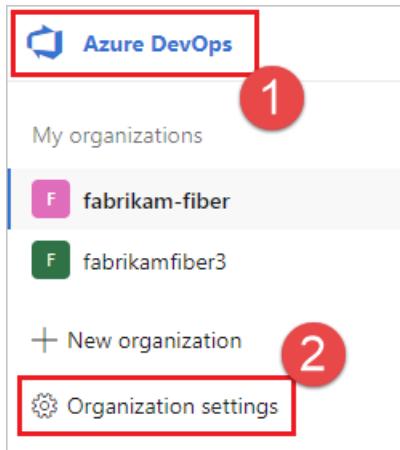
If you want to exclude environment variables as capabilities, you can designate them by setting an environment variable `VSO_AGENT_IGNORE` with a comma-delimited list of variables to ignore.

## FAQ

### How do I make sure I have the latest v2 agent version?

1. Navigate to the **Agent pools** tab:

1. Choose **Azure DevOps, Organization settings**.



2. Choose Agent pools.

This screenshot shows the 'Agent pools' section of the Azure DevOps portal. On the left is a sidebar with links: Security, Policies, Permissions, Boards, Process, Pipelines, Agent pools (highlighted with a red box and a red circle containing the number 1), Settings, Deployment pools, and Parallel jobs. The main area is titled 'Agent pools' and lists four entries: 'Azure Pipelines' (with a cloud icon), 'Default' (with a mobile device icon), 'Test' (with a mobile device icon), and 'Test2' (with a mobile device icon). The 'Name' column is also labeled.

1. Choose Azure DevOps, Collection settings.

This screenshot shows the 'Collection settings' section of the Azure DevOps portal. At the top left is the 'Azure DevOps' logo. Below it is a list of collections: 'fabrikam-fiber' (highlighted with a red box and a red circle containing the number 1) and 'fabrikamfiber3'. At the bottom right of this list is another red box with a red circle containing the number 2, highlighting the 'Collection Settings' link.

2. Choose Agent pools.

Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards  
Process

Pipelines

Agent pools **1**

Settings  
Deployment pools  
Parallel jobs

**Agent pools**

Name
Azure Pipelines Azure Pipelines
Default
Test
Test2

1. Choose Azure DevOps, Collection settings.

Azure DevOps **1**

Collections  
fabrikam-fiber **2**  
fabrikamfiber3  
+ New organization

Collection Settings

2. Choose Agent pools.

Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards  
Process

Pipelines

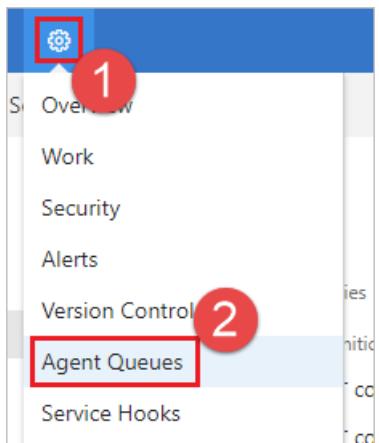
Agent pools **1**

Settings  
Deployment pools  
Parallel jobs

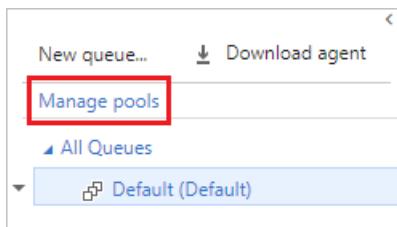
**Agent pools**

Name
Azure Pipelines Azure Pipelines
Default
Test
Test2

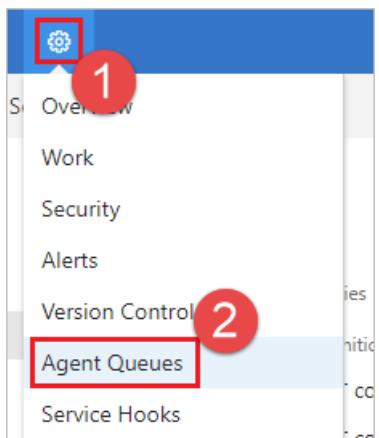
1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



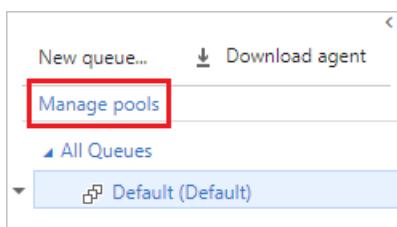
2. Choose **Manage pools**.



1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



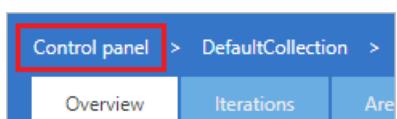
2. Choose **Manage pools**.



1. Navigate to your project and choose **Manage project** (gear icon).



2. Choose **Control panel**.

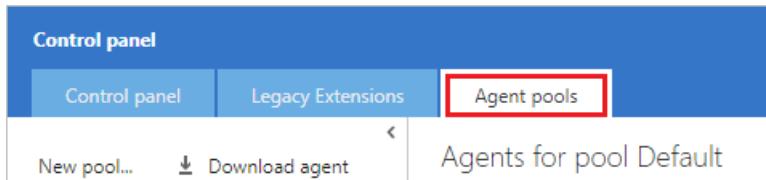


3. Select **Agent pools**.

**Control panel**

Control panel   Legacy Extensions   **Agent pools**

New pool...   Download agent   Agents for pool Default



2. Click the pool that contains the agent.
3. Make sure the agent is enabled.
4. Navigate to the capabilities tab:
  1. From the Agent pools tab, select the desired agent pool.

**Azure DevOps**   FabrikamFiber / Organization Settings / Agent pools

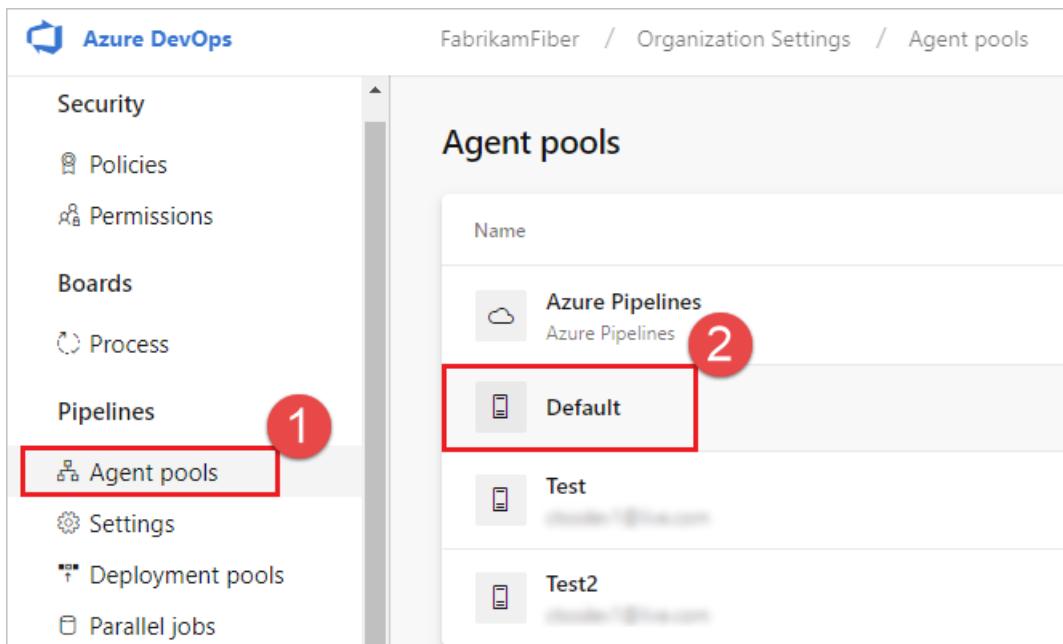
Security  
Policies  
Permissions

Boards  
Process

Pipelines  
**Agent pools** (1)  
Settings  
Deployment pools  
Parallel jobs

**Agent pools**

Name
Azure Pipelines
<b>Default</b> (2)
Test
Test2

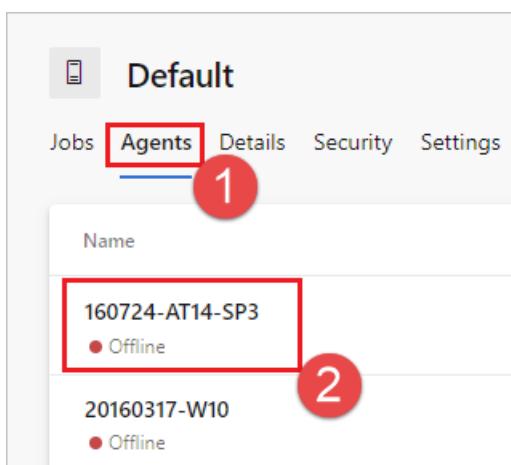


2. Select **Agents** and choose the desired agent.

**Default**

Jobs   **Agents** (1)   Details   Security   Settings

Name
160724-AT14-SP3 (2)
20160317-W10



3. Choose the **Capabilities** tab.

160724-AT14-SP3

Jobs Capabilities

User-defined capabilities

No user-defined capabilities

Add a new capability

System capabilities

Search by keyword

Name	Value
Agent.Name	160724-AT14-SP3
Agent.Version	2.107.0
Agent.ComputerName	160724-AT14-SP3

**NOTE**

Microsoft-hosted agents don't display system capabilities. For a list of software installed on Microsoft-hosted agents, see [Use a Microsoft-hosted agent](#).

- From the Agent pools tab, select the desired pool.

Azure DevOps / Organization Settings / Agent pools

Agent pools

Name
Azure Pipelines
<b>Default</b>
Test
Test2

1

2

- Select Agents and choose the desired agent.

The screenshot shows the 'Agents' tab selected in the top navigation bar. Below the tabs, a table lists two agents. The first agent, '160724-AT14-SP3', is highlighted with a red box and labeled '2'. It is marked as 'Offline'. The second agent, '20160317-W10', is also listed and marked as 'Offline'.

Name	Status
160724-AT14-SP3	Offline
20160317-W10	Offline

3. Choose the **Capabilities** tab.

The screenshot shows the 'Capabilities' tab selected for the agent '160724-AT14-SP3'. The 'User-defined capabilities' section is currently empty, showing a placeholder message 'No user-defined capabilities' and a link to 'Add a new capability'. The 'System capabilities' section displays a table with the following data:

Name	Value
Agent.Name	160724-AT14-SP3
Agent.Version	2.107.0
Agent.ComputerName	160724-AT14-SP3

1. From the **Agent pools** tab, select the desired pool.

Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions  
Boards  
Process  
Pipelines  
**Agent pools** (1)  
Settings  
Deployment pools  
Parallel jobs

## Agent pools

Name
Azure Pipelines (2)
<b>Default</b> (2)
Test
Test2

2. Select **Agents** and choose the desired agent.

**Default**

Jobs Agents Details Security Settings (1)

Name
160724-AT14-SP3 (2)
● Offline
20160317-W10
● Offline

3. Choose the **Capabilities** tab.

160724-AT14-SP3

Jobs Capabilities

User-defined capabilities



No user-defined capabilities

Add a new capability

System capabilities

Search by keyword

Name	Value
Agent.Name	160724-AT14-SP3
Agent.Version	2.107.0
Agent.ComputerName	160724-AT14-SP3

Select the desired agent, and choose the **Capabilities** tab.

Agents for pool Default

Agents Roles

Enabled	Name	Current Status	X
<input checked="" type="checkbox"/>	20160317-W10	Idle	X
<input checked="" type="checkbox"/>	DEV15-TFS-RTM	Idle	X
<input checked="" type="checkbox"/>	DEV15VS	Idle	X

1 Requests Capabilities 2

**USER CAPABILITIES**  
Shows information about user-defined capabilities supported by this host  
[+ Add capability](#)

[Save changes](#) [Undo changes](#)

**SYSTEM CAPABILITIES**  
Shows information about the capabilities provided by this host

AegisRoot	C:\AegisTools
Agent.ComputerName	20160317-W10

Select the desired agent, and choose the **Capabilities** tab.

Agents for pool Default

Agents Roles

Enabled	Name	Current Status	X
<input checked="" type="checkbox"/>	20160317-W10	Idle	X
<input checked="" type="checkbox"/>	DEV15-TFS-RTM	Idle	X
<input checked="" type="checkbox"/>	DEV15VS	Idle	X

1 Requests Capabilities 2

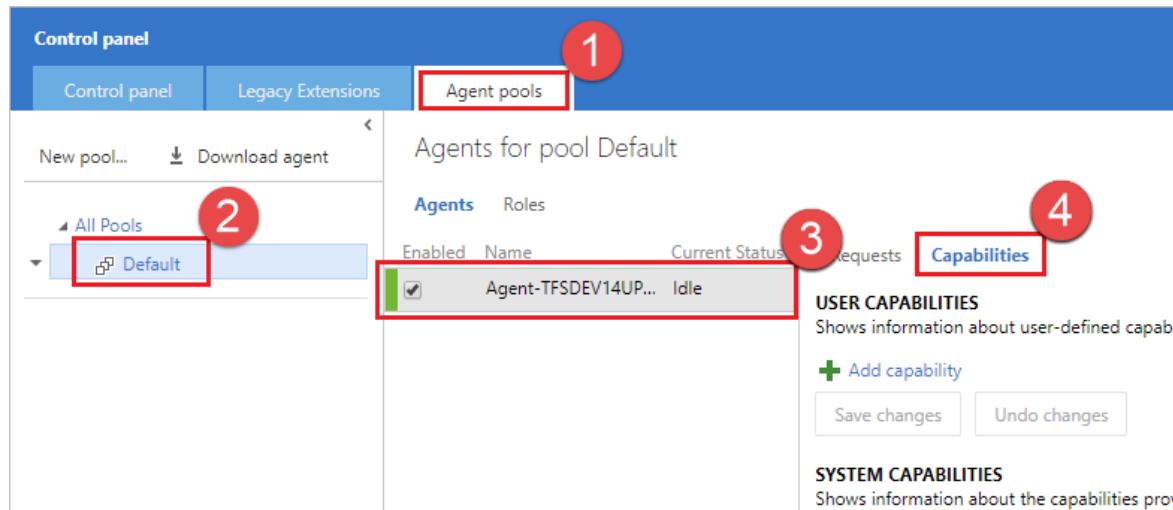
**USER CAPABILITIES**  
Shows information about user-defined capabilities supported by this host  
[+ Add capability](#)

[Save changes](#) [Undo changes](#)

**SYSTEM CAPABILITIES**  
Shows information about the capabilities provided by this host

AegisRoot	C:\AegisTools
Agent.ComputerName	20160317-W10

From the Agent pools tab, select the desired agent, and choose the Capabilities tab.



5. 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.
6. Each agent automatically updates itself when it runs a task that requires a newer version of the agent. If you want to manually update some agents, right-click the pool, and select **Update all agents**.

#### Can I update my v2 agents that are part of an Azure DevOps Server pool?

Yes. Beginning with Azure DevOps Server 2019, you can configure your server to look for the agent package files on a local disk. This configuration will override the default version that came with the server at the time of its release. This scenario also applies when the server doesn't 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 by using a method of your choice (such as USB drive, Network transfer, and so on). Place the agent files under the `%ProgramData%\Microsoft\Azure DevOps\Agents` folder.
3. You're all set! Your Azure DevOps Server will now use the local files whenever the agents are 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}}.vstmr.visualstudio.com
https://{{organization_name}}.pkgs.visualstudio.com
https://{{organization_name}}.vsblob.visualstudio.net
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
https://vstsagentpackage.azureedge.net
https://*.vsblob.visualstudio.net
https://*.vssps.visualstudio.com
```

To ensure your organization works with any existing firewall or IP restrictions, ensure that `dev.azure.com` and `*dev.azure.com` are open and update your allow-listed IPs to include the following IP addresses, based on your IP version. If you're currently allow-listing the `13.107.6.183` and `13.107.9.183` IP addresses, leave them in place, as you don't need to remove them.

#### IPv4 ranges

- `13.107.6.0/24`
- `13.107.9.0/24`
- `13.107.42.0/24`
- `13.107.43.0/24`

#### IPv6 ranges

- `2620:1ec:4::/48`
- `2620:1ec:a92::/48`
- `2620:1ec:21::/48`
- `2620:1ec:22::/48`

#### NOTE

For more information about allowed addresses, see [Allowed address lists and network connections](#).

### 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 restart the agent

If you are running the agent interactively, see the restart instructions in [Run interactively](#). If you are running the agent as a systemd service, follow the steps to [Stop](#) and then [Start](#) the agent.

### 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}}.vstmr.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
https://vstsagentpackage.azureedge.net
```

To ensure your organization works with any existing firewall or IP restrictions, ensure that `dev.azure.com` and `*dev.azure.com` are open and update your allow-listed IPs to include the following IP addresses, based on your IP version. If you're currently allow-listing the `13.107.6.183` and `13.107.9.183` IP addresses, leave them in place, as you don't need to remove them.

#### IPv4 ranges

- `13.107.6.0/24`
- `13.107.9.0/24`
- `13.107.42.0/24`
- `13.107.43.0/24`

#### IPv6 ranges

- `2620:1ec:4::/48`
- `2620:1ec:a92::/48`
- `2620:1ec:21::/48`
- `2620:1ec:22::/48`

#### 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](#).

# Self-hosted macOS agents

11/2/2020 • 21 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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 pipelines are in [Azure Pipelines](#) and a [Microsoft-hosted agent](#) meets your needs, you can skip setting up a self-hosted 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 [Azure Pipelines agents](#).

## Check prerequisites

Make sure your machine has these prerequisites:

- macOS Sierra (10.12) or higher
- Git 2.9.0 or higher (latest version strongly recommended - you can easily install with [Homebrew](#))

These prereqs are required for agent version 2.125.0 and higher.

These prereqs are required for agent version 2.124.0 and below. **If you're able, we recommend upgrading to a newer macOS (10.12+) and upgrading to the newest agent.**

Make sure your machine has these prerequisites:

- OS X Yosemite (10.10), El Capitan (10.11), or macOS Sierra (10.12)
- Git 2.9.0 or higher (latest version strongly recommended)
- Meets all prereqs for [.NET Core 1.x](#)

If you'll be using TFVC, you will also need the [Oracle Java JDK 1.6](#) or higher. (The Oracle JRE and OpenJDK are not sufficient for this purpose.)

## Prepare permissions

If you're building from a Subversion repo, you must install the Subversion client on the machine.

You should run agent setup manually the first time. After you get a feel for how agents work, or if you want to automate setting up many agents, consider using [unattended config](#).

## 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 your Team Foundation Server web portal (  
`https://{{your-server}}:8080/tfs/`).

1. Sign in with the user account you plan to use in your Azure DevOps Server web portal (  
`https://{{your-server}}/DefaultCollection/`).

1. Sign in with the user account you plan to use in your Azure DevOps organization (  
`https://dev.azure.com/{{your_organization}}`).

2. From your home page, open your profile. Go to your security details.

3. [Create a personal access token](#).

2. From your home page, open your user settings, and then select **Personal access tokens**.

3. [Create a personal access 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.

Select **Show all scopes** at the bottom of the **Create a new personal access token window** window to see the complete list of scopes.

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

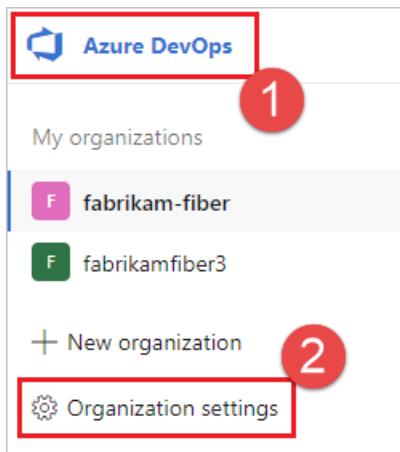
Make sure the user account that you're going to use has permission to register the agent.

Is the user an Azure DevOps organization owner or TFS or Azure DevOps 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 Azure DevOps Server or TFS server:

1. Choose **Azure DevOps, Organization settings**.



2. Choose Agent pools.

This screenshot shows the 'Agent pools' section of the Azure DevOps portal. On the left is a sidebar with navigation links: Security, Policies, Permissions, Boards, Process, Pipelines, Agent pools (which is selected and highlighted in red), Settings, Deployment pools, and Parallel jobs. The main content area is titled 'Agent pools' and lists four entries: 'Azure Pipelines' (with a cloud icon), 'Default' (with a mobile device icon), 'Test' (with a mobile device icon), and 'Test2' (with a mobile device icon). The 'Name' column is on the left, and the agent icons are on the right.

1. Choose Azure DevOps, Collection settings.

This screenshot shows the 'Collection settings' section of the Azure DevOps portal. On the left is a sidebar with navigation links: Collections, fabrikam-fiber (selected and highlighted in blue), fabrikamfiber3, '+ New organization', and Collection Settings (highlighted in red). A red circle with the number '1' is above the 'fabrikam-fiber' entry, and another red circle with the number '2' is above the 'Collection Settings' button.

2. Choose Agent pools.

 Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards  
Process

Pipelines

Agent pools  

Settings  
Deployment pools  
Parallel jobs

**Agent pools**

Name
Azure Pipelines Azure Pipelines
Default
Test
Test2

1. Choose Azure DevOps, Collection settings.

 Azure DevOps 1

Collections

f fabrikam-fiber 2

F fabrikamfiber3

+ New organization

Collection Settings  

2. Choose Agent pools.

 Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards  
Process

Pipelines

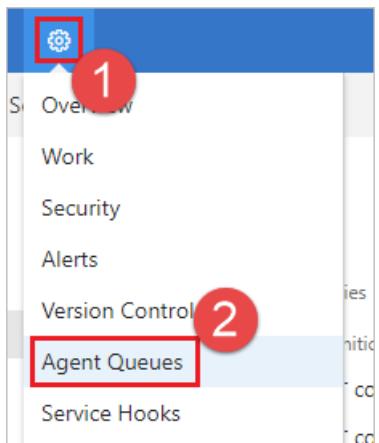
Agent pools  

Settings  
Deployment pools  
Parallel jobs

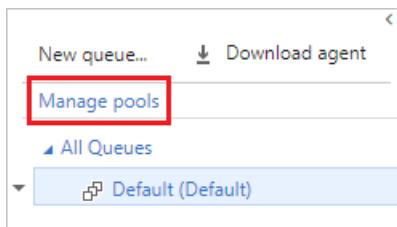
**Agent pools**

Name
Azure Pipelines Azure Pipelines
Default
Test
Test2

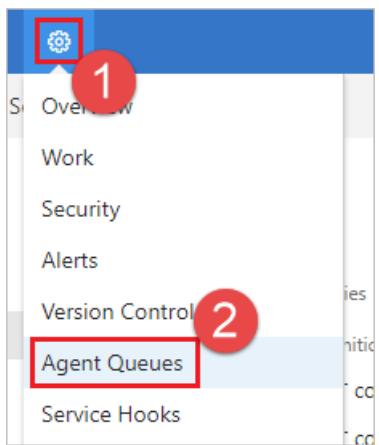
1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



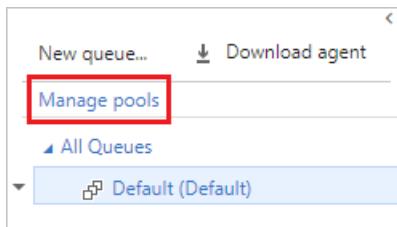
2. Choose **Manage pools**.



1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



2. Choose **Manage pools**.



1. Navigate to your project and choose **Manage project** (gear icon).



2. Choose **Control panel**.



3. Select **Agent pools**.

The screenshot shows the Azure DevOps Control panel. At the top, there are three tabs: 'Control panel' (selected), 'Legacy Extensions', and 'Agent pools'. The 'Agent pools' tab is highlighted with a red border. Below the tabs, there are two buttons: 'New pool...' and 'Download agent'. To the right of these buttons, the text 'Agents for pool Default' is displayed.

2. Select the pool on the right side of the page and then click **Security**.
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 or Azure DevOps 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 or Azure DevOps 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](#).

#### NOTE

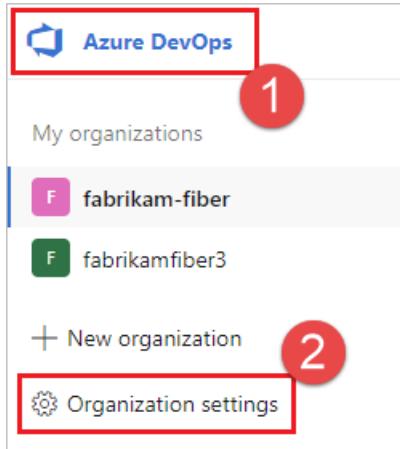
If you see a message like this: **Sorry, we couldn't add the identity. Please try a different identity.**, you probably followed the above steps for an organization owner or TFS or Azure DevOps Server administrator. You don't need to do anything; you already have permission to administer the agent queue.

## 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:

1. Choose **Azure DevOps, Organization settings**.



2. Choose **Agent pools**.

Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards  
Process

Pipelines

Agent pools **1**

Settings  
Deployment pools  
Parallel jobs

**Agent pools**

Name
Azure Pipelines Azure Pipelines
Default
Test
Test2

1. Choose Azure DevOps, Collection settings.

Azure DevOps **1**

Collections  
fabrikam-fiber **2**  
fabrikamfiber3  
+ New organization  
Collection Settings

2. Choose Agent pools.

Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards  
Process

Pipelines

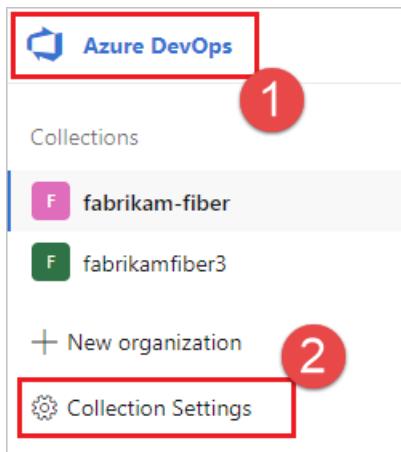
Agent pools **1**

Settings  
Deployment pools  
Parallel jobs

**Agent pools**

Name
Azure Pipelines Azure Pipelines
Default
Test
Test2

1. Choose Azure DevOps, Collection settings.



2. Choose Agent pools.

A screenshot of the Azure DevOps portal. The top navigation bar shows 'FabrikamFiber / Organization Settings / Agent pools'. On the left, there's a sidebar with the following items:

- Security
- Policies
- Permissions
- Boards
- Process
- Pipelines
- Agent pools (highlighted with a red box)
- Settings
- Deployment pools
- Parallel jobs

The main area is titled 'Agent pools' and lists four entries:

Name
Azure Pipelines
Default
Test
Test2

1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.

A screenshot of the Azure DevOps project settings menu. The top bar has a gear icon (highlighted with a red box) and the text 'Settings Overview'. The menu items are:

- Work
- Security
- Alerts
- Version Control
- Agent Queues (highlighted with a red box)
- Service Hooks

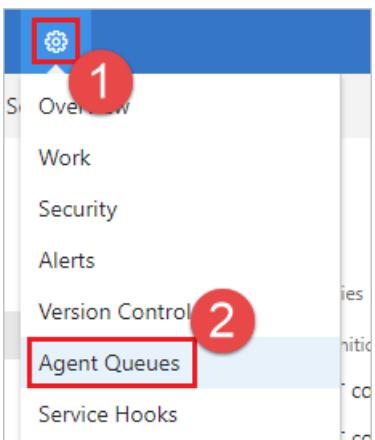
Red numbers 1 and 2 are overlaid on the screen: 1 is above the gear icon, and 2 is above the Agent Queues item.

2. Choose **Manage pools**.

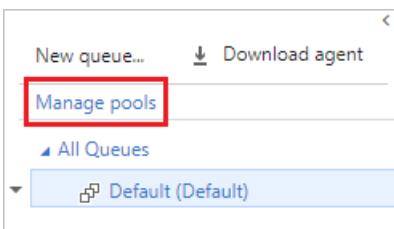
A screenshot of the 'Agent Queues' page. At the top are buttons for 'New queue...' and 'Download agent'. Below them is a list:

- Manage pools (highlighted with a red box)
- All Queues
- Default (Default)

1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



2. Choose **Manage pools**.



1. Navigate to your project and choose **Manage project** (gear icon).



2. Choose **Control panel**.



3. Select **Agent pools**.



3. Select the **Default** pool, select the **Agents** tab, and choose **New agent**.

4. On the **Get the agent** dialog box, click **macOS**.

5. Click the **Download** button.

6. Follow the instructions on the page.

7. Clear the extended attribute on the tar file: `xattr -c vsts-agent-osx-x64-V.v.v.tar.gz`.

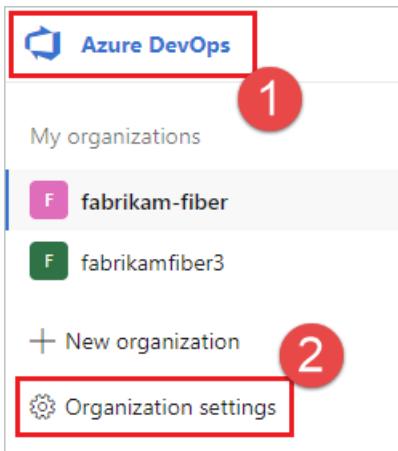
8. Unpack the agent into the directory of your choice. `cd` to that directory and run `./config.sh`. Make sure that the path to the directory contains no spaces because tools and scripts don't always properly escape spaces.

## Azure DevOps Server 2019 and Azure DevOps Server 2020

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 DevOps Server, and navigate to the **Agent pools** tab:

1. Choose Azure DevOps, Organization settings.

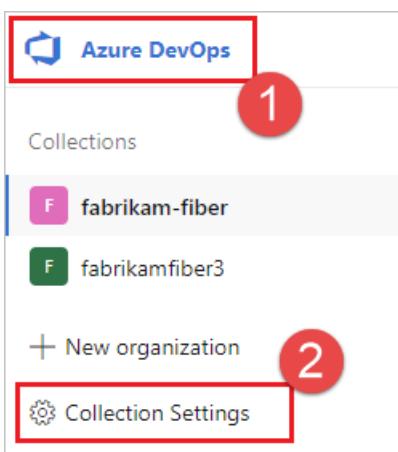


2. Choose Agent pools.

A screenshot showing the 'Agent pools' section within the 'Organization Settings' of the 'FabrikamFiber' organization. On the left, a sidebar lists 'Security', 'Policies', 'Permissions', 'Boards', 'Process', 'Pipelines', 'Agent pools' (highlighted with a red box and circled with a red number 1), 'Settings', 'Deployment pools', and 'Parallel jobs'. The main area is titled 'Agent pools' and shows a table with four rows:

Name
Azure Pipelines
Default
Test
Test2

1. Choose Azure DevOps, Collection settings.



2. Choose Agent pools.

 Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards  
Process

Pipelines

Agent pools  

Settings  
Deployment pools  
Parallel jobs

**Agent pools**

Name
Azure Pipelines
Default
Test
Test2

1. Choose Azure DevOps, Collection settings.

 Azure DevOps 1

Collections

f fabrikam-fiber 2

F fabrikamfiber3

+ New organization

Collection Settings 2

2. Choose Agent pools.

 Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards  
Process

Pipelines

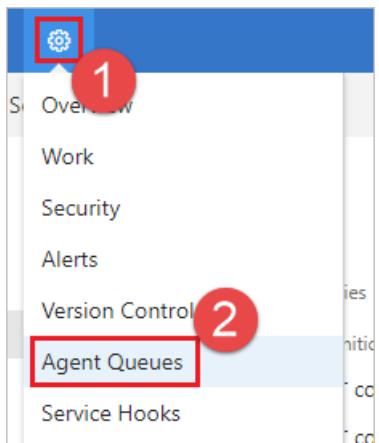
Agent pools  

Settings  
Deployment pools  
Parallel jobs

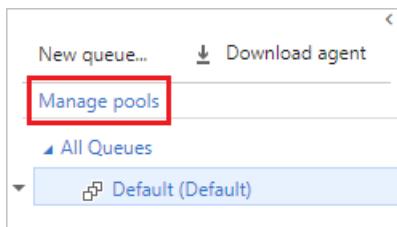
**Agent pools**

Name
Azure Pipelines
Default
Test
Test2

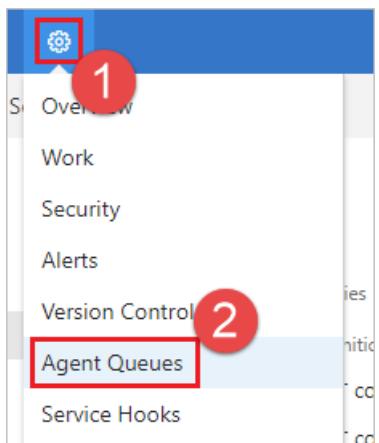
1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



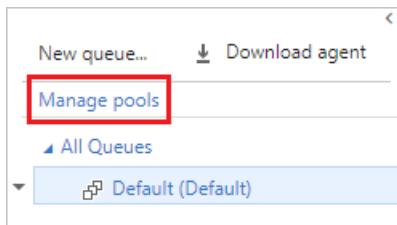
2. Choose **Manage pools**.



1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



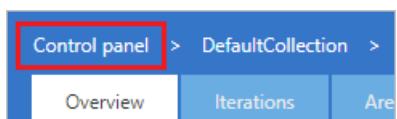
2. Choose **Manage pools**.



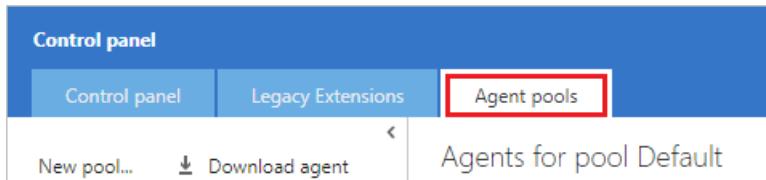
1. Navigate to your project and choose **Manage project** (gear icon).



2. Choose **Control panel**.



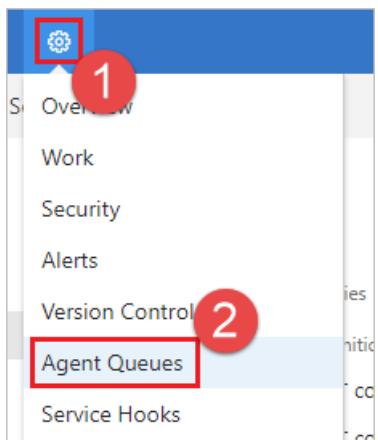
3. Select **Agent pools**.



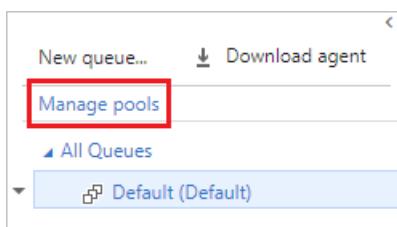
3. Click **Download agent**.
4. On the **Get agent** dialog box, click **macOS**.
5. Click the **Download** button.
6. Follow the instructions on the page.
7. Clear the extended attribute on the tar file: `xattr -c vsts-agent-osx-x64-V.v.v.tar.gz`.
8. Unpack the agent into the directory of your choice. `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 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 Azure Pipelines or TFS, and navigate to the **Agent pools** tab:
  - a. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



- b. Choose **Manage pools**.



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. Clear the extended attribute on the tar file: `xattr -c vsts-agent-osx-x64-V.v.v.tar.gz`.
8. 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.

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}/tfs](https://{your_server}/tfs)

TFS 2015: [http://{your\\_server}:8080/tfs](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 or Azure DevOps Server

#### 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 or Azure DevOps Server 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 or Azure DevOps Server 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 Azure DevOps Server or TFS instance and the agent machine are not in a trusted domain. PAT authentication is handled by your Azure DevOps Server or 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 Azure DevOps Server and the 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 restart the agent, press Ctrl+C and then run `run.sh` to restart it.

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 once

For agents configured to run interactively, you can choose to have the agent accept only one job. To run in this configuration:

```
./run.sh --once
```

Agents in this mode will accept only one job and then spin down gracefully (useful for running on a service like Azure Container Instances).

## 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.{tfs-name}.{agent-name}:

/Users/{your-name}/Library/LaunchAgents/vsts.{tfs-name}.{agent-name}.testsrv.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.

## Stop

Command:

```
./svc.sh stop
```

Output:

```
stopping vsts.agent.{tfs-name}.{agent-name}
status vsts.agent.{tfs-name}.{agent-name}:

/Users/{your-name}/Library/LaunchAgents/vsts.{tfs-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](#).

#### NOTE

For more information, see the [Terminally Geeky: use automatic login more securely](#) blog. The .plist file mentioned in that blog may no longer be available at the source, but a copy can be found here: [Lifehacker - Make OS X load your desktop before you log in](#).

## 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):

```
./env.sh
./svc.sh stop
./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.

To configure an agent, it must know the URL to your organization or collection and credentials of someone authorized to set up agents. All other responses are optional. Any command-line parameter can be specified using an environment variable instead: put its name in upper case and prepend `VSTS_AGENT_INPUT_`. For example, `VSTS_AGENT_INPUT_PASSWORD` instead of specifying `--password`.

### Required options

- `--unattended` - agent setup will not prompt for information, and all settings must be provided on the command line
- `--url <url>` - URL of the server. For example: <https://dev.azure.com/myorganization> or <http://my-azure-devops-server:8080/tfs>

- `--auth <type>` - authentication type. Valid values are:
  - `pat` (Personal access token)
  - `negotiate` (Kerberos or NTLM)
  - `alt` (Basic authentication)
  - `integrated` (Windows default credentials)

## Authentication options

- If you chose `--auth pat`:
  - `--token <token>` - specifies your personal access token
- If you chose `--auth negotiate` or `--auth alt`:
  - `--userName <userName>` - specifies a Windows username in the format `domain\userName` or `userName@domain.com`
  - `--password <password>` - specifies a password

## Pool and agent names

- `--pool <pool>` - pool name for the agent to join
- `--agent <agent>` - agent name
- `--replace` - replace the agent in a pool. If another agent is listening by the same name, it will start failing with a conflict

## Agent setup

- `--work <workDirectory>` - work directory where job data is stored. Defaults to `_work` under the root of the agent directory. The work directory is owned by a given agent and should not share between multiple agents.
- `--acceptTeeEula` - accept the Team Explorer Everywhere End User License Agreement (macOS and Linux only)
- `--disableloguploads` - don't stream or send console log output to the server. Instead, you may retrieve them from the agent host's filesystem after the job completes.

## Windows-only startup

- `--runAsService` - configure the agent to run as a Windows service (requires administrator permission)
- `--runAsAutoLogon` - configure auto-logon and run the agent on startup (requires administrator permission)
- `--windowsLogonAccount <account>` - used with `--runAsService` or `--runAsAutoLogon` to specify the Windows user name in the format `domain\userName` or `userName@domain.com`
- `--windowsLogonPassword <password>` - used with `--runAsService` or `--runAsAutoLogon` to specify Windows logon password
- `--overwriteAutoLogon` - used with `--runAsAutoLogon` to overwrite the existing auto logon on the machine
- `--noRestart` - used with `--runAsAutoLogon` to stop the host from restarting after agent configuration completes

## Deployment group only

- `--deploymentGroup` - configure the agent as a deployment group agent
- `--deploymentGroupName <name>` - used with `--deploymentGroup` to specify the deployment group for the agent to join
- `--projectName <name>` - used with `--deploymentGroup` to set the project name
- `--addDeploymentGroupTags` - used with `--deploymentGroup` to indicate that deployment group tags should be added
- `--deploymentGroupTags <tags>` - used with `--addDeploymentGroupTags` to specify the comma separated list of tags for the deployment group agent - for example "web, db"

## Environments only

- `--addvirtualmachineresourcetags` - used to indicate that environment resource tags should be added
- `--virtualmachineresourcetags <tags>` - used with `--addvirtualmachineresourcetags` to specify the comma separated list of tags for the environment resource agent - for example "web, db"

```
./config.sh --help
```

 always lists the latest required and optional responses.

## Diagnostics

If you're having trouble with your self-hosted agent, you can try running diagnostics. After configuring the agent:

```
./run.sh --diagnostics
```

This will run through a diagnostic suite that may help you troubleshoot the problem. The diagnostics feature is available starting with agent version 2.165.0.

## 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

Capabilities include all environment variables and the values that are set when the agent runs. If any of these values change while the agent is running, the agent must be restarted to pick up the new values. 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.

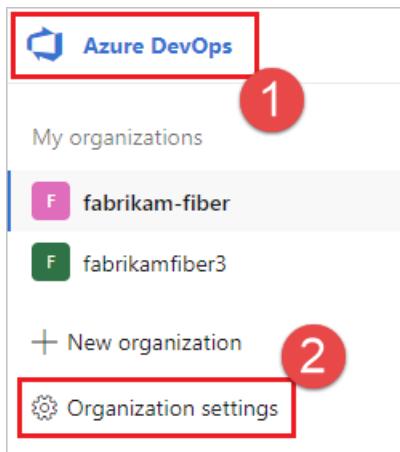
If you want to exclude environment variables as capabilities, you can designate them by setting an environment variable `VSO_AGENT_IGNORE` with a comma-delimited list of variables to ignore.

## FAQ

### How do I make sure I have the latest v2 agent version?

1. Navigate to the Agent pools tab:

1. Choose Azure DevOps, Organization settings.



2. Choose Agent pools.

This screenshot shows the 'Agent pools' section of the Azure DevOps portal. On the left is a sidebar with links: Security, Policies, Permissions, Boards, Process, Pipelines, Agent pools (highlighted with a red box and a red circle containing the number 1), Settings, Deployment pools, and Parallel jobs. The main area is titled 'Agent pools' and lists four entries: 'Azure Pipelines' (with a cloud icon), 'Default' (with a mobile device icon), 'Test' (with a mobile device icon), and 'Test2' (with a mobile device icon). The 'Name' column is also labeled.

1. Choose Azure DevOps, Collection settings.

This screenshot shows the 'Collection settings' section of the Azure DevOps portal. At the top left is the 'Azure DevOps' logo. Below it is a list of collections: 'fabrikam-fiber' (highlighted with a red box and a red circle containing the number 1) and 'fabrikamfiber3'. At the bottom right of this list is another red box with a red circle containing the number 2, highlighting the 'Collection Settings' link.

2. Choose Agent pools.

 Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions  
Boards  
Process  
Pipelines  
**Agent pools**  
Settings  
Deployment pools  
Parallel jobs

**Agent pools**

Name
Azure Pipelines
Default
Test
Test2

1. Choose Azure DevOps, Collection settings.

 Azure DevOps      1

Collections  
fabrikam-fiber      2  
fabrikamfiber3  
+ New organization  
Collection Settings

2. Choose Agent pools.

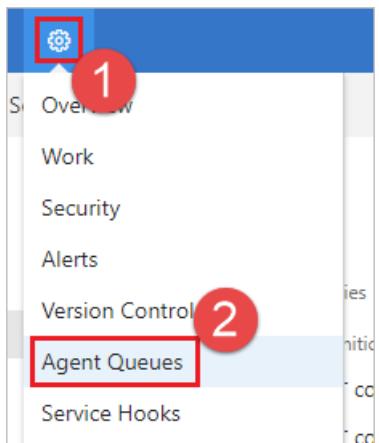
 Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions  
Boards  
Process  
Pipelines  
**Agent pools**  
Settings  
Deployment pools  
Parallel jobs

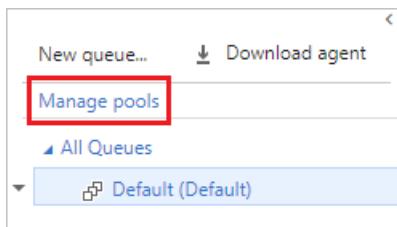
**Agent pools**

Name
Azure Pipelines
Default
Test
Test2

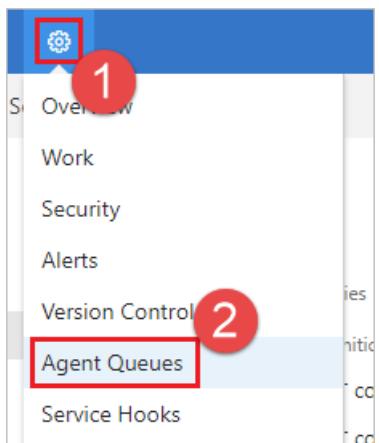
1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



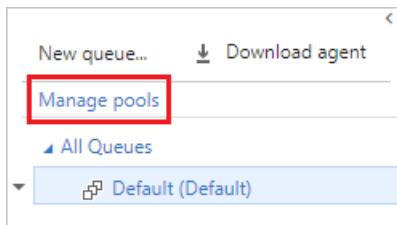
2. Choose **Manage pools**.



1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



2. Choose **Manage pools**.



1. Navigate to your project and choose **Manage project** (gear icon).



2. Choose **Control panel**.

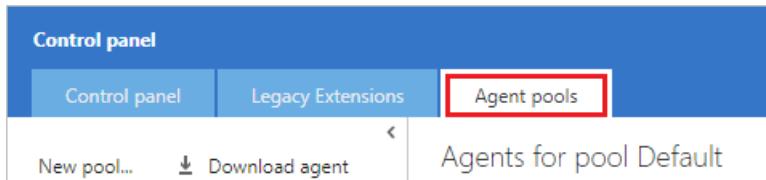


3. Select **Agent pools**.

**Control panel**

Control panel   Legacy Extensions   **Agent pools**

New pool...   Download agent   Agents for pool Default



2. Click the pool that contains the agent.
3. Make sure the agent is enabled.
4. Navigate to the capabilities tab:
  1. From the Agent pools tab, select the desired agent pool.

**Azure DevOps**   FabrikamFiber / Organization Settings / Agent pools

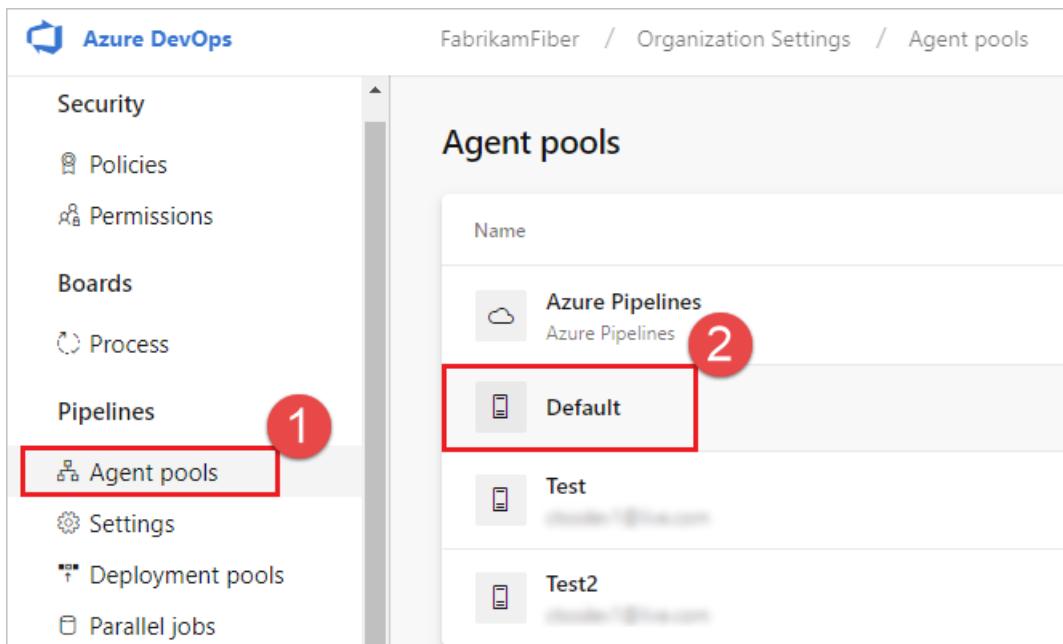
Security  
Policies  
Permissions

Boards  
Process

Pipelines  
**Agent pools** (1)  
Settings  
Deployment pools  
Parallel jobs

**Agent pools**

Name
Azure Pipelines
<b>Default</b> (2)
Test
Test2

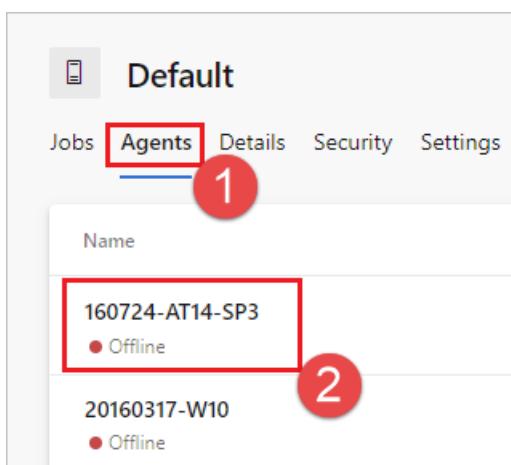


2. Select **Agents** and choose the desired agent.

**Default**

Jobs   **Agents** (1)   Details   Security   Settings

Name
160724-AT14-SP3 (2)
20160317-W10



3. Choose the **Capabilities** tab.

160724-AT14-SP3

Jobs Capabilities

User-defined capabilities

No user-defined capabilities

Add a new capability

System capabilities

Search by keyword

Name	Value
Agent.Name	160724-AT14-SP3
Agent.Version	2.107.0
Agent.ComputerName	160724-AT14-SP3

**NOTE**

Microsoft-hosted agents don't display system capabilities. For a list of software installed on Microsoft-hosted agents, see [Use a Microsoft-hosted agent](#).

- From the Agent pools tab, select the desired pool.

Azure DevOps / Organization Settings / Agent pools

Agent pools

Name
Azure Pipelines
<b>Default</b>
Test
Test2

1

2

- Select Agents and choose the desired agent.

The screenshot shows the 'Agents' tab selected in the top navigation bar. Below the tabs, a table lists two agents. The first agent, '160724-AT14-SP3', is highlighted with a red box and a red circle labeled '2'. It is listed as 'Offline'. The second agent, '20160317-W10', is also listed as 'Offline'.

Name	Status
160724-AT14-SP3	Offline
20160317-W10	Offline

3. Choose the **Capabilities** tab.

The screenshot shows the 'Capabilities' tab selected for the agent '160724-AT14-SP3'. The 'User-defined capabilities' section is currently empty, showing a placeholder message 'No user-defined capabilities' and a link to 'Add a new capability'. The 'System capabilities' section displays a table with the following data:

Name	Value
Agent.Name	160724-AT14-SP3
Agent.Version	2.107.0
Agent.ComputerName	160724-AT14-SP3

1. From the **Agent pools** tab, select the desired pool.

Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions  
Boards  
Process  
Pipelines  
**Agent pools** (1)  
Settings  
Deployment pools  
Parallel jobs

## Agent pools

Name
Azure Pipelines (2)
<b>Default</b> (2)
Test
Test2

2. Select **Agents** and choose the desired agent.

**Default**

Jobs Agents Details Security Settings (1)

Name
160724-AT14-SP3 (2)
● Offline
20160317-W10
● Offline

3. Choose the **Capabilities** tab.

160724-AT14-SP3

Jobs Capabilities

User-defined capabilities



No user-defined capabilities

Add a new capability

System capabilities

Search by keyword

Name	Value
Agent.Name	160724-AT14-SP3
Agent.Version	2.107.0
Agent.ComputerName	160724-AT14-SP3

Select the desired agent, and choose the **Capabilities** tab.

Agents for pool Default

Agents Roles

Enabled	Name	Current Status	X
<input checked="" type="checkbox"/>	20160317-W10	Idle	X
<input checked="" type="checkbox"/>	DEV15-TFS-RTM	Idle	X
<input checked="" type="checkbox"/>	DEV15VS	Idle	X

1 Requests Capabilities 2

**USER CAPABILITIES**  
Shows information about user-defined capabilities supported by this host

+ Add capability

Save changes Undo changes

**SYSTEM CAPABILITIES**  
Shows information about the capabilities provided by this host

AegisRoot	C:\AegisTools
Agent.ComputerName	20160317-W10

Select the desired agent, and choose the **Capabilities** tab.

Agents for pool Default

Agents Roles

Enabled	Name	Current Status	X
<input checked="" type="checkbox"/>	20160317-W10	Idle	X
<input checked="" type="checkbox"/>	DEV15-TFS-RTM	Idle	X
<input checked="" type="checkbox"/>	DEV15VS	Idle	X

1 Requests Capabilities 2

**USER CAPABILITIES**  
Shows information about user-defined capabilities supported by this host

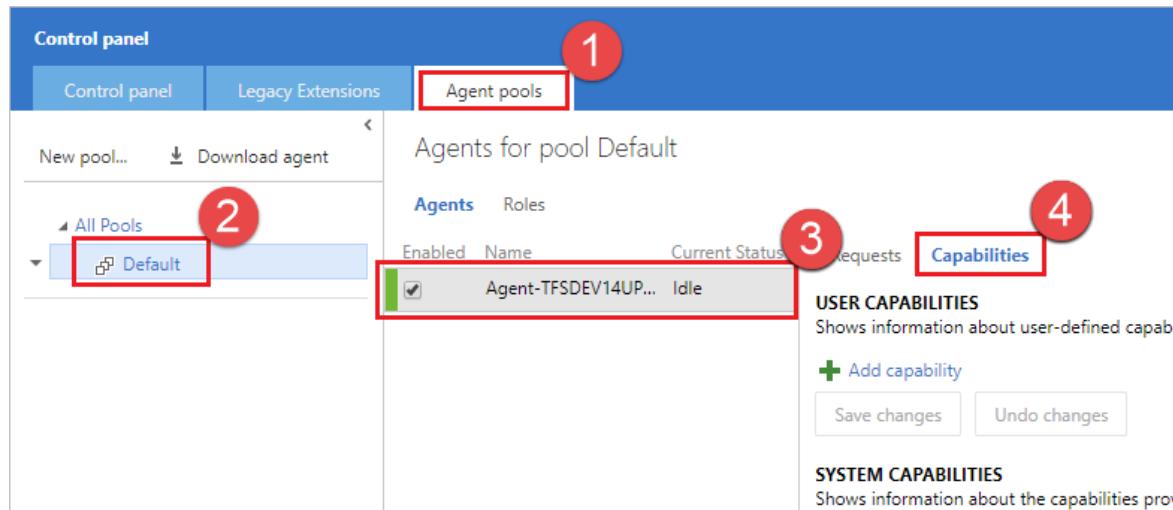
+ Add capability

Save changes Undo changes

**SYSTEM CAPABILITIES**  
Shows information about the capabilities provided by this host

AegisRoot	C:\AegisTools
Agent.ComputerName	20160317-W10

From the Agent pools tab, select the desired agent, and choose the Capabilities tab.



5. 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.
6. Each agent automatically updates itself when it runs a task that requires a newer version of the agent. If you want to manually update some agents, right-click the pool, and select **Update all agents**.

#### Can I update my v2 agents that are part of an Azure DevOps Server pool?

Yes. Beginning with Azure DevOps Server 2019, you can configure your server to look for the agent package files on a local disk. This configuration will override the default version that came with the server at the time of its release. This scenario also applies when the server doesn't 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 by using a method of your choice (such as USB drive, Network transfer, and so on). Place the agent files under the `%ProgramData%\Microsoft\Azure DevOps\Agents` folder.
3. You're all set! Your Azure DevOps Server will now use the local files whenever the agents are 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}}.vstmr.visualstudio.com
https://{{organization_name}}.pkgs.visualstudio.com
https://{{organization_name}}.vsblob.visualstudio.net
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
https://vstsagentpackage.azureedge.net
https://*.vsblob.visualstudio.net
https://*.vssps.visualstudio.com
```

To ensure your organization works with any existing firewall or IP restrictions, ensure that `dev.azure.com` and `*dev.azure.com` are open and update your allow-listed IPs to include the following IP addresses, based on your IP version. If you're currently allow-listing the `13.107.6.183` and `13.107.9.183` IP addresses, leave them in place, as you don't need to remove them.

#### IPv4 ranges

- `13.107.6.0/24`
- `13.107.9.0/24`
- `13.107.42.0/24`
- `13.107.43.0/24`

#### IPv6 ranges

- `2620:1ec:4::/48`
- `2620:1ec:a92::/48`
- `2620:1ec:21::/48`
- `2620:1ec:22::/48`

#### NOTE

For more information about allowed addresses, see [Allowed address lists and network connections](#).

### 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 restart the agent

If you are running the agent interactively, see the restart instructions in [Run interactively](#). If you are running the agent as a service, follow the steps to [Stop](#) and then [Start](#) the agent.

### 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}}.vstmn.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
https://vstsagentpackage.azureedge.net
```

To ensure your organization works with any existing firewall or IP restrictions, ensure that `dev.azure.com` and `*dev.azure.com` are open and update your allow-listed IPs to include the following IP addresses, based on your IP version. If you're currently allow-listing the `13.107.6.183` and `13.107.9.183` IP addresses, leave them in place, as you don't need to remove them.

#### IPv4 ranges

- `13.107.6.0/24`
- `13.107.9.0/24`
- `13.107.42.0/24`
- `13.107.43.0/24`

#### IPv6 ranges

- `2620:1ec:4::/48`
- `2620:1ec:a92::/48`
- `2620:1ec:21::/48`
- `2620:1ec:22::/48`

#### 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](#).

# Self-hosted Windows agents

3/6/2021 • 19 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#) | [Previous versions \(XAML builds\)](#)

## IMPORTANT

For TFS 2015, see [Self-hosted Windows agents - TFS 2015](#).

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 self-hosted 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 [Azure Pipelines agents](#).

## Check prerequisites

Make sure your machine has these prerequisites:

- Windows 7, 8.1, or 10 (if using a client OS)
- Windows 2008 R2 SP1 or higher (if using a server OS)
- [PowerShell 3.0](#) or higher
- [.NET Framework](#) 4.6.2 or higher

## IMPORTANT

Starting December 2019, the minimum required .NET version for build agents is 4.6.2 or higher.

Recommended:

- [Visual Studio build tools](#) (2015 or higher)

If you're building from a Subversion repo, you must install the [Subversion client](#) on the machine.

You should run agent setup manually the first time. After you get a feel for how agents work, or if you want to automate setting up many agents, consider using [unattended config](#).

## 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 the hosted

agents code using pipelines that utilize [hosted agents](#). On the other hand, the bulk of the Azure DevOps code is built by 24-core server class machines running 4 self-hosted 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 your Team Foundation Server web portal (  
`https://{{your-server}}:8080/tfs/`).

1. Sign in with the user account you plan to use in your Azure DevOps Server web portal (  
`https://{{your-server}}/DefaultCollection/`).

1. Sign in with the user account you plan to use in your Azure DevOps organization (  
`https://dev.azure.com/{{your_organization}}`).

2. From your home page, open your profile. Go to your security details.

3. [Create a personal access token](#).

2. From your home page, open your user settings, and then select **Personal access tokens**.

3. [Create a personal access 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.

Select **Show all scopes** at the bottom of the **Create a new personal access token window** window to see the complete list of scopes.

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

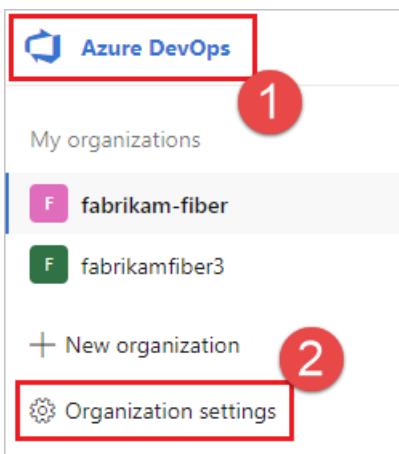
Make sure the user account that you're going to use has permission to register the agent.

Is the user an Azure DevOps organization owner or TFS or Azure DevOps 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 Azure DevOps Server or TFS server:

1. Choose **Azure DevOps, Organization settings**.

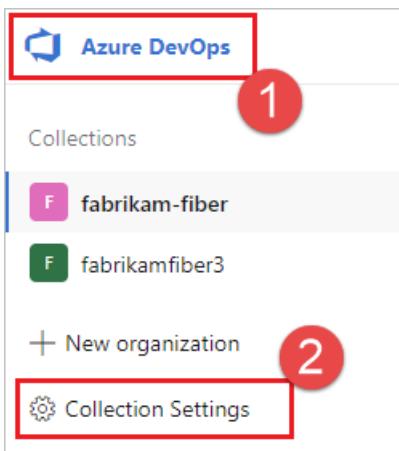


2. Choose **Agent pools**.

The screenshot shows the 'Agent pools' page under 'Organization Settings'. The left sidebar has a red box around the 'Agent pools' item, which is highlighted in blue. A red circle with the number '1' is above the sidebar. The main area shows a table titled 'Agent pools' with four rows:

Name
Azure Pipelines
Default
Test
Test2

1. Choose **Azure DevOps, Collection settings**.



2. Choose **Agent pools**.

 Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions  
Boards  
Process  
Pipelines  
**Agent pools**  
Settings  
Deployment pools  
Parallel jobs

**Agent pools**

Name
Azure Pipelines
Default
Test
Test2

1. Choose Azure DevOps, Collection settings.

 Azure DevOps      1

Collections  
fabrikam-fiber      2  
fabrikamfiber3  
+ New organization  
Collection Settings

2. Choose Agent pools.

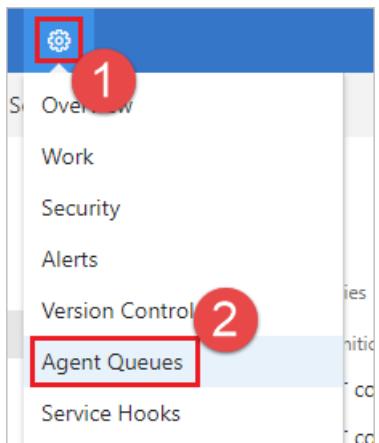
 Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions  
Boards  
Process  
Pipelines  
**Agent pools**  
Settings  
Deployment pools  
Parallel jobs

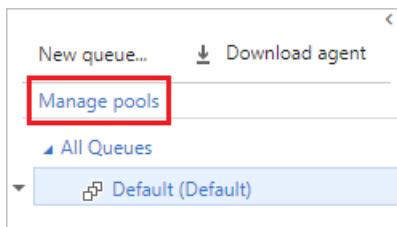
**Agent pools**

Name
Azure Pipelines
Default
Test
Test2

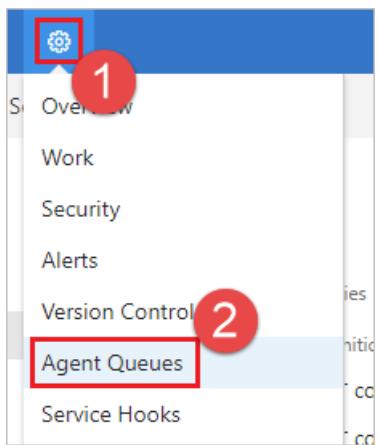
1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



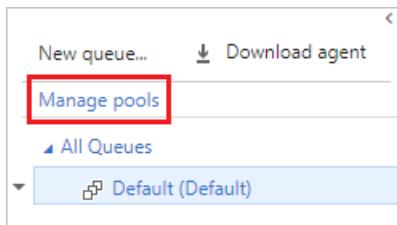
2. Choose **Manage pools**.



1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



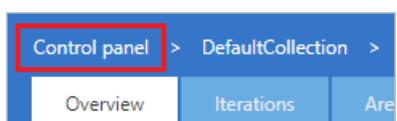
2. Choose **Manage pools**.



1. Navigate to your project and choose **Manage project** (gear icon).



2. Choose **Control panel**.



3. Select **Agent pools**.

The screenshot shows the Azure DevOps Control panel. At the top, there are three tabs: 'Control panel' (selected), 'Legacy Extensions', and 'Agent pools'. The 'Agent pools' tab is highlighted with a red border. Below the tabs, there are two buttons: 'New pool...' and 'Download agent'. To the right of these buttons, the text 'Agents for pool Default' is displayed.

2. Select the pool on the right side of the page and then click **Security**.
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 or Azure DevOps 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 or Azure DevOps 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](#).

#### NOTE

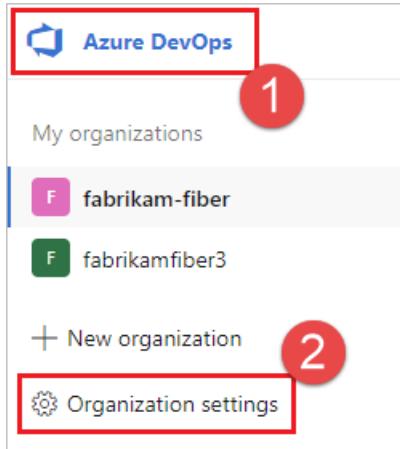
If you see a message like this: **Sorry, we couldn't add the identity. Please try a different identity.**, you probably followed the above steps for an organization owner or TFS or Azure DevOps Server administrator. You don't need to do anything; you already have permission to administer the agent queue.

## 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:

1. Choose **Azure DevOps, Organization settings**.



2. Choose **Agent pools**.

Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards  
Process

Pipelines

Agent pools **1**

Settings  
Deployment pools  
Parallel jobs

**Agent pools**

Name
Azure Pipelines
Default
Test
Test2

1. Choose Azure DevOps, Collection settings.

Azure DevOps **1**

Collections  
fabrikam-fiber **2**  
fabrikamfiber3

+ New organization

Collection Settings

2. Choose Agent pools.

Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards  
Process

Pipelines

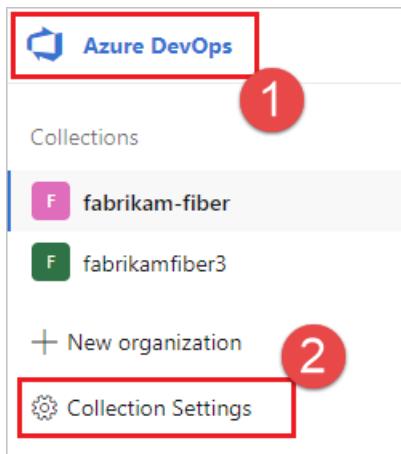
Agent pools **1**

Settings  
Deployment pools  
Parallel jobs

**Agent pools**

Name
Azure Pipelines
Default
Test
Test2

1. Choose Azure DevOps, Collection settings.



2. Choose Agent pools.

Agent pools

Name
Azure Pipelines
Default
Test
Test2

1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.

1

2

2. Choose **Manage pools**.

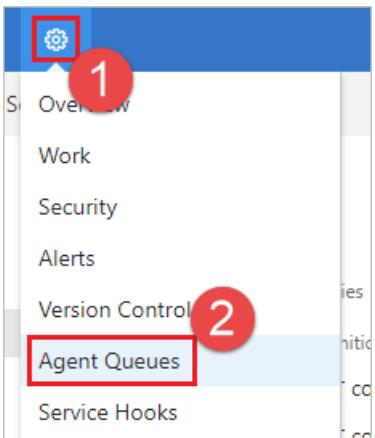
New queue... Download agent

Manage pools

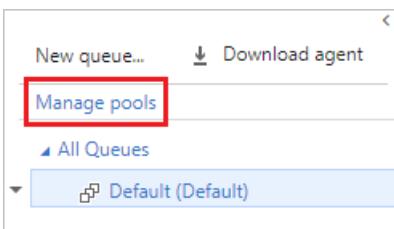
All Queues

Default (Default)

1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



2. Choose **Manage pools**.



1. Navigate to your project and choose **Manage project** (gear icon).



2. Choose **Control panel**.



3. Select **Agent pools**.



3. Select the **Default** pool, select the **Agents** tab, and choose **New agent**.

4. On the **Get the agent** dialog box, choose **Windows**.

5. On the left pane, select the processor architecture of 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. If you aren't sure which version of Windows is installed, [follow these instructions to find out](#).

6. On the right pane, click the **Download** button.

7. Follow the instructions on the page to download the agent.

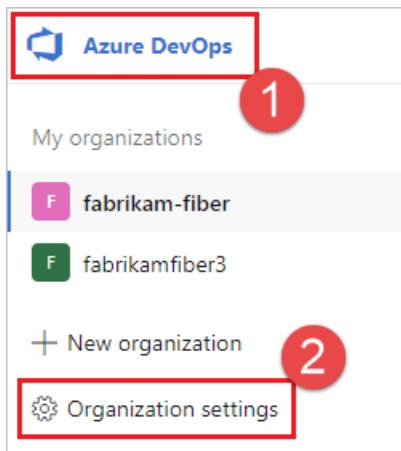
8. Unpack the agent into the directory of your choice. Then run `config.cmd`. This will ask you a series of questions to configure the agent.

## Azure DevOps Server 2019 and Azure DevOps Server 2020

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 DevOps Server 2019, and navigate to the Agent pools tab:

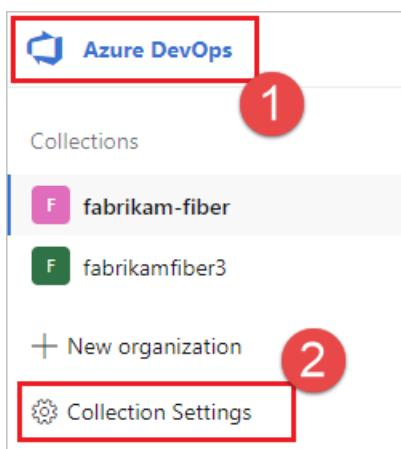
1. Choose Azure DevOps, Organization settings.



2. Choose Agent pools.

A screenshot of the 'Agent pools' page under 'Organization Settings'. The left sidebar shows navigation options: Security, Policies, Permissions, Boards, Process, Pipelines, Agent pools (which is highlighted with a red box and has a red number '1'), Settings, Deployment pools, and Parallel jobs. The main area is titled 'Agent pools' and lists four entries: 'Azure Pipelines' (cloud icon), 'Default' (server icon), 'Test' (server icon), and 'Test2' (server icon).

1. Choose Azure DevOps, Collection settings.



2. Choose Agent pools.

 Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards  
Process

Pipelines

Agent pools  

Settings  
Deployment pools  
Parallel jobs

**Agent pools**

Name
Azure Pipelines
Default
Test
Test2

1. Choose Azure DevOps, Collection settings.

 Azure DevOps 1

Collections

f fabrikam-fiber 2

F fabrikamfiber3

+ New organization

Collection Settings 2

2. Choose Agent pools.

 Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards  
Process

Pipelines

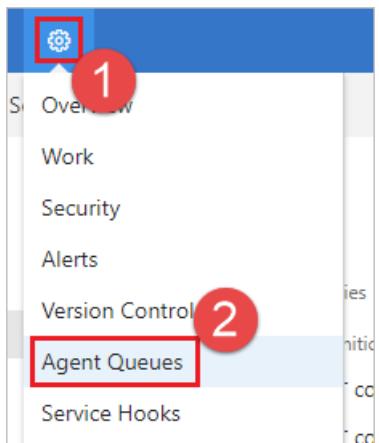
Agent pools  

Settings  
Deployment pools  
Parallel jobs

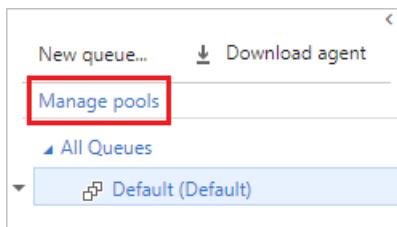
**Agent pools**

Name
Azure Pipelines
Default
Test
Test2

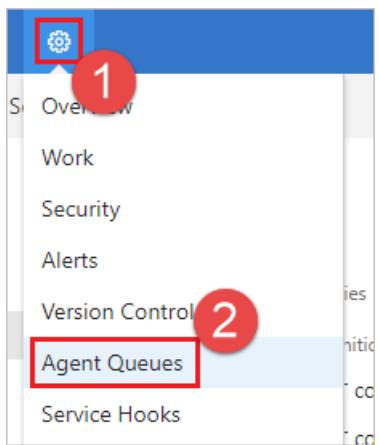
1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



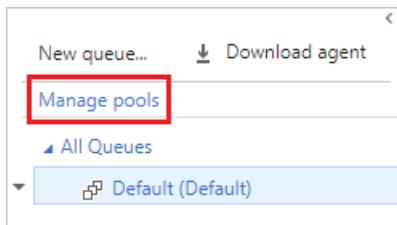
2. Choose **Manage pools**.



1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



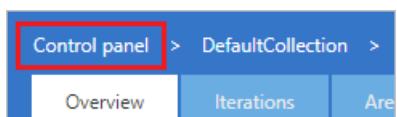
2. Choose **Manage pools**.



1. Navigate to your project and choose **Manage project** (gear icon).



2. Choose **Control panel**.



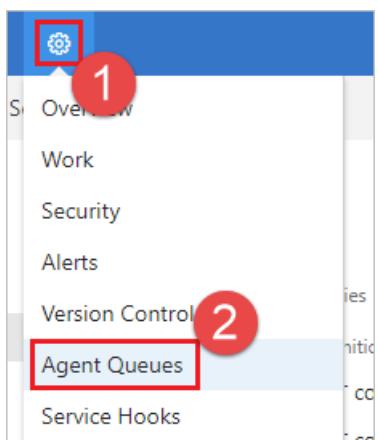
3. Select **Agent pools**.

The screenshot shows the 'Control panel' interface with the 'Agent pools' tab highlighted by a red box. Below the tabs, there are two buttons: 'New pool...' and 'Download agent'. To the right of the buttons, the text 'Agents for pool Default' is displayed.

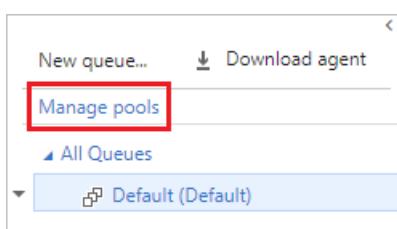
3. Click **Download agent**.
4. On the **Get agent** dialog box, click **Windows**.
5. On the left pane, select the processor architecture of the installed Windows OS version on your machine. The x64 agent version is intended for 64-bit Windows, whereas the x86 version is intended for 32-bit Windows. If you aren't sure which version of Windows is installed, [follow these instructions to find out](#).
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`. This will ask you a series of questions to configure the agent.

#### 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:
  - a. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



- b. Choose **Manage pools**.



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 strongly recommend you configure the agent from an elevated PowerShell window. If you want to configure as a service, this is **required**.

### **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}/tfs` .

When setup asks for your authentication type, choose **PAT**. Then paste the [PAT token you created](#) into the command prompt window.

#### **NOTE**

When using PAT as the authentication method, the PAT token is only used during the initial configuration of the agent.

Later, if the PAT expires or needs to be renewed, no further changes are required by the agent.

#### **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. If the PAT needs to be regenerated, no further changes are needed to 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](#).

If you choose to run as a service (which we recommend), the username you run as should be 20 characters or fewer.

# Run the agent

## Run interactively

If you configured the agent to run interactively, to run it:

```
.\run.cmd
```

To restart the agent, press **Ctrl+C** to stop the agent and then run `run.cmd` to restart it.

## Run once

For agents configured to run interactively, you can choose to have the agent accept only one job. To run in this configuration:

```
.\run.cmd --once
```

Agents in this mode will accept only one job and then spin down gracefully (useful for running in [Docker](#) on a service like Azure Container Instances).

## Run as a service

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 one of:

- "Azure Pipelines Agent (*name of your agent*)".
- "VSTS Agent (*name of your agent*)".
- "vstsagent.(*organization name*).(*name of your agent*)".

To restart the agent, right-click the entry and choose **Restart**.

### 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.

To configure an agent, it must know the URL to your organization or collection and credentials of someone authorized to set up agents. All other responses are optional. Any command-line parameter can be specified using an environment variable instead: put its name in upper case and prepend `VSTS_AGENT_INPUT_`. For example, `VSTS_AGENT_INPUT_PASSWORD` instead of specifying `--password`.

## Required options

- `--unattended` - agent setup will not prompt for information, and all settings must be provided on the command line
- `--url <url>` - URL of the server. For example: <https://dev.azure.com/myorganization> or <http://my-azure-devops-server:8080/tfs>
- `--auth <type>` - authentication type. Valid values are:
  - `pat` (Personal access token)
  - `negotiate` (Kerberos or NTLM)
  - `alt` (Basic authentication)
  - `integrated` (Windows default credentials)

## Authentication options

- If you chose `--auth pat`:
  - `--token <token>` - specifies your personal access token
- If you chose `--auth negotiate` or `--auth alt`:
  - `--userName <userName>` - specifies a Windows username in the format `domain\userName` or `userName@domain.com`
  - `--password <password>` - specifies a password

## Pool and agent names

- `--pool <pool>` - pool name for the agent to join
- `--agent <agent>` - agent name
- `--replace` - replace the agent in a pool. If another agent is listening by the same name, it will start failing with a conflict

## Agent setup

- `--work <workDirectory>` - work directory where job data is stored. Defaults to `_work` under the root of the agent directory. The work directory is owned by a given agent and should not share between multiple agents.
- `--acceptTeeEula` - accept the Team Explorer Everywhere End User License Agreement (macOS and Linux only)
- `--disableloguploads` - don't stream or send console log output to the server. Instead, you may retrieve them from the agent host's filesystem after the job completes.

## Windows-only startup

- `--runAsService` - configure the agent to run as a Windows service (requires administrator permission)
- `--runAsAutoLogon` - configure auto-logon and run the agent on startup (requires administrator permission)
- `--windowsLogonAccount <account>` - used with `--runAsService` or `--runAsAutoLogon` to specify the Windows user name in the format `domain\userName` or `userName@domain.com`
- `--windowsLogonPassword <password>` - used with `--runAsService` or `--runAsAutoLogon` to specify Windows logon password

- `--overwriteAutoLogon` - used with `--runAsAutoLogon` to overwrite the existing auto logon on the machine
- `--noRestart` - used with `--runAsAutoLogon` to stop the host from restarting after agent configuration completes

## Deployment group only

- `--deploymentGroup` - configure the agent as a deployment group agent
- `--deploymentGroupName <name>` - used with `--deploymentGroup` to specify the deployment group for the agent to join
- `-- projectName <name>` - used with `--deploymentGroup` to set the project name
- `--addDeploymentGroupTags` - used with `--deploymentGroup` to indicate that deployment group tags should be added
- `--deploymentGroupTags <tags>` - used with `--addDeploymentGroupTags` to specify the comma separated list of tags for the deployment group agent - for example "web, db"

## Environments only

- `--addvirtualmachineresourcetags` - used to indicate that environment resource tags should be added
- `--virtualmachineresourcetags <tags>` - used with `--addvirtualmachineresourcetags` to specify the comma separated list of tags for the environment resource agent - for example "web, db"

```
.\config --help
```

 always lists the latest required and optional responses.

## Diagnostics

If you're having trouble with your self-hosted agent, you can try running diagnostics. After configuring the agent:

```
.\run --diagnostics
```

This will run through a diagnostic suite that may help you troubleshoot the problem. The diagnostics feature is available starting with agent version 2.165.0.

## 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

Capabilities include all environment variables and the values that are set when the agent runs. If any of these values change while the agent is running, the agent must be restarted to pick up the new values. 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.

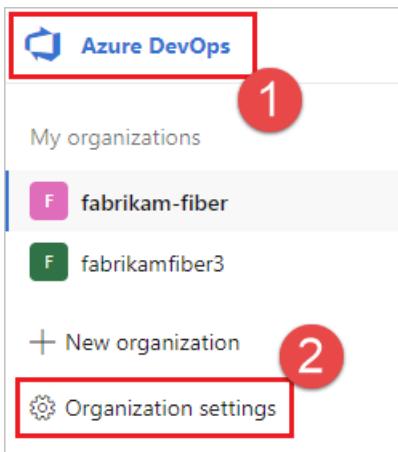
If you want to exclude environment variables as capabilities, you can designate them by setting an environment variable `VSO_AGENT_IGNORE` with a comma-delimited list of variables to ignore.

## FAQ

### How do I make sure I have the latest v2 agent version?

1. Navigate to the Agent pools tab:

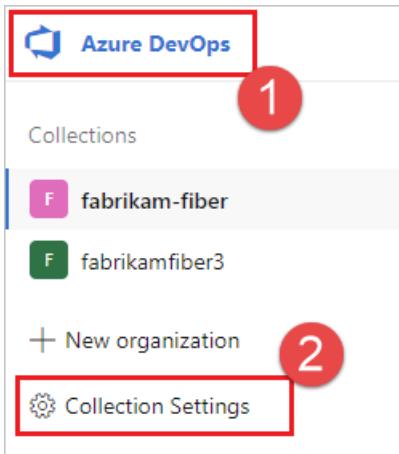
1. Choose Azure DevOps, Organization settings.



2. Choose Agent pools.

The screenshot shows the 'Agent pools' section of the organization settings. On the left, a sidebar has a red box around the 'Agent pools' button, with a red circle labeled '1' next to it. The main content area is titled 'Agent pools' and lists four entries: 'Azure Pipelines' (with a cloud icon), 'Default' (with a laptop icon), 'Test' (with a laptop icon), and 'Test2' (with a laptop icon). The URL in the browser bar is 'FabrikamFiber / Organization Settings / Agent pools'.

1. Choose Azure DevOps, Collection settings.



2. Choose Agent pools.

Azure DevOps

FabrikamFiber / Organization Settings / Agent pools

Security

Policies

Permissions

Boards

Process

Pipelines

Agent pools

Settings

Deployment pools

Parallel jobs

## Agent pools

Name	Type
Azure Pipelines	Azure Pipelines
Default	Hosted
Test	Hosted
Test2	Hosted

1. Choose Azure DevOps, Collection settings.

Azure DevOps

Collections

fabrikam-fiber

fabrikamfiber3

+ New organization

Collection Settings

2. Choose Agent pools.

Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards  
Process

Pipelines

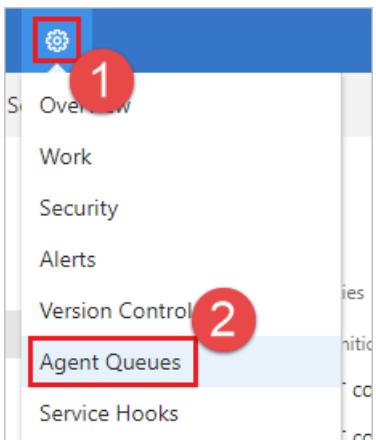
Agent pools **1**

Settings  
Deployment pools  
Parallel jobs

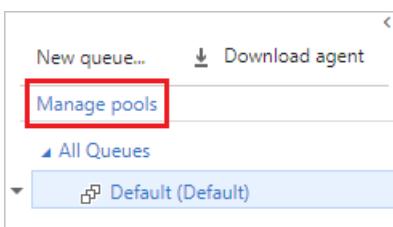
## Agent pools

Name
Azure Pipelines
Default
Test
Test2

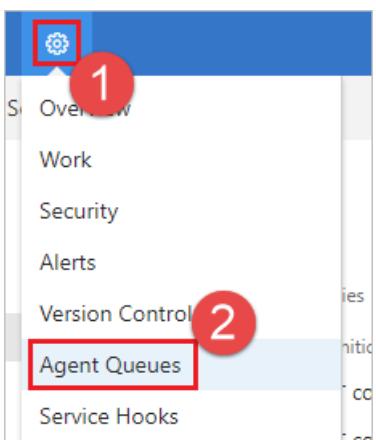
1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



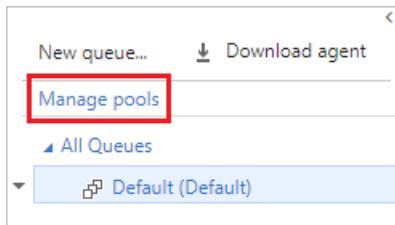
2. Choose **Manage pools**.



1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



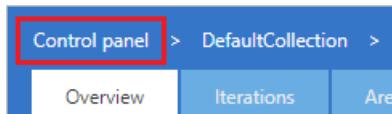
2. Choose **Manage pools**.



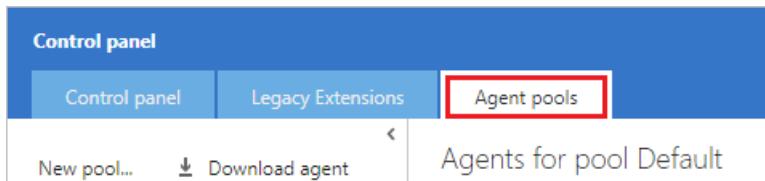
1. Navigate to your project and choose **Manage project** (gear icon).



2. Choose **Control panel**.



3. Select **Agent pools**.



2. Click the pool that contains the agent.

3. Make sure the agent is enabled.

4. Navigate to the capabilities tab:

1. From the **Agent pools** tab, select the desired agent pool.

A screenshot of the 'Agent pools' page in Azure DevOps. The left sidebar shows 'Pipelines' (highlighted with a red box) and 'Agent pools' (highlighted with a red box, labeled with a red circle '1'). The main area lists agent pools: 'Azure Pipelines' (disabled), 'Default' (selected and highlighted with a red box, labeled with a red circle '2'), 'Test', and 'Test2'. The URL in the browser is 'FabrikamFiber / Organization Settings / Agent pools'.

2. Select **Agents** and choose the desired agent.

The screenshot shows the 'Agents' tab selected in the top navigation bar. Below the tabs, a list of agents is displayed. The first agent, '160724-AT14-SP3', is highlighted with a red box and labeled '2'. This agent is marked as 'Offline'. Another agent, '20160317-W10', is listed below it, also marked as 'Offline'.

3. Choose the **Capabilities** tab.

The screenshot shows the 'Capabilities' tab selected for the agent '160724-AT14-SP3'. The 'User-defined capabilities' section is currently empty, showing a '+ Add new capability' button. The 'System capabilities' section displays the following table:

Name	Value
Agent.Name	160724-AT14-SP3
Agent.Version	2.107.0
Agent.ComputerName	160724-AT14-SP3

**NOTE**

Microsoft-hosted agents don't display system capabilities. For a list of software installed on Microsoft-hosted agents, see [Use a Microsoft-hosted agent](#).

1. From the **Agent pools** tab, select the desired pool.

Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards

Process

Pipelines  
1

Agent pools  
2

Settings

Deployment pools

Parallel jobs

## Agent pools

Name
Azure Pipelines
Default
Test
Test2

2. Select **Agents** and choose the desired agent.

Default

Jobs Agents Details Security Settings  
1

Name

160724-AT14-SP3 ● Offline
20160317-W10 ● Offline

2

3. Choose the **Capabilities** tab.

160724-AT14-SP3

Jobs Capabilities

User-defined capabilities



No user-defined capabilities

Add a new capability

System capabilities

Search by keyword

Name	Value
Agent.Name	160724-AT14-SP3
Agent.Version	2.107.0
Agent.ComputerName	160724-AT14-SP3

- From the Agent pools tab, select the desired pool.

The screenshot shows the Azure DevOps interface for managing agent pools. On the left, there's a sidebar with navigation links: Security, Policies, Permissions, Boards, Process, Pipelines (with a red box around it and a red number 1), Agent pools (highlighted with a red box and a red number 1), Settings, Deployment pools, and Parallel jobs. The main content area is titled "Agent pools". It lists three pools: "Azure Pipelines" (highlighted with a red box and a red number 2), "Default" (highlighted with a red box and a red number 2), and "Test".

- Select Agents and choose the desired agent.

Default

Jobs Agents Details Security Settings

Name
160724-AT14-SP3 ● Offline
20160317-W10 ● Offline

3. Choose the **Capabilities** tab.

160724-AT14-SP3

Jobs Capabilities

User-defined capabilities

No user-defined capabilities

Add a new capability

System capabilities

Name	Value
Agent.Name	160724-AT14-SP3
Agent.Version	2.107.0
Agent.ComputerName	160724-AT14-SP3

Select the desired agent, and choose the **Capabilities** tab.

Agents for pool Default

Agents Roles

Enabled	Name	Current Status	X
<input checked="" type="checkbox"/>	20160317-W10	Idle	X
<input checked="" type="checkbox"/>	DEV15-TFS-RTM	Idle	X
<input checked="" type="checkbox"/>	DEV15VS	Idle	X

Requests Capabilities

USER CAPABILITIES

Shows information about user-defined capabilities supported by this host

+ Add capability

Save changes Undo changes

SYSTEM CAPABILITIES

Shows information about the capabilities provided by this host

AegisRoot	C:\AegisTools
Agent.ComputerName	20160317-W10

Select the desired agent, and choose the **Capabilities** tab.

Agents for pool Default

Enabled	Name	Current Status	X
<input checked="" type="checkbox"/>	20160317-W10	Idle	X
<input checked="" type="checkbox"/>	DEV15-TFS-RTM	Idle	X
<input checked="" type="checkbox"/>	DEV15VS	Idle	X

Requests Capabilities

**USER CAPABILITIES**  
Shows information about user-defined capabilities supported by this host

+ Add capability

Save changes Undo changes

**SYSTEM CAPABILITIES**  
Shows information about the capabilities provided by this host

AegisRoot	C:\AegisTools
Agent.ComputerName	20160317-W10

From the Agent pools tab, select the desired agent, and choose the Capabilities tab.

Control panel

Control panel Legacy Extensions Agent pools

New pool... Download agent

All Pools Default

Enabled	Name	Current Status	X
<input checked="" type="checkbox"/>	Agent-TFSDEV14UP...	Idle	X

Requests Capabilities

**USER CAPABILITIES**  
Shows information about user-defined capabilities supported by this host

+ Add capability

Save changes Undo changes

**SYSTEM CAPABILITIES**  
Shows information about the capabilities provided by this host

5. 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.
6. Each agent automatically updates itself when it runs a task that requires a newer version of the agent. If you want to manually update some agents, right-click the pool, and select **Update all agents**.

#### Can I update my v2 agents that are part of an Azure DevOps Server pool?

Yes. Beginning with Azure DevOps Server 2019, you can configure your server to look for the agent package files on a local disk. This configuration will override the default version that came with the server at the time of its release. This scenario also applies when the server doesn't 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 by using a method of your choice (such as USB drive, Network transfer, and so on). Place the agent files under the `%ProgramData%\Microsoft\Azure DevOps\Agents` folder.
3. You're all set! Your Azure DevOps Server will now use the local files whenever the agents are 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}}.vstmr.visualstudio.com
https://{{organization_name}}.pkgs.visualstudio.com
https://{{organization_name}}.vsblob.visualstudio.net
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
https://vstsagentpackage.azureedge.net
https://*.vsblob.visualstudio.net
https://*.vssps.visualstudio.com
```

To ensure your organization works with any existing firewall or IP restrictions, ensure that `dev.azure.com` and `*dev.azure.com` are open and update your allow-listed IPs to include the following IP addresses, based on your IP version. If you're currently allow-listing the `13.107.6.183` and `13.107.9.183` IP addresses, leave them in place, as you don't need to remove them.

### IPv4 ranges

- `13.107.6.0/24`
- `13.107.9.0/24`
- `13.107.42.0/24`
- `13.107.43.0/24`

### IPv6 ranges

- `2620:1ec:4::/48`
- `2620:1ec:a92::/48`
- `2620:1ec:21::/48`
- `2620:1ec:22::/48`

#### NOTE

For more information about allowed addresses, see [Allowed address lists and network connections](#).

## 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 restart the agent

If you are running the agent interactively, see the restart instructions in [Run interactively](#). If you are running the agent as a service, restart the agent by following the steps in [Run as a service](#).

### 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}}.vstmr.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
https://vstsagentpackage.azureedge.net
```

To ensure your organization works with any existing firewall or IP restrictions, ensure that `dev.azure.com` and `*dev.azure.com` are open and update your allow-listed IPs to include the following IP addresses, based on your IP version. If you're currently allow-listing the `13.107.6.183` and `13.107.9.183` IP addresses, leave them in place, as you don't need to remove them.

#### IPv4 ranges

- `13.107.6.0/24`
- `13.107.9.0/24`
- `13.107.42.0/24`
- `13.107.43.0/24`

#### IPv6 ranges

- `2620:1ec:4::/48`

- 2620:1ec:a92::/48
- 2620:1ec:21::/48
- 2620:1ec:22::/48

#### 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

11/2/2020 • 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 [Azure Pipelines 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**

Capabilities include all environment variables and the values that are set when the agent runs. If any of these values change while the agent is running, the agent must be restarted to pick up the new values. 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.

If you want to exclude environment variables as capabilities, you can designate them by setting an environment variable `VSO_AGENT_IGNORE` with a comma-delimited list of variables to ignore.

# FAQ

## **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 5: [Windows Management Framework 5.0](#)

## **Why do I get this message when I try to queue my build?**

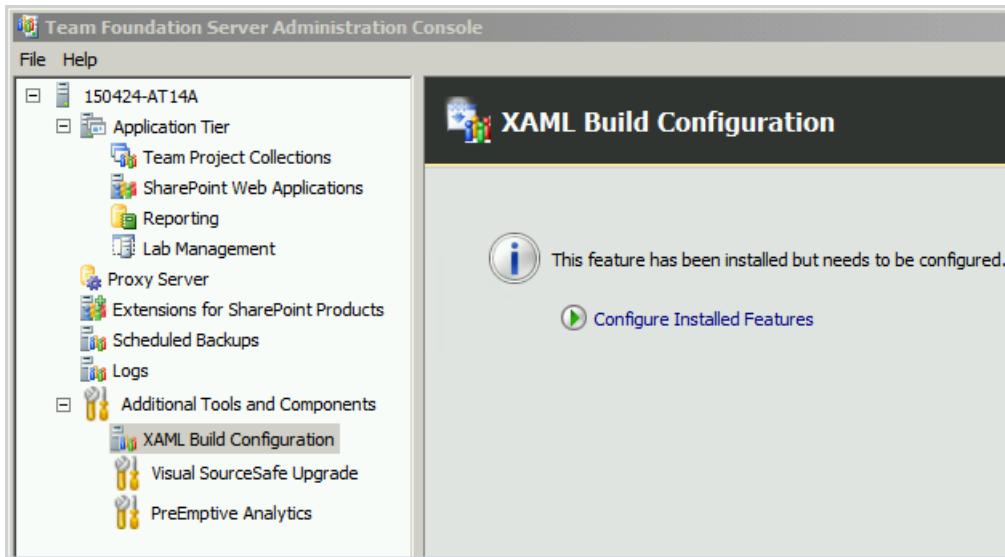
When you try to queue a build or a deployment, you might get a message such as: "No agent could be found with the following capabilities: msbuild, visualstudio, vstest." One common cause of this problem is that you need to install prerequisite software such as Visual Studio on the machine where the build 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](#).

# Azure virtual machine scale set agents

7/1/2021 • 21 minutes to read • [Edit Online](#)

## Azure Pipelines

Azure virtual machine scale set agents, hereafter referred to as scale set agents, are a form of self-hosted agents that can be autoscaled to meet your demands. This elasticity reduces your need to run dedicated agents all the time. Unlike Microsoft-hosted agents, you have flexibility over the size and the image of machines on which agents run.

If you like Microsoft-hosted agents but are limited by what they offer, you should consider scale set agents. Here are some examples:

- You need more memory, more processor, more storage, or more IO than what we offer in native Microsoft-hosted agents.
- You need NCv2 VM with particular instruction sets for machine learning.
- You need to deploy to a private Azure App Service in a private VNET with no inbound connectivity.
- You need to open corporate firewall to specific IP addresses so that Microsoft-hosted agents can communicate with your servers.
- You need to restrict network connectivity of agent machines and allow them to reach only approved sites.
- You can't get enough agents from Microsoft to meet your needs.
- Your jobs exceed the Microsoft-hosted agent timeout.
- You can't partition Microsoft-hosted parallel jobs to individual projects or teams in your organization.
- You want to run several consecutive jobs on an agent to take advantage of incremental source and machine-level package caches.
- You want to run additional configuration or cache warmup before an agent begins accepting jobs.

If you like self-hosted agents but wish that you could simplify managing them, you should consider scale set agents. Here are some examples:

- You don't want to run dedicated agents around the clock. You want to de-provision agent machines that are not being used to run jobs.
- You run untrusted code in your pipeline and want to re-image agent machines after each job.
- You want to simplify periodically updating the base image for your agents.

### NOTE

You cannot run Mac agents using scale sets. You can only run Windows or Linux agents this way.

## Create the scale set

In preparation for creating scale set agents, you must first create a virtual machine scale set in the Azure portal. You must create the virtual machine scale set in a certain way so that Azure Pipelines can manage it. In particular, you must disable Azure's autoscaling so that Azure Pipelines can determine how to perform scaling based on number of incoming pipeline jobs. We recommend that you use the following steps to create the scale set.

In the following example, a new resource group and virtual machine scale set are created with Azure Cloud Shell using the UbuntuLTS VM image.

## NOTE

In this example, the UbuntuLTS VM image is used for the scale set. If you require a customized VM image as the basis for your agent, create the customized image before creating the scale set, by following the steps in [Create a scale set with custom image, software, or disk size](#).

1. Browse to [Azure Cloud Shell](#) at <https://shell.azure.com/>.

2. Run the following command to verify your default Azure subscription.

```
az account list -o table
```

If your desired subscription isn't listed as the default, select your desired subscription.

```
az account set -s <your subscription ID>
```

3. Create a resource group for your virtual machine scale set.

```
az group create \
--location westus \
--name vmssagents
```

4. Create a virtual machine scale set in your resource group. In this example the UbuntuLTS VM image is specified.

```
az vmss create \
--name vmssagentspool \
--resource-group vmssagents \
--image UbuntuLTS \
--vm-sku Standard_D2_v3 \
--storage-sku StandardSSD_LRS \
--authentication-type SSH \
--instance-count 2 \
--disable-overprovision \
--upgrade-policy-mode manual \
--single-placement-group false \
--platform-fault-domain-count 1 \
--load-balancer ""
```

Because Azure Pipelines manages the scale set, the following settings are required or recommended:

- `--disable-overprovision` - required
- `--upgrade-policy-mode manual` - required
- `--load-balancer ""` - Azure Pipelines doesn't require a load balancer to route jobs to the agents in the scale set agent pool, but configuring a load balancer is one way to get an IP address for your scale set agents that you could use for firewall rules. Another option for getting an IP address for your scale set agents is to create your scale set using the `--public-ip-address` options. For more information about configuring your scale set with a load balancer or public IP address, see the [Virtual Machine Scale Sets documentation](#) and `az vmss create`.
- `--instance-count 2` - this setting is not required, but it will give you an opportunity to verify that the scale set is fully functional before you create an agent pool. Creation of the two VMs can take several minutes. Later, when you create the agent pool, Azure Pipelines will delete these two VMs and create new ones.

#### IMPORTANT

If you run this script using Azure CLI on Windows, you must enclose the `""` in `--load-balancer ""` with single quotes like this: `--load-balancer '""'`

If your VM size supports [Ephemeral OS disks](#), the following parameters to enable Ephemeral OS disks are optional but recommended to improve virtual machine reimage times.

- `--ephemeral-os-disk true`
- `--os-disk-caching readonly`

#### IMPORTANT

Ephemeral OS disks are not supported on all VM sizes. For list of supported VM sizes, see [Ephemeral OS disks for Azure VMs](#).

Select any Linux or Windows image - either from Azure Marketplace or your own custom image - to create the scale set. Do not pre-install Azure Pipelines agent in the image. Azure Pipelines automatically installs the agent as it provisions new virtual machines. In the above example, we used a plain `UbuntuLTS` image. For instructions on creating and using a custom image, see [FAQ](#).

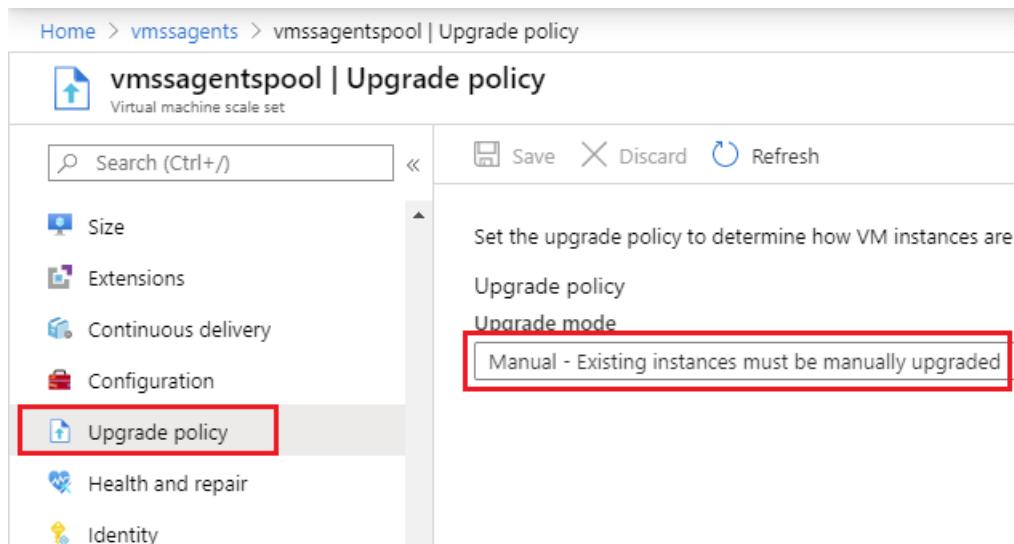
Select any VM SKU and storage SKU.

#### NOTE

Licensing considerations limit us from distributing Microsoft-hosted images. We are unable to provide these images for you to use in your scale set agents. But, the [scripts](#) that we use to generate these images are open source. You are free to use these scripts and create your own custom images.

5. After creating your scale set, navigate to your scale set in the Azure portal and verify the following settings:

- **Upgrade policy - Manual**



You can also verify this setting by running the following Azure CLI command.

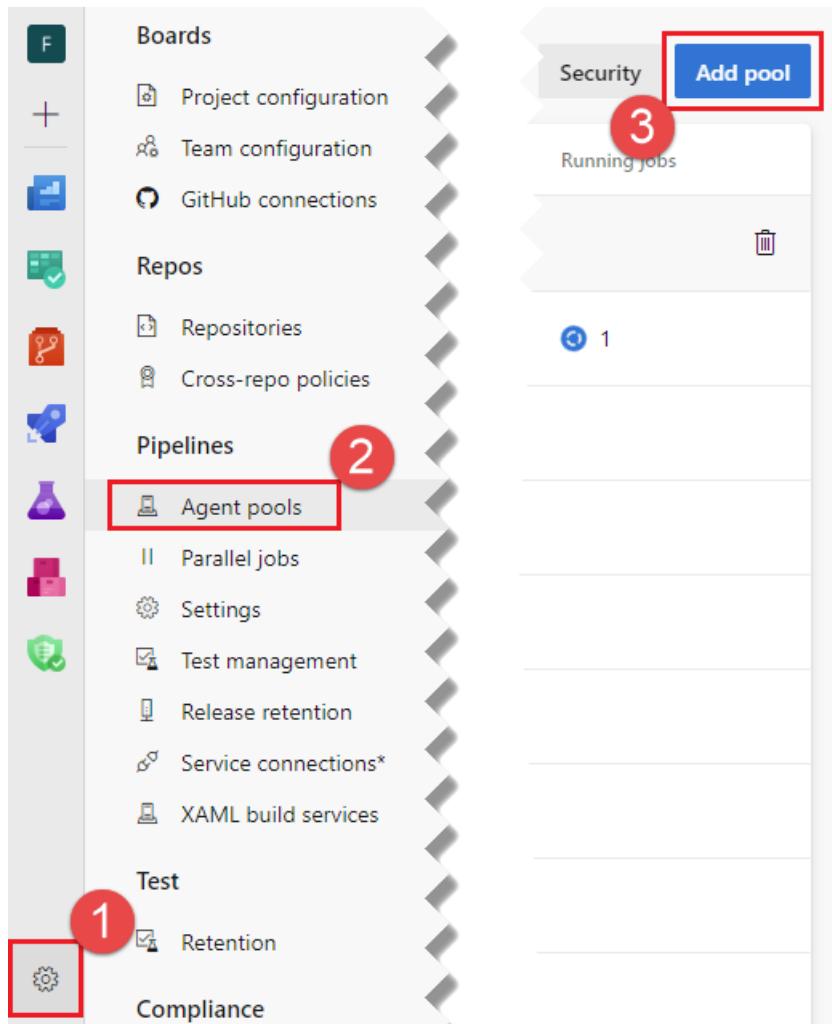
az vmss show --resource-group vmssagents --name vmssagentspool --output table						
Name	ResourceGroup	Location	Zones	Capacity	Overprovision	
UpgradePolicy						
vmssagentspool	vmssagents	westus		0	False	Manual

- Scaling - Manual scale

The screenshot shows the Azure portal interface for managing a Virtual Machine Scale Set (VMSS). The URL in the address bar is [Home > vmssagents > vmssagentspool | Scaling](#). The main title is "vmssagentspool | Scaling". On the left, there's a sidebar with various configuration tabs: Instances, Networking, Scaling (which is selected and highlighted with a red box), Storage, Operating system, Security, Size, Extensions, Continuous delivery, Configuration, Upgrade policy, Health and repair, Identity, and Properties. The main content area has a "Configure" tab selected. A section titled "Choose how to scale your resource" contains two options: "Manual scale" (selected) and "Custom autoscale". The "Manual scale" section includes a sub-section for "Override condition" and a slider for "Instance count". Below the "Configure" tab, there are links for "Save", "Discard", "Refresh", and "Provide feedback".

## Create the scale set agent pool

1. Navigate to your Azure DevOps Project settings, select Agent pools under Pipelines, and select Add pool to create a new agent pool.



#### IMPORTANT

You may create your scale set pool in [Project settings](#) or [Organization settings](#), but when you delete a scale set pool, you must delete it from [Organization settings](#), and not [Project settings](#).

2. Select **Azure virtual machine scale set** for the pool type. Select the **Azure subscription** that contains the scale set, choose **Authorize**, and choose the desired virtual machine scale set from that subscription. If you have an existing [service connection](#) you can choose that from the list instead of the subscription.

#### IMPORTANT

- To configure a scale set agent pool, you must have either [Owner](#) or [User Access Administrator](#) permissions on the selected subscription. If you have one of these permissions but get an error when you choose **Authorize**, see [troubleshooting](#).
- The only service connection currently supported is an Azure Resource Manager (ARM) service connection based on a service principal key. ARM service connections based on a certificate credential or a Managed Identity will fail. When you attempt to list the existing scale sets in your subscription, you'll see an error like this:

```
Invalid Service Endpoint with Id <guid> and Scope <guid>
```

3. Choose the desired virtual machine scale set from that subscription.
4. Specify a name for your agent pool.
5. Configure the following options:

- **Automatically tear down virtual machines after every use** - A new VM instance is used for every job. After running a job, the VM will go offline and be reimaged before it picks up another job.
- **Save an unhealthy agent for investigation** - Whether to save [unhealthy agent VMs](#) for troubleshooting instead of deleting them.
- **Maximum number of virtual machines in the scale set** - Azure Pipelines will automatically scale out the number of agents, but won't exceed this limit.
- **Number of agents to keep on standby** - Azure Pipelines will automatically scale in the number of agents, but will ensure that there are always this many agents available to run new jobs. If you set this to 0, for example to conserve cost for a low volume of jobs, Azure Pipelines will start a VM only when it has a job.
- **Delay in minutes before deleting excess idle agents** - To account for the variability in build load throughout the day, Azure Pipelines will wait this long before deleting an excess idle agent.
- **Configure VMs to run interactive tests (Windows Server OS Only)** - Windows agents can either be configured to run unelevated with autologon and with interactive UI, or they can be configured to run with elevated permissions. Check this box to run unelevated with interactive UI. In either case, the agent user is a member of the Administrators group.

6. When your settings are configured, choose **Create** to create the agent pool.

## Use scale set agent pool

Using a scale set agent pool is similar to any other agent pool. You can use it in classic build, release, or YAML pipelines. User permissions, pipeline permissions, approvals, and other checks work the same way as in any other agent pool. For more information, see [Agent pools](#).

### IMPORTANT

Caution must be exercised when making changes directly to the scale set in the Azure portal.

- You may not change many of the the scale set configuration settings in the Azure portal. Azure Pipelines updates the configuration of the scale set. Any manual changes you make to the scale set may interfere with the operation of Azure Pipelines.
- You may not rename or delete a scale set without first deleting the scale set pool in Azure Pipelines.

## How Azure Pipelines manages the scale set

Once the scale set agent pool is created, Azure Pipelines automatically scales the agent machines.

Azure Pipelines samples the state of the agents in the pool and virtual machines in the scale set every 5 minutes. The decision to scale in or out is based on the number of idle agents at that time. An agent is considered idle if it is online and is not running a pipeline job. Azure Pipelines performs a scale out operation if either of the following conditions is satisfied:

- The number of idle agents falls below the number of standby agents you specify
- There are no idle agents to service pipeline jobs waiting in the queue

If one of these conditions is met, Azure Pipelines grows the number of VMs. Scaling out is done in increments of a certain percentage of the maximum pool size. Allow 20 minutes for machines to be created for each step.

Azure Pipelines scales in the agents when the number of idle agents exceeds the standby count for more than 30 minutes (configurable using **Delay in minutes before deleting excess idle agents**).

To put all of this into an example, consider a scale set agent pool that is configured with 2 standby agents and 4 maximum agents. Let us say that you want to tear down the VM after each use. Also, let us assume that there are no VMs to start with in the scale set.

- Since the number of idle agents is 0, and since the number of idle agents is below the standby count of 2, Azure Pipelines scales out and adds two VMs to the scale set. Once these agents come online, there will be 2 idle agents.
- Let us say that 1 pipeline job arrives and is allocated to one of the agents.
- At this time, the number of idle agents is 1, and that is less than the standby count of 2. So, Azure Pipelines scales out and adds 2 more VMs (the increment size used in this example). At this time, the pool has 3 idle agents and 1 busy agent.
- Let us say that the job on the first agent completes. Azure Pipeline takes that agent offline to re-image that machine. After a few minutes, it comes back with a fresh image. At this time, we'll have 4 idle agents.
- If no other jobs arrive for 30 minutes (configurable using **Delay in minutes before deleting excess idle agents**), Azure Pipelines determines that there are more idle agents than are necessary. So, it scales in the pool to two agents.

Throughout this operation, the goal for Azure Pipelines is to reach the desired number of idle agents on standby. Pools scale out and in slowly. Over the course of a day, the pool will scale out as requests are queued in the morning and scale in as the load subsides in the evening. You may observe more idle agents than you desire at various times. This is expected as Azure Pipelines converges gradually to the constraints that you specify.

#### **NOTE**

It can take an hour or more for Azure Pipelines to scale out or scale in the virtual machines. Azure Pipelines will scale out in steps, monitor the operations for errors, and react by deleting unusable machines and by creating new ones in the course of time. This corrective operation can take over an hour.

To achieve maximum stability, scale set operations are done sequentially. For example if the pool needs to scale out and there are also unhealthy machines to delete, Azure Pipelines will first scale out the pool. Once the pool has scaled out to reach the desired number of idle agents on standby, the unhealthy machines will be deleted, depending on the **Save an unhealthy agent for investigation** setting. For more information, see [Unhealthy agents](#).

Due to the sampling size of 5 minutes, it is possible that all agents can be running pipelines for a short period of time and no scaling out will occur.

## Customizing Pipeline Agent Configuration

You can customize the configuration of the Azure Pipeline Agent by defining environment variables in your operating system custom image for your scale set. For example, the scale set agent working directory defaults to C:\a for Windows and /agent/\_work for Linux. If you want to change the working directory, set an environment variable named VSTS\_AGENT\_INPUT\_WORK with the desired working directory. More information can be found in the [Pipelines Agent Unattended Configuration](#) documentation. Some examples include:

- `VSTS_AGENT_INPUT_WORK`
- `VSTS_AGENT_INPUT_PROXYURL`
- `VSTS_AGENT_INPUT_PROXYUSERNAME`
- `VSTS_AGENT_INPUT_PROXYPASSWORD`

## IMPORTANT

Caution must be exercised when customizing the Pipelines agent. Some settings will conflict with other required settings, causing the agent to fail to register, and the VM to be deleted. These settings should not be set or altered:

- `VSTS_AGENT_INPUT_URL`
- `VSTS_AGENT_INPUT_AUTH`
- `VSTS_AGENT_INPUT_TOKEN`
- `VSTS_AGENT_INPUT_USERNAME`
- `VSTS_AGENT_INPUT_PASSWORD`
- `VSTS_AGENT_INPUT_POOL`
- `VSTS_AGENT_INPUT_AGENT`
- `VSTS_AGENT_INPUT_RUNASSERVICE`
- ... and anything related to Deployment Groups.

## Customizing Virtual Machine Startup via the Custom Script Extension

Users may want to execute startup scripts on their scaleset agent machines before those machines start running pipeline jobs. Some common use cases for startup scripts include installing software, warming caches, or fetching repos. You can execute startup scripts by installing the [Custom Script Extension for Windows](#) or [Custom Script Extension for Linux](#). This extension will be executed on every virtual machine in the scaleset immediately after it is created or reimaged. The custom script extension will be executed before the Azure Pipelines agent extension is executed.

Here is an example to create a custom script extension for Linux.

```
az vmss extension set \
--vmss-name <scaleset name> \
--resource-group <resource group> \
--name CustomScript \
--version 2.0 \
--publisher Microsoft.Azure.Extensions \
--settings '{ \"fileUris\":[\"https://<myGitHubRepoUrl>/myScript.sh\"], \"commandToExecute\": \"bash \
./myScript.sh /myArgs \" }'
```

Here is an example to create a custom script extension for Windows.

```
az vmss extension set \
--vmss-name <scaleset name> \
--resource-group <resource group> \
--name CustomScriptExtension \
--version 1.9 \
--publisher Microsoft.Compute \
--settings '{ \"FileUris\":[\"https://<myGitHubRepoUrl>/myscript.ps1\"], \"commandToExecute\": \
\"Powershell.exe -ExecutionPolicy Unrestricted -File myscript.ps1 -myargs 0 \" }'
```

## IMPORTANT

The scripts executed in the Custom Script Extension must return with exit code 0 in order for the VM to finish the VM creation process. If the custom script extension throws an exception or returns a non-zero exit code, the Azure Pipeline extension will not be executed and the VM will not register with Azure DevOps agent pool.

## Lifecycle of a Scale Set Agent

Here is the flow of operations for an Azure Pipelines Virtual Machine Scale Set Agent

1. The Azure DevOps Scale Set Agent Pool sizing job determines the pool has too few idle agents and needs to scale out. Azure Pipelines makes a call to Azure Scale Sets to increase the scale set capacity.
2. The Azure Scale Set begins creating the new virtual machines. Once the virtual machines are running, Azure Scale Sets sequentially executes any installed VM extensions.
3. If the Custom Script Extension is installed, it is executed before the Azure Pipelines Agent extension. If the Custom Script Extension returns a non-zero exit code, the VM creation process is aborted and will be deleted.
4. The Azure Pipelines Agent extension is executed. This extension downloads the latest version of the Azure Pipelines Agent along with a configuration script which can be found here.
  - <https://vstsagenttools.blob.core.windows.net/tools/ElasticPools/Linux/6/enableagent.sh>
  - <https://vstsagenttools.blob.core.windows.net/tools/ElasticPools/Windows/6/enableagent.ps1>

**NOTE**

These URLs may change.

5. The configuration script creates a local user named `AzDevops` if the operating system is Windows Server or Linux. For Windows 10 Client OS, the agent runs as LocalSystem. The script then unzips, installs, and configures the Azure Pipelines Agent. As part of configuration, the agent registers with the Azure DevOps agent pool and appears in the agent pool list in the Offline state.
6. For most scenarios, the configuration script then immediately starts the agent to run as the local user `AzDevops`. The agent goes Online and is ready to run pipeline jobs.

If the pool is configured for interactive UI, the virtual machine reboots after the agent is configured. After reboot, the local user will auto-login and immediately start the pipelines agent. The agent then goes Online and is ready to run pipeline jobs.

## Create a scale set with custom image, software, or disk size

If you just want to create a scale set with the default 128 GB OS disk using a publicly available Azure image, then skip straight to step 10 and use the public image name (UbuntuLTS, Win2019DataCenter, etc.) to create the scale set. Otherwise follow these steps to customize your VM image.

1. Create a VM with your desired OS image and optionally expand the OS disk size from 128 GB to

`<myDiskSizeGb>`.

- If starting with an available Azure Image, for example = (Win2019DataCenter, UbuntuLTS):

```
az vm create --resource-group <myResourceGroup> --name <MyVM> --image <myBaseImage> --os-disk-size-gb <myDiskSize> --admin-username myUserName --admin-password myPassword
```

- If starting with a generalized VHD:

- a. First create the VM with an unmanaged disk of the desired size and then convert to a managed disk:

```
az vm create --resource-group <myResourceGroup> --name <MyVM> --image <myVhdUrl> --os-type windows --os-disk-size-gb <myDiskSizeGb> --use-unmanaged-disk --admin-username <myUserName> --admin-password <myPassword> --storage-account <myVhdStorageAccount>
```

b. Shut down the VM

```
az vm stop --resource-group <myResourceGroup> --name <MyVM>
```

c. Deallocate the VM

```
az vm deallocate --resource-group <myResourceGroup> --name <MyVM>
```

d. Convert to a managed disk

```
az vm convert --resource-group <myResourceGroup> --name <MyVM>
```

e. Restart the VM

```
az vm start --resource-group <myResourceGroup> --name <MyVM>
```

2. Remote Desktop (or SSH) to the VM's public IP address to customize the image. You may need to open ports in the firewall to unblock the RDP (3389) or SSH (22) ports.

- a. **Windows** - If `<MyDiskSizeGb>` is greater than 128 GB, extend the OS disk size to fill the disk size you declared above.

Open DiskPart tool as administrator and run these DiskPart commands:

- a. `list volume` (to see the volumes)
- b. `select volume 2` (depends on which volume is the OS drive)
- c. `extend size 72000` (to extend the drive by 72 GB, from 128 GB to 200 GB)

3. Install any additional software on the VM.

4. To customize the permissions of the pipeline agent user, you can create a user named `AzDevOps`, and grant that user the permissions you require. This user will be created by the scaleset agent startup script if it does not already exist.

5. Reboot the VM when finished with customizations

6. Generalize the VM.

- **Windows** - From an admin console window:

```
C:\Windows\System32\sysprep\sysprep.exe /generalize /oobe /shutdown
```

- **Linux:**

```
sudo waagent -deprovision-user -force
```

#### IMPORTANT

Wait for the VM to finish generalization and shutdown. Do not proceed until the VM has stopped. Allow 60 minutes.

7. Deallocate the VM

```
az vm deallocate --resource-group <myResourceGroup> --name <MyVM>
```

## 8. Mark the VM as Generalized

```
az vm generalize --resource-group <myResourceGroup> --name <MyVM>
```

## 9. Create a VM Image based on the generalized image. When performing these steps to update an existing scaleset image, make note of the image ID url in the output.

```
az image create --resource-group <myResourceGroup> --name <MyImage> --source <MyVM>
```

## 10. Create the scale set based on the custom VM image

```
az vmss create --resource-group <myResourceGroup> --name <myScaleSet> --image <MyImage> --admin-username <myUsername> --admin-password <myPassword> --instance-count 2 --disable-overprovision --upgrade-policy-mode manual --load-balancer """"
```

## 11. Verify that both VMs created in the scale set come online, have different names, and reach the Succeeded state

You are now ready to create an agent pool using this scale set.

## Update an existing scale set with a new custom image

To update the image on an existing scaleset, follow the steps in the previous [Create a scale set with custom image, software, or disk size](#) section up through the `az image create` step to generate the custom OS image. Make note of the ID property URL that is output from the `az image create` command. Then update the scaleset with the new image as shown in the following example. After the scaleset image has been updated, all future VMs in the scaleset will be created with the new image.

```
az vmss update --resource-group <myResourceGroup> --name <myScaleSet> --set virtualMachineProfile.storageProfile.imageReference.id=<id url>
```

## Supported Operating Systems

Scale set agents currently supports Ubuntu Linux, Windows Server/DataCenter 2016/2019, and Windows 10 client.

### Known issues

- Debian or RedHat Linux are not supported. Only Ubuntu is.
- Windows 10 client does not support running the pipeline agent as a local user and therefore the agent cannot interact with the UI. The agent will run as Local Service instead.

## Troubleshooting issues

Navigate to your Azure DevOps Project settings, select Agent pools under Pipelines, and select your agent pool. Click the tab labeled Diagnostics.

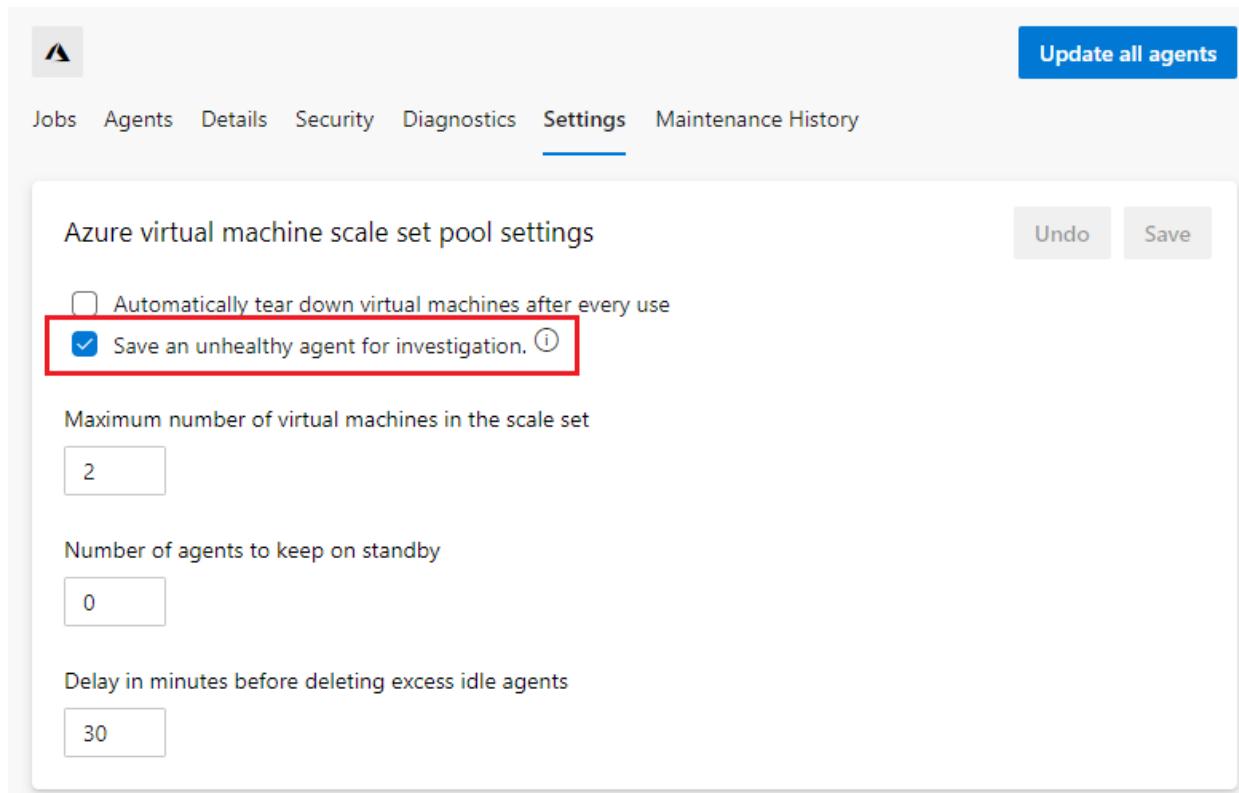
The Diagnostic tab shows all actions executed by Azure DevOps to Create, Delete, or Reimage VMs in your Azure Scale Set. Diagnostics also logs any errors encountered while trying to perform these actions. Review the errors to make sure your scaleset has sufficient resources to scale out. If your Azure subscription has reached the

resource limit in VMs, CPU cores, disks, or IP Addresses, those errors will show up here.

## Unhealthy Agents

When agents or virtual machines are failing to start, not connecting to Azure DevOps, or going offline unexpectedly, Azure DevOps logs the failures to the Agent Pool's **Diagnostics** tab and tries to delete the associated virtual machine. Networking configuration, image customization, and pending reboots can cause these issues. Connecting to the VM to debug and gather logs can help with the investigation.

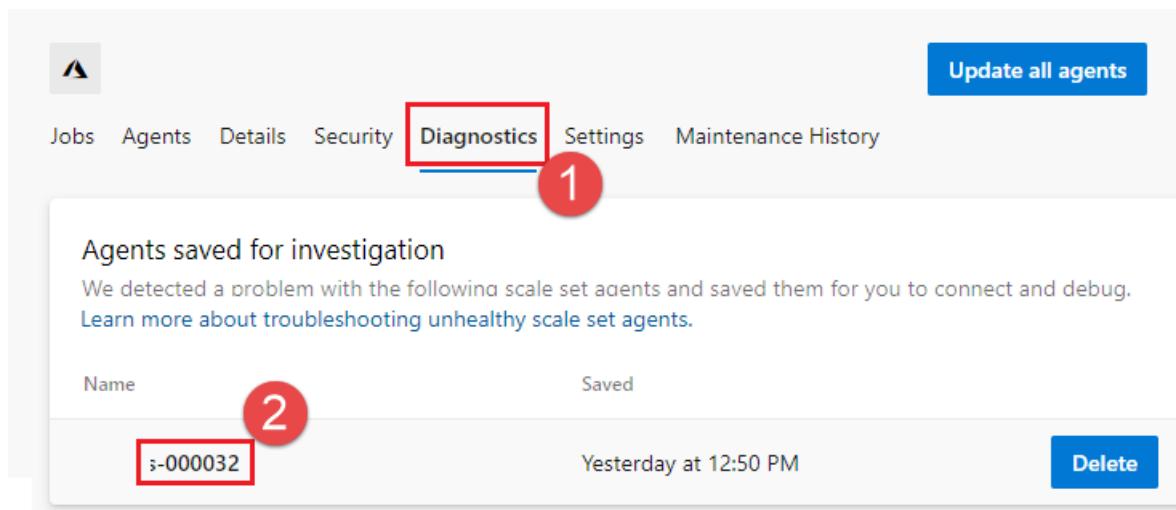
If you would like Azure DevOps to save an unhealthy agent VM for investigation and not automatically delete it when it detects the unhealthy state, navigate to your Azure DevOps **Project settings**, select **Agent pools** under **Pipelines**, and select your agent pool. Choose **Settings**, select the option **Save an unhealthy agent for investigation**, and choose **Save**.



The screenshot shows the 'Agent pool Settings' page for an 'Azure virtual machine scale set pool'. At the top, there are tabs for Jobs, Agents, Details, Security, Diagnostics, Settings (which is selected), and Maintenance History. A blue 'Update all agents' button is at the top right. Below the tabs, the page title is 'Azure virtual machine scale set pool settings'. There are two buttons at the top right: 'Undo' and 'Save'. Under the title, there are several configuration options:

- A checkbox for 'Automatically tear down virtual machines after every use' which is unchecked.
- A checked checkbox for 'Save an unhealthy agent for investigation.' with an information icon next to it. This checkbox is highlighted with a red border.
- A text input field for 'Maximum number of virtual machines in the scale set' containing the value '2'.
- A text input field for 'Number of agents to keep on standby' containing the value '0'.
- A text input field for 'Delay in minutes before deleting excess idle agents' containing the value '30'.

Now, when an unhealthy agent is detected in the scale set, Azure DevOps saves that agent and associated virtual machine. The saved agent will be visible on the **Diagnostics** tab of the Agent pool UI. Navigate to your Azure DevOps **Project settings**, select **Agent pools** under **Pipelines**, select your agent pool, choose **Diagnostics**, and make note of the agent name.



The screenshot shows the 'Diagnostics' tab of the 'Agent pool Settings' page for the same 'Azure virtual machine scale set pool'. A red circle with the number '1' highlights the 'Diagnostics' tab itself. Another red circle with the number '2' highlights the 'Name' column in the table below. The table lists a single agent:

Name	Saved
s-000032	Yesterday at 12:50 PM

At the bottom right of the table is a blue 'Delete' button.

Find the associated virtual machine in your Azure virtual machine scale set via the Azure portal, in the

## Instances list.

Home >

Instances

Virtual machine scale set | Directory: Microsoft

Search (Ctrl+ /) Start Restart Stop Reimage Delete Upgrade Refresh

Overview Activity log Access control (IAM) Tags Diagnose and solve problems

Settings Instances

Name	Computer name	Status
_110	:000032	Running

Select the instance, choose **Connect**, and perform your investigation.

Home >

Scale set instance

Search (Ctrl+ /) Connect Start Restart Stop

Overview Settings

Instance ID : 110  
Status : Running, 1 more

To delete the saved agent when you are done with your investigation, navigate to your Azure DevOps **Project settings**, select **Agent pools** under **Pipelines**, and select your agent pool. Choose the tab labeled **Diagnostics**. Find the agent on the **Agents saved for investigation** card, and choose **Delete**. This removes the agent from the pool and deletes the associated virtual machine.

Update all agents

Jobs Agents Details Security **Diagnostics** Settings Maintenance History

Agents saved for investigation

We detected a problem with the following scale set agents and saved them for you to connect and debug. [Learn more about troubleshooting unhealthy scale set agents.](#)

Name	Saved
:000032	Yesterday at 12:50 PM

Delete

## FAQ

- [Where can I find the images used for Microsoft-hosted agents?](#)
- [How do I configure scale set agents to run UI tests?](#)
- [How can I delete agents?](#)
- [Can I configure the scale set agent pool to have zero agents on standby?](#)

## **Where can I find the images used for Microsoft-hosted agents?**

Licensing considerations limit us from distributing Microsoft-hosted images. We are unable to provide these images for you to use in your scale set agents. But, the [scripts](#) that we use to generate these images are open source. You are free to use these scripts and create your own custom images.

## **How do I configure scale set agents to run UI tests?**

Create a Scale Set with a Windows Server OS and when creating the Agent Pool select the "Configure VMs to run interactive tests" option.

## **How can I delete agents?**

Navigate to your Azure DevOps [Project settings](#), select **Agent pools** under **Pipelines**, and select your agent pool. Click the tab labeled **Agents**. Click the 'Enabled' toggle button to disable the agent. The disabled agent will complete the pipeline it is currently running and will not pick up additional work. Within a few minutes after completing its current pipeline job, the agent will be deleted.

## **Can I configure the scale set agent pool to have zero agents on standby?**

Yes, if you set **Number of agents to keep on standby** to zero, for example to conserve cost for a low volume of jobs, Azure Pipelines starts a VM only when it has a job.

# Run a self-hosted agent behind a web proxy

2/26/2020 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

When your self-hosted agent requires a web proxy, you can inform the agent about the proxy during configuration. This allows your agent to connect to Azure Pipelines or TFS through the proxy. This in turn allows the agent to get sources and download artifacts. Finally, it passes the proxy details through to tasks which also need proxy settings in order to reach the web.

## Azure Pipelines, TFS 2018 RTM and newer

(Applies to agent version 2.122 and newer.)

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 responsibly on each platform to prevent accidental leakage. 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 [azure-pipelines-task-lib](#) methods to retrieve proxy configuration and [handle the proxy](#) within their task.

Note that many tools do not automatically use the agent configured proxy settings. For example, tools such as `curl` and `dotnet` may require proxy environment variables such as `http_proxy` to also be set on the machine.

## TFS 2017.2 and older

### IMPORTANT

You also can use this method for Azure Pipelines and newer versions of TFS. We strongly recommend the more modern method, which you can access by switching to the TFS 2018 or Azure Pipelines docs.

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 a self-hosted agent in Docker

5/6/2021 • 9 minutes to read • [Edit Online](#)

This article provides instructions for running your Azure Pipelines agent in Docker. You can set up a self-hosted agent in Azure Pipelines to run inside a Windows Server Core (for Windows hosts), or Ubuntu container (for Linux hosts) with Docker. This is useful when you want to run agents with outer orchestration, such as [Azure Container Instances](#). In this article, you'll walk through a complete container example, including handling agent self-update.

Both [Windows](#) and [Linux](#) are supported as container hosts. Windows containers should run on a Windows `vmImage`. To run your agent in Docker, you'll pass a few [environment variables](#) to `docker run`, which configures the agent to connect to Azure Pipelines or Azure DevOps Server. Finally, you [customize the container](#) to suit your needs. Tasks and scripts might depend on specific tools being available on the container's `PATH`, and it's your responsibility to ensure that these tools are available.

This feature requires agent version 2.149 or later. Azure DevOps 2019 didn't ship with a compatible agent version. However, you can [upload the correct agent package to your application tier](#) if you want to run Docker agents.

## Windows

### Enable Hyper-V

Hyper-V isn't enabled by default on Windows. If you want to provide isolation between containers, you must enable Hyper-V. Otherwise, Docker for Windows won't start.

- [Enable Hyper-V on Windows 10](#)
- [Enable Hyper-V on Windows Server 2016](#)

#### NOTE

You must enable virtualization on your machine. It's typically enabled by default. However, if Hyper-V installation fails, refer to your system documentation for how to enable virtualization.

### Install Docker for Windows

If you're using Windows 10, you can install the [Docker Community Edition](#). For Windows Server 2016, install the [Docker Enterprise Edition](#).

### Switch Docker to use Windows containers

By default, Docker for Windows is configured to use Linux containers. To allow running the Windows container, confirm that Docker for Windows [is running the Windows daemon](#).

### Create and build the Dockerfile

Next, create the Dockerfile.

1. Open a command prompt.
2. Create a new directory:

```
mkdir C:\dockeragent
```

3. Change directories to this new directory:

```
cd C:\dockeragent
```

4. Save the following content to a file called `c:\dockeragent\Dockerfile` (no file extension):

```
FROM mcr.microsoft.com/windows/servercore:ltsc2019

WORKDIR /azp

COPY start.ps1 .

CMD powershell .\start.ps1
```

5. Save the following content to `c:\dockeragent\start.ps1`:

```
if (-not (Test-Path Env:AZP_URL)) {
 Write-Error "error: missing AZP_URL environment variable"
 exit 1
}

if (-not (Test-Path Env:AZP_TOKEN_FILE)) {
 if (-not (Test-Path Env:AZP_TOKEN)) {
 Write-Error "error: missing AZP_TOKEN environment variable"
 exit 1
 }

 $Env:AZP_TOKEN_FILE = "\azp\.token"
 $Env:AZP_TOKEN | Out-File -FilePath $Env:AZP_TOKEN_FILE
}

Remove-Item Env:AZP_TOKEN

if ((Test-Path Env:AZP_WORK) -and -not (Test-Path $Env:AZP_WORK)) {
 New-Item $Env:AZP_WORK -ItemType directory | Out-Null
}

New-Item "\azp\agent" -ItemType directory | Out-Null

Let the agent ignore the token env variables
$Env:VSO_AGENT_IGNORE = "AZP_TOKEN,AZP_TOKEN_FILE"

Set-Location agent

Write-Host "1. Determining matching Azure Pipelines agent..." -ForegroundColor Cyan

$base64AuthInfo = [Convert]::ToBase64String([Text.Encoding]::ASCII.GetBytes(":$(Get-Content $Env:AZP_TOKEN_FILE)"))
$package = Invoke-RestMethod -Headers @{Authorization=("Basic $base64AuthInfo")}
"${$Env:AZP_URL}/_apis/distributedtask/packages/agent?platform=win-x64&$top=1"
$packageUrl = $package[0].Value.downloadUrl

Write-Host $packageUrl

Write-Host "2. Downloading and installing Azure Pipelines agent..." -ForegroundColor Cyan

$wc = New-Object System.Net.WebClient
$wc.DownloadFile($packageUrl, "$(Get-Location)\agent.zip")

Expand-Archive -Path "agent.zip" -DestinationPath "\azp\agent"

try
{
 Write-Host "3. Configuring Azure Pipelines agent..." -ForegroundColor Cyan
```

```

.\config.cmd --unattended `

--agent "$($if (Test-Path Env:AZP_AGENT_NAME) { ${Env:AZP_AGENT_NAME} } else { ${Env:computername} })"
`--url "$($Env:AZP_URL)" `
`--auth PAT `
`--token "$($Get-Content ${Env:AZP_TOKEN_FILE})" `
`--pool $($if (Test-Path Env:AZP_POOL) { ${Env:AZP_POOL} } else { 'Default' })"
`--work $($if (Test-Path Env:AZP_WORK) { ${Env:AZP_WORK} } else { '_work' })"
`--replace

Write-Host "4. Running Azure Pipelines agent..." -ForegroundColor Cyan

.\run.cmd
}

finally
{
 Write-Host "Cleanup. Removing Azure Pipelines agent..." -ForegroundColor Cyan

.\config.cmd remove --unattended `

--auth PAT `
--token "$($Get-Content ${Env:AZP_TOKEN_FILE})"
}

```

## 6. Run the following command within that directory:

```
docker build -t dockeragent:latest .
```

This command builds the Dockerfile in the current directory.

The final image is tagged `dockeragent:latest`. You can easily run it in a container as `dockeragent`, because the `latest` tag is the default if no tag is specified.

### Start the image

Now that you have created an image, you can run a container.

1. Open a command prompt.
2. Run the container. This installs the latest version of the agent, configures it, and runs the agent. It targets the `Default` pool of a specified Azure DevOps or Azure DevOps Server instance of your choice:

```
docker run -e AZP_URL=<Azure DevOps instance> -e AZP_TOKEN=<PAT token> -e
AZP_AGENT_NAME=mydockeragent dockeragent:latest
```

Optionally, you can control the pool and agent work directory by using additional [environment variables](#).

If you want a fresh agent container for every pipeline run, pass the `--once` flag to the `run` command. You must also use a container orchestration system, like Kubernetes or [Azure Container Instances](#), to start new copies of the container when the work completes.

## Linux

### Install Docker

Depending on your Linux Distribution, you can either install [Docker Community Edition](#) or [Docker Enterprise Edition](#).

### Create and build the Dockerfile

Next, create the Dockerfile.

1. Open a terminal.
2. Create a new directory (recommended):

```
mkdir ~/dockeragent
```

3. Change directories to this new directory:

```
cd ~/dockeragent
```

4. Save the following content to `~/dockeragent/Dockerfile`:

```
FROM ubuntu:18.04

To make it easier for build and release pipelines to run apt-get,
configure apt to not require confirmation (assume the -y argument by default)
ENV DEBIAN_FRONTEND=noninteractive
RUN echo "APT::Get::Assume-Yes \"true\";" > /etc/apt/apt.conf.d/90assumeyes

RUN apt-get update && apt-get install -y --no-install-recommends \
 ca-certificates \
 curl \
 jq \
 git \
 iputils-ping \
 libcurl4 \
 libicu60 \
 libunwind8 \
 netcat \
 libssl11.0 \
 && rm -rf /var/lib/apt/lists/*
&& rm -rf /var/lib/apt/lists/*

RUN curl -LsS https://aka.ms/InstallAzureCLIDeb | bash \
 && rm -rf /var/lib/apt/lists/*

ARG TARGETARCH=amd64
ARG AGENT_VERSION=2.185.1

WORKDIR /azp
RUN if ["$TARGETARCH" = "amd64"]; then \
 AZP_AGENTPACKAGE_URL=https://vstsagentpackage.azureedge.net/agent/${AGENT_VERSION}/vsts-agent-\
linux-x64-${AGENT_VERSION}.tar.gz; \
else \
 AZP_AGENTPACKAGE_URL=https://vstsagentpackage.azureedge.net/agent/${AGENT_VERSION}/vsts-agent-\
linux-${TARGETARCH}-${AGENT_VERSION}.tar.gz; \
fi; \
curl -LsS "$AZP_AGENTPACKAGE_URL" | tar -xz

COPY ./start.sh .
RUN chmod +x start.sh

ENTRYPOINT ["./start.sh"]
```

#### NOTE

Tasks might depend on executables that your container is expected to provide. For instance, you must add the `zip` and `unzip` packages to the `RUN apt-get` command in order to run the `ArchiveFiles` and `ExtractFiles` tasks.

5. Save the following content to `~/dockeragent/start.sh`, making sure to use Unix-style (LF) line endings:

```

#!/bin/bash
set -e

if [-z "$AZP_URL"]; then
 echo 1>&2 "error: missing AZP_URL environment variable"
 exit 1
fi

if [-z "$AZP_TOKEN_FILE"]; then
 if [-z "$AZP_TOKEN"]; then
 echo 1>&2 "error: missing AZP_TOKEN environment variable"
 exit 1
 fi

 AZP_TOKEN_FILE=/azp/.token
 echo -n $AZP_TOKEN > "$AZP_TOKEN_FILE"
fi

unset AZP_TOKEN

if [-n "$AZP_WORK"]; then
 mkdir -p "$AZP_WORK"
fi

export AGENT_ALLOW_RUNASROOT="1"

cleanup() {
 if [-e config.sh]; then
 print_header "Cleanup. Removing Azure Pipelines agent..."

 # If the agent has some running jobs, the configuration removal process will fail.
 # So, give it some time to finish the job.
 while true; do
 ./config.sh remove --unattended --auth PAT --token $(cat "$AZP_TOKEN_FILE") && break

 echo "Retrying in 30 seconds..."
 sleep 30
 done
 fi
}

print_header() {
 lightcyan='\033[1;36m'
 nocolor='\033[0m'
 echo -e "${lightcyan}$1${nocolor}"
}

Let the agent ignore the token env variables
export VSO_AGENT_IGNORE=AZP_TOKEN,AZP_TOKEN_FILE

source ./env.sh

print_header "1. Configuring Azure Pipelines agent..."

./config.sh --unattended \
--agent "${AZP_AGENT_NAME:-$(hostname)}" \
--url "$AZP_URL" \
--auth PAT \
--token $(cat "$AZP_TOKEN_FILE") \
--pool "${AZP_POOL:-Default}" \
--work "${AZP_WORK:-_work}" \
--replace \
--acceptTeeEula & wait $!

print_header "2. Running Azure Pipelines agent..."

trap 'cleanup; exit 0' EXIT
trap 'cleanup; exit 130' INT

```

```
trap 'cleanup; exit 143' TERM

To be aware of TERM and INT signals call run.sh
Running it with the --once flag at the end will shut down the agent after the build is executed
./run.sh "$@"
```

#### NOTE

You must also use a container orchestration system, like Kubernetes or [Azure Container Instances](#), to start new copies of the container when the work completes.

- Run the following command within that directory:

```
docker build -t dockeragent:latest .
```

This command builds the Dockerfile in the current directory.

The final image is tagged `dockeragent:latest`. You can easily run it in a container as `dockeragent`, because the `latest` tag is the default if no tag is specified.

#### Start the image

Now that you have created an image, you can run a container.

- Open a terminal.
- Run the container. This installs the latest version of the agent, configures it, and runs the agent. It targets the `Default` pool of a specified Azure DevOps or Azure DevOps Server instance of your choice:

```
docker run -e AZP_URL=<Azure DevOps instance> -e AZP_TOKEN=<PAT token> -e
AZP_AGENT_NAME=mydockeragent dockeragent:latest
```

If you want a fresh agent container for every pipeline job, pass the `--once` flag to the `run` command.

```
docker run -e AZP_URL=<Azure DevOps instance> -e AZP_TOKEN=<PAT token> -e
AZP_AGENT_NAME=mydockeragent dockeragent:latest --once
```

Optionally, you can control the pool and agent work directory by using additional [environment variables](#).

## Environment variables

ENVIRONMENT VARIABLE	DESCRIPTION
AZP_URL	The URL of the Azure DevOps or Azure DevOps Server instance.
AZP_TOKEN	<a href="#">Personal Access Token (PAT)</a> with <a href="#">Agent Pools (read, manage)</a> scope, created by a user who has permission to <a href="#">configure agents</a> , at <code>AZP_URL</code> .
AZP_AGENT_NAME	Agent name (default value: the container hostname).
AZP_POOL	Agent pool name (default value: <code>Default</code> ).

ENVIRONMENT VARIABLE	DESCRIPTION
AZP_WORK	Work directory (default value: <code>_work</code> ).

## Add tools and customize the container

You have created a basic build agent. You can extend the Dockerfile to include additional tools and their dependencies, or build your own container by using this one as a base layer. Just make sure that the following are left untouched:

- The `start.sh` script is called by the Dockerfile.
- The `start.sh` script is the last command in the Dockerfile.
- Ensure that derivative containers don't remove any of the dependencies stated by the Dockerfile.

## Use Docker within a Docker container

In order to use Docker from within a Docker container, you bind-mount the Docker socket.

**Caution**

Doing this has serious security implications. The code inside the container can now run as root on your Docker host.

If you're sure you want to do this, see the [bind mount](#) documentation on Docker.com.

## Use Azure Kubernetes Service cluster

### Deploy and configure Azure Kubernetes Service

Follow the steps in [Quickstart: Deploy an Azure Kubernetes Service \(AKS\) cluster by using the Azure portal](#). After this, your PowerShell or Shell console can use the `kubectl` command line.

### Deploy and configure Azure Container Registry

Follow the steps in [Quickstart: Create an Azure container registry by using the Azure portal](#). After this, you can push and pull containers from Azure Container Registry.

### Configure secrets and deploy a replica set

1. Create the secrets on the AKS cluster.

```
kubectl create secret generic azdevops \
--from-literal=AZP_URL=https://dev.azure.com/yourOrg \
--from-literal=AZP_TOKEN=YourPAT \
--from-literal=AZP_POOL=NameOfYourPool
```

2. Run this command to push your container to Container Registry:

```
docker push <acr-server>/dockeragent:latest
```

3. Configure Container Registry integration for existing AKS clusters.

```
az aks update -n myAKSCluster -g myResourceGroup --attach-acr <acr-name>
```

4. Save the following content to `~/AKS/ReplicationController.yaml`:

```

apiVersion: apps/v1
kind: Deployment
metadata:
 name: azdevops-deployment
 labels:
 app: azdevops-agent
spec:
 replicas: 1 #here is the configuration for the actual agent always running
 selector:
 matchLabels:
 app: azdevops-agent
 template:
 metadata:
 labels:
 app: azdevops-agent
 spec:
 containers:
 - name: kubepodcreation
 image: AKRTestcase.azurecr.io/kubepodcreation:5306
 env:
 - name: AZP_URL
 valueFrom:
 secretKeyRef:
 name: azdevops
 key: AZP_URL
 - name: AZP_TOKEN
 valueFrom:
 secretKeyRef:
 name: azdevops
 key: AZP_TOKEN
 - name: AZP_POOL
 valueFrom:
 secretKeyRef:
 name: azdevops
 key: AZP_POOL
 volumeMounts:
 - mountPath: /var/run/docker.sock
 name: docker-volume
 volumes:
 - name: docker-volume
 hostPath:
 path: /var/run/docker.sock

```

This Kubernetes YAML creates a replica set and a deployment, where `replicas: 1` indicates the number or the agents that are running on the cluster.

## 5. Run this command:

```
kubectl apply -f ReplicationController.yaml
```

Now your agents will run the AKS cluster.

## Common errors

If you're using Windows, and you get the following error:

```
'standard_init_linux.go:178: exec user process caused "no such file or directory"
```

Install Git Bash by downloading and installing [git-scm](#).

Run this command:

```
dos2unix ~/dockeragent/Dockerfile
dos2unix ~/dockeragent/start.sh
git add .
git commit -m 'Fixed CR'
git push
```

Try again. You no longer get the error.

# Run the agent with a self-signed certificate

2/26/2020 • 3 minutes to read • [Edit Online](#)

Azure DevOps Server 2020 | Azure DevOps Server 2019 | 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://tfs.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 certificates generated by IIS or PowerShell command may not be compatible 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 information
  - 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](#).

# Provision deployment groups

3/6/2021 • 4 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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](#).

Deployment groups are only available with Classic release pipelines. Deployment groups are different from [deployment jobs](#). Deployment jobs are a collection of deployment-related steps used in a YAML pipeline.

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](#). Tags are each limited to 256 characters, but there is no limit to the number of tags you can use. 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 an installed build and release agent. You can do this using the script 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 if:

- 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 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.

The screenshot shows the 'Deployment groups' page. At the top, there's a breadcrumb navigation: 'Deployment groups > Contoso SHARED'. Below this is a toolbar with tabs: 'Details' (underlined), 'Targets', 'Save', 'Share', 'Security', and 'Help'.  
  
The main area starts with a 'Deployment group name' field containing 'Contoso'. Below it is a 'Description' field with an empty text area.  
  
At the bottom, there's a 'Deployment pool' section. It shows a dropdown menu with 'deployment-groups-Release' selected. To the right of the dropdown is a button labeled 'Manage' with a gear icon, which is highlighted with a red box.

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

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.

### Automatically deploy to new targets in a deployment group

When new targets get added to a deployment group, you can configure the environment to automatically deploy the last successful release to the new targets.

All pipelines > **Fabfiber-portal.CD**

Save + Release ...

Pipeline Tasks Variables Retention Options History

Environments | [+ Add](#)

**TEST**  
2 phases, 3 tasks

**Post-deployment conditions**

**TEST**

**Post-deployment approvals**  Disabled  
Select the users who can approve or reject deployments to this environment

**Gates**  Disabled  
Define gates to evaluate after the deployment. [Learn more](#)

**Auto redeploy trigger**  Enabled  
Configure the events that trigger automated redeployment.

Select events \*  
New target with required tags becomes available

Select action \*  
Redeploy the last successful deployment on this environment

## 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
- Get advice on [Stack Overflow](#), and feel free to post your questions, search for answers, or suggest a feature on our [Azure DevOps Developer Community](#). [Support](#) page.

# Provision agents for deployment groups

3/17/2021 • 9 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

**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 ARM Template 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](#).

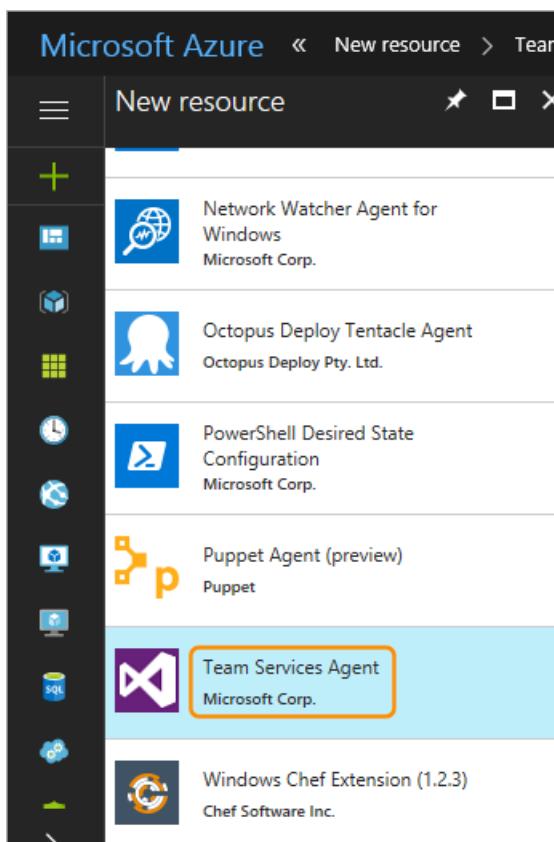
- When prompted for the user account, press *Return* to accept the defaults.
- Wait for the script to finish with the message

```
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 ARM Template deployment task

### IMPORTANT

These instructions refer to version 2 of the task. Switch your **Task version** from 3 to 2.

You can use the [ARM Template 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 ARM Template 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`.

## Troubleshoot the extension

These are some known issues with the extension:

- **Status file getting too big:** This issue occurs on Windows VMs; it has not been observed on Linux VMs.

The status file contains a JSON object that describes the current status of the extension. The object is a placeholder to list the operations performed so far. Azure reads this status file and passes the status object as response to API requests. The file has a maximum allowed size; if the size exceeds the threshold, Azure cannot read it completely and gives an error for the status. On each machine reboot, some operations are performed by the extension (even though it might be installed successfully earlier), which append the status file. If the machine is rebooted a large number of times, the status file size exceeds the threshold, which causes this error. The error message reads:

```
Handler Microsoft.VisualStudio.Services.TeamServicesAgent:1.27.0.2 status file 0.status size xxxxxxxx
bytes is too big. Max Limit allowed: 131072 bytes
```

. Note that extension installation might have succeeded, but this error hides the actual state of the extension.

We have fixed this issue for machine reboots (version `1.27.0.2` for Windows extension and `1.21.0.1` for Linux extension onward), so on a reboot, nothing will be added to the status file. If you had this issue with your extension before the fix was made (that is, you were having this issue with earlier versions of the extension) and your extension was auto-updated to the versions with the fix, the issue will still persist. This is because on extension update, the newer version of the extension still works with the earlier status file. Currently, you could still be facing this issue if you are using an earlier version of the extension with the flag to turn off minor version auto-updates, or if a large status file was carried from an earlier extension version to the newer versions that contains the fix, or for any other reason. If that is the case, you can get past this issue by uninstalling and re-installing the extension. Uninstalling the extension cleans up the entire extension directory, so a new status file will be created for fresh install. You need to install latest version of the extension. This solution is a permanent fix, and after following this, you should not face the issue again.

- **Issue with custom data:** This issue is not with the extension, but some customers have reported confusion regarding the customdata location on the VM on switching OS versions. We suggest the following workaround. Python 2 has been deprecated, so we have made the extension to work with Python 3. If you are still using earlier OS versions that don't have Python 3 installed by default, to run the extension, you should either install Python 3 on the VM or switch to OS versions that have Python 3 installed by default. On linux VMs, **custom data** is copied to the file `/var/lib/waagent/ovf-env.xml` for earlier Microsoft Azure Linux Agent versions, and to `/var/lib/waagent/CustomData` for newer Microsoft Azure Linux Agent versions. It appears that customers who have hardcoded only one of these two paths face issues while switching OS versions because the file does not exist on the new OS version, but the other file is present. So, to avoid breaking the VM provisioning, you should consider both the files in the template so that if one fails, the other should succeed.

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 **ARM Template 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 **ARM Template 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
- Get advice on [Stack Overflow](#), and feel free to post your questions, search for answers, or suggest a feature on our [Azure DevOps Developer Community Support](#) page.

# Deployment group jobs

11/2/2020 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

[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](#)
- [Classic](#)

## NOTE

Deployment group jobs are not yet supported in YAML. You can use [Virtual machine resources in Environments](#) to do a rolling deployment to VMs in YAML pipelines.

Rolling deployments can be configured by specifying the keyword `rolling:` under `strategy:` node of a [deployment job](#).

```
strategy:
 rolling:
 maxParallel: [number or percentage as x%]
 preDeploy:
 steps:
 - script: [script | bash | pwsh | powershell | checkout | task | templateReference]
 deploy:
 steps:
 ...
 routeTraffic:
 steps:
 ...
 postRouteTraffic:
 steps:
 ...
 on:
 failure:
 steps:
 ...
 success:
 steps:
 ...
```

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](#)
- [Conditions](#)

# Deploy to Azure VMs using deployment groups in Azure Pipelines

3/18/2021 • 8 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

In earlier versions of Azure Pipelines, applications that needed to be deployed to multiple servers required a significant amount of planning and maintenance. Windows PowerShell remoting had to be enabled manually, required ports opened, and deployment agents installed on each of the servers. The pipelines then had to be managed manually if a roll-out deployment was required.

All the above challenges have been evolved seamlessly with the introduction of the [Deployment Groups](#).

A deployment group installs a deployment agent on each of the target servers in the configured group and instructs the release pipeline to gradually deploy the application to those servers. Multiple pipelines can be created for the roll-out deployments so that the latest version of an application can be delivered in a phased manner to multiple user groups for validation of newly introduced features.

## NOTE

Deployment groups are a concept used in Classic pipelines. If you are using YAML pipelines, see [Environments](#).

In this tutorial, you learn about:

- Provisioning VM infrastructure to Azure using a template
- Creating an Azure Pipelines deployment group
- Creating and running a CI/CD pipeline to deploy the solution with a deployment group

## Prerequisites

- A Microsoft Azure account.
- An Azure DevOps organization.

Use the [Azure DevOps Demo Generator](#) to provision the tutorial project on your Azure DevOps organization.

## Setting up the Azure deployment environment

The following resources are provisioned on the Azure using an ARM template:

- Six Virtual Machines (VM) web servers with IIS configured
- SQL server VM (DB server)
- Azure Network Load Balancer

1. Click the **Deploy to Azure** button below to initiate resource provisioning. Provide all the necessary

information and select **Purchase**. You may use any combination of allowed administrative usernames and passwords as they are not used again in this tutorial. The **Env Prefix Name** is prefixed to all of the resource names in order to ensure that those resources are generated with globally unique names. Try to use something personal or random, but if you see a naming conflict error during validation or creation, try changing this parameter and running again.

 Deploy to Azure

**BASICS**

Subscription \* Contoso Azure

Resource group \* (New) DeploymentGroups  
Create new

Location \* (US) West US 2

**SETTINGS**

Env Prefix Name ⓘ cust1

Username \* ⓘ vmadmin

Password \* ⓘ .....  
Sql Authentication Login \* sysadmin

Sql Authentication Password \* .....

Web Srv VM Size ⓘ Standard\_D2\_v2

Number Of Web Srvs ⓘ 6

Sql VM Size ⓘ Standard\_D2\_v2

Storage Account Type ⓘ Standard\_LRS

**TERMS AND CONDITIONS**

Azure Marketplace Terms | Azure Marketplace

By clicking "Purchase," I (a) agree to the applicable legal terms associated with the offering; (b) authorize Microsoft to charge or bill my current payment method for the fees associated the offering(s), including applicable taxes, with the same billing frequency as my Azure subscription, until I discontinue use of the offering(s); and (c) agree that, if the deployment involves 3rd party offerings, Microsoft may share my contact information and other details of such deployment with the publisher of that offering.

I agree to the terms and conditions stated above

**Purchase**

#### NOTE

It takes approximately 10-15 minutes to complete the deployment. If you receive any naming conflict errors, try changing the parameter you provide for **Env Prefix Name**.

- Once the deployment completes, you can review all of the resources generated in the specified resource group using the Azure portal. Select the DB server VM with **sqlSrv** in its name to view its details.

Name ↑↓	Type ↑↓	Location ↑↓	
cust1SqlPip	Public IP address	West US 2	...
<b>cust1sqlSrv14</b>	Virtual machine	West US 2	...
cust1sqlSrv14Nic	Network interface	West US 2	...
cust1stgjo5zndv53idtw	Storage account	West US 2	...
cust1Vnet	Virtual network	West US 2	...
cust1webSrv0	Virtual machine	West US 2	...
cust1webSrv1	Virtual machine	West US 2	...
cust1webSrv2	Virtual machine	West US 2	...

3. Make a note of the **DNS name**. This value is required in a later step. You can use the copy button to copy it to the clipboard.

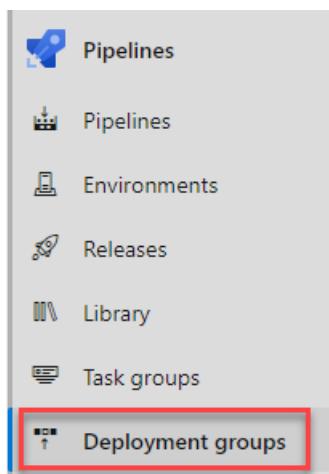
Resource group (change) : DeploymentGroups	Operating system	: Windows (Windows Server 2016 Datacenter)
Status : Running	Size	: Standard D2 v2 (2 vcpus, 7 GiB memory)
Location : West US 2	Public IP address	: 52.175.214.35
Subscription (change) : Contoso	Colocation status	: N/A
Subscription ID : 01234567-89ab-cdef-0123-456789abcdef	Virtual network/subnet	: cust1Vnet/DBSubnetName
	DNS name	: cust1sqljo5zndv53idtw.westus2.cloudapp.azure.com 

## Creating and configuring a deployment group

Azure Pipelines makes it easier to organize servers required for deploying applications. A deployment group is a collection of machines with deployment agents. Each of the machines interacts with Azure Pipelines to coordinate the deployment of the app.

Since there is no configuration change required for the build pipeline, the build is triggered automatically after the project is provisioned. When you queue a release later on, this build is used.

1. Navigate to the Azure DevOps project created by the demo generator.
2. From under **Pipelines**, navigate to **Deployment groups**.



3. Select **Add a deployment group**.
4. Enter the **Deployment group name of Release** and select **Create**. A registration script is generated. You can register the target servers using the script provided if working on your own. However, in this tutorial, the target servers are automatically registered as part of the release pipeline. The release definition uses stages to deploy the application to the target servers. A stage is a logical grouping of the tasks that defines the runtime target on which the tasks will execute. Each deployment group stage

executes tasks on the machines defined in the deployment group.

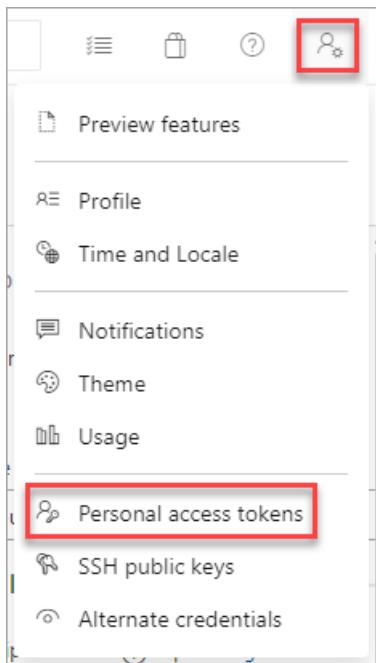
5. From under **Pipelines**, navigate to **Releases**. Select the release pipeline named **Deployment Groups** and select **Edit**.
6. Select the **Tasks** tab to view the deployment tasks in pipeline. The tasks are organized as three stages called **Agent phase**, **Deployment group phase**, and **IIS Deployment phase**.
7. Select the **Agent phase**. In this stage, the target servers are associated with the deployment group using the Azure Resource Group Deployment task. To run, an agent pool and specification must be defined. Select the **Azure Pipelines** pool and **vs2017-win2016** specification.

The screenshot shows the 'Agent phase' configuration in the Azure DevOps interface. The 'Run on agent' section is highlighted with a red box. The 'Agent selection' section contains fields for 'Agent pool' (set to 'Azure Pipelines') and 'Agent Specification' (set to 'vs2017-win2016'), both of which are also highlighted with red boxes.

8. Select the **Azure Resource Group Deployment** task. Configure a service connection to the Azure subscription used earlier to create infrastructure. After authorizing the connection, select the resource group created for this tutorial.

The screenshot shows the configuration of the 'Azure Resource Group Deployment' task. The task name is highlighted with a red box. The 'Azure subscription' dropdown is set to 'Contoso Azure' and is also highlighted with a red box. The 'Resource group' dropdown is set to 'DeploymentGroups' and is also highlighted with a red box.

9. This task will run on the virtual machines hosted in Azure, and will need to be able to connect back to this pipeline in order to complete the deployment group requirements. To secure the connection, they will need a **personal access token (PAT)**. From the **User settings** dropdown, open **Personal access tokens** in a new tab. Most browsers support opening a link in a new tab via right-click context menu or **Ctrl+Click**.

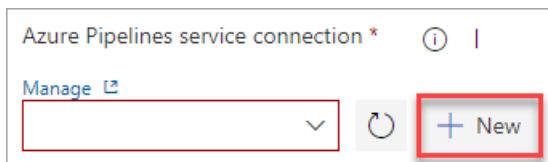


10. In the new tab, select **New Token**.
11. Enter a name and select the **Full access** scope. Select **Create** to create the token. Once created, copy the token and close the browser tab. You return to the Azure Pipeline editor.

**Create a new personal access token**

Name	Deployment Groups
Organization	contoso
Expiration (UTC)	30 days 7/19/2020
Scopes	Authorize the scope of access associated with this token
Scopes	<input checked="" type="radio"/> Full access <input type="radio"/> Custom defined
<b>Create</b> Cancel	

12. Under **Azure Pipelines service connection**, select **New**.



13. Enter the **Connection URL** to the current instance of Azure DevOps. This URL is something like [https://dev.azure.com/\[Your account\]](https://dev.azure.com/[Your account]). Paste in the **Personal Access Token** created earlier and specify a **Service connection name**. Select **Verify and save**.

**New service connection**

Authentication method

Token Based Authentication

Basic Authentication

Connection URL

URL of the Azure DevOps organization or the TFS Project Collection to connect to.

**Authentication**

Personal Access Token

Personal access tokens are applicable only for connections targeting Azure DevOps organizations or TFS 2017 (and higher). Refer to Personal Access Token link for information on how to create a personal access token.

**Verify**

**Details**

Service connection name

Description (optional)

Security

Grant access permission to all pipelines

[Learn more](#)

**Verify and save**

[Troubleshoot](#)

#### NOTE

To register an agent, you must be a member of the Administrator role in the agent pool. The identity of the agent pool administrator is needed only at the time of registration. The administrator identity isn't persisted on the agent, and it's not used in any subsequent communication between the agent and Azure Pipelines or TFS. After the agent is registered, there's no need to renew the personal access token because it's required only at the time of registration.

14. Select the current **Team project** and the **Deployment group** created earlier.

Manage 

Azure Pipelines  

Team project \* 



Deployment Group \* 



15. Select the **Deployment group phase** stage. This stage executes tasks on the machines defined in the deployment group. This stage is linked to the **SQL-Svr-DB** tag. Choose the **Deployment Group** from the dropdown.

16. Select the **IIS Deployment phase** stage. This stage deploys the application to the web servers using the specified tasks. This stage is linked to the **WebSrv** tag. Choose the **Deployment Group** from the dropdown.
17. Select the **Disconnect Azure Network Load Balancer** task. As the target machines are connected to the NLB, this task will disconnect the machines from the NLB prior to the deployment and reconnect them back to the NLB after the deployment. Configure the task to use the Azure connection, resource group, and load balancer (there should only be one).
18. Select the **IIS Web App Manage** task. This task runs on the deployment target machines registered with the deployment group configured for the task/stage. It creates a web app and application pool locally with the name **PartsUnlimited** running under the port **80**
19. Select the **IIS Web App Deploy** task. This task runs on the deployment target machines registered with the deployment group configured for the task/stage. It deploys the application to the IIS server using **Web Deploy**.
20. Select the **Connect Azure Network Load Balancer** task. Configure the task to use the Azure connection, resource group, and load balancer (there should only be one).
21. Select the **Variables** tab and enter the variable values as below.

VARIABLE NAME	VARIABLE VALUE
DatabaseName	PartsUnlimited-Dev
DBPassword	P2ssw0rd@123
DBUserName	sqladmin
DefaultConnectionString	Data Source=[YOUR_DNS_NAME];Initial Catalog=PartsUnlimited-Dev;User ID=sqladmin;Password=P2ssw0rd@123;MultipleActiveResultSets=False;Connection Timeout=30;
ServerName	localhost

#### IMPORTANT

Make sure to replace your SQL server DNS name (which you noted from Azure portal earlier) in **DefaultConnectionString** variable.

Your DefaultConnectionString should be similar to this string after replacing the SQL DNS:

```
Data Source=cust1sqljo5zndv53idtw.westus2.cloudapp.azure.com;Initial Catalog=PartsUnlimited-Dev;User ID=sqladmin;Password=P2ssw0rd@123;MultipleActiveResultSets=False;Connection Timeout=30;
```

The final variable list should look something like this:

Pipeline	Tasks	Variables	Retention	Options	History
Pipeline variables					
Variable groups					
Predefined variables		Name	Value	Scope	
		DatabaseName	PartsUnlimited-Dev	Release	▼
		DBPassword	P2ssw0rd@123	Release	▼
		DBUserName	sqladmin	Release	▼
		DefaultConnectionString	Data Source=cust1sqljo5zndv53idtw.westus2.cloudapp.azure.com;Initial Catalog=PartsUnlimited-Dev;User ID=sqladmin;Password=P2ssw0rd@123;MultipleActiveResultSets=False;Connection Timeout=30;	Release	▼
		ServerName	localhost	Release	▼

#### NOTE

You may receive an error that the `DefaultConnectionString` variable must be saved as a secret. If that happens, select the variable and click the padlock icon that appears next to its value to protect it.

## Queuing a release and reviewing the deployment

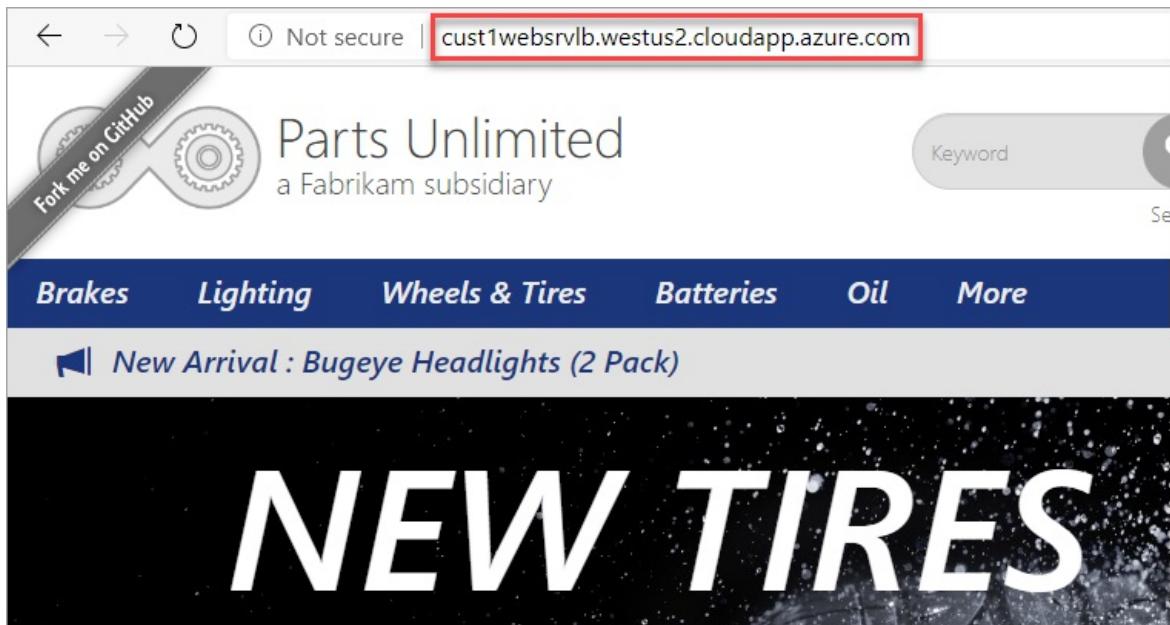
1. Select **Save** and confirm.
2. Select **Create release** and confirm. Follow the release through to completion. The deployment is then ready for review.
3. In the Azure portal, open one of the web VMs in your resource group. You can select any that have `websrv` in the name.

<input type="checkbox"/>	Name ↑↓	Type ↑↓
<input type="checkbox"/>	cust1SqlPip	Public IP address
<input type="checkbox"/>	cust1sqlSrv14	Virtual machine
<input type="checkbox"/>	cust1sqlSrv14Nic	Network interface
<input type="checkbox"/>	cust1stgj05zndv53idtw	Storage account
<input type="checkbox"/>	cust1Vnet	Virtual network
<input type="checkbox"/>	cust1webSrv0	Virtual machine
<input type="checkbox"/>	cust1webSrv1	Virtual machine
<input type="checkbox"/>	cust1webSrv2	Virtual machine

4. Copy the **DNS** of the VM. The [Azure Load Balancer](#) will distribute incoming traffic among healthy instances of servers defined in a load-balanced set. As a result, the **DNS** of all web server instances is the same.

Resource group ( <a href="#">change</a> ) : DeploymentGroups	Operating system	: Windows (Windows Server 2012 R2 Datacenter)
Status : Running	Size	: Standard D2 v2 (2 vcpus, 7 GiB memory)
Location : West US 2	Public IP address	: <a href="#">52.175.217.169</a>
Subscription ( <a href="#">change</a> ) : Contoso	Colocation status	: N/A
Subscription ID : 01234567-89ab-cdef-0123-456789abcdef	Virtual network/subnet	: cust1Vnet/FESubnetName
	DNS name	: cust1websrvlb.westus2.cloudapp.azure.com

5. Open a new browser tab to the DNS of the VM. Confirm the deployed app is running.



## Summary

In this tutorial, you deployed a web application to a set of Azure VMs using Azure Pipelines and Deployment Groups. While this scenario covered a handful of machines, you can easily scale the process up to support hundreds, or even thousands, of machines using virtually any configuration.

## Cleaning up resources

This tutorial created an Azure DevOps project and some resources in Azure. If you're not going to continue to use these resources, delete them with the following steps:

1. Delete the Azure DevOps project created by the Azure DevOps Demo Generator.
2. All Azure resources created during this tutorial were assigned to the resource group specified during creation. Deleting that group will delete the resources they contain. This deletion can be done via the CLI or portal.

## Next steps

[Provision agents for deployment groups](#)

# Set retention policies for builds, releases, and tests

6/28/2021 • 18 minutes to read • [Edit Online](#)

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Retention policies let you set how long to keep runs, releases, and tests stored in the system. To save storage space, you want to delete older runs, tests, and releases.

The following retention policies are available in Azure DevOps in your **Project settings**:

1. **Pipeline** - Set how long to keep artifacts, symbols, attachments, runs, and pull request runs.
2. **Release (classic)** - Set whether to save builds and view the default and maximum retention settings.
3. **Test** - Set how long to keep automated and manual test runs, results, and attachments.

The screenshot shows the 'Project Settings' page for the 'FabrikamFiber' project. The left sidebar lists various settings categories. In the main content area, the 'Settings' tab is selected, showing the 'Retention policy' configuration. A note states: 'The artifacts, symbols and attachments retention setting is being ignored because the runs retention setting is evaluated first.' Below this, four retention settings are defined: 'Days to keep artifacts, symbols and attachments' (30), 'Days to keep runs' (30), 'Days to keep pull request runs' (10), and 'Number of recent runs to retain per pipeline' (3). A link 'Learn more about run retention' is provided. At the bottom of the page, under the 'General' section, several job authorization and metadata publishing options are listed with toggle switches.

Project Settings  
FabrikamFiber

General

- Overview
- Teams
- Permissions
- Notifications
- Service hooks
- Dashboards

Boards

- Project configuration
- Team configuration
- GitHub connections

Pipelines

- Agent pools
- Parallel jobs
- 1 Settings
- 2 Test management
- Release retention
- Service connections
- XAML build services

Repos

- Repositories

Artifacts

- Storage

Test

- 3 Retention

Settings

Retention policy

The artifacts, symbols and attachments retention setting is being ignored because the runs retention setting is evaluated first.

Days to keep artifacts, symbols and attachments: 30

Days to keep runs: 30

Days to keep pull request runs: 10

Number of recent runs to retain per pipeline: 3

Learn more about run retention

General

- Disable anonymous access to badges
- Limit variables that can be set at queue time
- Limit job authorization scope to current project for non-release pipelines
- Limit job authorization scope to current project for release pipelines
- Publish metadata from pipelines (preview)
- Limit job authorization scope to referenced Azure DevOps repositories

#### NOTE

If you are using an on-premises server, you can also specify retention policy defaults for a project and when releases are permanently destroyed. Learn more about [release retention](#) later in this article.

## Prerequisites

By default, members of the Contributors, Build Admins, Project Admins, and Release Admins groups can manage retention policies.

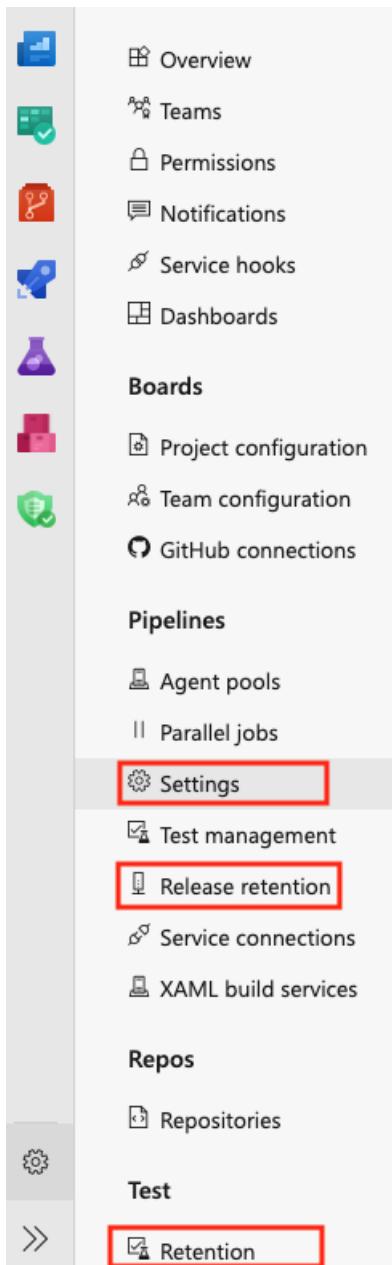
To manage test results, you must have one of the following subscriptions:

- [Enterprise](#)
- [Test Professional](#)
- [MSDN Platforms](#)

You can also buy monthly access to Azure Test Plans and assign the [Basic + Test Plans](#) access level. See [Testing access by user role](#).

## Configure retention policies

1. Sign in to your project (<https://dev.azure.com/{yourorganization}/{yourproject}>).
2. Go to the  **Settings** tab of your project's settings.
3. Select **Settings** or **Release retention** under **Pipelines** or **Retention** under **Test**.
  - Select **Settings** to configure retention policies for runs, artifacts, symbols, attachments, and pull request runs.
  - Select **Release retention** to set up your release retention policies and configure when to delete or permanently destroy releases.
  - Select **Retention** to set up how long to keep manual and automated test runs.



## Set run retention policies

In most cases, you don't need to retain completed runs longer than a certain number of days. Using retention policies, you can control **how many days** you want to keep each run before deleting it.

Along with defining how many days to retain runs, you can also decide the minimum number of runs that should be kept for each pipeline.

1. Go to the **Settings** tab of your project's settings.
2. Select **Settings** in the Pipelines section.
  - Set the number of days to keep [artifacts](#), symbols, and attachments.
  - Set the number of days to keep [runs](#)
  - Set the number of days to keep pull request [runs](#)
  - Set the number of recent [runs](#) to keep for each pipeline

**WARNING**

Azure DevOps no longer supports per-pipeline retention rules. The only way to configure retention policies for YAML and classic pipelines is through the project settings described above. You can no longer configure per-pipeline retention policies.

The setting for number of recent runs to keep for each pipeline requires a little more explanation. The interpretation of this setting varies based on the type of repository you build in your pipeline.

- **Azure Repos:** Azure Pipelines always retains the configured number of latest runs for the default branch and for each protected branch of the repository. A branch that has any branch policies configured is considered to be a protected branch. As an example, consider a repository with the default branch called **main**. Also, let us assume that the **release** branch in this repository has a branch policy. In this case, if you configured the policy to retain 3 runs, then the latest 3 runs of main as well as the latest 3 runs of release branch are retained. In addition, the latest 3 runs of this pipeline (irrespective of the branch) are also retained.

To clarify this logic further, let us say that the list of runs for this pipeline is as follows with the most recent run at the top. The table shows which runs will be retained if you have configured to retain the latest 3 runs (ignoring the effect of the number of days setting):

RUN #	BRANCH	RETAINED / NOT RETAINED	WHY?
Run 10	main	Retained	Latest 3 for main
Run 9	branch1	Retained	Latest 3 for pipeline
Run 8	branch2	Retained	Latest 3 for pipeline
Run 7	main	Retained	Latest 3 for main
Run 6	main	Retained	Latest 3 for main
Run 5	main	Not retained	Neither latest 3 for main, nor for pipeline
Run 4	main	Not retained	Neither latest 3 for main, nor for pipeline
Run 3	branch1	Not retained	Neither latest 3 for main, nor for pipeline
Run 2	release	Retained	Latest 3 for release
Run 1	main	Not retained	Neither latest 3 for main, nor for pipeline

- **All other Git repositories:** Azure Pipelines retains the configured number of latest runs for the default branch of the repository and for the whole pipeline.
- **TFVC:** Azure Pipelines retains the configured number of latest runs for the whole pipeline, irrespective of the branch.

### What parts of the run 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

The following information is deleted when a run is deleted:

- Logs
- All pipeline and build artifacts
- All symbols
- Binaries
- Test results
- Run metadata

Universal packages, NuGet, npm, and other packages are not tied to pipelines retention.

### **When are runs deleted**

Your retention policies are processed once a day. The time that the policies get processed variables because we spread the work throughout the day for load-balancing purposes. There is no option to change this process.

A run is deleted if all of the following conditions are true:

- It exceeds the number of days configured in the retention settings
- It is not one of the recent runs as configured in the retention settings
- It is not marked to be retained indefinitely
- It is not retained by a release

Your retention policies run every day at 3:00 A.M. UTC. There is no option to change the time the policies run.

### **Automatically set retention lease on pipeline runs**

Retention leases are used to manage the lifetime of pipeline runs beyond the configured retention periods. Retention leases can be added or deleted on a pipeline run by calling the [Lease API](#). This API can be invoked within the pipeline using a script and using [predefined variables](#) for runId and definitionId.

A retention lease can be added on a pipeline run for a specific period. For example, a pipeline run which deploys to a test environment can be retained for a shorter duration while a run deploying to production environment can be retained longer.

## Delete a run

You can delete runs using the [More actions menu](#) on the [Pipeline run details](#) page.

## NOTE

If any retention policies currently apply to the run, they must be removed before the run can be deleted. For instructions, see [Pipeline run details - delete a run](#).

The screenshot shows the Azure Pipelines interface for a pipeline run. At the top, it displays the pipeline name '#20200528.2 Set up CI with Azure Pipelines' and the repository 'on azure-quickstart-templates Retained'. Below this, there are tabs for 'Summary' (which is selected) and 'Code Coverage'. The summary section includes details like 'Manually run by Rami Bououni', 'Repository and version' (ramiMSFT/azure-quickstart-templates, master, e75a65e), 'Time started and elapsed' (May 28 at 9:35 AM, 3m 13s), 'Related' (0 work items, 0 artifacts), and 'Tests and coverage' (Get started). To the right, a context menu is open with options: 'Run new', 'Download logs', 'Add tags', 'Edit pipeline', 'View retention leases', 'Retain', 'Delete' (which is highlighted with a red box), and 'Release'. Below the summary is a 'Jobs' section with a table:

Name	Status	Duration
Job	Success	3m 3s

## Set release retention policies

The release retention policies for a classic release pipeline determine how long a release and the run 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. See FAQ below for more details about how release retention works.

As an author of a release pipeline, you can customize retention policies for releases of your pipeline on the **Retention** tab.

The retention policy for YAML and build pipelines is the same. You can see your pipeline's retention settings in **Project Settings for Pipelines** in the **Settings** section.

You can also learn how to customize these policies on a [stage-by-stage basis](#) later in this article.

### Global release retention policy

If you are using an on-premises Team Foundation Server or Azure DevOps 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).

## Retention policy settings

### Maximum retention policy

Days to retain a release	<input type="text" value="365"/>
Minimum releases to keep	<input type="text" value="25"/>

### Default retention policy

Days to retain a release	<input type="text" value="30"/>
Minimum releases to keep	<input type="text" value="3"/>
Retain build	<input checked="" type="checkbox"/>

### Permanently destroy releases

Days to keep releases after deletion	<input type="text" value="15"/>
--------------------------------------	---------------------------------

**Save changes**

**Undo changes**

If you are using Azure DevOps Services, you can view but not change these settings for your project.

Global release retention policy settings can be managed from the **Release retention** settings of your project:

- Azure DevOps Services:

```
https://dev.azure.com/{organization}/{project}/_settings/release?app=ms.vss-build-web.build-release-hub-group
```

- 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.

## Set collection-level retention policies

For on-premises servers, you can also set the collection-level retention policies with custom retention rules.

These retention policies apply to Classic build pipelines. The page at

[https://{{your\\_server}}/{{collection\\_name}}/\\_settings/buildqueue](https://{{your_server}}/{{collection_name}}/_settings/buildqueue) governs your maximum values and default values.

## Collection Settings

DefaultCollection

### General

- Projects
- Auditing
- Global notifications
- Extensions
- Analytics

### Security

- Security

### Boards

- Process

### Pipelines

- Agent pools

- Settings

- Deployment pools

- Retention

### Maximum retention policy

Days to keep:	<input type="text" value="30"/>
Minimum to keep:	<input type="text" value="10"/>
Delete build record:	true
Delete source label:	false
Delete file share:	true
Delete symbols:	true
Delete test results:	true
Branch filters:	All

### Default retention policy

Days to keep:	<input type="text" value="10"/>
Minimum to keep:	<input type="text" value="1"/>
Delete build record:	true
Delete source label:	false
Delete file share:	true
Delete symbols:	true
Delete test results:	true
Branch filters:	All

### Permanently destroy builds

Days to keep build record after deletion:

[Save changes](#) [Undo changes](#)

### NOTE

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 policies

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 sections: 'Dev' (Keep for 30 days, 3 good releases and keep artifacts), 'Production' (Keep for 30 days, 3 good releases and keep artifacts), and 'QA' (Keep for 30 days, 3 good releases and keep artifacts). The 'Dev' section is highlighted. On the right, under 'Settings for Dev', there are fields for 'Days to retain a release' (set to 30), 'Minimum releases to keep' (set to 3), and a checked checkbox for 'Retain associated artifacts'. Below these fields is a link to 'View or manage retention policy defaults'.

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 policies

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 associated artifacts** 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.

#### NOTE

In TFS, interaction between build and release retention is available in TFS 2017 and newer.

## Set test retention policies

You can set manual and automated test run policies.

### Manual test-runs retention policies

To delete manual test results after a specific number of days, set the retention limit at the project level. Azure DevOps keeps manual test results related to builds, even after you delete those builds. That way, build policies don't delete your test results before you can analyze the data.

1. Sign into your Azure DevOps. You'll need at least project administrator permissions.
2. Go to your project and then select project settings at the bottom of the page.
3. In the Retention page under the Test section, select a limit for how long you want to keep manual test data.

**Project Settings**

ARM\_Template

**General**

- Overview
- Teams
- Permissions
- Notifications
- Service hooks
- Dashboards

**Boards**

- Project configuration
- Team configuration
- GitHub connections

**Repos**

- Repositories

**Pipelines**

- Agent pools
- Parallel jobs
- Settings
- Test management
- Release retention
- Service connections

**Test**

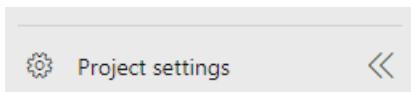
- Retention

Save changes      Undo changes

## Automated test-runs retention policies

By default, Azure DevOps keeps automated test results related to builds only as long as you keep those builds. To keep test results after you delete your builds, edit the build retention policy. If you use Git for version control, you can specify how long to keep automated test results based on the branch.

1. Sign into Azure DevOps. You'll need at least build level permissions to edit build pipelines.
2. Go to your project and then select project settings at the bottom of the page.



3. Select Settings under Pipelines and modify your retention policies.

The screenshot shows the 'Project Settings' page for a project named 'ARM\_Template'. The left sidebar lists various settings categories: General, Boards, Repos, Pipelines, and Service connections. The 'Pipelines' category is expanded, showing sub-options like 'Parallel jobs' and 'Settings'. The 'Settings' option under Pipelines is highlighted with a red box. The main content area is titled 'Settings' and contains two sections: 'Retention policy' and 'General'. The 'Retention policy' section includes fields for 'Days to keep artifacts, symbols and attachments' (set to 30), 'Days to keep runs' (set to 30), 'Days to keep pull request runs' (set to 10), and 'Number of recent runs to retain per pipeline' (set to 3). A warning message states: 'The artifacts, symbols and attachments retention setting is being ignored because the runs retention setting is evaluated first.' The 'General' section contains several toggle switches: 'Disable anonymous access to badges' (off), 'Limit variables that can be set at queue time' (off), 'Limit job authorization scope to current project for non-release pipelines' (off), 'Publish metadata from pipelines (preview)' (off), and 'Limit job authorization scope to referenced Azure DevOps repositories' (on).

## Other automated test results

To clean up automated test results that are left over from deleted builds or test results that aren't related to builds, for example, results published from external test systems, set the retention limits at the project level as shown in the [Manual test-runs retention policies](#)

## Set artifact retention policies

You can set artifact retention policies for pipeline runs in the Pipeline settings.

1. Sign in to your project (<https://dev.azure.com/{yourorganization}/{yourproject}>).
2. Go to on the **Settings** tab of your project's settings.
3. Select **Settings** in **Pipelines**.
4. Edit **Days to keep artifacts, symbols, and attachments**.

## Use the Copy Files task to save data longer

You can use the [Copy Files task](#) to save your build and artifact data for longer than what is set in the retention policies. The [Copy Files task](#) is preferable to the [Publish Build Artifacts task](#) because data saved with the [Publish Build Artifacts task](#) will get periodically cleaned up and deleted.

- [YAML](#)
- [Classic](#)

```
- task: CopyFiles@2
 displayName: 'Copy Files to: \\mypath\storage\$(Build.BuildNumber)'
 inputs:
 SourceFolder: '$(Build.SourcesDirectory)'
 Contents: '_buildOutput/**'
 TargetFolder: '\\mypath\storage\$(Build.BuildNumber)'
```

You can also customize these policies on a branch-by-branch basis if you are building from [Git repositories](#).

## Global build retention policy

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).

- TFS 2017 and newer: [https://{your\\_server}/tfss/DefaultCollection/\\_admin/\\_buildQueue](https://{your_server}/tfss/DefaultCollection/_admin/_buildQueue)
- TFS 2015.3: [http://{your\\_server}:8080/tfs/DefaultCollection/\\_admin/\\_buildQueue](http://{your_server}:8080/tfs/DefaultCollection/_admin/_buildQueue)
- TFS 2015 RTM: [http://{your\\_server}:8080/tfs/DefaultCollection/\\_admin/\\_buildQueue#a=settings](http://{your_server}:8080/tfs/DefaultCollection/_admin/_buildQueue#a=settings)

The **maximum retention policy** sets the upper limit for how long runs 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 **Permanently destroy releases** helps you keep the runs 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.
- Other/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.
- Main 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:

## Definitions / Our build | Builds

Build Options Repository Variables Triggers General **Retention** History

Save Queue build... Undo

Add new rule...

Three retention policies are listed:

- Policy 1:** Days to keep: 5, Minimum to keep: 1. Delete build record:  Delete source label:  Delete test results:  Branch filters: Include  users/\* Add new filter
- Policy 2:** 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 Add new filter
- Policy 3:** Days to keep: 15, Minimum to keep: 1. Delete build record:  Delete source label:  Delete test results:  Branch filters: Include  \* Add new filter

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 0 builds with the following branch filter:

refs/pull/\*

Days to keep: 10  
Minimum to keep: 0

Delete build record:   
Delete source label:   
Delete file share:   
Delete symbols:   
Delete test results:

Branch filters: Include  refs/pull/\* Add new filter

### 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.

<input checked="" type="checkbox"/> Days to keep: 30	Delete build record: true	Branch filters: All
Minimum to keep: 10	Delete source label: true	
	Delete test results: true	

## FAQ

### If I mark a run 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 run or release to be retained indefinitely. It will remain until you stop retaining it indefinitely.

### How do I specify that runs deployed to production will be retained longer?

If you use classic releases to deploy to production, then 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 runs associated with that release are to be retained. This will override the run retention policy.

If you use multi-stage YAML pipelines to production, the only retention policy you can configure is in the project settings. You cannot customize retention based on the environment to which the build is deployed.

### I did not mark runs to be retained indefinitely. However, I see a large number of runs being retained. How can I prevent this?

This could be for one of the following reasons:

- The runs are marked by someone in your project to be retained indefinitely.
- The runs are consumed by a release, and the release holds a retention lock on these runs. Customize the release retention policy as explained above.

If you believe that the runs are no longer needed or if the releases have already been deleted, then you can manually delete the runs.

### How does 'minimum releases to keep' setting work?

Minimum releases to keep are defined at stage level. It denotes that Azure DevOps will always retain the given number of last deployed releases for a stage even if the releases are out of retention period. A release will be considered under minimum releases to keep for a stage only when the deployment started on that stage. Both successful and failed deployments are considered. Releases pending approval are not considered.

### How is retention period decided when release is deployed to multiple stages having different retention period?

Final retention period is decided by considering days to retain settings of all the stages on which release is deployed and taking max days to keep among them. Minimum releases to keep is governed at stage level and do not change based on release deployed to multiple stages or not. Retain associated artifacts will be applicable when release is deployed to a stage for which it is set true.

### I deleted a stage for which I have some old releases. What retention will be considered for this case?

As the stage is deleted, so the stage level retention settings are not applicable now. Azure DevOps will fall back to project level default retention for such case.

### My organization requires us to retain builds and releases longer than what is allowed in the settings. How can I request a longer retention?

The only way to retain a run or a release longer than what is allowed through retention settings is to manually mark it to be retained indefinitely. There is no way to configure a longer retention setting. You can also explore the possibility of using the REST APIs in order to download information and artifacts about the runs and upload them to your own storage or artifact repository.

### I lost some runs. Is there a way to get them back?

If you believe that you have lost runs due to a bug in the service, create a support ticket immediately to recover the lost information. If the runs were manually deleted more than a week earlier, it isn't possible to recover the lost runs. If the runs were deleted as expected due to a retention policy, it isn't possible to recover the lost runs.

### **How do I use the `Build.Cleanup` capability of agents?**

Setting a `Build.Cleanup` capability on agents will cause the pool's cleanup jobs to be directed to just those agents, leaving the rest free to do regular work. When a pipeline run is deleted, artifacts stored outside of Azure DevOps are cleaned up through a job run on the agents. When the agent pool gets saturated with cleanup jobs, this can cause a problem. The solution to that is to designate a subset of agents in the pool that are the cleanup agents. If any agents have `Build.Cleanup` set, only those agents will run the cleanup jobs, leaving the rest of the agents free to continue running pipeline jobs.

### **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 retained as specified by the retention policy configured for the test results. The test results do not get retained until the release is retained. If you need the test results as long as the release, set the retention settings for automated test runs in the Project settings accordingly to Never delete. This makes sure the test results are deleted only when the release is deleted.

### **Are manual test results deleted?**

No. Manual test results are not deleted.

### **How do I preserve my version control labels?**

Version control labels created during a build will be deleted when your build is deleted. If you need to preserve version control labels, you'll need to retain any associated builds.

## Related articles

- [Control how long to keep test results](#)
- [Delete test artifacts](#)

# Configure and pay for parallel jobs

6/8/2021 • 15 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

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.

## IMPORTANT

Starting with Azure DevOps Server 2019, you do not have to pay for self-hosted concurrent jobs in releases. You are only limited by the number of agents that you have.

Learn how to estimate how many parallel jobs you need and buy more parallel jobs for your organization.

## NOTE

We have temporarily disabled the free grant of parallel jobs for public projects and for certain private projects in new organizations. However, you can request this grant by submitting [a request](#). Existing organizations and projects are not affected. Please note that it takes us 2-3 business days to respond to your free tier requests.

## What is a parallel job?

When you define a pipeline, you can define it as a collection of [jobs](#). When a pipeline runs, you can run multiple jobs as part of that pipeline. Each running job consumes a *parallel job* that runs on an agent. When there aren't enough parallel jobs available for your organization, the jobs are queued up and run one after the other.

In Azure Pipelines, you can run parallel jobs on Microsoft-hosted infrastructure or your own (self-hosted) infrastructure. Each parallel job allows you to run a single job at a time in your organization. You do not need to pay for parallel jobs if you are using an on-premises server. The concept of parallel jobs only applies to Azure DevOps Services.

### Microsoft-hosted vs. self-hosted parallel jobs

If you want to run your jobs on machines that Microsoft manages, use *Microsoft-hosted parallel jobs*. Your jobs will run on [Microsoft-hosted agents](#).

If you want Azure Pipelines to orchestrate your builds and releases, but use your own machines to run them, use *self-hosted parallel jobs*. For self-hosted parallel jobs, you'll start by deploying our [self-hosted agents](#) on your machines. You can register any number of these self-hosted agents in your organization.

## How much do parallel jobs cost?

We provide a *free tier* of service by default in every organization for both hosted and self-hosted parallel jobs. Parallel jobs are purchased at the organization level, and they are shared by all projects in an organization.

- [Microsoft-hosted](#)
- [Self-hosted](#)

For Microsoft-hosted parallel jobs, you can get up to 10 free Microsoft-hosted parallel jobs that can run for up to 360 minutes (6 hours) each time for **public projects**. When you create a new Azure DevOps organization, you are not given this free grant by default.

For **private projects**, you can get one free job that can run for up to 60 minutes each time. When you create a new Azure DevOps organization you may not always be given this free grant by default.

To request the free grant for public or private projects, submit [a request](#).

**NOTE**

It takes us 2-3 business days to respond to your free tier request.

There is no time limit on parallel jobs for public projects and a 30 hour time limit per month for private projects.

	NUMBER OF PARALLEL JOBS	TIME LIMIT
Public project	Up to 10 free Microsoft-hosted parallel jobs that can run for up to 360 minutes (6 hours) each time	No overall time limit per month
Private project	One free job that can run for up to 60 minutes each time	1,800 minutes (30 hours) per month

When the free tier is no longer sufficient, you can pay for additional capacity per parallel job. For pricing cost per parallel job, see the [Azure DevOps pricing page](#). 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](#).

When you purchase your first Microsoft-hosted parallel job, the number of parallel jobs you have in the organization is still one. To be able to run two jobs concurrently, you will need to purchase two parallel jobs if you are currently on the free tier. The first purchase only removes the time limits on the first job.

**TIP**

If your pipeline exceeds the maximum job timeout, try splitting your pipeline into multiple jobs. For more information on jobs, see [Specify jobs in your pipeline](#).

Do I need parallel jobs in TFS 2015? Short answer: no. [More details](#)

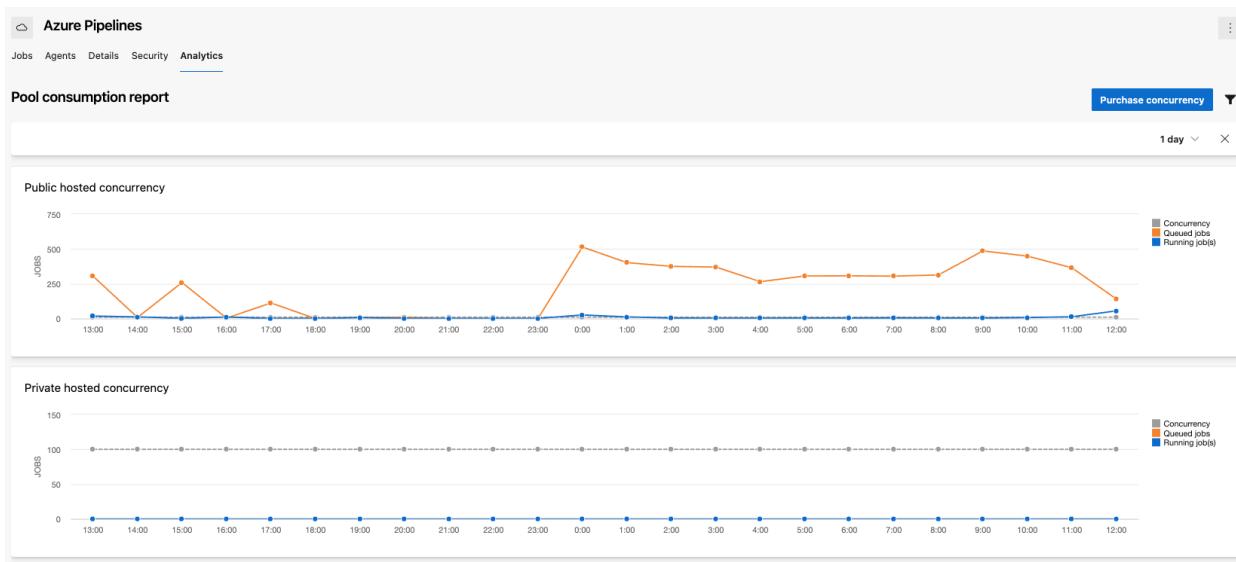
## How many parallel jobs do I need?

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. There are several methods you can use to check your parallel job limits and job history.

### View job history using the pool consumption report

You can use the **Pool consumption report**, available on the **Analytics** tab of your agent pool, to see a chart of

running and queued jobs graphed with your parallel jobs for the previous 30 days. If you have a backlog of queued jobs and your running jobs are at the concurrency limit, you may wish to purchase more parallel jobs. For more information, see [Pool consumption report](#).



## Check the parallel jobs setting directly

Figure out how many parallel jobs you need by first seeing how many parallel jobs your organization currently uses:

1. Browse to [Organization settings > Pipelines > Retention and parallel jobs > Parallel jobs](#).

Organization Settings > Retention and parallel jobs

The screenshot shows the 'Parallel jobs' tab of the 'Retention and parallel jobs' section. On the left is a navigation menu with 'General', 'Work', 'Pipelines' (selected), 'Agent pools', 'Deployment pools', 'Retention and parallel jobs' (selected), and 'OAuth configurations'. The main area displays two sections: 'Microsoft-hosted' and 'Self-hosted'. Under Microsoft-hosted, it says 'Currently 0/1800 minutes are consumed' and has a 'Purchase parallel jobs' button. Under Self-hosted, it shows 2 parallel jobs, with details: 'Free parallel jobs' (1), 'Visual Studio Enterprise subscribers' (1), and 'Monthly purchases' (0).

URL example: [https://{{your\\_organization}}/\\_admin/\\_buildQueue?a=resourceLimits](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.

## Estimate costs

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

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

- If you have multiple teams, and if each of them require CI, you'll likely need a parallel job for each team.
- If your CI 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.

## How do I buy more parallel jobs?

To buy more parallel jobs:

- [Billing must be set up for your organization](#)
- You need [Project Collection Administrator](#) or [organization Owner](#) permissions

### Buy parallel jobs

Buy more parallel jobs within your organization settings:

1. Sign in to your organization (<https://dev.azure.com/{yourorganization}>).
2. Select [Organization settings](#).

The screenshot shows the Azure DevOps organization settings interface. On the left, there's a sidebar with a list of organizations: 'FabrikamFiber01' (selected and highlighted in blue), 'fabrikamfiber02', 'fabrikamffiber', and '14 more organizations'. Below this is a link to 'New organization'. A 'What's new' section highlights 'Sprint 162 release notes' and mentions the return of the Sprint Burndown. At the bottom of the sidebar, a red box highlights the 'Organization settings' link. The main content area is titled 'FabrikamFiber01' and shows a project named 'Fabrikam Fiber' with a thumbnail icon. Navigation tabs at the top of the main area include 'Projects', 'My work items', and 'My pull requests'.

3. Select **Parallel jobs** under Pipelines, and then select either **Purchase parallel jobs** for Microsoft-hosted jobs or **Change** for self-hosted jobs.

The screenshot shows the Azure DevOps Settings page for the organization 'Fabrikam'. The left sidebar contains navigation links for General, Security, Boards, Pipelines, and Artifacts. Under Pipelines, the 'Parallel jobs' link is highlighted with a red box. The main content area displays 'Private projects' and 'Public projects' sections. In the Private projects section, there are two rows: 'Microsoft-hosted' (Free tier, 1 parallel job up to 49999 mins/mo) and 'Self-hosted' (9999 Parallel jobs). A red box highlights the 'Purchase parallel jobs' button next to the Microsoft-hosted row. In the Public projects section, there are two rows: 'Microsoft-hosted' (10 Parallel jobs) and 'Self-hosted' (Unlimited Parallel jobs). A red box highlights the 'Change' button next to the Self-hosted row.

4. Enter your desired amount, and then **Save**.

For pricing cost per parallel job, see the [Azure DevOps pricing page](#).

## How do I change the quantity of parallel jobs for my organization?

1. Sign in to your organization (<https://dev.azure.com/{yourorganization}>).

2. Select **Organization settings**.

The screenshot shows the Azure DevOps Home page for the organization 'FabrikamFiber01'. The left sidebar lists organizations: 'FabrikamFiber01' (selected), 'fabrikamfiber02', and 'fabrikamfiber'. Below these are links for '14 more organizations' and 'New organization'. A 'What's new' section highlights the 'Sprint 162 release notes' with a message about the sprint burndown. At the bottom of the sidebar, the 'Organization settings' link is highlighted with a red box. The main content area displays the organization's logo ('FF') and name 'Fabrikam Fiber'.

3. Select **Parallel jobs** under Pipelines, and then select either **Purchase parallel jobs** or **Change** for Microsoft-hosted jobs or **Change** for self-hosted jobs.

The screenshot shows the Azure DevOps Settings page for the 'Fabrikam' organization. On the left, there's a navigation sidebar with sections like General, Security, Boards, Pipelines, and Artifacts. Under Pipelines, 'Parallel jobs' is highlighted with a red box. In the main content area, there are two sections: 'Private projects' and 'Public projects'. The 'Private projects' section shows the 'Free tier' for Microsoft-hosted parallel jobs (1 job up to 49999 mins/mo) and a 'Purchase parallel jobs' button. It also lists self-hosted parallel jobs (9999) and Visual Studio Enterprise subscribers (0). A 'Change' button is highlighted with a red box. The 'Public projects' section shows Microsoft-hosted parallel jobs (10) and self-hosted parallel jobs (Unlimited).

4. Enter a lesser or greater quantity of Microsoft-hosted or self-hosted jobs, and then select **Save**.

#### IMPORTANT

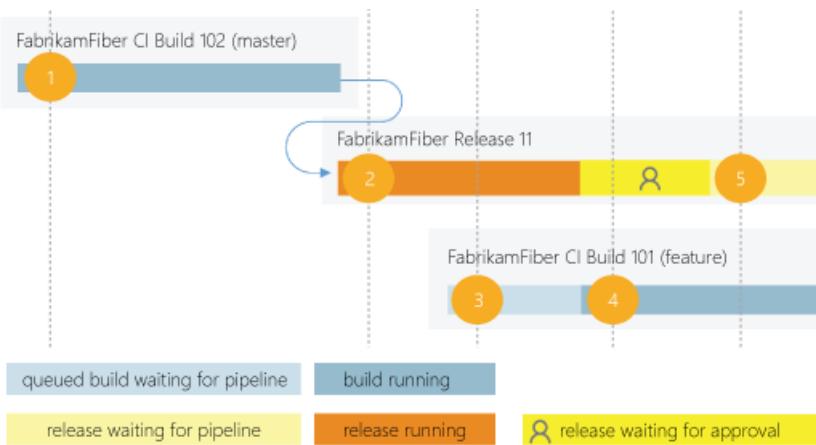
Hosted XAML build controller isn't supported. If you have an organization where you need to run [XAML builds](#), set up an [on-premises build server](#) and switch to an [on-premises build controller](#). For more information about the hosted XAML model, see [Get started with XAML](#).

## How is a parallel job consumed in DevOps Services?

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

If you use release or YAML pipelines, then a run 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.

When you run a [server job](#) or deploy to a [deployment group](#) using release pipelines, you don't consume any parallel jobs.

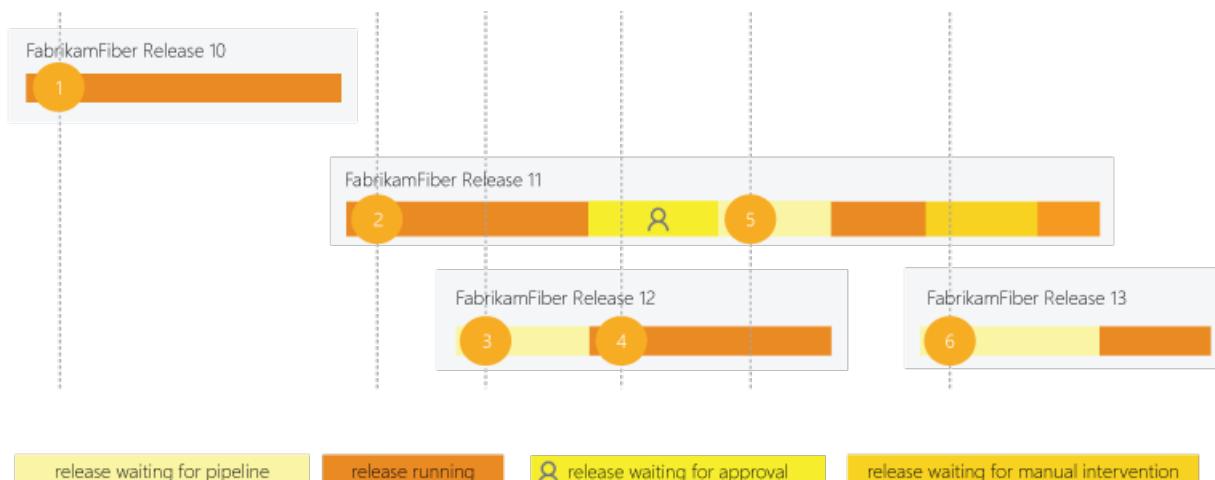


1. FabrikamFiber CI Build 102 (main branch) starts first.
2. Deployment of FabrikamFiber Release 11 is triggered by completion of FabrikamFiber CI Build 102.
3. FabrikamFiber CI Build 101 (feature branch) is triggered. The build can't start yet because Release 11's deployment is active. So the build 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.

## How is a parallel job 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.



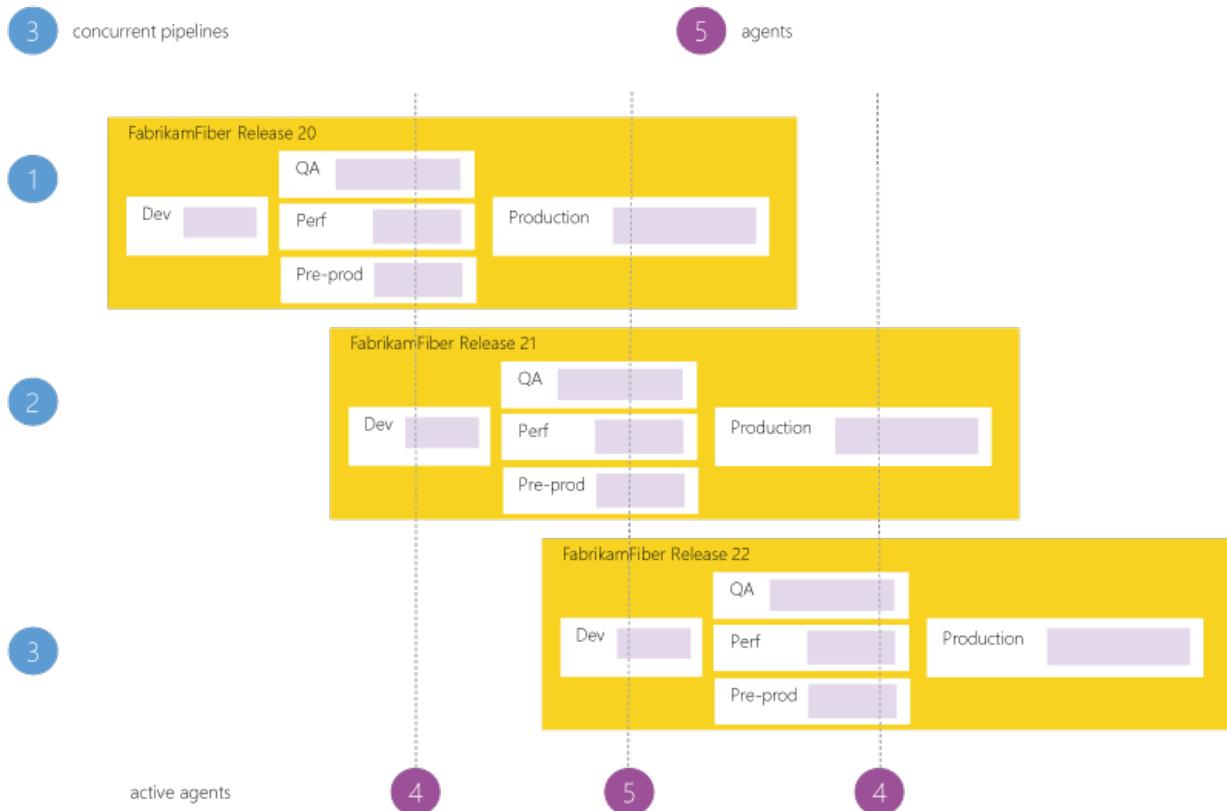
1. FabrikamFiber Release 10 is first to be deployed.
2. Deployment of FabrikamFiber Release 11 starts after Release 10's deployment is complete.
3. Release 12 is queued until Release 11's deployment is active.
4. Release 11 waits for an approval. Release 12's deployment starts because a release waiting 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

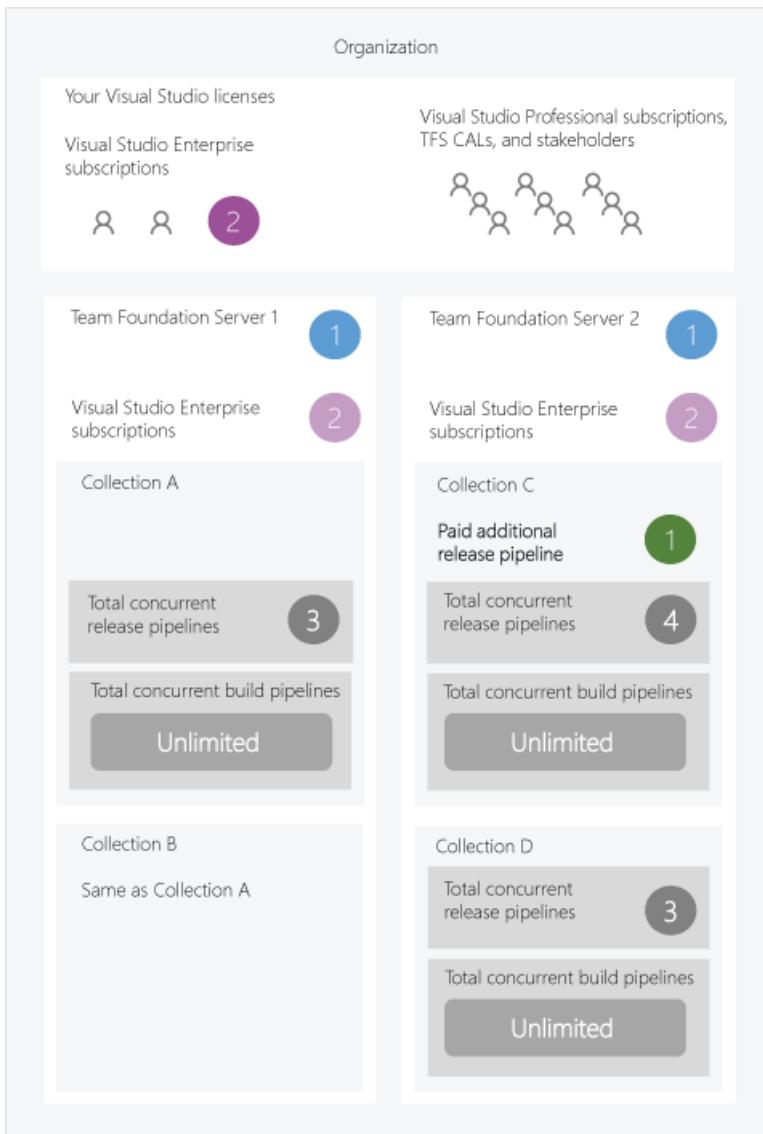
Parallel processing within a single release does not require additional parallel jobs. So long as you have enough agents, you can deploy to multiple stages in a release at the same time.

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 the number of 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' dropdown, 'Home' and 'Rooms' links, and a gear icon for settings. Below the main bar, there's a secondary navigation bar with three tabs: 'Control panel', 'Access levels' (which is underlined to indicate it's selected), 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' with 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 tabs: 'Overview', 'Users', 'Security', 'Build and Release' (which is underlined to indicate it's selected), 'Agent pools', 'Extensions', 'Settings', and 'Resource limits'.

URL example: `http://{your_server}:8080/tfs/DefaultCollection/_admin/_buildQueue?_a=resourceLimits`

2. View or edit the number of purchased parallel jobs.

## FAQ

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

You qualify for 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.

For information on how to apply for the grant of free parallel jobs, see [How much do parallel jobs cost \(Microsoft-hosted\)?](#)

### Can I assign a parallel job to a specific project or agent pool?

Currently, there isn't a way to partition or dedicate parallel job capacity to a specific project or agent pool. For example:

- You purchase two parallel jobs in your organization.
- You start two runs in the first project, and both the parallel jobs are consumed.
- You start a run in the second project. That run won't start until one of the runs in your first project is completed.

### 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 pipelines 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.

### **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.

## **Related articles**

- [Set up billing](#)
- [Manage paid access](#)
- [Buy access to test hub](#)
- [Add user for billing management](#)
- [Azure DevOps billing overview](#)

# Set different levels of pipeline permissions

3/11/2021 • 15 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Pipeline permissions are the permissions associated with pipelines in an Azure DevOps project. Permissions in Azure DevOps are hierarchical and can be set at the organization, server (for on-premises), project, and object levels.

Object-level permissions are designed to be more granular than organization-level permissions. For example, a user could have access to your Azure repository thanks to their organization-level permissions. However, that same user could be prevented from running a pipeline manually because of that pipeline's permissions.

You can increase the security of your pipeline by fine-tuning the object-level permissions associated with your pipeline. To learn more about best practices for pipeline security, see [Security Azure Pipelines](#).

To learn more about how Azure DevOps permissions work overall, including how the permissions hierarchy, see [Get started with permissions, access, and security groups](#).

## Prerequisites

- To manage permissions for project-level groups, you'll need to be a **Project Administrator**. Learn more about [project-level group permissions](#). You'll need to be a member of the Project Administrators group to [add users to Azure Pipelines](#).
- To manage permissions for collection groups, you'll need to be a **Project Collection Administrator**. [Learn more about collection-level group permissions](#).

## Default permissions assigned to built-in security groups

### Build

TASK	STAKEHOLDERS	READERS	CONTRIBUTORS	BUILD ADMINS	PROJECT ADMINS
View builds	✓	✓	✓	✓	✓
View build pipeline	✓	✓	✓	✓	✓
Administer build permissions				✓	✓
Delete or Edit build pipeline			✓	✓	✓

Delete or Destroy builds				✓	✓
Edit build quality			✓	✓	✓
Manage build qualities				✓	✓
Manage build queue				✓	✓
Override check-in validation by build					✓
Queue builds			✓	✓	✓
Retain indefinitely				✓	✓
Stop builds				✓	✓
Update build information					✓

## Release

TASK	STAKEHOLDERS	READERS	CONTRIBUTORS	PROJECT ADMINS	RELEASE ADMINS
Approve releases	✓		✓	✓	✓
View releases	✓	✓	✓	✓	✓
View release pipeline		✓	✓	✓	✓
Administer release permissions				✓	✓
Delete release pipeline or release stage			✓	✓	✓
Delete releases			✓	✓	✓
Edit release pipeline				✓	✓
Edit release stage			✓	✓	✓
Manage deployments				✓	✓

Manage release approvers			✓	✓	✓
Manage releases				✓	✓

## Task groups

TASK	STAKEHOLDERS	READERS	CONTRIBUTORS	BUILD ADMINS	PROJECT ADMINS	RELEASE ADMINS
Administer task group permissions				✓	✓	✓
Delete task group				✓	✓	✓
Edit task group				✓	✓	✓

## Build

TASK	STAKEHOLDERS	READERS	CONTRIBUTORS	BUILD ADMINS	PROJECT ADMINS
View builds		✓	✓	✓	✓
View build definition		✓	✓	✓	✓
Administer build permissions				✓	✓
Delete or Edit build definitions			✓	✓	✓
Delete or Destroy builds				✓	✓
Edit build quality			✓	✓	✓
Manage build qualities				✓	✓
Manage build queue				✓	✓
Override check-in validation by build					✓
Queue builds			✓	✓	✓
Retain indefinitely				✓	✓

Stop builds				✓	✓
Update build information					✓

## Release

TASK	STAKEHOLDERS	READERS	CONTRIBUTORS	PROJECT ADMINS	RELEASE ADMINS
Approve releases	✓		✓	✓	✓
View releases	✓	✓	✓	✓	✓
View release definition		✓	✓	✓	✓
Administer release permissions				✓	✓
Delete release definition or release stage			✓	✓	✓
Delete releases			✓	✓	✓
Edit release definition				✓	✓
Edit release stage			✓	✓	✓
Manage deployments				✓	✓
Manage release approvers			✓	✓	✓
Manage releases				✓	✓

## Set pipeline permissions

You can update pipeline permissions with security groups or by adding individual users. To learn how to add a user to Azure Pipelines, see [Add users to Azure Pipelines](#).

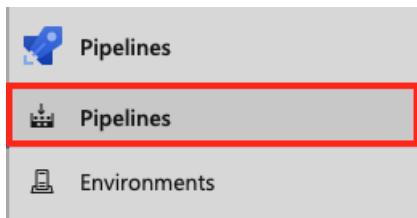
When it comes to security, there are different best practices and levels of permissiveness.

- In many cases, you probably also want to set **Delete 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.

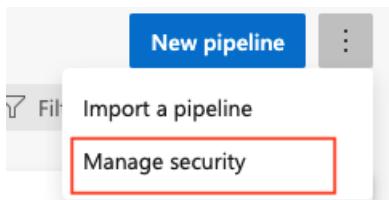
## Update pipeline permissions at the project-level

To set the permissions at project level for all pipelines, choose **Manage security** from contextual menu for all pipelines.

1. In your project, go to **Pipelines > Pipelines**.



2. Select **Manage security** from More actions .



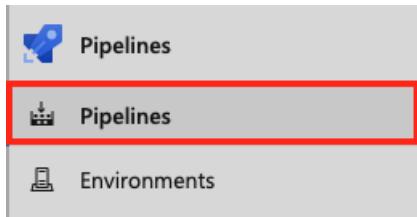
3. Modify the permissions associated with an [Azure DevOps group](#) (example: Build Administrators) or [individual user](#).

4. Set permissions by selecting **Allow** or **Deny** for the permission for a security group or an individual user.

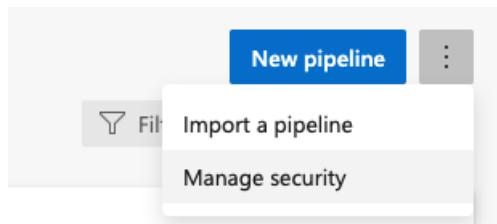
## Update pipeline permissions for one pipeline

To set or override the permissions for a specific pipeline, choose **Security** from the context menu of the individual pipeline.

1. In your project, go to **Pipelines > Pipelines**.



2. Open an individual pipeline.



Build and YAML pipeline permissions 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 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 pipeline, choose **Security** from the context menu of the pipeline.

The following permissions are defined for pipelines. All of these can be set at both the levels.

## Pipeline permissions reference

These permissions are defined for pipelines. All of these permissions can be set for both all pipelines in a project or for an individual pipeline.

To learn more about how permissions are set, including inheritance, see [About permissions and inheritance](#). To learn how inheritance is supported for role-based membership, see [About security roles](#)

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 <b>retained</b> in the <b>Deleted</b> tab for a period of time before they are destroyed.
Destroy builds	Can delete builds from the <b>Deleted</b> tab.
Edit build pipeline	Can create pipelines and 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 <a href="#">TFVC gated check-in builds</a> . 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 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.

Once you've been added as a team member, you're a member of the Contributors group. This group permission allows you to define and manage builds and releases. The most common built-in groups include Readers, Contributors, and Project Administrators.

For more information on default permissions, see [Default permissions and access quick reference](#).

# Set release permissions

Permissions for release pipelines 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.

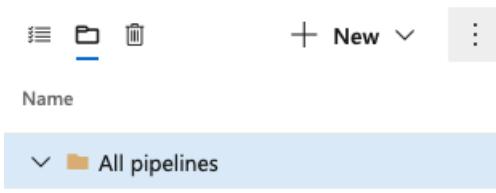
## Update pipeline permissions for all releases

To update permissions for all releases, open **Releases** in the **Pipelines** tab.

1. Select the file view.



2. Select the **All pipelines** folder.



3. Open More actions and select **Security**.

## Update pipeline permissions for one release

To update permissions for one release, first open **Releases** in the **Pipelines** tab.

1. Select the release you want to modify.

2. Open More actions and select **Security**.

To set permissions at project level for all release definitions in a project, open the shortcut menu from the

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 **More actions** on a stage in the release pipeline editor.

## Release permissions reference

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
Administer release permissions	Can change any of the other permissions listed here. <i>Scopes:</i> Project, Release pipeline, Stage
Create releases	Can create new releases. <i>Scopes:</i> Project, Release pipeline
Delete release pipeline	Can delete release pipeline(s). <i>Scopes:</i> Project, Release pipeline
Delete release stage	Can delete stage(s) in release pipeline(s). <i>Scopes:</i> Project, Release pipeline, Stage

Permission	Description
<b>Delete releases</b>	Can delete releases for a pipeline. <i>Scopes:</i> Project, Release pipeline
<b>Edit release pipeline</b>	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 <b>Edit release stage</b> permission. <i>Scopes:</i> Project, Release pipeline
<b>Edit release stage</b>	Can edit stage(s) in release pipeline(s). To save the changes to the release pipeline, the user also needs <b>Edit release pipeline</b> permission. This permission also controls whether a user can edit the configuration inside the stage of a specific release instance. The user also needs <b>Manage releases</b> permission to save the modified release. <i>Scopes:</i> Project, Release pipeline, Stage
<b>Manage deployments</b>	Can initiate a deployment of a release to a stage. This permission is only for deployments that are manually initiated by selecting the <b>Deploy</b> or <b>Redeploy</b> 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. If the condition is set to start after some stage, manually initiated deployments do not wait for those stages to be successful. <i>Scopes:</i> Project, Release pipeline, Stage
<b>Manage release approvers</b>	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. <i>Scopes:</i> Project, Release pipeline, Stage
<b>Manage releases</b>	Can edit the configuration in releases. To edit the configuration of a specific stage in a release instance (including variables marked as <code>settable at release time</code> ), the user also needs <b>Edit release stage</b> permission. <i>Scopes:</i> Project, Release pipeline
<b>View release pipeline</b>	Can view release pipeline(s). <i>Scopes:</i> Project, Release pipeline
<b>View releases</b>	Can view releases belonging to release pipeline(s). <i>Scopes:</i> 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**.

## Set task group permissions

You can use task groups to combine a sequence of tasks already defined in a pipeline into a single, reusable task.

[Task groups](#) are defined in the [Task groups](#) tab for [Azure Pipelines](#).

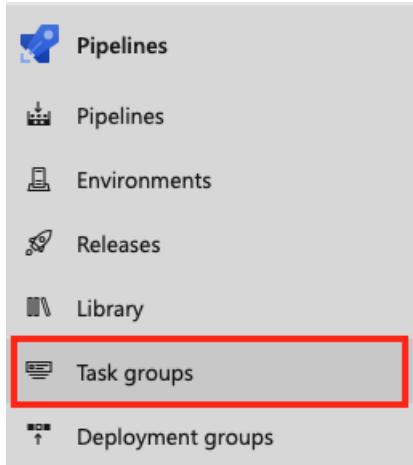
Task group permissions follow a hierarchical model. Defaults for all the permissions can be set at the project level and can be overridden on an individual task group pipeline.

## Set task group permissions at the project-level

### NOTE

Task groups are not supported in YAML pipelines. Instead, in that case you can use templates. See [YAML schema reference](#).

1. Open [Pipelines > Task groups](#) in your project.



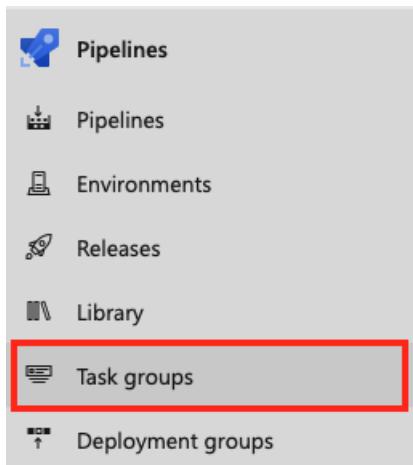
2. Select [Security](#).



3. Set permissions by selecting [Allow](#) or [Deny](#) for the permission for a security group or an individual user.

## Set task group permissions at the pipeline-level

1. Open [Pipelines > Task groups](#) in your project.



2. Select a task group.

3. Select [More actions](#) to open the [Security](#) menu.

4. Set permissions by selecting [Allow](#) or [Deny](#) for the permission for a security group or an individual user.

## Task group permissions reference

PERMISSION	DESCRIPTION
Administer task group permissions	Can add and remove users or groups to task group security.
Delete task group	Can delete a task group.
Edit task group	Can create, modify, or delete a task group.

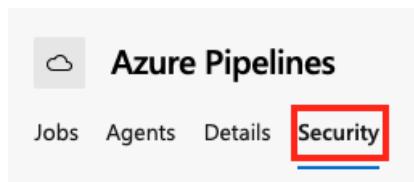
## Set agent pool permissions

You can use pre-defined roles to configure [security on agent pools](#). You can configure this in a hierarchical manner either for all pools, or for an individual pool.

To configure agent pool permissions, open **Settings**  . Select **Security** to configure security settings for all agent pools.



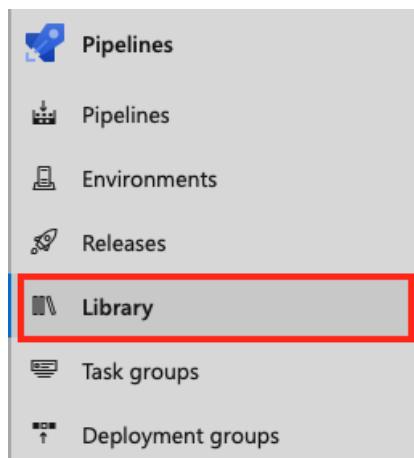
To configure permissions for one agent pool, select the agent. Then, select the **Security** tab.



## Set library permissions

Permissions for library artifacts, such as variable groups and secure files, are managed by roles. You can use a variable group to store values that you want to make available across multiple build and release pipelines. Roles are also defined to help you configure security on shared [library entities](#) such as [variable groups](#). You can configure these roles hierarchically.

To [define and manage variable groups](#) and [secure files](#), open the **Library** tab in [Azure Pipelines](#).



You can configure Library object permissions for everything in your library or for an individual variable group or secure file. Select the **Security** option in the navigation for all of your variable groups and secure files or for one instance.



Variable group



Security



Help

## Library permissions reference

ROLE	DESCRIPTION
Administrator	Can edit/delete and manage security for library items.
Creator	Can create library items.
Reader	Can only read library items.
User	Can consume library items in pipelines.

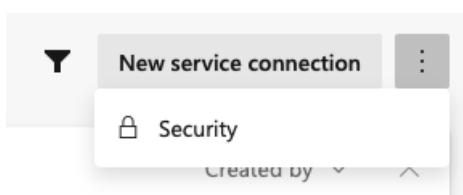
## Set service connection permissions

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](#).

To configure [service connection](#) permissions, open [Settings](#) . Select [Service connections](#) in [Pipelines](#) to configure service connections.

You can configure permissions in a hierarchical manner either for all service connections, or for an individual connection.

To configure permissions for all service connections, select **Security** from **More actions** .



To configure permissions for one service connection, open the service connection and select **Security** from **More actions** within an individual service connection.

If you're having trouble with permissions and service connections, see [Troubleshoot Azure Resource Manager service connections](#).

## Service connection permissions reference

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.

## Set deployment pool permissions

You can set default security roles for all [deployment groups](#) and manage security roles for an individual deployment group.

To set default deployment group permissions, open **Deployment groups** in the **Pipelines** tab. Then, select **Security**.

## Deployment groups

Groups Available pools 7 | + New **Security** Help

To set permissions for a specific deployment group, select the deployment group. On the deployment group page, select **Security**.

### Deployment pool permissions reference

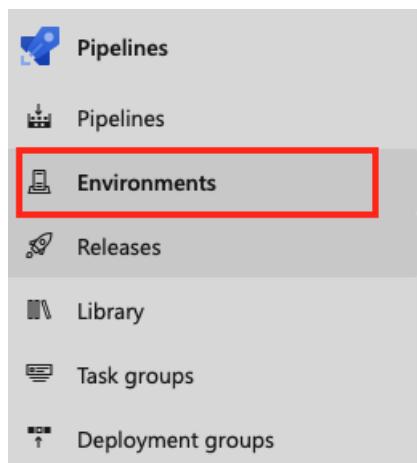
ROLE	DESCRIPTION
Reader	Can only view deployment pools.
Service Account	Can view agents, create sessions, and listen for jobs from the agent pool.
User	Can view and use the deployment pool for creating deployment groups.
Administrator	Can administer, manage, view and use deployment pools.

## Set environment permissions

You can use roles and user permissions to control who [can create, view, and manage environments](#).

Environment roles are hierarchical and can be applied to all environments or one environment.

To change environment permissions, open **Environments** in the **Pipelines** tab.



Select **Security** from **More actions** to change permissions for all environment.

To change the permission for one environment, select the environment. Then, select **Security** from **More actions** .

### Environment permissions reference

#### NOTE

When you create an environment in a YAML, contributors and project administrators will be granted the administrator role. When you create an environment through the UI, only the creator will have the administrator role.

ROLE	DESCRIPTION
Creator	Global role, available from environments hub security option. Members of this role can create the environment in the project. Contributors are added as members by default. Not applicable for environments auto created from YAML pipeline.
Reader	Members of this role can view the environment.
User	Members of this role can use the environment when authoring yaml pipelines.
Administrator	In addition to using the environment, members of this role can manage membership of all other roles for the environment. Creators are added as members by default.

## Related notes

- [Set build and release permissions](#)
- [Default permissions and access](#)
- [Permissions and groups reference](#)
- [Troubleshoot Azure Resource Manager service connections](#)

## FAQ

### Why can't I create a new pipeline?

You need edit build pipeline permissions to create a new pipeline. To add the permission, open the security settings for all pipelines and verify that **Edit build pipeline** is set to *Allow* for your security group.

If you're still unable to create a pipeline after updating this permission, check to see if your [access level](#) is set to **Stakeholder**. If you have stakeholder access, change your access to **Basic**.

### Why do I see the message that I need to authorize a resource before the run can continue?

You'll need to authorize resources before you can use them. The exception to this is that when you create a pipeline for the first time, all the resources that are referenced in the YAML file are automatically authorized. The resource are authorized for the pipeline as long as the user running the pipeline has access to the resource.

If you want to authorize **all pipelines** to access a resource like an agent pool, go to the permissions setting for that resource.

For example, if you want all pipelines to be able to use an agent pool, go to **Settings**  and select **Agent Pools in Pipelines**.

Next, select the **Security** tab for a specific agent pool and update permissions to grant access to all pipelines.

Jobs Agents Details **Security**

### Pipeline permissions



Grant access permission to all pipelines

To learn more about resources, see [Resources in YAML](#).

# Add users to Azure Pipelines

6/8/2021 • 4 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

Permissions for build and release pipelines are primarily set at the object-level for a specific build or release, or for select tasks, at the collection level.

You can manage security for different types of resources such as variable groups, secure files, and deployment groups by adding users or groups to that role. Project administrator can grant or restrict access to project resources. If you want to allow a team member to edit pipelines, you must be a project administrator in order to do so.

## Add users to your project

1. Navigate to your project's summary page: <https://dev.azure.com/{your-organization}/{your-project}>
2. Select the **Invite** button to add a user to your project, and then fill out the required fields. Select **Add** when you are done.

The screenshot shows the 'Invite members to FabrikamFiber' dialog box. At the top, there is a breadcrumb navigation: ramiMSFTDevOps / FabrikamFiber / Overview / Summary. To the right are standard navigation icons: a search bar, a three-dot menu, a gear icon, a help icon, and a profile picture. Below the breadcrumb, the project name 'FabrikamFiber' is displayed with a blue 'F' icon. To the right of the project name are 'Private' and 'Invite' buttons; the 'Invite' button is highlighted with a red box. The main area of the dialog box is titled 'Invite members to FabrikamFiber' with a close button 'X'. It contains a search bar with placeholder text 'Search and add users to your FabrikamFiber'. A 'Users' section shows a list box containing 'Fabrikam' with a delete 'X' icon and a plus '+' icon. Below this is a 'Add to team(s)' section with a dropdown menu showing 'FabrikamFiber Team'. A note message states: 'Fabrikam has not been assigned an access level, we will attempt to assign Stakeholder.' with a 'Learn more' link. At the bottom are 'Cancel' and 'Add' buttons, with the 'Add' button also highlighted with a red box.

3. The new user must accept the invitation before they can start creating or modifying pipelines.

## Verify permissions for contributors

### NOTE

A security best practice is to only grant permissions to required users or groups. The **Contributors** group may be too broad in a given project.

To verify the permissions for your project's contributors, make sure you are a member of the *Build Administrators group* or the *Project Administrators group*. See [Set permissions at the project- or collection-level](#) for more details.

- From within your project, select **Pipelines > Pipelines**. Select the All tab, and then select the more actions menu then **Manage security**.

The screenshot shows the 'Pipelines' page in the Azure DevOps interface. On the left sidebar, under 'FabrikamFiber', the 'Pipelines' item is selected and has a red circle with the number '1' above it. In the main area, the 'Pipelines' tab is selected and has a red circle with the number '2' above it. In the top right corner, there is a 'More actions' menu with a red circle and the number '3' above it. The 'Manage security' option is highlighted with a red box.

- On the permissions dialog box, make sure the following **Contributors** permissions are set to Allow.

The screenshot shows the 'Contributors' permissions dialog box. On the left, a tree view lists groups and users: 'Azure DevOps Groups' (Build Administrators, Contributors, Project Administrators, Readers, Project Collection Administrators, Project Collection Build Administrators, Project Collection Build Service Accounts, Project Collection Test Service Accounts), 'Users' (FabrikamFiber Build Service (microsoft)), and 'FabrikamFiber\Contributors'. On the right, a table lists permissions with their current settings:

Permission	Setting
Delete build pipeline	Allow
Delete builds	Allow
Destroy builds	Allow
Edit build pipeline	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 pipeline	Allow
View builds	Allow

Permissions highlighted with a red box include: Delete build pipeline, Delete builds, Destroy builds, Edit build pipeline, Retain indefinitely, Stop builds, and Update build information.

## Set permissions for build pipelines

- From within your project, select **Build and Release**, and then select **Builds** to access your build pipelines.

The screenshot shows the 'Build and Release' navigation bar with the 'Builds' tab selected and highlighted with a red box. Below the navigation bar, the 'Build Definitions' section is visible, showing tabs for 'Mine' (selected), 'Definitions', and 'Queued'.

- Select **Security** to set the permissions for all build pipelines.

The screenshot shows the Microsoft DevOps interface for the 'Builds' section. At the top, there's a navigation bar with 'Fabrikam-Fiber' selected. Below it, tabs for 'Builds', 'Releases', 'Packages', 'Library', 'Task Groups', and 'Deployment Groups' are visible. A search bar says 'Search work items in this project'. On the right, there are icons for 'New', 'Import', and 'Security'. The 'Security' icon is highlighted with a red box.

Below the navigation, a table lists 'Recently built' build definitions: 'Fabrikam-Fiber-CI' and 'Fabrikam-Fiber-ASP.NET-CI'. Each entry has a status column and a '...' button for more options. A context menu is open over the 'Fabrikam-Fiber-CI' entry, with the 'Security...' option highlighted by a red box.

To set permissions for a specific build pipeline, select the context menu for that build and select **Security**.

3. Choose the group you want to set permissions for, and then change the permission setting to grant or restrict access. In the following example, we change the contributors permission to allow editing build definitions.

## Permissions for Fabrikam-Fiber

ACCESS CONTROL SUMMARY  
Shows information about the permissions being granted to this identity

Permission	Setting
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
Manage build qualities	Not set
Manage build queue	Not set
Override check-in validation by build	Not set
Queue builds	Allow
Retain indefinitely	Not set
Stop builds	Not set
Update build information	Not set
View build definition	Allow
View builds	Allow

**Clear explicit permissions**

Remove Save changes Undo changes

Close

4. Select **Save changes** when you are done.

## Set permissions for release pipelines

1. From within your project, select **Build and Release**, and then select **Releases** to access your release pipelines.

Fabrikam-Fiber

Builds Releases Packages Library Task Groups Deployment Groups

Release Management helps you automate the deployment and testing of your software in multiple environments. You can either fully automate the delivery of your software all the way to production, or set up semi-automated processes with approvals and on-demand deployments.

Start by creating a new release definition.

+ New definition Getting started Security

2. Select the context menu for **All release definitions**, and then select **Security**.

3. Choose the group you want to set permissions for, and then change the permission setting to grant or restrict access. In the following example, we change the contributors permission to prohibit the deletion of release definitions.

ACCESS CONTROL SUMMARY	
Shows information about the permissions being granted to this identity	
Administer release permissions	Not set
Create releases	Allow
Delete release definition	Deny
Delete release environment	Allow
Delete releases	Allow
Edit release definition	Allow
Edit release environment	Allow
Manage deployments	Allow
Manage release approvers	Allow
Manage releases	Allow
View release definition	Allow
View releases	Allow

**Clear explicit permissions**

Remove      **Save changes**      Undo changes

Cancel

4. Select **Save changes** when you are done.

## Manage Library roles for variable groups, secure files, and deployment groups

Permissions for variable groups, secure files, and deployment groups are managed by roles. Setting up permissions is similar for all the three different types. **Variable groups** and **Secure files** permissions are configured from **Build and Release > Library** while **Deployment groups** permissions are set from **Build and Release > Deployment groups**.

In the following example, we will configure permissions for variable groups.

- From within your project, select **Build and Release**, and then select **Library** then **Variable groups**.

The screenshot shows the 'Builds' tab selected at the top. Below it, the 'Library' tab is active, indicated by a blue underline. To the right of the tabs is a search bar labeled 'Search variable groups'. A blue button labeled '+ Variable group' is visible. On the far right, a red box highlights a link labeled 'Security'. Below the tabs, there are two links: 'Variable groups' (underlined in blue) and 'Secure files'.

If you want to manage the permissions for a specific variable group, select the ellipsis for that variable group and then select **Security**.

This screenshot shows the same interface as the previous one, but with a context menu open over a row in the table. The row contains the variable group 'FF group', modified by 'Jamal', with a description 'some group here' and last modified 'just now'. The context menu includes options: 'Edit', 'Delete', and 'Security'. The 'Security' option is highlighted with a red box.

- Add the user or group and choose the role you want them to have.

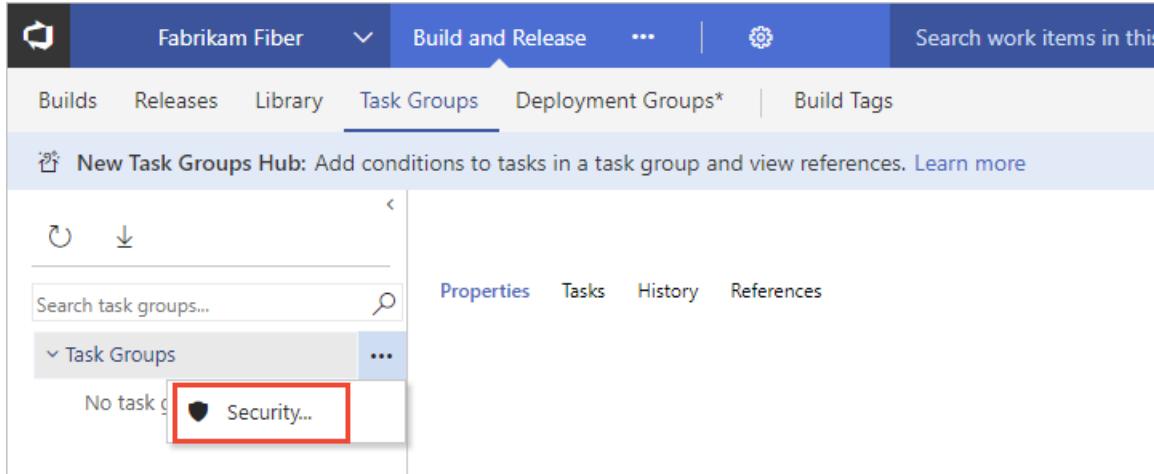
The screenshot shows the 'Assign security roles for Library' dialog. At the top left, a red box highlights the '+ Add' button. A red arrow points from this button to a smaller 'Add user' dialog window. This dialog has fields for 'User or group' (containing 'Raisa Pokrovskaya') and 'Role' (set to 'User'). Below the dialog, a note states: 'User can use, but cannot manage the library items.' At the bottom of the dialog are 'Add' and 'Close' buttons.

- Select **Add** when you are done.

## Manage task group permissions

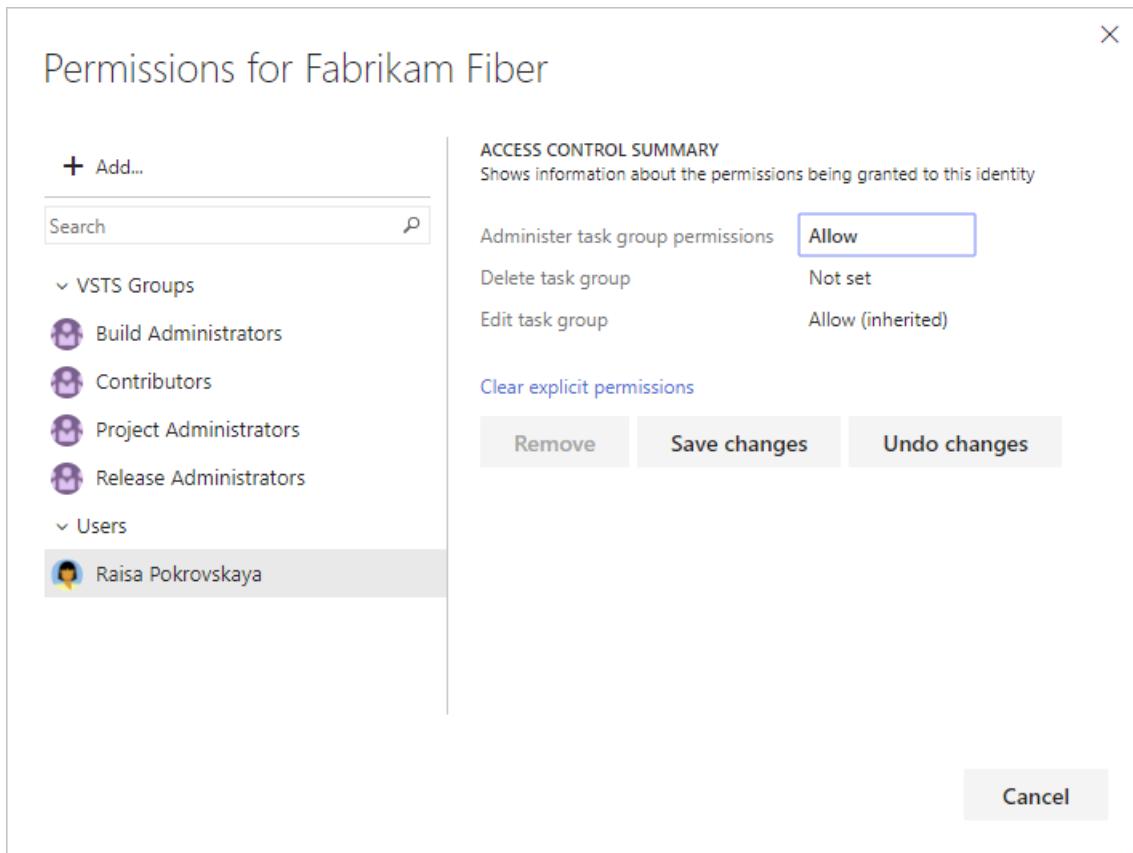
Permissions for task groups are subject to a hierarchical model. You use task groups to encapsulate a sequence of tasks already defined in a build or a release pipeline into a single reusable task.

- From within your project, select **Build and Release**, and then select **Task groups**.



If you want to manage the permissions for a specific task group, select the ellipsis for that task group and then select **Security**.

- Add the user or group and then set the permissions you want them to have.



- Select **Save changes** when you are done.

## Manage permissions for build administrators group

- From within your project, select the gear icon button , and then select **Collection settings**.
- Select **Security**, and then select **Project Collection Build Administrators**. In this example, we want to

allow the usage of build resources.

The screenshot shows the Microsoft VSTS security settings interface. On the left, there's a sidebar with a 'Create group' button and a 'Filter users and groups' input field. Below that is a list of 'VSTS Groups' including 'Freedom Group', 'Project Collection Administrators', 'Project Collection Build Administrators' (which is selected and highlighted in blue), 'Project Collection Build Service Accounts', 'Project Collection Proxy Service Accounts', 'Project Collection Service Accounts', 'Project Collection Test Service Accounts', 'Project Collection Valid Users', 'Readers group', and 'Security Service Group'. The main panel shows the 'fabrikam > Project Collection Build Administrator' group details. It has tabs for 'Permissions', 'Members', and 'Member of'. Under 'Permissions', there's a table of build-related permissions with their current status: 'Allow', 'Not set', or 'Allow (inherited)'. One permission, 'Use build resources', has its 'Allow' button highlighted with a blue border. At the bottom of the permissions table are 'Clear explicit permissions', '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. Select **Save changes** when you are done.

## Manage permissions for service connections

You can set up permissions for service connections or agent pools by adding users or groups to a specific role. You will need to be a member of the Project Administrator group to manage the permissions for these resources.

In the following example, we will add an Administrator to a service connection.

1. From within your project, select the gear icon button , and then select **Project settings**.
2. Select **Services**, and then select the service connection that you want to manage. Select **Roles** to add a new role.

The screenshot shows the 'Endpoints' section of the Azure DevOps Services interface. A service endpoint named 'gitConnect' is selected. The 'Roles' tab is active, displaying a list of users and their assigned roles. One user, 'Raisa Pokrovskaya' from the group '[Fabrikam Fiber]\Endpoint Administrators', is listed with the role 'Administrator'. The 'Inherited' access level is also shown.

3. Add the user or group and choose the role you want them to have.

The screenshot shows the 'Add user' dialog box overlaid on the main interface. The 'User' tab is selected. In the main interface, the 'Add' button is highlighted with a red box and a red arrow points to the 'Add' button in the dialog. The dialog itself shows a search bar for 'User or group' containing 'Raisa Pokrovskaya' and a dropdown for 'Role' set to 'Administrator'. A descriptive message at the bottom states: 'Administrator can use and manage the service endpoint connection.' There are 'Add' and 'Close' buttons at the bottom of the dialog.

4. Select **Add** when you are done.

## Manage permissions for deployment pools

You can set up permissions for deployment pools by adding users or groups to a specific role. You will need to be a member of the Project Collection Administrator group to manage the pool's permissions.

In the following example, we will add an Administrator role to all deployment pools.

1. From within your project, select the gear icon button , and then select **Project settings**.
2. Select **Deployment Pools**, and then select **Security** to manage permissions for all deployment pools.

The screenshot shows the 'Deployment Pools' page in the Azure DevOps interface. The top navigation bar includes 'fabrikam', 'Projects', 'My favorites', '...', and a gear icon. Below the navigation is a secondary menu with links: 'Overview', 'Settings', 'Security', 'Users', 'Process', 'Build and Release', 'Agent Pools', and 'Deployment Pools\*'. The 'Deployment Pools\*' link is underlined, indicating it is the active page. A red box highlights the 'Security' tab. On the left, there's a 'Deployment Pools' section with a '+ New' button and a 'Search deployment pools' input field. The main area displays a table with columns: 'Name', 'Status', and 'Team project references'. One row is visible: 'Fabrikam Fiber-Test group' with status 'Active' and 'Fabrikam Fiber' under 'Team project references'.

3. Add the user or group and choose the role you want them to have.

The screenshot shows a modal dialog titled 'Manage default security roles for all deployment pools'. The dialog has a 'User' tab selected. Inside the dialog, there is an 'Add' button highlighted with a red box and a red arrow pointing to it from the top-left. The 'Add user' form contains fields for 'User or group' (set to 'Raisa Pokrovskaya') and 'Role' (set to 'Administrator'). A descriptive note below the form states: 'Administrator can administer, manage, view and use deployment pools.' At the bottom of the dialog are 'Add' and 'Close' buttons.

4. Select **Add** when you are done.

## Related articles

- [Grant version control permissions to the build service](#)
- [Set pipelines permissions](#)
- [Set retention policies for builds, releases, and tests](#)
- [Default permissions and access](#)
- [Permissions and groups reference](#)

# Run Git commands in a script

3/31/2021 • 5 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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 main.

Git is available on [Microsoft-hosted agents](#) and on [on-premises agents](#).

## Enable scripts to run Git commands

## NOTE

Before you begin, be sure your account's default identity is set with:

```
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
```

### Grant version control permissions to the build service

1. Go to the project settings page for your organization at [Organization Settings > General > Projects](#).



2. Select the project you want to edit.

<input type="checkbox"/>	Name ↑	Description
<input type="checkbox"/>	P	ContosoAds

3. Within **Project Settings**, select **Repositories**. Select the repository you want to run Git commands on.

4. Select **Permissions** to edit your repository permissions.

## All Repositories

Repositories Settings Policies Permissions

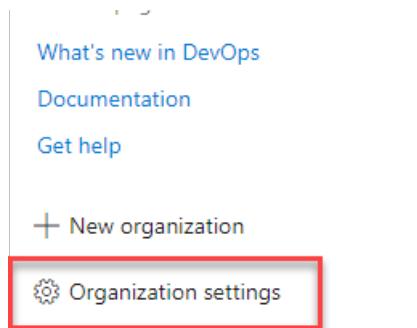
ContosoAds

5. Search for **Project Collection Build Service**. Choose the identity **Project Collection Build Service ({your organization})**. By default, this identity can read from the repo but cannot push any changes back to it. Grant permissions needed for the Git commands you want to run. Typically you'll want to grant:

- **Create branch:** Allow
- **Contribute:** Allow
- **Read:** Allow
- **Create tag:** Allow

Go to the Version Control control panel tab

- Azure Repos: [https://dev.azure.com/{your-organization}/{your-project}/\\_admin/\\_versioncontrol](https://dev.azure.com/{your-organization}/{your-project}/_admin/_versioncontrol)
- On-premises: [https://{your-server}:8080/tfs/DefaultCollection/{your-project}/\\_admin/\\_versioncontrol](https://{your-server}:8080/tfs/DefaultCollection/{your-project}/_admin/_versioncontrol)



If you see this page, select the repo, and then click the link:

The screenshot shows the 'Organization Settings' page. The left sidebar has sections for 'General' (Overview, Projects, Users, Global notifications, Usage, Extensions), 'Repos' (Repositories, Policies), and 'Active Directory'. The 'Projects' section shows two projects: 'webappcoretest' (Scrum, Online) and 'Work Item Test' (Agile SSH, Online). The 'Repositories' link under 'Repos' is highlighted with a red box.

Project name	Process	Status	Description
webappcoretest	Scrum	Online	
Work Item Test	Agile SSH	Online	

On the **Version Control** tab, select the repository in which you want to run Git commands, and then select **Project Collection Build Service**. By default, this identity can read from the repo but cannot push any changes back to it.

The screenshot shows the 'Security for all Git repositories' settings in VSTS. On the left, there's a sidebar with 'Repositories' and a 'New repository' button. Under 'Git repositories', several projects are listed: 'OurProject', 'Portal-Client', 'Portal-Server', 'Store-Client', and 'Store-Server'. The main area is titled 'Security for all Git repositories' with 'Inheritance' dropdowns. A search bar is at the top. Below it, under 'VSTS Groups', 'Project Collection Administrators' is selected. A list of users follows, with 'Project Collection Build Service (Application...)' highlighted and enclosed in a red box.

Grant permissions needed for the Git commands you want to run. Typically you'll want to grant:

- **Create branch:** Allow
- **Contribute:** Allow
- **Read:** Allow
- **Create tag:** 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](#)
- [Classic](#)

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](#)
- [Classic](#)

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
 <a href="#">Utility: Command Line</a> List the files in the Git repo.	Tool: <code>git</code> Arguments: <code>ls-files</code>

### Merge a feature branch to main

You want a CI build to merge to main 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/main (
 ECHO Building main branch so no merge is needed.
 EXIT
)
SET sourceBranch=origin/%BUILD_SOURCEBRANCH:refs/heads/=%
ECHO GIT CHECKOUT MAIN
git checkout main
ECHO GIT STATUS
git status
ECHO GIT MERGE
git merge %sourceBranch% -m "Merge to main"
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
 <a href="#">Utility: Batch Script</a> Run merge.bat.	Path: <input type="text" value="merge.bat"/>

## FAQ

### Can I run Git commands if my remote repo is in GitHub or another Git service such as Bitbucket Cloud?

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. Here are examples:

- `git commit -m "This is a commit message ***NO_CI***"`
- `git merge origin/features/hello-world -m "Merge to main ***NO_CI***"`

Add `[skip ci]` to your commit message or description. Here are examples:

- `git commit -m "This is a commit message [skip ci]"`
- `git merge origin/features/hello-world -m "Merge to main [skip ci]"`

You can also use any of the variations below. This is supported for commits to Azure Repos Git, Bitbucket Cloud, GitHub, and GitHub Enterprise Server.

- `[skip ci]` or `[ci skip]`
- `skip-checks: true` or `skip-checks:true`

- `[skip azurepipelines]` or `[azurepipelines skip]`
- `[skip azpipelines]` or `[azpipelines skip]`
- `[skip azp]` or `[azp skip]`
- `***NO_CI***`

### How does enabling scripts to run Git commands affect how the build pipeline gets build sources?

When you set `system.usegit` 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.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

### 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](#).

# Securing Azure Pipelines

5/10/2021 • 2 minutes to read • [Edit Online](#)

Azure Pipelines poses unique security challenges. You can use a pipeline to run scripts or deploy code to production environments. But you want to ensure your CI/CD pipelines don't become avenues to run malicious code. You also want to ensure only code you intend to deploy is deployed. Security must be balanced with giving teams the flexibility and power they need to run their own pipelines.

## NOTE

Azure Pipelines is one among a collection of Azure DevOps Services, all built on the same secure infrastructure in Azure. To understand the main concepts around security for all of Azure DevOps Services, see [Azure DevOps Data Protection Overview](#) and [Azure DevOps Security and Identity](#).

Traditionally, organizations implemented security through draconian lock-downs. Code, pipelines, and production environments had severe restrictions on access and use. In small organizations with a small number of users and projects, this stance was relatively easy to manage. However, that's not the case in larger organizations. Where many users have contributor access to code, one must "assume breach". Assuming breach means behaving as if an adversary has contributor access to some (if not all) of the repositories.

The goal in this case is to prevent that adversary from running malicious code in the pipeline. Malicious code may steal secrets or corrupt production environments. Another goal is to prevent lateral exposure to other projects, pipelines, and repositories from the compromised pipeline.

This series of topics outlines recommendations to help you put together a secure YAML-based CI/CD pipeline. It also covers the places where you can make trade-offs between security and flexibility. The series also assumes familiarity with [Azure Pipelines](#), the core [Azure DevOps security constructs](#), and [Git](#).

Topics covered:

- [Incremental approach to improving security](#)
- [Repository protection](#)
- [Pipeline resources](#)
- [Project structure](#)
- [Security through templates](#)
- [Variables and parameters](#)
- [Shared infrastructure](#)
- [Other security considerations](#)

# Plan how to secure your YAML pipelines

3/8/2021 • 2 minutes to read • [Edit Online](#)

We recommend that you use an incremental approach to secure your pipelines. Ideally, you would implement all of the guidance that we offer. But don't be daunted by the number of recommendations. And don't hold off making *some* improvements just because you can't make all the changes right now.

## Security recommendations depend on each other

Security recommendations have complex interdependencies. Your security posture depends heavily on which recommendations you choose to implement. The recommendations that you choose, in turn, depend on the concerns of your DevOps and security teams. They also depend on the policies and practices of your organization.

You might choose to tighten security in one critical area and accept less security but more convenience in another area. For example, if you use [extends](#) templates to require all builds to run in containers, then you might not need a [separate agent pool for each project](#).

## Begin with a nearly empty template

A good place to start is by enforcing extension from a nearly empty template. This way, as you start to apply security practices, you have a centralized place that already catches every pipeline.

For more information, see [Templates](#).

## Next steps

After you plan your security approach, consider how your [repositories](#) provide protection.

# Repository protection

5/19/2021 • 2 minutes to read • [Edit Online](#)

Source code, the pipeline's YAML file, and necessary scripts & tools are all stored in a version control repository. Permissions and branch policies must be employed to ensure changes to the code and pipeline are safe. You can also [add pipeline permissions and checks to repositories](#).

Also, you should review [default access control](#) for repositories.

Because of Git's design, protection at a branch level will only carry you so far. Users with push access to a repo can usually create new branches. If you use GitHub open-source projects, anyone with a GitHub account can fork your repository and propose contributions back. Since pipelines are associated with a repository and not with specific branches, you must assume the code and YAML files are untrusted.

## Forks

If you build public repositories from GitHub, you must consider your stance on fork builds. Forks are especially dangerous since they come from outside your organization. To protect your products from contributed code, consider the following recommendations.

### NOTE

The following recommendations apply primarily to building public repos from GitHub.

### Don't provide secrets to fork builds

By default, your pipelines do not build forks. If you decide to build forks, secrets and protected resources are not made available to the jobs in those pipelines by default. Don't turn off this latter protection.

The screenshot shows the Azure Pipelines interface. At the top, there's a navigation bar with 'buildcanary / PipelineCanary / Pipelines'. Below it is a search bar and some icons. The main area is titled 'CIX-Pipelines > xamarin-ios'. On the left, a sidebar lists various trigger types: 'Continuous integration' (microsoft/azure-pipelines-canary, Enabled), 'Pull request validation' (microsoft/azure-pipelines-canary, Enabled), 'Scheduled' (+ Add), and 'Build completion' (+ Add). The 'Scheduled' section shows 'No builds scheduled'. The 'Build completion' section shows 'Build when another build completes'. The right side of the screen displays configuration options for the selected 'microsoft/azure-pipelines-canary' trigger. It includes sections for 'Override the YAML pull request trigger from here', 'Forks' (with a note about 'Important security considerations'), 'Comments' (with options for 'On all pull requests' or 'Only on pull requests from non-team members'), and a 'Learn more' link.

#### NOTE

Even if you enable fork builds to access secrets, Azure Pipelines restricts the access token used for fork builds. It has more limited access to open resources than a normal access token. You cannot disable this protection.

### Consider manually triggering fork builds

You can turn off automatic fork builds and instead use pull request comments as a way to manually building these contributions. This setting will give you an opportunity to review the code before triggering a build.

### Use Microsoft-hosted agents for fork builds

Don't run builds from forks on self-hosted agents. By doing so, you are effectively providing a path to external organizations to run outside code on machines inside your corporate network. Use Microsoft-hosted agents or some form of network isolation for your self-hosted agents.

## User branches

Users in your organization with the right permissions can create new branches containing new or updated code. That code can run through the same pipeline as your protected branches. Further, if the YAML file in the new branch is changed, then the updated YAML will be used to run the pipeline. While this design allows for great flexibility and self-service, not all changes are safe (whether made maliciously or not).

If your pipeline consumes source code or is defined in Azure Repos, you must fully understand the [Azure Repos permissions model](#). In particular, a user with **Create Branch** permission at the repository level can introduce code to the repo even if that user lacks **Contribute** permission.

## Next steps

Next, learn about the more protection offered by checks on [protected resources](#).

# Pipeline resources

5/21/2021 • 3 minutes to read • [Edit Online](#)

Azure Pipelines offers security mechanisms beyond just protecting the YAML file and source code. When pipelines run, access to resources goes through a system called [checks](#). Checks can suspend or even fail a pipeline run in order to keep resources safe. A pipeline can access two types of resources, protected and open.

## Protected resources

Your pipelines often have access to secrets. For instance, to sign your build, you need a signing certificate. To deploy to a production environment, you need a credential to that environment. In Azure Pipelines, all of the following are considered *protected* resources:

- [agent pools](#)
- [variable groups](#)
- [secure files](#)
- [service connections](#)
- [environments](#)
- [repositories](#)

"Protected" means:

- They can be made accessible to specific users and specific pipelines within the project. They cannot be accessed by users and pipelines outside of a project.
- You can run additional manual or automated checks every time a pipeline uses one of these resources.

## Protecting repository resources

Repositories can optionally be protected. At the organization or project level, you may choose to limit the scope of the Azure Pipelines access token to mentioned repositories. When you do this, Azure Pipelines will add two additional protections:

1. The access token given to the agent for running jobs will only have access to repositories explicitly mentioned in the `resources` section of the pipeline.
2. Repositories added to the pipeline will have to be authorized by someone with read access to the repository the first time that pipeline uses the repository.

This setting is on by default for all organizations created after May 2020. Organizations created before that should enable it in [Organization settings](#).

## Open resources

All the other resources in a project are considered *open* resources. Open resources include:

- artifacts
- pipelines
- test plans
- work items

You'll learn more about which pipelines can access what resources in the section on [projects](#).

## User permissions

The first line of defense for protected resources is user permissions. In general, ensure that you only give permissions to users who require them. All protected resources have a similar security model. A member of user role for a resource can:

- Remove approvers and checks configured on that resource
- Grant access to other users or pipelines to use that resource

The screenshot shows the 'User permissions' section of the 'Security' settings for a 'dev' environment. It lists four entries:

User	Role	Access
[PipelineCanary]\Contributors	Administrator	Assigned
[PipelineCanary]\Project Administrators	Administrator	Assigned
[PipelineCanary]\Project Valid Users	Reader	Inherited
Nilkamal Adak	Administrator	Assigned

Below this, the 'Pipeline permissions' section is shown with a lock icon and the message: 'No permitted pipelines'. It states: 'This resource cannot be used until at least one pipeline has permission' and includes a 'Learn more' link.

## Pipeline permissions

When you use YAML pipelines, user permissions are not enough to secure your protected resources. You can easily copy the name of a protected resource (for example, a service connection for your production environment) and include that in a different pipeline. Pipeline permissions protect against such copying. For each of the protected resources, ensure that you have disabled the option to grant access to "all pipelines". Instead, explicitly granted access to specific pipelines that you trust.

The screenshot shows the 'Security' page for the 'dev' environment in PipelineCanary. On the left, there's a sidebar with various icons for project management. The main area has a header '← Security' and a breadcrumb trail: PipelineCanary / Pipelines / Environments / dev / Security. A search bar and several navigation icons are at the top right. The 'User permissions' section contains a table with four rows:

User	Role	Access
[PipelineCanary]\Contributors	Administrator	Assigned
[PipelineCanary]\Project Administrators	Administrator	Assigned
[PipelineCanary]\Project Valid Users	Reader	Inherited

The 'Pipeline permissions' section shows a lock icon and a message: 'No permitted pipelines'. It says 'This resource cannot be used until at least one pipeline has pe' and has a 'Learn more' link. To the right, a search bar and a list of pipelines are shown:

- Run all pools builds
- node8-typescript-container
- service-containers
- deployment-tasks-test
- ant
- maven

## Checks

In YAML, a combination of user and pipeline permissions is not enough to fully secure your protected resources. Pipeline permissions to resources are granted to the whole pipeline. Nothing prevents an adversary from creating another branch in your repository, injecting malicious code, and using the same pipeline to access that resource. Even without malicious intent, most pipelines need a second set of eyes look over changes (especially to the pipeline itself) before deploying to production. **Checks** allow you to pause the pipeline run until certain conditions are met:

- **Manual approval check.** Every run that uses a project protected resource is blocked for your manual approval before proceeding. This gives you the opportunity to review the code and ensure that it is coming from the right branch.
- **Protected branch check.** If you have manual code review processes in place for some of your branches, you can extend this protection to pipelines. Configure a protected branch check on each of your resources. This will automatically stop your pipeline from running on top of any user branches.



← **dev**  
Automatically created environment

**Deployments**

Run	Jobs	
Merge 5471531a2be613e5495f2ea0bb0ec7... #20191218.4 on deploymentJob-output-variables	✓ DeploymentJobSettingVariable and 1 more	Delete 🕒 9m 30s
Adding test for deployment job output varia... #20191218.1 on deploymentJob-output-variables	✓ DeploymentJobSettingVariable and 1 more	Edit 🕒 Yesterday 🕒 38s
Adding test for deployment job output varia... #20191217.1 on deploymentJob-output-variables	✓ DeploymentJobSettingVariable and 1 more	Edit 🕒 Yesterday 🕒 41s

  
**You're almost done!**

To get full end to end traceability per resource

[Add resource](#)

## Next steps

Next, consider how you group resources into a [project structure](#).

# Recommendations to securely structure projects in your pipeline

3/8/2021 • 2 minutes to read • [Edit Online](#)

Beyond the scale of individual resources, you should also consider groups of resources. In Azure DevOps, resources are grouped by team projects. It's important to understand what resources your pipeline can access based on project settings and containment.

Every job in your pipeline receives an access token. This token has permissions to read open resources. In some cases, pipelines might also update those resources. In other words, your user account might not have access to a certain resource, but scripts and tasks that run in your pipeline might have access to that resource. The security model in Azure DevOps also allows access to these resources from other projects in the organization. If you choose to shut off pipeline access to some of these resources, then your decision applies to all pipelines in a project. A specific pipeline can't be granted access to an open resource.

## Separate projects

Given the nature of open resources, you should consider managing each product and team in a separate project. This practice ensures that a pipeline from one product can't access open resources from another product. In this way, you prevent lateral exposure. When multiple teams or products share a project, you can't granularly isolate their resources from one another.

If your Azure DevOps organization was created before August 2019, then runs might be able to access open resources in all of your organization's projects. Your organization administrator must review a key security setting in Azure Pipelines that enables project isolation for pipelines. You can find this setting at **Azure DevOps > Organization settings > Pipelines > Settings**. Or go directly to this Azure DevOps location: [https://dev.azure.com/ORG-NAME/\\_settings/pipelinessettings](https://dev.azure.com/ORG-NAME/_settings/pipelinessettings).

## Organization Settings

buildcanary

Search Settings

### General

- Disable anonymous access to badges ⓘ
- Limit variables that can be set at queue time ⓘ
- Limit job authorization scope to current project ⓘ

## Next steps

After you've set up the right project structure, enhance runtime security by using [templates](#).

# Security through templates

3/8/2021 • 6 minutes to read • [Edit Online](#)

[Checks on protected resources](#) are the basic building block of security for Azure Pipelines. Checks work no matter the structure - the stages and jobs - of your pipeline. If several pipelines in your team or organization have the same structure, you can further simplify security using [templates](#).

Azure Pipelines offers two kinds of templates: [includes](#) and [extends](#). Included templates behave like `#include` in C++: it's as if you paste the template's code right into the outer file, which references it. To continue the C++ metaphor, [extends](#) templates are more like inheritance: the template provides the outer structure of the pipeline and a set of places where the template consumer can make targeted alterations.

## Use `extends` templates

For the most secure pipelines, we recommend starting with [extends](#) templates. By providing the outer structure, a template can prevent malicious code from getting into your pipeline. You can still use [includes](#), both in the template and in the final pipeline, to factor out common pieces of configuration. To use an `extends` template, your pipeline might look like the below example.

```
template.yml
parameters:
- name: usersteps
 type: stepList
 default: []
steps:
- ${{ each step in parameters.usersteps }}:
 - ${{ step }}
```

```
azure-pipelines.yml
resources:
 repositories:
 - repository: templates
 type: git
 name: MyProject/MyTemplates
 ref: refs/tags/v1

extends:
 template: template.yml@templates
 parameters:
 usersteps:
 - script: echo This is my first step
 - script: echo This is my second step
```

When you set up [extends](#) templates, consider anchoring them to a particular Git branch or tag. That way, if breaking changes need to be made, existing pipelines won't be affected. The examples above use this feature.

## Security features enforced through YAML

There are several protections built into the YAML syntax, and an `extends` template can enforce the usage of any or all of them.

### Step targets

Restrict some steps to run in a container instead of the host. Without access to the agent's host, user steps can't

modify agent configuration or leave malicious code for later execution. Run code on the host first to make the container more secure. For instance, we recommend limiting access to network. Without open access to the network, user steps will be unable to access packages from unauthorized sources, or upload code and secrets to a network location.

```
resources:
 containers:
 - container: builder
 image: mysecurebuildcontainer:latest
 steps:
 - script: echo This step runs on the agent host, and it could use docker commands to tear down or limit the container's network
 - script: echo This step runs inside the builder container
 target: builder
```

## Agent logging command restrictions

Restrict what services the Azure Pipelines agent will provide to user steps. Steps request services using "logging commands" (specially formatted strings printed to stdout). In restricted mode, most of the agent's services such as uploading artifacts and attaching test results are unavailable.

```
this task will fail because its `target` property instructs the agent not to allow publishing artifacts
- task: PublishBuildArtifacts@1
 inputs:
 artifactName: myartifacts
 target:
 commands: restricted
```

One of the commands still allowed in restricted mode is the `setvariable` command. Because pipeline variables are exported as environment variables to subsequent tasks, tasks that output user-provided data (for example, the contents of open issues retrieved from a REST API) can be vulnerable to injection attacks. Such user content can set environment variables that can in turn be used to exploit the agent host. To disallow this, pipeline authors can explicitly declare which variables are settable via the `setvariable` logging command. Specifying an empty list disallows setting all variables.

```
this task will fail because the task is only allowed to set the 'expectedVar' variable, or a variable
prefixed with "ok"
- task: PowerShell@2
 target:
 commands: restricted
 settableVariables:
 - expectedVar
 - ok*
 inputs:
 targetType: 'inline'
 script: |
 Write-Host "##vso[task.setvariable variable=BadVar]myValue"
```

## Conditional insertion of stages or jobs

Restrict stages and jobs to run under specific conditions. Conditions can help, for example, to ensure that you are only building certain branches.

```

jobs:
- job: buildNormal
 steps:
 - script: echo Building the normal, unsensitive part
- ${{ if eq(variables['Build.SourceBranchName'], 'refs/heads/main') }}:
 - job: buildMainOnly
 steps:
 - script: echo Building the restricted part that only builds for main branch

```

## Require certain syntax with extends templates

Templates can iterate over and alter/disallow any YAML syntax. Iteration can force the use of particular YAML syntax including the above features.

A template can rewrite user steps and only allow certain approved tasks to run. You can, for example, prevent inline script execution.

### WARNING

In the example below, only the literal step type "script" is prevented. For full lockdown of ad-hoc scripts, you would also need to block "bash", "pwsh", "powershell", and the tasks which back these steps.

```

template.yml
parameters:
- name: usersteps
 type: stepList
 default: []
steps:
- ${{ each step in parameters.usersteps }}:
 - ${{ each pair in step }}:
 ${{{ if ne(pair.key, 'script') }}}:
 ${{{ pair.key }}}: ${{{ pair.value }}}

```

```

azure-pipelines.yml
extends:
 template: template.yml
parameters:
 usersteps:
 - task: MyTask@1
 - script: echo This step will be stripped out and not run!
 - task: MyOtherTask@2

```

## Type-safe parameters

Templates and their parameters are turned into constants before the pipeline runs. Template parameters provide type safety to input parameters. For instance, it can restrict which pools can be used in a pipeline by offering an enumeration of possible options rather than a freeform string.

```
template.yml
parameters:
- name: userpool
 type: string
 default: Azure Pipelines
 values:
 - Azure Pipelines
 - private-pool-1
 - private-pool-2

pool: ${{ parameters.userpool }}
steps:
- script: # ... removed for clarity
```

```
azure-pipelines.yml
extends:
 template: template.yml
parameters:
 userpool: private-pool-1
```

## Set required templates

To require that a specific template gets used, you can set the [required template check](#) for a resource or environment. The required template check can be used when extending from a template.

You can check on the status of a check when viewing a pipeline job. When a pipeline doesn't extend from the require template, the check will fail and the run will stop. You will see that your check failed.

### Jobs

 0/1 checks passed

Name

 Job

When the required template is used, you'll see that your check passed.

### Jobs

 1 check passed

Name

 Job

Here the template `params.yml` is required with an approval on the resource. To trigger the pipeline to fail, comment out the reference to `params.yml`.

```

params.yml
parameters:
- name: yesNo
 type: boolean
 default: false
- name: image
 displayName: Pool Image
 type: string
 default: ubuntu-latest
 values:
 - windows-latest
 - vs2017-win2016
 - ubuntu-latest
 - ubuntu-16.04
 - macOS-latest
 - macOS-10.14

steps:
- script: echo ${{ parameters.yesNo }}
- script: echo ${{ parameters.image }}

```

```

azure-pipeline.yml

resources:
containers:
- container: my-container
 endpoint: my-service-connection
 image: mycontainerimages

extends:
 template: params.yml
 parameters:
 yesNo: true
 image: 'windows-latest'

```

## Additional steps

A template can add steps without the pipeline author having to include them. These steps can be used to run credential scanning or static code checks.

```

template to insert a step before and after user steps in every 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: CredScan@1 # Pre steps
 - ${{ job.steps }} # Users steps
 - task: PublishMyTelemetry@1 # Post steps
 condition: always()

```

## Next steps

Next, learn about taking inputs safely through [variables and parameters](#).

# How to securely use variables and parameters in your pipeline

3/8/2021 • 2 minutes to read • [Edit Online](#)

This article discusses how to securely use variables and parameters to gather input from pipeline users.

## Variables

Variables can be a convenient way to collect information from the user up front. You can also use variables to pass data from step to step within a pipeline.

But use variables with caution. Newly created variables, whether they're defined in YAML or written by a script, are read-write by default. A downstream step can change the value of a variable in a way that you don't expect.

For instance, imagine your script reads:

```
msbuild.exe myproj.proj -property:Configuration=$(MyConfig)
```

A preceding step could set `MyConfig` to `Debug & deltree /y c:`. Although this example would only delete the contents of your build agent, you can imagine how this setting could easily become far more dangerous.

You can make variables read-only. System variables like `Build.SourcesDirectory`, task output variables, and queue-time variables are always read-only. Variables that are created in YAML or created at run time by a script can be designated as read-only. When a script or task creates a new variable, it can pass the `isReadOnly=true` flag in its logging command to make the variable read-only.

In YAML, you can specify read-only variables by using a specific key:

```
variables:
- name: myReadOnlyVar
 value: myValue
 readonly: true
```

Queue-time variables are exposed to the end user who manually runs a pipeline. As originally designed, this concept was only for the UI. The underlying API would accept user overrides of any variable, even variables that weren't designated as queue-time variables. This arrangement was confusing and insecure. So we've added a setting that makes the API accept only variables that can be set at queue time. We recommend that you turn on this setting.

## Parameters

Unlike variables, pipeline parameters can't be changed by a pipeline while it's running. Parameters have data types such as `number` and `string`, and they can be restricted to a subset of values. Restricting the parameters is useful when a user-configurable part of the pipeline should take a value only from a constrained list. The setup ensures that the pipeline won't take arbitrary data.

## Next steps

After you secure your inputs, you also need to secure your [shared infrastructure](#).

# Recommendations to secure shared infrastructure in Azure Pipelines

3/8/2021 • 2 minutes to read • [Edit Online](#)

Protected resources in Azure Pipelines are an abstraction of real infrastructure. Follow these recommendations to protect the underlying infrastructure.

## Use Microsoft-hosted pools

Microsoft-hosted pools offer isolation and a clean VM for each run of a pipeline. If possible, use Microsoft-hosted pools rather than self-hosted pools.

## Separate agents for each project

An agent can be bound to only one pool. You might want to share agents across projects by sharing the pool with multiple projects. In other words, multiple projects might run jobs on the same agent, one after another. Although this practice saves infrastructure costs, it can allow lateral movement.

To eliminate that form of lateral movement and to prevent one project from "poisoning" an agent for another project, keep separate agent pools with separate agents for each project.

## Use low-privileged accounts to run agents

It's tempting but dangerous to run the agent under an identity that can directly access Azure DevOps resources. This problematic setup is common in organizations that use Azure Active Directory (Azure AD). If you run the agent under an identity that's backed by Azure AD, then it can directly access Azure DevOps APIs without using the job's access token. You should instead run the agent as a nonprivileged local account such as *Network Service*.

Azure DevOps has a group that's misleadingly named *Project Collection Service Accounts*. By inheritance, members of Project Collection Service Accounts are also members of Project Collection Administrators. Customers sometimes run their build agents by using an identity that's backed by Azure AD and that's a member of Project Collection Service Accounts. If adversaries run a pipeline on one of these build agents, then they can take over the entire Azure DevOps organization.

We've also seen self-hosted agents run under highly privileged accounts. Often, these agents use privileged accounts to access secrets or production environments. But if adversaries run a compromised pipeline on one of these build agents, then they can access those secrets. Then the adversaries can move laterally through other systems that are accessible through those accounts.

To keep your systems secure, use the lowest-privilege account to run self-hosted agents. For example, use your machine account or a managed service identity. Let Azure Pipelines manage access to secrets and environments.

## Minimize the scope of service connections

Service connections must be able to access only the resources that they require. For instance, an Azure service connection should use Azure Resource Manager and service principals that are scoped to the resources that they need to access. They shouldn't have broad contributor rights for the entire Azure subscription.

When you create a new Azure Resource Manager Service Connection, always select a resource group. Ensure that your resource group contains only the VMs or resources that the build requires. Similarly, when you

configure the GitHub app, grant access only to the repositories that you want to build by using Azure Pipelines.

## Next steps

Consider a few [general recommendations](#) for security.

# Other security considerations

3/8/2021 • 2 minutes to read • [Edit Online](#)

There are a handful of other things you should consider when securing pipelines.

## Relying on PATH

Relying on the agent's `PATH` setting is dangerous. It may not point where you think it does, since a previous script or tool could have altered it. For security-critical scripts and binaries, always use a fully qualified path to the program.

## Logging of secrets

Azure Pipelines attempts to scrub secrets from logs wherever possible. This filtering is on a best-effort basis and cannot catch every way that secrets can be leaked. Avoid echoing secrets to the console, using them in command line parameters, or logging them to files.

## Lock down containers

Containers have a few system-provided volume mounts mapping in the tasks, the workspace, and external components required to communicate with the host agent. You can mark any or all of these volumes read-only.

```
resources:
 containers:
 - container: example
 image: ubuntu:18.04
 mountReadOnly:
 externals: true
 tasks: true
 tools: true
 work: false # the default; shown here for completeness
```

Most people should mark the first three read-only and leave `work` as read-write. If you know you won't write to the work directory in a given job or step, go ahead and make `work` read-only as well. If you have tasks in your pipeline which self-modify, you may need to leave `tasks` read-write.

## Control available tasks

You can disable the ability to install and run tasks from the Marketplace. This will allow you greater control over the code which executes in a pipeline. You may also disable all the in-the-box tasks (except Checkout, which is a special action on the agent). We recommend that you don't disable in-the-box tasks under most circumstances.

Tasks directly installed with `tfx` are always available. With both of these features enabled, only those tasks are available.

## Use the Auditing service

A number of pipeline events are recorded in the Auditing service. Review the audit log periodically to ensure no malicious changes have slipped past. Visit [https://dev.azure.com/ORG-NAME/\\_settings/audit](https://dev.azure.com/ORG-NAME/_settings/audit) to get started.

## Next steps

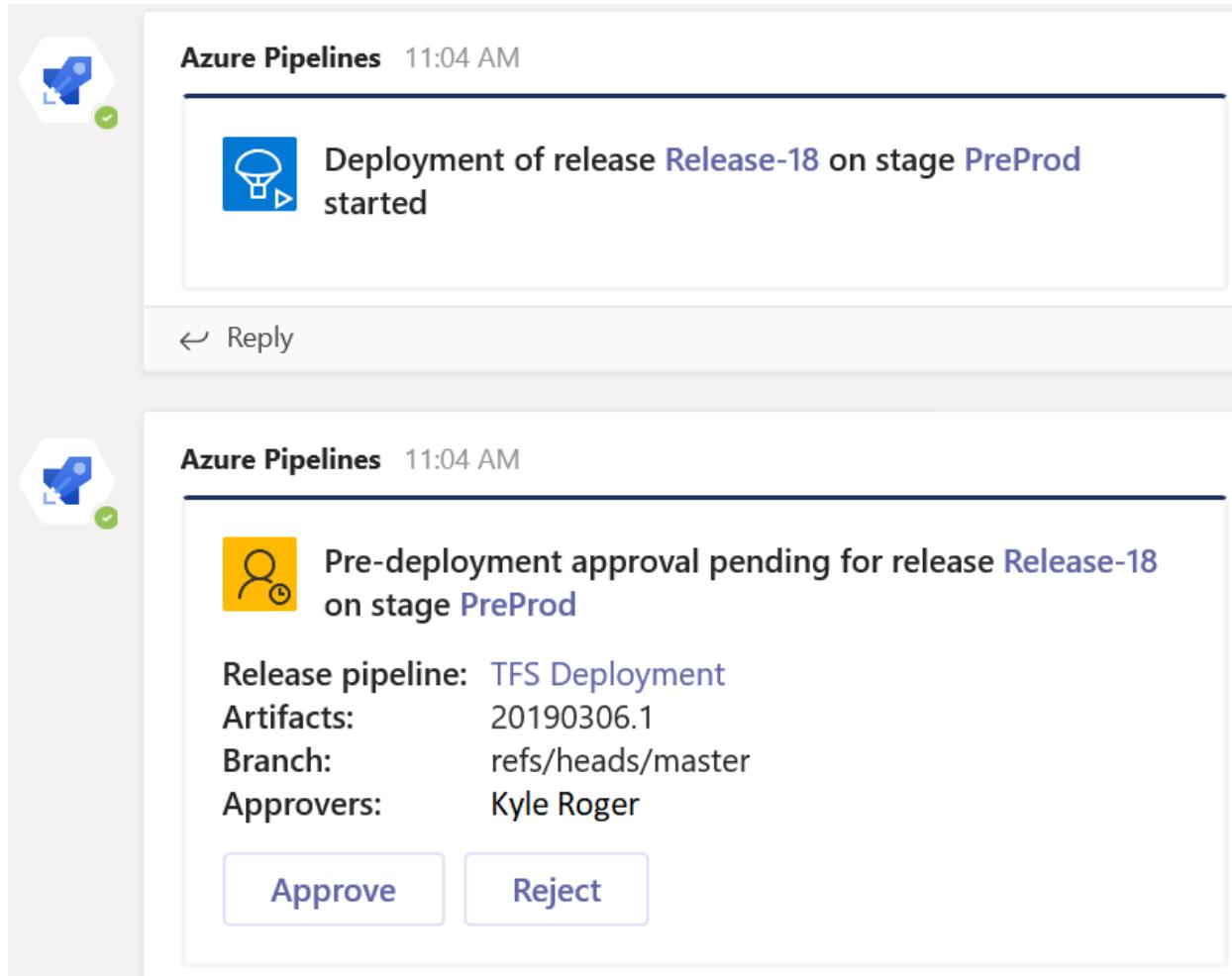
Return to the [overview](#) and make sure you've covered every topic.

# Azure Pipelines with Microsoft Teams

11/2/2020 • 7 minutes to read • [Edit Online](#)

## Azure Pipelines

If [Microsoft Teams](#) is your choice for collaboration, you can use the Azure Pipelines app built for Microsoft Teams to easily monitor the events for your pipelines. Set up and manage subscriptions for builds, releases, YAML pipelines, pending approvals and more from the app and get notifications for these events in your Teams channels.



The screenshot shows two messages from the "Azure Pipelines" channel in Microsoft Teams. The first message, at 11:04 AM, displays a blue icon of a basketball hoop and a green checkmark, followed by the text: "Deployment of release [Release-18](#) on stage [PreProd](#) started". Below this message is a "Reply" button. The second message, also at 11:04 AM, displays a yellow icon of a person with a gear, followed by the text: "Pre-deployment approval pending for release [Release-18](#) on stage [PreProd](#)". Below this text, it lists pipeline details: "Release pipeline: [TFS Deployment](#)", "Artifacts: 20190306.1", "Branch: refs/heads/master", and "Approvers: Kyle Roger". At the bottom of this message are two buttons: "Approve" and "Reject".

### NOTE

This feature is only available on Azure DevOps Services. Typically, new features are introduced in the cloud service first, and then made available on-premises in the next major version or update of Azure DevOps Server. To learn more, see [Azure DevOps Feature Timeline](#).

## Add Azure Pipelines app to your team

Visit the App store in Microsoft Teams and search for the Azure Pipelines app. Upon installing, a welcome message from the app displays as shown in the following example. Use the `@azure pipelines` handle to start interacting with the app.



Azure Pipelines 10:53 AM

👋 Hi Adam Chu! Please sign in before making any request.

← Reply

## Connect the Azure Pipelines app to your pipelines

Once the app is installed in your team, you can connect the app to the pipelines you want to monitor. The app asks you to sign in & authenticate to Azure Pipelines before running any commands. Use [Sign in with different email](#) if your Microsoft Teams and Azure Boards are in different tenants.



Azure Pipelines

Sign in



Connect your Azure Pipelines account to use the service in Microsoft Teams.



Sign in

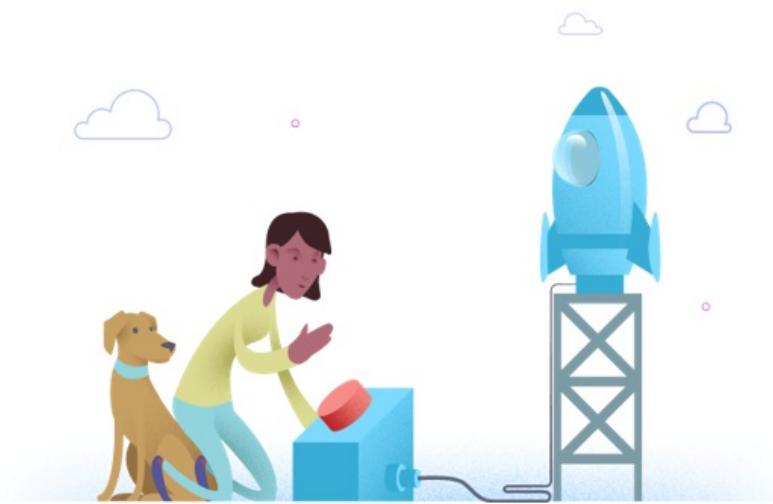
[Sign in with different email](#)

If you facing issue, please refer [troubleshooting](#) guide.



Azure Pipelines

Sign in



You've signed in as **adam.c@wowmail.com** and you can now run any command.  
Simply start a message with '**@Azure Pipelines**' to see what you can do.

**Close**

To start monitoring all pipelines in a project, use the following command inside a channel:

```
@azure pipelines subscribe [project url]
```

The project URL can be to any page within your project (except URLs to pipelines).

For example:

```
@azure pipelines subscribe https://dev.azure.com/myorg/myproject/
```

You can also monitor a specific pipeline using the following command:

```
@azure pipelines subscribe [pipeline url]
```

The pipeline URL can be to any page within your pipeline that has a `definitionId` or `buildId/releaseId` present in the URL.

For example:

```
@azure pipelines subscribe https://dev.azure.com/myorg/myproject/_build?definitionId=123
```

or:

```
@azure pipelines subscribe https://dev.azure.com/myorg/myproject/_release?
definitionId=123&view=mine&_a=releases
```

For Build pipelines, the channel is subscribed to the *Build completed* notification. For Release pipelines, the channel is subscribed to the *Release deployment started*, *Release deployment completed*, and *Release deployment approval pending* notifications. For YAML pipelines, subscriptions are created for the *Run stage state changed* and *Run stage waiting for approval* notifications.

Azure Pipelines Yesterday 8:58 PM  
Adam Chu has added subscriptions to **Deployment started**, **Deployment completed** and **Deployment approval pending** notifications for [Cortana\\_LearningModel](#). Add or Remove subscriptions for this channel with : [@Azure Pipelines subscriptions](#)

Reply

## Add or remove subscriptions

To manage the subscriptions for a channel, use the following command:

```
@azure pipelines subscriptions
```

This command lists all of the current subscriptions for the channel and allows you to add/remove subscriptions.

[!NOTE] Team administrators aren't able to remove or modify subscriptions created by Project administrators.

Kyle roger Yesterday 9:00 PM  
Azure Pipelines subscriptions

Azure Pipelines Yesterday 9:00 PM

This channel has **6** subscriptions from **2** pipelines:

**Cortana\_LearningModel**  
Release deployment approval pending  
For *any* stage with *any* approval type

**Cortana\_LearningModel**  
Release deployment completed  
For *any* stage with *any* environment status

**Cortana\_LearningModel**  
Release deployment started  
For *any* stage

and **3** more subscriptions

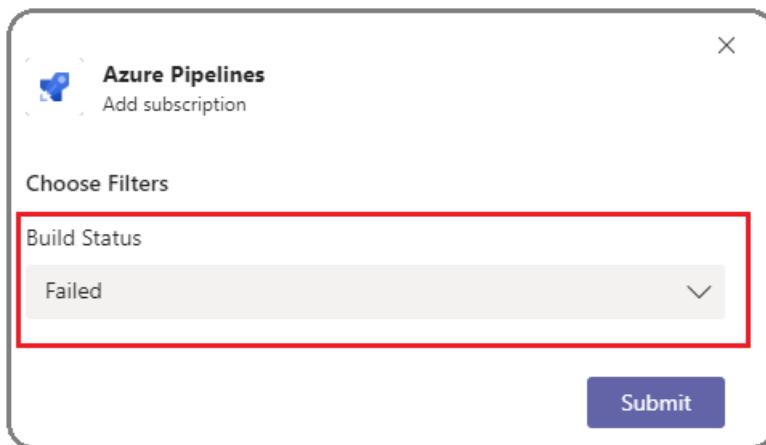
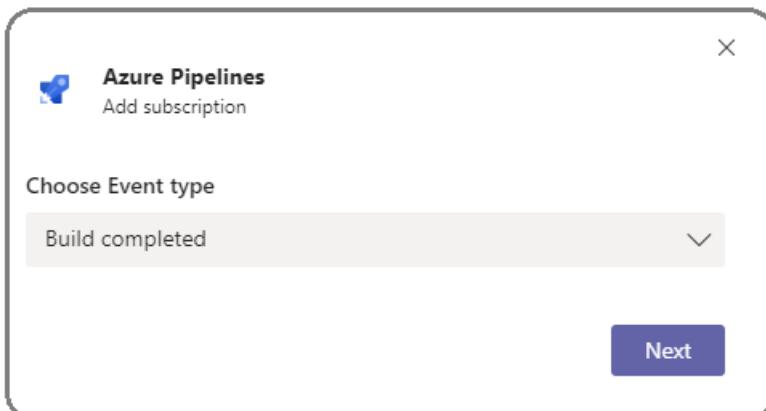
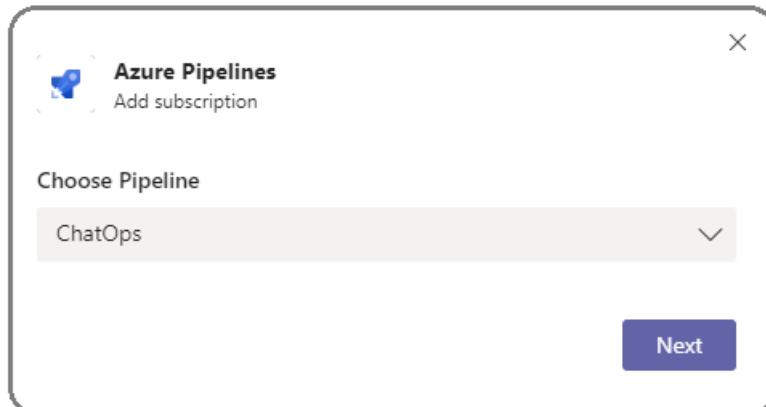
[View all subscriptions](#) [Add subscription](#)

## Using filters effectively to customize subscriptions

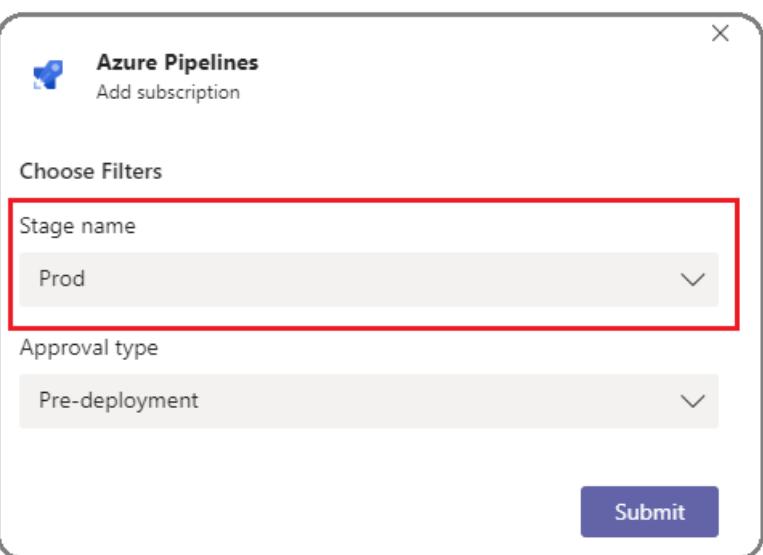
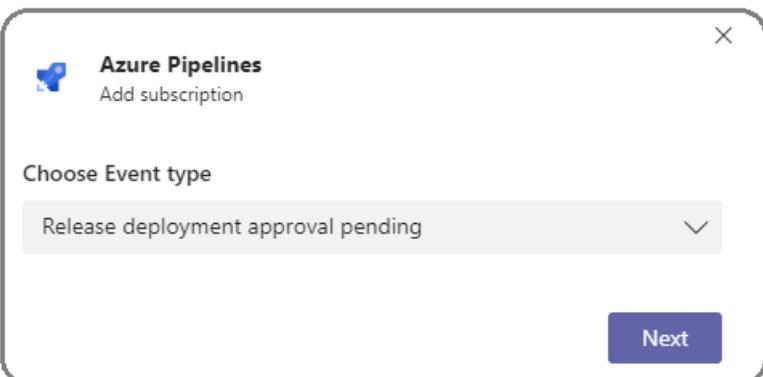
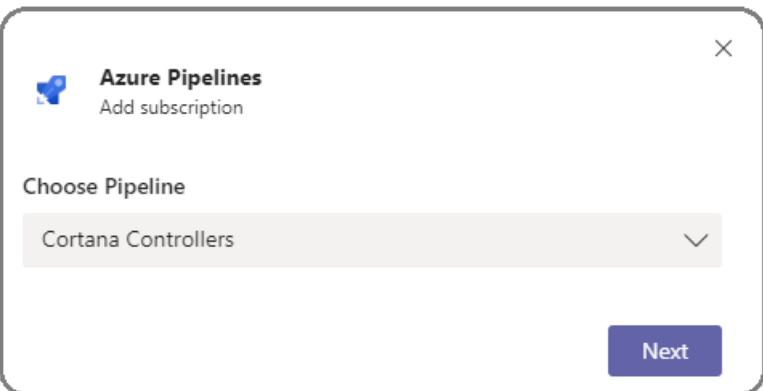
When a user subscribes to any pipeline, a few subscriptions are created by default without any filters being applied. Often, users have the need to customize these subscriptions. For example, users may want to get notified only when builds fail or when deployments are pushed to a production environment. The Azure Pipelines app supports filters to customize what you see in your channel.

1. Run the `@Azure Pipelines subscriptions` command
2. Select **View all subscriptions**. In the list of subscriptions, if there is a subscription that is unwanted or should be modified (Example: creating noise in the channel), select **Remove**
3. Scroll down and select the **Add subscription** button
4. Select the required pipeline and the event
5. Select the appropriate filters and save

**Example: Get notifications only for failed builds**

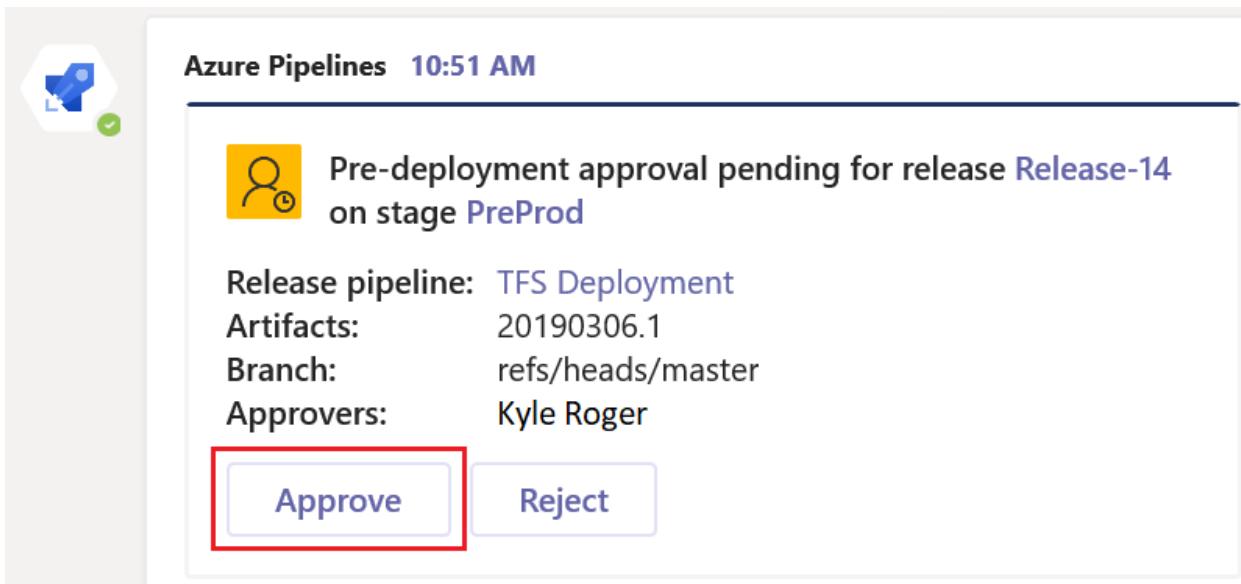


**Example: Get notifications only if the deployments are pushed to prod environment**

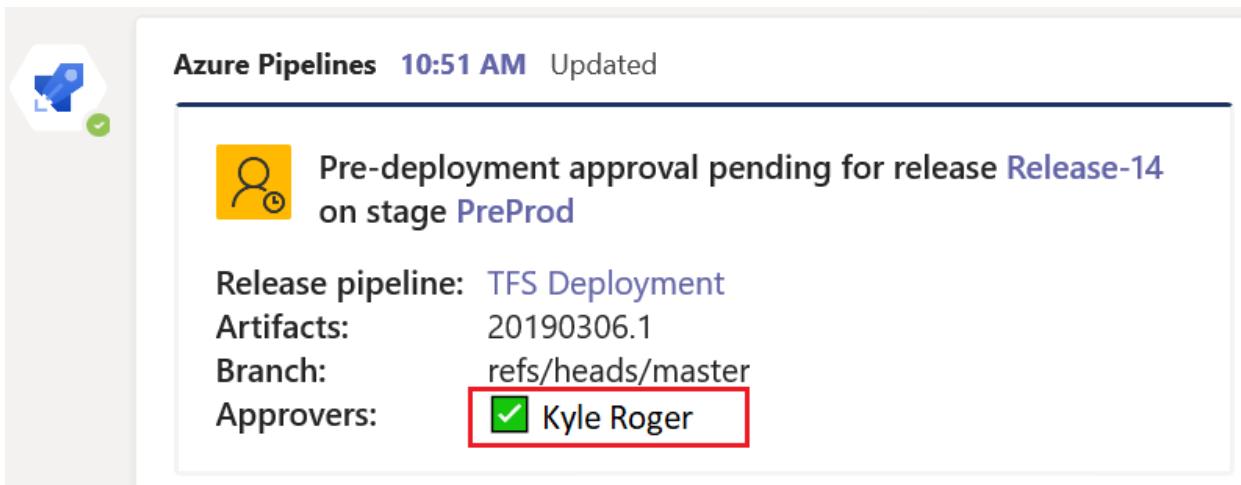


## Approve deployments from your channel

You can approve deployments from within your channel without navigating to the Azure Pipelines portal by subscribing to the *Release deployment approval pending* notification for classic Releases or the *Run stage waiting for approval* notification for YAML pipelines. Both of these subscriptions are created by default when you subscribe to the pipeline.



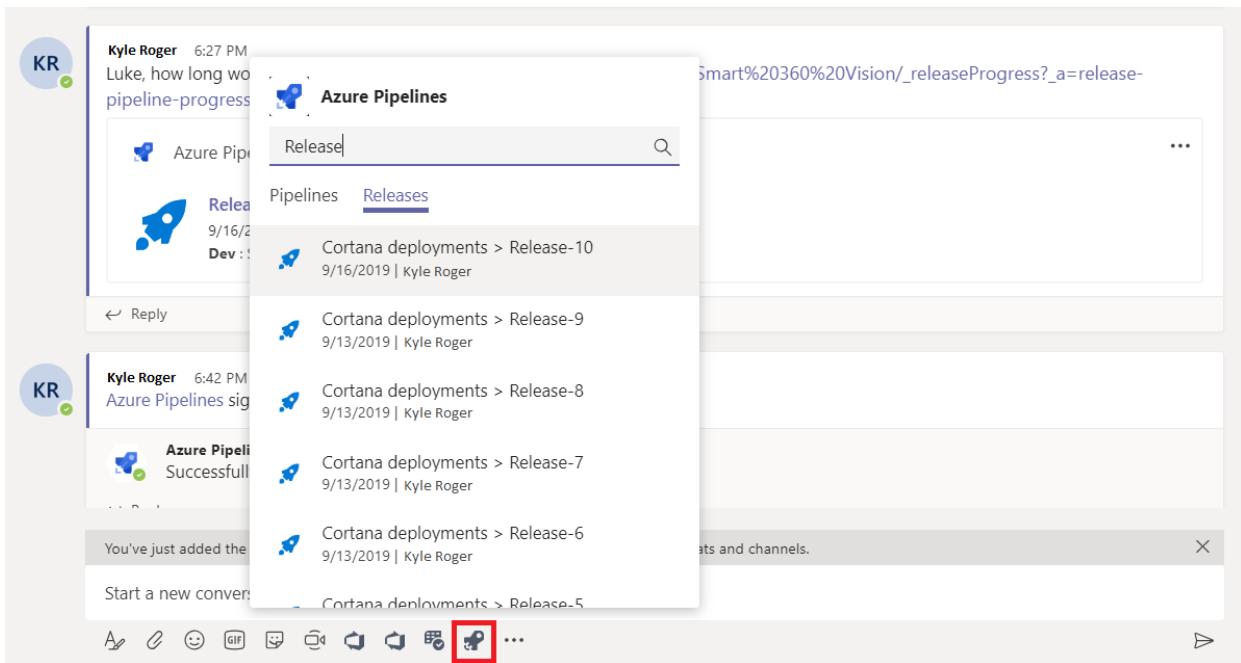
Whenever the running of a stage is pending for approval, a notification card with options to approve or reject the request is posted in the channel. Approvers can review the details of the request in the notification and take appropriate action. In the following example, the deployment was approved and the approval status is displayed on the card.



The app supports all of the checks and approval scenarios present in the Azure Pipelines portal, like single approver, multiple approvers (any one user, any order, in sequence), and teams as approvers. You can approve requests as an individual or on behalf of a team.

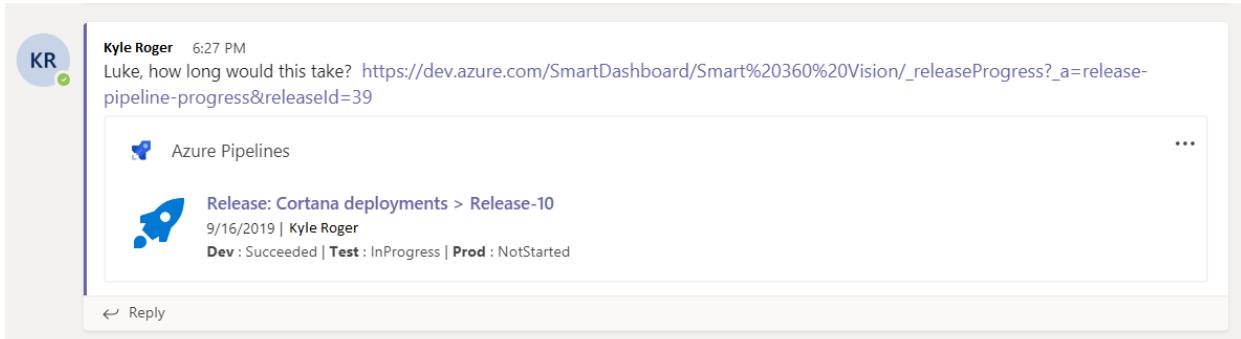
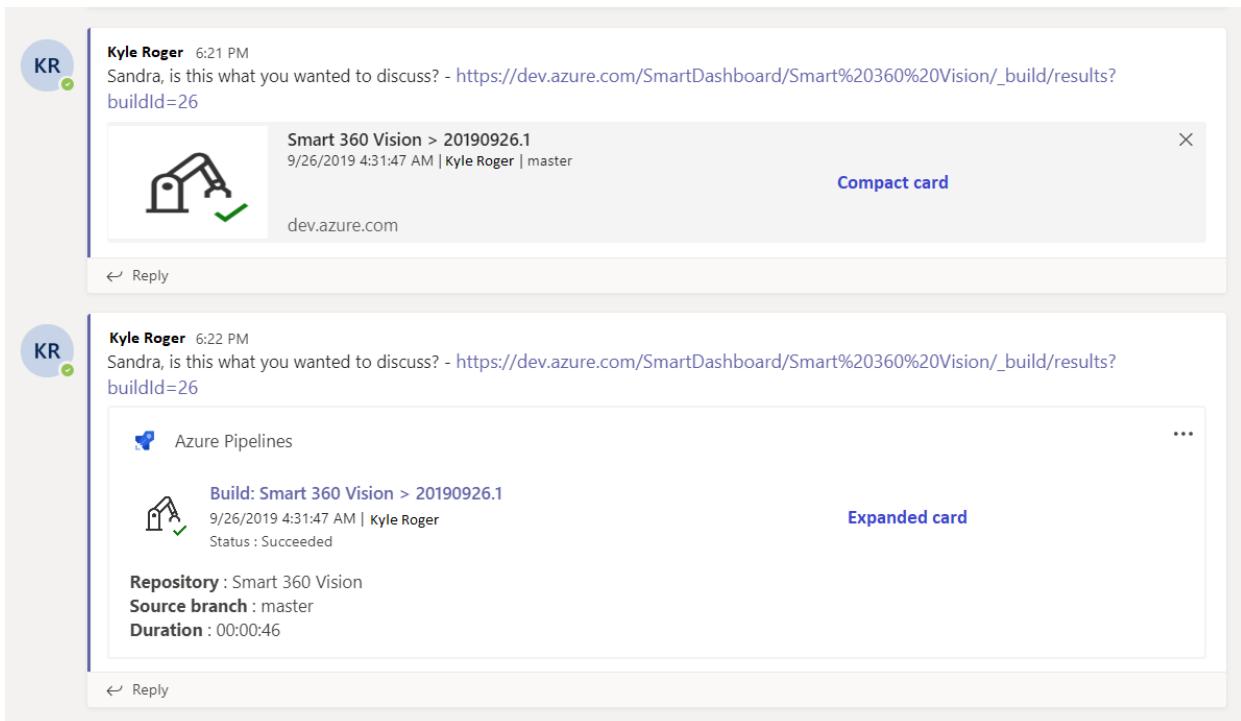
## Search and share pipeline information using compose extension

To help users search and share information about pipelines, Azure Pipelines app for Microsoft Teams supports compose extension. You can now search for pipelines by pipeline ID or by pipeline name. For compose extension to work, users will have to sign into Azure Pipelines project that they are interested in either by running `@azure pipelines signin` command or by signing into the compose extension directly.



## Previews of pipeline URLs

When a user pastes a pipeline URL, a preview is shown similar to that in the following image. This helps to keep pipeline related conversations relevant and accurate. Users can choose between compact and expanded cards.



For this feature to work, users have to be signed-in. Once they are signed in, this feature will work for all channels in a team in Microsoft Teams.

# Remove subscriptions and pipelines from a channel

If you want to clean up your channel, use the following commands to unsubscribe from all pipelines within a project.

```
@azure pipelines unsubscribe all [project url]
```

For example:

```
@azure pipelines unsubscribe all https://dev.azure.com/myorg/myproject
```

This command deletes all the subscriptions related to any pipeline in the project and removes the pipelines from the channel.

## IMPORTANT

Only project administrators can run this command.

## Threaded notifications

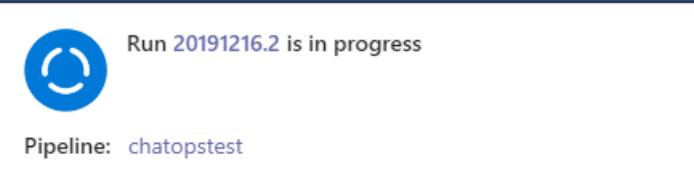
To logically link a set of related notifications and also to reduce the space occupied by notifications in a channel, notifications are threaded. All notifications linked to a particular run of a pipeline will be linked together.

The following example shows the compact view of linked notifications.

The screenshot shows a threaded notification view for a pipeline run. The top message is a status update: "Run 20191216.2 is in progress" with a blue circular icon. Below it is a reply from "Azure Pipelines Yesterday 2:57 PM" about an approval pending for the run on Stage PPE, with a yellow user icon. At the bottom of the thread is a "Reply" button.

When expanded, you can see all the of the linked notifications, as shown in the following example.

Azure Pipelines Yesterday 2:57 PM

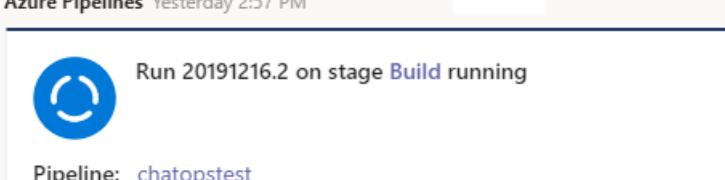


Run 20191216.2 is in progress

Pipeline: chatopstest

▼ Collapse all

Azure Pipelines Yesterday 2:57 PM



Run 20191216.2 on stage Build running

Pipeline: chatopstest

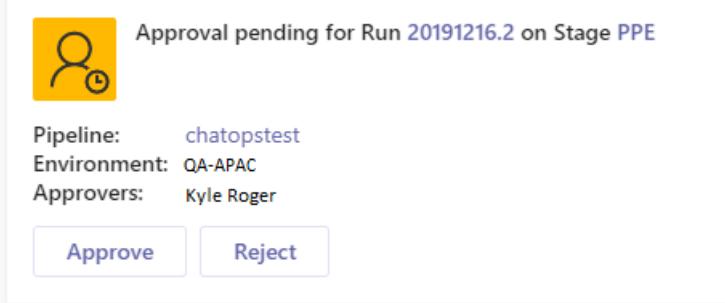
---



Run 20191216.2 on stage Build succeeded

Pipeline: chatopstest

---

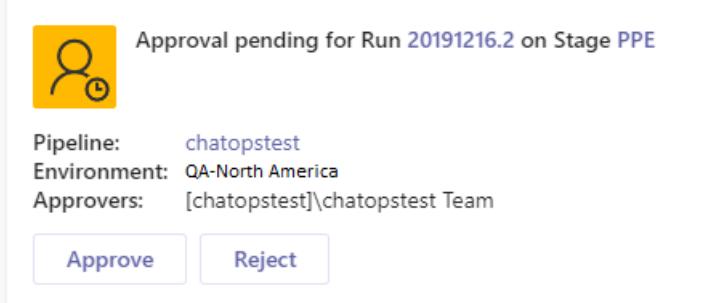


Approval pending for Run 20191216.2 on Stage PPE

Pipeline: chatopstest  
Environment: QA-APAC  
Approvers: Kyle Roger

[Approve](#) [Reject](#)

---



Approval pending for Run 20191216.2 on Stage PPE

Pipeline: chatopstest  
Environment: QA-North America  
Approvers: [chatopstest]\chatopstest Team

[Approve](#) [Reject](#)

## Commands reference

Here are all the commands supported by the Azure Pipelines app:

SLASH COMMAND	FUNCTIONALITY
@azure pipelines subscribe [pipeline url/ project url]	Subscribe to a pipeline or all pipelines in a project to receive notifications
@azure pipelines subscriptions	Add or remove subscriptions for this channel
@azure pipelines feedback	Report a problem or suggest a feature

SLASH COMMAND	FUNCTIONALITY
@azure pipelines help	Get help on the slash commands
@azure pipelines signin	Sign in to your Azure Pipelines account
@azure pipelines signout	Sign out from your Azure Pipelines account
@azure pipelines unsubscribe all [project url]	Remove all pipelines (belonging to a project) and their associated subscriptions from a channel

#### NOTE

- You can use the Azure Pipelines app for Microsoft Teams only with a project hosted on Azure DevOps Services at this time.
- The user must be an admin of the project containing the pipeline to set up the subscriptions
- Notifications are currently not supported inside chat/direct messages
- Deployment approvals which have applied the **Revalidate identity of approver before completing the approval** policy are not supported
- 'Third party application access via OAuth' must be enabled to receive notifications for the organization in Azure DevOps (Organization Settings -> Security -> Policies)

## Multi-tenant support

In your organization if you are using a different email or tenant for Microsoft Teams and Azure DevOps, perform the following steps to sign in and connect based on your use case.

Case	Email ID and tenant in Microsoft Teams	Email ID and tenant in Azure DevOps	Steps to take
1	<a href="mailto:email1@abc.com">email1@abc.com</a> (tenant 1)	<a href="mailto:email1@abc.com">email1@abc.com</a> (tenant 1)	Sign in using <b>Sign in</b> button.
2	<a href="mailto:email1@abc.com">email1@abc.com</a> (tenant 1)	<a href="mailto:email1@abc.com">email1@abc.com</a> (tenant 2)	<ul style="list-style-type: none"> <li>• Sign in the Azure DevOps account</li> <li>• In the same browser, start a new tab, navigate to <a href="https://teams.microsoft.com/">https://teams.microsoft.com/</a></li> <li>• Run the <code>signin</code> command and choose the <b>Sign in</b> button.</li> </ul>
3	<a href="mailto:email1@abc.com">email1@abc.com</a> (tenant 1)	<a href="mailto:email2@pqr.com">email2@pqr.com</a> (tenant 2)	Sign in using <b>Sign in with different email address</b> , in the email id picker use the email2 to sign in to Azure DevOps.
4	<a href="mailto:email1@abc.com">email1@abc.com</a> (tenant 1)	<a href="mailto:email2@pqr.com">email2@pqr.com</a> (non default tenant 3)	This scenario is not supported today

# Troubleshooting

If you are experiencing the following errors when using the [Azure Pipelines app for Microsoft Teams](#), follow the procedures in this section.

- [Sorry, something went wrong. Please try again.](#)
- [Configuration failed. Please make sure that the organization '{organization name}' exists and that you have sufficient permissions.](#)

## Sorry, something went wrong. Please try again.

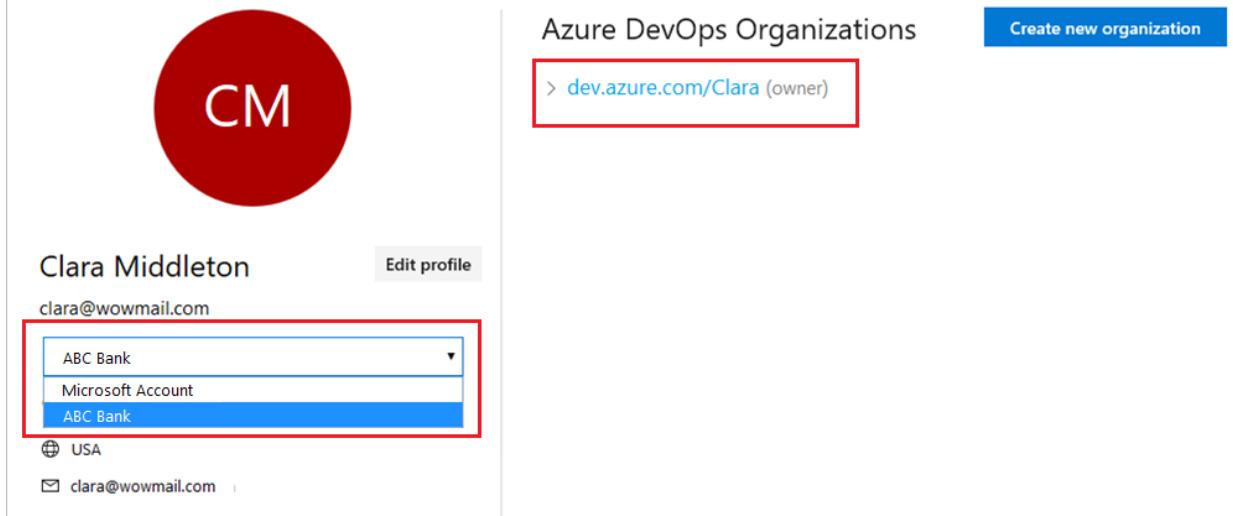
The Azure Pipelines app uses the OAuth authentication protocol, and requires [Third-party application access via OAuth for the organization](#) to be enabled. To enable this setting, navigate to **Organization Settings > Security > Policies**, and set the **Third-party application access via OAuth for the organization** setting to **On**.

The screenshot shows the 'Organization Settings' page in Azure DevOps. The left sidebar has a red box around 'Organization Settings' (labeled 1) and another red box around 'Policies' (labeled 2). The main content area shows the 'Policy' section with 'Application connection policies' and 'Security policies'. The 'Third-party application access via OAuth' option under 'Application connection policies' is highlighted with a red box and labeled 3. It is currently set to 'Off'. Other options like 'Alternate authentication credentials' and 'SSH authentication' are also listed. The 'Security policies' section includes 'Allow public projects' and 'Enable Azure Active Directory Conditional Access Policy Validation'. The 'User policies' section includes 'External guest access'.

## Configuration failed. Please make sure that the organization '{organization name}' exists and that you have sufficient permissions.

Sign out of Azure DevOps by navigating to <https://aka.ms/VsSignout> using your browser.

Open an **In private or incognito** browser window and navigate to <https://aex.dev.azure.com/me> and sign in. In the dropdown under the profile icon to the left, select the directory that contains the organization containing the pipeline for which you wish to subscribe.



Azure DevOps Organizations [Create new organization](#)

Clara Middleton [Edit profile](#)

clara@wowmail.com

ABC Bank

Microsoft Account

ABC Bank

USA

clara@wowmail.com

In the **same browser**, start a new tab and sign in to <https://teams.microsoft.com/>. Run the `@Azure Pipelines signout` command and then run the `@Azure Pipelines signin` command in the channel where the Azure Pipelines app for Microsoft Teams is installed.

Select the **Sign in** button and you'll be redirected to a consent page like the one in the following example. Ensure that the directory shown beside the email is same as what was chosen in the previous step. Accept and complete the sign-in process.



Visual Studio | My Benefits

Clara Sign out

## Azure Pipelines Microsoft Teams Integration by Azure DevOps

App requests the following permissions from: **clara@wowmail.com (ABC Bank)**

### Service hooks (read and write)

Grants the ability to create and update service hook subscriptions and read metadata, including supported events, consumers, and actions.

### Build (read and execute)

Grants the ability to access build artifacts, including build results, definitions, and requests, and the ability to queue a build, update build properties, and the ability to receive notifications about build events via service hooks.

### Release (read, write, execute and manage)

Grants the ability to read, update and delete release artifacts, including folders, releases, release definitions and release environment and the ability to queue and approve a new release.

### Project and team (read)

Grants the ability to read projects and teams.

### Teams Integration

Grants this application the ability to act on your behalf on Azure DevOps from within Microsoft Teams

[Learn more](#)

If you change your mind at any time, you can manage authorizations on your [profile page](#).

**Accept**

**Deny**

By clicking **Accept**, you allow this app to perform the above actions on your behalf and you agree to Azure DevOps Terms of Use and Privacy Statement.

If these steps don't resolve your authentication issue, reach out to us at [Developer Community](#).

## Related articles

- [Azure Boards with Microsoft Teams](#)
- [Azure Repos with Microsoft Teams](#)

# Azure Pipelines with Slack

11/2/2020 • 6 minutes to read • [Edit Online](#)

## Azure Pipelines

If you use [Slack](#), you can use the [Azure Pipelines app for Slack](#) to easily monitor the events for your pipelines. Set up and manage subscriptions for builds, releases, YAML pipelines, pending approvals and more from the app and get notifications for these events in your Slack channels.

The screenshot shows a Slack conversation with the Azure Pipelines app. The messages include:

- Deployment of release [Release-ChatOps\\_Deploy-1](#) on stage **Dev** started
- Release pipeline [ChatOps\\_Deploy](#)
- Deployment of release [Release-ChatOps\\_Deploy-1](#) on stage **QA** started
- Release pipeline [ChatOps\\_Deploy](#)
- Pre-deployment approval pending for release [Release-ChatOps\\_Deploy-1](#) on stage **QA**
- Release pipeline [ChatOps\\_Deploy](#)
- Branch [refs/heads/master](#)
- Approvers Atin Bansal
- Build [ChatOps-CL\\_master\\_20190202.1](#)
- Pending since 02-February-2019 06:55:52 AM (UTC)
- Deployment of release [Release-ChatOps\\_Deploy-1](#) on stage **Dev** succeeded
- Release pipeline [ChatOps\\_Deploy](#)
- Duration 00:01:01

At the bottom, there is a message input field with a plus icon, the text "Message #test", and reaction icons (@ and 😊).

### NOTE

This feature is only available on Azure DevOps Services. Typically, new features are introduced in the cloud service first, and then made available on-premises in the next major version or update of Azure DevOps Server. To learn more, see [Azure DevOps Feature Timeline](#).

## Add the Azure Pipelines app to your Slack workspace

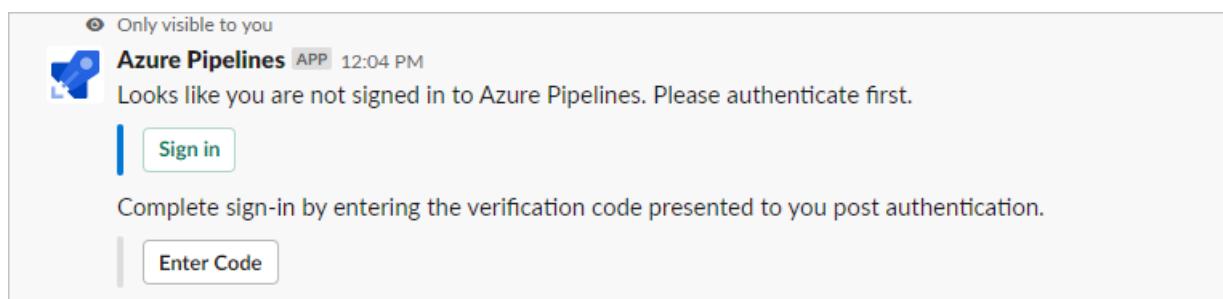
Navigate to [Azure Pipelines Slack app](#) to install the Azure Pipelines app to your Slack workspace. Once added, you will see a welcome message from the app as below. Use the `/azpipelines` handle to start interacting with the app.

The welcome message from the Azure Pipelines app is:

Hi! Let's start monitoring your pipelines  
Subscribe to one or more pipelines with: `/azpipelines subscribe [pipeline url]`  
To see what else you can do, type `/azpipelines help`

# Connect the Azure Pipelines app to your pipelines

Once the app has been installed in your Slack workspace, you can connect the app to the pipelines you want to monitor. The app will ask you to authenticate to Azure Pipelines before running any commands.



A screenshot of the Azure Pipelines app in a Slack workspace. The message is from the Azure Pipelines app at 12:04 PM. It says "Only visible to you" and "Looks like you are not signed in to Azure Pipelines. Please authenticate first." There are two buttons: "Sign in" and "Enter Code". Below the message, it says "Complete sign-in by entering the verification code presented to you post authentication."

To start monitoring all pipelines in a project, use the following slash command inside a channel:

```
/azpipelines subscribe [project url]
```

The project URL can be to any page within your project (except URLs to pipelines).

For example:

```
/azpipelines subscribe https://dev.azure.com/myorg/myproject/
```

You can also monitor a specific pipeline using the following command:

```
/azpipelines subscribe [pipeline url]
```

The pipeline URL can be to any page within your pipeline that has `definitionId` or `buildId/releaseId` in the URL.

For example:

```
/azpipelines subscribe https://dev.azure.com/myorg/myproject/_build?definitionId=123
```

or:

```
/azpipelines subscribe https://dev.azure.com/myorg/myproject/_release?definitionId=123&view=mine&_a=releases
```

The subscribe command gets you started with a few subscriptions by default. For Build pipelines, the channel is subscribed to *Build completed* notification. For Release pipelines, the channel will start receiving *Release deployment started*, *Release deployment completed* and *Release deployment approval pending* notifications. For YAML pipelines, subscriptions are created for the *Run stage state changed* and *Run stage waiting for approval* notifications.

`@Atin` has added subscriptions to **Deployment started**, **Deployment completed** and **Deployment approval pending** notifications for `ChatOps_Deploy`

## Manage subscriptions

To manage the subscriptions for a channel, use the following command:

```
/azpipelines subscriptions
```

This command will list all the current subscriptions for the channel and allow you to add new subscriptions.

[!NOTE] Team administrators aren't able to remove or modify subscriptions created by Project administrators.

The screenshot shows a Microsoft Teams channel interface with the Azure Pipelines app. The channel header says "Only visible to you". The Azure Pipelines app card displays the following information:

- Azure Pipelines APP 12:48 PM**
- This channel has 4 subscriptions from 2 pipelines:
- ChatOps\_Deploy**  
Release deployment approval pending  
For any stage with any approval type  
**Remove**
- ChatOps\_Deploy**  
Release deployment completed  
For any stage with any environment status  
**Remove**
- ChatOps\_Deploy**  
Release deployment started  
For any stage  
**Remove**
- ChatOps-CI**  
Build completed  
For any build status  
**Remove**

At the bottom of the list is a button labeled **Add subscription...**.

## Using filters effectively to customize subscriptions

When a user subscribes to any pipeline, a few subscriptions are created by default without any filters being applied. Often, users have the need to customize these subscriptions. For example, users may want to hear only about failed builds or get notified only when deployments are pushed to production. The Azure Pipelines app supports filters to customize what you see in your channel.

1. Run the `/azpipelines subscriptions` command
2. In the list of subscriptions, if there is a subscription that is unwanted or must be modified (Example: creating noise in the channel), select the **Remove** button
3. Select the **Add subscription** button
4. Select the required pipeline and the desired event
5. Select the appropriate filters to customize your subscription

### Example: Get notifications only for failed builds

The screenshot shows the "Add subscription" dialog for the Azure Pipelines app. The steps are as follows:

1. Pipeline: ChatOps
2. Event: Build completed
3. Filter: Build Status - Failed (This step is highlighted with a red box)

At the bottom are **Cancel** and **Save** buttons.

### Example: Get notifications only if the deployments are pushed to production environment

Azure Pipelines APP 6:32 PM

Add a new subscription to this channel. Can't find a pipeline? Subscribe to a new one with: /azpipelines subscribe [pipeline url]

Pipeline

Cortana Controllers

Which event would you like to add subscription for?

Release deployment...

Stage name

Prod

Approval type

Pre-deployment

Cancel Save

## Approve deployments from your channel

You can approve deployments from within your channel without navigating to the Azure Pipelines portal by subscribing to the *Release deployment approval pending* notification for classic Releases or the *Run stage waiting for approval* notification for YAML pipelines. Both of these subscriptions are created by default when you subscribe to the pipeline.

Christie Church 6:55 PM  
@Jamal I have created a release to deploy hotfix for bug #1441

Azure Pipelines APP 6:56 PM  
Pre-deployment approval pending for release **Prod Release-6** on stage **Linux\_Deployment**

Hotfix deployment to address bug #1441 - global scorecard

<b>Release pipeline</b>	<b>Artifacts</b>
Integrated_Reportng	879767c0b

**Branch**  
master

**Approvers**  
**Jamal Hartnett**

Approve Reject

Whenever the running of a stage is pending for approval, a notification card with options to approve or reject the request is posted in the channel. Approvers can review the details of the request in the notification and take appropriate action. In the following example, the deployment was approved and the approval status is displayed on the card.

Azure Pipelines APP 6:56 PM  
Pre-deployment approval pending for release **Prod Release-6** on stage **Linux\_Deployment**

Hotfix deployment to address bug #1441 - global scorecard

<b>Release pipeline</b>	<b>Artifacts</b>
Integrated_Reportng	879767c0b

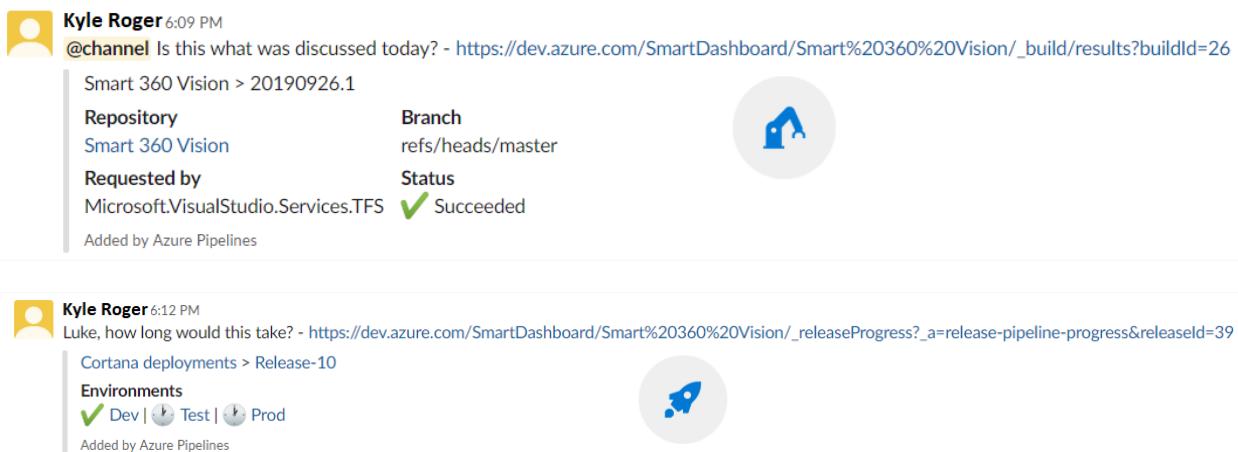
**Branch**  
master

**Approvers**  
**Jamal Hartnett**

The app supports all the checks and approval scenarios present in Azure Pipelines portal, like single approver, multiple approvers (any one user, any order, in sequence) and teams as approvers. You can approve requests as an individual or on behalf of a team.

## Previews of pipeline URLs

When a user pastes a pipeline URL, a preview is shown similar to that in the following image. This helps to keep pipeline related conversations relevant and accurate.



The screenshot shows a Microsoft Teams channel interface. A message from Kyle Roger at 6:09 PM asks if a specific pipeline was discussed today, providing a link to the Azure DevOps pipeline details. Below this, another message from Kyle Roger at 6:12 PM asks about the duration of a release, also providing a link to the release progress page. Each message includes a preview card for the respective pipeline or release, showing details like repository, branch, status, environments (Dev, Test, Prod), and deployment steps.

For this feature to work, users have to be signed-in. Once they are signed in, this feature will work for all channels in a workspace.

## Remove subscriptions and pipelines from a channel

If you want to clean up your channel, use the following commands to unsubscribe from all pipelines within a project.

```
/azpipelines unsubscribe all [project url]
```

For example:

```
/azpipelines unsubscribe all https://dev.azure.com/myorg/myproject
```

This command deletes all the subscriptions related to any pipeline in the project and removes the pipelines from the channel.

### IMPORTANT

Only project administrators can run this command.

## Commands reference

Here are all the commands supported by the Azure Pipelines app:

SLASH COMMAND	FUNCTIONALITY
/azpipelines subscribe [pipeline url/ project url]	Subscribe to a pipeline or all pipelines in a project to receive notifications
/azpipelines subscriptions	Add or Remove subscriptions for this channel
/azpipelines feedback	Report a problem or suggest a feature
/azpipelines help	Get help on the slash commands

SLASH COMMAND	FUNCTIONALITY
/azpipelines signin	Sign in to your Azure Pipelines account
/azpipelines signout	Sign out from your Azure Pipelines account
/azpipelines unsubscribe all [project url]	Remove all pipelines (belonging to a project) and their associated subscriptions from a channel

## Notifications in Private channels

The Azure Pipelines app can help you monitor the pipelines activity in your private channels as well. You will need to invite the bot to your private channel by using `/invite @azpipelines`. Post that, you can set up and manage your notifications the same way as you would for a public channel.

### NOTE

- You can use the Azure Pipelines app for Slack only with a project hosted on Azure DevOps Services at this time.
- The user has to be an admin of the project containing the pipeline to set up the subscriptions
- Notifications are currently not supported inside direct messages
- Deployment approvals which have 'Revalidate identity of approver before completing the approval' policy applied, are not supported
- 'Third party application access via OAuth' must be enabled to receive notifications for the organization in Azure DevOps (Organization Settings -> Security -> Policies)

## Troubleshooting

If you are experiencing the following errors when using the [Azure Pipelines App for Slack](#), follow the procedures in this section.

- [Sorry, something went wrong. Please try again.](#)
- [Configuration failed. Please make sure that the organization '{organization name}' exists and that you have sufficient permissions.](#)

### **Sorry, something went wrong. Please try again.**

The Azure Pipelines app uses the OAuth authentication protocol, and requires [Third-party application access via OAuth for the organization](#) to be enabled. To enable this setting, navigate to **Organization Settings > Security > Policies**, and set the **Third-party application access via OAuth for the organization** setting to **On**.

The screenshot shows the 'Organization Settings' page under 'ABC Domestic Banking'. The left sidebar has a red box around 'Organization Settings' (labeled 1) and another red box around 'Policies' (labeled 2). A third red box labeled 3 highlights the 'Third-party application access via OAuth' policy, which is set to 'Off'. Other policies shown include 'Alternate authentication credentials' (Off), 'SSH authentication' (On), 'Allow public projects' (On), and 'Enable Azure Active Directory Conditional Access Policy Validation' (Off).

**Configuration failed. Please make sure that the organization '{organization name}' exists and that you have sufficient permissions.**

Sign out of Azure DevOps by navigating to <https://aka.ms/VsSignout> using your browser.

Open an **In private or incognito** browser window and navigate to <https://aex.dev.azure.com/me> and sign in. In the dropdown under the profile icon to the left, select the directory that contains the organization containing the pipeline for which you wish to subscribe.

The screenshot shows the user profile page for Clara Middleton. It includes her profile picture (CM), name, email (clara@wowmail.com), and a dropdown menu for selecting an organization. The dropdown shows 'ABC Bank', 'Microsoft Account', and 'ABC Bank' again, with 'ABC Bank' currently selected. To the right, it shows 'Azure DevOps Organizations' with a link to 'dev.azure.com/Clara (owner)'. There is also a 'Create new organization' button.

In the **same browser**, start a new tab, navigate to <https://slack.com>, and sign in to your work space (**use web client**). Run the `/azpipelines signout` command followed by the `/azpipelines signin` command.

Select the `Sign in` button and you'll be redirected to a consent page like the one in the following example. Ensure that the directory shown beside the email is same as what was chosen in the previous step. Accept and complete the sign-in process.

## Azure Pipelines Slack integration by Azure DevOps

App requests the following permissions from: **clara@wowmail.com (ABC Bank)**

### Service hooks (read and write)

Grants the ability to create and update service hook subscriptions and read metadata, including supported events, consumers, and actions.

### Build (read and execute)

Grants the ability to access build artifacts, including build results, definitions, and requests, and the ability to queue a build, update build properties, and the ability to receive notifications about build events via service hooks.

### Release (read, write, execute and manage)

Grants the ability to read, update and delete release artifacts, including folders, releases, release definitions and release environment and the ability to queue and approve a new release.

### Project and team (read)

Grants the ability to read projects and teams.

### Slack integration

Grants this application the ability to act on your behalf on Azure DevOps from within Slack

[Learn more](#)

If you change your mind at any time, you can manage authorizations on your [profile page](#).

**Accept**

**Deny**

By clicking **Accept**, you allow this app to perform the above actions on your behalf and you agree to Azure DevOps Terms of Use and Privacy Statement.

If these steps don't resolve your authentication issue, reach out to us at [Developer Community](#).

## Related articles

- [Azure Boards with Slack](#)
- [Azure Repos with Slack](#)
- [Create a service hook for Azure DevOps with Slack](#)

# Integrate with ServiceNow change management

6/18/2021 • 6 minutes to read • [Edit Online](#)

## Azure Pipelines

Azure Pipelines and ServiceNow bring an integration of Azure Pipelines with ServiceNow Change Management to enhance collaboration between development and IT teams. By including change management in CI/CD pipelines, teams can reduce the risks associated with changes and follow service management methodologies such as ITIL, while gaining all DevOps benefits from Azure Pipelines.

This topic covers:

- Configuring ServiceNow for integrating with Azure Pipelines
- Including ServiceNow change management process as a release gate or a check
- Monitoring change management process from releases and pipeline runs
- Keeping ServiceNow change requests updated with deployment result

## Prerequisites

This tutorial extends [Use approvals and gates](#) and [Define approvals & checks](#).

You'll need a non-developer instance of [ServiceNow](#) to which applications can be installed from the store.

## Configure the ServiceNow instance

1. Install the [Azure Pipelines](#) application on your ServiceNow instance. You'll require Hi credentials to complete the installation. Refer to the [documentation](#) for more details on getting and installing applications from the ServiceNow store.
2. Create a new user in ServiceNow and grant it the `x_mioms_azpipeline.pipelinesExecution` role.

User  
Azure Pipelines Service Account

User ID	AzPipelinesUser	Email	
First name	Azure Pipelines Service Account	Calendar integration	Outlook
Last name		Time zone	System (US/Pacific)
Title		Date format	System (yyyy-MM-dd)
Department		Business phone	
Password		Mobile phone	
Password needs reset	<input type="checkbox"/>	Photo	Click to add...
Locked out	<input type="checkbox"/>		
Active	<input checked="" type="checkbox"/>		
Web service access only	<input type="checkbox"/>		
Internal Integration User	<input type="checkbox"/>		
SSO Source			

**Related Links**

- [View Subscriptions](#)
- [Reset a password](#)

**Roles**

	Role	State	Granted by	Inherited
<input type="checkbox"/>	x_mioms_azpipeline.pipelinesExecution	Active	(empty)	false
<input type="checkbox"/>	import_transformer	Active	(empty)	true

## Set up the Azure DevOps organization

1. Install the [ServiceNow Change Management extension](#) on your Azure DevOps organization.

Visual Studio | Marketplace

Azure DevOps > Azure Pipelines > ServiceNow Change Management

**ServiceNow Change Management**  
Microsoft | 125 installs | ★★★★★ (0) | Free

Integrate ServiceNow Change Management with Azure Pipelines

[Get it free](#)

[Overview](#) [Q & A](#) [Rating & Review](#)

**ServiceNow Change Management Extension**

ServiceNow is a software-as-a-service (SaaS) provider of IT service management (ITSM) software, including change management. Specific change management subprocesses include change risk assessment, change scheduling, change approvals and oversight. With change management, your organization can reduce the risks associated with change, while speeding up the deployments with Azure Pipelines.

This extension enables integration of ServiceNow Change Management with Azure Pipelines. It includes a release gate to create a change request in ServiceNow and hold the pipeline till the change management process signals the implementation. An agentless task to close (update state of) the change request after the deployment is also provided.

The deployment process in Azure Pipelines helps in automation of the deployment and complement the controls offered by ServiceNow.

**Usage**

Integration requires the [Azure Pipelines application](#) to be installed on the ServiceNow instance. A service account that has been created in ServiceNow and granted the [x\\_mioms\\_azpipeline.pipelinesExecution role](#) would be used for all the communication.

**Categories**

- Azure Pipelines

**Tags**

- Change Management
- DevOps
- Pipelines
- Release
- Release Gates
- ServiceNow
- Utility task

Follow the instructions to "Get it Free"

2. Create a new ServiceNow [service connection](#) in the Azure DevOps project used for managing your pipelines. You can use basic auth and use the user name, password for the service account created in ServiceNow. Alternatively, you can also authenticate using OAuth. [Learn More](#)

**New ServiceNow service connection** X

Authentication method

Basic Authentication  
 OAuth2

ServiceNow Url

Url of the ServiceNow instance to connect to.

**Authentication**

Username (optional)

Username

Password (optional)

Password

**Verify**

Details

Service connection name

[Learn more](#) [Troubleshoot](#) Back **Verify and save** ▾

## Designer release pipelines

### Add a gate to the pipeline

1. In your release pipeline, add a pre-deployment gate for ServiceNow Change Management.

Pipeline Tasks Variables Retention Options History

+ Add ↞

Pre-deployment conditions

Prod

Triggers ↞ Define the trigger that will start deployment to this stage

- Invoke Azure Function
- Invoke REST API
- Query Azure Monitor Alerts
- Query Work Items
- ServiceNow Change Management

Pre-deployment approvers ↞ Select the users who can approve this deployment

Gates ↞ Define gates to evaluate before deployment

The delay before evaluation: 5

Deployment gates ↞ + Add ↞

2. Select the ServiceNow service connection you created earlier and enter the values for the properties of the change request.

Deployment gates [\(1\)](#) [+ Add](#) [v](#)

**ServiceNow change management** [Enabled](#) [Edit](#)

ServiceNow Change Management [\(1\)](#)

Display name [\\*](#)  
ServiceNow change management

ServiceNow connection [\(1\)](#) | [Manage](#) [\(1\)](#)

VendorInstance [\(1\)](#) [\(1\)](#) [New](#)

Short Description [\\*](#) [\(1\)](#)  
Deployment to \${Release.EnvironmentName} of Release \${Release.ReleaseId}

Description [\(1\)](#)  
Deployment to \${Release.EnvironmentName} of Release \${Release.ReleaseId}

Category [\\*](#) [\(1\)](#)  
Software [\(1\)](#) [\(1\)](#)

Priority [\\*](#) [\(1\)](#)  
Low [\(1\)](#) [\(1\)](#)

Risk [\(1\)](#)  
Low [\(1\)](#) [\(1\)](#)

Impact [\\*](#) [\(1\)](#)  
2 - Medium [\(1\)](#) [\(1\)](#)

Configuration Item [\(1\)](#)  
7300 [\(1\)](#) [\(1\)](#)

Assignment Group [\(1\)](#)  
eCAB Approval [\(1\)](#) [\(1\)](#)

Schedule of change request [▼](#)

Advanced [▼](#)

Success criteria [^](#)

Desired status of change request [\\*](#) [\(1\)](#)  
Implement [\(1\)](#) [\(1\)](#)

Output Variables [^](#)

Reference name [\(1\)](#)  
gate1 [\(1\)](#)

Variables list

- gate1.CHANGE\_REQUEST\_NUMBER [\(1\)](#)
- gate1.CHANGE\_SYSTEM\_ID [\(1\)](#)

### Inputs for the gate:

- **ServiceNow connection:** Connection to the ServiceNow instance used for change management.
- **Action:** Gate on status of new change request or an existing change request.
- **Change type:** Type of the change request.
- **Standard change template:** Change template name for the change request.
- **Short description:** A summary of the change.
- **Change query criteria:** Criteria for querying change request. Must uniquely identify the change request. Gate would fail if multiple matching change requests are found.
- **Query string/Change request number:** Change request to use.

Additional properties can be set in the created change request using the following inputs. **Note:** Available inputs change based on the selected change type.

- **Description:** A detailed description of the change.
- **Category:** The category of the change eg. Hardware, Network, Software.
- **Priority:** Priority of the change.
- **Risk:** The risk level for the change.
- **Impact:** The effect that the change has on business.
- **Configuration Item:** Configuration item (CI) that the change applies to.
- **Assignment group:** The group that the change is assigned to.
- **Schedule of change request:** Schedule of the change as honored by the ServiceNow workflow.  
Date and time should be in UTC and format should be  
`yyyy-MM-ddTHH:mm:ssZ`. eg. `2018-01-31T07:56:59Z`.
- **Additional change request parameters:** Additional properties of the change request to set. Name must be field name (not label) prefixed with 'u\_' eg. `u_backout_plan`. Value must be a valid, accepted value in ServiceNow. Invalid entries are ignored.

#### Gate Success Criteria :

- **Desired state of change request:** The gate would succeed and the pipeline continues when the change request status is same as the provided value.
- **Advanced:** Specifies an expression that controls when this gate should succeed. The change request is captured as `root['result']` in the response from ServiceNow. Example -  
`and(eq(root['result'].state, 'New'), eq(root['result'].risk, 'Low'))`. [Learn more](#).

#### Gate Output Variables :

ServiceNow gate produces output variables. You must specify reference name to be able to use these output variables in the deployment workflow. Gate variables can be accessed by using `"PREDEPLOYGATE"` as a `prefix` in an agentless job in the workflow. For eg. when reference name is set to 'gate1', then the change number can be obtained as `$(PREDEPLOYGATE.gate1.CHANGE_REQUEST_NUMBER)`.

- **CHANGE\_REQUEST\_NUMBER** : Number of the change request.
- **CHANGE\_SYSTEM\_ID** : Sys Id of the change request.

3. At the end of your deployment process, add an agentless phase with a task to update the status of the change after deployment.

Pipeline	Tasks	Variables	Retention	Options	History
Prod					
Deployment process					
Agent job					
Run on agent					
Deploy application					
PowerShell					
Agentless job					
Run on server					
now Update ServiceNow Change Request					
Update ServiceNow Change Request					

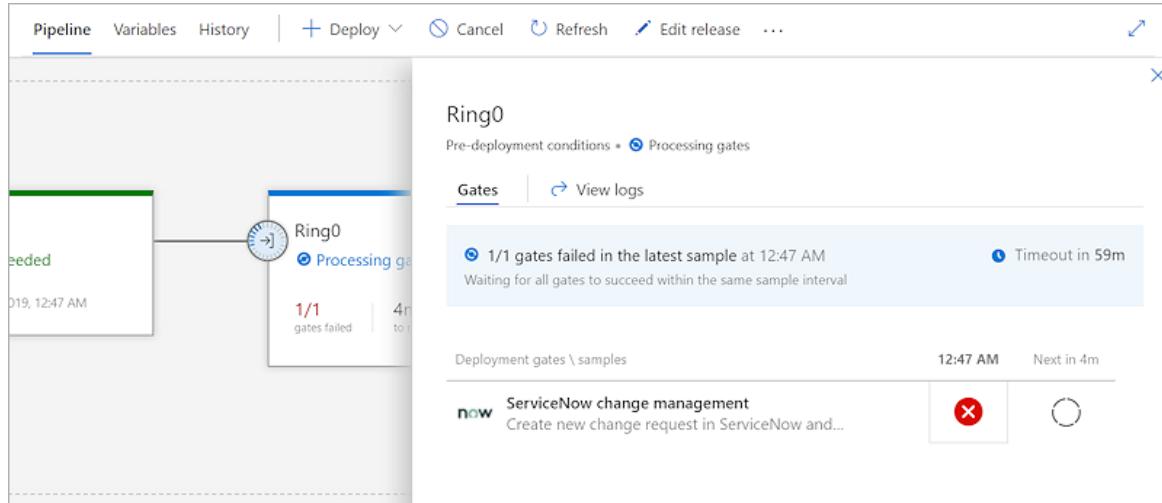
Inputs for Update change request task:

- **ServiceNow connection:** Connection to the ServiceNow instance used for change management.
- **Change request number:** Number of the change request to update.
- **Update status:** Select this option to update status of the change request.
- **Updated status of change request :** Status to set for the change request. This input is available if **Update status** is selected.
- **Close code and notes:** Closure information for the change request.
- **Work notes:** Work notes for the change request.
- **Additional change request parameters:** Additional properties of the change request to set.

The update task would fail if none of the fields in the change request are updated during the execution of the task. ServiceNow ignores invalid fields and values passed to the task.

## Execute a release

1. Create a new release from the configured release pipeline in Azure DevOps
2. After completing the Dev stage, the pipeline creates a new change request in ServiceNow for the release and waits for it to reach the desired state.



3. The values defined as gate parameters will be used. You can get the change number that was created from the logs.

```
✖ ServiceNow Change Management
✖ Sample at 12:55 AM - 1/1 gates Failed (latest) ▾ | ✎
1 2019-01-09T19:25:07.9589086Z POST https://ven01307.service-now.com//api/now/import/x_mioms_azpipeline_change_request_import
2 Request body: { "u_correlation_id": "d0b40ea2-58d5-4540-8c76-39c0f133dbd2", "u_x_mioms_azpi_eline_metadata": "Release: http://
3 Response Code: Created
4 Response: {"import_set": "ISET0010003", "staging_table": "x_mioms_azpipeline_change_request_import", "result": [
5 2019-01-09T19:25:09.5867362Z GET https://ven01307.service-now.com/api/now/table/change_request/5ecc89570f7a63001385c5ece10
6 Response Code: OK
7 Response: {"result": {"number": "CHG0030033", "state": "New"}}
8 Evaluation of expression 'eq(jsonPath('$result.state')[0], 'Implement')' failed.
9
```

4. The ServiceNow change owner will see the release in the queue as a new change.

5. The release that caused the change to be requested can be tracked from the **Azure DevOps Pipeline metadata** section of the change.

Azure DevOps Pipeline metadata

Pipeline metadata

```
Release: https://shashankb.visualstudio.com/f26adeb3-404e-49f8-8124-ac20d754ec32/_release?releaseId=437&_a=release-summary
EnvironmentName: Ring0
Requested By : Shashank S
```

6. The change goes through its normal life cycle: Approval, Scheduled, and more until it is ready for implementation.
7. When the change is ready for implementation (it is in the Implement state), the release in Azure DevOps proceeds. The gates status will be as shown here:

8. After the deployment, the change request is closed automatically.

## Yaml pipelines

This tutorial assumes you have a yaml pipeline with a single stage that deploys to an environment called 'latest'.

### Add a check

1. Add the 'ServiceNow Change Management' check to the environment. The check configuration details remain the same as the gate in designer release pipelines.

The screenshot shows the 'Add check' interface in Azure DevOps. At the top, there's a search bar labeled 'Search check types'. Below it is a list of check types, each with an icon and a brief description. The 'ServiceNow Change Management' option is highlighted with a red border around its icon and name.

Type	Display name	Timeout
ServiceNow Change Management	ServiceNow Change Management	30d

- Approvals  
Approvers should grant approval for deployment
- Branch control  
Allow deployments based on branches linked to the run
- Business Hours  
Ensure the deployment is started in a specific time window
- Invoke Azure Function  
Invoke an Azure Function
- Invoke REST API  
Invoke a REST API as a part of your pipeline.
- Query Azure Monitor alerts  
Observe the configured Azure Monitor rules for active alerts
- Required template  
Ensure the pipeline extends one or more YAML templates
- ServiceNow Change Management  
Gate on status of new change request or an existing change request in ServiceNow.

### Add the update change task to the job

1. Add a server job to the specific stage and execute the 'UpdateChangeRequest' task in it. The 'changenumber' field is optional. We shall leave it empty, so the pipeline updates the change created for the current stage.

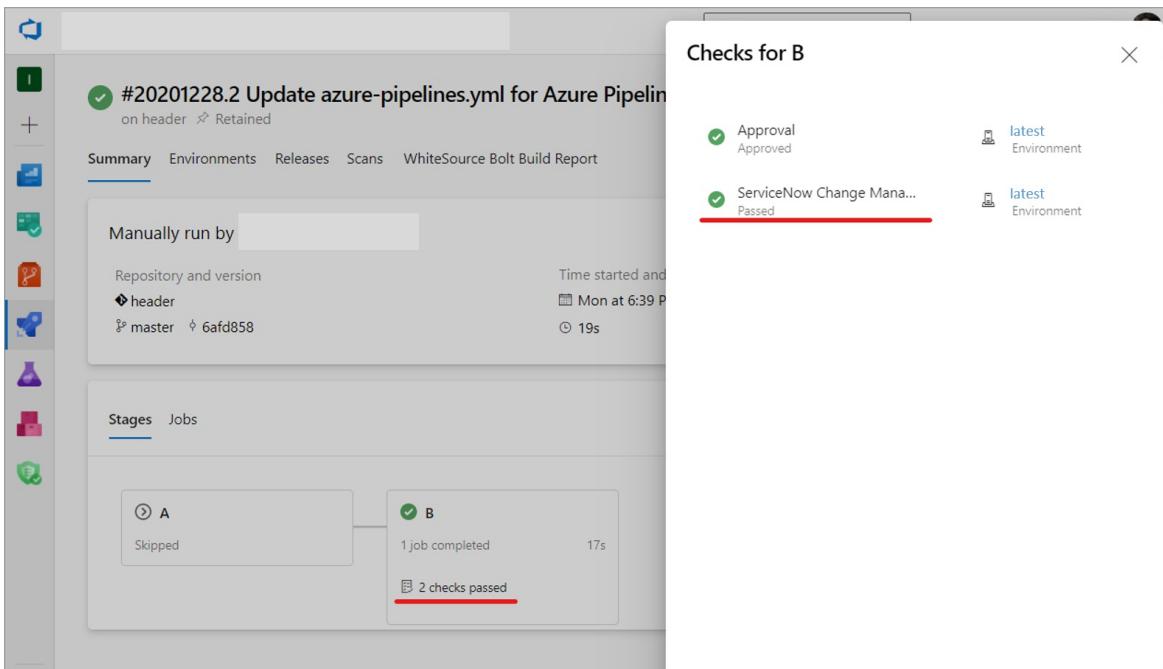
```

1 stages:
2 - stage: Stage1
3 jobs:
4 - deployment: DeployLatest
5 environment: Latest
6 # Rest of the tasks
7 - job: UpdateChangeRequest
8 steps:
9 - task: UpdateServiceNowChangeRequest@2
10 displayName: 'Update ServiceNow Change Request'
11 inputs:
12 ServiceNowConnection: 'snow06'
13 NewStatus: 'Assess'
14 WorkNotes: 'sample work notes'
15 otherParameters: '{"u_hello": "hello"}'
16 pool: server

```

### Run the pipeline

1. Create a new run for the pipeline
2. At the start of the stage. A new change request would be automatically created with the specified properties. The pipeline would wait for check to complete i.e. for the change to reach the desired state.
3. Once the check is complete, the stage shall start execution. The change would be closed when update task at the end of the stage executes.



## FAQs

### Q: What versions of ServiceNow are supported?

A: The integration is compatible with Kingston, London, and New York versions of ServiceNow.

### Q: What types of change request can be managed with the integration?

A: Normal, standard and emergency change requests are supported with this integration.

### Q: How do I set additional change properties?

A: You can specify additional change properties of the change request in the **Additional change request parameters** field. The properties are specified as key-value pairs in JSON format, the name being the field name (not the label) prefixed with `u_` in ServiceNow and a valid value.

### Q: Can I update custom fields in the change request with additional change request parameters?

A: If custom fields are defined in ServiceNow for the change requests, mapping of the custom fields in import set transform map must be added. See [ServiceNow Change Management Extension](#) for details.

### Q: I don't see drop-down values populated for Category, Status, and others. What should I do?

A: Change Management Core and Change Management - State Model plugins must be active on your ServiceNow instance for the drop-downs to work. See [Upgrade change management](#) and [Update change request states](#) for more details.

## Related topics

- [Approvals and gates overview](#)
- [Approvals and checks](#)
- [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
- Get advice on [Stack Overflow](#), and feel free to post your questions, search for answers, or suggest a feature on our [Azure DevOps Developer Community Support](#) page.

# Continuously deploy from a Jenkins build

4/20/2021 • 4 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

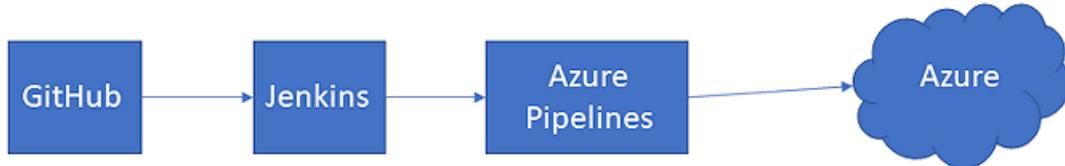
## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Azure Pipelines supports integration with Jenkins so that you can use Jenkins for Continuous Integration (CI) while gaining several DevOps benefits from an Azure Pipelines release pipeline that deploys to Azure:

- Reuse your existing investments in Jenkins build jobs
- Track work items and related code changes
- Get end-to-end traceability for your CI/CD workflow
- Consistently deploy to a range of cloud services
- Enforce quality of builds by gating deployments
- Define work flows such as manual approval processes and CI triggers
- Integrate Jenkins with JIRA and Azure Pipelines to show associated issues for each Jenkins job
- Integrate with other service management tools such as [ServiceNow](#)

A typical approach is to use Jenkins to build an app from source code hosted in a Git repository such as GitHub and then deploy it to Azure using Azure Pipelines.



## Before you begin

- You'll need the source code for your app hosted in a repository such as GitHub, Azure Repos, GitHub Enterprise Server, Bitbucket Cloud, or any another source control provider that Jenkins can interact with.
- You'll need a Jenkins server where you run your CI builds. You can quickly [set up a Jenkins server on Azure](#).
- You'll need a Jenkins project that builds your app. For example, you can [build a Java app with Maven](#) on Jenkins.

## Link Jenkins with Azure Pipelines

Create a Jenkins service connection from the **Service connections** section of the project settings page.

In TFS, open the **Services** page from the "settings" icon in the top menu bar.

For more information, see [Jenkins service connection](#). If you are not familiar with the general concepts in this section, see [Accessing your project settings](#) and [Creating and using a service connection](#).

## Add a Jenkins artifact

Create a new release pipeline and add a Jenkins artifact to it. After you select the Jenkins service connection, you can select an existing Jenkins job to deploy.

It's possible to [store the output from a Jenkins build in Azure blob storage](#). If you have configured this in your Jenkins project, choose **Download artifacts from Azure storage** and select the default version and source alias.

For more information, see [Jenkins artifacts](#). If you are not familiar with the general concepts in this section, see [Creating a release pipeline](#) and [Release artifacts and artifact sources](#).

## Define the deployment steps

Add the tasks you require to deploy your app to your chosen target in the **Agent job** section in the **Tasks** page of your release pipeline. For example, add the **Azure App Service Deploy** task to deploy a web app.

- [YAML](#)
- [Classic](#)

Add the **Azure App Service Deploy** task YAML code to a job in the `.yml` file at the root of the repository.

```
...
jobs:
- job: DeployMyApp
 pool:
 name: Default
 steps:
- task: AzureRmWebAppDeployment@4
 inputs:
 connectionType: 'AzureRM'
 azureSubscription: your-subscription-name
 appType: webAppLinux
 webAppName: 'MyApp'
 deployToSlotOrASE: false
 packageForLinux: '$(System.DefaultWorkingDirectory)/**/*.zip'
 takeAppOfflineFlag: true
...
...
```

YAML builds aren't yet available on TFS.

Whenever you trigger your Azure release pipeline, the artifacts published by the Jenkins CI job are downloaded and made available for your deployment. You get full traceability of your workflow, including the commits associated with each job.

[See more details of the Azure App Service Deploy task](#) If you are not familiar with the general concepts in this section, see [Build and release jobs](#) and [Using tasks in builds and releases](#).

## Enable continuous deployment

If your Jenkins server is **hosted in Azure**, or your Azure DevOps organization has **direct visibility** to your Jenkins server, you can easily enable a continuous deployment (CD) trigger within your release pipeline that causes a release to be created and a deployment started every time the source artifact is updated.

To enable continuous deployment for an Azure hosted or directly visible Jenkins server:

1. Open the [continuous deployment trigger](#) pane from the **Pipelines** page of your release pipeline.

2. Change the setting to **Enabled**.
3. Choose **Add** and select the branch you want to create the trigger for. Or select the default branch.

However, if you have an **on-premises** Jenkins server, or your Azure DevOps organization **does not** have direct visibility to your Jenkins Server, you can trigger a release for an Azure pipeline from a Jenkins project using the following steps:

1. Create a [Personal Access Token](#) (PAT) in your Azure DevOps or TFS organization. Jenkins requires this information to access your organization. Ensure you keep a copy of the token information for upcoming steps in this section.
2. Install the [Team Foundation Server plugin](#) on your Jenkins server.
3. Within your Jenkins project, you will find a new post build action named **Trigger release in TFS/Team Services**. Add this action to your project.
4. Enter the collection URL for your Azure DevOps organization or TFS server as  
`https://<accountname>.visualstudio.com/DefaultCollection/`
5. Leave **username** empty and enter your PAT as the password.
6. Select the Azure DevOps project and the release definition to trigger.

Now a new CD release will be triggered every time your Jenkins CI job is completed.

## See also

- [Artifacts](#)
- [Stages](#)
- [Triggers](#)
- [YAML schema reference](#)

# Quickstart: Migrate your classic pipeline to YAML

3/6/2021 • 2 minutes to read • [Edit Online](#)

Get started with YAML pipelines by converting your existing Classic pipeline to use YAML. YAML-based pipelines enable you to implement your CI/CD strategy as code. You can use the **Export as YAML** option to export your YAML from an existing Classic pipeline.

## Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- An active Azure DevOps organization. [Sign up for Azure Pipelines](#).
- A working pipeline that uses the Classic user interface editor.

## Create a new YAML pipeline

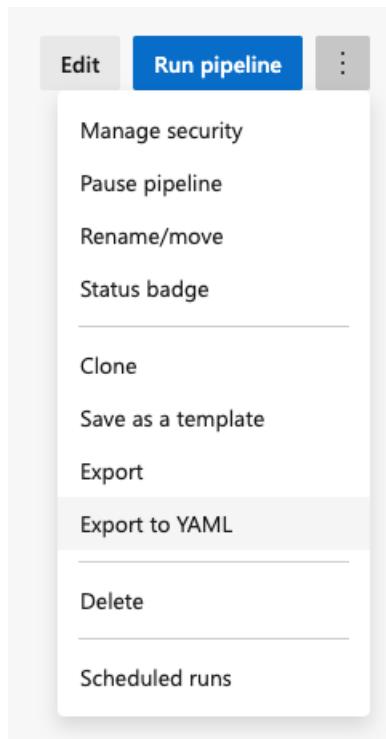
Before you get started, it is helpful to have a sample YAML pipeline file in your code. You'll update this YAML pipeline file later with your exported code from the Classic interface editor.

1. Sign in to your Azure DevOps organization and navigate to your project.
2. In your project, navigate to the Pipelines page. Then choose the action to create a new pipeline.
3. Select GitHub or Azure Repos as the location of your source code.
4. When the list of repositories appears, select your desired repository.
5. Azure Pipelines will analyze your repository and recommend a template. Use with **Starter template**.  
Select Save and run, then select Commit directly to the main branch, and then choose Save and run again.
6. A new run is started. Wait for the run to finish.

## Export your Classic pipeline

Export your Classic pipeline to a YAML file that you can use in the editor.

1. Select **Pipelines** in the menu and open your pipeline.
2. Open the context menu by selecting the ellipses.
3. Select **Export to YAML**.



4. Open the downloaded YAML file in your code editor.
5. If your YAML pipeline included variables defined in the Classic user interface, you'll need to define the variables again in your pipeline settings UI or in your YAML file. See [Define variables](#) to learn how to set variables in pipelines.
6. Review any `cron` schedules in your YAML file. By default, `cron` schedules in YAML are in UTC. In classic pipelines, they are in the organization's timezone. See [Configure schedules for pipelines](#) to learn more about configuring your pipeline to run on with a `cron` job.
7. Use the Task Assistant to make any other needed changes to the YAML file. The Task Assistant is a pane on the right side of the screen, which helps you correctly create and modify YAML steps.

```
master > PublicWeb / azure-pipelines.yml
1 type: git
2 ref: master
3
4 bs:
5 job: phase_1
6 displayName: Build WebSite
7 cancelTimeoutInMinutes: 1
8 pool:
9 vmImage: vs2017-win2016
10 steps:
11 - checkout: self
12 Settings
13 - task: UseDotNet@2
14 displayName: Use .NET Core sdk 2.1.403
15 inputs:
16 version: 2.1.403
17 Settings
18 - task: DotNetCoreCLI@2
19 displayName: Restore
20 inputs:
21 command: restore
22 projects: 'SmartHotel360.PublicWeb/SmartHotel360.PublicWeb.csproj'
23 Settings
24 - task: DotNetCoreCLI@2
25 displayName: Build
26 inputs:
```

8. Save and run your pipeline.

## Clean up resources

If you're not going to continue to use this pipeline, delete it from your project.

1. Select the ellipses (...) in the navigation and choose **Delete**.
2. Enter the name of your pipeline to permanently delete it and save.

## Next steps

Learn about the difference in features between YAML and Classic pipelines.

[Use Azure Pipelines](#)

# Migrate from Jenkins to Azure Pipelines

2/26/2020 • 7 minutes to read • [Edit Online](#)

Jenkins has traditionally been installed by enterprises in their own data centers and managed in an on-premises fashion, though a number of providers offer managed Jenkins hosting.

Azure Pipelines, on the other hand, is a cloud native continuous integration pipeline, providing the management of build and release pipelines and build agent virtual machines hosted in the cloud.

However, Azure Pipelines offers a fully on-premises option as well with [Azure DevOps Server](#), for those customers who have compliance or security concerns that require them to keep their code and build within the enterprise data center.

In addition, Azure Pipelines supports a hybrid cloud and on-premises model, where Azure Pipelines manages the build and release orchestration and enabling build agents both in the cloud and installed on-premises, for customers with custom needs and dependencies for some build agents but who are looking to move most workloads to the cloud.

This document provides a guide to translate a Jenkins pipeline configuration to Azure Pipelines, information about moving container-based builds and selecting build agents, mapping environment variables, and how to handle success and failures of the build pipeline.

## Configuration

You'll find a familiar transition from a Jenkins declarative pipeline into an Azure Pipelines YAML configuration. The two are conceptually similar, supporting "configuration as code" and allowing you to check your configuration into your version control system. Unlike Jenkins, however, Azure Pipelines uses the industry-standard [YAML to configure the build pipeline](#).

Despite the language difference, however, the concepts between Jenkins and Azure Pipelines and the way they're configured are similar. A Jenkinsfile lists one or more *stages* of the build process, each of which contains one or more *steps* that are performed in order. For example, a "build" stage may run a task to install build-time dependencies, then perform a compilation step. While a "test" stage may invoke the test harness against the binaries that were produced in the build stage.

For example:

[Jenkinsfile](#)

```
pipeline {
 agent none
 stages {
 stage('Build') {
 steps {
 sh 'npm install'
 sh 'npm run build'
 }
 }
 stage('Test') {
 steps {
 sh 'npm test'
 }
 }
 }
}
```

This translates easily to an Azure Pipelines YAML configuration, with a *job* corresponding to each stage, and steps to perform in each job:

#### azure-pipelines.yml

```
jobs:
- job: Build
 steps:
 - script: npm install
 - script: npm run build
- job: Test
 steps:
 - script: npm test
```

### Visual Configuration

If you are not using a Jenkins declarative pipeline with a `Jenkinsfile`, and are instead using the graphical interface to define your build configuration, then you may be more comfortable with the classic editor in Azure Pipelines.

## Container-Based Builds

Using containers in your build pipeline allows you to build and test within a docker image that has the exact dependencies that your pipeline needs, already configured. It saves you from having to include a build step that installs additional software or configures the environment. Both Jenkins and Azure Pipelines support container-based builds.

In addition, both Jenkins and Azure Pipelines allow you to share the build directory on the host agent to the container volume using the `-v` flag to docker. This allows you to chain multiple build jobs together that can use the same sources and write to the same output directory. This is especially useful when you use many different technologies in your stack; you may want to build your backend using a .NET Core container and your frontend with a TypeScript container.

For example, to run a build in an Ubuntu 14.04 ("Trusty") container, then run tests in an Ubuntu 16.04 ("Xenial") container:

#### Jenkinsfile

```

pipeline {
 agent none
 stages {
 stage('Build') {
 agent {
 docker {
 image 'ubuntu:trusty'
 args '-v $HOME:/build -w /build'
 }
 }
 steps {
 sh 'make'
 }
 }
 stage('Test') {
 agent {
 docker {
 image 'ubuntu:xenial'
 args '-v $HOME:/build -w /build'
 }
 }
 steps {
 sh 'make test'
 }
 }
 }
}

```

Azure Pipelines provides [container jobs](#) to enable you to run your build within a container:

#### `azure-pipelines.yml`

```

resources:
 containers:
 - container: trusty
 image: ubuntu:trusty
 - container: xenial
 image: ubuntu:xenial

jobs:
 - job: build
 container: trusty
 steps:
 - script: make
 - job: test
 dependsOn: build
 container: xenial
 steps:
 - script: make test

```

In addition, Azure Pipelines provides a [docker task](#) that allows you to run, build, or push an image.

## Agent Selection

Jenkins offers build agent selection using the `agent` option to ensure that your build pipeline - or a particular stage of the pipeline - runs on a particular build agent machine. Similarly, Azure Pipelines offers a number of options to configure where your build environment runs.

### Hosted Agent Selection

Azure Pipelines offers cloud hosted build agents for Linux, Windows, and macOS builds. To select the build environment, you can use the `vmimage` keyword. For example, to select a macOS build:

```
pool:
 vmImage: macOS-10.14
```

Additionally, you can specify a `container` and specify a docker image for finer grained control over how your build is run.

### On-Premises Agent Selection

If you host your build agents on-premises, then you can define the [build agent "capabilities"](#) based on the architecture of the machine or the software that you've installed on it. For example, if you've set up an on-premises build agent with the `java` capabilities, then you can ensure that your job runs on it using the `demands` keyword:

```
pool:
 demands: java
```

## Environment Variables

In Jenkins, you typically define environment variables for the entire pipeline. For example, to set two environment variables, `CONFIGURATION=debug` and `PLATFORM=x86`:

### Jenkinsfile

```
pipeline {
 environment {
 CONFIGURATION = 'debug'
 PLATFORM = 'x64'
 }
}
```

Similarly, in Azure Pipelines you can configure variables that are used both within the YAML configuration and are set as environment variables during job execution:

### azure-pipelines.yml

```
variables:
 configuration: debug
 platform: x64
```

Additionally, in Azure Pipelines you can define variables that are set only for the duration of a particular job:

### azure-pipelines.yml

```
jobs:
- job: debug build
 variables:
 configuration: debug
 steps:
 - script: ./build.sh $(configuration)
- job: release build
 variables:
 configuration: release
 steps:
 - script: ./build.sh $(configuration)
```

# Predefined Variables

Both Jenkins and Azure Pipelines set a [number of environment variables](#) to allow you to inspect and interact with the execution environment of the continuous integration system.

DESCRIPTION	JENKINS	AZURE PIPELINES
A unique numeric identifier for the current build invocation.	BUILD_NUMBER	BUILD_BUILDNUMBER
A unique identifier (not necessarily numeric) for the current build invocation.	BUILD_ID	BUILD_BUILDID
The URL that displays the build logs.	BUILD_URL	This is not set as an environment variable in Azure Pipelines but can be derived from other variables. <sup>1</sup>
The name of the machine that the current build is running on.	NODE_NAME	AGENT_NAME
The name of this project or build definition.	JOB_NAME	RELEASE_DEFINITIONNAME
A string for identification of the build; the build number is a good unique identifier.	BUILD_TAG	BUILD_BUILDNUMBER
A URL for the host executing the build.	JENKINS_URL	SYSTEM_TEAMFOUNDATIONCOLLECTIONURI
A unique identifier for the build executor or build agent that is currently running.	EXECUTOR_NUMBER	AGENT_NAME
The location of the checked out sources.	WORKSPACE	BUILD_SOURCESDIRECTORY
The Git Commit ID corresponding to the version of software being built.	GIT_COMMIT	BUILD_SOURCEVERSION
Path to the Git repository on GitHub, Azure Repos or another repository provider.	GIT_URL	BUILD_REPOSITORY_URI
The Git branch being built.	GIT_BRANCH	BUILD_SOURCEBRANCH

<sup>1</sup> To derive the URL that displays the build logs in Azure Pipelines, combine the following environment variables in this format:

```
 ${SYSTEM_TEAMFOUNDATIONCOLLECTIONURI}/#${SYSTEM_TEAMPROJECT}/_build/results?buildId=${BUILD_BUILDID}
```

## Success and Failure Handling

Jenkins allows you to run commands when the build has finished, using the `post` section of the pipeline. You can specify commands that run when the build succeeds (using the `success` section), when the build fails (using

the `failure` section) or always (using the `always` section). For example:

### Jenkinsfile

```
post {
 always {
 echo "The build has finished"
 }
 success {
 echo "The build succeeded"
 }
 failure {
 echo "The build failed"
 }
}
```

Similarly, Azure Pipelines has a rich conditional execution framework that allows you to run a job, or steps of a job, based on a number of conditions including pipeline success or failure.

To emulate the simple Jenkins `post`-build conditionals, you can define jobs that run based on the `always()`, `succeeded()` or `failed()` conditions:

### azure-pipelines.yml

```
jobs:
- job: always
 steps:
 - script: echo "The build has finished"
 condition: always()

- job: success
 steps:
 - script: echo "The build succeeded"
 condition: succeeded()

- job: failed
 steps:
 - script: echo "The build failed"
 condition: failed()
```

In addition, you can combine [other conditions](#), like the ability to run a task based on the success or failure of an individual task, environment variables, or the execution environment, to build a rich execution pipeline.

# Migrate from Travis to Azure Pipelines

3/6/2021 • 14 minutes to read • [Edit Online](#)

Azure Pipelines is more than just a Continuous Integration tool, it's a flexible build and release orchestration platform. It's designed for the software development and deployment process, but because of this extensibility, there are a number of differences from simpler build systems like Travis.

The purpose of this guide is to help you migrate from Travis to Azure Pipelines. This guide describes the philosophical differences between Travis and Azure Pipelines, examines the practical effects on the configuration of each system, and shows how to translate from a Travis configuration to an Azure Pipelines configuration.

We need your help to make this guide better! Submit comments below or contribute your changes directly.

## Key differences

There are numerous differences between Travis and Azure Pipelines, including version control configuration, environment variables, and virtual machine environments, but at a higher level:

- Azure Pipelines configuration is more precise and relies less on shorthand configuration and implied steps. You'll see this in places like language selection and in the way Azure Pipelines allows flow to be controlled.
- Travis builds have *stages*, *jobs* and *phases*, while Azure Pipelines simply has steps that can be arranged and executed in an arbitrary order or grouping that you choose. This gives you flexibility over the way that your steps are executed, including the way they're executed in parallel.
- Azure Pipelines allows job definitions and steps to be stored in separate YAML files in the same or a different repository, enabling steps to be shared across multiple pipelines.
- Azure Pipelines provides full support for building and testing on Microsoft-managed Linux, Windows, and macOS images. See [Microsoft-hosted agents](#) for more details.

## Before starting your migration

If you are new to Azure Pipelines, see the following to learn more about Azure Pipelines and how it works prior to starting your migration:

- [Create your first pipeline](#)
- [Key concepts for new Azure Pipelines users](#)
- [Building GitHub repositories](#)

## Language

Travis uses the `language` keyword to identify the prerequisite build environment to provision for your build. For example, to select NodeJS 8.x:

`.travis.yml`

```
language: node_js
node_js:
 - "8"
```

[Microsoft-hosted agents](#) contain the SDKs for many languages out-of-the-box and most languages need no configuration. But where a language has multiple versions installed, you may need to execute a [language selection task](#) to set up the environment.

For example, to select NodeJS 8.x:

`azure-pipelines.yml`

```

steps:
- task: NodeTool@0
 inputs:
 versionSpec: '8.x'

```

## Language mappings

The `language` keyword in Travis does not just imply that a particular version of language tools be used, but also that a number of build steps be implicitly performed. Azure Pipelines, on the other hand, does not do any work without your input, so you'll need to specify the commands that you want to execute.

Here is a translation guide from the `language` keyword to the commands that are executed automatically for the most commonly-used languages:

LANGUAGE	COMMANDS
c cpp	<code>./configure</code> <code>make</code> <code>make install</code>
csharp	<code>nuget restore [solution.sln]</code> <code>msbuild /p:Configuration=Release [solution.sln]</code>
clojure	<code>lein deps</code> <code>lein test</code>
go	<code>go get -t -v ./...</code> <code>make</code> or <code>go test</code>
java groovy	<b>Gradle:</b> <code>gradle assemble</code> <code>gradle check</code>
	<b>Maven:</b> <code>mvn install -DskipTests=true -Dmaven.javadoc.skip=true -B -V</code> <code>mvn test -B</code>
	<b>Ant:</b> <code>ant test</code>
node_js	<code>npm install</code> or <code>npm ci</code> or <code>yarn</code> <code>npm test</code>
objective-c swift	<code>pod install</code> or <code>bundle exec pod install</code> <code>xcodebuild -scheme [scheme] build test \  xcpretty</code>
perl	<code>cpanm --quiet --installdeps --notest .</code>
	<b>Build.PL:</b> <code>perl ./Build.pl</code> <code>./Build test</code>
	<b>Makefile.PL:</b> <code>perl Makefile.PL</code> <code>make test</code>
	<b>Makefile:</b> <code>make test</code>
php	<code>phpunit</code>
python	<code>pip install -r requirements.txt</code>
ruby	<code>bundle install --jobs=3 --retry=3</code> <code>rake</code>

In addition, less common languages can be enabled but require an additional dependency installation step or execution inside a docker container:

LANGUAGE	COMMANDS
crystal	<pre>docker run -v \$(pwd):/src -w /src crystallang/crystal shards install  docker run -v \$(pwd):/src -w /src crystallang/crystal crystal spec</pre>
d	<pre>sudo wget http://master.dl.sourceforge.net/project/d- apt/files/d-apt.list -O /etc/apt/sources.list.d/d- apt.list  sudo apt-get update sudo apt-get -y --allow-unauthenticated install -- reinstall d-apt-keyring  sudo apt-get update sudo apt-get install dmd-compiler dub dub test --compiler=dmd</pre>
dart	<pre>wget https://dl- ssl.google.com/linux/linux_signing_key.pub -o - \ sudo apt-key add - wget https://storage.googleapis.com/download.dartlang.org/linux/debian/dart_- -O /etc/apt/sources.list.d/dart_stable.list  sudo apt-get update sudo apt-get install dart /usr/lib/dart/bin/pub get /usr/lib/dart/bin/pub run test</pre>
erlang	<pre>sudo apt-get install rebar rebar get-deps rebar compile rebar skip_deps=true eunit</pre>
elixir	<pre>sudo apt-get install elixir mix local.rebar --force mix local.hex --force mix deps.get mix test</pre>
haskell	<pre>sudo apt-get install cabal-install cabal install --only-dependencies --enable-tests cabal configure --enable-tests cabal build cabal test</pre>
haxe	<pre>sudo apt-get install haxe yes \  haxelib install [hxml] haxe [hxml]</pre>
julia	<pre>sudo apt-get install julia julia -e "using Pkg; Pkg.build(); end" julia --check-bounds=yes -e "Pkg; Pkg.test(coverage=true); end"</pre>
nix	<pre>docker run -v \$(pwd):/src -w /src nixos/nix nix- build</pre>
perl6	<pre>sudo apt-get install rakudo PERL6LIB=lib prove -v -r --exec=perl6 t/</pre>
rust	<pre>curl -sSf https://sh.rustup.rs   sh -s -- -y cargo build --verbose cargo test --verbose</pre>

LANGUAGE	COMMANDS
scala	<pre>echo "deb https://dl.bintray.com/sbt/debian /"   sudo tee -a /etc/apt/sources.list.d/sbt.list sudo apt-key adv --keyserver hk://keyserver.ubuntu.com:80 --recv 2EE0EA64E40A89B84B2DF73499E82A75642AC823 sudo apt-get update sudo apt-get install sbt sbt ++2.11.6 test</pre>
smalltalk	<pre>docker run -v \$(pwd):/src -w /src hpiswa/smalltalkci smalltalkci</pre>

### Multiple language selection

You can also configure an environment that supports building different applications in multiple languages. For example, to ensure the build environment targets both NodeJS 8.x and Ruby 2.5 or better:

`azure-pipelines.yml`

```
steps:
- task: NodeTool@0
 inputs:
 versionSpec: '8.x'
- task: UseRubyVersion@0
 inputs:
 versionSpec: '>= 2.5'
```

## Phases

In Travis, steps are defined in a fixed set of named phases such as `before_install` or `before_script`. Azure Pipelines does not have named phases and steps can be grouped, named, and organized in whatever way makes sense for the pipeline.

For example:

`.travis.yml`

```
before_install:
 - npm install -g bower
install:
 - npm install
 - bower install
script:
 - npm run build
 - npm test
```

`azure-pipelines.yml`

```
steps:
- script: npm install -g bower
- script: npm install
- script: bower install
- script: npm run build
- script: npm test
```

Alternatively, steps can be grouped together and optionally named:

`azure-pipelines.yml`

```
steps:
- script: |
 npm install -g bower
 npm install
 bower install
 displayName: 'Install dependencies'
- script: npm run build
- script: npm test
```

## Parallel jobs

Travis provides parallelism by letting you define a stage, which is a group of jobs that are executed in parallel. A Travis build can have multiple stages; once all jobs in a stage have completed, the execution of the next stage can begin.

Azure Pipelines gives you finer grained control of parallelism. You can make each step dependent on any other step you want. In this way, you specify which steps run serially, and which can run in parallel. So you can fan out with multiple steps run in parallel after the completion of one step, and then fan back in with a single step that runs afterward. This model gives you options to define complex workflows if necessary. For now, here's a simple example:



For example, to run a build script, then upon its completion run both the unit tests and the integration tests in parallel, and once all tests have finished, package the artifacts and then run the deploy to pre-production:

.travis.yml

```
jobs:
 include:
 - stage: build
 script: ./build.sh
 - stage: test
 script: ./test.sh unit_tests
 - script: ./test.sh integration_tests
 - stage: package
 script: ./package.sh
 - stage: deploy
 script: ./deploy.sh pre_prod
```

azure-pipelines.yml

```
jobs:
- job: build
 steps:
 - script: ./build.sh
- job: test1
 dependsOn: build
 steps:
 - script: ./test.sh unit_tests
- job: test2
 dependsOn: build
 steps:
 - script: ./test.sh integration_tests
- job: package
 dependsOn:
 - test1
 - test2
 script: ./package.sh
- job: deploy
 dependsOn: package
 steps:
 - script: ./deploy.sh pre_prod
```

### Advanced parallel execution

In Azure Pipelines you have more options and control over how you orchestrate your pipeline. Unlike Travis, we don't require you to think of blocks that must be executed together. Instead, you can focus on the resources that a job needs to start and the resources that it produces when it's done.

For example, a team has a set of fast-running unit tests, and another set of slower integration tests. The team wants to begin creating the .ZIP file for a release as soon as the unit tests are completed because they provide high confidence that the build will provide a good package. But before they deploy to pre-production, they want to wait until all tests have passed:



In Azure Pipelines they can do it this way:

`azure-pipelines.yml`

```

jobs:
- job: build
 steps:
 - script: ./build.sh
- job: test1
 dependsOn: build
 steps:
 - script: ./test.sh unit_tests
- job: test2
 dependsOn: build
 steps:
 - script: ./test.sh integration_tests
- job: package
 dependsOn: test1
 script: ./package.sh
- job: deploy
 dependsOn:
 - test1
 - test2
 - package
 steps:
 - script: ./deploy.sh pre_prod

```

## Step reuse

Most teams like to reuse as much business logic as possible to save time and avoid replication errors, confusion, and staleness. Instead of duplicating your change, you can make the change in a common area, and your leverage increases when you have similar processes that build on multiple platforms.

In Travis you can use matrices to run multiple executions across a single configuration. In Azure Pipelines you can use matrices in the same way, but you can also implement configuration reuse by using YAML templates.

### **Example: Environment variable in a matrix**

One of the most common ways to run several builds with a slight variation is to change the execution using environment variables. For example, your build script can look for the presence of an environment variable and change the way your software is built, or the way its tested.

You can use a matrix to have run a build configuration several times, once for each value in the environment variable. For example, to run a given script three times, each time with a different setting for an environment variable:

#### .travis.yml

```

os: osx
env:
 matrix:
 - MY_ENVIRONMENT_VARIABLE: 'one'
 - MY_ENVIRONMENT_VARIABLE: 'two'
 - MY_ENVIRONMENT_VARIABLE: 'three'
 script: echo $MY_ENVIRONMENT_VARIABLE

```

#### azure-pipelines.yml

```

pool:
 vmImage: 'macOS-10.14'
strategy:
 matrix:
 set_env_to_one:
 MY_ENVIRONMENT_VARIABLE: 'one'
 set_env_to_two:
 MY_ENVIRONMENT_VARIABLE: 'two'
 set_env_to_three:
 MY_ENVIRONMENT_VARIABLE: 'three'
 steps:
 - script: echo $(MY_ENVIRONMENT_VARIABLE)

```

### **Example: Language versions in a matrix**

Another common scenario is to run against several different language environments. Travis supports an implicit definition using the `language` keyword, while Azure Pipelines expects an explicit task to define how to configure

that language version.

You can easily use the environment variable matrix options in Azure Pipelines to enable a matrix for different language versions. For example, you can set an environment variable in each matrix variable that corresponds to the language version that you want to use, then in the first step, use that environment variable to run the language configuration task:

#### .travis.yml

```
os: linux
matrix:
 include:
 - rvm: 2.3.7
 - rvm: 2.4.4
 - rvm: 2.5.1
 script: ruby --version
```

#### azure-pipelines.yml

```
vmImage: 'Ubuntu 16.04'
strategy:
 matrix:
 ruby 2.3:
 ruby_version: '2.3.7'
 ruby 2.4:
 ruby_version: '2.4.4'
 ruby 2.5:
 ruby_version: '2.5.1'
 steps:
 - task: UseRubyVersion@0
 inputs:
 versionSpec: $(ruby_version)
 - script: ruby --version
```

#### Example: Operating systems within a matrix

It's also common to run builds in multiple operating systems. Travis supports this definition using the `os` keyword, while Azure Pipelines lets you configure the operating system by selecting the pool to run in using the `vmImage` keyword.

For example, you can set an environment variable in each matrix variable that corresponds to the operating system image that you want to use. Then you can set the machine pool to the variable you've set:

#### .travis.yml

```
matrix:
 include:
 - os: linux
 - os: windows
 - os: osx
 script: echo Hello, world!
```

#### azure-pipelines.yml

```
strategy:
 matrix:
 linux:
 imageName: 'ubuntu-16.04'
 mac:
 imageName: 'macos-10.14'
 windows:
 imageName: 'vs2017-win2016'

 pool:
 vmImage: $(imageName)

 steps:
 - script: echo Hello, world!
```

## Success and failure handling

Travis allows you to specify steps that will be run when the build succeeds, using the `after_success` phase, or when the build fails, using the `after_failure` phase. Since Azure Pipelines doesn't limit you to a finite number of specially-named phases, you can define success and failure conditions based on the result of any step, which enables more flexible and powerful pipelines.

#### .travis.yml

```
build: ./build.sh
after_success: echo Success
after_failure: echo Failed
```

#### azure-pipelines.yml

```
steps:
- script: ./build.sh
- script: echo Success
 condition: succeeded()
- script: echo Failed
 condition: failed()
```

### Advanced success and failure handling

In Azure Pipelines you can program a flexible set of dependencies and conditions for flow control between jobs. You can configure jobs to run based on the success or failure of previous jobs or based on environment variables. You can even configure jobs to always run, regardless of the success of other jobs.

For example, if you want to run a script when the build fails, but only if it is running as a build on the main branch:

#### azure-pipelines.yml

```
jobs:
- job: build
 steps:
 - script: ./build.sh
- job: alert
 dependsOn: build
 condition: and(failed(), eq(variables['Build.SourceBranch'], 'refs/heads/main'))
 steps:
 - script: ./sound_the_alarms.sh
```

## Predefined variables

Both Travis and Azure Pipelines set a number of environment variables to allow you to inspect and interact with the execution environment of the CI system.

In most cases there's an Azure Pipelines variable to match the environment variable in Travis. Here's a list of commonly-used environment variables in Travis and their analog in Azure Pipelines:

TRAVIS	AZURE PIPELINES	DESCRIPTION
<code>CI=true</code> OR <code>TRAVIS=true</code>	<code>TF_BUILD=True</code>	Indicates that your build is running in the CI system; useful for scripts that are also intended to be run locally during development.
<code>TRAVIS_BRANCH</code>	<b>CI builds:</b> <code>BUILD_SOURCEBRANCH</code>  <b>Pull request builds:</b> <code>SYSTEM_PULLREQUEST_TARGETBRANCH</code>	The name of the branch the build was queued for, or the name of the branch the pull request is targeting.
<code>TRAVIS_BUILD_DIR</code>	<code>BUILD_SOURCESDIRECTORY</code>	The location of your checked out source and the default working directory.

TRAVIS	AZURE PIPELINES	DESCRIPTION
<code>TRAVIS_BUILD_NUMBER</code>	<code>BUILD_BUILDid</code>	A unique numeric identifier for the current build invocation.
<code>TRAVIS_COMMIT</code>	<b>CI builds:</b> <code>BUILD_SOURCEVERSION</code>	The commit ID currently being built.
<code>TRAVIS_COMMIT</code>	<b>Pull request builds:</b> <code>git rev-parse HEAD^2</code>	For pull request validation builds, Azure Pipelines sets <code>BUILD_SOURCEVERSION</code> to the resulting merge commit of the pull request into main; this command will identify the pull request commit itself.
<code>TRAVIS_COMMIT_MESSAGE</code>	<code>BUILD_SOURCEVERSIONMESSAGE</code>	The log message of the commit being built.
<code>TRAVIS_EVENT_TYPE</code>	<code>BUILD_REASON</code>	The reason the build was queued; a map of values is in the "build reasons" table below.
<code>TRAVIS_JOB_NAME</code>	<code>AGENT_JOBNAME</code>	The name of the current job, if specified.
<code>TRAVIS_OS_NAME</code>	<code>AGENT_OS</code>	The operating system that the job is running on; a map of values is in the "operating systems" table below.
<code>TRAVIS_PULL_REQUEST</code>	<b>Azure Repos:</b> <code>SYSTEM_PULLREQUEST_PULLREQUESTID</code>  <b>GitHub:</b> <code>SYSTEM_PULLREQUEST_PULLREQUESTNUMBER</code>	The pull request number that triggered this build. (For GitHub builds, this is a unique identifier that is <i>not</i> the pull request number.)
<code>TRAVIS_PULL_REQUEST_BRANCH</code>	<code>SYSTEM_PULLREQUEST_SOURCEBRANCH</code>	The name of the branch where the pull request originated.
<code>TRAVIS_PULL_REQUEST_SHA</code>	<b>Pull request builds:</b> <code>git rev-parse HEAD^2</code>	For pull request validation builds, Azure Pipelines sets <code>BUILD_SOURCEVERSION</code> to the resulting merge commit of the pull request into main; this command will identify the pull request commit itself.
<code>TRAVIS_PULL_REQUEST_SLUG</code>		The name of the forked repository, if the pull request originated in a fork. There's no analog to this in Azure Pipelines.
<code>TRAVIS_REPO_SLUG</code>	<code>BUILD_REPOSITORY_NAME</code>	The name of the repository that this build is configured for.
<code>TRAVIS_TEST_RESULT</code>	<code>AGENT_JOBSTATUS</code>	Travis sets this value to <code>0</code> if all previous steps have succeeded (returned <code>0</code> ). For Azure Pipelines, check that <code>AGENT_JOBSTATUS=Succeeded</code> .
<code>TRAVIS_TAG</code>	<code>BUILD_SOURCEBRANCH</code>	If this build was queued by the creation of a tag then this is the name of that tag. For Azure Pipelines, the <code>BUILD_SOURCEBRANCH</code> will be set to the full Git reference name, eg <code>refs/tags/tag_name</code> .

TRAVIS	AZURE PIPELINES	DESCRIPTION
<code>TRAVIS_BUILD_STAGE_NAME</code>		The name of the stage in Travis. As we saw earlier, Azure Pipelines handles flow control using jobs. You can reference <code>AGENT_JOBNAME</code> .

#### Build Reasons:

The `TRAVIS_EVENT_TYPE` variable contains values that map to values provided by the Azure Pipelines `BUILD_REASON` variable:

TRAVIS	AZURE PIPELINES	DESCRIPTION
<code>push</code>	<code>IndividualCI</code>	The build is a continuous integration build from a push.
<code>pull_request</code>	<code>PullRequest</code>	The build was queued to validate a pull request.
<code>api</code>	<code>Manual</code>	The build was queued by the REST API or a manual request on the web page.
<code>cron</code>	<code>Schedule</code>	The build was scheduled.

#### Operating Systems:

The `TRAVIS_OS_NAME` variable contains values that map to values provided by the Azure Pipelines `AGENT_OS` variable:

TRAVIS	AZURE PIPELINES	DESCRIPTION
<code>linux</code>	<code>Linux</code>	The build is running on Linux.
<code>osx</code>	<code>Darwin</code>	The build is running on macOS.
<code>windows</code>	<code>Windows_NT</code>	The build is running on Windows.

To learn more, see [Predefined environment variables](#).

If there isn't a variable for the data you need, then you can use a shell command to get it. For example, a good substitute of an environment variable containing the commit ID of the pull request being built is to run a git command: `git rev-parse HEAD^2`.

## Building specific branches

By default, both Travis and Azure Pipelines perform CI builds on all branches. Similarly, both systems allow you to limit these builds to specific branches. In Azure Pipelines, the list of branches to build should be listed in the `include` list and the branches *not* to build should be listed in the `exclude` list. Wildcards are supported.

For example, to build only the main branch and those that begin with the word "releases":

`.travis.yml`

```
branches:
 only:
 - main
 - /^releases.*/
```

`azure-pipelines.yml`

```
trigger:
 branches:
 include:
 - main
 - releases*
```

## Output caching

Travis supports caching dependencies and intermediate build output to improve build times. Azure Pipelines does not support caching intermediate build output, but does offer integration with [Azure Artifacts](#) for dependency storage.

## Git submodules

Travis and Azure Pipelines both clone git repos "recursively" by default. This means that submodules are cloned by the agent, which is useful since submodules usually contain dependencies. However, the extra cloning takes time, so if you don't need the dependencies then you can disable cloning submodules:

.travis.yml

```
git:
 submodules: false
```

azure-pipelines.yml

```
checkout:
 submodules: false
```

# Migrate from XAML builds to new builds

3/31/2021 • 13 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | XAML builds

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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.

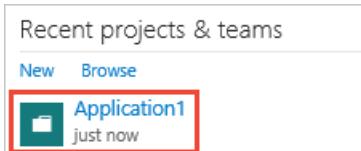
### WARNING

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

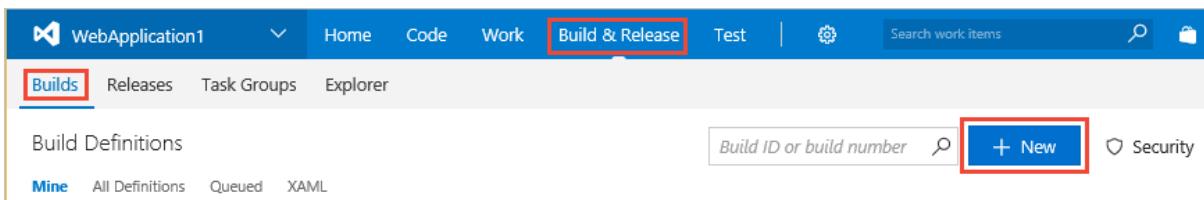


(If you don't see your project listed on the home page, select **Browse**.)

- 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)

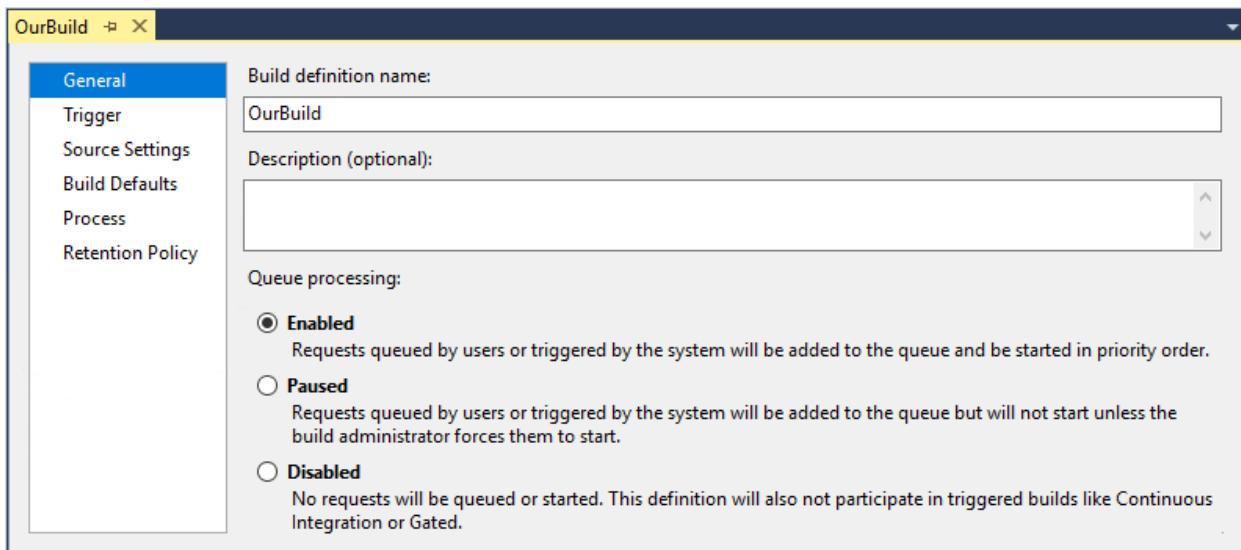


3. Select a template to add commonly used tasks to your build pipeline.
4. Make any necessary changes to your build pipeline to replicate your XAML build pipeline. The tasks added by the template should simply work in many cases. But if you changed process parameters or other settings in your XAML build 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



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 <b>Tasks</b> tab, in left pane click <b>Pipeline</b> , and the <b>Name</b> field appears in right pane.  In the <b>Builds</b> hub ( <b>Mine</b> or <b>All pipelines</b> tab), open the action menu and choose <b>Rename</b> .
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

The screenshot shows the 'Source Settings' tab for a TFVC template named 'TfvcTemplate.12.xaml'. It displays working folder mappings for 'Status' (Active and Cloaked) and 'Build Agent Folder' (\$SourceDir).

XAML SETTING	TFS 2017 AND NEWER EQUIVALENT	AZURE PIPELINES EQUIVALENT
Source Settings tab	On the <b>Repository</b> tab specify your mappings with Active paths as <b>Map</b> and Cloaked paths as <b>Cloak</b> .	On the <b>Tasks</b> tab, in left pane click <b>Get sources</b> . Specify your workspace mappings with Active paths as <b>Map</b> and Cloaked paths as <b>Cloak</b> .

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

<a href="#">General</a> <a href="#">Trigger</a> <b>Source Settings</b> <a href="#">Build Defaults</a> <a href="#">Process</a> <a href="#">Retention Policy</a>	<p>Choose the source of the build. You can choose a Team Foundation service.</p> <p><input checked="" type="checkbox"/> Get sources from a Team Foundation Git repository</p> <p>Repository name: TestGit</p> <p>Default branch for Manual and Scheduled builds: refs/heads/master</p> <p>Monitored branches for Continuous Integration and Rolling builds:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <th style="background-color: #f2f2f2;">Include/Exclude</th> <th style="background-color: #f2f2f2;">Branch</th> </tr> <tr> <td style="background-color: #0078d4; color: white;">Include</td> <td>refs/heads/master</td> </tr> </table> <p><a href="#">Click here to add a new row</a></p>	Include/Exclude	Branch	Include	refs/heads/master
Include/Exclude	Branch				
Include	refs/heads/master				

XAML SETTING	TFS 2017 AND NEWER EQUIVALENT	AZURE PIPELINES EQUIVALENT
Source Settings tab	On the <b>Repository</b> tab specify the repository and default branch.	On the <b>Tasks</b> tab, in left pane click <b>Get sources</b> . 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

<a href="#">General</a> <b>Trigger</b> <a href="#">Source Settings</a> <a href="#">Build Defaults</a> <a href="#">Process</a> <a href="#">Retention Policy</a>	<p>Select one of the following triggers:</p> <p><input type="radio"/> Manual - Check-ins do not trigger a new build</p> <p><input checked="" type="radio"/> Continuous Integration - Build each check-in</p> <p><input type="radio"/> Rolling builds - accumulate check-ins until the prior build finishes</p> <p style="margin-left: 20px;"><input type="checkbox"/> Build no more often than <u>every</u> <input type="text"/> minutes.</p> <p><input type="radio"/> Gated Check-in - accept check-ins only if the submitted changes merge and build successfully</p> <p style="margin-left: 20px;"><input type="checkbox"/> Merge and build up to <input type="text"/> submissions.</p> <p><input type="radio"/> Schedule - build every <u>week</u> on the following days</p> <p style="margin-left: 20px;"><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</p> <p style="margin-left: 20px;"><input type="checkbox"/> Saturday    <input type="checkbox"/> Sunday</p> <p>Queue the build on the build controller at:</p> <p style="margin-left: 20px;"><input type="text" value="03:00"/> Eastern Daylight Time (UTC -04:00)</p> <p><input type="checkbox"/> Build even if nothing has changed since the previous build</p>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

XAML SETTING	TFS 2017 AND NEWER, AZURE PIPELINES EQUIVALENT
Trigger tab	On the <b>Triggers</b> 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

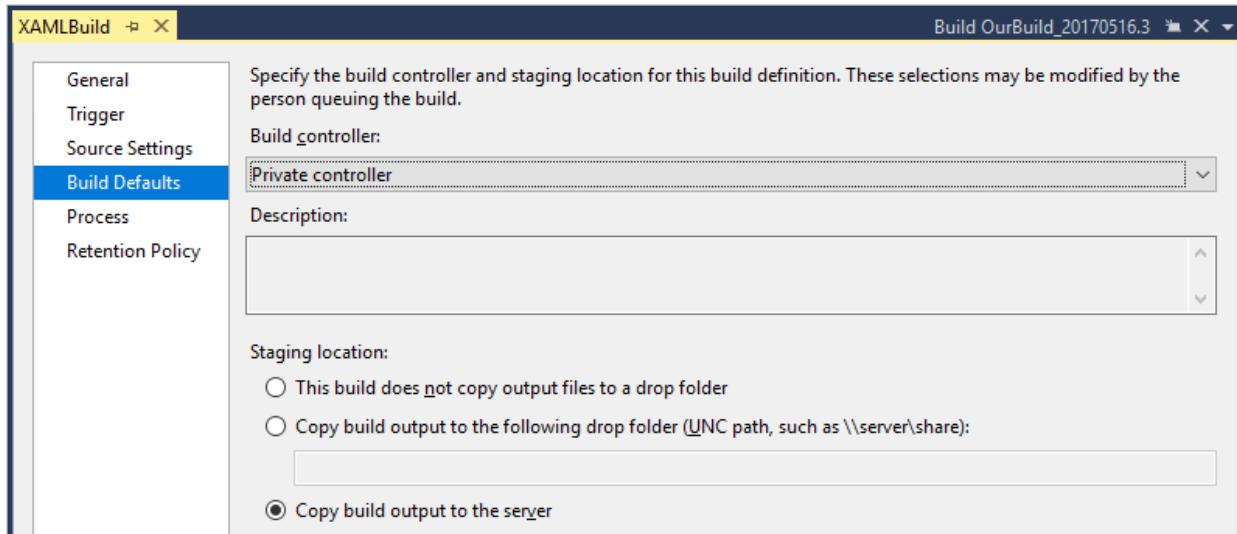
per branch.

- 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 <b>General</b> tab, select the default agent pool.	On the <b>Options</b> tab, select the default agent pool.
Staging location	On the <b>Tasks</b> tab, specify arguments to the Copy Files and Publish Build Artifacts tasks. See <a href="#">Build artifacts</a> .	On the <b>Tasks</b> tab, specify arguments to the Copy Files and Publish Build Artifacts tasks. See <a href="#">Build artifacts</a> .

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 <b>Repository</b> tab, open the <b>Clean</b> menu, and then select <b>true</b> .	On the <b>Tasks</b> tab, in left pane click <b>Get sources</b> . Display <b>Advanced settings</b> , and then select <b>Clean</b> . (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 <b>Repository</b> tab, select an option from the <b>Label sources</b> menu.	<b>Tasks</b> tab, in left pane click <b>Get sources</b> . Select one of the <b>Tag sources</b> options. (We plan to change the name of this to <b>Label sources</b> .)

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	<b>Repository</b> tab, open <b>Clean</b> menu, select <b>true</b> .	On the <b>Tasks</b> tab, in left pane click <b>Get sources</b> . Show <b>Advanced settings</b> , and then select <b>Clean</b> . (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 <a href="#">Visual Studio Build: How do I build multiple configurations for multiple platforms?</a>
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 <a href="#">Artifacts in Azure Pipelines</a> .
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 <a href="#">Use a PowerShell script to customize your build pipeline</a> .

#### 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	<b>1 set(s) of tests specified.</b>
1. Test source	- Run tests in test sources matching **\*test*.dll;**\*test*.appx, Target platform X86
Fail build on test failure	False
Run settings	<b>Run settings</b>
Run settings file	
Type of run settings	<b>Default</b>
Target platform for test execution	X86
Test case filter	
Test run name	
Test sources spec	<b>**\*test*.dll;**\*test*.appx</b>
2. Advanced	
Analyze test impact	<b>True</b>
Disable tests	False
Post-test script arguments	
Post-test script path	
Pre-test script arguments	
Pre-test script path	
> 4. Publish Symbols	
> 5. Advanced	

See [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 <b>Path to publish symbols</b> 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)
Maximum agent execution time	00:00:00
Maximum agent reservation wait time	<b>04:00:00</b>
Name filter	*
Tag comparison operator	<b>MatchExactly</b>
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 <b>Options</b> tab you can specify <b>Build job timeout in minutes</b> .
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 <a href="#">Agent capabilities</a> .	A build pipeline asserts demands that are matched with agent capabilities. See <a href="#">Agent capabilities</a> .
Build number format	On the <b>General</b> tab, copy your build number format into the <b>Build number format</b> field.	On the <b>General</b> tab, copy your build number format into the <b>Build number format</b> field.
Create work item on failure	On the <b>Options</b> tab, select this check box.	On the <b>Options</b> tab, enable this option.
Update work items with build number	None	On the <b>Options</b> tab you can enable <b>Automatically link new work in this build</b> .

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
<b>Triggered and Manual</b>		
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
<b>Private</b>		
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 <b>Retention</b> 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:

- [Create your first pipeline](#)
- [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.

Add build step...	<b>PowerShell Script</b>
Build solution \$/TestTFVC/C Visual Studio Build	Type Inline Script
PowerShell Script PowerShell	Arguments
	Inline Script 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 [Create your first pipeline](#).

### 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.

## FAQ

### 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

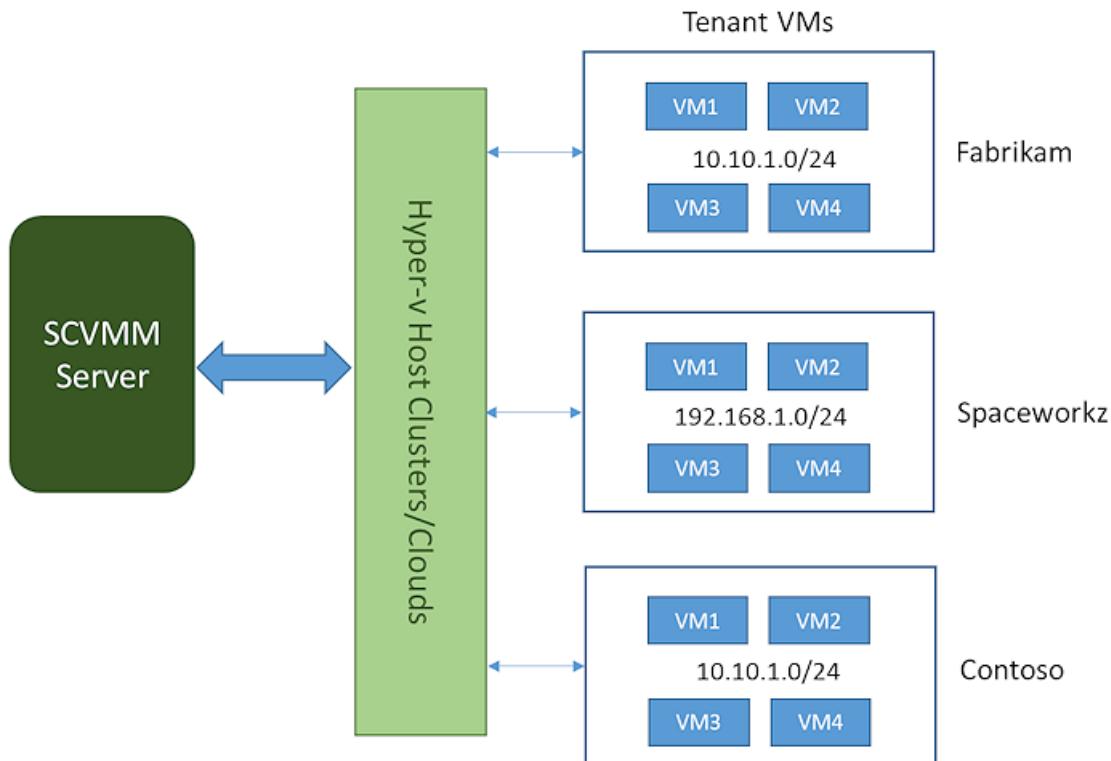
6/2/2020 • 10 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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.
- Windows 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). Your 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:

1. Open an RDP session to each of the host machines and open an administrator PowerShell session.
2. 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>
```

3. 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> -
```

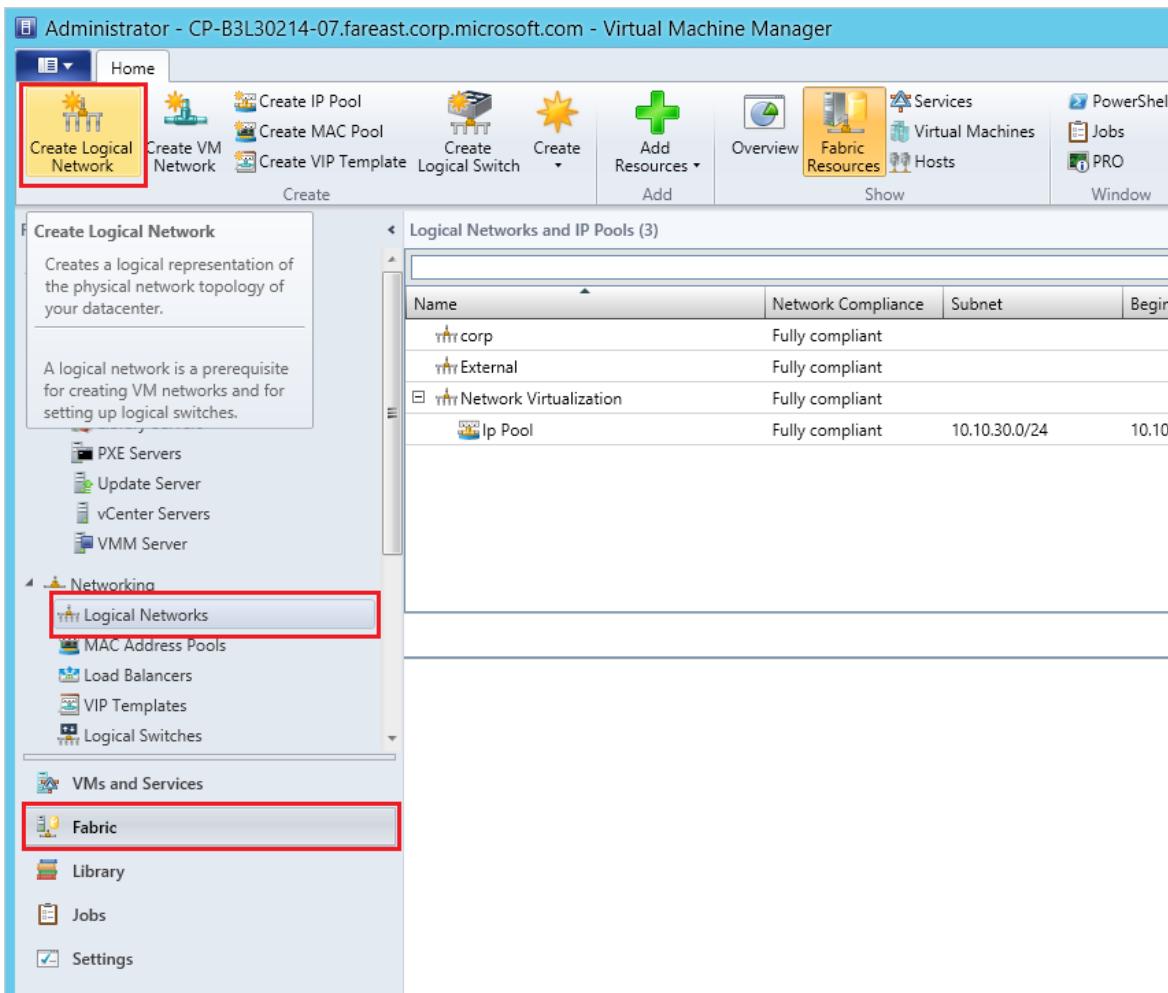
[Back to list of tasks](#)

## Create a Network Virtualization layer in SCVMM

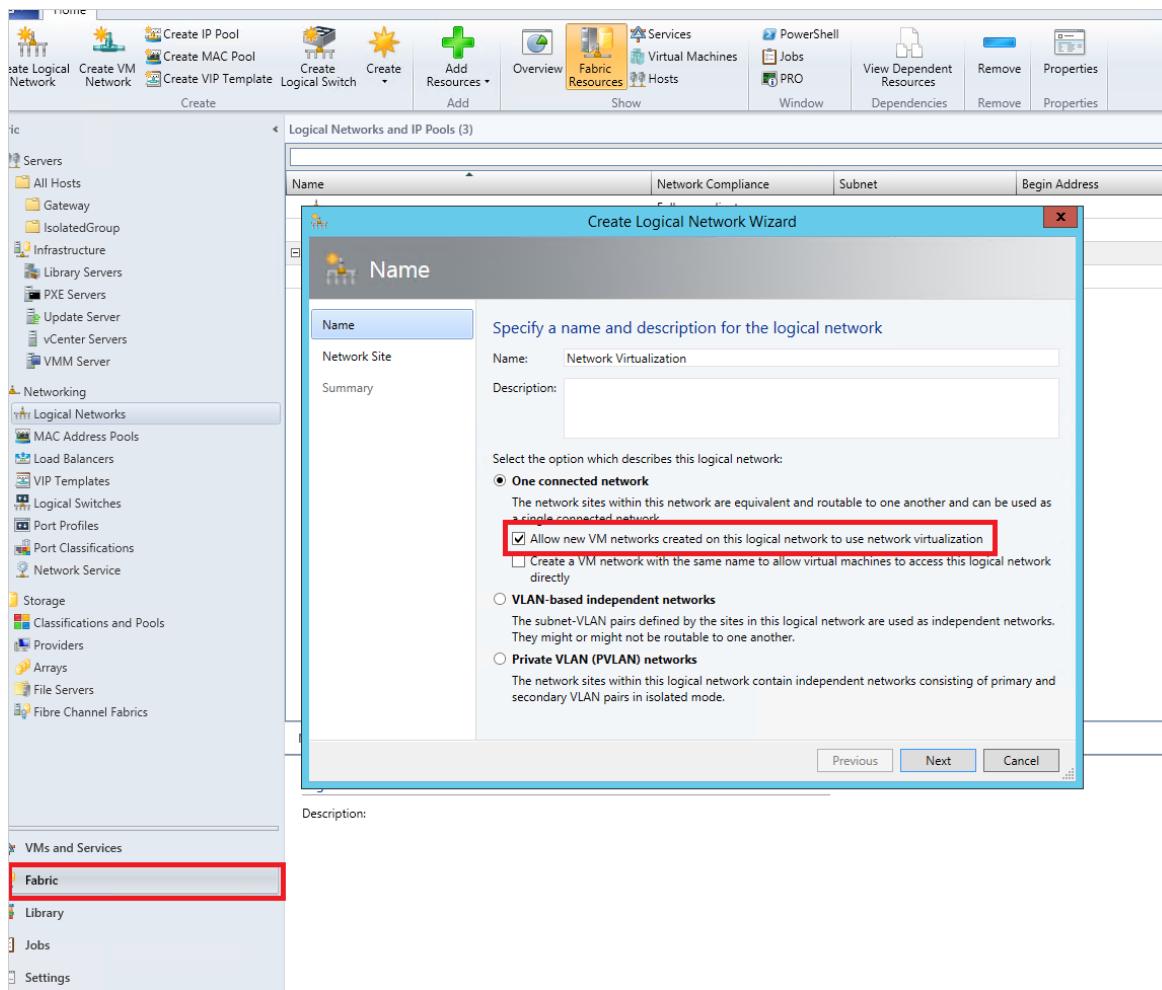
Setting up a network visualization layer in SCVMM includes creating logical networks, port profiles, logical switches, and adding the switches to the Hyper-V hosts.

### Create logical networks

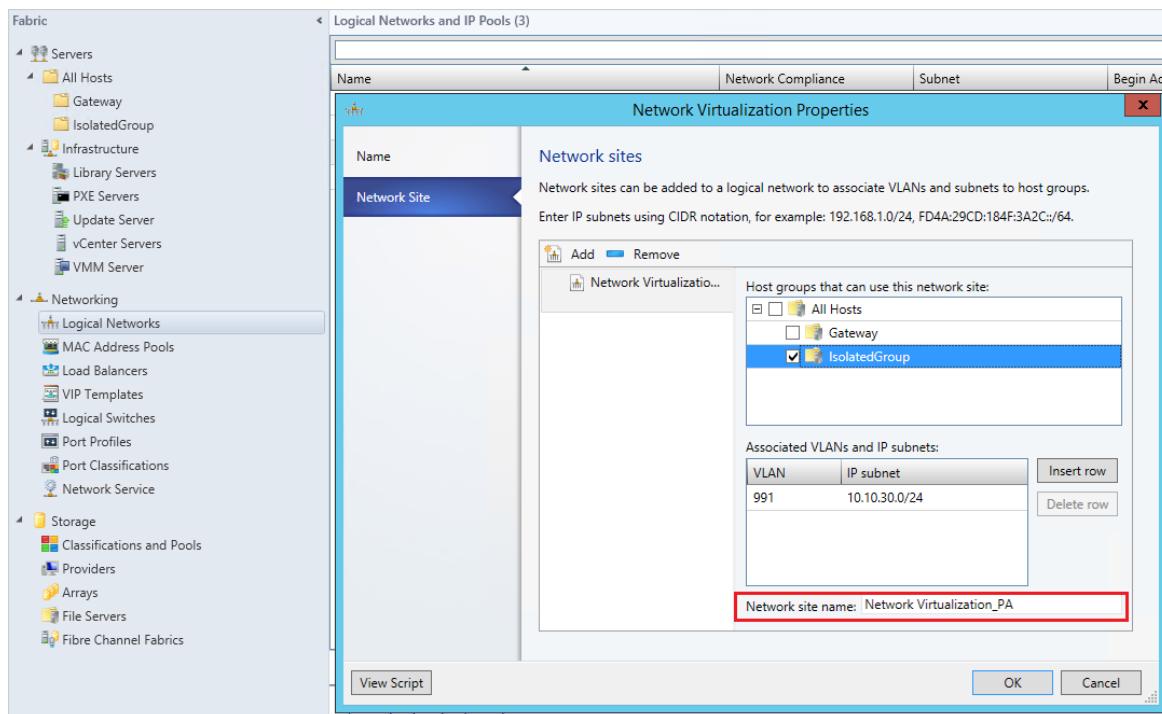
1. Log into the SCVMM admin console.
2. Go to Fabric -> Networking -> Logical Networks -> Create new Logical Network.



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.

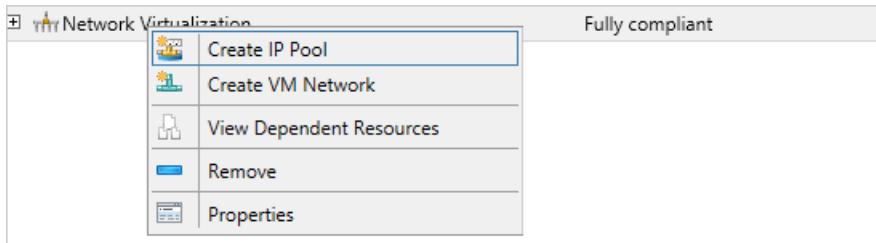


4. Add a new **Network Site** and select the host group to which the network site will be scoped. Enter the VLAN ID used to configure physical NIC in the Hyper-V host group and the corresponding routable IP subnet(s). To assist tracking, change the network site name to one that is memorable.

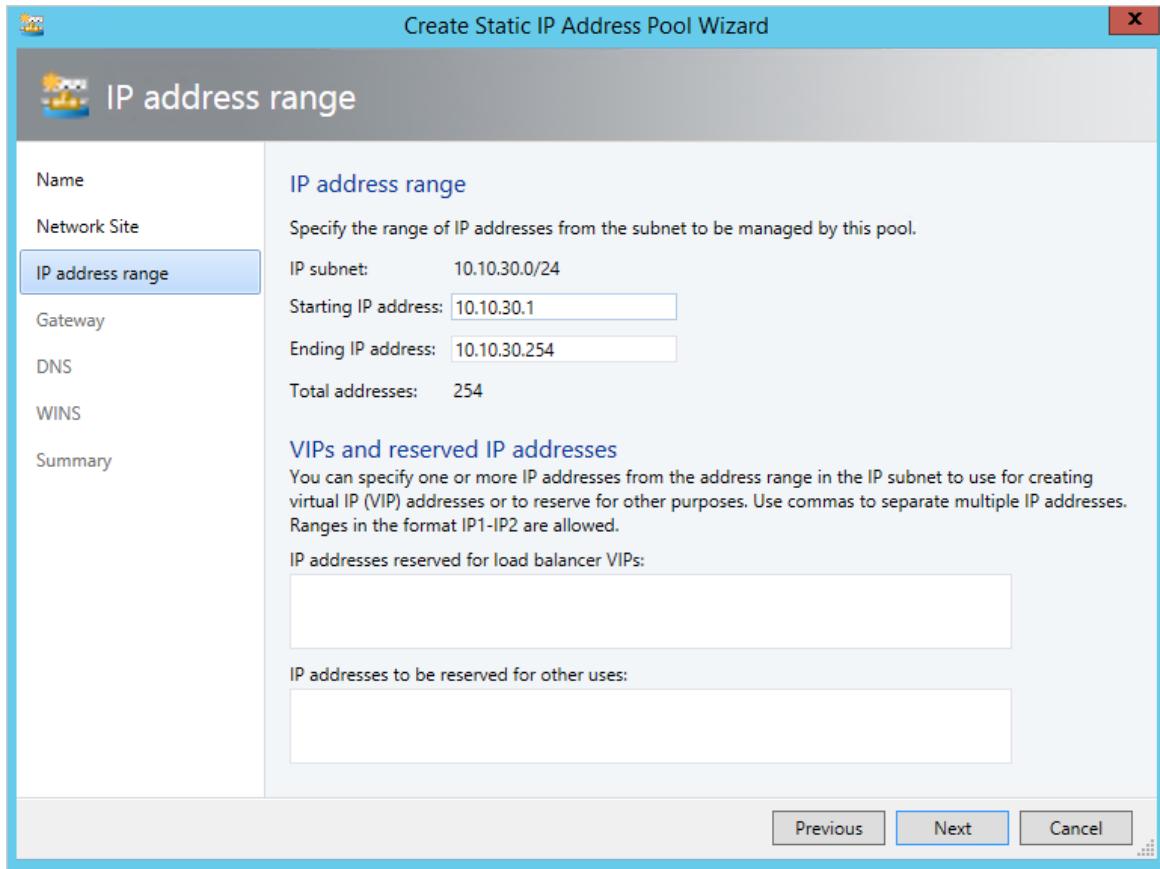


5. Click **Next** and **Save**.

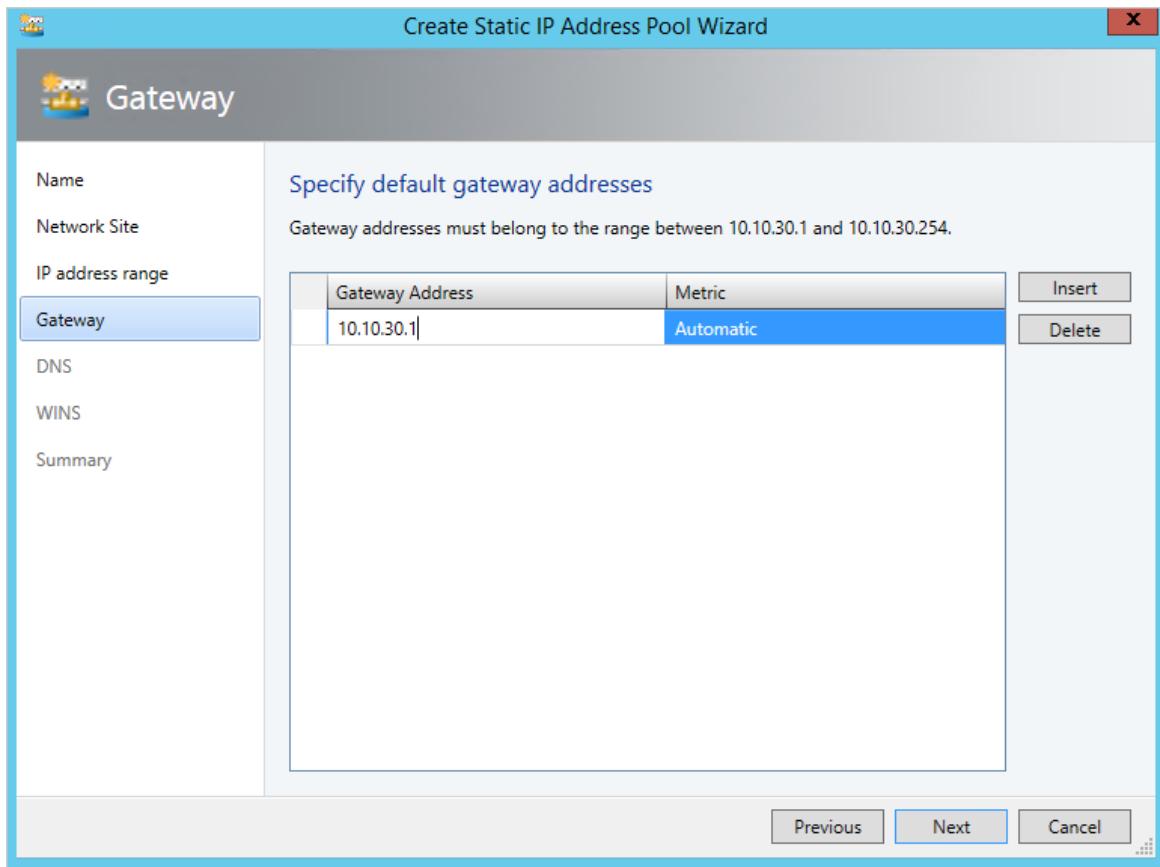
6. Create an IP pool for the new logical networks, enter a name, and click **Next**.



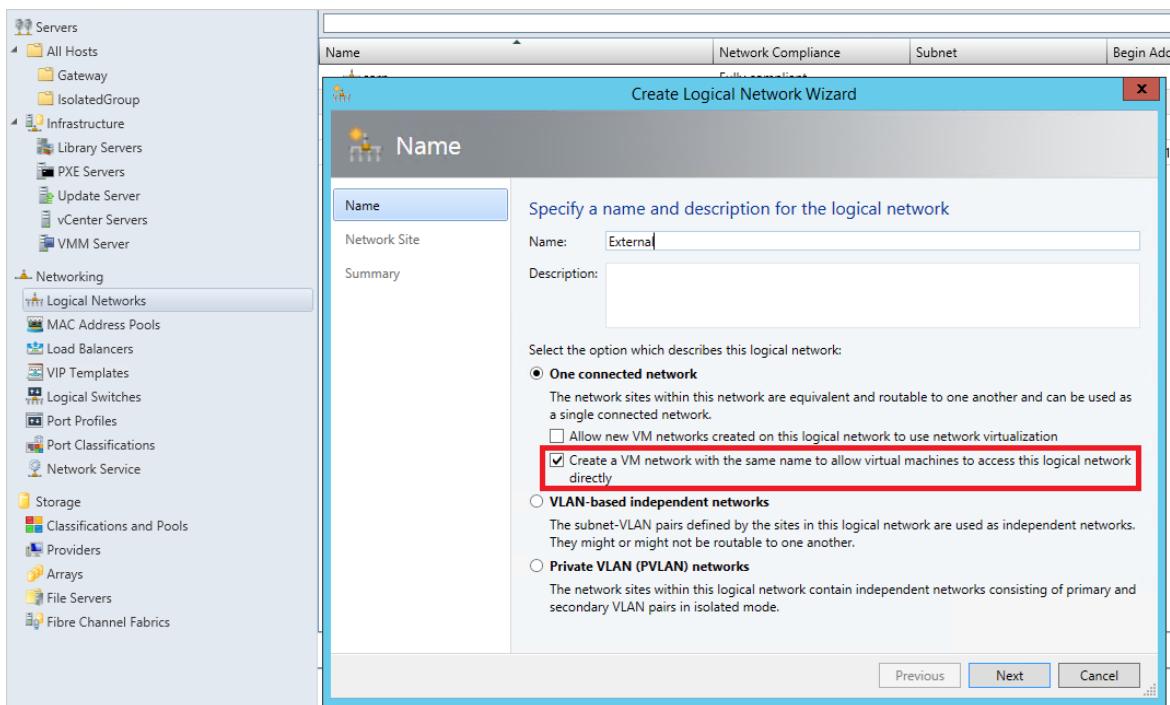
7. Select **Use and existing network site** and click **Next**. Enter the routable IP address range your network administrator configured for your VLAN and click **Next**. If you have multiple routable IP subnets associated with your VLAN, create an IP pool for each one.



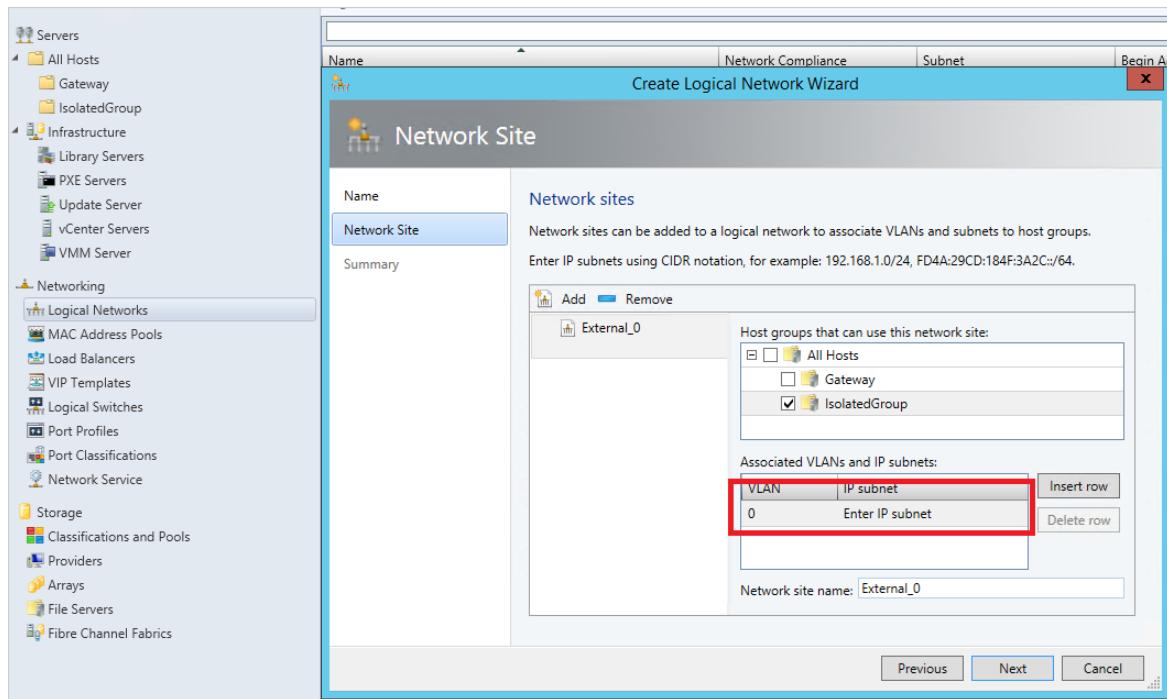
8. Provide the gateway address. By default, you can use the first IP address in your subnet.



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.

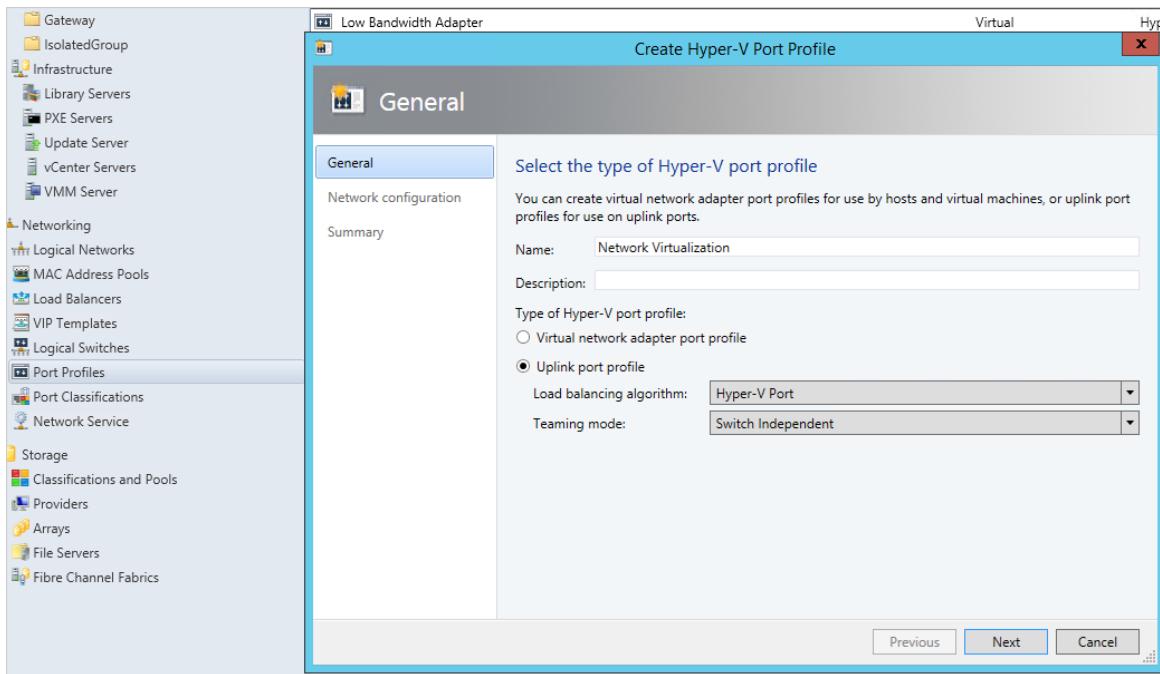
External	Fully compliant
Network Virtualization	Fully compliant
Ip Pool	Fully compliant 10.10.30.0/24    10.10.30.1    10.10.30.254    247    247

### Create port profiles

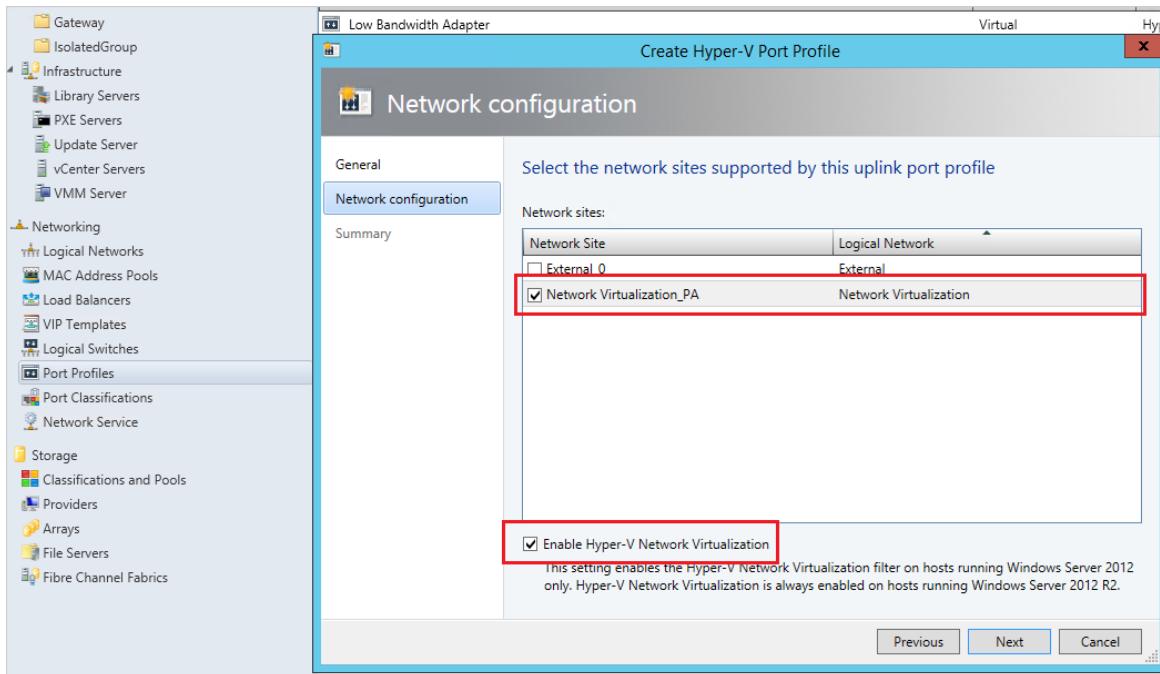
1. Go to Fabric -> Networking -> Port profiles and Create Hyper-V port profile.

Name:	Network Virtualization
Type:	Uplink
Applies to:	Hyper-V

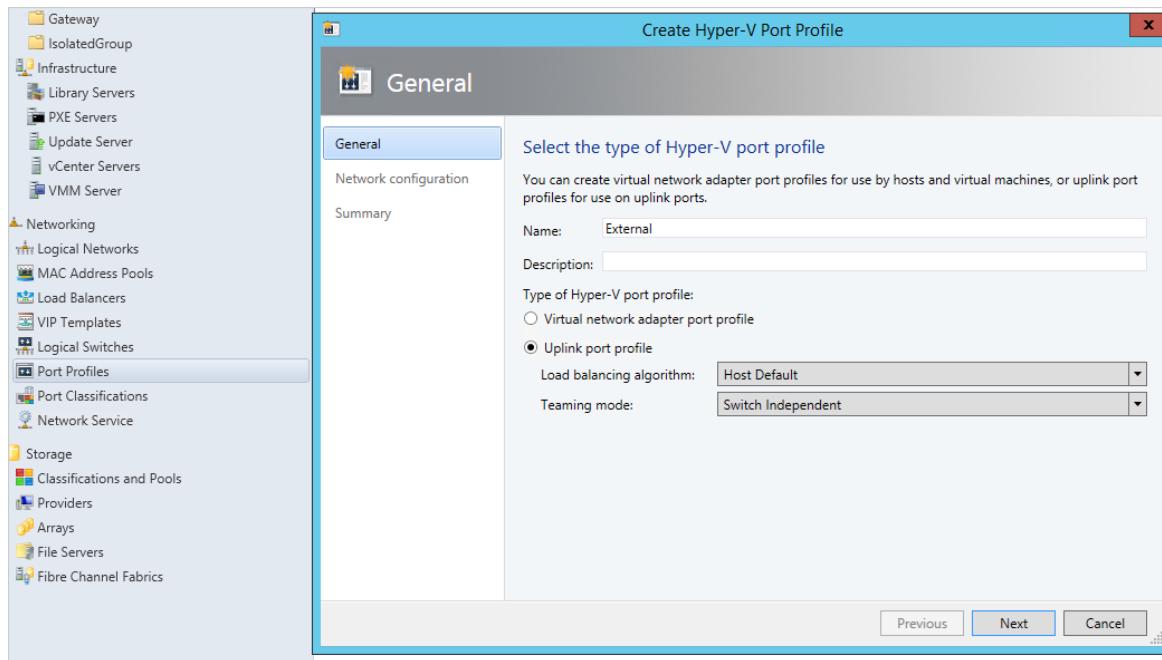
2. Select Uplink port profile and select Hyper-V Port as the load balancing algorithm, then click Next.



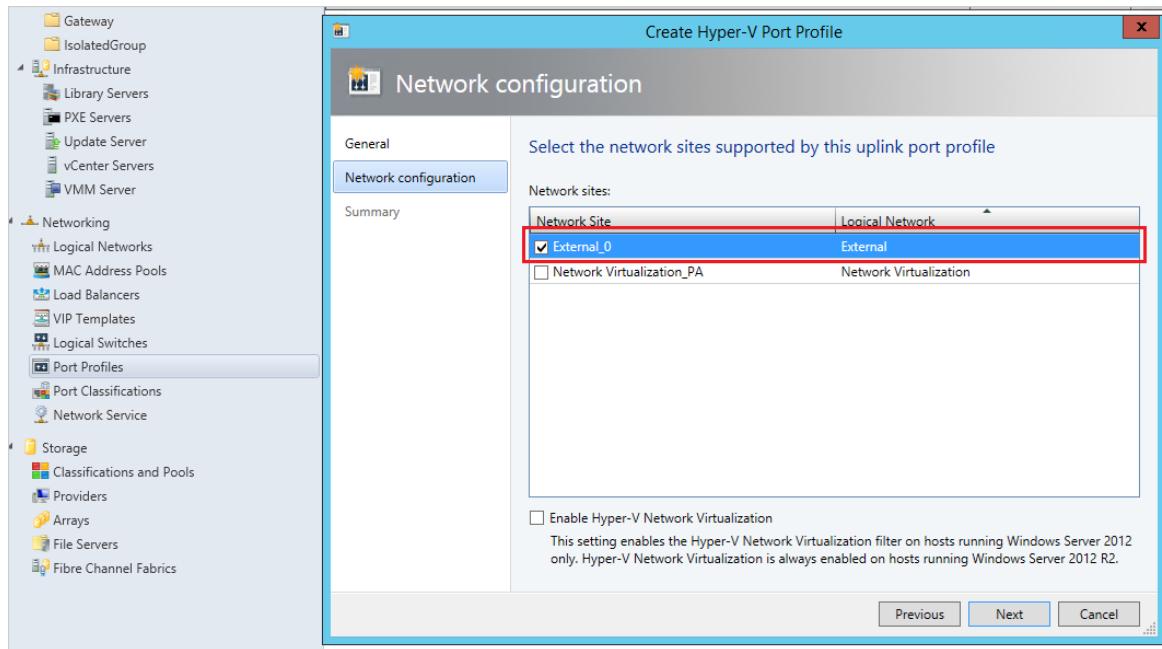
3. Select the Network Virtualization site created previously and choose the **Enable Hyper-V Network Virtualization** checkbox, then save the profile.



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**.

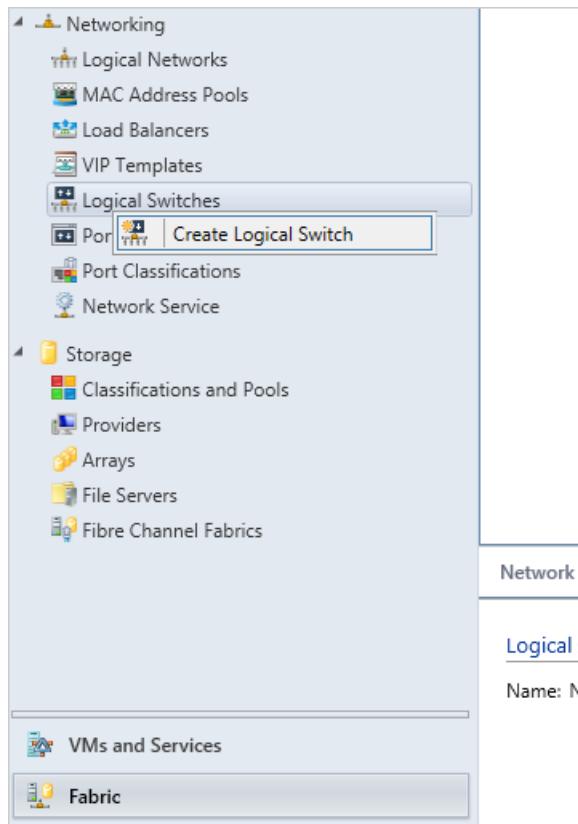


5. Select the other network site to be used for external communication, but this time don't enable network virtualization. Then save the profile.

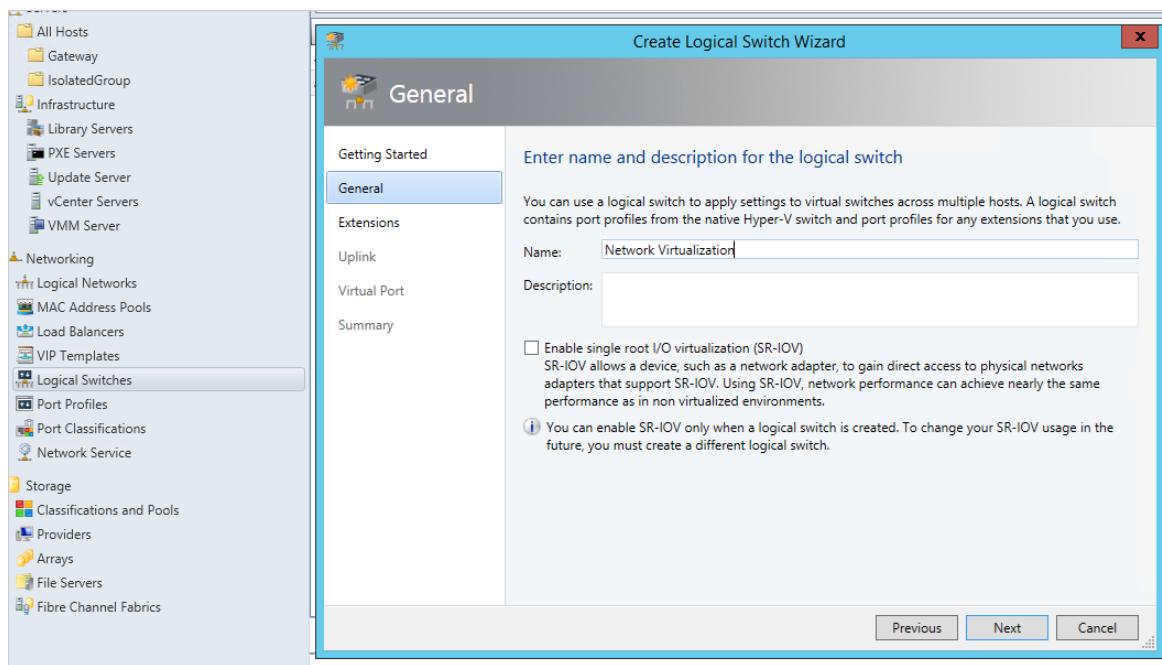


## Create logical switches

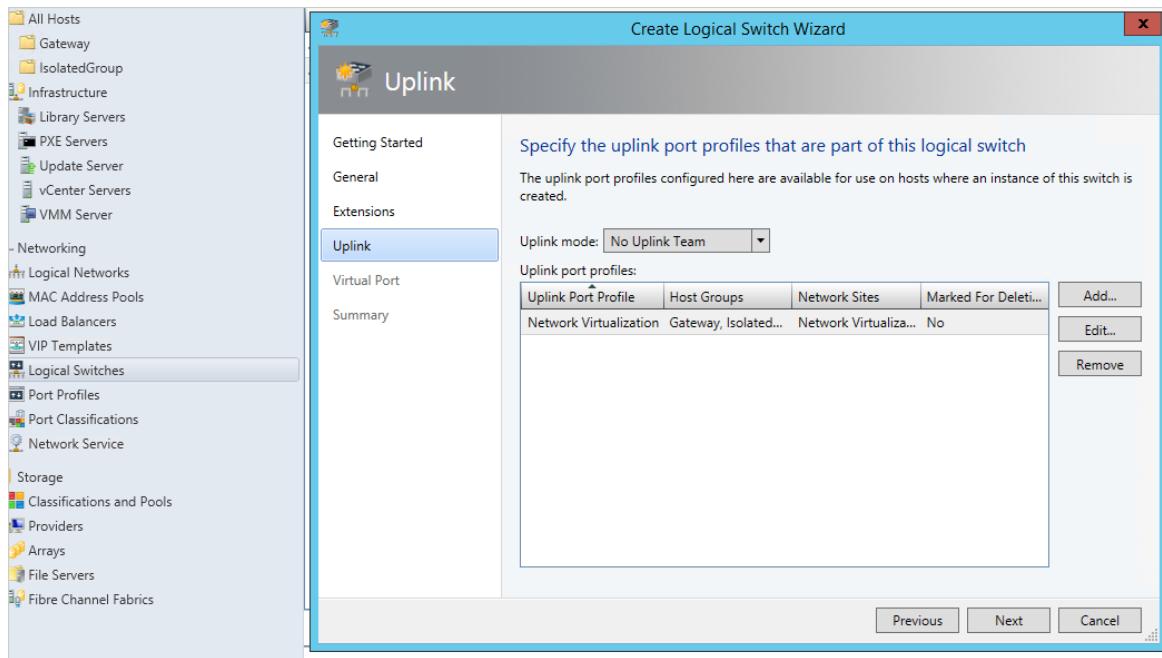
1. Go to Fabric -> Networking -> Logical switches and Create Logical Switch.



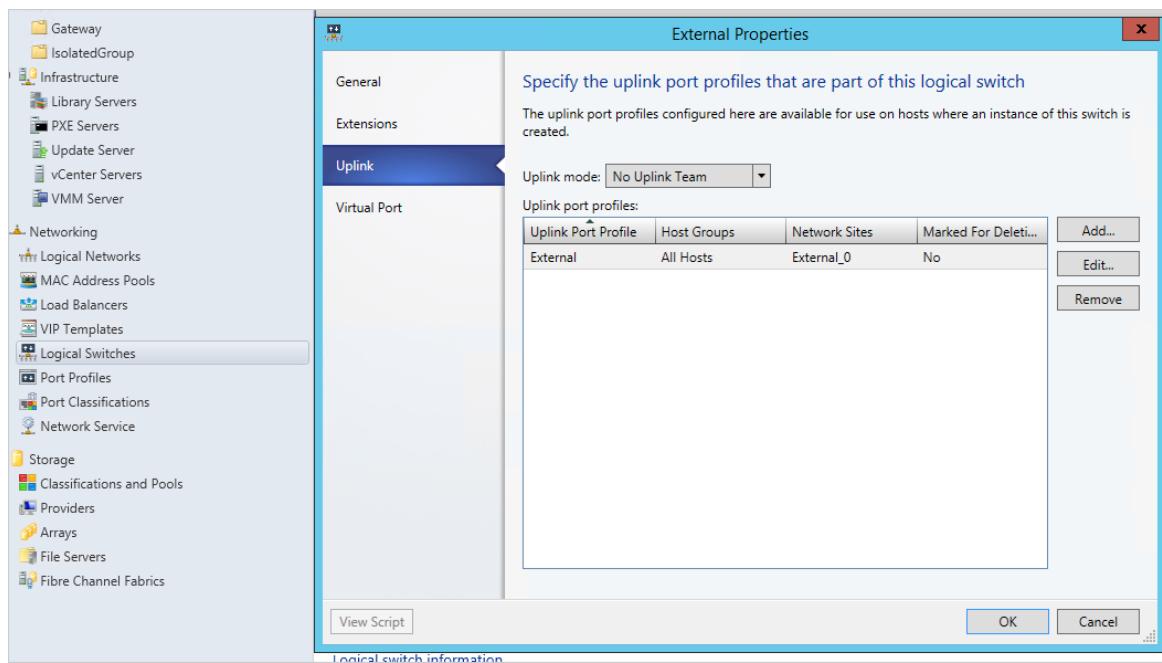
2. In the getting started wizard, click **Next** and enter a name for the switch, then click **Next**.



3. Click **Next** to open to **Uplink** tab. Click **Add uplink port profile** and add the network virtualization port profile you just created.

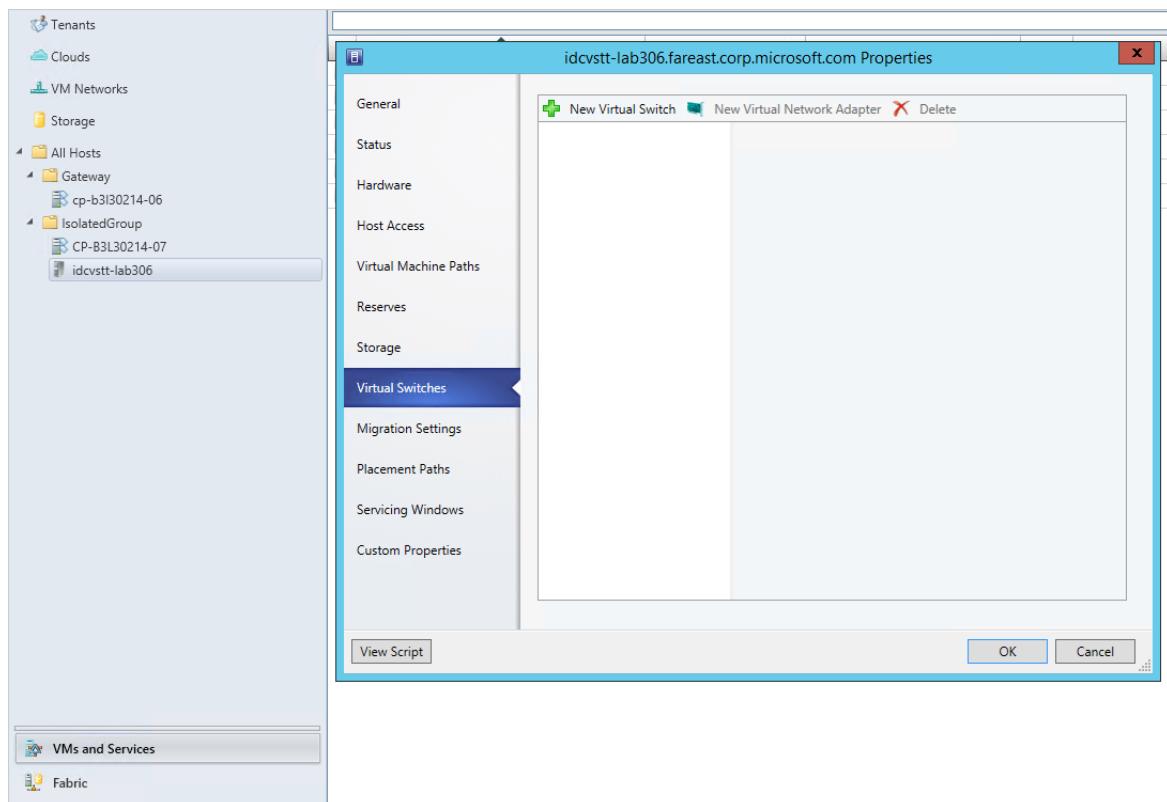


4. Click **Next** and save the logical switch.
5. Now create another logical switch for the external network for Internet communication. This time add the other uplink port profile you created for the external network.

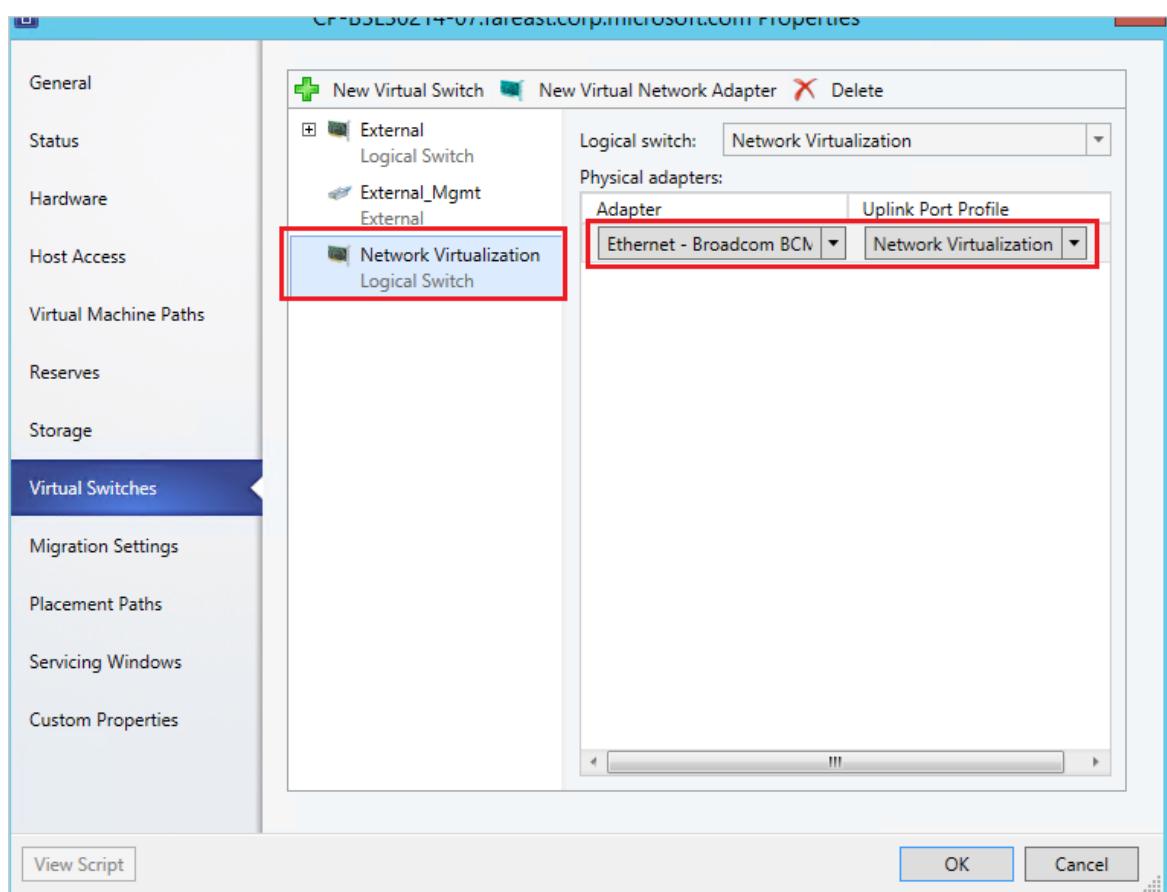


### Add logical switches to Hyper-V hosts

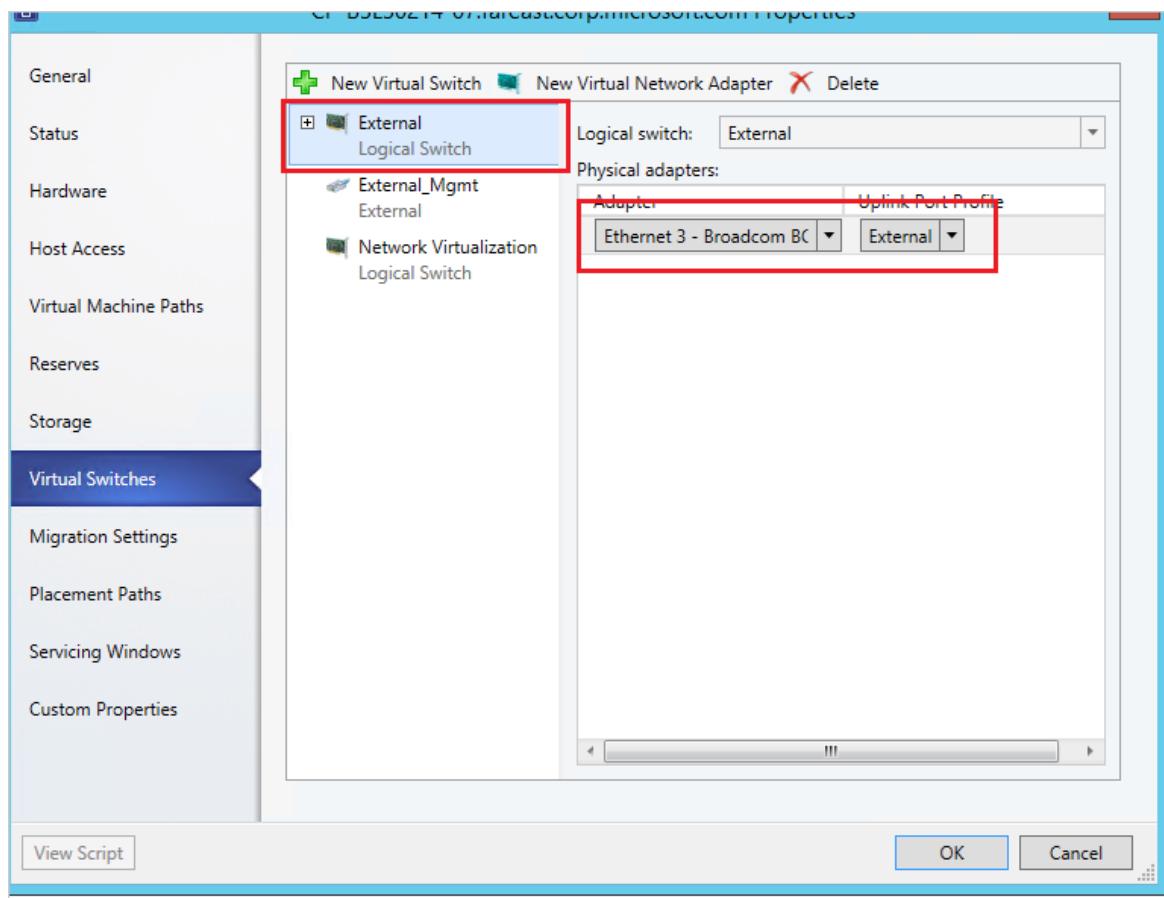
1. Go to **VM and Services** -> [Your host group] -> [each of the host machines in turn].
2. Right click and open the **Properties** -> **Virtual Switches** tab.



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.



- 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.

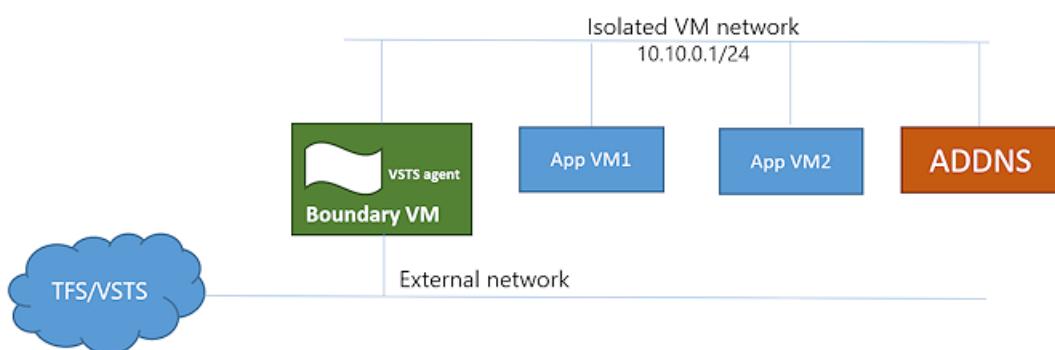
[Back to list of tasks](#)

## Create the required virtual network topology

Isolated virtual networks can be broadly classified into three topologies.

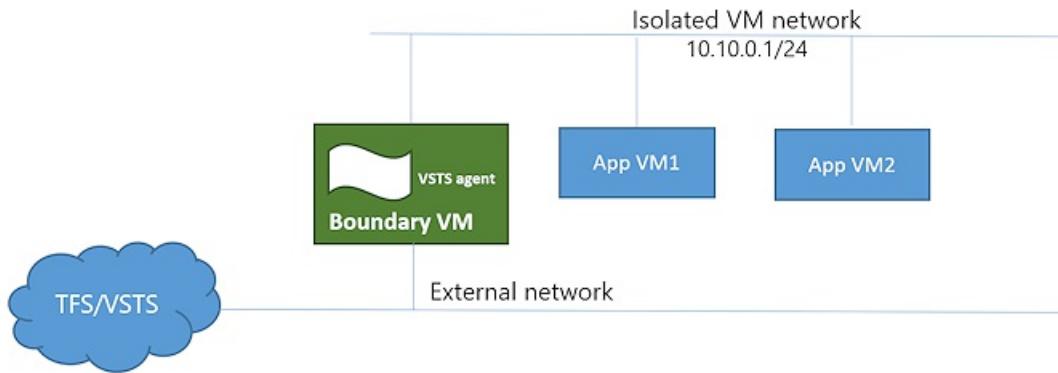
### Topology 1: AD-backed Isolated VMs

- 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.



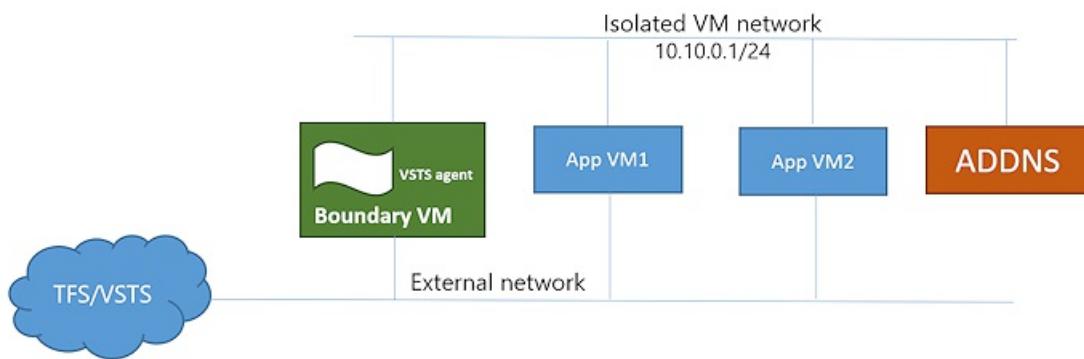
## Topology 2: Non-AD backed isolated VMs

- 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.

All definitions > BDT\_Scenario

Pipeline Tasks Variables Retention Options History

Environment 1 Deployment process

Agent phase Run on agent

SCVMM :NewVM action SCVMM

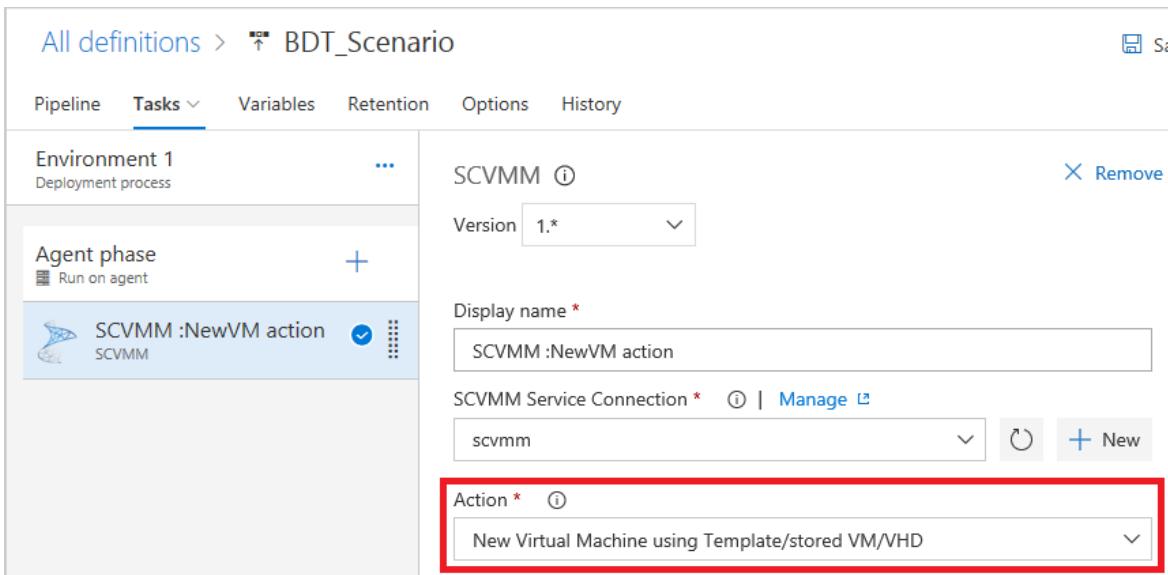
SCVMM (Remove)

Version 1.\*

Display name \* SCVMM :NewVM action

SCVMM Service Connection \* Manage scvmm

Action \* 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.

All definitions > BDT\_Scenario

Pipeline Tasks Variables Retention Options History

Environment 1 Deployment process

Agent phase Run on agent

SCVMM :NewVM action SCVMM

New Virtual Machine using Template/stored VM/VHD

VM Template options ^

Create virtual machines from VM Templates

Virtual machine names \* \$(AppVM1)

VM template names \* \$(AppVM\_Template)

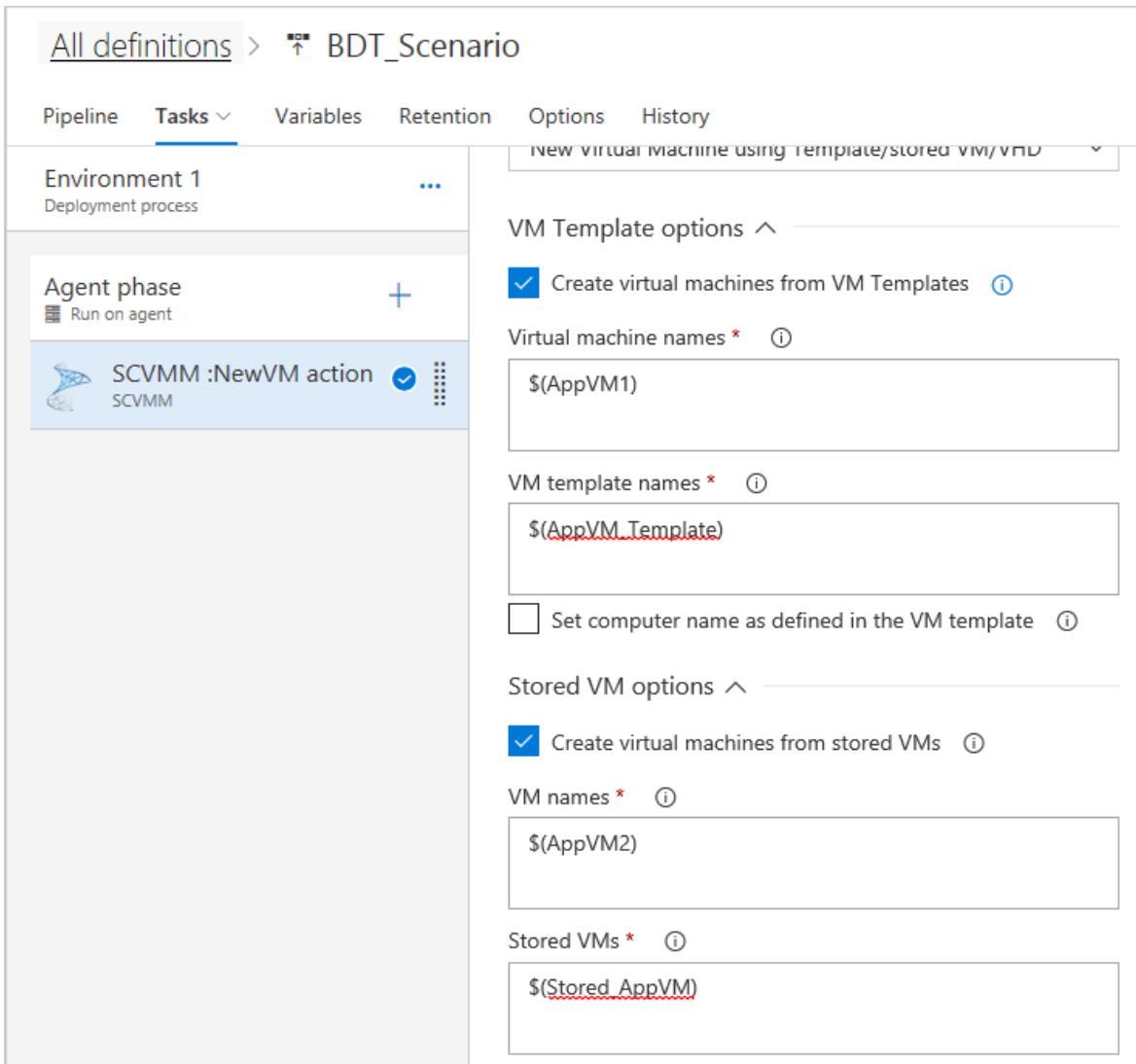
Set computer name as defined in the VM template

Stored VM options ^

Create virtual machines from stored VMs

VM names \* \$(AppVM2)

Stored VMs \* \$(Stored\_AppVM)



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.

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.

10. 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.
11. 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

12. 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.

## All definitions > BDT\_Scenario

Pipeline Tasks Variables Retention Options History

Environment 1 Deployment process

Agent phase Run on agent

SCVMM :NewVM action SCVMM

Subnet name for external VM network

Boundary VM agent type

Deployment Group agent  Automation Pool agent

Pool name \* \$(AutomationPool)

Personal Access Token \* \$(MyPAT)

Agent name \* \$(MyAgent)\_\$(Release.ReleaseName)

Agent installation path \* \$(AgentPath)

Boundary VM administrator \* \$(UserName)

Boundary VM password \* \$(Password)

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 'BDT\_Scenario' pipeline definition. On the left, there's a list of tasks: 'Environment 1' (Deployment process), 'Agent phase' (Run on agent), 'SCVMM :NewVM action' (SCVMM), 'Agent phase' (Run on agent), and 'Run PowerShell on \$(App)' (PowerShell on Target Machines). The 'Run PowerShell' task is currently selected. On the right, under 'Demands', there is a table with one row:

Name	Condition	Value
Agent.Name	equals	\$(MyAgent)_\$(Release.ReleaseName)

5. Inside the job, add the tasks you require for deployment and testing.

The screenshot shows the configuration for the 'Run PowerShell on \$(App)' task. The 'Machines' field contains the value '\$(AppVM1), \$(AppVM2)', which is highlighted with a red box. Other fields shown include 'Display name' (Run PowerShell on \$(AppVM1), \$(AppVM2)), 'Admin Login' (\$(UserName)), and 'Password' (\$(Password)).

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](#)

## FAQ

### 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
- Get advice on [Stack Overflow](#), and feel free to post your questions, search for answers, or suggest a feature on our [Azure DevOps Developer Community](#). [Support](#) page.

# Build and release tasks

11/2/2020 • 16 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015 | [Previous versions \(XAML builds\)](#)

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

This article provides an index of built-in tasks. To learn more about tasks, including creating custom tasks, custom extensions, and finding tasks on the Visual Studio Marketplace, see [Tasks concepts](#).

## Build

TASK	VERSIONS
 <a href="#">.NET Core CLI task</a> dotnet application, or run a custom dotnet command. For package commands, supports NuGet.org and authenticated feeds like Package Management and MyGet.	Azure Pipelines, TFS 2017 and newer
 <a href="#">Android build task (deprecated; use Gradle)</a> build and release task	Azure Pipelines, TFS 2015 RTM and newer
 <a href="#">Android signing build and release task</a> signing build and release task	Azure Pipelines, TFS 2015 RTM and newer
 <a href="#">Ant build and release task</a> Apache Ant	Azure Pipelines, TFS 2015 RTM and newer
 <a href="#">Azure IoT Edge task</a> applications quickly and efficiently to Azure IoT Edge	Azure Pipelines
 <a href="#">CMake build and release task</a> release task	Azure Pipelines, TFS 2015 RTM and newer
 <a href="#">Docker Compose task</a> container Docker applications. Task can be used with Docker or Azure Container registry.	Azure Pipelines, Azure DevOps Server 2019
 <a href="#">Docker task</a> container registry using Docker registry service connection	Azure Pipelines, TFS 2018 and newer

TASK	VERSIONS
 <a href="#">Go task</a> - Get, build, test a go application, or run a custom go command.	Azure Pipelines
 <a href="#">Gradle build and release task</a> task	- Gradle build and release Azure Pipelines, TFS 2015 RTM and newer
 <a href="#">Grunt build and release task</a> task	- Grunt build and release Azure Pipelines, TFS 2015.3 and newer
 <a href="#">Gulp build and release task</a> task	- Gulp build and release Azure Pipelines, TFS 2015 RTM and newer
 <a href="#">Index Sources &amp; Publish Symbols</a> Publish Symbols build and release task	- Index Sources & Azure Pipelines, TFS 2015 RTM and newer
 <a href="#">Jenkins Queue Job build and release task</a> on a Jenkins server build and release task	- Queue a job Azure Pipelines, TFS 2017 and newer
 <a href="#">Maven build and release task</a> release task	- Maven build and Azure Pipelines, TFS 2015 RTM and newer
 <a href="#">MSBuild build and release task</a> release task	- MSBuild build and Azure Pipelines, TFS 2015 RTM and newer
 <a href="#">SonarQube - Prepare Analysis Configuration</a> Configure all the required settings before executing the build	- Azure Pipelines, TFS 2015.3 and newer
 <a href="#">SonarQube - Publish Quality Gate Result</a> Quality Gate status in the build summary	- Display the Azure Pipelines, TFS 2015.3 and newer
 <a href="#">SonarQube - Run Code Analysis</a> the source code	- Run the analysis of Azure Pipelines, TFS 2015.3 and newer
 <a href="#">Visual Studio Build build and release task</a> Build build and release task	- Visual Studio Azure Pipelines, TFS 2015 RTM and newer
 <a href="#">Xamarin.Android build and release task</a> Xamarin.Android build and release task	- Azure Pipelines, TFS 2015 RTM and newer
 <a href="#">Xamarin.iOS build and release task</a> and release task	- Xamarin.iOS build Azure Pipelines, TFS 2015 RTM and newer

TASK	VERSIONS
 <a href="#">Xcode build and release task</a> task	- Xcode build and release Azure Pipelines
 <a href="#">Xcode Build build and release task</a> and release task	- Xcode Build build and release task TFS 2015, TFS 2017, TFS 2018
 <a href="#">Xcode Package iOS build and release task</a> Package iOS build and release task	- Xcode Package iOS build and release task Azure Pipelines, TFS 2015 RTM and newer

## Utility

TASK	VERSIONS
 <a href="#">Archive Files task</a> source folder	- Use an archive file to then create a source folder Azure Pipelines, TFS 2017 and newer
 <a href="#">Azure Network Load Balancer task</a> disconnect an Azure virtual machine's network interface to a load balancer's address pool	- Connect or disconnect an Azure virtual machine's network interface to a load balancer's address pool Azure Pipelines
 <a href="#">Bash task</a> Windows	- Run a Bash script on macOS, Linux, or Windows Azure Pipelines
 <a href="#">Batch Script task</a> building your code	- Execute .bat or .cmd scripts when building your code Azure Pipelines, TFS 2015 RTM and newer
<a href="#">Cache task</a> - Improve build performance by caching files, like dependencies, between pipeline runs.	Azure Pipelines, TFS 2017 and newer
 <a href="#">Command Line task</a> prompt when building code	- Execute tools from a command prompt when building code Azure Pipelines, TFS 2015 RTM and newer
 <a href="#">Copy and Publish Build Artifacts task</a> artifacts to a staging folder and publish them	- Copy build artifacts to a staging folder and publish them TFS 2015 RTM. Deprecated on Azure Pipelines and newer versions of TFS.
 <a href="#">Copy Files task</a> patterns when building code	- Copy files between folders with match patterns when building code Azure Pipelines, TFS 2015.3 and newer
 <a href="#">cURL Upload Files task</a> supported protocols	- Use cURL to upload files with supported protocols Azure Pipelines, TFS 2015 RTM and newer
 <a href="#">Decrypt File (OpenSSL) task</a> decryption using OpenSSL	- A thin utility task for file decryption using OpenSSL Azure Pipelines

TASK	VERSIONS
 <a href="#">Delay task</a> - Pause execution of a build or release pipeline for a fixed delay time	Azure Pipelines, Azure DevOps Server 2019
 <a href="#">Delete Files task</a> - Delete files from the agent working directory when building code	Azure Pipelines, TFS 2015.3 and newer
 <a href="#">Download Build Artifacts task</a> - Download Build Artifacts task for use in a build or release pipeline	Azure Pipelines
 <a href="#">Download Fileshare Artifacts task</a> - Download Fileshare Artifacts task Artifacts task for Azure Pipelines and TFS	Azure Pipelines
 <a href="#">Download GitHub Release task</a> - Download assets from your GitHub release as part of your pipeline	Azure Pipelines
 <a href="#">Download Package task</a> - Download a package from a Package Management feed in Azure Artifacts or TFS.	Azure Pipelines
 <a href="#">Download Pipeline Artifacts task</a> - Download Pipeline Artifacts task Artifacts task to download pipeline artifacts from earlier stages in this pipeline, or from another pipeline	Azure Pipelines
 <a href="#">Download Secure File task</a> - Download a secure file to a temporary location on the build or release agent in	Azure Pipelines
 <a href="#">Extract Files task</a> - Extract files from archives to a target folder using minimatch patterns on (TFS)	Azure Pipelines, TFS 2017 and newer
 <a href="#">File Transform task</a> - Apply configuration file transformations and variable substitution to a target package or folder	Azure Pipelines, Azure DevOps Server 2019
 <a href="#">FTP Upload task</a> - Upload files to a remote machine using the File Transfer Protocol (FTP), or securely with FTPS on (TFS)	Azure Pipelines, TFS 2017 and newer
 <a href="#">GitHub Release task</a> - Create, edit, or discard a GitHub release.	Azure Pipelines
 <a href="#">Install Apple Certificate task</a> - Install an Apple certificate required to build on a macOS agent on (TFS)	Azure Pipelines, TFS 2018 and newer
 <a href="#">Install Apple Provisioning Profile task</a> - Install an Apple provisioning profile required to build on a macOS agent	Azure Pipelines, TFS 2018 and newer

TASK	VERSIONS
 <a href="#">Install SSH Key task</a> or release - Install an SSH key prior to a build or release	Azure Pipelines
 <a href="#">Invoke Azure Function task</a> function in an Azure function app and parse the response - Invoke a HTTP triggered	Azure Pipelines, TFS 2017 and newer
 <a href="#">Invoke HTTP REST API task</a> invoke an HTTP API and parse the response with a build or release pipeline - Build and release task to	Azure Pipelines, TFS 2018 and newer
 <a href="#">Jenkins Download Artifacts task</a> produced by a Jenkins job - Download artifacts	Azure Pipelines, TFS 2017 and newer
 <a href="#">Manual Intervention task</a> within a stage in a release pipeline - Pause an active deployment	Azure Pipelines, Azure DevOps Server 2019
 <a href="#">PowerShell task</a> - Execute PowerShell scripts	Azure Pipelines, TFS 2015 RTM and newer
 <a href="#">Publish Build Artifacts task</a> Azure Pipelines, Team Foundation Server (TFS), or to a file share - Publish build artifacts to	Azure Pipelines, TFS 2015 RTM and newer
 <a href="#">Publish Pipeline Artifacts task</a> Azure Pipelines. - Publish artifacts to	Azure Pipelines
 <a href="#">Publish To Azure Service Bus task</a> an Azure Service Bus with a build or release pipeline - Send a message to	Azure Pipelines, Azure DevOps Server 2019
 <a href="#">Python Script task</a> release pipeline - Run a Python script in a build or	Azure Pipelines
 <a href="#">Query Azure Monitor Alerts task</a> configured Azure monitor rules for active alerts in a build or release pipeline - Observe the	Azure Pipelines, TFS 2017 and newer
 <a href="#">Query Work Items task</a> - Ensure the number of matching items returned by a work item query is within the configured threshold	Azure Pipelines, TFS 2017 and newer
 <a href="#">Service Fabric PowerShell Utility task</a> PowerShell task for use in build or release pipelines in - Service Fabric	Azure Pipelines, Azure DevOps Server 2019
 <a href="#">Shell Script task</a> code - Execute a bash script when building	Azure Pipelines, TFS 2015 RTM and newer

TASK	VERSIONS
 <a href="#">Update Service Fabric Manifests task</a> - Update the Service Fabric App versions	Azure Pipelines, TFS 2017 and newer
 <a href="#">Xamarin License task</a> - Activate or deactivate a Xamarin license when building code	Azure Pipelines, TFS 2015 RTM and newer

## Test

TASK	VERSIONS
 <a href="#">App Center Test task</a> - Test app packages with Visual Studio App Center.	Azure Pipelines, TFS 2017 and newer
 <a href="#">Cloud-based Apache JMeter Load Test task (Deprecated)</a> - Runs the Apache JMeter load test in cloud	Azure Pipelines
 <a href="#">Cloud-based Load Test task (Deprecated)</a> - Runs the Cloud-based Load Test task in cloud with a build or release pipeline with Azure Pipelines to integrate cloud-based load tests into your build and release pipelines	Azure Pipelines, TFS 2015 RTM and newer
 <a href="#">Cloud-based Web Performance Test task (Deprecated)</a> - Runs the Quick Web Performance Test with a build or release pipeline to easily verify your web application exists and is responsive	Azure Pipelines, TFS 2015 RTM and newer
 <a href="#">Container Structure Test Task</a> - Test container structure by container task and integrate test reporting into your build and release pipelines	Azure Pipelines
 <a href="#">Publish Code Coverage Results task</a> - Publish Cobertura or JaCoCo code coverage results from an Azure Pipelines or TFS build	Azure Pipelines, TFS 2015 RTM and newer
 <a href="#">Publish Test Results task</a> - Publish Test Results to integrate test reporting into your build and release pipelines	Azure Pipelines, TFS 2015 RTM and newer
 <a href="#">Run Functional Tests task</a> - Run Coded UI/Selenium/Functional tests on a set of machines using the Test Agent to integrate cloud-based load tests into your build and release pipelines	Azure Pipelines, TFS 2015 RTM and newer
 <a href="#">Visual Studio Test Agent Deployment task</a> - Deploy and configure the Test Agent to run tests on a set of machines to integrate cloud-based load tests into your build and release pipelines	Azure Pipelines, TFS 2015 RTM and newer

TASK	VERSIONS
 <b>Visual Studio Test task</b> - 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
 <b>Xamarin Test Cloud task</b> - This task is deprecated. Use the App Center Test task instead.	Azure Pipelines, TFS 2015 RTM and newer

## Package

TASK	VERSIONS
 <b>CocoaPods task</b> - Learn all about how you can use CocoaPods packages when you are building code in Azure Pipelines or Team Foundation Server (TFS).	Azure Pipelines, TFS 2015 RTM and newer
 <b>Conda Environment task</b> - How to create and activate a Conda environment when building code	Azure Pipelines
 <b>Maven Authenticate task (for task runners)</b> - Provides credentials for Azure Artifacts feeds and external Maven repositories.	Azure Pipelines
 <b>npm Authenticate task (for task runners)</b> - 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.	Azure Pipelines
 <b>npm task</b> - How to use npm packages when building code in Azure Pipelines	Azure Pipelines, TFS 2015 RTM and newer
 <b>NuGet Authenticate</b> - Configure NuGet tools to authenticate with Azure Artifacts and other NuGet repositories	Azure Pipelines
 <b>NuGet restore, pack, and publish task</b> - Learn all about how you can make use of NuGet packages when you are building code	Azure Pipelines, TFS 2018 and newer
 <b>PyPI Publisher task (Deprecated)</b> - How to upload a package to PyPI when building code	Azure Pipelines
 <b>Python Pip Authenticate</b> - Sets up authentication with pip so you can perform pip commands in your pipeline.	Azure Pipelines

TASK	VERSIONS
 <a href="#">Python Twine Upload Authenticate</a> - Sets up authentication with twine to Python feeds so you can publish Python packages in your pipeline.	Azure Pipelines
 <a href="#">Universal Package, download and publish task</a> - Learn all about how you can make use of Universal packages when you are building code	Azure Pipelines, TFS 2018 and newer

## Deploy

TASK	VERSIONS
 <a href="#">App Center Distribute task</a> - Distribute app builds to testers and users through App Center	Azure Pipelines, TFS 2017 and newer
 <a href="#">Azure App Service Deploy task</a> - The Azure App Service Deploy task is used to update Azure App Services to deploy Web Apps, Functions, and WebJobs.	Azure Pipelines, Azure DevOps Server 2019
 <a href="#">Azure App Service Manage task</a> - Start, Stop, Restart, Slot swap, Swap with Preview, Install site extensions, or Enable Continuous Monitoring for an Azure App Service	Azure Pipelines
 <a href="#">Azure App Service Settings task</a> Settings Task supports configuring App settings, connection strings and other general settings in bulk using JSON syntax on your web app or any of its deployment slots.	Azure Pipelines
 <a href="#">Azure CLI task</a> - build task to run a shell or batch script containing Microsoft Azure CLI commands	Azure Pipelines, Azure DevOps Server 2019
 <a href="#">Azure Cloud Service Deployment task</a> - Deploy an Azure Cloud Service	Azure Pipelines
 <a href="#">Azure Database for MySQL Deployment task</a> - Run your scripts and make changes to your Azure DB for MySQL.	Azure Pipelines
 <a href="#">Azure File Copy task</a> - build task to copy files to Microsoft Azure storage blobs or virtual machines (VMs)	Azure Pipelines, TFS 2015.3 and newer
 <a href="#">Azure Function App for Container task</a> - Deploy Azure Functions on Linux using custom images	Azure Pipelines

TASK	VERSIONS
 <a href="#">Azure Function App task</a> - The Azure App Service Deploy task is used to update Azure App Services to deploy Web Apps, Functions, and WebJobs.	Azure Pipelines
 <a href="#">Azure Key Vault task</a> - Azure Key Vault task for use in the jobs of all of your build and release pipelines	Azure Pipelines, Azure DevOps Server 2019
 <a href="#">Azure Monitor Alerts task</a> - Configure alerts on available metrics for an Azure resource	Azure Pipelines
 <a href="#">Azure Policy task</a> - Security and compliance assessment with Azure policies	Azure Pipelines, Azure DevOps Server 2019
 <a href="#">Azure PowerShell task</a> - Run a PowerShell script within an Azure environment	Azure Pipelines
 <a href="#">Azure Resource Group Deployment task</a> - Deploy, start, stop, or delete Azure Resource Groups	Azure Pipelines
 <a href="#">Azure Spring Cloud task</a> - Deploy applications to Azure Spring Cloud and manage Azure Spring Cloud deployments.	Azure Pipelines
 <a href="#">Azure SQL Database Deployment task</a> - Deploy Azure SQL DB using DACPAC or run scripts using SQLCMD	Azure Pipelines
 <a href="#">Azure virtual machine scale set deployment task</a> - Deploy virtual machine scale set image	Azure Pipelines
 <a href="#">Azure Web App for Container task</a> - Deploy Web Apps, Functions, and WebJobs to Azure App Services	Azure Pipelines
 <a href="#">Azure Web App task</a> - The Azure App Service Deploy task is used to update Azure App Services to deploy Web Apps, Functions, and WebJobs.	Azure Pipelines
 <a href="#">Build Machine Image task</a> - Build a machine image using Packer to use for Azure Virtual machine scale set deployment	Azure Pipelines
 <a href="#">Chef Knife task</a> - Run scripts with Knife commands on your Chef workstation	Azure Pipelines
 <a href="#">Chef task</a> - Deploy to Chef environments by editing environment attributes	Azure Pipelines

TASK	VERSIONS
 <b>Copy Files Over SSH task</b> - Copy Files Over SSH task for use in the jobs of all of your build and release pipelines	Azure Pipelines, TFS 2017 and newer
 <b>IIS Web App Deploy task</b> - Deploy a website or web app using WebDeploy	Azure Pipelines
 <b>IIS Web App Manage task</b> - Create or update a Website, Web App, Virtual Directory, or Application Pool	Azure Pipelines
 <b>Kubectl task</b> - Deploy, configure, or update a Kubernetes cluster in Azure Container Service by running kubectl commands.	Azure Pipelines
 <b>Kubernetes manifest task</b> - Bake and deploy manifests to Kubernetes clusters	Azure Pipelines
 <b>MySQL Database Deployment On Machine Group task</b> The task is used to deploy for MySQL Database.	Azure Pipelines
 <b>Package and Deploy Helm Charts task</b> - Deploy, configure, update your Kubernetes cluster in Azure Container Service by running helm commands.	Azure Pipelines, Azure DevOps Server 2019
 <b>PowerShell on Target Machines task</b> Target Machines build task	Azure Pipelines, TFS 2015 RTM and newer
 <b>Service Fabric Application Deployment task</b> Fabric Application Deployment task	Azure Pipelines, TFS 2017 and newer
 <b>Service Fabric Compose Deploy task</b> Compose Deploy Deployment task	Azure Pipelines, Azure DevOps Server 2019
 <b>SSH Deployment task</b> - SSH task for use in the jobs of all of your build and release pipelines	Azure Pipelines, TFS 2017 and newer
 <b>Windows Machine File Copy task</b> - Copy application files and other artifacts to remote Windows machines	Azure Pipelines, TFS 2015 RTM and newer
 <b>WinRM SQL Server DB Deployment task</b> - Deploy to SQL Server Database using DACPAC or SQL scripts	Azure Pipelines

## Tool

TASK	VERSIONS
 <a href="#">Docker Installer task</a> - Install the Docker CLI on an agent machine	Azure Pipelines, Azure DevOps Server 2019
 <a href="#">Go Tool Installer task</a> - Finds or downloads a specific version of the Go tool into the tools cache and adds it to the PATH	Azure Pipelines
 <a href="#">Helm installer task</a> - Install helm on an agent machine	Azure Pipelines
 <a href="#">Java Tool Installer task</a> - Change the version of Java	Azure Pipelines
 <a href="#">Kubectl installer task</a> - Install kubectl on an agent machine	Azure Pipelines
 <a href="#">Node.js Tool Installer task</a> - Find, download, and cache a specified version of Node.js and add it to the PATH	Azure Pipelines
 <a href="#">NuGet Tool Installer task</a> - Find, download, and cache a specified version of NuGet and add it to the PATH	Azure Pipelines
 <a href="#">Use .NET Core task</a> Core from the internet or the tools cache and adds it to the PATH - Acquires a specific version of .NET	Azure Pipelines
 <a href="#">Use Python Version task</a> run on an agent and optionally add it to PATH - Select a version of Python to run	Azure Pipelines
 <a href="#">Use Ruby Version task</a> on an agent and optionally add it to PATH - Select a version of Ruby to run	Azure Pipelines
 <a href="#">Visual Studio Test Platform Installer task</a> test platform from nuget.org or the tools cache and can allow you to run tests and collect diagnostic data - Acquires the	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.

## FAQ

**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.

### **I'm having problems. How can I troubleshoot them?**

See [Troubleshoot Build and Release](#).

### **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](#).

### **My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?**

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE.EXTRA.CA.CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

### **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

4/27/2021 • 10 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

## NOTE

The [NuGet Authenticate](#) task is the new recommended way to authenticate with Azure Artifacts and other NuGet repositories. The restore and push commands of this task no longer take new features and only critical bugs are addressed.

## Azure Pipelines

Use this task to build, test, package, or publish a dotnet application, or to run a custom dotnet command. For package commands, this task supports NuGet.org and authenticated feeds like Package Management and MyGet.

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

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## YAML snippet

```

.NET Core
Build, test, package, or publish a dotnet application, or run a custom dotnet command
- 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
 #testRunTitle: # Optional
 #zipAfterPublish: true # Optional
 #modifyOutputPath: true # Optional
 #feedsToUse: 'select' # Options: select, config
 #vstsFeed: # Required when feedsToUse == Select
 #feedRestore: # Required when command == restore. projectName/feedName for project-scoped feed. FeedName
only for organization-scoped feed.
 #includeNuGetOrg: true # Required when feedsToUse == Select
 #nugetConfigPath: # Required when feedsToUse == Config
 #externalFeedCredentials: # Optional
 #noCache: false
 restoreDirectory:
 #restoreArguments: # Optional
 #verbosityRestore: 'Detailed' # Options: -, quiet, minimal, normal, detailed, diagnostic
 #packagesToPush: '$(Build.ArtifactStagingDirectory)/*.nupkg' # Required when command == Push
 #nuGetFeedType: 'internal' # Required when command == Push# Options: internal, external
 #publishVstsFeed: # Required when command == Push && NuGetFeedType == Internal
 #publishPackageMetadata: true # Optional
 #publishFeedCredentials: # Required when command == Push && NuGetFeedType == External
 #packagesToPack: '**/*.csproj' # Required when command == Pack
 #packDirectory: '$(Build.ArtifactStagingDirectory)' # Optional
 #nobuild: false # Optional
 #includesymbols: false # Optional
 #includesource: 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
<code>command</code> Command	The dotnet command to run. Select <code>custom</code> to add arguments or use a command not listed here. Options: <code>build</code> , <code>push</code> , <code>pack</code> , <code>publish</code> , <code>restore</code> , <code>run</code> , <code>test</code> , <code>custom</code>
<code>selectOrConfig</code> Feeds to use	You can either choose to 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 using the <code>nugetConfigPath</code> argument. Options: <code>select</code> , <code>config</code> Argument aliases: <code>feedsToUse</code>

ARGUMENT	DESCRIPTION
<p><code>versioningScheme</code></p> <p>Automatic package versioning</p>	<p>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.</p> <p>If you choose 'Use an environment variable', you must select an environment variable and ensure it contains the version number you want to use.</p> <p>If you choose 'Use the build number', this will use the build number to version your package. <b>Note:</b> Under Options set the build number format to be <code>'\$(BuildDefinitionName)_\$(Year:yyyy).\$(Month).\$(DayOfMonth)\$(Rev:r)'</code></p> <p>Options: <code>off</code> , <code>byPrereleaseNumber</code> , <code>byEnvVar</code> , <code>byBuildNumber</code> ,</p>
<p><code>arguments</code></p> <p>Arguments</p>	<p>Arguments to the selected command. For example, build configuration, output folder, runtime. The arguments depend on the command selected</p> <p>Note: This input only currently accepts arguments for <code>build</code> , <code>publish</code> , <code>run</code> , <code>test</code> , <code>custom</code> . If you would like to add arguments for a command not listed, use <code>custom</code> .</p>
<p><code>projects</code></p> <p>Path to project(s)</p>	<p>The path to the csproj file(s) to use. You can use wildcards (e.g. <code>**/*.csproj</code> for all .csproj files in all subfolders), see <a href="#">file matching patterns</a>.</p>
<p><code>noCache</code></p> <p>Disable local cache</p>	<p>Prevents NuGet from using packages from local machine caches.</p>
<p><code>restoreArguments</code></p> <p>Restore arguments</p>	<p>Write the additional arguments to be passed to the <code>restore</code> command.</p>
<p><code>packagesDirectory</code></p> <p>Destination directory</p>	<p>Specifies the folder in which packages are installed. If no folder is specified, packages are restored into the default NuGet package cache</p> <p>Argument aliases: <code>restoreDirectory</code></p>
<p><code>buildProperties</code></p> <p>Additional build properties</p>	<p>Specifies a list of <code>token = value</code> pairs, separated by semicolons, where each occurrence of <code>\$token\$</code> in the .nuspec file will be replaced with the given value. Values can be strings in quotation marks</p>
<p><code>verbosityPack</code></p> <p>Verbosity</p>	<p>Specifies the amount of detail displayed in the output for the <code>pack</code> command.</p>
<p><code>verbosityRestore</code></p> <p>Verbosity</p>	<p>Specifies the amount of detail displayed in the output for the <code>restore</code> command.</p>
<p><code>workingDirectory</code></p> <p>Working Directory</p>	<p>Current working directory where the script is run. Empty is the root of the repo (build) or artifacts (release), which is <code>\$(System.DefaultWorkingDirectory)</code></p>

ARGUMENT	DESCRIPTION
<code>searchPatternPush</code> Path to NuGet package(s) to publish	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 <code>!</code> . <b>Example:</b> <code>**/*.nupkg; !**/*.Tests.nupkg.</code> Argument aliases: <code>packagesToPush</code>
<code>nuGetFeedType</code> Target feed location	Specifies whether the target feed is internal or external. Options: <code>internal</code> , <code>external</code>
<code>feedPublish</code> Target feed	Select a feed hosted in your organization. You must have Package Management installed and licensed to select a feed here Argument aliases: <code>publishVstsFeed</code>
<code>publishPackageMetadata</code> Publish pipeline metadata	Associate this build/release pipeline's metadata (run ID, source code information) with the package
<code>externalEndpoint</code> NuGet server	The NuGet <a href="#">service connection</a> that contains the external NuGet server's credentials. Argument aliases: <code>publishFeedCredentials</code>
<code>searchPatternPack</code> Path to csproj or nuspec file(s) to pack	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 <code>!</code> . <b>Example:</b> <code>**/*.csproj; !**/*.Tests.csproj</code> Argument aliases: <code>packagesToPack</code>
<code>configurationToPack</code> Configuration to Package	When using a csproj file this specifies the configuration to package. Argument aliases: <code>configuration</code>
<code>outputDir</code> Package Folder	Folder where packages will be created. If empty, packages will be created alongside the csproj file. Argument aliases: <code>packDirectory</code>
<code>nobuild</code> Do not build	Don't build the project before packing. Corresponds to the <code>--no-build</code> parameter of the `build` command.
<code>includesymbols</code> Include Symbols	Additionally creates symbol NuGet packages. Corresponds to the <code>--include-symbols</code> command line parameter.
<code>includesource</code> Include Source	Includes source code in the package. Corresponds to the <code>--include-source</code> command line parameter.
<code>publishWebProjects</code> Publish Web Projects	If <code>true</code> , the <code>projects</code> property value will be skipped and 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. In the absence of a web.config file or wwwroot folder, projects that use a web SDK, like Microsoft.NET.Sdk.Web, are selected. Note that this argument defaults to <code>true</code> if not specified.

ARGUMENT	DESCRIPTION
<code>zipAfterPublish</code> Zip Published Projects	If <code>true</code> , folder created by the publish command will be zipped and deleted.
<code>modifyOutputPath</code> Add project name to publish path	If <code>true</code> , 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.
<code>publishTestResults</code> Publish test results	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. Code coverage can be collected by adding <code>--collect "Code coverage"</code> to the command line arguments. This is currently only available on the Windows platform.
<code>testRunTitle</code> Test run title	Provides a name for the test run
<code>custom</code> Custom command	The command to pass to dotnet.exe for execution. For a full list of available commands, see the <a href="#">dotnet CLI documentation</a>
<code>feedRestore</code> Use packages from this Azure Artifacts/TFS feed	Include the selected feed in the generated NuGet.config. You must have Package Management installed and licensed to select a feed here. <code>projectName/feedName</code> for project-scoped feed. <code>FeedName</code> only for organization-scoped feed. Note that this is not supported for the test command. Argument aliases: <code>vstsFeed</code>
<code>includeNuGetOrg</code> Use packages from NuGet.org	Include NuGet.org in the generated NuGet.config000 0.
<code>nugetConfigPath</code> Path to NuGet.config	The NuGet.config in your repository that specifies the feeds from which to restore packages.
<code>externalEndpoints</code> Credentials for feeds outside this organization/collection	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 Argument aliases: <code>externalFeedCredentials</code>
<code>versionEnvVar</code> Environment variable	Enter the variable name without \$, \$env, or %
<code>requestedMajorVersion</code> Major	The 'X' in version <code>X.Y.Z</code> . Argument aliases: <code>majorVersion</code>
<code>requestedMinorVersion</code> Minor	The 'Y' in version <code>X.Y.Z</code> . Argument aliases: <code>minorVersion</code>

ARGUMENT	DESCRIPTION
<code>requestedPatchVersion</code> Patch	The 'Z' in version <code>X.Y.Z</code> . Argument aliases: <code>patchVersion</code>
CONTROL OPTIONS	

## Examples

### Build

#### Build a project

```
Build project
- task: DotNetCoreCLI@2
 inputs:
 command: 'build'
```

#### Build Multiple Projects

```
Build multiple projects
- task: DotNetCoreCLI@2
 inputs:
 command: 'build'
 projects: |
 src/proj1/proj1.csproj
 src/proj2/proj2.csproj
 src/other/other.sln # Pass a solution instead of a csproj.
```

### Push

#### Push NuGet packages to internal feed

```
Push non test NuGet packages from a build to internal organization Feed
- task: DotNetCoreCLI@2
 inputs:
 command: 'push'
 searchPatternPush:
 '$(Build.ArtifactStagingDirectory)/*.nupkg;!$(Build.ArtifactStagingDirectory)/*.Tests.nupkg'
 feedPublish: 'FabrikamFeed'
```

#### Push NuGet packages to external feed

```
Push all NuGet packages from a build to external Feed
- task: DotNetCoreCLI@2
 inputs:
 command: 'push'
 nugetFeedType: 'external'
 externalEndPoint: 'MyNugetServiceConnection'
```

### Pack

#### Pack a NuGetPackage to a specific output directory

```
Pack a NuGet package to a test directory
- task: DotNetCoreCLI@2
 inputs:
 command: 'pack'
 outputDir: '$(Build.ArtifactStagingDirectory)/TestDir'
```

## Pack a Symbol Package

```
Pack a symbol package along with NuGet package
- task: DotNetCoreCLI@2
 inputs:
 command: 'pack'
 includesymbols: true
```

# Publish

## Publish projects to specified folder

```
Publish projects to specified folder.
- task: DotNetCoreCLI@2
 displayName: 'dotnet publish'
 inputs:
 command: 'publish'
 publishWebProjects: false
 projects: '**/*.csproj'
 arguments: '-o $(Build.ArtifactStagingDirectory)/Output'
 zipAfterPublish: true
 modifyOutputPath: true
```

# Restore

```
#Restore packages with the .NET Core CLI task
- task: DotNetCoreCLI@2
 displayName: 'dotnet restore'
 inputs:
 command: 'restore'
 feedsToUse: 'select'
 feedRestore: ' projectName/feedName '
 projects: '**/*.csproj'
 includeNuGetOrg: true
```

# Test

## Run tests in your repository

```
Run tests and auto publish test results.
- task: DotNetCoreCLI@2
 inputs:
 command: 'test'
```

# FAQ

## Why is my build, publish, or test step failing to restore packages?

Most `dotnet` commands, including `build`, `publish`, and `test` 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.

In addition, the `test` command does not recognize the `feedRestore` or `vstsFeed` arguments and feeds specified in this manner will not be included in the generated NuGet.config file when the implicit `restore` step runs. It is recommended that an explicit `dotnet restore` step be used to restore packages. The `restore` command respects the `feedRestore` and `vstsFeed` arguments.

### 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.

## Troubleshooting

### File structure for output files is different from previous builds

Azure DevOps hosted agents are configured with .NET Core 3.0, 2.1 and 2.2. CLI for .NET Core 3.0 has a different behavior while publishing projects using output folder argument. When publishing projects with the output folder argument (-o), the output folder is created in the root directory and not in the project file's directory. Hence while publishing more than one project, all the files are published to the same directory, which causes an issue.

To resolve this issue, use the *Add project name to publish path* parameter (modifyOutputPath in YAML) in the .NET Core CLI task. This creates a sub folder with project file's name, inside the output folder. Hence all your projects will be published under different subfolder's inside the main output folder.

```
steps:
- task: DotNetCoreCLI@2
 displayName: 'dotnet publish'
 inputs:
 command: publish
 publishWebProjects: false
 projects: '**/*.csproj'
 arguments: '-o testpath'
 zipAfterPublish: false
 modifyOutputPath: true
```

### Project using Entity Framework has stopped working on Hosted Agents

The latest .NET Core 3.0 does not have Entity Framework(EF) built in. You will have to either install EF before beginning execution or add global.json to the project with required .NET Core SDK version. This will ensure that correct SDK is used to build EF project. If the required version is not present on the machine, add UseDotNetV2 task to your pipeline to install the required version. [Learn more about EF with .NET Core 3.0](#)

## Open Source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Android build task (deprecated; use Gradle)

4/10/2020 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

Use this task 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 <a href="#">The Gradle Wrapper</a>.</p>
Project Directory	Relative path from the repo root to the root directory of the application (likely where your build.gradle file is).
Gradle Arguments	<p>Provide any options to pass to the Gradle command line. The default value is <code>build</code></p> <p>See <a href="#">Gradle command line</a>.</p>

### ANDROID VIRTUAL DEVICE (AVD) OPTIONS

Name	Name of the AVD to be started or created.  <div style="border: 1px solid #ccc; padding: 5px;"><p><b>Note:</b> You must deploy your own <a href="#">agent</a> to use this option. You cannot use a Microsoft-hosted pool if you want to create an AVD.</p></div>
Create AVD	Select this check box if you would like the AVD to be created if it does not exist.

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 <a href="#">ABI Management</a> .
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 <a href="#">Android Command Line</a> .
<b>EMULATOR OPTIONS</b>	
Start and Stop Android Emulator	Check if you want the emulator to be started and stopped when Android Build task finishes.  <b>Note:</b> You must deploy your own <a href="#">agent</a> 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.
<b>CONTROL OPTIONS</b>	

## Related tasks

[Android Signing](#)

# Android signing task

2/26/2020 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

Use this task in a 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
<code>files</code> APK files	(Required) Relative path from the repo root to the APK(s) you want to sign. You can use wildcards to specify multiple files. For example: <ul style="list-style-type: none"><li><code>outputs\apk*.apk</code> to sign all .APK files in the <code>outputs\apk\</code> subfolder</li><li><code>*\bin\apk*.apk</code> to sign all .APK files in all bin subfolders</li></ul> Default value: <code>**/*.apk</code> Argument aliases: <code>apkFiles</code>
SIGNING OPTIONS	
<code>apksign</code> Sign the APK	(Optional) Select this option to sign the APK with a provided Android Keystore file. Unsigned APKs can only run in an emulator. APKs must be signed to run on a device. Default value: true

ARGUMENT	DESCRIPTION
<code>keystoreFile</code> Keystore file	(Required) Select or enter the file name of the Android Keystore file that should be used to sign the APK. This file must be uploaded to the <a href="#">secure files</a> library where it is securely stored with encryption. The Android Keystore file will be used to sign the APK, but will be removed from the agent machine when the pipeline completes. Argument aliases: <code>apksignerKeystoreFile</code>
<code>keystorePass</code> Keystore password	(Optional) Enter the password for the provided Android Keystore file.  <b>Important:</b> Use a new variable with its lock enabled on the Variables pane to encrypt this value. See <a href="#">secret variables</a> .
<code>keystoreAlias</code> Alias	(Optional) Enter the alias that identifies the public/private key pair to be used in the keystore file. Argument aliases: <code>apksignerKeystoreAlias</code>
<code>keyPass</code> Key password	(Optional) Enter the key password for the alias and Android Keystore file.  <b>Important:</b> Use a new variable with its lock enabled on the Variables pane to encrypt this value. See <a href="#">secret variables</a> .
<code>apksignerArguments</code> apksigner arguments	(Optional) Provide any options to pass to the apksigner command line. Default is: <code>--verbose</code> .  See the <a href="#">apksigner documentation</a> .  Default value: <code>--verbose</code>
<code>apksignerLocation</code> apksigner location	(Optional) Optionally specify the location of the apksigner executable used during signing. This defaults to the apksigner found in the Android SDK version folder that your application builds against.  Argument aliases: <code>apksignerFile</code>
<b>ZIPALIGN OPTIONS</b>	
<code>zipalign</code> Zipalign	(Optional) Select if you want to zipalign your package. This reduces the amount of RAM consumed by an app.  Default value: true

ARGUMENT	DESCRIPTION
<code>zipalignLocation</code> Zipalign location	(Optional) Optionally specify the location of the zipalign executable used during signing. This defaults to the zipalign found in the Android SDK version folder that your application builds against.
Argument aliases: <code>zipalignFile</code>	

## CONTROL OPTIONS

## Related tasks

[Android Build](#)

# Ant task

6/2/2020 • 4 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

Use this task 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
 #codeCoverageClassDirectories: '.' # Required when codeCoverageToolOptions != None
 #codeCoverageClassFilter: # Optional. Comma-separated list of filters to include or exclude classes from
collecting code coverage. For example: +:com.*,+:org.*,-:my.app.*'
 #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
<code>antBuildFile</code> Ant build file	(Required) Relative path from the repository root to the Ant build file. For more information about build files, see <a href="#">Using Apache Ant</a> Default value: build.xml Argument aliases: <code>buildFile</code>

ARGUMENT	DESCRIPTION
<code>options</code> Options	(Optional) Options that you want to pass to the Ant command line. 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> ). See <a href="#">Running Apache Ant</a> .
<code>targets</code> Target(s)	(Optional) Target(s) for Ant to execute for this build. See <a href="#">Using Apache Ant Targets</a> .
<b>JUNIT TEST RESULTS</b>	
<code>publishJUnitResults</code> Publish to Azure Pipelines	(Required) 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. Default value: true
<code>testResultsFiles</code> Test Results Files	(Required) Test results files path. Wildcards can be used. For example, <code>*/TEST-.xml</code> for all xml files whose name starts with TEST-." Default value: <code>**/TEST-*.xml</code>
<code>testRunTitle</code> Test Run Title	(Optional) Assign a title for the JUnit test case results for this build.
<b>CODE COVERAGE</b>	
<code>codeCoverageTool</code> Code Coverage Tool	(Optional) Select the code coverage tool you want to use.  If you are using the <a href="#">Microsoft-hosted agents</a> , then the tools are set up for you. If you are using on-premises <a href="#">Windows agent</a> , then if you select: <ul style="list-style-type: none"><li>• JaCoCo, make sure jacocoant.jar is available in lib folder of Ant installation. See <a href="#">JaCoCo</a>.</li><li>• Cobertura, set up an environment variable COBERTURA_HOME pointing to the Cobertura jar files location. See <a href="#">Cobertura</a>.</li></ul> After you select one of these tools, the following arguments appear.  Default value: None Argument aliases: <code>codeCoverageToolOptions</code>

ARGUMENT	DESCRIPTION
<b>classFilesDirectories</b> Class Files Directories	<p>(Required) 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.</p> <p>For example: target/classes,target/testClasses.</p> <p>Default value: .</p> <p>Argument aliases: <code>codeCoverageClassFilesDirectories</code></p>
<b>classFilter</b> Class Inclusion/Exclusion Filters	<p>(Optional) Specify a comma-separated list of filters to include or exclude classes from collecting code coverage.</p> <p>For example: <code>+:com.,+:org.,-:my.app.</code></p> <p>Argument aliases: <code>codeCoverageClassFilter</code></p>
<b>srcDirectories</b> Source Files Directories	<p>(Optional) 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: src/java,src/Test.</p> <p>Argument aliases: <code>codeCoverageSourceDirectories</code></p>
<b>failIfCoverageEmpty</b> Fail when code coverage results are missing	<p>(Optional) Fail the build if code coverage did not produce any results to publish</p> <p>Default value: false</p> <p>Argument aliases: <code>codeCoverageFailIfEmpty</code></p>
ADVANCED	
<b>antHomeUserInputPath</b> Set ANT_HOME Path	<p>(Optional) If set, overrides any existing ANT_HOME environment variable with the given path.</p> <p>Argument aliases: <code>antHomeDirectory</code></p>
<b>javaHomeSelection</b> Set JAVA_HOME by	<p>(Required) Sets JAVA_HOME either by selecting a JDK version that will be discovered during builds or by manually entering a JDK path.</p> <p>Default value: JDKVersion</p> <p>Argument aliases: <code>javaHomeOption</code></p>
<b>jdkVersion</b> JDK version	<p>(Optional) Will attempt to discover the path to the selected JDK version and set JAVA_HOME accordingly.</p> <p>Default value: default</p> <p>Argument aliases: <code>jdkVersionOption</code></p>
<b>jdkUserInputPath</b> JDK Path	<p>(Required) Sets JAVA_HOME to the given path</p> <p>Argument aliases: <code>jdkUserInputDirectory</code></p>

ARGUMENT	DESCRIPTION
<code>jdkArchitecture</code> JDK Architecture	(Optional) Optionally supply the architecture (x86, x64) of the JDK. Default value: x64 Argument aliases: <code>jdkArchitectureOption</code>
CONTROL OPTIONS	

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### Do I need an agent?

You need at least one [agent](#) to run your build or release.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

### 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 IoT Edge task

3/24/2021 • 3 minutes to read • [Edit Online](#)

Use this task to build, test, and deploy applications quickly and efficiently to Azure IoT Edge.

## Container registry types

### Azure Container Registry

PARAMETERS	DESCRIPTION
<code>containerregistrytype</code> Container registry type	(Required) Select Azure Container Registry for ACR or Generic Container Registry for generic registries including Docker hub.
<code>azureSubscriptionEndpoint</code> Azure subscription	(Required, if <code>containerregistrytype</code> = Azure Container Registry) Select an Azure subscription.
<code>azureContainerRegistry</code> Azure Container Registry	(Required) Select an Azure Container Registry.

### Other container registries

PARAMETERS	DESCRIPTION
<code>containerregistrytype</code> Container registry type	(Required) Select Azure Container Registry for ACR or Generic Container Registry for generic registries including Docker hub.
<code>dockerRegistryEndpoint</code> Docker Registry Connection	(Required) Select a generic Docker registry connection. Required for Build and Push Argument aliases: <code>dockerRegistryConnection</code>

## Build module images

PARAMETERS	DESCRIPTION
<code>action</code> Action	(Required) Select an Azure IoT Edge action. Default value: Build module images.
<code>templateFilePath</code> .template.json file	(Required) The path of your Azure IoT Edge solution .template.json file. This file defines the modules and routes in an Azure IoT Edge solution. The filename must end with .template.json. Default value: deployment.template.json.
<code>defaultPlatform</code> Default platform	(Required) In your .template.json file you can leave the modules platform unspecified, in which case the default platform will be used. Default value: amd64.

The following YAML example builds module images:

```

- task: AzureIoTEdge@2
 displayName: AzureIoTEdge - Build module images
 inputs:
 action: Build module images
 templateFilePath: deployment.template.json
 defaultPlatform: amd64

```

## Push module images

PARAMETERS	DESCRIPTION
<code>action</code> Action	(Required) Select an Azure IoT Edge action. Default value: Build module images.
<code>templateFilePath</code> .template.json file	(Required) The path of your Azure IoT Edge solution .template.json file. This file defines the modules and routes in an Azure IoT Edge solution. The filename must end with .template.json. Default value: deployment.template.json.
<code>defaultPlatform</code> Default platform	(Required) In your .template.json file you can leave the modules platform unspecified, in which case the default platform will be used. Default value: amd64.
<code>bypassModules</code> Bypass module(s)	(Optional) Specify the module(s) that you do not need to build or push from the list of module names separated by commas in the .template.json file. For example, if you have two modules, "SampleModule1,SampleModule2" in your file and you want to build or push just SampleModule1, specify SampleModule2 as the bypass module(s). Leave empty to build or push all the modules in .template.json.

The following YAML example pushes module images:

```

variables:
 azureSubscriptionEndpoint: Contoso
 azureContainerRegistry: contoso.azurecr.io

steps:
- task: AzureIoTEdge@2
 displayName: AzureIoTEdge - Push module images
 inputs:
 action: Push module images
 containerregistrytype: Azure Container Registry
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureContainerRegistry: {"loginServer": "$(azureContainerRegistry)", "id": "/subscriptions/$(azureSubscriptionEndpoint)"}
 templateFilePath: deployment.template.json
 defaultPlatform: amd64

```

## Deploy to IoT Edge devices

PARAMETERS	DESCRIPTION
<code>action</code> Action	(Required) Select an Azure IoT Edge action. Default value: Build module images.

PARAMETERS	DESCRIPTION
<code>deploymentFilePath</code> Deployment file	(Required) Select the deployment JSON file. If this task is in a release pipeline, you must specify the location of the deployment file within the artifacts (the default value works for most conditions). If this task is in a build pipeline, you must specify the Path of output deployment file. Default value: \$(System.DefaultWorkingDirectory)/*.json.
<code>connectedServiceNameARM</code> Azure subscription contains IoT Hub	(Required) Select an Azure subscription that contains an IoT Hub Argument aliases: <code>azureSubscription</code>
<code>iothubname</code> IoT Hub name	(Required) Select the IoT Hub
<code>deviceOption</code> Choose single/multiple device	(Required) Choose to deploy to a single device, or to multiple devices specified by using tags.
<code>deploymentId</code> IoT Edge deployment ID	(Required) Enter the IoT Edge Deployment ID. If an ID already exists, it will be overridden. Up to 128 lowercase letters, numbers, and the characters <code>- : + % _ # * ? ! ( ) , = @ ;</code> . <a href="#">More details.</a> Default value: \$(System.TeamProject)-devops-deployment.
<code>priority</code> IoT Edge deployment priority	(Required) A positive integer used to resolve deployment conflicts. When a device is targeted by multiple deployments it will use the one with highest priority or, in the case of two deployments with the same priority, the one with the latest creation time. Default value: 0.
<code>targetcondition</code> IoT Edge device target condition	(Required) Specify the target condition of the devices to which you want to deploy. For example, <code>tags.building=9</code> and <code>tags.environment='test'</code> . Do not include double quotes. <a href="#">More details.</a>
<code>deviceId</code> IoT Edge device ID	(Required) Specify the IoT Edge Device ID.

The following YAML example deploys module images:

```
steps:
- task: AzureIoTEdge@2
 displayName: 'Azure IoT Edge - Deploy to IoT Edge devices'
 inputs:
 action: 'Deploy to IoT Edge devices'
 deploymentFilePath: deployment.template.json
 azureSubscription: $(azureSubscriptionEndpoint)
 iothubname: iothubname
 deviceOption: 'Single Device'
 deviceId: deviceId
```

# CMake task

6/2/2020 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

Use this task to build with the CMake cross-platform build system.

## Demands

cmake

### IMPORTANT

The [Microsoft-hosted agents](#) have CMake installed by default so you don't need to include a demand for CMake in your `azure-pipelines.yml` file. If you do include a demand for CMake you may receive an error. To resolve, remove the demand.

## YAML snippet

```
CMake
Build with the CMake cross-platform build system
- task: CMake@1
 inputs:
 #workingDirectory: 'build' # Optional
 #cmakeArgs: # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<code>cwd</code> Working Directory	(Optional) 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 <a href="#">variables</a> . For example: <code>\$(Build.ArtifactStagingDirectory)\build</code>  If the path you specify does not exist, CMake creates it.  Default value: <code>build</code> Argument aliases: <code>workingDirectory</code>
<code>cmakeArgs</code> Arguments	(Optional) Arguments that you want to pass to CMake.
<b>CONTROL OPTIONS</b>	

# Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

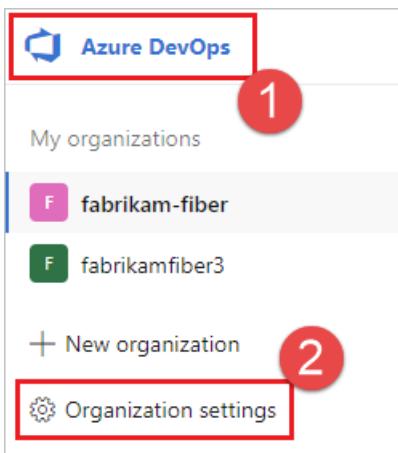
## FAQ

### How do I enable CMake for Microsoft-hosted agents?

The [Microsoft-hosted agents](#) have CMake installed already so you don't need to do anything. You do not need to add a demand for CMake in your `azure-pipelines.yml` file.

### How do I enable CMake for my on-premises agent?

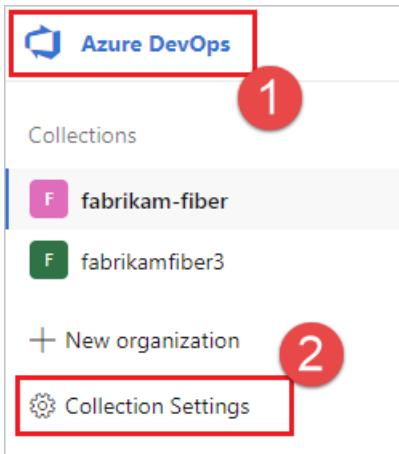
1. [Deploy an agent](#).
2. [Install CMake](#) and make sure to add it to the path of the user that the agent is running as on your agent machine.
3. In your web browser, navigate to Agent pools:
  1. Choose [Azure DevOps, Organization settings](#).



2. Choose [Agent pools](#).

A screenshot of the Azure DevOps Agent pools interface. At the top left is the 'Azure DevOps' logo. The top navigation bar shows 'FabrikamFiber / Organization Settings / Agent pools'. On the left is a sidebar with links: 'Security', 'Policies', 'Permissions', 'Boards', 'Process', 'Pipelines', 'Agent pools' (which is highlighted with a red box and a red circle containing the number '1'), 'Settings', 'Deployment pools', and 'Parallel jobs'. The main content area is titled 'Agent pools' and lists four agent pools: 'Azure Pipelines' (with a cloud icon), 'Default' (with a laptop icon), 'Test' (with a laptop icon), and 'Test2' (with a laptop icon). A red circle containing the number '2' points to the 'Agent pools' list.

1. Choose [Azure DevOps, Collection settings](#).



2. Choose Agent pools.

Azure DevOps

FabrikamFiber / Organization Settings / Agent pools

Security

Policies

Permissions

Boards

Process

Pipelines

Agent pools

Settings

Deployment pools

Parallel jobs

## Agent pools

Name	Type
Azure Pipelines	Azure Pipelines
Default	Hosted
Test	Hosted
Test2	Hosted

1. Choose Azure DevOps, Collection settings.

Azure DevOps

Collections

fabrikam-fiber

fabrikamfiber3

+ New organization

Collection Settings

2. Choose Agent pools.

Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards  
Process

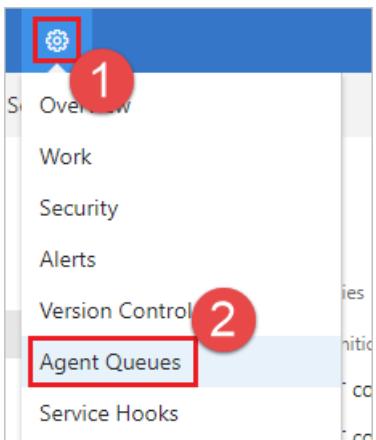
Pipelines

Agent pools **(Red Box)**  
Settings  
Deployment pools  
Parallel jobs

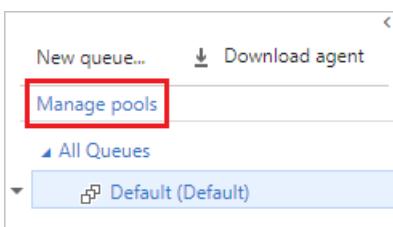
Agent pools

Name
Azure Pipelines
Default
Test
Test2

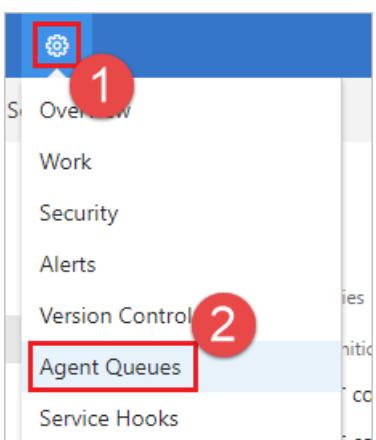
1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



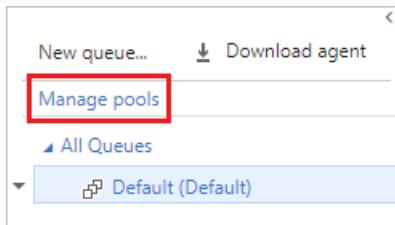
2. Choose **Manage pools**.



1. Navigate to your project and choose **Settings** (gear icon) > **Agent Queues**.



2. Choose **Manage pools**.



1. Navigate to your project and choose **Manage project** (gear icon).



2. Choose **Control panel**.



3. Select **Agent pools**.



4. Navigate to the capabilities tab:

1. From the **Agent pools** tab, select the desired agent pool.

A screenshot of the Azure DevOps interface showing the 'Agent pools' tab selected (highlighted with a red box, labeled '1'). On the left, a sidebar lists several tabs: Security, Boards, Process, Pipelines, Agent pools (selected), Settings, Deployment pools, and Parallel jobs. In the main area, the title is 'Agent pools'. A table lists agent pools with columns 'Name' and 'Status'. The 'Default' pool is highlighted with a red box and labeled '2'. Other entries include 'Azure Pipelines' and 'Test'.

2. Select **Agents** and choose the desired agent.

The screenshot shows the 'Default' pool settings. The 'Agents' tab is selected, indicated by a red circle labeled '1'. Below it, the first agent entry, '160724-AT14-SP3', is highlighted with a red box labeled '2'. This agent is marked as 'Offline'. Another agent entry, '20160317-W10', is also listed below it.

3. Choose the **Capabilities** tab.

The screenshot shows the 'Capabilities' tab for the agent '160724-AT14-SP3'. The 'Capabilities' tab is selected, indicated by a red box. The 'User-defined capabilities' section has a '+ Add new capability' button. The 'System capabilities' section lists the following entries:

Name	Value
Agent.Name	160724-AT14-SP3
Agent.Version	2.107.0
Agent.ComputerName	160724-AT14-SP3

**NOTE**

Microsoft-hosted agents don't display system capabilities. For a list of software installed on Microsoft-hosted agents, see [Use a Microsoft-hosted agent](#).

1. From the **Agent pools** tab, select the desired pool.

Azure DevOps      FabrikamFiber / Organization Settings / Agent pools

Security  
Policies  
Permissions

Boards

Process

Pipelines  
1

Agent pools  
2

Settings

Deployment pools

Parallel jobs

## Agent pools

Name
Azure Pipelines
Default
Test
Test2

2. Select **Agents** and choose the desired agent.

Default

Jobs Agents Details Security Settings  
1

Name

160724-AT14-SP3 ● Offline
20160317-W10 ● Offline

2

3. Choose the **Capabilities** tab.

160724-AT14-SP3

Jobs Capabilities

User-defined capabilities



No user-defined capabilities

Add a new capability

System capabilities

Search by keyword

Name	Value
Agent.Name	160724-AT14-SP3
Agent.Version	2.107.0
Agent.ComputerName	160724-AT14-SP3

- From the Agent pools tab, select the desired pool.

The screenshot shows the Azure DevOps interface for managing agent pools. On the left, there's a sidebar with navigation links: Security, Policies, Permissions, Boards, Process, Pipelines (with a red box around it and a red number 1), Agent pools (highlighted with a red box and a red number 1), Settings, Deployment pools, and Parallel jobs. The main content area is titled "Agent pools". It lists three pools: "Azure Pipelines" (highlighted with a red box and a red number 2), "Default" (highlighted with a red box and a red number 2), and "Test".

- Select Agents and choose the desired agent.

Default

Jobs Agents Details Security Settings

Name

160724-AT14-SP3  
● Offline

20160317-W10  
● Offline

3. Choose the **Capabilities** tab.

160724-AT14-SP3

Jobs Capabilities

User-defined capabilities +

No user-defined capabilities

Add a new capability

System capabilities Search by keyword

Name	Value
Agent.Name	160724-AT14-SP3
Agent.Version	2.107.0
Agent.ComputerName	160724-AT14-SP3

Select the desired agent, and choose the **Capabilities** tab.

Agents for pool Default

Agents Roles

Enabled	Name	Current Status	X
<input checked="" type="checkbox"/>	20160317-W10	Idle	X
<input checked="" type="checkbox"/>	DEV15-TFS-RTM	Idle	X
<input checked="" type="checkbox"/>	DEV15VS	Idle	X

Requests Capabilities

**USER CAPABILITIES**  
Shows information about user-defined capabilities supported by this host

+ Add capability

Save changes Undo changes

**SYSTEM CAPABILITIES**  
Shows information about the capabilities provided by this host

AegisRoot	C:\AegisTools
Agent.ComputerName	20160317-W10

Select the desired agent, and choose the **Capabilities** tab.

Agents for pool Default

Enabled	Name	Current Status	X
<input checked="" type="checkbox"/>	20160317-W10	Idle	X
<input checked="" type="checkbox"/>	DEV15-TFS-RTM	Idle	X
<input checked="" type="checkbox"/>	DEV15VS	Idle	X

Requests Capabilities **2**

**USER CAPABILITIES**  
Shows information about user-defined capabilities supported by this host

+ Add capability

Save changes Undo changes

**SYSTEM CAPABILITIES**  
Shows information about the capabilities provided by this host

AegisRoot	C:\AegisTools
Agent.ComputerName	20160317-W10

From the Agent pools tab, select the desired agent, and choose the Capabilities tab.

Control panel

Control panel Legacy Extensions **1** Agent pools

New pool... Download agent

All Pools **2** Default

Enabled	Name	Current Status	X
<input checked="" type="checkbox"/>	Agent-TFSDEV14UP...	Idle	X

Requests Capabilities **4**

**USER CAPABILITIES**  
Shows information about user-defined capabilities supported by this host

+ Add capability

Save changes Undo changes

**SYSTEM CAPABILITIES**  
Shows information about the capabilities provided by this host

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.

#### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

#### 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](#).

#### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

#### 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

3/6/2021 • 4 minutes to read • [Edit Online](#)

Use this task to build and push Docker images to any container registry using Docker registry service connection.

## Overview

Following are the key benefits of using Docker task as compared to directly using docker client binary in script -

- **Integration with Docker registry service connection** - The task makes it easy to use a Docker registry service connection for connecting to any container registry. Once logged in, the user can author follow up tasks to execute any tasks/scripts by leveraging the login already done by the Docker task. For example, you can use the Docker task to sign in to any Azure Container Registry and then use a subsequent task/script to build and push an image to this registry.
- **Metadata added as labels** - The task adds traceability-related metadata to the image in the form of the following labels -
  - com.azure.dev.image.build.buildnumber
  - com.azure.dev.image.build.builduri
  - com.azure.dev.image.build.definitionname
  - com.azure.dev.image.build.repository.name
  - com.azure.dev.image.build.repository.uri
  - com.azure.dev.image.build.sourcebranchname
  - com.azure.dev.image.build.sourceversion
  - com.azure.dev.image.release.definitionname
  - com.azure.dev.image.release.releaseid
  - com.azure.dev.image.release.releaseweburl
  - com.azure.dev.image.system.teamfoundationcollectionuri
  - com.azure.dev.image.system.teamproject

## Task Inputs

PARAMETERS	DESCRIPTION
<code>command</code> Command	(Required) Possible values: <code>buildAndPush</code> , <code>build</code> , <code>push</code> , <code>login</code> , <code>logout</code> Added in version 2.173.0: <code>start</code> , <code>stop</code> Default value: <code>buildAndPush</code>
<code>containerRegistry</code> Container registry	(Optional) Name of the <a href="#">Docker registry service connection</a>
<code>repository</code> Repository	(Optional) Name of repository within the container registry corresponding to the Docker registry service connection specified as input for <code>containerRegistry</code>

PARAMETERS	DESCRIPTION
<code>container</code> Container	(Required for commands <code>start</code> and <code>stop</code> ) The container resource to start or stop
<code>tags</code> Tags	(Optional) Multiline input where each line contains a tag to be used in <code>build</code> , <code>push</code> or <code>buildAndPush</code> commands Default value: <code>\$(Build.BuildId)</code>
<code>Dockerfile</code> Dockerfile	(Optional) Path to the Dockerfile. The task will use the <b>first</b> dockerfile it finds to build the image. Default value: <code>**/Dockerfile</code>
<code>buildContext</code> Build context	(Optional) Path to the build context Default value: <code>**</code>
<code>arguments</code> Arguments	(Optional) Additional arguments to be passed onto the docker client Be aware that if you use value <code>buildAndPush</code> for the <code>command</code> parameter, then the <code>arguments</code> property will be ignored.
<code>addPipelineData</code> Add Pipeline Data	(Optional) Adds the above mentioned metadata as labels to the image Possible values: <code>true</code> , <code>false</code> Default value: <code>true</code>

## Login

Following YAML snippet showcases container registry login using a Docker registry service connection -

```
- task: Docker@2
displayName: Login to ACR
inputs:
 command: login
 containerRegistry: dockerRegistryServiceConnection1
```

## Build and Push

A convenience command called `buildAndPush` allows for build and push of images to container registry in a single command. The following YAML snippet is an example of building and pushing multiple tags of an image to multiple registries -

```

steps:
- task: Docker@2
 displayName: Login to ACR
 inputs:
 command: login
 containerRegistry: dockerRegistryServiceConnection1
- task: Docker@2
 displayName: Login to Docker Hub
 inputs:
 command: login
 containerRegistry: dockerRegistryServiceConnection2
- task: Docker@2
 displayName: Build and Push
 inputs:
 command: buildAndPush
 repository: contosoRepository
 tags: |
 tag1
 tag2

```

In the above snippet, the images `contosoRepository:tag1` and `contosoRepository:tag2` are built and pushed to the container registries corresponding to `dockerRegistryServiceConnection1` and `dockerRegistryServiceConnection2`.

If one wants to build and push to a specific authenticated container registry instead of building and pushing to all authenticated container registries at once, the `containerRegistry` input can be explicitly specified along with `command: buildAndPush` as shown below -

```

steps:
- task: Docker@2
 displayName: Build and Push
 inputs:
 command: buildAndPush
 containerRegistry: dockerRegistryServiceConnection1
 repository: contosoRepository
 tags: |
 tag1
 tag2

```

## Logout

Following YAML snippet showcases container registry logout using a Docker registry service connection -

```

- task: Docker@2
 displayName: Logout of ACR
 inputs:
 command: logout
 containerRegistry: dockerRegistryServiceConnection1

```

## Start/stop

This task can also be used to control job and service containers. This usage is uncommon, but occasionally used for unique circumstances.

```

resources:
 containers:
 - container: builder
 image: ubuntu:18.04
 steps:
 - script: echo "I can run inside the container (it starts by default)"
 target:
 container: builder
 - task: Docker@2
 inputs:
 command: stop
 container: builder
 # any task beyond this point would not be able to target the builder container
 # because it's been stopped

```

## Other commands and arguments

The command and argument inputs can be used to pass additional arguments for build or push commands using docker client binary as shown below -

```

steps:
- task: Docker@2
 displayName: Login to ACR
 inputs:
 command: login
 containerRegistry: dockerRegistryServiceConnection1
- task: Docker@2
 displayName: Build
 inputs:
 command: build
 repository: contosoRepository
 tags: tag1
 arguments: --secret id=mysecret,src=mysecret.txt

```

### NOTE

The arguments input is evaluated for all commands except `buildAndPush`. As `buildAndPush` is a convenience command (`build` followed by `push`), `arguments` input is ignored for this command.

## Troubleshooting

### Why does Docker task ignore arguments passed to buildAndPush command?

Docker task configured with `buildAndPush` command ignores the arguments passed since they become ambiguous to the build and push commands that are run internally. You can split your command into separate build and push steps and pass the suitable arguments. See this [stackoverflow post](#) for example.

### DockerV2 only supports Docker registry service connection and not support ARM service connection. How can I use an existing Azure service principal (SPN) for authentication in Docker task?

You can create a Docker registry service connection using your Azure SPN credentials. Choose the Others from Registry type and provide the details as follows:

Docker Registry:	Your container registry URL (eg. <a href="https://myacr.azurecr.io">https://myacr.azurecr.io</a> )
Docker ID:	Service principal client ID
Password:	Service principal key

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Docker Compose task

5/18/2021 • 11 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task to build, push or run multi-container Docker applications. This task can be used with a Docker registry or an Azure Container Registry.

## Container registry types

### Azure Container Registry

PARAMETERS	DESCRIPTION
<code>containerregistrytype</code> (Container registry type)	(Optional) Azure Container Registry if using ACR or Container Registry if using any other container registry. Default value: Azure Container Registry
<code>azureSubscriptionEndpoint</code> (Azure subscription)	(Required) Name of the Azure Service Connection. See <a href="#">Azure Resource Manager service connection</a> to manually set up the connection. Argument aliases: <code>azureSubscription</code>
<code>azureContainerRegistry</code> (Azure Container Registry)	(Required) Name of the Azure Container Registry. Example: <code>Contoso.azurecr.io</code>

This YAML example specifies the inputs for Azure Container Registry:

```
variables:
 azureContainerRegistry: Contoso.azurecr.io
 azureSubscriptionEndpoint: Contoso
steps:
- task: DockerCompose@0
 displayName: Container registry login
 inputs:
 containerregistrytype: Azure Container Registry
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureContainerRegistry: $(azureContainerRegistry)
```

### Other container registries

The `containerregistrytype` value is required when using any container registry other than ACR. Use

`containerregistrytype: Container Registry` in this case.

PARAMETERS	DESCRIPTION
<code>containerregistrytype</code> (Container registry type)	(Required) Azure Container Registry if using ACR or Container Registry if using any other container registry. Default value: Azure Container Registry
<code>dockerRegistryEndpoint</code> (Docker registry service connection)	(Required) <a href="#">Docker registry service connection</a> .

This YAML example specifies a container registry other than ACR where `Contoso` is the name of the Docker

registry service connection for the container registry:

```
- task: DockerCompose@0
displayName: Container registry login
inputs:
 containerregistrytype: Container Registry
 dockerRegistryEndpoint: Contoso
```

## Build service images

PARAMETERS	DESCRIPTION
<code>containerregistrytype</code> (Container Registry Type)	(Required) Azure Container Registry if using ACR or Container Registry if using any other container registry. Default value: Azure Container Registry
<code>azureSubscriptionEndpoint</code> (Azure subscription)	(Required) Name of the Azure Service Connection.
<code>azureContainerRegistry</code> (Azure Container Registry)	(Required) Name of the Azure Container Registry.
<code>dockerComposeFile</code> (Docker Compose File)	(Required) Path to the primary Docker Compose file to use. Default value: **/docker-compose.yml
<code>additionalDockerComposeFiles</code> (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.
<code>dockerComposeFileArgs</code> (Environment Variables)	(Optional) Environment variables to be set up during the command. Specify each name = value pair on a new line. You need to use the   operator in YAML to indicate that newlines should be preserved. Example: <code>dockerComposeFileArgs:   DOCKER_BUILD_SOURCE=\$(CustomVar)</code>
<code>projectName</code> (Project Name)	(Optional) Project name used for default naming of images and containers. Default value: \$(Build.Repository.Name)
<code>qualifyImageNames</code> (Qualify Image Names)	(Optional) Qualify image names for built services with the Docker registry service connection's hostname if not otherwise specified. Default value: true
<code>action</code> (Action)	(Required) Select a Docker Compose action. Default value: Run a Docker Compose command
<code>additionalImageTags</code> (Additional Image Tags)	(Optional) Additional tags for the Docker images being built or pushed.

PARAMETERS	DESCRIPTION
<code>includeSourceTags</code> (Include Source Tags)	(Optional) Include Git tags when building or pushing Docker images. Default value: false
<code>includeLatestTag</code> (Include Latest Tag)	(Optional) Include the latest tag when building or pushing Docker images. Default value: false

This YAML example builds the image where the image name is qualified on the basis of the inputs related to Azure Container Registry:

```
- task: DockerCompose@0
 displayName: Build services
 inputs:
 action: Build services
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureContainerRegistry: $(azureContainerRegistry)
 dockerComposeFile: docker-compose.yml
 projectName: $(Build.Repository.Name)
 qualifyImageNames: true
 additionalImageTags: $(Build.BuildId)
 dockerComposeFileArgs: |
 firstArg=$(firstArg)
 secondArg=$(secondArg)
```

## Push service images

PARAMETERS	DESCRIPTION
<code>containerregistrytype</code> (Container Registry Type)	(Required) Azure Container Registry if using ACR or Container Registry if using any other container registry. Default value: Azure Container Registry
<code>azureSubscriptionEndpoint</code> (Azure subscription)	(Required) Name of the Azure Service Connection.
<code>azureContainerRegistry</code> (Azure Container Registry)	(Required) Name of the Azure Container Registry.
<code>dockerComposeFile</code> (Docker Compose File)	(Required) Path to the primary Docker Compose file to use. Default value: **/docker-compose.yml
<code>additionalDockerComposeFiles</code> (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.
<code>dockerComposeFileArgs</code> (Environment Variables)	(Optional) Environment variables to be set up during the command. Specify each name=value pair on a new line.

PARAMETERS	DESCRIPTION
<code>projectName</code> (Project Name)	(Optional) Project name used for default naming of images and containers. Default value: \$(Build.Repository.Name)
<code>qualifyImageNames</code> (Qualify Image Names)	(Optional) Qualify image names for built services with the Docker registry service connection's hostname if not otherwise specified. Default value: true
<code>action</code> (Action)	(Required) Select a Docker Compose action. Default value: Run a Docker Compose command
<code>additionalImageTags</code> (Additional Image Tags)	(Optional) Additional tags for the Docker images being built or pushed.
<code>includeSourceTags</code> (Include Source Tags)	(Optional) Include Git tags when building or pushing Docker images. Default value: false
<code>includeLatestTag</code> (Include Latest Tag)	(Optional) Include the latest tag when building or pushing Docker images. Default value: false

This YAML example pushes an image to a container registry:

```
- task: DockerCompose@0
displayName: Push services
inputs:
 action: Push services
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureContainerRegistry: $(azureContainerRegistry)
 dockerComposeFile: docker-compose.yml
 projectName: $(Build.Repository.Name)
 qualifyImageNames: true
 additionalImageTags: $(Build.BuildId)
```

## Run service images

PARAMETERS	DESCRIPTION
<code>dockerComposeFile</code> (Docker Compose File)	(Required) Path to the primary Docker Compose file to use. Default value: **/docker-compose.yml
<code>additionalDockerComposeFiles</code> (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.
<code>dockerComposeFileArgs</code> (Environment Variables)	(Optional) Environment variables to be set up during the command. Specify each name=value pair on a new line.

PARAMETERS	DESCRIPTION
<code>projectName</code> (Project Name)	(Optional) Project name used for default naming of images and containers. Default value: \$(Build.Repository.Name)
<code>qualifyImageNames</code> (Qualify Image Names)	(Optional) Qualify image names for built services with the Docker registry service connection's hostname if not otherwise specified. Default value: true
<code>action</code> (Action)	(Required) Select a Docker Compose action. Default value: Run a Docker Compose command
<code>buildImages</code> (Build Images)	(Optional) Build images before starting service containers. Default value: true
<code>detached</code> (Run in Background)	(Optional) Run the service containers in the background. Default value: true

This YAML example runs services:

```
- task: DockerCompose@0
 displayName: Run services
 inputs:
 action: Run services
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureContainerRegistry: $(azureContainerRegistry)
 dockerComposeFile: docker-compose.ci.build.yml
 projectName: $(Build.Repository.Name)
 qualifyImageNames: true
 buildImages: true
 abortOnContainerExit: true
 detached: false
```

## Run a specific service image

PARAMETERS	DESCRIPTION
<code>dockerComposeFile</code> (Docker Compose File)	(Required) Path to the primary Docker Compose file to use. Default value: **/docker-compose.yml
<code>additionalDockerComposeFiles</code> (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.
<code>dockerComposeFileArgs</code> (Environment Variables)	(Optional) Environment variables to be set up during the command. Specify each name=value pair on a new line.
<code>projectName</code> (Project Name)	(Optional) Project name used for default naming of images and containers. Default value: \$(Build.Repository.Name)

PARAMETERS	DESCRIPTION
<code>qualifyImageNames</code> (Qualify Image Names)	(Optional) Qualify image names for built services with the Docker registry service connection's hostname if not otherwise specified. Default value: true
<code>action</code> (Action)	(Required) Select a Docker Compose action. Default value: Run a Docker Compose command
<code>serviceName</code> (Service Name)	(Required) Name of the specific service to run.
<code>containerName</code> (Container Name)	(Optional) Name of the specific service container to run.
<code>ports</code> (Ports)	(Optional) Ports in the specific service container to publish to the host. Specify each host-port:container-port binding on a new line.
<code>workDir</code> (Working Directory)	(Optional) The working directory for the specific service container. Argument aliases: <code>workingDirectory</code>
<code>entrypoint</code> (Entry Point Override)	(Optional) Override the default entry point for the specific service container.
<code>containerCommand</code> (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 <code>python app.py</code> to launch the web application.
<code>detached</code> (Run in Background)	(Optional) Run the service containers in the background. Default value: true

This YAML example runs a specific service:

```
- task: DockerCompose@0
 displayName: Run a specific service
 inputs:
 action: Run a specific service
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureContainerRegistry: $(azureContainerRegistry)
 dockerComposeFile: docker-compose.yml
 projectName: $(Build.Repository.Name)
 qualifyImageNames: true
 serviceName: myhealth.web
 ports: 80:80
 detached: true
```

## Lock service images

PARAMETERS	DESCRIPTION
<code>dockerComposeFile</code> (Docker Compose File)	(Required) Path to the primary Docker Compose file to use. Default value: <code>**/docker-compose.yml</code>

PARAMETERS	DESCRIPTION
<code>additionalDockerComposeFiles</code> (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.
<code>dockerComposeFileArgs</code> (Environment Variables)	(Optional) Environment variables to be set up during the command. Specify each name=value pair on a new line.
<code>projectName</code> (Project Name)	(Optional) Project name used for default naming of images and containers. Default value: \$(Build.Repository.Name)
<code>qualifyImageNames</code> (Qualify Image Names)	(Optional) Qualify image names for built services with the Docker registry service connection's hostname if not otherwise specified. Default value: true
<code>action</code> (Action)	(Required) Select a Docker Compose action. Default value: Run a Docker Compose command
<code>removeBuildOptions</code> (Remove Build Options)	(Optional) Remove the build options from the output Docker Compose file. Default value: false
<code>baseResolveDirectory</code> (Base Resolve Directory)	(Optional) The base directory from which relative paths in the output Docker Compose file should be resolved.
<code>outputDockerComposeFile</code> (Output Docker Compose File)	(Required) Path to an output Docker Compose file. Default value: \$(Build.StagingDirectory)/docker-compose.yml

This YAML example locks services:

```
- task: DockerCompose@0
 displayName: Lock services
 inputs:
 action: Lock services
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureContainerRegistry: $(azureContainerRegistry)
 dockerComposeFile: docker-compose.yml
 projectName: $(Build.Repository.Name)
 qualifyImageNames: true
 outputDockerComposeFile: $(Build.StagingDirectory)/docker-compose.yml
```

## Write service image digests

PARAMETERS	DESCRIPTION
<code>dockerComposeFile</code> (Docker Compose File)	(Required) Path to the primary Docker Compose file to use. Default value: **/docker-compose.yml

PARAMETERS	DESCRIPTION
<code>additionalDockerComposeFiles</code> (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.
<code>dockerComposeFileArgs</code> (Environment Variables)	(Optional) Environment variables to be set up during the command. Specify each name=value pair on a new line.
<code>projectName</code> (Project Name)	(Optional) Project name used for default naming of images and containers. Default value: \$(Build.Repository.Name)
<code>qualifyImageNames</code> (Qualify Image Names)	(Optional) Qualify image names for built services with the Docker registry service connection's hostname if not otherwise specified. Default value: true
<code>action</code> (Action)	(Required) Select a Docker Compose action. Default value: Run a Docker Compose command
<code>imageDigestComposeFile</code> (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. Default value: \$(Build.StagingDirectory)/docker-compose.images.yml

This YAML example writes service image digests:

```
- task: DockerCompose@0
 displayName: Write service image digests
 inputs:
 action: Write service image digests
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureContainerRegistry: $(azureContainerRegistry)
 dockerComposeFile: docker-compose.yml
 projectName: $(Build.Repository.Name)
 qualifyImageNames: true
 imageDigestComposeFile: $(Build.StagingDirectory)/docker-compose.images.yml
```

## Combine configuration

PARAMETERS	DESCRIPTION
<code>dockerComposeFile</code> (Docker Compose File)	(Required) Path to the primary Docker Compose file to use. Default value: **/docker-compose.yml
<code>additionalDockerComposeFiles</code> (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.
<code>dockerComposeFileArgs</code> (Environment Variables)	(Optional) Environment variables to be set up during the command. Specify each name=value pair on a new line

PARAMETERS	DESCRIPTION
<code>projectName</code> (Project Name)	(Optional) Project name used for default naming of images and containers. Default value: \$(Build.Repository.Name)
<code>qualifyImageNames</code> (Qualify Image Names)	(Optional) Qualify image names for built services with the Docker registry service connection's hostname if not otherwise specified. Default value: true
<code>action</code> (Action)	(Required) Select a Docker Compose action. Default value: Run a Docker Compose command
<code>removeBuildOptions</code> (Remove Build Options)	(Optional) Remove the build options from the output Docker Compose file. Default value: false
<code>baseResolveDirectory</code> (Base Resolve Directory)	(Optional) The base directory from which relative paths in the output Docker Compose file should be resolved.
<code>outputDockerComposeFile</code> (Output Docker Compose File)	(Required) Path to an output Docker Compose file. Default value: \$(Build.StagingDirectory)/docker-compose.yml

This YAML example combines configurations:

```
- task: DockerCompose@0
 displayName: Combine configuration
 inputs:
 action: Combine configuration
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureContainerRegistry: $(azureContainerRegistry)
 dockerComposeFile: docker-compose.yml
 additionalDockerComposeFiles: docker-compose.override.yml
 projectName: $(Build.Repository.Name)
 qualifyImageNames: true
 outputDockerComposeFile: $(Build.StagingDirectory)/docker-compose.yml
```

## Run a Docker Compose command

PARAMETERS	DESCRIPTION
<code>dockerComposeFile</code> (Docker Compose File)	(Required) Path to the primary Docker Compose file to use. Default value: **/docker-compose.yml
<code>additionalDockerComposeFiles</code> (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.
<code>dockerComposeFileArgs</code> (Environment Variables)	(Optional) Environment variables to be set up during the command. Specify each name=value pair on a new line.

PARAMETERS	DESCRIPTION
<code>projectName</code> (Project Name)	(Optional) Project name used for default naming of images and containers. Default value: \$(Build.Repository.Name)
<code>qualifyImageNames</code> (Qualify Image Names)	(Optional) Qualify image names for built services with the Docker registry service connection's hostname if not otherwise specified. Default value: true
<code>action</code> (Action)	(Required) Select a Docker Compose action. Default value: Run a Docker Compose command
<code>dockerComposeCommand</code> (Command)	(Required) Docker Compose command to execute with the help of arguments. For example, rm to remove all stopped service containers.

This YAML example runs a docker Compose command:

```
- task: DockerCompose@0
 displayName: Run a Docker Compose command
 inputs:
 action: Run a Docker Compose command
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureContainerRegistry: $(azureContainerRegistry)
 dockerComposeFile: docker-compose.yml
 projectName: $(Build.Repository.Name)
 qualifyImageNames: true
 dockerComposeCommand: rm
```

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

# Go task

6/2/2020 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task 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
<code>command</code> Command	(Required) Select a Go command to run. Select 'Custom' to use a command not listed here. Default value: get
<code>customCommand</code> Custom command	(Required) Custom Go command for execution. For example: to execute go version, enter version.
<code>arguments</code> Arguments	(Optional) Arguments to the selected command. For example, build time arguments for go build command.
<code>workingDirectory</code> 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)

## Example

```
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:
- task: GoTool@0
 displayName: 'Use Go 1.10'

- task: Go@0
 displayName: 'go get'
 inputs:
 arguments: '-d'

- task: Go@0
 displayName: 'go build'
 inputs:
 command: build
 arguments: '-o "$(System.TeamProject).exe"'

- task: ArchiveFiles@2
 displayName: 'Archive files'
 inputs:
 rootFolderOrFile: '$(Build.Repository.LocalPath)'
 includeRootFolder: False

- task: PublishBuildArtifacts@1
 displayName: 'Publish artifact'
 condition: succeededOrFailed()
```

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

# Gradle task

5/26/2021 • 6 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

Use this task to build using a Gradle wrapper script.

## YAML snippet

```
Gradle
Build using a Gradle wrapper script
- task: Gradle@2
 inputs:
 #gradleWrapperFile: 'gradlew'
 #cwd: # Optional
 #options: # Optional
 #tasks: 'build' # A list of tasks separated by spaces, such as 'build test'
 #publishJUnitResults: true
 #testResultsFiles: '**/TEST-*.xml' # Required when publishJUnitResults == True
 #testRunTitle: # Optional
 #codeCoverageToolOption: 'None' # Optional. Options: none, cobertura, jaCoCo
 #codeCoverageClassFilesDirectories: 'build/classes/main/' # Required when codeCoverageToolOption ==
False
 #codeCoverageClassFilter: # Optional. Comma-separated list of filters to include or exclude classes from
collecting code coverage. For example: +:com.*,:org.*,-:my.app.*
 #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

Default value: true

ARGUMENT	DESCRIPTION

ARGUMENT	DESCRIPTION
<code>wrapperScript</code> Gradle Wrapper	(Required) 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 <a href="#">The Gradle Wrapper</a> .
	Default value: <code>gradlew</code> Argument aliases: <code>gradleWrapperFile</code>
<code>options</code> Options	(Optional) Specify any command line options you want to pass to the Gradle wrapper.  See <a href="#">Gradle Command Line</a> .
<code>tasks</code> Tasks	(Required) The task(s) for Gradle to execute. A list of task names should be separated by spaces and can be taken from <code>gradlew tasks</code> issued from a command prompt.  See <a href="#">Gradle Build Script Basics</a> .
Default value: <code>build</code>	
JUNIT TEST RESULTS	
<code>publishJUnitResults</code> Publish to Azure Pipelines	(Required) Select this option to publish JUnit Test results produced by the Gradle build to Azure Pipelines/TFS.
<code>testResultsFiles</code> Test results files	(Required) Test results files path. Wildcards can be used. For example, <code>*/TEST-.xml</code> for all xml files whose name starts with TEST-."  Default value: <code>**/TEST-* .xml</code>
<code>testRunTitle</code> Test run title	(Optional) Assign a title for the JUnit test case results for this build.
CODE COVERAGE	
<code>codeCoverageTool</code> Code coverage tool	(Optional) Choose a code coverage tool to determine the code that is covered by the test cases for the build.  Default value: None Argument aliases: <code>codeCoverageToolOption</code>
<code>classFilesDirectories</code> Class files directories	(Required) Comma-separated list of directories containing class files and archive files (JAR, WAR, etc.). Code coverage is reported for class files in these directories. Normally, classes under `build/classes/main` are searched, which is the default class directory for Gradle builds  Default value: <code>build/classes/main/</code> Argument aliases: <code>codeCoverageClassFilesDirectories</code>

ARGUMENT	DESCRIPTION
<code>classFilter</code> Class inclusion/exclusion filters	(Optional) Comma-separated list of filters to include or exclude classes from collecting code coverage. For example: <code>+:com.*,:org.*,-:my.app*.*.</code> Argument aliases: <code>codeCoverageClassFilter</code>
<code>failIfCoverageEmpty</code> Fail when code coverage results are missing	(Optional) Fail the build if code coverage did not produce any results to publish Default value: false Argument aliases: <code>codeCoverageFailIfEmpty</code>
<b>ADVANCED</b>	
<code>cwd</code> Working directory	(Optional) Working directory in which to run the Gradle build. If not specified, the repository root directory is used
<code>javaHomeSelection</code> Set JAVA_HOME by	(Required) Sets JAVA_HOME either by selecting a JDK version that will be discovered during builds or by manually entering a JDK path Default value: <code>JDKVersion</code> Argument aliases: <code>javaHomeOption</code>
<code>jdkVersion</code> JDK version	(Optional) Will attempt to discover the path to the selected JDK version and set JAVA_HOME accordingly Argument aliases: <code>jdkDirectory</code>
<code>jdkUserInputPath</code> JDK path	(Required) Sets JAVA_HOME to the given path Default value: <code>default</code> Argument aliases: <code>jdkVersionOption</code>
<code>jdkArchitecture</code> JDK Architecture	(Optional) Optionally supply the architecture (x86, x64) of JDK. Default value: <code>x64</code> Argument aliases: <code>jdkArchitectureOption</code>
<code>gradleOpts</code> Set GRADLE_OPTS	(Optional) Sets the GRADLE_OPTS environment variable, which is used to send command-line arguments to start the JVM. The xmx flag specifies the maximum memory available to the JVM. Default value: <code>-Xmx1024m</code> Argument aliases: <code>gradleOptions</code>
<b>CODE ANALYSIS</b>	
<code>sqAnalysisEnabled</code> Run SonarQube or SonarCloud Analysis	(Required) This option has changed from version 1 of the <a href="#">Gradle</a> task to use the <a href="#">SonarQube</a> and <a href="#">SonarCloud</a> marketplace extensions. Enable this option to run <a href="#">SonarQube</a> or <a href="#">SonarCloud analysis</a> after executing tasks in the <a href="#">Tasks</a> field. You must also add a <a href="#">Prepare Analysis Configuration</a> task from one of the extensions to the build pipeline before this Gradle task Default value: false Argument aliases: <code>sonarQubeRunAnalysis</code>

ARGUMENT	DESCRIPTION
<code>sqGradlePluginVersionChoice</code> SonarQube scanner for Gradle version	(Required) The SonarQube Gradle plugin version to use. You can declare it in your Gradle configuration file, or specify a version here Default value: specify
<code>sqGradlePluginVersion</code> SonarQube scanner for Gradle plugin version	(Required) <a href="#">Refer</a> for all available versions Default value: 2.6.1 Argument aliases: <code>sonarQubeGradlePluginVersion</code>
<code>checkstyleAnalysisEnabled</code> Run Checkstyle	(Optional) Run the Checkstyle tool with the default Sun checks. Results are uploaded as build artifacts. Default value: false Argument aliases: <code>checkStyleRunAnalysis</code>
<code>findbugsAnalysisEnabled</code> Run FindBugs	(Optional) Use the FindBugs static analysis tool to look for bugs in the code. Results are uploaded as build artifacts. In Gradle 6.0 this plugin was removed. Use spotbugs plugin instead. <a href="#">More info</a> . Default value: false Argument aliases: <code>findBugsRunAnalysis</code>
<code>pmdAnalysisEnabled</code> Run PMD	(Optional) Use the PMD Java static analysis tool to look for bugs in the code. Results are uploaded as build artifacts Default value: false Argument aliases: <code>pmdRunAnalysis</code>
<code>spotBugsAnalysisEnabled</code> Run Spotbugs	(Required) Enable this option to run spotBugs. This plugin works with Gradle v5.6 or later. Results are uploaded as build artifacts. <a href="#">More info</a> . Please make sure that you are using Gradle 5.6 or later. If you are using an earlier version of Gradle, the plugin may work in an unexpected way or may not work at all. Default value: false Argument aliases: <code>spotBugsAnalysis</code>
<code>spotBugsGradlePluginVersionChoice</code> Spotbugs plugin for Gradle version	(Required) The Spotbugs Gradle plugin version to use. You can declare it in your Gradle configuration file, or specify a version here. Default value: specify
<code>spotbugsGradlePluginVersion</code> Spotbugs for Gradle plugin version	(Required) <a href="#">Refer</a> for all available versions. Default value: 4.7.0 Argument aliases: <code>spotbugsGradlePluginVersion</code>
<b>CONTROL OPTIONS</b>	

## Example

[Build your Java app with Gradle](#)

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

## 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.

## I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

## 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](#).

## My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

## 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

6/2/2020 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015.3

Use this task to run Grunt tasks using the JavaScript Task Runner.

## Demands

The build agent must have the following capability:

- Grunt

## YAML snippet

```
Grunt
Run the Grunt 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
<code>gruntFile</code> Grunt File Path	(Required) Relative path from the repo root to the Grunt script that you want to run. Default value: gruntfile.js
<code>targets</code> Grunt task(s)	(Optional) Space delimited list of tasks to run. If you leave it blank, the default task will run.
<code>arguments</code> Arguments	Additional arguments passed to Grunt. See <a href="#">Using the CLI</a> . Tip: --gruntfile is not needed. This argument is handled by the Grunt file path argument shown above.
<code>cwd</code> Working Directory	(Optional) 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 aliases: <code>workingDirectory</code>

ARGUMENT	DESCRIPTION
<code>gruntCli</code> grunt-cli location	(Required) grunt-cli to run when agent can't find global installed grunt-cli. Defaults to the grunt-cli under node_modules folder of the working directory. Default value: node_modules/grunt-cli/bin/grunt Argument aliases: <code>workingDirectory</code>
<code>publishJUnitResults</code> Publish to Azure Pipelines	Select this option to publish JUnit test results produced by the Grunt build to Azure Pipelines Default value: false
<code>testResultsFiles</code> Test Results Files	(Required) Test results files path. Wildcards can be used. For example, <code>**/TEST-.xml</code> for all XML files whose name starts with TEST-. Default value: <code>**/TEST-.xml</code>
<code>testRunTitle</code> Test Run Title	(Optional) Provide a name for the test run
<code>enableCodeCoverage</code> Enable Code Coverage	(Optional) Select this option to enable Code Coverage using Istanbul Default value: false
<code>testFramework</code> Test Framework	(Optional) Select your test framework Default value: Mocha
<code>srcFiles</code> Source Files	(Optional) Provide the path to your source files which you want to hookRequire()
<code>testFiles</code> Test Script Files	(Required) Provide the path to your test script files Default value: <code>test/*.js</code>

## Example

See [Sample Gruntfile](#).

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### Do I need an agent?

You need at least one [agent](#) to run your build or release.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file`

environment variable to your build agent, or you can add the `NODE.EXTRA.CA.CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

**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

6/2/2020 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

Use this task to run gulp tasks using the Node.js streaming task-based build system.

## Demands

gulp

## YAML snippet

```
gulp
Run the 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
<code>gulpFile</code> gulp File Path	(Required) Relative path from the repo root of the gulp file script that you want to run. Default value: <code>gulpfile.js</code>
<code>targets</code> gulp Task(s)	(Optional) Space-delimited list of tasks to run. If not specified, the default task will run.
<code>arguments</code> Arguments	Additional arguments passed to gulp. Tip: <code>--gulpfile</code> is not needed since already added via <code>gulpFile</code> input above
<code>cwd</code> Working Directory	(Optional) Current working directory when the script is run. Defaults to the folder where the script is located. Argument aliases: <code>workingDirectory</code>
<code>gulpjs</code> gulp.js location	(Optional) Path to an alternative <code>gulp.js</code> , relative to the working directory. Argument aliases: <code>workingDirectory</code>

ARGUMENT	DESCRIPTION
<code>publishJUnitResults</code> Publish to Azure Pipelines	Select this option to publish JUnit test results produced by the gulp build to Azure Pipelines Default value: false
<code>testResultsFiles</code> Test Results Files	(Required) Test results files path. Wildcards can be used. For example, <code>**/TEST-*.xml</code> for all XML files whose name starts with TEST-. Default value: <code>**/TEST-*.xml</code>
<code>testRunTitle</code> Test Run Title	(Optional) Provide a name for the test run
<code>enableCodeCoverage</code> Enable Code Coverage	(Optional) Select this option to enable Code Coverage using Istanbul Default value: false
<code>testFramework</code> Test Framework	(Optional) Select your test framework Default value: Mocha
<code>srcFiles</code> Source Files	(Optional) Provide the path to your source files, that you want to hookRequire ()
<code>testFiles</code> Test Script Files	(Required) Provide the path to your test script files Default value: <code>test/*.js</code>

## Example

### Run gulp.js

On the [Build](#) tab:

 Package: npm	Install npm. <ul style="list-style-type: none"> <li>Command: <code>install</code></li> </ul>
 Build: gulp	Run your script. <ul style="list-style-type: none"> <li>gulp file path: <code>gulpfile.js</code></li> <li>Advanced, <code>gulp.js</code> location: <code>node_modules/gulp/bin/gulp.js</code></li> </ul>

### 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.

## FAQ

### Do I need an agent?

You need at least one [agent](#) to run your build or release.

**I'm having problems. How can I troubleshoot them?**

See [Troubleshoot Build and Release](#).

**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](#).

**My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?**

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE.EXTRA.CA.CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

**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

3/6/2021 • 6 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - 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 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 and 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
<code>SymbolsFolder</code> Path to symbols folder	(Optional) The path to the folder that is searched for symbol files. The default is \$(Build.SourcesDirectory), Otherwise specify a rooted path. Note that UNC paths aren't supported if you select the Azure Artifacts symbol server as the server type. For example: \$(Build.BinariesDirectory)/MyProject
<code>SearchPattern</code> Search pattern	(Required) <a href="#">File matching pattern(s)</a> The pattern used to discover the pdb files to publish  Default value: **/bin/**/*.pdb
<code>IndexSources</code> Index sources	(Optional) Indicates whether to inject source server information into the PDB files  Default value: true
<code>PublishSymbols</code> Publish symbols	(Optional) Indicates whether to publish the symbol files  Default value: true
<code>SymbolServerType</code> Symbol server type	(Required) Choose where to publish symbols. Symbols published to the Azure Artifacts symbol server are accessible by any user with access to the organization/collection. Azure DevOps Server only supports the "File share" option. Follow <a href="#">these instructions</a> to use Symbol Server in Azure Artifacts. <b>TeamServices:</b> <ul style="list-style-type: none"><li>Symbol Server in this organization/collection (requires Azure Artifacts)</li></ul> <b>File share:</b> <ul style="list-style-type: none"><li>Select this option to use the file share supplied in the next input.</li></ul>
<code>SymbolsPath</code> Path to publish symbols	(Optional) The file share that hosts your symbols. This value will be used in the call to <code>symstore.exe add</code> as the /s parameter.  To prepare your SymStore symbol store: <ol style="list-style-type: none"><li>Set up a folder on a file-sharing server to store the symbols. For example, set up <code>\fabrikam-share\symbols</code>.</li><li>Grant full control permission to the <a href="#">build agent service account</a>.</li></ol> 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 <a href="#">Publish Build Artifacts</a> ).

ARGUMENT	DESCRIPTION
<code>CompressSymbols</code> Compress symbols	(Required) Only available when <b>File share</b> is selected as the <b>Symbol server type</b> . Compresses your <code>pdb</code> s to save space. Default value: false
<b>ADVANCED</b>	
<code>DetailedLog</code> Verbose logging	(Optional) Enables additional log details. Default value: true
<code>TreatNotIndexedAsWarning</code> Warn if not indexed	(Optional) Indicates whether to warn if sources are not indexed for a PDB file. Otherwise the messages are logged as normal output. A common cause of sources to not be indexed are when your solution depends on binaries that it doesn't build. Even if you don't select this option, the messages are written in log.  Default value: false
<code>SymbolsMaximumWaitTime</code> Max wait time (min)	(Optional) The number of minutes to wait before failing this task. If you leave it blank, limit is 2 hours.
<code>SymbolsProduct</code> Product	(Optional) Specify the product parameter to symstore.exe. The default is \$(Build.DefinitionName)
<code>SymbolsVersion</code> Version	(Optional) Specify the version parameter to symstore.exe. The default is \$(Build.BuildNumber).
<code>SymbolsArtifactName</code> Artifact name	(Optional) Specify the artifact name to use for the Symbols artifact. The default is Symbols_{BuildConfiguration}. Default value: Symbols_{BuildConfiguration}

For more information about the different types of tasks and their uses, see [Task control options](#).

#### IMPORTANT

If you want to delete symbols that were published using the `Index Sources & Publish Symbols` task, you must first remove the build that generated those symbols. This can be accomplished by [using retention policies to clean up your build](#) or by [manually deleting the run](#). For information about debugging your app, see [Use indexed symbols to debug your app](#), [Debug with symbols in Visual Studio](#), [Debug with symbols in WinDbg](#).

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### 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.

## I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

## 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](#).

## My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

## 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 Indefinitely.

# Jenkins Queue Job task

4/20/2021 • 4 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

Use this task to queue a job on a Jenkins server.

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## 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: true
 #capturePipeline: true # Required when captureConsole == True
 isParameterizedJob:
 #jobParameters: # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<code>serverEndpoint</code> Jenkins service connection	(Required) Select the service connection for your Jenkins instance. To create one, click <b>Manage</b> and create a new Jenkins service connection.
<code>jobName</code> Job name	(Required) The name of the Jenkins job to queue. This job name must exactly match the job name on the Jenkins server.
<code>isMultibranchJob</code> Job is of multibranch pipeline type	(Optional) This job is of multibranch pipeline type. If selected, enter the appropriate branch name. Requires Team Foundation Server Plugin for Jenkins v5.3.4 or later  Default value: false

ARGUMENT	DESCRIPTION
<code>multibranchPipelineBranch</code> Multibranch pipeline branch	(Required) Queue this multibranch pipeline job on the specified branch. Requires Team Foundation Server Plugin for Jenkins v5.3.4 or later
<code>captureConsole</code> Capture console output and wait for completion	(Required) 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.  Default value: true
<code>capturePipeline</code> Capture pipeline output and wait for pipeline completion	(Required) 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.  Default value: true
<code>isParameterizedJob</code> Parameterized job	(Required) Select if the Jenkins job accepts parameters. This job should be selected even if all default parameter values are used and no parameters are specified.  Default value: false
<code>jobParameters</code> Job parameters	This option is available for parameterized jobs. Specify job parameters, one per line, in the form <b>parameterName=parameterValue</b> preceded by   on the first line. Example:  jobParameters:   parameter1=value1 parameter2=value2  To set a parameter to an empty value (useful for overriding a default value), omit the parameter value. For example, specify <b>parameterName=</b>  Variables are supported. For example, to define the <b>commitId</b> parameter to be the <b>git commit ID</b> for the build, use: <b>commitId=\$(Build.SourceVersion)</b> .  Supported Jenkins parameter types are: <ul style="list-style-type: none"><li>• Boolean</li><li>• String</li><li>• Choice</li><li>• Password</li></ul>

## 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 (for example, Publish Test Results, and Publish Code Coverage Results).

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

**Q: I'm having issues with publishing my artifacts. How can I view the detailed logs?**

To enable detailed logs for your pipeline:

1. Edit your pipeline and select **Variables**
2. Add a new variable with the name `System.Debug` and value `true`
3. **Save**

**Q: Which variables are available to me?**

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just few of the variables you can use in your pipeline. Variables are available as [expressions](#) or scripts.

See [Define variables](#), [predefined variables](#), and [Classic release and artifacts variables](#) to learn about the different types of variables.

### Do I need an agent?

You need at least one [agent](#) to run your build or release.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE.EXTRA.CA.CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

### 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

5/24/2021 • 7 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

Use this task to build your Java code.

## Demands

The build agent must have the following capability:

- Maven

## YAML snippet

```
Maven
Build, test, and deploy 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
 #allowBrokenSymlink: false # Optional. Has effect only when publishJUnitResults == True
 #codeCoverageToolOption: 'None' # Optional. Options: none, cobertura, jaCoCo. Enabling code coverage
 #codeCoverageClassFilter: # Optional. Comma-separated list of filters to include or exclude classes from
 #collecting code coverage. For example: +:com.*,:org.*,-:my.app.*'
 #codeCoverageClassDirectories: # 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
 #effectivePomSkip: false
 #sonarQubeRunAnalysis: false
 #sqMavenPluginVersionChoice: 'latest' # Required when sonarQubeRunAnalysis == True# Options: latest, pom
 #checkStyleRunAnalysis: false # Optional
 #pmdRunAnalysis: false # Optional
 #findBugsRunAnalysis: false # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
----------	-------------

ARGUMENT	DESCRIPTION
<code>mavenPOMFile</code> Maven POM file	(Required) Relative path from the repository root to the Maven POM file. See <a href="#">Introduction to the POM</a> . Default value: pom.xml Argument aliases: <code>mavenPomFile</code>
<code>goals</code> Goal(s)	(Optional) 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 <a href="#">Introduction to the Maven build lifecycle</a> . Default value: package
<code>options</code> Options	(Optional) Specify any Maven command-line options you want to use.
<code>publishJUnitResults</code> Publish to Azure Pipelines	(Required) Select this option to publish JUnit test results produced by the Maven build to Azure Pipelines. Each test results file matching <code>Test Results Files</code> will be published as a test run in Azure Pipelines. Default value: true
<code>testResultsFiles</code> Test results files	(Required) Specify the path and pattern of test results files to publish. Wildcards can be used ( <a href="#">more information</a> ). For example, <code>*/TEST-.xml</code> for all XML files whose name starts with <code>TEST-</code> . If no root path is specified, files are matched beneath the default working directory, the value of which is available in the variable: <code>\$(System.DefaultWorkingDirectory)</code> . For example, a value of ' <code>/TEST-.xml</code> ' <i>will actually result in matching files from <code>'\$(System.DefaultWorkingDirectory)// /TEST-.xml'</code></i> . Default value: <code>**/surefire-reports/TEST-*.*xml</code>
<code>testRunTitle</code> Test run title	(Optional) Provide a name for the test run.
<code>allowBrokenSymbolicLinks</code> Test results files	(Optional) Set false to fail build when the task meets broken symbolic links during publishing tests. It has an effect only if <code>publishJUnitResults</code> is set <code>true</code> Default value: true Argument aliases: <code>allowBrokenSymlink</code>
<code>codeCoverageTool</code> Code coverage tool	(Optional) Select the code coverage tool. Enabling code coverage inserts the <code>clean</code> goal into the Maven goals list when Maven runs. Default value: None Argument aliases: <code>codeCoverageToolOption</code>
<code>classFilter</code> Class inclusion/exclusion filters	(Optional) Comma-separated list of filters to include or exclude classes from collecting code coverage. For example: <code>+:com,+:org,-:my.app</code> . Argument aliases: <code>codeCoverageClassFilter</code>

ARGUMENT	DESCRIPTION
<p><code>classFilesDirectories</code> Class files directories</p>	<p>(Optional) This field is required for a multi-module project. Specify a comma-separated list of relative paths from the Maven POM file to directories containing class files and archive files (JAR, WAR, etc.). Code coverage is reported for class files in these directories.</p> <p>For example: target/classes,target/testClasses.</p> <p>Argument aliases: <code>codeCoverageClassFilesDirectories</code></p>
<p><code>srcDirectories</code> Source files directories</p>	<p>(Optional) This field is required for a multi-module project. Specify a comma-separated list of relative paths from the Maven POM file to source code directories. Code coverage reports will use these to highlight source code.</p> <p>For example: src/java,src/Test.</p> <p>Argument aliases: <code>codeCoverageSourceDirectories</code></p>
<p><code>failIfCoverageEmpty</code> Fail when code coverage results are missing</p>	<p>(Optional) Fail the build if code coverage did not produce any results to publish.</p> <p>Default value: false</p> <p>Argument aliases: <code>codeCoverageFailIfEmpty</code></p>
<p><code>javaHomeSelection</code> Set JAVA_HOME by</p>	<p>(Required) Sets JAVA_HOME either by selecting a JDK version that will be discovered during builds or by manually entering a JDK path. Please note that if you already have java installed on agent machine - you can specify it by setting up 'javaHomeOption' as 'path', and 'jdkDirectory' - as a path to jdk installed directory.</p> <p>Default value: JDKVersion</p> <p>Argument aliases: <code>javaHomeOption</code></p>
<p><code>jdkVersion</code> JDK version</p>	<p>(Optional) Will attempt to discover the path to the selected JDK version and set JAVA_HOME accordingly.</p> <p><b>Note:</b> If running on an agent not hosted by Microsoft, and the requested Java version is not the one indicated by the JAVA_HOME variable set on the agent machine, the task will rely on the variable <code>JAVA_HOME_&lt;version&gt;_&lt;arch&gt;</code> (e.g. <code>JAVA_HOME_8_X64</code>), to locate the necessary JDK. Ensure this variable is set on self-hosted agents for any version and architecture of the JDK that may be requested by this parameter and/or by <code>jdkArchitecture</code>.</p> <p>Default value: default</p> <p>Argument aliases: <code>jdkVersionOption</code></p>
<p><code>jdkUserInputPath</code> JDK path</p>	<p>(Required) Sets JAVA_HOME to the given path.</p> <p>Argument aliases: <code>jdkDirectory</code></p>
<p><code>jdkArchitecture</code> JDK architecture</p>	<p>(Optional) Optionally supply the architecture (x86, x64) of the JDK.</p> <p><b>Note:</b> If running on an agent not hosted by Microsoft, and the requested Java architecture is not the one indicated by the JAVA_HOME variable set on the agent machine, the task will rely on the variable <code>JAVA_HOME_&lt;version&gt;_&lt;arch&gt;</code> (e.g. <code>JAVA_HOME_8_X64</code>), to locate the necessary JDK. Ensure this variable is set on self-hosted agents for any version and architecture of the JDK that may be requested by this parameter and/or by <code>jdkVersion</code>.</p> <p>Default value: x64</p> <p>Argument aliases: <code>jdkArchitectureOption</code></p>

ARGUMENT	DESCRIPTION
<code>mavenVersionSelection</code> Maven version	(Required) Uses either the default Maven version or the version in the specified custom path. Default value: Default Argument aliases: <code>mavenVersionOption</code>
<code>mavenPath</code> Maven path	(Required) Supply the custom path to the Maven installation (e.g., /usr/share/maven). Argument aliases: <code>mavenDirectory</code>
<code>mavenSetM2Home</code> Set M2_HOME variable	(Required) Sets the M2_HOME variable to a custom Maven installation path. Default value: false
<code>mavenOpts</code> Set MAVEN_OPTS to	(Optional) Sets the MAVEN_OPTS environment variable, which is used to send command-line arguments to start the JVM. The -Xmx flag specifies the maximum memory available to the JVM. Default value: -Xmx1024m Argument aliases: <code>mavenOptions</code>
<code>mavenFeedAuthenticate</code> Authenticate built-in Maven feeds	(Required) Automatically authenticate Maven feeds from Azure Artifacts. If built-in Maven feeds are not in use, deselect this option for faster builds. Default value: false Argument aliases: <code>mavenAuthenticateFeed</code>
<code>skipEffectivePom</code> Skip generating effective POM while authenticating built-in feeds	(Required) Authenticate built-in Maven feeds using the POM only, allowing parent POMs in Azure Artifacts/Azure DevOps Server [Package Management] feeds. Default value: false Argument aliases: <code>effectivePomSkip</code>
<code>sqAnalysisEnabled</code> Run SonarQube or SonarCloud analysis	(Required) This option has changed from version 1 of the Maven task to use the <a href="#">SonarQube</a> and <a href="#">SonarCloud</a> marketplace extensions. Enable this option to run <a href="#">SonarQube or SonarCloud analysis</a> after executing goals in the <b>Goals</b> field. The <b>install</b> or <b>package</b> goal should run first. You must also add a <b>Prepare Analysis Configuration</b> task from one of the extensions to the build pipeline before this Maven task. Default value: false Argument aliases: <code>sonarQubeRunAnalysis</code>
<code>sqMavenPluginVersionChoice</code> <> SonarQube scanner for Maven version	(Required) The SonarQube Maven plugin version to use. You can use latest version, or rely on the version in your pom.xml. Default value: latest
<code>checkstyleAnalysisEnabled</code> Run Checkstyle	(Optional) Run the Checkstyle tool with the default Sun checks. Results are uploaded as build artifacts. Default value: false Argument aliases: <code>checkStyleRunAnalysis</code>

ARGUMENT	DESCRIPTION
<code>pmdAnalysisEnabled</code> Run PMD	(Optional) Use the PMD static analysis tool to look for bugs in the code. Results are uploaded as build artifacts. Default value: false Argument aliases: <code>pmdRunAnalysis</code>
<code>findbugsAnalysisEnabled</code> Run FindBugs	(Optional) Use the FindBugs static analysis tool to look for bugs in the code. Results are uploaded as build artifacts. Default value: false Argument aliases: <code>findBugsRunAnalysis</code>

#### CONTROL OPTIONS

##### IMPORTANT

When using the `-q` option in your MAVEN\_OPTS, an effective pom won't be generated correctly and Azure Artifacts feeds may not be able to be authenticated.

## 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.

## FAQ

### Do I need an agent?

You need at least one [agent](#) to run your build or release.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

### 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

3/9/2021 • 7 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Use this task 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, 16.0, 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
 #logFileVerbosity: 'normal' # Optional. Options: quiet, minimal, normal, detailed, diagnostic
```

## Arguments

ARGUMENT	DESCRIPTION
----------	-------------

ARGUMENT	DESCRIPTION
<p><code>solution</code> Project</p>	<p>(Required) 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"> <li>• If you use TFVC, make sure that the project is a child of one of the mappings on the Repository tab.</li> <li>• If you use Git, make sure that the project or project is in your Git repo, in a branch that you're building.</li> </ul> <p>Tip: If you are building a solution, we recommend you use the <a href="#">Visual Studio build task</a> instead of the MSBuild task.</p> <p>Default value: <code>**/*.sln</code></p>
<p><code>msbuildLocationMethod</code> MSBuild</p>	<p>(Optional) Default value: <code>version</code></p>
<p><code>msbuildVersion</code> MSBuild Version</p>	<p>(Optional) If the preferred version cannot be found, the latest version found will be used instead. On an macOS agent, xbuild (Mono) will be used if version is lower than 15.0 Default value: <code>latest</code></p>
<p><code>msbuildArchitecture</code> MSBuild Architecture</p>	<p>(Optional) Optionally supply the architecture (x86, x64) of MSBuild to run Default value: <code>x86</code></p>
<p><code>msbuildLocation</code> Path to MSBuild</p>	<p>(Optional) Optionally supply the path to MSBuild</p>
<p><code>platform</code> Platform</p>	<p>(Optional) 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"> <li>• If you are targeting an MSBuild project (.proj) file instead of a solution, specify <code>AnyCPU</code> (no whitespace).</li> <li>• 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.</li> </ul>

ARGUMENT	DESCRIPTION
<code>configuration</code> Configuration	(Optional) 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.
<code>msbuildArguments</code> MSBuild Arguments	(Optional) Additional arguments passed to MSBuild (on Windows) and xbuild (on macOS)
<code>clean</code> 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> <p>See, <a href="#">repo options</a></p> <p>Default value: false</p>
<code>maximumCpuCount</code> Build in Parallel	(Optional) If your MSBuild target configuration is compatible with building in parallel, you can optionally check this input to pass the <code>/m</code> switch to MSBuild (Windows only). If your target configuration is not compatible with building in parallel, checking this option may cause your build to result in file-in-use errors, or intermittent or inconsistent build failures. Default value: false
<code>restoreNugetPackages</code> Restore NuGet Packages	<b>(Important)</b> This option is deprecated. Make sure to clear this checkbox and instead use the <a href="#">NuGet Installer</a> build task. Default value: false
ADVANCED	
<code>logProjectEvents</code> Record Project Details	Optionally record timeline details for each project (Windows only) Default value: false
<code>createLogFile</code> Create Log File	Optionally create a log file (Windows only)  Default value: false
<code>logfileVerbosity</code> Log File Verbosity	Optional log file verbosity  Default value: normal
CONTROL OPTIONS	

# Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### 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 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.

## I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

## 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](#).

## My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

## 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](#).

# Troubleshooting

This section provides troubleshooting tips for common issues that a user might encounter when using the MSBuild task.

- [Build failed with the following error: An internal failure occurred while running MSBuild](#)

## Build failed with the following error: An internal failure occurred while running MSBuild

- [Possible causes](#)
- [Troubleshooting suggestions](#)

### Possible causes

- Change in the MSBuild version.
- Issues with a third-party extension.
- New updates to Visual Studio that can cause missing assemblies on the build agent.
- Moved or deleted some of the necessary NuGet packages.

### Troubleshooting suggestions

- [Run the pipeline with diagnostics to retrieve detailed logs](#)
- [Try to reproduce the error locally](#)
- [What else can I do?](#)

#### Run the pipeline with diagnostics to retrieve detailed logs

One of the available options to diagnose the issue is to take a look at the generated logs. You can view your pipeline logs by selecting the appropriate task and job in your pipeline run summary.

To get the logs of your pipeline execution [Get logs to diagnose problems](#)

You can also setup and download a customized verbose log to assist with your troubleshooting:

- [Configure verbose logs](#)
- [View and download logs](#)

In addition to the pipeline diagnostic logs, you can also check these other types of logs that contain more information to help you debug and solve the problem:

- [Worker diagnostic logs](#)

- [Agent diagnostic logs](#)
- [Other logs](#) (Environment and capabilities)

**Try to reproduce the error locally**

If you are using a hosted build agent, you might want to try to reproduce the error locally. This will help you to narrow down whether the failure is the result of the build agent or the build task.

Run the same MSBuild command on your local machine using the same arguments. Check out [MSBuild command](#) for reference

**TIP**

If you can reproduce the problem on your local machine, then your next step is to investigate the [MSBuild](#) issue.

For more information on [Microsoft hosted agents](#)

To setup your own self-hosted agent and run the build jobs:

- [Self-hosted Windows agents](#)
- [Self-hosted Linux agents](#)
- [Self-hosted MacOS agents](#)

**What else can I do?**

At the bottom of this page, check out the GitHub issues in the [Open](#) and [Closed](#) tabs to see if there is a similar issue that has been resolved previously by our team.

Some of the MSBuild errors are caused by a change in Visual Studio so you can search on [Visual Studio Developer Community](#) to see if this issue has been reported. We also welcome your questions, suggestions, and feedback.

# Visual Studio Build task

11/2/2020 • 6 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Use this task 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, 16.0, 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
 #logFileVerbosity: 'normal' # Optional. Options: quiet, minimal, normal, detailed, diagnostic
```

## Arguments

ARGUMENT	DESCRIPTION
----------	-------------

Argument	Description
<p><code>solution</code> Solution</p>	<p>(Required) 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 (*) and recursive wildcards (**). For example, `**.sln` searches for all .sln 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"> <li>• If you use TFVC, make sure that the solution is a child of one of the mappings on the Repository tab.</li> <li>• If you use Git, make sure that the project or solution is in your Git repo, and in a branch that you're building.</li> </ul> <p>Tips:</p> <ul style="list-style-type: none"> <li>• You can also build MSBuild project (*.proj) files.</li> <li>• If you are building a customized MSBuild project file, we recommend you use the <a href="#">MSBuild task</a> instead of the Visual Studio Build task.</li> </ul> <p>Default value: **\*.sln</p>
<p><code>vsVersion</code> Visual Studio Version</p>	<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></p> <p>argument to the MSBuild command run by the build. For example, if you select <b>Visual Studio 2015</b>,  <code>/p:VisualStudioVersion=14.0</code> is added to the MSBuild command.</p> <p><b>Azure Pipelines:</b> If your team wants to use Visual Studio 2017 with the Microsoft-hosted agents, select <b>Hosted VS2017</b> as your default build pool. See <a href="#">Microsoft-hosted agents</a>.</p> <p>Default value: latest</p>
<p><code>msbuildArgs</code> MSBuild Arguments</p>	<p>(Optional) You can pass additional arguments to MSBuild. For syntax, see <a href="#">MSBuild Command-Line Reference</a>.</p>

ARGUMENT	DESCRIPTION
<code>platform</code> Platform	<p>(Optional) 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"> <li>• If you are targeting an MSBuild project (<code>*proj</code>) file instead of a solution, specify <code>AnyCPU</code> (no whitespace).</li> <li>• 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.</li> </ul>
<code>configuration</code> Configuration	<p>(Optional) 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>
<code>clean</code> Clean	<p>(Optional) 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>
<b>ADVANCED</b>	
<code>maximumCpuCount</code> Build in Parallel	<p>(Optional) If your MSBuild target configuration is compatible with building in parallel, you can optionally check this input to pass the <code>/m</code> switch to MSBuild (Windows only). If your target configuration is not compatible with building in parallel, checking this option may cause your build to result in file-in-use errors, or intermittent or inconsistent build failures.</p> <p>Default value: false</p>
<code>restoreNugetPackages</code> Restore NuGet Packages	<p><b>(Important)</b> This option is deprecated. Make sure to clear this checkbox and instead use the <a href="#">NuGet Installer</a> build task.</p> <p>Default value: false</p>

ARGUMENT	DESCRIPTION
<code>msbuildArchitecture</code> MSBuild Architecture	<p>Optional supply the architecture (x86, x64) of MSBuild to run</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> <p>Default value: x86</p>
<code>logProjectEvents</code> Record Project Details	<p>Optional record timeline details for each project</p> <p>Default value: true</p>
<code>createLogFile</code> Create Log File	<p>Optional create a log file (Windows only)</p> <p>Default value: false</p>
<code>logfileVerbosity</code> Log File Verbosity	<p>Optional log file verbosity</p> <p>Default value: normal</p>
<b>CONTROL OPTIONS</b>	

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### 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 **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)`
4. Under the agent job of the assigned task, on the Parallelism tab, select **Multi-configuration** and specify the Multipliers separated by commas. For example: `BuildConfiguration, BuildPlatform`

### 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.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

### 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

11/2/2020 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

Use this task to build an Android app with Xamarin.

## 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, 16.0, 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
<code>project</code> Project	(Required) Relative path from repo root of Xamarin.Android project(s) to build. Wildcards can be used <a href="#">more information</a> . For example, <code>**/*.csproj</code> for all csproj files in all subfolders. The project must have a PackageForAndroid target if <code>Create App Package</code> is selected. Default value: <code>**/*.csproj</code> Argument aliases: <code>projectFile</code>
<code>target</code> Target	(Optional) Build these targets in this project. Use a semicolon to separate multiple targets.
<code>outputDir</code> Output Directory	Optionally provide the output directory for the build. Example: <code>\$(build.binariesDirectory)/bin/Release</code> Argument aliases: <code>outputDirectory</code>

ARGUMENT	DESCRIPTION
<code>configuration</code> Configuration	(Optional) 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.
<code>createAppPackage</code> Create app package	(Optional) Passes the target ( <code>/t:PackageForAndroid</code> ) during build to generate an APK. Default value: true
<code>clean</code> Clean	(Optional) Passes the clean target ( <code>/t:clean</code> ) during build Default value: false
MSBUILD OPTIONS	
<code>msbuildLocationMethod</code> MSBuild	(Optional) Path to MSBuild (on Windows) or xbuild (on macOS). Default behavior is to search for the latest version. Default value: version Argument aliases: <code>msbuildLocationOption</code>
<code>msbuildVersion</code> MSBuild version	(Optional) If the preferred version cannot be found, the latest version found will be used instead. On macOS, xbuild (Mono) or MSBuild (Visual Studio for Mac) will be used. Default value: 15.0 Argument aliases: <code>msbuildVersionOption</code>
<code>msbuildLocation</code> MSBuild location	(Required) Optionally supply the path to MSBuild (on Windows) or xbuild (on macOS) Default value: version Argument aliases: <code>msbuildFile</code>
<code>msbuildArchitecture</code> MSBuild architecture	Optionally supply the architecture (x86, x64) of MSBuild to run Default value: x86 Argument aliases: <code>msbuildArchitectureOption</code>
<code>msbuildArguments</code> Additional Arguments	(Optional) Additional arguments passed to MSBuild (on Windows) or xbuild (on macOS).
JDK OPTIONS	
<code>jdkSelection</code> Select JDK to use for the build	(Required) 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"><li>• JDK Version: Select the JDK version you want to use.</li><li>• JDK Path: Specify the path to the JDK you want to use.</li></ul> Default value: JDKVersion Argument aliases: <code>jdkOption</code>

ARGUMENT	DESCRIPTION
<code>jdkVersion</code> JDK version	(Optional) Use the selected JDK version during build. Default value: default Argument aliases: <code>jdkVersionOption</code>
<code>jdkUserInputPath</code> JDK path	(Required) Use the selected JDK version during build. Default value: default Argument aliases: <code>jdkDirectory</code>
<code>jdkArchitecture</code> JDK Architecture	Optionally supply the architecture (x86, x64) of JDK Default value: x64 Argument aliases: <code>jdkArchitectureOption</code>
<b>CONTROL OPTIONS</b>	

## Example

[Build your Xamarin app](#)

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### Do I need an agent?

You need at least one [agent](#) to run your build or release.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

### 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

11/2/2020 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

Use this task in a pipeline to build an iOS app with Xamarin on macOS. For more information, see the [Xamarin guidance](#) and [Sign your app during CI](#).

## 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: true
 #buildForSimulator: false # Optional
 #runNugetRestore: false
 #args: # Optional
 #workingDirectory: # Optional
 #mdtoolFile: # Optional
 #signingIdentity: # Optional
 #signingProvisioningProfileID: # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<code>solution</code> Solution	(Required) Relative path from the repository root of the Xamarin.iOS solution or csproj project to build. May contain wildcards. Default value: <code>**/*.sln</code> Argument aliases: <code>solutionFile</code>
<code>configuration</code> Configuration	(Required) Standard configurations are Ad-Hoc, AppStore, Debug, Release. Default value: Release
<code>clean</code> Clean	(Optional) Run a clean build ( <code>/t:clean</code> ) prior to the build. Default value: false
<code>packageApp</code> Create app package	(Required) Indicates whether an IPA should be generated as a part of the build. Default value: true

ARGUMENT	DESCRIPTION
<code>forSimulator</code> Build for iOS Simulator	(Optional) Optionally build for the iOS Simulator instead of physical iOS devices. Default value: false Argument aliases: <code>buildForSimulator</code>
<code>runNugetRestore</code> Run NuGet restore	(Required) Optionally run <code>nuget restore</code> on the Xamarin iOS solution to install all referenced packages before build. The 'nuget' tool in the PATH of the build agent machine will be used. To use a different version of NuGet or set additional arguments, use the <a href="#">NuGet Tool Installer</a> task. Default value: false
<code>args</code> Arguments	(Optional) Additional command line arguments that should be used to build.
<code>cwd</code> Working directory	(Optional) Working directory in which builds will run. When empty, the root of the repository is used. Argument aliases: <code>workingDirectory</code>
<code>buildToolLocation</code> Build tool path	(Optional) Optionally supply the full path to MSBuild (the Visual Studio for Mac build tool). When empty, the default MSBuild path is used. Argument aliases: <code>mdtoolFile</code> , <code>mdtoolLocation</code>
<code>iosSigningIdentity</code> Signing identity	(Optional) Optionally override the signing identity that will be used to sign the build. If nothing is entered, the setting in the project will be used. Argument aliases: <code>signingIdentity</code>
<code>provProfileUuid</code> Provisioning profile UUID	(Optional) Optional UUID of an installed provisioning profile to be used for this build. Argument aliases: <code>signingProvisioningProfileID</code>
<b>CONTROL OPTIONS</b>	

## Example

[Build your Xamarin app](#)

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### Do I need an agent?

You need at least one [agent](#) to run your build or release.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

**My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?**

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE.EXTRA.CA.CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

**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

3/6/2021 • 7 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

Use this task 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 8' # Optional. Default value: iPhone8 for Xcode 11 and iPhone 7 for other iOS projects; Apple TV for tvOS projects.
 #destinationDevices: # Optional
 #args: # Optional
 #workingDirectory: # Optional
 #useXcpretty: true # Optional
 #publishJUnitResults: # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<code>actions</code> Actions	(Required) 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 <a href="#">Apple: Building from the command line with Xcode FAQ</a> . Default value: <code>build</code>

ARGUMENT	DESCRIPTION
<code>configuration</code> Configuration	(Optional) 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 <a href="#">Variables</a> tab. Default value: <code>\$(Configuration)</code>
<code>sdk</code> SDK	(Optional) 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 <a href="#">Variables</a> tab. Default value: <code>\$(SDK)</code>
<code>xcWorkspacePath</code> 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> . Default value: <code>**/*.{xcodeproj,project.xcworkspace}</code>
<code>scheme</code> Scheme	(Optional) Enter a scheme name defined in Xcode. It must be a shared scheme, with its <b>Shared</b> checkbox enabled under <b>Managed Schemes</b> in Xcode. If you specify a <b>Workspace or project path</b> above without specifying a scheme, and the workspace has a single shared scheme, it will be automatically used.
<code>xcodeVersion</code> Xcode version	(Optional) 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. Default value: <code>default</code>
<code>xcodeDeveloperDir</code> 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.
<b>(OPTIONAL) SIGNING &amp; PROVISIONING</b>	
<code>signingOption</code> Signing style	(Optional) 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. Default value: <code>nosign</code>
<code>signingIdentity</code> 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.

ARGUMENT	DESCRIPTION
<code>provisioningProfileUuid</code> 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).
<code>provisioningProfileName</code> Provisioning profile name	(Optional) Enter the name of an installed provisioning profile to be used for this build. If specified, this takes precedence over the provisioning profile UUID. Use separate build tasks with different schemes or targets to specify separate provisioning profiles by target in a single workspace (iOS, tvOS, watchOS).
<code>teamId</code> Team ID	(Optional, unless you are a member of multiple development teams.) Specify the 10-character development team ID.
PACKAGE OPTIONS	
<code>packageApp</code> Create app package	Indicate whether an IPA app package file should be generated as a part of the build. Default value: false
<code>archivePath</code> Archive path	(Optional) Specify a directory where created archives should be placed.
<code>exportPath</code> Export path	(Optional) Specify the destination for the product exported from the archive. Default value: output/\$(SDK)/\$(Configuration)
<code>exportOptions</code> Export options	(Optional) 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 <b>Export method</b> and <b>Team ID</b> . Default value: auto
<code>exportMethod</code> Export method	(Required) 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> . Default value: development
<code>exportTeamId</code> Team ID	(Optional) Enter the 10-character team ID from the Apple Developer Portal to use during export.
<code>exportOptionsPlist</code> Export options plist	(Required) Enter the path to the plist file that contains options to use during export.
<code>exportArgs</code> Export arguments	(Optional) Enter additional command line arguments to be used during export.
DEVICES & SIMULATORS	
<code>destinationPlatformOption</code> Destination platform	(Optional) 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. Default value: default

ARGUMENT	DESCRIPTION
<code>destinationPlatform</code> Custom destination platform	(Optional) 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. Default value: default
<code>destinationTypeOption</code> Destination type	(Optional) 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 <b>Devices and Simulators</b> in Xcode. Default value: simulators
<code>destinationSimulators</code> Simulators	(Optional) 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 <b>Hosted macOS</b> agent can be <a href="#">found here</a> . Default value: iPhone8 for Xcode 11 and iPhone 7 for other iOS projects; Apple TV for tvOS projects.
<code>destinationDevices</code> Devices	(Optional) 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.
<b>ADVANCED</b>	
<code>args</code> 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 <a href="#">Apple: Building from the command line with Xcode FAQ</a> .
<code>cwd</code> 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. Argument aliases: <code>workingDirectory</code>
<code>useXcpretty</code> Use xcpretty	(Optional) 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 <a href="#">xcpretty</a> on GitHub. Default value: true
<code>xPRETTYArgs</code> Arguments for xcpretty	(Optional) If xcpretty is enabled above, specify arguments for xcpretty. See <a href="#">xcpretty</a> on GitHub for a list of xcpretty arguments.
<code>publishJUnitResults</code> Publish test results to Azure Pipelines/TFS	(Optional) If xcpretty is enabled above, specify whether to publish JUnit test results to Azure Pipelines/TFS. Default value: false
<code>testRunTitle</code> Test run title	(Optional) If xcpretty and publishJUnitResults are enabled above, you can specify test run title.
<b>CONTROL OPTIONS</b>	

## Example

[Build your Xcode app](#)

## Open source

This task is open source on [GitHub](#). Feedback and contributions are welcome.

## Using multiple provisioning profiles

Currently there's no support of multiple provisioning profiles for Xcode task (for example for iOS App Extension)

## FAQ

### Do I need an agent?

You need at least one [agent](#) to run your build or release.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE.EXTRA.CA.CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

# Xcode Build task

6/2/2020 • 4 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | TFS 2018 | TFS 2017 | TFS 2015

Use this task to build an Xcode workspace on macOS.

## Demands

xcode

## Arguments

ARGUMENT	DESCRIPTION
Actions	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> would run a clean build. See <a href="#">Apple: Building from the command line with Xcode FAQ</a> .
Configuration	The default value is <code>\$(Configuration)</code> . Make sure to specify a value (for example, <code>Release</code> ) on the <a href="#">variables tab</a> .
SDK	The default value is <code>\$(SDK)</code> . Make sure to specify a value (for example, <code>iphonesimulator</code> ) on the <a href="#">variables tab</a> .
Workspace Path	(Optional) Relative path from repo root to where the xcworkspace folder exists. If you specify a value, you must also specify the Scheme. Do not specify a value if you are specifying the <code>-target</code> flag in Advanced, Arguments (described below).
Scheme	(Optional, unless you specify the Workspace Path, described above.) Scheme name in Xcode. Must be a shared scheme (with shared check box under managed schemes in Xcode selected).
Create App Package	Select if you want to create an .IPA app package file.
<b>(OPTIONAL) SIGNING &amp; PROVISIONING</b>	
Xcode 8 Automatic Signing	Select this if you have an Xcode 8 project setup for Automatic Signing.
Team ID	(Optional) Specify the 10 digit developer Team ID. This is required if you are a member of multiple development teams.

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 <b>File Contents</b> to use a P12 certificate and provisioning profile. Choose <b>Identifiers</b> 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	<p>Password to the .p12 file.</p> <p><b>Important:</b> Use a <a href="#">secret variable</a> 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	<p>Select if you want the contents of the provisioning profile file to be removed from the build agent after the build is complete.</p> <p><b>Important:</b> Select only if you are running one agent per user.</p>
PACKAGE OPTIONS	
Archive Path	Optionally specify a directory where created archives should be placed.
Export Path	Optionally specify the destination for the product exported from the archive.
Export Options	<p>Pick a way to pass in Export Options when exporting the archive. The task automatically detects the export method from the archive when the default <b>Auto</b> is selected. You can specify the export method and Team ID in the task by selecting <b>Specify</b> or provide the export options via a plist file by choosing <b>Plist</b>.</p>
Export Method	Method for how Xcode should export the archive. E.g. app-store, package, ad-hoc, enterprise, development.
Team ID	The 10 digit Team ID from the Apple Developer Portal to use for this export.
Export Options Plist	Path to a plist file that configures archive exporting.

ARGUMENT	DESCRIPTION
<b>ADVANCED</b>	
Arguments	(Optional) Specify additional command-line arguments for this build. See <a href="#">Apple: Building from the command line with Xcode FAQ</a> .
Working Directory	Working directory for the build. If you leave it blank, it is the root of the repo.
Output Directory	Relative path where build output (binaries) will be placed. The default value includes build variables. Make sure to specify values on the <a href="#">variables tab</a> .
Xcode Developer Path	(Optional) Path to Xcode Developer folder if you are not using the system default. Specify this argument when you've got multiple versions of Xcode installed your build agent. For example: <code>/Applications/Xcode 7.app/Contents/Developer</code>
Use xcpretty	Use <a href="#">xcpretty</a> to format xcodebuild output and generate JUnit test results report. You must have xcpretty installed on the agent.  If you select this option, make sure that <b>Use xctool</b> is cleared.
Publish to Azure Pipelines/TFS	If you have selected Use xcpretty (described above), select this option if you want to publish JUnit Test results to Azure Pipelines/TFS.
<b>XCTOOL (DEPRECATED)</b>	
Use xctool	Select if you want to run <a href="#">xctool</a> instead of <a href="#">xcodebuild</a> . You must have xctool installed on the agent.  If you select this option, make sure that <b>Use xcpretty</b> is cleared.
xctool Test Reporter Format	You can specify this argument if you have selected Use xctool and included <code>test</code> in Actions (arguments described above). Specify <code>junit:output-file-path-here.xml</code> to generate a file that you can publish using the <a href="#">Publish Test Results</a> task. When specified, <code>plain</code> is automatically added as well.
<b>CONTROL OPTIONS</b>	

## Example

[Build and test Xcode projects](#)

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# FAQ

## Do I need an agent?

You need at least one [agent](#) to run your build or release.

## I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

## 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](#).

## My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE.EXTRA.CA.CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

## 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 Package iOS task

6/2/2020 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

Use this task 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 <code>xcodebuild -showsdk</code> s 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 <a href="#">variables tab</a> .

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 <a href="#">variables tab</a>.</p>
<b>CONTROL OPTIONS</b>	

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### Do I need an agent?

You need at least one [agent](#) to run your build or release.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

### 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

3/8/2021 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

Use this task 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
Compress files into .7z, .tar.gz, or .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
 #verbose: # Optional
 #quiet: # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<code>rootFolderOrFile</code> Root folder or file to archive	(Required) Enter the root folder or file path to add to the archive. If a folder, everything under the folder will be added to the resulting archive Default value: <code>\$(Build.BinariesDirectory)</code>
<code>includeRootFolder</code> Prepend root folder name to archive paths	(Required) If selected, the root folder name will be prefixed to file paths within the archive. Otherwise, all file paths will start one level lower. For example, suppose the selected root folder is: <code>/home/user/output/classes/</code> , and contains: <code>com/acme/Main.class</code> . • 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
<code>archiveType</code> Archive type	(Required) Specify the compression scheme used. To create <code>foo.jar</code> , for example, choose zip 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> . <ul style="list-style-type: none"> <li><code>zip</code> - default, zip format, choose this for all zip compatible types, (.zip, .jar, .war, .ear)</li> <li><code>7z</code> - 7-Zip format, (.7z)</li> <li><code>tar</code> - tar format, choose this for compressed tars, (.tar.gz, .tar.bz2, .tar.xz)</li> <li><code>wim</code> - wim format, (.wim)</li> </ul>
<code>sevenZipCompression</code> 7z compression	Optionally choose a compression level, or choose <code>None</code> to create an uncompressed 7z file Default value: <code>5</code>
<code>tarCompression</code> Tar compression	Optionally choose a compression scheme, or choose <code>None</code> to create an uncompressed tar file. <ul style="list-style-type: none"> <li><code>gz</code> - default, gzip compression (.tar.gz, .tar.tgz, .taz)</li> <li><code>bz2</code> - bzip2 compression (.tar.bz2, .tz2, .tbz2)</li> <li><code>xz</code> - xz compression (.tar.xz, .txz)</li> <li><code>None</code> - no compression, choose this to create a uncompressed tar file (.tar)</li> </ul> Default value: <code>gz</code>
<code>archiveFile</code> Archive file to create	(Required) Specify the name of the archive file to create. <b>For example</b> , to create <code>foo.tgz</code> , select the <code>tar</code> archive type and <code>gz</code> for tar compression. Default value: <code>\$(Build.ArtifactStagingDirectory)/\$(Build.BuildId).zip</code>
<code>replaceExistingArchive</code> Replace existing archive	(Required) 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. If adding to an existing archive, these types are supported: <ul style="list-style-type: none"> <li><code>zip</code></li> <li><code>7z</code></li> <li><code>tar</code> - uncompressed only</li> <li><code>wim</code></li> </ul>
<code>verbose</code> Forces verbose output	(Optional) If set to true, forces tools to use verbose output. Overrides 'quiet'. Default value: false
<code>quiet</code> Forces quiet output	(Optional) If set to true, forces tools to use quiet output. Can be overridden by 'verbose'. Default value: false

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# FAQ

**Q: I'm having issues with publishing my artifacts. How can I view the detailed logs?**

To enable detailed logs for your pipeline:

1. Edit your pipeline and select **Variables**
2. Add a new variable with the name `System.Debug` and value `true`
3. Save

**Q: Which variables are available to me?**

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just few of the variables you can use in your pipeline. Variables are available as [expressions](#) or scripts.

See [Define variables](#), [predefined variables](#), and [Classic release and artifacts variables](#) to learn about the different types of variables.

**Do I need an agent?**

You need at least one [agent](#) to run your build or release.

**I'm having problems. How can I troubleshoot them?**

See [Troubleshoot Build and Release](#).

**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](#).

**My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?**

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

**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

4/10/2020 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task 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 or disconnect an Azure virtual machine's network interface to a Load Balancer's back end address
pool
- task: AzureNLBManagement@1
 inputs:
 azureSubscription:
 resourceName:
 loadBalancer:
 action: # Options: disconnect, connect
```

## Arguments

ARGUMENT	DESCRIPTION
<code>ConnectedServiceName</code> Azure Subscription	(Required) Select the Azure Resource Manager subscription for the deployment Argument aliases: <code>azureSubscription</code>
<code>ResourceGroupName</code> Resource Group	(Required) Select the resource group name
<code>LoadBalancer</code> Load Balancer Name	(Required) Select or enter the load balancer
<code>Action</code> Action	(Required) <b>Disconnect:</b> Removes the virtual machine's primary network interface from the load balancer's backend pool. So that it stops receiving network traffic. <b>Connect:</b> 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

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Bash task

3/8/2021 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task 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: '# echo Hello world' # Required when targetType == inline
 workingDirectory: # Optional
 failOnStderr: false # Optional
 noProfile: true # Optional
 noRc: true # 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
<code>targetType</code> Type	(Optional) Target script type: File Path or Inline Default value: <code>filePath</code>
<code>filePath</code> Script Path	(Required) Path of the script to execute. Must be a fully qualified path or relative to <code>\$(System.DefaultWorkingDirectory)</code> .
<code>arguments</code> Arguments	(Optional) Arguments passed to the Bash script.
<code>script</code> Script	(Required, if Type is inline) Contents of the script Default value: <code>"# Write your commands here\n\necho 'Hello world'\n"</code>
<code>workingDirectory</code> Working Directory	(Optional) 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>

ARGUMENT	DESCRIPTION
<code>failOnStderr</code> Fail on standard error	(Optional) If this is true, this task will fail if any errors are written to stderr. Default value: <code>false</code>
<code>noProfile</code> Don't load the system-wide startup/initialization files	(Optional) Don't load the system-wide startup file <code>/etc/profile</code> or any of the personal initialization files
<code>noRc</code> Don't read the <code>~/.bashrc</code> file	(Optional) If this is true, the task will not process <code>.bashrc</code> from the user's home directory. Default value: <code>true</code>
<code>env</code> Environment variables	(Optional) 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:

```
steps:
- task: Bash@3
 inputs:
 targetType: 'inline'
 script: echo $MYSECRET
 env:
 MYSECRET: $(Foo)
```

This is equivalent to:

```
steps:
- script: echo $MYSECRET
 env:
 MYSECRET: $(Foo)
```

The Bash task will find the first Bash implementation on your system. Running `which bash` on Linux/macOS or `where bash` on Windows will give you an idea of which one it'll select.

Bash scripts checked into the repo should be set executable (`chmod +x`). Otherwise, the task will show a warning and `source` the file instead.

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Batch Script task

3/25/2021 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

Use this task to run a Windows .bat or .cmd script. Optionally, allow it to permanently modify environment variables.

## NOTE

This task is not compatible with Windows containers. If you need to run a batch script on a Windows container, use the [command line task](#) instead.

For information on supporting multiple platforms, see [cross platform scripting](#).

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## YAML snippet

```
Batch script
Run a Windows command or batch 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
<code>filename</code> Path	(Required) Path of the cmd or bat script to execute. Should be fully qualified path or relative to the default working directory (please note that working directory could differ from 'workingFolder' which could be specified for this task).
<code>arguments</code> Arguments	(Optional) Specify arguments to pass to the script.
<code>modifyEnvironment</code> Modify environment	(Optional) Determines whether environment variable modifications will affect subsequent tasks Default value: <code>False</code>

ARGUMENT	DESCRIPTION
<code>workingFolder</code> Working folder	(Optional) Current working directory when script is run. Defaults to the agent's default working directory
<code>failOnStandardError</code> Fail on Standard Error	(Optional) If this is true, this task will fail if any errors are written to the StandardError stream. Default value: <code>false</code>

## 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:

 <b>Utility: Batch Script</b>	Run test.bat. <ul style="list-style-type: none"> <li>Path: <code>test.bat</code></li> </ul>
---------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### 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?

To learn more about defining build variables in a script, see [Define and modify your build variables in a script](#).

To learn more about defining release variables in a script, see [Define and modify your release variables in a script](#).

### Q: I'm having issues with publishing my artifacts. How can I view the detailed logs?

To enable detailed logs for your pipeline:

1. Edit your pipeline and select **Variables**
2. Add a new variable with the name `System.Debug` and value `true`
3. Save

### Q: Which variables are available to me?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just few of the variables you can use in your

pipeline. Variables are available as [expressions](#) or scripts.

See [Define variables, predefined variables](#), and [Classic release and artifacts variables](#) to learn about the different types of variables.

### **Do I need an agent?**

You need at least one [agent](#) to run your build or release.

### **I'm having problems. How can I troubleshoot them?**

See [Troubleshoot Build and Release](#).

### **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](#).

### **My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?**

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

### **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](#).

# Cache task

3/8/2021 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

Improve build performance by caching files, like dependencies, between pipeline runs.

## Demands

None

## YAML snippet

```
Cache
Cache files between runs
- task: Cache@2
 inputs:
 key:
 path:
 #cacheHitVar: # Optional
 #restoreKeys: # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
Key (unique identifier) for the cache	This should be a string that can be segmented using ' '. File paths can be absolute or relative to \$(System.DefaultWorkingDirectory).
Path of the folder to cache	Can be fully qualified or relative to \$(System.DefaultWorkingDirectory). Wildcards are not supported. <a href="#">Variables</a> are supported.
Cache hit variable	Variable to set to 'true' when the cache is restored (a cache hit), otherwise set to 'false'.
Additional restore key prefixes	Additional restore key prefixes that are used if the primary key misses. This can be a newline-delimited list of key prefixes.

## Control options

## Open-source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## Next steps

Learn more about pipeline caching:

[Pipeline Caching](#)

# Command Line task

3/8/2021 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

Use this task to run a program from the command prompt.

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## Demands

None

## YAML snippet

```
Command line
Run a command line script using Bash on Linux and macOS and cmd.exe on Windows
- 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: { string: string } # mapping of environment variables to add
```

## Running batch and .CMD files

Azure Pipelines puts your inline script contents into a temporary batch file (.cmd) in order to run it. When you want to run a batch file from another batch file in Windows CMD, you must use the `call` command, otherwise the first batch file is terminated. This will result in Azure Pipelines running your intended script up until the first batch file, then running the batch file, then ending the step. Additional lines in the first script wouldn't be run. You should always prepend `call` before executing a batch file in an Azure Pipelines script step.

## IMPORTANT

You may not realize you're running a batch file. For example, `npm` on Windows, along with any tools that you install using `npm install -g`, are actually batch files. Always use `call npm <command>` to run NPM commands in a Command Line task on Windows.

# Arguments

ARGUMENT	DESCRIPTION
<code>script</code> Script	(Required) Contents of the script you want to run Default value: <code>echo Write your commands here\n\neco Hello world\n"</code>
<code>workingDirectory</code> Working directory	(Optional) 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> .
<code>failOnStderr</code> Fail on Standard Error	If this is true, this task will fail if any errors are written to stderr
<code>env</code> Environment variables	(Optional) 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 as shown in the following example.

```
- script: echo %MYSECRET%
env:
 MySecret: $(Foo)
```

## Example

- [YAML](#)
- [Classic](#)

```
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
env:
 aVarFromYaml: someValue
```

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### 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.

**I'm having problems. How can I troubleshoot them?**

See [Troubleshoot Build and Release](#).

**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](#).

**My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?**

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE.EXTRA.CA.CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

**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

3/6/2021 • 2 minutes to read • [Edit Online](#)

## TFS 2015

Use this task 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

This task is deprecated. If you're using Team Foundation Server 2017 or newer, we recommend that you use [Pipeline Artifacts](#).

### IMPORTANT

This task is deprecated. If you're using Team Foundation Server 2017 or newer, we recommend that you use [Pipeline Artifacts](#).

Use this task only if you're using Team Foundation Server (TFS) 2015 RTM. You can find this task under the **Build** category **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"><li>• <code>**</code> copies all files in the root folder.</li><li>• <code>**\*</code> copies all files in the root folder and all files in all sub-folders.</li><li>• <code>**\bin</code> copies files in any sub-folder named bin.</li></ul>
Artifact Name	Specify the name of the artifact. For example: <code>drop</code>

ARGUMENT	DESCRIPTION
Artifact Type	Choose <b>server</b> to store the artifact on your Team Foundation Server. This is the best and simplest option in most cases. See <a href="#">Artifacts in Azure Pipelines</a> .
CONTROL OPTIONS	

## FAQ

**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 issues with publishing my artifacts. How can I view the detailed logs?**

To enable detailed logs for your pipeline:

1. Edit your pipeline and select **Variables**
2. Add a new variable with the name `System.Debug` and value `true`
3. **Save**

**Q: Which variables are available to me?**

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just few of the variables you can use in your pipeline. Variables are available as [expressions](#) or scripts.

See [Define variables](#), [predefined variables](#), and [Classic release and artifacts variables](#) to learn about the different types of 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

3/8/2021 • 4 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015.3

Use this task to copy files from a source folder to a target folder using match patterns.

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## Demands

None

## YAML snippet

```
Copy files
Copy files from a source folder to a target folder using patterns matching file paths (not folder paths)
- task: CopyFiles@2
 inputs:
 #sourceFolder: # Optional
 #contents: '**'
 targetFolder:
 #cleanTargetFolder: false # Optional
 #overWrite: false # Optional
 #flattenFolders: false # Optional
 #preserveTimestamp: false # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<code>SourceFolder</code> Source Folder	(Optional) 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> ). 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.

ARGUMENT	DESCRIPTION
<code>Contents</code> Contents	<p>(Required) File paths to include as part of the copy. Supports multiple lines of match patterns.</p> <p>For example:</p> <ul style="list-style-type: none"> <li>• <code>*</code> copies all files in the specified source folder</li> <li>• <code>**</code> copies all files in the specified source folder and all files in all sub-folders</li> <li>• <code>**\bin\**</code> copies all files recursively from any bin folder</li> </ul> <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>. You must use the path separator that matches your build agent type. <b>Example</b>, <code>/</code> must be used for Linux agents. More examples are shown below.</p> <p>Default value: <code>**</code></p>
<code>TargetFolder</code> Target Folder	<p>(Required) Target folder or UNC path files will copy to. You can use <a href="#">variables</a>.</p> <p>Example: <code>\$(build.artifactstagingdirectory)</code></p>
<code>CleanTargetFolder</code> Clean Target Folder	<p>(Optional) Delete all existing files in target folder before copy</p> <p>Default value: <code>false</code></p>
<code>OverWrite</code> Overwrite	<p>(Optional) Replace existing files in target folder</p> <p>Default value: <code>false</code></p>
<code>flattenFolders</code> Flatten Folders	<p>(Optional) Flatten the folder structure and copy all files into the specified target folder</p> <p>Default value: <code>false</code></p>
<code>preserveTimestamp</code> Preserve Target Timestamp	<p>(Optional) Using the original source file, preserve the target file timestamp.</p> <p>Default value: <code>false</code></p>

## 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.

## Example

```

steps:
- 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](#)
- [Classic](#)

### 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)'
```

## Example with variables in content section

```
- task: CopyFiles@2
 inputs:
 Contents: '$(Build.Repository.LocalPath)/**'
 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](#)
- [Classic](#)

## 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.

## FAQ

### 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 issues with publishing my artifacts. How can I view the detailed logs?**

To enable detailed logs for your pipeline:

1. Edit your pipeline and select **Variables**
2. Add a new variable with the name `System.Debug` and value `true`
3. Save

**Q: Which variables are available to me?**

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just few of the variables you can use in your pipeline. Variables are available as [expressions](#) or scripts.

See [Define variables, predefined variables](#), and [Classic release and artifacts variables](#) to learn about the different types of variables.

### Do I need an agent?

You need at least one [agent](#) to run your build or release.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

### 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

6/2/2020 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

Use this task 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's supported protocols 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
<code>files</code> Files	(Required) File(s) to be uploaded. Wildcards can be used. For example, <code>**/*.zip</code> for all ZIP files in all subfolders
<code>authType</code> Authentication Method	Default value: <code>ServiceEndpoint</code>
<code>serviceEndpoint</code> Service Connection	(Required) The service connection with the credentials for the server authentication. Use the Generic service connection type for the service connection
<code>username</code> Username	(Optional) Specify the username for server authentication.
<code>password</code> Password	(Optional) Specify the password for server authentication. <b>Important:</b> Use a <a href="#">secret variable</a> to avoid exposing this value

ARGUMENT	DESCRIPTION
<code>url</code> URL	(Required) 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.  Acceptable URL protocols include DICT:///, FILE:///, FTP:///, FTPS:///, GOPHER:///, HTTP:///, HTTPS:///, IMAP:///, IMAPS:///, LDAP:///, LDAPS:///, POP3:///, POP3S:///, RTMP:///, RTSP:///, SCP:///, SFTP:///, SMTP:///, SMTPS:///, TELNET:///, and <code>TFTP://</code>
<code>remotePath</code> Remote Directory	(Optional) If supplied, this is the sub-folder on the remote server for the URL supplied in the credentials Default value: <code>upload/\${Build.BuildId}/</code>
<code>options</code> Optional Arguments	(Optional) Arguments to pass to cURL.
<code>redirectStderr</code> Redirect Standard Error to Standard Out	Adds <code>--stderr -</code> as an argument to cURL. By default, cURL writes its progress bar to stderr, which is interpreted by the build as error output. Enabling this checkbox suppresses that behavior Default value: <code>true</code>

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### 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.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

### 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

4/10/2020 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task to decrypt files using OpenSSL.

## YAML snippet

```
Decrypt file (OpenSSL)
Decrypt a file using OpenSSL
- task: DecryptFile@1
 inputs:
 cipher: 'des3'
 inFile:
 passphrase:
 outFile: # Optional
 #workingDirectory: # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<code>cipher</code> Cypher	(Required) Encryption cypher to use. See <a href="#">cipher suite names</a> for a complete list of possible values Default value: <code>des3</code>
<code>inFile</code> Encrypted file	(Required) Relative path of file to decrypt.
<code>passphrase</code> Passphrase	(Required) Passphrase to use for decryption. Use a Variable to encrypt the passphrase.
<code>outFile</code> Decrypted file path	(Optional) Optional filename for decrypted file. Defaults to the Encrypted File with a ".out" extension
<code>cwd</code> Working directory	(Optional) Working directory for decryption. Defaults to the root of the repository Argument aliases: <code>workingDirectory</code>

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Delay task

6/2/2020 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task in an [agentless job](#) of a 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 a workflow by a fixed time
jobs:
- job: RunsOnServer
 pool: Server
 steps:
- task: Delay@1
 inputs:
 #delayForMinutes: '0'
```

## Arguments

ARGUMENTS	DESCRIPTION
<code>delayForMinutes</code> Delay Time (minutes)	(Required) Delay the execution of the workflow by specified time in minutes. 0 value means that workflow execution will start without delay Default value: <code>0</code>

Also see this task on [GitHub](#).

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### Do I need an agent?

You need at least one [agent](#) to run your build or release.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE.EXTRA.CA.CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

# Delete Files task

5/18/2021 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015.3

Use this task to delete files or folders from the agent working directory.

## Demands

None

## YAML snippet

```
Delete files
Delete folders, or files matching a pattern
- task: DeleteFiles@1
 inputs:
 #SourceFolder: # Optional
 #Contents: 'myFileShare'
 #RemoveSourceFolder: # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<code>SourceFolder</code> Source Folder	(Optional) Folder that contains the files you want to delete. If you leave it empty, the deletions are done from the root folder of the repository (same as if you had specified <code>\$(Build.SourcesDirectory)</code> ). If your build produces artifacts outside of the sources directory, specify <code>\$(Agent.BuildDirectory)</code> to delete files from the build agent working directory.
<code>Contents</code> Contents	(Required) File/folder paths to delete. Supports multiple lines of minimatch patterns; each one is processed before moving onto the next line. <a href="#">More Information</a> . For example: <ul style="list-style-type: none"><li>• <code>**/*</code> deletes all files and folders in the root folder.</li><li>• <code>temp</code> deletes the <code>temp</code> folder in the root folder.</li><li>• <code>temp*</code> deletes any file or folder in the root folder with a name that begins with <code>temp</code>.</li><li>• <code>**/temp/*</code> deletes all files and folders in any sub-folder named <code>temp</code>.</li><li>• <code>**/temp*</code> deletes any file or folder with a name that begins with <code>temp</code>.</li><li>• <code>!(*.vsix)</code> deletes all files in the root folder that do not have a <code>.vsix</code> extension.</li></ul>

ARGUMENT	DESCRIPTION
<code>RemoveSourceFolder</code> Remove SourceFolder	(Optional) Attempt to remove the source folder after attempting to remove <code>Contents</code> . Default value: <code>false</code> . If you want to remove the whole folder, set this to <code>true</code> and set <code>Contents</code> to <code>*</code> .
<b>Advanced</b>	
<code>RemoveDotFiles</code> Remove DotFiles	(Optional) Delete files starting with a dot (.git, .dockerfile). Omits these files if it's not specified explicitly (for example, <code>'/*'</code> ). <a href="#">More information</a> . Default value: <code>false</code> .

## Examples

### Delete several patterns

This example will delete `some/file`, all files beginning with `test`, and all files in all subdirectories called `bin`.

```
steps:
- task: DeleteFiles@1
 displayName: 'Remove unneeded files'
 inputs:
 contents: |
 some/file
 test*
 **/bin/*
```

### Delete all but one subdirectory

This example will delete `some/one`, `some/three` and `some/four` but will leave `some/two`.

```
steps:
- task: DeleteFiles@1
 displayName: 'Remove unneeded files'
 inputs:
 contents: |
 some/!(two)
```

### Delete using brace expansion

This example will delete `some/one` and `some/four` but will leave `some/two` and `some/three`.

```
steps:
- task: DeleteFiles@1
 displayName: 'Remove unneeded files'
 inputs:
 contents: |
 some/{one,four}
```

### Delete files starting with a dot

This example will delete all `.txt` files. Files starting with a dot will be deleted as well.

```
steps:
- task: DeleteFiles@1
 displayName: 'Remove unneeded files'
 inputs:
 contents: |
 /some/*.txt
 removeDotFiles: true
```

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### Q: What's a minimatch pattern? How does it work?

A: See:

- <https://github.com/isaacs/minimatch>
- <https://realguess.net/tags/minimatch/>
- <http://man7.org/linux/man-pages/man3/fnmatch.3.html>

### Q: I'm having issues with publishing my artifacts. How can I view the detailed logs?

To enable detailed logs for your pipeline:

1. Edit your pipeline and select **Variables**
2. Add a new variable with the name `System.Debug` and value `true`
3. Save

### Q: Which variables are available to me?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just few of the variables you can use in your pipeline. Variables are available as [expressions](#) or scripts.

See [Define variables](#), [predefined variables](#), and [Classic release and artifacts variables](#) to learn about the different types of variables.

### Do I need an agent?

You need at least one [agent](#) to run your build or release.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

### 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

11/2/2020 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task to download build artifacts.

## YAML snippet

```
Download build artifacts
Download files that were saved as artifacts of a completed build
- 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
 #allowPartiallySucceededBuilds: false # Optional
 #branchName: 'refs/heads/master' # Required when buildType == Specific && BuildVersionToDownload ==
 LatestFromBranch
 #buildId: # Required when buildType == Specific && BuildVersionToDownload == Specific
 #tags: # Optional
 #downloadType: 'single' # Choose whether to download a single artifact or all artifacts of a specific
 build. Options: single, specific
 #artifactName: # Required when downloadType == Single
 #itemPattern: '**' # Optional
 #downloadPath: '$(System.ArtifactsDirectory)'
 #parallelizationLimit: '8' # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<code>buildType</code> Download artifacts produced by	(Required) Download artifacts produced by the current build, or from a specific build Default value: <code>current</code>
<code>project</code> Project	(Required) The project from which to download the build artifacts
<code>definition</code> Build pipeline	(Required) Select the build pipeline name Argument aliases: <code>pipeline</code>
<code>specificBuildWithTriggering</code> When appropriate, download artifacts from the triggering build	(Optional) If true, 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. Default value: <code>false</code>

ARGUMENT	DESCRIPTION
<code>buildVersionToDownload</code> Build version to download	(Required) Specify which version of the build to download. <ul style="list-style-type: none"> <li>Choose <code>latest</code> to download the latest available build version</li> <li>Choose <code>latestFromBranch</code> to download the latest available build version of the branch specified by <code>branchName</code> and tags specified by <code>tags</code></li> <li>Choose <code>specific</code> to download the build version specified by <code>buildId</code></li> </ul>
<code>allowPartiallySucceededBuilds</code> Download artifacts even from partially succeeded builds	(Optional) If checked, this build task will try to download artifacts whether the build is succeeded or partially succeeded Default value: <code>false</code>
<code>branchName</code> Branch name	(Required) Specify to filter on branch/ref name. Default value: <code>refs/heads/develop</code>
<code>buildId</code> Build	(Required) The build from which to download the artifacts
<code>tags</code> Build tags	(Optional) A comma-delimited list of tags. Only builds with these tags will be returned.
<code>downloadType</code> Download type	(Required) Choose whether to download a single artifact or all artifacts of a specific build. Default value: <code>single</code>
<code>artifactName</code> Artifact name	(Required) The name of the artifact to download
<code>itemPattern</code> Matching pattern	(Optional) Specify files to be downloaded as multi-line minimatch pattern. <a href="#">More Information</a> . The default pattern will download all files across all artifacts in the build if the <code>Specific files</code> option is selected. To download all files within an artifact drop use <code>drop/</code> Default value: <code>\*\*</code>
<code>downloadPath</code> Destination directory	(Required) Path on the agent machine where the artifacts will be downloaded Default value: <code>\$(System.ArtifactsDirectory)</code>
<code>parallelizationLimit</code> Parallelization limit	(Optional) Number of files to download simultaneously Default value: <code>8</code>

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Download Fileshare Artifacts task

11/2/2020 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task to download fileshare artifacts.

## YAML snippet

```
Download artifacts from file share
Download artifacts from a file share, like \\share\drop
- task: DownloadFileshareArtifacts@1
 inputs:
 filePath:
 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. <a href="#">More Information</a> . 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.
<b>CONTROL OPTIONS</b>	

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Download GitHub Release task

3/6/2021 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task in your pipeline to download assets from your [GitHub release](#) as part of your CI/CD pipeline.

## Prerequisites

### GitHub service connection

This task requires a [GitHub service connection](#) with **Read** 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

```
Download GitHub Release
Downloads a GitHub Release from a repository
- task: DownloadGitHubRelease@0
 inputs:
 connection: GitHub Connection
 userrepository: Repository
 defaultVersionType: 'latest' # Options: latest, specificVersion, specificTag
 #version: # Required when defaultVersionType != Latest
 #itemPattern: '**' # Optional
 #downloadPath: '$(System.ArtifactsDirectory)'
```

## Arguments

ARGUMENT	DESCRIPTION
<b>connection</b> GitHub Connection	(Required) Enter the service connection name for your GitHub connection. Learn more about service connections <a href="#">here</a> .
<b>userrepository</b> Repository	(Required) Select the name of GitHub repository in which GitHub releases will be downloaded from.
<b>defaultVersionType</b> Default version	(Required) The version of the GitHub Release from which the assets are downloaded. The version type can be 'Latest Release', 'Specific Version' or 'Specific Tag' Default value: <code>latest</code>
<b>version</b> Release	(Required) This option shows up if 'Specific Version' or 'Specific Tag' is selected as Default Release version type. Based on the version type selected, Release name or the Tag needs to be provided.
<b>itemPattern</b> Item pattern	(Optional) Minimatch pattern to filter files to be downloaded from the available release assets. To download all files within release use <code>**</code> .

ARGUMENT	DESCRIPTION
<code>downloadPath</code> Destination directory	(Required) Path on the agent machine where the release assets will be downloaded. Default value: <code>\$(System.ArtifactsDirectory)</code>

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Download Package task

11/2/2020 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task to download a package from a package management feed in Azure Artifacts or TFS. Requires the Package Management extension.

## YAML snippet

```
Download package
Download a package from a package management feed in Azure Artifacts
- task: DownloadPackage@1
 inputs:
 packageType: # 'nuget' Options: maven, npm, nuget, pypi, upack
 feed: # <feedId> for organization-scoped feeds, <projectId>/<feedId> for project-scoped feeds.
 #view: '' # Optional
 definition: # '$(packageName)'
 version: # '1.0.0'
 #files: '**' # Optional
 #extract: true # Optional
 downloadPath: # '$(System.ArtifactsDirectory)'
```

## Arguments

ARGUMENT	DESCRIPTION
PackageType	(Required) Select the type of package to download.
Feed	(Required) ID of the feed that contains the package. For project-scoped feeds, the format is projectId/feedID. See our <a href="#">FAQ</a> below for information on how to get a feed or project ID, or information on using project and feed name instead.
View	(Optional) Select a view to see package versions only promoted to that view.
Definition	(Required) Select the package to download. This can be the artifact ID or the package name.
Version	(Required) Version of the package.
Files	(Optional) Specify files to be downloaded as multiline minimatch patterns. <a href="#">More Information</a> . The default pattern (** will download all files within the artifact.
Extract	(Optional) Specify whether to extract the package contents at the destination directory.
DownloadPath	(Required) Path on the agent machine where the package will be downloaded.

## Examples

### Download a NuGet package from an organization-scoped feed and extract to destination directory

```
Download an artifact with id 'cf01b64-ded4-47b7-a569-2ac17cbcedbd' to $(System.ArtifactsDirectory)
- task: DownloadPackage@1
 inputs:
 packageType: 'nuget'
 feed: '6a60ef3b-e29f-41b6-9885-7874278baac7'
 definition: 'cf01b64-ded4-47b7-a569-2ac17cbcedbd' # Can also be package name
 version: '1.0.0'
 extract: true
 downloadPath: '$(System.ArtifactsDirectory)'
```

### Download a maven package from a project-scoped feed and download only pom files.

```
Download an artifact with name 'com.test:testpackage' to $(System.ArtifactsDirectory)
- task: DownloadPackage@1
 inputs:
 packageType: 'maven'
 feed: '132f5c2c-2aa0-475a-8b47-02c79617954b/c85e5de9-7b12-4cf8-9293-1b33cdff540e' # <projectId>/<feedId>
 definition: 'com.test:testpackage'
 version: '1.0.0-snapshot' # Should be normalized version
 files: '*.pom'
 downloadPath: '$(System.ArtifactsDirectory)'
```

## FAQ

### How do I find the ID of the feed (or project) I want to download my artifact from

The get feed api can be used to retrieve the feed and project ID for your feed. The api is documented [here](#).

### Can I use the project or feed name instead of IDs

Yes, you can use the project or feed name in your definition, however if your project or feed is renamed in the future, your task will also have to be updated or it might fail.

## Open-source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Download Pipeline Artifacts task

6/9/2021 • 3 minutes to read • [Edit Online](#)

Use this task to download pipeline artifacts from earlier stages in this pipeline, or from another pipeline.

## NOTE

For more information, including Azure CLI commands, see [downloading artifacts](#).

## YAML snippet

```
Download pipeline artifacts
Download build and pipeline artifacts
- task: DownloadPipelineArtifact@2
 inputs:
 #source: 'current' # Options: current, specific
 #project: # Required when source == Specific
 #pipeline: # Required when source == Specific
 #preferTriggeringPipeline: false # Optional
 #runVersion: 'latest' # Required when source == Specific# Options: latest, latestFromBranch, specific
 #runBranch: 'refs/heads/master' # Required when source == Specific && RunVersion == LatestFromBranch
 #runId: # Required when source == Specific && RunVersion == Specific
 #tags: # Optional
 #artifact: # Optional
 #patterns: '**' # Optional
 #path: '$(Pipeline.Workspace)'
```

## NOTE

The `publish` and `download` keywords are shortcuts for the **Publish Pipeline Artifact** task. You can use them in your pipeline to publish and download artifacts. For more information, see [Publish](#) and [Download](#) in the YAML schema.

## Arguments

ARGUMENT	DESCRIPTION
<code>source</code> Download artifacts produced by	(Required) Download artifacts produced by the current pipeline run, or from a specific pipeline run. Options: <code>current</code> , <code>specific</code> Default value: <code>current</code> Argument aliases: <code>buildType</code>
<code>project</code> Project	(Required) The project name or GUID from which to download the pipeline artifacts.
<code>pipeline</code> Build Pipeline	(Required) The definition ID of the build pipeline. Argument aliases: <code>definition</code>

ARGUMENT	DESCRIPTION
<code>preferTriggeringPipeline</code> When appropriate, download artifacts from the triggering build	(Optional) A boolean specifying whether to download artifacts from a triggering build. Default value: <code>false</code> Argument aliases: <code>specificBuildWithTriggering</code>
<code>runVersion</code> Build version to download	(Required) Specifies which build version to download. Options: <code>latest</code> , <code>latestFromBranch</code> , <code>specific</code> Default value: <code>latest</code> Argument aliases: <code>buildVersionToDownload</code>
<code>runBranch</code> Branch Name	Specify to filter on branch/ref name. For example: <code>refs/heads/develop</code> . Default value: <code>refs/heads/master</code> Argument aliases: <code>branchName</code>
<code>runId</code> Build	(Required) The build from which to download the artifacts. For example: <code>1764</code> Argument aliases: <code>pipelineId</code> , <code>buildId</code>
<code>tags</code> Build Tags	(Optional) A comma-delimited list of tags. Only builds with these tags will be returned.
<code>allowPartiallySucceededBuilds</code> Download artifacts from partially succeeded builds	(Optional) If checked, this build task will try to download artifacts whether the build is succeeded or partially succeeded Default value: <code>false</code>
<code>allowFailedBuilds</code> Download artifacts from failed builds	(Optional) If checked, this build task will try to download artifacts whether the build is succeeded or failed Default value: <code>false</code>
<code>artifact</code> Artifact Name	(Optional) The name of the artifact to download. If left empty, all artifacts associated to the pipeline run will be downloaded. Argument aliases: <code>artifactName</code>
<code>patterns</code> Matching Patterns	(Optional) One or more file matching patterns (new line delimited) that limit which files get downloaded. <a href="#">More Information on file matching patterns</a> Default value: <code>**</code> Argument aliases: <code>itemPattern</code>
<code>path</code> Destination Directory	(Required) Directory to download the artifact files. Can be relative to the pipeline workspace directory or absolute. If multi-download option is applied (by leaving an empty artifact name), a sub-directory will be created for each. See <a href="#">Artifacts in Azure Pipelines</a> . Default value: <code>\$(Pipeline.Workspace)</code> Argument aliases: <code>targetPath</code> , <code>downloadPath</code>

#### NOTE

If you want to consume artifacts as part of CI/CD flow, refer to the download shortcut [here](#).

# Examples

## Download a specific artifact

```
Download an artifact named 'WebApp' to 'bin' in $(Build.SourcesDirectory)
- task: DownloadPipelineArtifact@2
 inputs:
 artifact: 'WebApp'
 path: $(Build.SourcesDirectory)/bin
```

## Download artifacts from a specific project/pipeline

```
Download artifacts from a specific pipeline.
- task: DownloadPipelineArtifact@2
 inputs:
 source: 'specific'
 project: 'FabrikamFiber'
 pipeline: 12
 runVersion: 'latest'
```

## Download artifacts from a specific branch

```
Download artifacts from a specific branch with a tag
- task: DownloadPipelineArtifact@2
 inputs:
 source: 'specific'
 project: 'FabrikamFiber'
 pipeline: 12
 runVersion: 'latestFromBranch'
 runBranch: 'refs/heads/master'
 tags: 'testTag'
```

## Download an artifact from a specific build run

```
Download an artifact named 'WebApp' from a specific build run to 'bin' in $(Build.SourcesDirectory)
- task: DownloadPipelineArtifact@2
 inputs:
 source: 'specific'
 artifact: 'WebApp'
 path: $(Build.SourcesDirectory)/bin
 project: 'FabrikamFiber'
 pipeline: 12
 runVersion: 'specific'
 runId: 40
```

# FAQ

## How can I find the ID of the Pipeline I want to download an artifact from?

You can find the ID of the pipeline in the 'Pipeline variables'. The pipeline ID is the [system.definitionId](#) variable.

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Download Secure File task

3/6/2021 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task in a pipeline to download a [secure file](#) to the agent machine. When specifying the name of the file (using the `secureFile` input) use the name you specified when uploading it rather than the actual filename.

Once downloaded, use the `name` value that is set on the task (or "Reference name" in the classic editor) to reference the path to the secure file on the agent machine. For example, if the task is given the name `mySecureFile`, its path can be referenced in the pipeline as `$(mySecureFile.secureFilePath)`. Alternatively, downloaded secure files can be found in the directory given by `$(Agent.TempDirectory)`. See a full example [below](#).

When the pipeline job completes, no matter whether it succeeds, fails, or is canceled, the secure file is deleted from its download location.

It is unnecessary to use this task with the [Install Apple Certificate](#) or [Install Apple Provisioning Profile](#) tasks because they automatically download, install, and delete (at the end of the pipeline job) the secure file.

This task currently supports only one file task per instance.

## YAML snippet

```
Download secure file
Download a secure file to the agent machine
- task: DownloadSecureFile@1
 name: mySecureFile # The name with which to reference the secure file's path on the agent, like
 $(mySecureFile.secureFilePath)
 inputs:
 secureFile: # The file name or GUID of the secure file
 #retryCount: 5 # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<code>secureFile</code> Secure File	(Required) The file name or unique identifier (GUID) of the secure file to download to the agent machine. The file will be deleted when the pipeline job completes.
<code>retryCount</code> Retry Count	(Optional) Number of times to retry downloading a secure file if the download fails. Default value: 8

## Example

This example downloads a secure certificate file and installs it to a trusted certificate authority (CA) directory on Linux:

```
- task: DownloadSecureFile@1
 name: caCertificate
 displayName: 'Download CA certificate'
 inputs:
 secureFile: 'myCACertificate.pem'

- script: |
 echo Installing $(caCertificate.secureFilePath) to the trusted CA directory...
 sudo chown root:root $(caCertificate.secureFilePath)
 sudo chmod a+r $(caCertificate.secureFilePath)
 sudo ln -s -t /etc/ssl/certs/ $(caCertificate.secureFilePath)
```

# Extract Files task

3/25/2021 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

Use this task 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
 #overwriteExistingFiles: false
 #pathToSevenZipTool:
```

## Arguments

ARGUMENT	DESCRIPTION
Archive file patterns	<p>Patterns to match the archives you want to extract. By default, patterns start in the root folder of the repo (same as if you had specified <code>\$(Build.SourcesDirectory)</code> ).</p> <p>Specify pattern filters, one per line, that match the archives to extract. For example:</p> <ul style="list-style-type: none"><li>• <code>test.zip</code> extracts the test.zip file in the root folder.</li><li>• <code>test/*.zip</code> extracts all .zip files in the test folder.</li><li>• <code>**/*.tar</code> extracts all .tar files in the root folder and sub-folders.</li><li>• <code>**/bin/*.7z</code> extracts all ".7z" files in any sub-folder named bin.</li></ul> <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> ).

ARGUMENT	DESCRIPTION
Clean destination folder before extracting	Select this check box to delete all existing files in the destination folder before beginning to extract archives.
Overwrite files in the destination directory	Select this option to overwrite files in the output directory if they already exist.
Path to 7z utility	You can specify custom path to 7z utility using this option. If it's not specified on Windows - default 7zip version supplied with a task will be used.

#### CONTROL OPTIONS

## Examples

### Extract all .zip files recursively

This example will extract all .zip files recursively, including both root files and files from sub-folders

```
steps:
- task: ExtractFiles@1
 inputs:
 archiveFilePatterns: '**/*.zip'
 cleanDestinationFolder: true
 overwriteExistingFiles: false
```

### Extract all .zip files from subfolder

This example will extract `test/one.zip`, `test/two.zip` but will leave `test/nested/three.zip`.

```
steps:
- task: ExtractFiles@1
 inputs:
 archiveFilePatterns: 'test/*.zip'
 cleanDestinationFolder: true
 overwriteExistingFiles: false
```

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### Where can I learn more about file matching patterns?

[File matching patterns reference](#)

### Q: I'm having issues with publishing my artifacts. How can I view the detailed logs?

To enable detailed logs for your pipeline:

1. Edit your pipeline and select **Variables**
2. Add a new variable with the name `System.Debug` and value `true`
3. **Save**

### Q: Which variables are available to me?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just few of the variables you can use in your

pipeline. Variables are available as [expressions](#) or scripts.

See [Define variables, predefined variables](#), and [Classic release and artifacts variables](#) to learn about the different types of variables.

### **Do I need an agent?**

You need at least one [agent](#) to run your build or release.

### **I'm having problems. How can I troubleshoot them?**

See [Troubleshoot Build and Release](#).

### **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](#).

### **My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?**

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

### **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](#).

# File Transform task

3/6/2021 • 3 minutes to read • [Edit Online](#)

Use this task to apply file transformations and variable substitutions on configuration and parameters files. For details of how translations are processed, see [File transforms and variable substitution reference](#).

## IMPORTANT

This task is intended for web packages and requires a web package file, and does not work on standalone json files.

## File transformations

- At present file transformations are supported for only XML files.
- To apply XML transformation to configuration files (\*.config) you must specify a newline-separated list of transformation file rules using the syntax:

```
-transform <path to the transform file> -xml <path to the source file> -result <path to the result file>
```
- File transformations are useful in many scenarios, particularly when you are deploying to an App service and want to add, remove or modify configurations for different environments (such as Dev, Test, or Prod) by following the standard [Web.config Transformation Syntax](#).
- You can also use this functionality to transform other files, including Console or Windows service application configuration files (for example, FabrikamService.exe.config).
- Config file transformations are run before variable substitutions.

## Variable substitution

- At present only XML and JSON file formats are supported for variable substitution.
- Tokens defined in the target configuration files are updated and then replaced with variable values.
- Variable substitutions are run after config file transformations.
- Variable substitution is applied for only the JSON keys predefined in the object hierarchy. It does not create new keys.

## Examples

If you need XML transformation to run on all the configuration files named with pattern `.Production.config`, the transformation rule should be specified as:

```
-transform ***.Production.config -xml ***.config
```

If you have a configuration file named based on the stage name in your pipeline, you can use:

```
-transform ***.$(Release.EnvironmentName).config -xml ***.config
```

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 must define a variable as `Data.DefaultConnection.ConnectionString` in the build or release pipeline (or in a stage within the release pipeline).

```
{
 "Data": {
 "DefaultConnection": {
 "ConnectionString": "Server=(localdb)\SQLEXPRESS;Database=MyDB;Trusted_Connection=True"
 }
 }
}
```

### NOTE

Only custom variables defined in build and release pipelines are used in substitution. Default and system pipeline variables are excluded.

Here's a list of currently excluded prefixes:

- 'agent.'
- 'azure\_http\_user\_agent'
- 'build.'
- 'common.'
- 'release.'
- 'system.'
- 'tf\_'

If the same variables are defined in both the release pipeline and in a stage, the stage-defined variables supersede the pipeline-defined variables.

See also: [File transforms and variable substitution reference](#).

## Demands

None

## YAML snippet

```
File transform
Replace tokens with variable values in XML or JSON configuration files
- task: FileTransform@1
 inputs:
 #folderPath: '$(System.DefaultWorkingDirectory)/**/*.zip'
 #enableXmlTransform: # Optional
 #xmlTransformationRules: '-transform ***.Release.config -xml ****.config-transform
 .$(Environment.Name).config -xml *.config' # Optional
 #fileType: # Optional. Options: xml, json
 #targetFiles: # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
Package or folder FolderPath	File path to the package or a folder. Variables ( Build   Release ), wildcards are supported. For example, <code>\$(System.DefaultWorkingDirectory)/*/.zip</code> . For zipped folders, the contents are extracted to the TEMP location, transformations executed, and the results zipped in original artifact location.

ARGUMENT	DESCRIPTION
XML transformation enableXmlTransform	Enable this option to apply XML transformations based on the rules specified below. Config transforms run prior to any variable substitution. XML transformations are supported only for the Windows platform.
Transformation rules xmlTransformationRules	<p>Provide a newline-separated list of transformation file rules using the syntax</p> <pre>-transform &lt;path to the transform file&gt; -xml &lt;path to the source configuration file&gt; -result &lt;path to the result file&gt;</pre> <p>The result file path is optional and, if not specified, the source configuration file will be replaced with the transformed result file.</p>
File format fileType	Specify the file format on which substitution is to be performed. Variable substitution runs after any configuration transforms. For XML, Variables defined in the build or release pipelines will be matched against the token ('key' or 'name') entries in the appSettings, applicationSettings, and connectionStrings sections of any config file and parameters.xml file.
Target files targetFiles	Provide a newline-separated list of files for variable substitution. Files names must be specified relative to the root folder.

## Open source

This task is open source on [GitHub](#). Feedback and contributions are welcome.

# FTP Upload task

11/2/2020 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

Use this task to upload files to a remote machine using the File Transfer Protocol (FTP), or securely with FTPS.

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## Demands

None

## YAML snippet

```
FTP upload
Upload files using FTP
- task: FtpUpload@2
 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
 #preservePaths: false
 #trustSSL: false
```

## Arguments

ARGUMENT	DESCRIPTION
<code>credsType</code> Authentication Method	(Required) Use FTP service connection or enter connection credentials Default value: serviceEndpoint Argument aliases: <code>credentialsOption</code>

ARGUMENT	DESCRIPTION
<code>serverEndpoint</code> FTP Service Connection	(Required) 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, Example, <a href="ftp://server.example.com">ftp://server.example.com</a> , 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. For example, <code>ftps://server.example.com</code> . Connections to servers not supporting FTPS will fail if <code>ftps://</code> is specified.
<code>serverUrl</code> Server URL	(Required)
<code>username</code> Username	(Required)
<code>password</code> Password	(Required)
<code>rootFolder</code> Root folder	(Required) The source folder to upload files from Argument aliases: <code>rootDirectory</code>
<code>filePatterns</code> File patterns	(Required) File paths or patterns of the files to upload. Supports multiple lines of minimatch patterns. <a href="#">More Information</a> . Default value: **
<code>remotePath</code> Remote directory	(Required) Upload files to this directory on the remote FTP server. Default value: /upload/\$(Build.BuildId)/ Argument aliases: <code>remoteDirectory</code>
<code>enableUtf8</code> Enable UTF8 support	(Optional) Enables UTF-8 support for the FTP connection ('OPTS UTF8 ON') Default value: false
<code>clean</code> Delete remote directory	(Required) Delete the remote directory including its contents before uploading Default value: false
<code>cleanContents</code> Clear remote directory contents	(Required) Recursively delete all contents of the remote directory before uploading. The existing directory will not be deleted. For better performance, consider using <code>Delete remote directory</code> instead Default value: false

ARGUMENT	DESCRIPTION
<code>preservePaths</code> Preserve file paths	(Required) If selected, the relative local directory structure is recreated under the remote directory where files are uploaded. Otherwise, files are uploaded directly to the remote directory without creating additional subdirectories. For example, suppose your source folder is: <code>/home/user/source/</code> and contains the file: <code>foo/bar/foobar.txt</code> , and your remote directory is: <code>/uploads/</code> . If selected, the file is uploaded to: <code>/uploads/foo/bar/foobar.txt</code> . Otherwise, to: <code>/uploads/foobar.txt</code> Default value: false
<code>trustSSL</code> Trust server certificate	(Required) 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). Default value: false
<code>customCmds</code> FTP Commands	(Optional) Optional FTP Commands that will be sent to the remote FTP server upon connection

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### Where can I learn more about file matching patterns?

[File matching patterns reference](#)

### Q: I'm having issues with publishing my artifacts. How can I view the detailed logs?

To enable detailed logs for your pipeline:

1. Edit your pipeline and select **Variables**
2. Add a new variable with the name `System.Debug` and value `true`
3. **Save**

### Q: Which variables are available to me?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just few of the variables you can use in your pipeline. Variables are available as [expressions](#) or scripts.

See [Define variables](#), [predefined variables](#), and [Classic release and artifacts variables](#) to learn about the different types of variables.

### Do I need an agent?

You need at least one [agent](#) to run your build or release.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE.EXTRA.CA.CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

**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 task

11/2/2020 • 4 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 delete a GitHub release
- task: GitHubRelease@0
 inputs:
 gitHubConnection:
 #repositoryName: '$(Build.Repository.Name)'
 #action: 'create' # Options: create, edit, delete
 #target: '$(Build.SourceVersion)' # Required when action == Create || Action == Edit
 #tagSource: 'auto' # Required when action == Create# Options: auto, manual
 #tagPattern: # Optional
 #tag: # Required when action == Edit || Action == Delete || 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
 #compareWith: 'lastFullRelease' # Required when addChangeLog == True. Options: lastFullRelease,
 lastRelease, lastReleaseByTag
 #releaseTag: # Required when compareWith == LastReleaseByTag
```

## Arguments

ARGUMENT	DESCRIPTION
GitHub Connection	(Required) Enter the service connection name for your GitHub connection. Learn more about service connections <a href="#">here</a> .
Repository	(Required) Select the name of GitHub repository in which GitHub releases will be created.

ARGUMENT	DESCRIPTION
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.
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	

# Examples

## Create a GitHub release

The following YAML creates a GitHub release every time the task runs. The build number is used as the tag version for the release. All .exe files and README.txt files in the \$(Build.ArtifactStagingDirectory) folder are uploaded as assets. By default, the task also generates a change log (a list of commits and issues that are part of this release) and publishes it as release notes.

```
- task: GithubRelease@0
 displayName: 'Create GitHub Release'
 inputs:
 gitHubConnection: zenithworks
 repositoryName: zenithworks/javaAppWithMaven
 tagSource: manual
 tag: $(Build.BuildNumber)
 assets: |
 $(Build.ArtifactStagingDirectory)/*.exe
 $(Build.ArtifactStagingDirectory)/README.txt
```

You can also control the creation of the release based on repository tags. The following YAML creates a GitHub release only when the commit that triggers the pipeline has a Git tag associated with it. The GitHub release is created with the same tag version as the associated Git tag.

```
- task: GithubRelease@0
 displayName: 'Create GitHub Release'
 inputs:
 gitHubConnection: zenithworks
 repositoryName: zenithworks/javaAppWithMaven
 assets: $(Build.ArtifactStagingDirectory)/*.exe
```

You may also want to use the task in conjunction with task conditions to get even finer control over when the task runs, thereby restricting the creation of releases. For example, in the following YAML the task runs only when the pipeline is triggered by a Git tag matching the pattern 'refs/tags/release-v\*'.

```
- task: GithubRelease@0
 displayName: 'Create GitHub Release'
 condition: startsWith(variables['Build.SourceBranch'], 'refs/tags/release-v')
 inputs:
 gitHubConnection: zenithworks
 repositoryName: zenithworks/javaAppWithMaven
 assets: $(Build.ArtifactStagingDirectory)/*.exe
```

## Edit a GitHub release

The following YAML updates the status of a GitHub release from 'draft' to 'published'. The release to be edited is determined by the specified tag.

```
- task: GithubRelease@0
 displayName: 'Edit GitHub Release'
 inputs:
 gitHubConnection: zenithworks
 repositoryName: zenithworks/javaAppWithMaven
 action: edit
 tag: $(myDraftReleaseVersion)
 isDraft: false
```

## Delete a GitHub release

The following YAML deletes a GitHub release. The release to be deleted is determined by the specified tag.

```
- task: GithubRelease@0
 displayName: 'Delete GitHub Release'
 inputs:
 gitHubConnection: zenithworks
 repositoryName: zenithworks/javaAppWithMaven
 action: delete
 tag: $(myDraftReleaseVersion)
```

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Install Apple Certificate task

4/10/2020 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018

Use this task 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 machine
- 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 <a href="#">Secure Files</a> 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 <a href="#">Variables</a> 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 <a href="#">Variables</a> 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

4/10/2020 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018

Use this task 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.

## Demand

xcode

## YAML snippet

```
Install Apple provisioning profile
Install an Apple provisioning profile required to build on a macOS agent machine
- task: InstallAppleProvisioningProfile@1
 inputs:
 #provisioningProfileLocation: 'secureFiles' # Options: secureFiles, sourceRepository
 #provProfileSecureFile: # Required when provisioningProfileLocation == SecureFiles
 #provProfileSourceRepository: # Required when provisioningProfileLocation == SourceRepository
 #removeProfile: true # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
Provisioning Profile Location ( <b>Azure Pipelines</b> )	Select the location of the provisioning profile to install. The provisioning profile can be uploaded to <b>Secure Files</b> or stored in your source repository or a local path on the agent.
Provisioning Profile	Select the provisioning profile that was uploaded to <b>Secure Files</b> 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

3/9/2021 • 3 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task in a pipeline to install an SSH key prior to a build or release step.

## YAML snippet

```
Install SSH key
Install an SSH key prior to a build or deployment
- task: InstallSSHKey@0
 inputs:
 knownHostsEntry:
 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	(Optional) The contents of the public SSH key.
SSH Passphrase	(Optional) The passphrase for the SSH key, if any.
SSH Key (Secure File)	(Required) Select the SSH key that was uploaded to <a href="#">Secure Files</a> to install on the agent.

### CONTROL OPTIONS

## Prerequisites

- GitBash for Windows

## Example setup using GitHub

This section describes how to use a private GitHub repository with YAML from within Azure Pipelines.

If you have a repository that you don't want to expose to the open-source community, a common practice is to make the repository private. However, a CI/CD tool like Azure DevOps needs access to the repository if you want to use the tool to manage the repository. To give Azure DevOps access, you might need an SSH key to authenticate access to GitHub.

Here are the steps to complete to use an SSH key to authenticate access to GitHub:

1. Generate a key pair to use to authenticate access from GitHub to Azure DevOps:
  - a. In GitBash, run the following command:

```
ssh-keygen -t rsa
```

- b. Enter a name for the SSH key pair. In our example, we use **myKey**.

```
Exampleuser MINGW64 ~/ssh
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Exampleuser/.ssh/id_rsa):
```

- c. (Optional) You can enter a passphrase to encrypt your private key. This step is optional. Using a passphrase is more secure than not using one.

```
Exampleuser MINGW64 ~/ssh
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Exampleuser/.ssh/id_rsa): myKey
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
```

The SSH key pairs are created and the following success message appears:

```
Exampleuser MINGW64 ~/ssh
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Exampleuser/.ssh/id_rsa): myKey
Enter passphrase (empty for no passphrase): ...
Enter same passphrase again: ...
Your identification has been saved in myKey.
Your public key has been saved in myKey.pub.
The key fingerprint is:
SHA256:TocLjss ...
The key's randomart image is:
+---[RSA 3072]---+
|o
|o
|o
|o
|=+
|o
|o
|o
|.
+---[SHA256]---
```

- d. In Windows File Explorer, check your newly created key pair:

This PC > Windows (C:) > Users > Exampleuser > .ssh				
	Name	Date modified	Type	Size
	keys	4/16/2020 6:18 PM	File	4 KB
	PE keys	4/16/2020 6:18 PM	Microsoft Publish...	1 KB

2. Add the public key to the GitHub repository. (The public key ends in ".pub"). To do this, go the following URL in your browser: [https://github.com/\(organization-name\)/\(repository-name\)/settings/keys](https://github.com/(organization-name)/(repository-name)/settings/keys).

- a. Select **Add deploy key**.

- b. In the **Add new** dialog box, enter a title, and then copy and paste the SSH key:

## Deploy keys / Add new

Title

Key

Begins with 'ssh-rsa', 'ssh-ed25519', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', or 'ecdsa-sha2-nistp521'

Allow write access

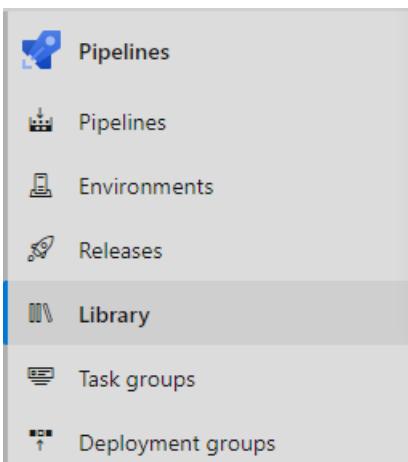
Can this key be used to push to this repository? Deploy keys always have pull access.

**Add key**

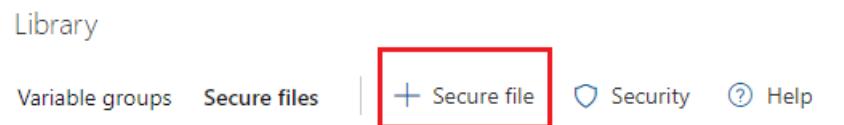
c. Select **Add key**.

3. Upload your private key to Azure DevOps:

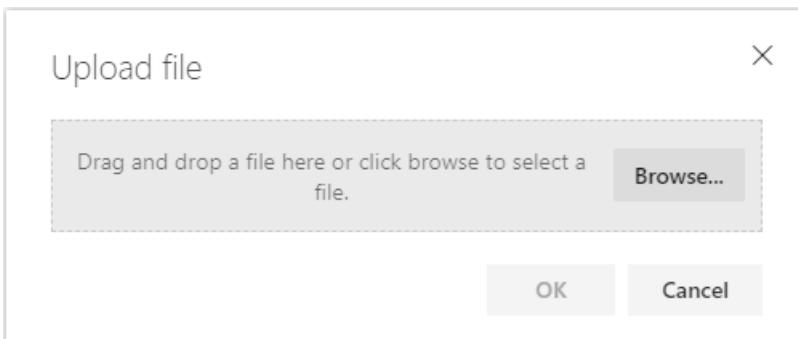
a. In Azure DevOps, in the left menu, select **Pipelines > Library**.



b. Select **Secure files > + Secure file**:



c. Select **Browse**, and then select your private key:



4. Recover your "Known Hosts Entry". In GitBash, enter the following command:

```
ssh-keyscan github.com
```

Your "Known Hosts Entry" is the displayed value that doesn't begin with # in the GitBash results:

```
Exampleuser MINGW64 ~/ssh
$ ssh-keyscan github.com
github.com:22 SSH-2.0-babeld-76c80caa
github.com ssh-rsa AAAAB3NzaC1yc2EAAQABIwAAAQEAq2A7hRG
XVa172J+UX2B+2RPW3RcT0eOzQqlJL3RKrTJvdsjE3JEAvGq3TGHS
github.com:22 SSH-2.0-babeld-76c80caa
github.com:22 SSH-2.0-babeld-76c80caa
```

## 5. Create a YAML pipeline.

To create a YAML pipeline, in the YAML definition, add the following task:

```
- task: InstallSSHKey@0
 inputs:
 knownHostsEntry: #{Enter your Known Hosts Entry Here}
 sshPublicKey: #{Enter your Public key Here}
 sshKeySecureFile: #{Enter the name of your key in "Secure Files" Here}
```

Now, the SSH keys are installed and you can proceed with the script to connect by using SSH, and not the default HTTPS.

## Usage and best practices

If you install an SSH key in the [hosted pools](#), in later steps in your pipeline, you can connect to a remote system in which the matching public key is already in place. For example, you can connect to a Git repository or to a VM in Azure.

We recommend that you don't pass in your public key as plain text to the task configuration. Instead, [set a secret variable](#) in your pipeline for the contents of your `mykey.pub` file. Then, call the variable in your pipeline definition as `$(myPubKey)`. For the secret part of your key, use the [Secure File library](#) in Azure Pipelines.

To create your task, use the following example of a well-configured Install SSH Key task:

```
steps:
- task: InstallSSHKey@0
 displayName: 'Install an SSH key'
 inputs:
 knownHostsEntry: 'SHA256:1Hyr55tsxGifESBMc0s+2NtutnR/4+LOkVwr0GrIp8U johndoe@contoso'
 sshPublicKey: '$(myPubKey)'
 sshKeySecureFile: 'id_rsa'
```

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Invoke a function task

3/6/2021 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

Use this task in an [agentless job](#) of a release pipeline to invoke an HTTP triggered function in a function app that is created and hosted in Azure Functions and parse the response.

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## Demands

Can be used in only an [agentless job](#) of a release pipeline.

## YAML snippet

```
Invoke Azure Function
Invoke an Azure Function
- 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
Function URL	Required. The URL of the 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 function call in JSON format.
Completion Event	Required. How the task reports completion. Can be <b>API response</b> (the default) - completion is when function returns success and success criteria evaluates to true, or <b>Callback</b> - the function makes a callback to update the timeline record.
Success criteria	Optional. How to parse the response body for success.
Control options	See <a href="#">Control options</a>

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](#).

## Open source

This task is open source on [GitHub](#). Feedback and contributions are welcome.

## FAQ

### Where should a task signal completion when Callback is chosen as the completion event?

To signal completion, the function should POST completion data to the following pipelines REST endpoint.

```
{planUri}/{projectId}/_apis/distributedtask/hubs/{hubName}/plans/{planId}/events?api-version=2.0-preview.1

Request Body
{ "name": "TaskCompleted", "taskId": "taskInstanceId", "jobId": "jobId", "result": "succeeded" }
```

See [this simple cmdline application](#) for specifics. In addition, a C# helper library is available to enable live logging and managing task status for agentless tasks. [Learn more](#)

### Why does the task failed within 1 minute when the timeout is longer?

If the function executes for more than 1 minute, use the **Callback** completion event. The API Response completion option is supported for requests that complete within 60 seconds.

# Invoke REST API task

7/2/2021 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018

Use this task to invoke an HTTP API and parse the response.

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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, 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)}'
 #body: # Required when method != GET && Method != HEAD
 #urlSuffix: # Optional
 #waitForCompletion: 'false' # Options: true, false
 #successCriteria: # Optional
```

## Arguments

PARAMETER	COMMENTS
Connection type	Required. Select <b>Azure Resource Manager</b> to invoke an Azure management API or <b>Generic</b> for all other APIs.
Generic service connection	Required. Generic service connection that provides the baseUrl for the call and the authorization to use.
Azure subscription	Required. Azure Resource Manager subscription to configure and use for invoking Azure management APIs.

PARAMETER	COMMENTS
<b>Method</b>	Required. The HTTP method with which the API will be invoked; for example, <b>GET</b> , <b>PUT</b> , or <b>UPDATE</b> .
<b>Headers</b>	Optional. The header in JSON format to be attached to the request sent to the API.
<b>Body</b>	Optional. The request body for the function call in JSON format.
<b>URL suffix and parameters</b>	The string to append to the baseUrl from the Generic service connection while making the HTTP call
<b>Wait for completion</b>	Required. How the task reports completion. Can be <b>API response</b> (the default) - completion is when the function returns success within 20 seconds and the success criteria evaluates to true, or <b>Callback</b> - the external service makes a callback to update the timeline record.
<b>Success criteria</b>	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.
<b>Control options</b>	See <a href="#">Control options</a>

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 editor with a pipeline named 'Dev'. The 'Invoke REST API: POST' task is selected. The task configuration includes:

- Version:** 1.1\*
- Display name:** Invoke REST API: POST
- Connection type:** Generic
- Generic service connection:** MyGenericConnection
- Method:** POST
- Headers:** { "Content-Type": "application/json" }
- Body:** { "count": 19, "value": [ ] }
- URL suffix and parameters:** (empty)
- Advanced:**
  - Completion event:** ApiResponse
  - Success criteria:** (empty)
- Control Options:** (dropdown menu)

For more information about using this task, see [Approvals and gates overview](#).

## Open source

Also see [this task on GitHub](#).

## FAQ

### What base URLs are used when invoking Azure Management APIs?

Azure management APIs are invoked using *ResourceManagerEndpoint* of the selected environment. For example `https://management.Azure.com` is used when the subscription is in *AzureCloud* environment.

### Where should a task signal completion when Callback is chosen as the completion event?

To signal completion, the external service should POST completion data to the following pipelines REST endpoint.

```
{planUri}/{projectId}/_apis/distributedtask/hubs/{hubName}/plans/{planId}/events?api-version=2.0-preview.1
```

**\*\*Request Body\*\***

```
{ "name": "TaskCompleted", "taskId": "taskInstanceId", "jobId": "jobId", "result": "succeeded" }
```

See [this simple cmdline application](#) for specifics.

In addition, a C# helper library is available to enable live logging and managing task status for agentless tasks.

[Learn more](#)

### Can I use the response body as the input for another task?

No, as this task is an agentless task and uses TFS's internal HttpRequest, which doesn't return the content of the HTTP request.

# Jenkins Download Artifacts task

11/2/2020 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

Use this task to download artifacts produced by a Jenkins job.

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## 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. <a href="#">More Information</a>  The default pattern <code>()</code> will download all files across all artifacts produced by the Jenkins job. To download all files within artifact drop use <code>drop/</code> .
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 <code>commits_{suffix}.json</code> and <code>workitem_{suffix}.json</code> . If this input is not provided attachments will be create with the name <code>commits.json</code> and <code>workitems.json</code>
Artifacts are propagated to Azure	(Optional) Check this if Jenkins artifacts were propagated to Azure. To upload Jenkins artifacts to azure, refer to this <a href="#">Jenkins plugin</a>
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.
<b>CONTROL OPTIONS</b>	

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Manual Intervention task

3/18/2021 • 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. This article refers only to classic pipelines. For YAML usage, see [Manual Validation task](#).

The screenshot shows the Azure Pipelines interface for a 'Fabrikam' release pipeline. The 'Tasks' tab is selected. A 'Manual Intervention' task is highlighted with a red box. The task details show a display name 'Manual Intervention' and instructions: 'Ready to deploy to \$(Release.EnvironmentName) for customer \${customerName}. Please contact customer to confirm deployment requirements have been met.'

## 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 <a href="#">Control options</a>

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.

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](#).

# Manual Validation task

3/17/2021 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task in a YAML pipeline to pause a run within a stage, typically to perform some manual actions or validations, and then resume/reject the run.

## Demands

This task is supported only in YAML pipelines. Can be used only in an [agentless job](#) of a YAML pipeline.

## Arguments

PARAMETER	COMMENTS
<b>instructions</b>	Optional. The instruction text to display to the user when the task is activated.
<b>notifyUsers</b>	Optional. The list of users that will be notified that the task has been activated.
<b>onTimeout</b>	Required. The action to take (reject or resume) if the task times out. Defaults to reject.
<b>Control options</b>	See <a href="#">Control options</a>

The **Manual Validation** task allows you to pause a pipeline run within a stage, typically to perform some manual steps or actions, and then continue with the pipeline. For example, the user may need to manually validate certain deployment configurations before the pipeline starts a long running computational intensive job.

The **Manual Validation** 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 pause. You can configure the task to send email notifications to users and user groups when it is awaiting a review, and specify the automatic response (reject or resume) after a configurable timeout occurs.

You can specify the timeout value for the task using the `timeoutInMinutes` parameter available in control options. This parameter is optional.

### NOTE

For the task to run completely, the timeout value of the job should be higher than the timeout value of the task. See [default job timeout values](#).

### TIP

You can use variables to specify email addresses in the `notifyUsers` parameter.

When the Manual validation task is activated during a pipeline, it displays a message bar containing a link that

opens the Manual validation dialog containing the instructions. After carrying out the manual steps, the administrator or user can choose to resume the run, or reject it. Users with 'Queue builds' permission on the pipeline can resume or reject the run.

## Example

- [YAML](#)
- [Classic](#)

```
jobs:
- job: waitForValidation
 displayName: Wait for external validation
 pool: server
 timeoutInMinutes: 4320 # job times out in 3 days
 steps:
 - task: ManualValidation@0
 timeoutInMinutes: 1440 # task times out in 1 day
 inputs:
 notifyUsers: |
 test@test.com
 example@example.com
 instructions: 'Please validate the build configuration and resume'
 onTimeout: 'resume'
```

# PowerShell task

5/13/2021 • 5 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

Use this task to run a PowerShell script.

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## Demands

- DotNetFramework

## YAML snippet

```
PowerShell
Run a PowerShell script on Linux, macOS, or Windows
- task: PowerShell@2
 inputs:
 targetType: 'filePath' # Optional. Options: filePath, inline
 #filePath: # Required when targetType == FilePath
 #arguments: # Optional
 #script: '# Write your PowerShell commands here.Write-Host Hello World' # 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 two shortcuts in YAML:

```
- powershell: # inline script
 workingDirectory: #
 displayName: #
 failOnStderr: #
 errorActionPreference: #
 ignoreLASTEXITCODE: #
 env: # mapping of environment variables to add
```

```
- pwsh: # inline script
 workingDirectory: #
 displayName: #
 failOnStderr: #
 errorActionPreference: #
 ignoreLASTEXITCODE: #
 env: # mapping of environment variables to add
```

Both of these resolve to the `PowerShell@2` task. `powershell` runs Windows PowerShell and will only work on a

Windows agent. `pwsh` runs PowerShell Core, which must be installed on the agent or container.

#### NOTE

Each PowerShell session lasts only for the duration of the job in which it runs. Tasks that depend on what has been bootstrapped must be in the same job as the bootstrap.

## Arguments

ARGUMENT	DESCRIPTION
<code>targetType</code> Type	(Optional) Sets whether this is an inline script or a path to a <code>.ps1</code> file. Defaults to <code>filepath</code> Default value: <code>filePath</code>
<code>filePath</code> Script Path	(Required) 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>
<code>arguments</code> Arguments	(Optional) Arguments passed to the Powershell script. For example, <code>-Name someName -Path -Value "Some long string value"</code>  Note: unused when Type is <code>inline</code> .
<code>script</code> Script	(Required) Contents of the script. Required if targetType is <code>inline</code> . The maximum supported inline script length is 32766 characters. If you need more than that, use the script from file. Default value: # Write your PowerShell commands here. <code>Write-Host "Hello World"</code>
<code>errorActionPreference</code> ErrorActionPreference	(Optional) Prepends the line <code>\$ErrorActionPreference = 'VALUE'</code> at the top of your script Default value: <code>stop</code>
<code>failOnStderr</code> 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. Otherwise the task will rely on the exit code to determine failure Default value: <code>false</code>
<code>ignoreLASTEXITCODE</code> Ignore \$LASTEXITCODE	(Optional) If this is false, the line <code>if ((Test-Path -LiteralPath variable:\\LASTEXITCODE)) { exit \$LASTEXITCODE }</code> is appended to the end of your script. This will cause the last exit code from an external command to be propagated as the exit code of powershell. Otherwise the line is not appended to the end of your script Default value: <code>false</code>

ARGUMENT	DESCRIPTION
<code>pwsh</code> Use PowerShell Core	(Optional) If this is true, then on Windows the task will use pwsh.exe from your PATH instead of powershell.exe Default value: false
<code>workingDirectory</code> Working directory	(Optional) 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>
Environment 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:

```
- powershell: echo $env:MYSECRET
 env:
 MySecret: $(Foo)
```

## 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

Add the PowerShell task, set the Type to `inline`, and paste in this script:

```
Writes a warning to build summary and to log in yellow text
Write-Host "##vso[task.LogIssue type=warning;]This is the warning"
```

### Write an error

Add the PowerShell task, set the Type to `inline`, and paste in this script:

```
Writes an error to build summary and to log in red text
Write-Host "##vso[task.LogIssue type=error;]This is the error"
```

## TIP

If you want this error to fail the build, then add this line:

```
exit 1
```

## ApplyVersionToAssemblies.ps1

[Use a script to customize your build pipeline](#)

### Call PowerShell script with multiple arguments

Create PowerShell script `test2.ps1`:

```
param ($input1, $input2)
Write-Host "$input1 $input2"
```

In your YAML pipeline, call:

```
- task: PowerShell@2
 inputs:
 targetType: 'filePath'
 filePath: $(System.DefaultWorkingDirectory)\test2.ps1
 arguments: > # Use this to avoid newline characters in multiline string
 -input1 "Hello"
 -input2 "World"
 displayName: 'Print Hello World'
```

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### 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?

To learn more about defining build variables in a script, see [Define and modify your build variables in a script](#).

To learn more about defining release variables in a script, see [Define and modify your release variables in a script](#).

### Q: I'm having issues with publishing my artifacts. How can I view the detailed logs?

To enable detailed logs for your pipeline:

1. Edit your pipeline and select **Variables**

2. Add a new variable with the name `System.Debug` and value `true`

3. Save

**Q: Which variables are available to me?**

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just few of the variables you can use in your pipeline. Variables are available as [expressions](#) or scripts.

See [Define variables](#), [predefined variables](#), and [Classic release and artifacts variables](#) to learn about the different types of variables.

**Do I need an agent?**

You need at least one [agent](#) to run your build or release.

**I'm having problems. How can I troubleshoot them?**

See [Troubleshoot Build and Release](#).

**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](#).

**My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?**

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

**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

6/23/2021 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017 | TFS 2015.3

## NOTE

This task is deprecated. If you're using Team Foundation Server 2017 or newer, we recommend that you use the [Pipeline Artifacts](#) task instead.

Use this task in a build pipeline to publish build artifacts to Azure Pipelines, TFS, or a file share.

## Demands

None

## YAML snippet

```
Publish build artifacts
Publish build artifacts to Azure Pipelines or a Windows 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
 #fileCopyOptions: #Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<code>pathToPublish</code> Path to publish <i>required</i>	The folder or file path to publish. This can be a fully-qualified path or a path relative to the root of the repository. Wildcards are not supported. See <a href="#">Artifacts in Azure Pipelines</a> . Default value: <code>\$(Build.ArtifactStagingDirectory)</code>
<code>ArtifactName</code> Artifact name <i>required</i>	Specify the name of the artifact that you want to create. It can be whatever you want. Default value: <code>drop</code>
<code>publishLocation</code> Artifact publish location <i>required</i>	Choose whether to store the artifact in Azure Pipelines ( <code>Container</code> ), or to copy it to a file share ( <code>FilePath</code> ) that must be accessible from the build agent. To learn more, see <a href="#">Artifacts in Azure Pipelines</a> . Default value: <code>Container</code>

ARGUMENT	DESCRIPTION
<code>TargetPath</code> File share path <i>required, if publishLocation=FilePath</i>	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.
<code>Parallel</code> 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.
<code>ParallelCount</code> 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. Default value: 8
<code>FileCopyOptions</code> File copy options	Pass additional options to the Robocopy command. For example, the recursive minimatch pattern <code>**/*</code> .
Control options	

#### NOTE

You cannot use **Bin**, **App\_Data** and other folder names reserved by IIS as an Artifact name because this content is not served in response to Web requests. Please see [ASP.NET Web Project Folder Structure](#) for more details.

## Example

```
steps:
- task: CopyFiles@2
 inputs:
 contents: '_buildOutput/**'
 targetFolder: $(Build.ArtifactStagingDirectory)
- task: PublishBuildArtifacts@1
 inputs:
 pathToPublish: $(Build.ArtifactStagingDirectory)
 artifactName: MyBuildOutputs
```

## FAQ

**Q: I'm having issues with publishing my artifacts. How can I view the detailed logs?**

To enable detailed logs for your pipeline:

1. Edit your pipeline and select **Variables**
2. Add a new variable with the name `System.Debug` and value `true`
3. **Save**

**Q: Which variables are available to me?**

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just few of the variables you can use in your pipeline. Variables are available as [expressions](#) or scripts.

See [Define variables](#), [predefined variables](#), and [Classic release and artifacts variables](#) to learn about the different types of variables.

# Publish Pipeline Artifacts task

5/7/2021 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task in a pipeline to publish your artifacts(note that publishing is NOT supported in release pipelines. It is supported in multi-stage pipelines, build pipelines, and yaml pipelines).

### TIP

Looking to get started with build artifacts? See [Artifacts in Azure Pipelines](#).

## YAML snippet

```
Publish pipeline artifacts
Publish (upload) a file or directory as a named artifact for the current run
- task: PublishPipelineArtifact@1
 inputs:
 #targetPath: '$(Pipeline.Workspace)' # Required
 #artifactName: 'drop' # Optional
 #artifactType: 'pipeline' # Required. Options: pipeline, filepath. Default value: pipeline
 #fileSharePath: '\server\folderName' # Required when artifactType = filepath
 #parallel: false # Optional. Default value: false
 #parallelCount: 1 # Optional. Value must be at least 1 and not greater than 128. Default value: 8
```

### NOTE

The `publish` and `download` keywords are shortcuts for the **Publish Pipeline Artifact** task. You can use them in your pipeline to publish and download artifacts. For more information, see [Publish](#) and [Download](#) in the YAML schema.

## Arguments

ARGUMENT	DESCRIPTION
targetPath	(Required) Path to the folder or file you want to publish. Wildcards are not supported. The path must be a fully-qualified path or a valid path relative to the root directory of your repository. See <a href="#">Artifacts in Azure Pipelines</a> .
artifactName	(Optional) Specify the name of the artifact that you want to create. It can be whatever you want. For example: <code>drop</code>
artifactType	(Required) Artifacts publish location. Choose whether to publish your file as a pipeline artifact, or to copy it to a file share that must be accessible from the pipeline agent. Options: pipeline, filepath. Default value: pipeline
fileSharePath	(Optional) The file share path that the artifact will be published to. This can include variables. Required when <code>artifactType</code> = <code>filepath</code> . E.g: <code>\server\folderName</code>

ARGUMENT	DESCRIPTION
parallel	(Optional) Select whether to copy files in parallel using multiple threads. If this setting is not enabled, one thread will be used. Default value: false
parallelCount	(Optional) Enter the degree of parallelism, or number of threads used to publish a package. The value must be at least 1 and not greater than 128.

## FAQ

**Q: I'm having issues with publishing my artifacts. How can I view the detailed logs?**

To enable detailed logs for your pipeline:

1. Edit your pipeline and select **Variables**
2. Add a new variable with the name `System.Debug` and value `true`
3. **Save**

**Q: Which variables are available to me?**

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just few of the variables you can use in your pipeline. Variables are available as [expressions](#) or scripts.

See [Define variables](#), [predefined variables](#), and [Classic release and artifacts variables](#) to learn about the different types of variables.

## Open-source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Publish To Azure Service Bus task

11/2/2020 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task in an [agentless job](#) of a 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 is required)
- task: PublishToAzureServiceBus@1
 inputs:
 azureSubscription:
 #messageBody: # Optional
 #sessionId: # 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 <a href="#">Control options</a>

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### Do I need an agent?

You do not need an agent to run this task. This task can be used in only an [agentless job](#) of a release pipeline.

## Where should a task signal completion?

To signal completion, the external service should POST completion data to the following pipelines REST endpoint.

```
{planUri}/{projectId}/_apis/distributedtask/hubs/{hubName}/plans/{planId}/events?api-version=2.0-preview.1
Request Body
{ "name": "TaskCompleted", "taskId": "taskInstanceId", "jobId": "jobId", "result": "succeeded" }
```

See [this simple cmdline application](#) for specifics.

In addition, a C# helper library is available to enable live logging and managing task status for agentless tasks.

[Learn more](#)

# Python Script task

3/8/2021 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task to run a Python script.

## YAML snippet

```
Python script
Run a Python file or inline 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
<code>scriptSource</code> Type	(Required) Target script type: File path or Inline
<code>scriptPath</code> Script Path	(Required when <code>scriptSource</code> == <code>filePath</code> ) Path of the script to execute. Must be a fully qualified path or relative to <code>\$(System.DefaultWorkingDirectory)</code> .
<code>script</code> Script	(Required when <code>scriptSource</code> == <code>inline</code> ) The Python script to run
<code>arguments</code> Arguments	(Optional) A string containing arguments passed to the script. They'll be available through <code>sys.argv</code> as if you passed them on the command line.
<code>pythonInterpreter</code> Python interpreter	(Optional) Absolute path to the Python interpreter to use. If not specified, the task assumes a Python interpreter is available on the PATH and simply attempts to run the <code>python</code> command.
<code>workingDirectory</code> Working directory	(Optional)
<code>failOnStderr</code> Fail on standard error	(Optional) If true, this task will fail if any text is written to <code>stderr</code> .

## Control options

## Remarks

By default, this task will invoke `python` from the system path. Run [Use Python Version](#) to put the version you want in the system path.

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Query Azure Monitor Alerts task

4/10/2020 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

Use this task in an [agentless job](#) of 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

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## Demands

None

## YAML snippet

```
Query Azure Monitor alerts
Observe the configured Azure Monitor rules for active alerts
- task: AzureMonitor@1
 inputs:
 connectedServiceNameARM:
 resourceGroupName:
 #filterType: 'none' # Options: resource, alertrule, none
 #resource: # Required when filterType == Resource
 #alertRule: # Required when filterType == Alertrule
 #severity: 'Sev0,Sev1,Sev2,Sev3,Sev4' # Optional. Options: sev0, sev1, sev2, sev3, sev4
 #timeRange: '1h' # Optional. Options: 1h, 1d, 7d, 30d
 #alertState: 'Acknowledged,New' # Optional. Options: new, acknowledged, closed
 #monitorCondition: 'Fired' # Optional. Options: fired , resolved
```

## Arguments

PARAMETER	COMMENTS
<b>Azure subscription</b>	Required. Select an Azure Resource Manager service connection.
<b>Resource group</b>	Required. The resource group being monitored in the subscription.
<b>Resource type</b>	Required. Select the resource type in the selected group.
<b>Resource name</b>	Required. Select the resources of the chosen types in the selected group.

PARAMETER	COMMENTS
<b>Alert rules</b>	Required. Select from the currently configured alert rules to query for status.
<b>Control options</b>	See <a href="#">Control options</a>

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

4/10/2020 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

Use this task in an [agentless job](#) of a 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

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## Demands

None

## YAML snippet

```
Query work items
Execute a work item query and check the number of items returned
- task: queryWorkItems@0
 inputs:
 queryId:
 #maxThreshold: '0'
 #minThreshold: '0'
```

## Arguments

PARAMETER	COMMENTS
<b>Query</b>	Required. Select a work item query within the current project. Can be a built-in or custom query.
<b>Upper threshold</b>	Required. Maximum number of matching workitems for the query. Default value = 0
<b>Lower threshold</b>	Required. Minimum number of matching workitems for the query. Default value = 0
<b>Control options</b>	See <a href="#">Control options</a>

Succeeds if *minimum-threshold* <= #-matching-workitems <= *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

6/2/2020 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task 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 in 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
<b>Cluster Connection</b>	The Azure Service Fabric service connection to use to connect and authenticate to the cluster.
<b>Script Type</b>	Specify whether the script is provided as a file or inline in the task.
<b>Script Path</b>	Path to the PowerShell script to run. Can include wildcards and variables. Example: <code>\$(system.defaultworkingdirectory)/**/drop/projectartifacts/**/dockcompose.yml</code> . Note: combining compose files is not supported as part of this task.
<b>Script Arguments</b>	Additional parameters to pass to the PowerShell script. Can be either ordinal or named parameters.
<b>Inline Script</b>	The PowerShell commands to run on the build agent. <a href="#">More information</a>
<b>Control options</b>	See <a href="#">Control options</a>

Also see: [Service Fabric Compose Deploy task](#)

## Open source

This task is open source on [GitHub](#). Feedback and contributions are welcome.

## FAQ

### Do I need an agent?

You need at least one [agent](#) to run your build or release.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE.EXTRA.CA.CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

# Shell Script task

3/8/2021 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

Use this task to run a shell script using bash.

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## 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: <input type="text" value="test.sh"/>
---------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------

This example also works with release pipelines.

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### 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?

To learn more about defining build variables in a script, see [Define and modify your build variables in a script](#).

To learn more about defining release variables in a script, see [Define and modify your release variables in a script](#).

### Q: I'm having issues with publishing my artifacts. How can I view the detailed logs?

To enable detailed logs for your pipeline:

1. Edit your pipeline and select **Variables**
2. Add a new variable with the name  and value
3. **Save**

### Q: Which variables are available to me?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just few of the variables you can use in your pipeline. Variables are available as [expressions](#) or scripts.

See [Define variables](#), [predefined variables](#), and [Classic release and artifacts variables](#) to learn about the different types of variables.

### Do I need an agent?

You need at least one [agent](#) to run your build or release.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

**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](#).

**My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?**

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

**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 Manifests task

5/26/2021 • 5 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

## NOTE

In TFS 2017 this task is named **Update Service Fabric App Versions** task.

Use this task in a build 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.

## NOTE

This task requires Windows PowerShell.

This task is not yet available in **release** pipelines.

## Demands

None

## YAML snippet

```
Update Service Fabric manifests
Automatically update portions of application and service manifests in a packaged Azure 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: # Required when updateType == Manifest versions && updateOnlyChanged == true
 #logAllChanges: true # Optional
 #compareType: 'LastSuccessful' # Optional. Options: lastSuccessful, specific
 #buildNumber: # Optional
 #overwriteExistingPkgArtifact: true # Optional
 #imageNamePath: # Optional
 #imageDigestsPath: # Required when updateType == Docker Image Settings
```

## Arguments

ARGUMENT	DESCRIPTION
Application Package	The location of the Service Fabric application package to be deployed to the cluster.

ARGUMENT	DESCRIPTION
	<p>Example:  <code>\$(system.defaultworkingdirectory)/**/drop/applicationpackage</code></p>
	Can include wildcards and variables.
Version Value	<p>The value appended to the versions in the manifest files.  Default is <code>.(Build.BuildNumber)</code>.</p>
	<p><b>Tip:</b> You can modify the <a href="#">build number format</a> directly or use a <a href="#">logging command</a> 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	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><b>Note:</b> By default, the compiler will create different outputs even if you made no changes. Use the <a href="#">deterministic compiler flag</a> 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](#)

This task can only be used in a build pipeline to automatically update the versions of a packaged Service Fabric app.

This task support two types of update:

- Manifest version:** It will update Service and Application versions specified in manifest files in Service

fabric application package. If specified, it compares current files against a previous build and updates version only for those services which have been changed.

2. **Docker image settings:** It will update docker container image settings specified in manifest files in Service fabric application package. The image settings to be placed are picked from two files:

a. **Image names file:** This file is generated by build task

b. **Image digests file:** This file is generated by docker task when it pushes images to registry

## Task Inputs

PARAMETER	NAME	DESCRIPTION
<code>updateType</code>	Update Type	(Required) Specify the type of update that should be made to the manifest files. In order to use both update types, add an instance of this task to the build pipeline for each type of update to be executed. Default value: Manifest versions
<code>applicationPackagePath</code>	Application Package	(Required) Path to the application package. <a href="#">Variables</a> and wildcards can be used in the path
<code>versionSuffix</code>	Version Value	(Required) The value used to specify the version in the manifest files. Default is <code>\$(Build.BuildNumber)</code> . Default value: <code>\$(Build.BuildNumber)</code>
<code>versionBehavior</code>	Version Behavior	Specify whether to append the version value to existing values in the manifest files or replace them. Default value: Append
<code>updateOnlyChanged</code>	Update only if changed	(Required) Incrementally update only the packages that have changed. Use the <a href="#">deterministic compiler flag</a> to ensure builds with the same inputs produce the same outputs. Default value: false
<code>pkgArtifactName</code>	Package Artifact Name	The name of the artifact containing the application package for comparison
<code>logAllChanges</code>	Log all changes	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 faster performance. Default value: true
<code>compareType</code>	Compare against	The build for comparison. Default value: LastSuccessful
<code>buildNumber</code>	Build Number	The build number for comparison

PARAMETER	NAME	DESCRIPTION
<code>overwriteExistingPkgArtifact</code>	Overwrite Existing Package Artifact	Always download a new copy of the artifact. Otherwise use an existing copy, if present. Default value: true
<code>imageNamesPath</code>	Image Names Path	Path to a text file that contains the names of the Docker images associated with the Service Fabric application that should be updated with digests. Each image name must be on its own line and must be in the same order as the digests in Image Digests file. If the images are created by the Service Fabric project, this file is generated as part of the Package target and its output location is controlled by the property <code>BuiltDockerImagesFilePath</code>
<code>imageDigestsPath</code>	Image Digests Path	(Required) Path to a text file that contains the digest values of the Docker images associated with the Service Fabric application. This file can be output by the <a href="#">Docker task</a> when using the push action. The file should contain lines of text in the format of ' <code>registry/image_name@digest_value</code> '

## Example

Also see: [Service Fabric Application Deployment task](#)

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### Do I need an agent?

You need at least one [agent](#) to run your build or release.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

### 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 License task

11/2/2020 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

This task was used in a build or release pipeline to activate or deactivate Xamarin licenses.

## Deprecated

### NOTE

This task is deprecated because you no longer need a Xamarin license to [build your Xamarin app](#). Use the free version of Xamarin from <https://store.xamarin.com>.

## Demands

None

## Arguments

ARGUMENT	DESCRIPTION
Action	Select:  <b>Activate</b> for the first instance of this build task, which should come before any instances of the Xamarin.Android or Xamarin.iOS tasks.  <b>Deactivate</b> for the second instance of this build task, which should come after all instances of the Xamarin.Android and Xamarin.iOS tasks. You should also select <b>Always run</b> under <b>Control options</b> for the last instance of the Xamarin license task.
Email	Xamarin account email address.
Password	Xamarin account password.  Use a <a href="#">secret variable</a> to avoid exposing this value.
Xamarin Product	Select the build task that you're running in this build pipeline, such as <b>Xamarin.Android</b> or <b>Xamarin.iOS</b> .
Advanced - Timeout in Seconds	Specify how long you want to allow the build task to wait for the activation or deactivation.
Control options	

## Example

[Build your Xamarin app](#)

# Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### Do I need an agent?

You need at least one [agent](#) to run your build or release.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

### 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

11/2/2020 • 5 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

This task lets you run test suites against an application binary ( `.apk` or `.ipa` file) using App Center Test.

- [Sign up with App Center](#) first.
- For details about using this task, see the App Center documentation article [Using Azure DevOps for UI Testing](#).

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## YAML snippet

```

App Center test
Test app packages with Visual Studio App Center
- task: AppCenterTest@1
 inputs:
 appFile:
 #artifactsDirectory: '$(Build.ArtifactStagingDirectory)/AppCenterTest'
 #prepareTests: true # 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
 #uitestStorePath: # Optional
 #uitestStorePassword: # Optional
 #uitestKeyAlias: # Optional
 #uitestKeyPassword: # Optional
 #uitestToolsDirectory: # Optional
 #signInfo: # Optional
 #xcUITestBuildDirectory: # Optional
 #xcUITestIpaFile: # Optional
 #prepareOptions: # Optional
 #runTests: true # 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
<b>app</b> Binary application file path	(Required) Relative path from the repo root to the APK or IPA file that you want to test. Argument alias: <code>appFile</code>
<b>artifactsDir</b> 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. Default value: <code>\$(Build.ArtifactStagingDirectory)/AppCenterTest</code> Argument alias: <code>artifactsDirectory</code>
<b>enablePrepare</b> Prepare tests	(Optional) Specify whether to prepare tests. Default value: <code>true</code> Argument alias: <code>prepareTests</code>

ARGUMENT	DESCRIPTION
<code>framework</code> Test framework	(Required) Options: appium, calabash, espresso, uitest (Xamarin UI Test), xcuitest. Default value: <code>appium</code> Argument alias: <code>frameworkOption</code>
<code>appiumBuildDir</code> Build directory (Appium)	(Required) Path to directory with Appium tests. Argument alias: <code>appiumBuildDirectory</code>
<code>espressoBuildDir</code> Build directory (Espresso)	(Optional) Path to Espresso output directory Argument alias: <code>espressoBuildDirectory</code>
<code>espressoTestApkPath</code> Test APK path (Espresso)	(Optional) Path to APK file with Espresso tests. If not set, build-dir is used to discover it. Wildcard is allowed. Argument alias: <code>espressoTestApkFile</code>
<code>calabashProjectDir</code> Project directory (Calabash)	(Required) Path to Calabash workspace directory. Argument alias: <code>calabashProjectDirectory</code>
<code>calabashConfigFile</code> Cucumber config file (Calabash)	(Optional) Path to Cucumber configuration file, usually cucumber.yml.
<code>calabashProfile</code> Profile to run (Calabash)	(Optional) Profile to run. This value must exists in the Cucumber configuration file.
<code>calabashSkipConfigCheck</code> Skip Configuration Check (Calabash)	(Optional) Force running without Cucumber profile. Default value: <code>false</code>
<code>uitestBuildDir</code> Build directory (Xamarin UI Test)	(Required) Path to directory with built test assemblies Argument alias: <code>uiTestBuildDirectory</code>
<code>uitestStorePath</code> Store file (Xamarin UI Test)	(Optional) Path to the store file used to sign the app.
<code>uitestStorePass</code> Store password (Xamarin UI Test)	(Optional) Password of the store file used to sign the app. Use a new variable with its lock enabled on the Variables tab to encrypt this value. Argument alias: <code>uiTestStorePassword</code>
<code>uitestKeyAlias</code> Key alias (Xamarin UI Test)	(Optional) Enter the alias that identifies the public/private key pair used in the store file
<code>uitestKeyPass</code> Key password (Xamarin UI Test)	(Optional) Enter the key password for the alias and store file. Use a new variable with its lock enabled on the Variables tab to encrypt this value. Argument alias: <code>uiTestKeyPassword</code>
<code>uitestToolsDir</code> Test tools directory (Xamarin UI Test)	(Optional) Path to directory with Xamarin UI test tools that contains <code>test-cloud.exe</code> Argument alias: <code>uiTestToolsDirectory</code>

ARGUMENT	DESCRIPTION
<code>signInfo</code> Signing information (Calabash/Xamarin UI Test)	(Optional) Use Signing Information for signing the test server.
<code>xcuitestBuildDir</code> Build directory (XCUITest)	(Optional) Path to the build output directory, usually <code>\$(ProjectDir)/Build/Products/Debug-iphoneos</code> . Argument alias: <code>xcUITestBuildDirectory</code>
<code>xcuitestTestIPAPath</code> Test IPA path (XCUITest)	(Optional) Path to the *.ipa file with the XCUITest tests. Argument alias: <code>xcUITestIPAFile</code>
<code>prepareOpts</code> Additional options (for preparing tests)	(Optional) Additional arguments passed to the App Center test prepare step. Argument alias: <code>prepareOptions</code>
<code>enableRun</code> Run tests	(Optional) Specify whether to run the tests. Default value: <code>true</code> Argument alias: <code>runTests</code>
<code>credsType</code> Authentication method	(Required) Use App Center service connection or enter credentials to connect to App Center. Default value: <code>serviceEndpoint</code> Argument alias: <code>credentialsOption</code>
<code>serverEndpoint</code> App Center service connection	(Required) Select the service connection for App Center. Create a new App Center service connection in Azure DevOps project settings.
<code>username</code> App Center username (when not using a service connection)	(Required) Visit <a href="https://appcenter.ms/settings/profile">https://appcenter.ms/settings/profile</a> to get your username.
<code>password</code> App Center password (when not using a service connection)	(Required) Visit <a href="https://appcenter.ms/settings/profile">https://appcenter.ms/settings/profile</a> to set your password. It can accept a variable defined in build or release pipelines as <code>\$(passwordVariable)</code> . You may mark variable type as <code>secret</code> to secure it.
<code>appSlug</code> App slug	(Required) The app slug is in the format of <code>{username}/{app_identifier}</code> . To locate <code>{username}</code> and <code>{app_identifier}</code> for an app, click on its name from <a href="https://appcenter.ms/apps">https://appcenter.ms/apps</a> , and the resulting URL is in the format of <a href="https://appcenter.ms/users/{username}/apps/{app_identifier}">https://appcenter.ms/users/{username}/apps/{app_identifier}</a> .
<code>devices</code> 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.
<code>series</code> Test series	(Optional) The series name for organizing test runs (e.g. master, production, beta). Default value: <code>master</code>
<code>dsymDir</code> dSYM directory	(Optional) Path to iOS symbol files. Argument alias: <code>dsymDirectory</code>

ARGUMENT	DESCRIPTION
<code>locale</code> System language	(Required) Options: da_DK, de_DE, en_GB, en_US, es_ES, es_MX, fr_FR, ja_JP, nl_NL, ru_RU, user. If your language isn't an option, use <code>user/Other</code> and enter its locale below, such as <code>en_US</code> . Default value: <code>en_US</code> Argument alias: <code>localeOption</code>
<code>userDefinedLocale</code> Other locale	(Optional) Enter any two-letter ISO-639 language code along with any two-letter ISO 3166 country code in the format <code>[language]_[country]</code> , such as <code>en_US</code> .
<code>loginOpts</code> Additional options for login	(Optional) Additional arguments passed to the App Center login step. Argument alias: <code>loginOptions</code>
<code>runOpts</code> Additional options for run	(Optional) Additional arguments passed to the App Center test run. Argument alias: <code>runOptions</code>
<code>async</code> Do not wait for test result	(Optional) Specify whether to execute tests asynchronously, exiting just after tests are uploaded, without waiting for test results. Default value: <code>false</code> Argument alias: <code>skipWaitingForResults</code>
<code>cliLocationOverride</code> App Center CLI location	(Optional) Path to the App Center CLI on the build or release agent. Argument alias: <code>cliFile</code>
<code>debug</code> Enable debug output	(Optional) Add --debug to the App Center CLI for verbose output. Argument alias: <code>showDebugOutput</code>

## Example

This example runs Espresso tests on an Android app using the App Center Test task.

```
steps:
- task: AppCenterTest@1
 displayName: 'Espresso Test - Synchronous'
 inputs:
 appFile: 'Espresso/espresso-app.apk'
 artifactsDirectory: '$(Build.ArtifactStagingDirectory)/AppCenterTest'
 frameworkOption: espresso
 espressoBuildDirectory: Espresso
 serverEndpoint: 'myAppCenterServiceConnection'
 appSlug: 'xplatbg1/EspressoTests'
 devices: a84c93af
```

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

# Cloud-based Apache JMeter Load Test task

11/2/2020 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

### Caution

The cloud-based load testing service is deprecated. More information about the deprecation, the service availability, and alternative services can be found [here](#).

Use this task 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
Run an Apache JMeter load test in the 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
Azure Pipelines 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

ARGUMENT	DESCRIPTION
<a href="#">CONTROL OPTIONS</a>	

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

# Cloud-based Load Test task

3/6/2021 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

## Caution

The cloud-based load testing service is deprecated. More information about the deprecation, the service availability, and alternative services can be found [here](#).

Use this task 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

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## Demands

The agent must have the following capability:

- Azure PowerShell

## YAML snippet

```
Cloud-based load test
Run a 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
----------	-------------

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"> <li>- Required for builds and releases on TFS and must specify a connection to the Azure DevOps organization where the load test will run.</li> <li>- Optional for builds and releases on Azure Pipelines. In this case, if not provided, the current Azure Pipelines connection is used.</li> <li>- See <a href="#">Generic service connection</a>.</li> </ul>
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 <code>myfile.loadtest</code> ) 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 <a href="#">Control options</a>

## Examples

- [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](#)

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### 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.

#### **I'm having problems. How can I troubleshoot them?**

See [Troubleshoot Build and Release](#).

#### **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](#).

#### **My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?**

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

#### **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
- Get advice on [Stack Overflow](#), and get support via the [Developer Community](#)

# Cloud-based Web Performance Test task

11/2/2020 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

## Caution

The cloud-based load testing service is deprecated. More information about the deprecation, the service availability, and alternative services can be found [here](#).

Use this task 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

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## Demands

The agent must have the following capability:

- Azure PowerShell

## YAML snippet

```
Cloud-based web performance test
Run 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
----------	-------------

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"> <li>- Required for builds and releases on TFS and must specify a connection to the Azure DevOps organization where the load test will run.</li> <li>- Optional for builds and releases on Azure Pipelines. In this case, if not provided, the current Azure Pipelines connection is used.</li> <li>- See <a href="#">Generic service connection</a>.</li> </ul>
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 <b>Default</b> to generate the load from the location associated with your Azure DevOps organization.
Run load test using	<p>Select <b>Automatically provisioned agents</b> 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.</p> <p>Select <b>Self-provisioned agents</b> 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 <a href="#">Testing private/intranet applications using Cloud-based load testing</a>.</p>
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 <a href="#">Control options</a>

## More Information

- [Cloud-based Load Testing](#)

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### 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
- Get advice on [Stack Overflow](#), and get support via the [Developer Community](#)

# Container Structure Tests

4/20/2020 • 2 minutes to read • [Edit Online](#)

The Container Structure Tests provide a powerful framework to validate the structure of a container image. These tests can be used to check the output of commands in an image, as well as verify metadata and contents of the filesystem. Tests can be run either through a standalone binary, or through a Docker image.

Tests within this framework are specified through a YAML or JSON config file. Multiple config files may be specified in a single test run. The config file will be loaded in by the test runner, which will execute the tests in order. Within this config file, four types of tests can be written:

- Command Tests (testing output/error of a specific command issued)
- File Existence Tests (making sure a file is, or isn't, present in the file system of the image)
- File Content Tests (making sure files in the file system of the image contain, or do not contain, specific contents)
- Metadata Test, singular (making sure certain container metadata is correct)

## Container Structure Test Task

This task helps you run container structure tests and publish test results to Azure Pipelines and provides a comprehensive test reporting and analytics experience.

### NOTE

This is an early preview feature. More upcoming features will be rolled out in upcoming sprints.

## Arguments

ARGUMENT	DESCRIPTION
<code>dockerRegistryServiceConnection</code> Docker registry service connection	(Required) Select a Docker registry service connection. Required for commands that need to authenticate with a registry.
<code>repository</code> Container repository	(Required) Name of the repository
<code>tag</code> Tag	The tag is used in pulling the image from docker registry service connection Default value: <code>\$(Build.BuildId)</code>
<code>configFile</code> Config file path	(Required) Config files path, that contains container structure tests. Either .yaml or .json files
<code>testRunTitle</code> Test run title	(Optional) Provide a name for the Test Run
<code>failTaskOnFailedTests</code> Fail task if there are test failures	(Optional) Fail the task if there are any test failures. Check this option to fail the task if test failures are detected.

# Build, Test and Publish Test

The container structure test task can be added in the classic pipeline as well as in unified pipeline (multi-stage) & YAML based pipelines.

- [YAML](#)
- [Classic](#)

In the new YAML based unified pipeline, you can search for task in the window.

New pipeline

## Review your pipeline YAML

Variables Save and run

Tasks

Container Structure

Container Structure Test

Uses container-structure-test (<https://github.com...>)

```
◆ HalaRM / azure-pipelines.yml *
```

```
38 ...- testAssemblyVer2: |
39**\TestProject*.dll
40!**\obj**
41platform: '$(BuildPlatform)'
42configuration: '$(BuildConfiguration)'
43continueOnError: true
44enabled: false
45 ...
46 Settings
47 - task: PublishTestResults@2
48 displayName: 'Publish Test Results **/ResultsFileSamples/VsTsResults/*.trx'
49 inputs:
50 - testResultsFormat: VSTest
51 - testResultsFiles: '**/ResultsFileSamples/VsTsResults/*.trx'
52 - continueOnError: true
53 - enabled: true
54 Settings
55 - task: PublishBuildArtifacts@1
56 displayName: 'Publish Artifact: drop'
57 inputs:
58 - PathToPublish: '$(build.artifactstagingdirectory)'
59 - condition: succeededOrFailed()
60
61
```

Once the task is added, you need to set the config file path, docker registry service connection, container repository and tag, if required. Task input in the yaml based pipeline is created.

✓ Connect ✓ Select ✓ Configure Review

New pipeline

## Review your pipeline YAML

Variables Save and run

← Container Structure Test

Config file path \*

/home/navin/cstfiles/fileexisttest.yaml

Fail task if there are test failures

Container Repository

Docker registry service connection \*

navin\_dockerHub

Container repository \*

navb/helldocker

Tag

v1|

About this task Add

```
◆ HalaRM / azure-pipelines.yml *
```

```
38 ...- testAssemblyVer2: |
39**\TestProject*.dll
40!**\obj**
41platform: '$(BuildPlatform)'
42configuration: '$(BuildConfiguration)'
43continueOnError: true
44enabled: false
45 ...
46 Settings
47 - task: PublishTestResults@2
48 displayName: 'Publish Test Results **/ResultsFileSamples/VsTsResults/*.trx'
49 inputs:
50 - testResultsFormat: VSTest
51 - testResultsFiles: '**/ResultsFileSamples/VsTsResults/*.trx'
52 - continueOnError: true
53 - enabled: true
54 Settings
55 - task: PublishBuildArtifacts@1
56 displayName: 'Publish Artifact: drop'
57 inputs:
58 - PathToPublish: '$(build.artifactstagingdirectory)'
59 - condition: succeededOrFailed()
60
```

## YAML file

#### ❖ HalaRM / azure-pipelines.yml \*

```
41 ... platform: '$(BuildPlatform)'
42 ... configuration: '$(BuildConfiguration)'
43 ... continueOnError: true
44 ... enabled: false
45
46 Settings
47 - task: PublishTestResults@2
48 ... displayName: 'Publish Test Results **/ResultsFileSamples/VsTsResults/*.trx'
49 ... inputs:
50 ... testResultsFormat: VSTest
51 ... testResultsFiles: '**/ResultsFileSamples/VsTsResults/*.trx'
52 ... continueOnError: true
53 ... enabled: true
54
55 Settings
56 - task: PublishBuildArtifacts@1
57 ... displayName: 'Publish Artifact: drop'
58 ... inputs:
59 ... PathToPublish: '$(build.artifactstagingdirectory)'
59 ... condition: succeededOrFailed()
60
61 Settings
62 - task: ContainerStructureTest@0
63 ... inputs:
64 ... dockerRegistryServiceConnection: 'navin_dockerHub'
65 ... repository: 'navb/helldocker'
66 ... tag: 'v1'
67 ... configFile: '/home/navin/cstfiles/fileexisttest.yaml'
67 ... failTaskOnFailedTests: true
```

#### Sample YAML

```
steps:
- task: ContainerStructureTest@0
 displayName: 'Container Structure Test '
 inputs:
 dockerRegistryServiceConnection: 'Container_dockerHub'
 repository: adma/helldocker
 tag: v1
 configFile: /home/user/cstfiles/fileexisttest.yaml
```

## View test report

Once the task is executed, you can directly go to test tab to view the full report. The published test results are displayed in the [Tests tab](#) in the pipeline summary and help you to measure pipeline quality, review traceability, troubleshoot failures, and drive failure ownership.

#24050 Update azure-pipelines.yml for Azure Pipelines

on HalaRM-CST-CL

Run new

Summary Tests

Summary

1 Run(s) Completed ( 0 Passed, 1 Failed ) 1 unique failing test in the last 14 days

2 Total tests +2

1 Passed  
1 Failed  
0 Others

1 Failed tests (+1)  
1 New  
0 Existing

50% Pass percentage  
↑ 50%

593ms Run duration  
↑ +593ms

0 Tests not reported

Bug Link Flaky

Filter by test or run name

Test	Duration	Failing since	Owner	Outcome	Tags
Container Structure Test (1/2)	0:00:00.594			Aborted (+1)	
File Existence Test: Navin File	0:00:00.000	1h ago			Current build

# Publish Code Coverage Results task

5/25/2021 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

Use this task in a build pipeline to publish code coverage results produced when running tests to Azure Pipelines or TFS in order to obtain coverage reporting. The task supports popular coverage result formats such as [Cobertura](#) and [JaCoCo](#).

This task can only be used in Build pipelines and is not supported in Release pipelines.

Tasks such as [Visual Studio Test](#), [.NET Core](#), [Ant](#), [Maven](#), [Gulp](#), [Grunt](#) also provide the option to publish code coverage data to the pipeline. If you are using these tasks, you do not need a separate [Publish Code Coverage Results task](#) in the pipeline.

## Demands

To generate the HTML code coverage report you need dotnet 2.0.0 or later on the agent. The dotnet folder needs to be in the environment path. If there are multiple folders containing dotnet, the one with version 2.0.0 must be before any others in the path list.

## YAML snippet

```
Publish code coverage results
Publish Cobertura or JaCoCo code coverage results from a build
- task: PublishCodeCoverageResults@1
 inputs:
 summaryFileLocation:
 #pathToSources: # 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 Ecosystems section of these topics, which also includes examples for other languages.

## Arguments

ARGUMENT	DESCRIPTION
<code>summaryFileLocation</code> Path to summary files	(Required) Path of the summary file containing code coverage statistics, such as line, method, and class coverage. Multiple summary files will be merged into a single report. The value may contain minimatch patterns. For example: <code>\$(System.DefaultWorkingDirectory)/MyApp/**/site/cobertura/coverage</code>

ARGUMENT	DESCRIPTION
<code>pathToSources</code> Path to Source files	(Optional) Path to source files is required when coverage XML reports do not contain absolute path to source files. For example, JaCoCo reports do not use absolute paths and when publishing JaCoCo coverage for Java apps, the pattern would be similar to <code>\$(System.DefaultWorkingDirectory)/MyApp/src/main/java/</code>  This input is also needed if tests are run in a docker container. This input should point to absolute path to source files on the host. For example, <code>\$(System.DefaultWorkingDirectory)/MyApp/</code>
<code>failIfCoverageEmpty</code> Fail if code coverage results are missing	(Optional) Fail the task if code coverage did not produce any results to publish.

## 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.

## FAQ

### Is code coverage data merged when multiple files are provided as input to the task or multiple tasks are used in the pipeline?

At present, the code coverage reporting functionality provided by this task is limited and it does not merge coverage data. If you provide multiple files as input to the task, only the first match is considered. If you use multiple publish code coverage tasks in the pipeline, the summary and report is shown for the last task. Any previously uploaded data is ignored.

## Help and support

- See our [troubleshooting](#) page
- Get advice on [Stack Overflow](#), and get support via the [Developer Community](#)

## Known issues

The publish code coverage results task generates and publishes the HTML report, a set of HTML files that are linked from the main `index.htm` file. If the code coverage tab fails to show the code coverage report, check whether the size of the `index.htm` file is close to or larger than 7 MB. Complete the following steps to check the size of the file. Then, if the file size is close to or larger than 7 MB, you can use the following workaround to view the coverage report:

1. Select the build **Summary** tab, and then select the **published** link:

The screenshot shows the Azure DevOps build summary for pull request #438. The 'Summary' tab is active. At the top right, there is a red box around the text '1 published; 1 consumed'. Other visible details include the repository 'CC1' and branch 'master', the start time 'May 6 at 4:53 PM', and a duration of '3m 59s'.

2. Next to the *Code Coverage Report\_\** artifact, select **Download artifacts**:

The screenshot shows the 'Artifacts' page with the 'Published' tab selected. It lists one artifact named 'Code Coverage Report\_438'. A red box highlights the 'Download artifacts' button next to the artifact name.

3. When the code coverage report is downloaded, extract the .zip file.
4. In the code coverage report, check the size of *index.html* to help determine whether the file size is causing the issue described here.
5. Open *index.html* in a browser to view the code coverage report.

# Publish Test Results task

6/30/2021 • 14 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

This task publishes test results to Azure Pipelines or TFS when tests are executed to provide a comprehensive test reporting and analytics experience. You can use the test runner of your choice that supports the results format you require. Supported results formats include [CTest](#), [JUnit](#) (including [PHPUnit](#)), [NUnit 2](#), [NUnit 3](#), Visual Studio Test (TRX), and [xUnit 2](#).

Other built-in tasks such as [Visual Studio Test task](#) and [Dot NetCore CLI task](#) automatically publish test results to the pipeline, while tasks such as [Ant](#), [Maven](#), [Gulp](#), [Grunt](#), [.NET Core](#) and [Xcode](#) provide publishing results as an option within the task, or build libraries such as [Cobertura](#) and [JaCoCo](#). If you are using any of these tasks, you do not need a separate **Publish Test Results** task in the pipeline.

The published test results are displayed in the [Tests tab](#) in the pipeline summary and help you to measure pipeline quality, review traceability, troubleshoot failures, and drive failure ownership.

The following example shows the task configured to publish test results.

The screenshot shows the Azure Pipelines pipeline editor. On the left, the pipeline structure is visible with steps: Get sources, Agent job 1, and Publish Test Results. The Publish Test Results step is currently selected. On the right, the configuration details for this task are shown:

- Task version:** 2.\*
- Display name:** Publish Test Results \*\*/TEST-\*.xml
- Test result format:** JUnit
- Test results files:** \*\*/TEST-\*.xml
- Search folder:** \$(System.DefaultWorkingDirectory)
- Merge test results:**
- Fail if there are test failures:**
- Test run title:** (empty field)

You can also use this task in a build pipeline to publish code coverage results produced when running tests to Azure Pipelines or TFS in order to obtain coverage reporting.

## Check prerequisites

If you're using a Windows self-hosted agent, be sure that your machine has this prerequisite installed:

- [.NET Framework](#) 4.6.2 or a later version

## Demands

[none]

## YAML snippet

```
Publish Test Results
Publish test results to Azure Pipelines
- task: PublishTestResults@2
 inputs:
 #testResultsFormat: 'JUnit' # Options: JUnit, NUnit, VSTest, xUnit, cTest
 #testResultsFiles: '**/TEST-*.xml'
 #searchFolder: '$(System.DefaultWorkingDirectory)' # Optional
 #mergeTestResults: false # Optional
 #failTaskOnFailedTests: 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 Ecosystems](#) 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
<code>testRunner</code> Test result format	(Required) Specify the format of the results files you want to publish. The following formats are supported: - <a href="#">CTest</a> , <a href="#">JUnit</a> , <a href="#">NUnit 2</a> , <a href="#">NUnit 3</a> , Visual Studio Test (TRX) and <a href="#">xUnit 2</a> Default value: <code>JUnit</code> Argument alias: <code>testResultsFormat</code>

ARGUMENT	DESCRIPTION
<code>testResultsFiles</code> Test results files	(Required) 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 newline. - Additionally accepts <a href="#">minimatch patterns</a> . For example, <code>!TEST[1-3].xml</code> excludes files named <code>TEST1.xml</code> , <code>TEST2.xml</code> , or <code>TEST3.xml</code> . Default value: <code>**/TEST-*.xml</code>
<code>searchFolder</code> Search folder	(Optional) Folder to search for the test result files. Default value: <code>\$(System.DefaultWorkingDirectory)</code>
<code>mergeTestResults</code> Merge test results	When this option is selected, test results from all the files will be reported against a single <a href="#">test run</a> . If this option is not selected, a separate test run will be created for each test result file. Note: Use merge test results to combine files from same test framework to ensure results mapping and duration are calculated correctly. Default value: <code>false</code>
<code>failTaskOnFailedTests</code> Fail if there are test failures	(Optional) When selected, the task will fail if any of the tests in the results file is marked as failed. The default is false, which will simply publish the results from the results file. Default value: <code>false</code>
<code>testRunTitle</code> Test run title	(Optional) 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.
<code>platform</code> Build Platform	(Optional) 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. Argument alias: <code>buildPlatform</code>
<code>configuration</code> Build Configuration	Build configuration against which the Test Run should be reported. For example, Debug or Release. If you have defined a variable for <code>configuration</code> in your build task, use that here. Argument alias: <code>buildConfiguration</code>
<code>publishRunAttachments</code> Upload test results files	(Optional) When selected, the task will upload all the test result files as attachments to the test run. Default value: <code>true</code>

## 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.

- [Visual Studio Test \(TRX\)](#)
- [JUnit](#)

- [NUnit 2](#)
- [NUnit 3](#)
- [xUnit 2](#)
- [CTest](#)

SCOPE	FIELD	VISUAL STUDIO TEST (TRX)
<a href="#">Test run</a>	Title	<a href="#">Test run title</a> specified in the task
	Date started	/TestRun/Times.Attributes["start"].Value
	Date completed	/TestRun/Times.Attributes["finish"].Value
	Duration	Date completed - Date started
	Attachments	Refer to <a href="#">Attachments support</a> section below
<a href="#">Test result</a>	Title	/TestRun/Results/UnitTestResult.Attributes["testName"].Value Or /TestRun/Results/WebTestResult.Attributes["testName"].Value Or /TestRun/Results/TestResultAggregation.Attributes["testName"].Value
	Date started	/TestRun/Results/UnitTestResult.Attributes["startTime"].Value Or /TestRun/Results/WebTestResult.Attributes["startTime"].Value Or /TestRun/Results/TestResultAggregation.Attributes["startTime"].Value
	Date completed	/TestRun/Results/UnitTestResult.Attributes["startTime"].Value + /TestRun/Results/UnitTestResult.Attributes["duration"].Value Or /TestRun/Results/WebTestResult.Attributes["startTime"].Value + /TestRun/Results/WebTestResult.Attributes["duration"].Value Or /TestRun/Results/TestResultAggregation.Attributes["startTime"].Value + /TestRun/Results/TestResultAggregation.Attributes["duration"].Value
	Duration <sup>1</sup>	/TestRun/Results/UnitTestResult.Attributes["duration"].Value Or /TestRun/Results/WebTestResult.Attributes["duration"].Value Or /TestRun/Results/TestResultAggregation.Attributes["duration"].Value
	Owner	/TestRun/TestDefinitions/UnitTest/Owners/Owner.Attributes["name"].Value

SCOPE	FIELD	VISUAL STUDIO TEST (TRX)
	Outcome	/TestRun/Results/UnitTestResult.Attributes["outcome"].Value Or /TestRun/Results/WebTestResult.Attributes["outcome"].Value Or /TestRun/Results/TestResultAggregation.Attributes["outcome"].Value
	Error message	/TestRun/Results/UnitTestResult/Output/ErrorInfo/Message.InnerText Or /TestRun/Results/WebTestResultOutput/ErrorInfo/Message.InnerText Or /TestRun/Results/TestResultAggregation/Output/ErrorInfo/Message.InnerText
	Stack trace	/TestRun/Results/UnitTestResult/Output/ErrorInfo/StackTrace.InnerText Or /TestRun/Results/WebTestResultOutput/ErrorInfo/StackTrace.InnerText Or /TestRun/Results/TestResultAggregation/Output/ErrorInfo/StackTrace.InnerText
	Attachments	Refer to <b>Attachments support</b> section below
	Console log	/TestRun/Results/UnitTestResult/Output/StdOut.InnerText Or /TestRun/Results/WebTestResultOutput/Output/StdOut.InnerText Or /TestRun/Results/TestResultAggregation/Output/StdOut.InnerText
	Console error log	/TestRun/Results/UnitTestResult/Output/StdErr.InnerText Or /TestRun/Results/WebTestResultOutput/Output/StdErr.InnerText Or /TestRun/Results/TestResultAggregation/Output/StdErr.InnerText
	Agent name	/TestRun/Results/UnitTestResult.Attributes["computerName"].Value Or /TestRun/Results/WebTestResult.Attributes["computerName"].Value Or /TestRun/Results/TestResultAggregation.Attributes["computerName"].Value
	Test file	/TestRun/TestDefinitions/UnitTest.Attributes["storage"].Value
	Priority	/TestRun/TestDefinitions/UnitTest.Attributes["priority"].Value

<sup>1</sup> Duration is used only when Date started and Date completed are not available.

<sup>2</sup> The fully Qualified name format is Namespace.Testclass.Methodname with a character limit of 512. If the test is data driven and has parameters, the character limit will include the parameters.

# 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 Dockerfile: 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 Dockerfile: 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 mcr.microsoft.com/dotnet/sdk:2.1
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 dotnet/core/aspnet:2.1
Create app directory
WORKDIR /app
Copy only the deployment artifacts
COPY /out .
ENTRYPOINT ["dotnet", "dotnetcore-sample.dll"]
```

### Define the build pipeline

- [YAML](#)
- [Classic](#)

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'
 failTaskOnFailedTests: true

- 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'
 failTaskOnFailedTests: true

- 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 main 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 for 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. For public projects, we support 2GB of total attachments.

### Visual Studio Test (TRX)

SCOPE	TYPE	PATH
Test run	Data Collector	/TestRun/ResultSummary/CollectorDataEntries/Collector/UriAttachments/UriAttachment/A.Attributes[" <b>href</b> "].Value
	Test Result	/TestRun/ResultSummary/ResultFiles/ResultFile.Attributes[" <b>path</b> "].Value
	Code Coverage	/TestRun/TestSettings/Execution/AgentRule/DataCollectors/DataCollector/Configuration/CodeCoverage/Regular/CodeCoverageItem.Attributes[" <b>binaryFile</b> "].Value And /TestRun/TestSettings/Execution/AgentRule/DataCollectors/DataCollector/Configuration/CodeCoverage/Regular/CodeCoverageItem.Attributes[" <b>pdbFile</b> "].Value
Test result	Data Collectors	/TestRun/Results/UnitTestResult/CollectorDataEntries/Collector/UriAttachments/UriAttachment/A.Attributes[" <b>href</b> "].Value Or /TestRun/Results/WebTestResult/CollectorDataEntries/Collector/UriAttachments/UriAttachment/A.Attributes[" <b>href</b> "].Value Or /TestRun/Results/TestResultAggregation/CollectorDataEntries/Collector/UriAttachments/UriAttachment/A.Attributes[" <b>href</b> "].Value

SCOPE	TYPE	PATH
	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](#)

## FAQ

### What is the maximum permissible limit of FQN?

The maximum FQN limit is 512 characters.

### Does the FQN Character limit also include properties and their values in case of data driven tests?

Yes, the FQN character limit includes properties and their values.

### Will the FQN be different for sub-results?

Currently, sub-results from the data driven tests will not show up in the corresponding data.

Example: I have a Test case: Add product to cart Data 1: Product = Apparel Data 2: Product = Footwear

All test sub-result published will only have the test case name and the data of the first row.

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## Help and support

- See our [troubleshooting](#) page
- Get advice on [Stack Overflow](#), and get support via the [Developer Community](#)

# Run Functional Tests task

11/2/2020 • 9 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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 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
<b>Machines</b>	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: <ul style="list-style-type: none"> <li>- The name of an <a href="#">Azure Resource Group</a>.</li> <li>- A comma-delimited list of machine names. Example: dbserver.fabrikam.com, dbserver_int.fabrikam.com:5986, 192.168.34:5</li> <li>- An output variable from a previous task.</li> </ul>
<b>Test Drop Location</b>	Required. The location on the test machine(s) where the test binaries have been copied by a <a href="#">Windows Machine File Copy</a> or <a href="#">Azure File Copy</a> 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>
<b>Test Selection</b>	Required. Whether the tests are to be selected from test assemblies or from a test plan.
<b>Test Assembly</b>	Required when <b>Test Selection</b> is set to <b>Test Assembly</b> . The test assemblies from which the tests should be executed. Paths are relative to the sources directory. <ul style="list-style-type: none"> <li>- Separate multiple paths with a semicolon.</li> <li>- Default is <code>**\*test*.dll</code></li> <li>- For JavaScript tests, enter the path and name of the <code>.js</code> files containing the tests.</li> <li>- Wildcards can be used. Example: <code>**\commontests\*test*.dll;</code> <code>**\frontenttests\*test*.dll</code></li> </ul>
<b>Test Filter criteria</b>	Optional when <b>Test Selection</b> is set to <b>Test Assembly</b> . 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>
<b>Test Plan</b>	Required if <b>Test Suite</b> is not specified when <b>Test Selection</b> is set to <b>Test Plan</b> . Select a test plan already configured for this organization.
<b>Test Suite</b>	Required if <b>Test Plan</b> is not specified when <b>Test Selection</b> is set to <b>Test Plan</b> . Select a test suite from the selected test plan.
<b>Test Configuration</b>	Optional when <b>Test Selection</b> is set to <b>Test Plan</b> . Select a test configuration from the selected test plan.
<b>Run Settings File</b>	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.
<b>Override Test Run Parameters</b>	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>
<b>Code Coverage Enabled</b>	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.

ARGUMENT	DESCRIPTION
<b>Distribute tests by number of machines</b>	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.
<b>Test Run Title</b>	Optional. A name for this test run, used to identify it for reporting and in comparison with other test runs.
<b>Platform</b>	Optional. The build platform against which the test run should be reported. Used only for reporting. - If you are using the <b>Build - Visual Studio</b> 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.
<b>Configuration</b>	Optional. The build configuration against which the test run should be reported. Used only for reporting. - If you are using the <b>Build - Visual Studio</b> 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.
<b>Test Configurations</b>	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 <b>Chrome</b> in the <b>Fully Qualified Name</b> and map them to configuration ID 12 defined in Microsoft Test Manager. - Use <code>DefaultTestConfiguration:{Id}</code> as a catch-all.
<b>Application Under Test Machines</b>	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 <b>Code Coverage Enabled</b> setting. The list can be a comma-delimited list of machine names or an output variable from an earlier task.
<b>Control options</b>	See <a href="#">Control options</a>

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](#)
- [Run continuous tests 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.

## FAQ

### 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.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE.EXTRA.CA.CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

### 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

- See our [troubleshooting](#) page
- Get advice on [Stack Overflow](#), and get support via the [Developer Community](#)

# Visual Studio Test task

11/2/2020 • 11 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task 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 in the [.runsettings file](#).

Tests can be distributed on multiple agents using version 2 of this task. For more information, see [Run tests in parallel using the Visual Studio Test task](#).

## Check prerequisites

If you're using a Windows self-hosted agent, be sure that your machine has this prerequisite installed:

- [.NET Framework](#) 4.6.2 or a later version

## 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: | # Required when testSelector == TestAssemblies
 # ***test*.dll
 # !***TestAdapter.dll
 # !**\obj**
 #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, 16.0, 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
 #failOnMinTestsNotRun: false # Optional
 #minimumExpectedTests: '1' # Optional
 #diagnosticsEnabled: false # 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
<code>testSelector</code> Select tests using	(Required) <b>Test assembly</b> : 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. <b>Test plan</b> : 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 <a href="#">Associate automated tests with test cases</a> . <b>Test run</b> : Use this option when you are setting up an environment to <a href="#">run tests from test plans</a> . This option should not be used when running tests in a continuous integration/continuous deployment (CI/CD) pipeline. Default value: <code>testAssemblies</code>
<code>testAssemblyVer2</code> Test files	(Required) Run tests from the specified files. Ordered tests and webtests can be run by specifying the <code>.orderedtest</code> and <code>.webtest</code> files respectively. To run <code>.webtest</code> , Visual Studio 2017 Update 4 or higher is needed. The file paths are relative to the search folder. Supports multiple lines of minimatch patterns. <a href="#">More Information</a> Default value: <code>**\*\*test*.dll\n!**\*\*TestAdapter.dll\n!**\*\obj\*\*</code>
<code>testPlan</code> Test plan	(Required) Select a test plan containing test suites with automated test cases.
<code>testSuite</code> Test suite	(Required) Select one or more test suites containing automated test cases. Test case work items must be associated with an automated test method. <a href="#">Learn more</a> .
<code>testConfiguration</code> Test configuration	(Required) Select Test Configuration.
<code>tcmTestRun</code> Test Run	(Optional) Test run based selection is used when triggering <a href="#">automated test runs from test plans</a> . This option cannot be used for running tests in the CI/CD pipeline.
<code>searchFolder</code> Search folder	(Required) Folder to search for the test assemblies.
<code>testFiltercriteria</code> Test filter criteria	(Optional) Additional criteria to filter tests from Test assemblies. For example: <code>Priority=1 Name=MyTestMethod</code> . <a href="#">More information</a>
<code>runOnlyImpactedTests</code> Run only impacted tests	(Optional) Automatically select, and run only the tests needed to validate the code change. <a href="#">More information</a>
<code>runAllTestsAfterXBuilds</code> Number of builds after which all tests should be run	(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.

ARGUMENT	DESCRIPTION
<code>uiTests</code> Test mix contains UI tests	(Optional) To run UI tests, ensure that the agent is set to <a href="#">run in interactive mode</a> 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.
<code>vstestLocationMethod</code> Select test platform using	(Optional) Specify which test platform should be used.
<code>vsTestVersion</code> 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 <a href="#">Installed by tools installer</a> option in the UI or toolsInstaller in YAML. Be sure to include the 'Visual Studio Test Platform Installer' task to acquire the test platform from NuGet.
<code>vstestLocation</code> Path to vstest.console.exe	(Optional) Specify the path to VSTest.
<code>runSettingsFile</code> Settings file	(Optional) Path to <code>runsettings</code> or <code>testsettings</code> 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 <code>.testsettings</code> file to a <code>.runsettings</code> file, see <a href="#">this topic</a> .
<code>overrideTestRunParameters</code> Override test run parameters	(Optional) Override parameters defined in the TestRunParameters section of runsettings file or Properties section of testsettings file. For example: <code>-key1 value1 -key2 value2</code> . <b>Note:</b> Properties specified in <code>testsettings</code> file can be accessed via the <code>TestContext</code> using Visual Studio 2017 Update 4 or higher
<code>pathToCustomTestAdapters</code> 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.
<code>runInParallel</code> 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 MaxCpuCount if specified in your runsettings file. Click <a href="#">here</a> to learn more about how tests are run in parallel.
<code>runTestsInIsolation</code> 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.
<code>codeCoverageEnabled</code> Code coverage enabled	(Optional) Collect code coverage information from the test run.

ARGUMENT	DESCRIPTION
<code>otherConsoleOptions</code> Other console options	(Optional) Other console options that can be passed to <code>vstest.console.exe</code> , as documented <a href="#">here</a> . These options are not supported and will be ignored when running tests using the <b>Multi agent</b> parallel setting of an agent job or when running tests using <b>Test plan</b> option. The options can be specified using a settings file instead.
<code>distributionBatchType</code> Batch tests	(Optional) A batch is a group of tests. A batch of tests runs its tests at the same time and results are published for the 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. <b>Based on the number of tests and agents:</b> Simple batching based on the number of tests and agents participating in the test run. <b>Based on past running time of tests:</b> This batching considers past running time to create batches of tests such that each batch has approximately equal running time. <b>Based on test assemblies:</b> Tests from an assembly are batched together. Default value: <code>basedOnTestCases</code>
<code>batchingBasedOnAgentsOption</code> Batch options	(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 $(\text{total number of tests} / \text{number of agents})$ tests. If a batch size is specified, each batch will contain the specified number of tests. Default value: <code>autoBatchSize</code>
<code>customBatchSizeValue</code> Number of tests per batch	(Required) Specify batch size Default value: <code>10</code>
<code>batchingBasedOnExecutionTimeOption</code> Batch options	(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. Default value: <code>autoBatchSize</code>
<code>customRunTimePerBatchValue</code> Running time (sec) per batch	(Required) Specify the running time (sec) per batch Default value: <code>60</code>
<code>dontDistribute</code> Replicate tests instead of distributing when multiple agents are used in the job	(Optional) Choosing this option will not distribute tests across agents when the task is running in a multi-agent job. 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. Default value: <code>False</code>
<code>testRunTitle</code> Test run title	(Optional) Provide a name for the test run

ARGUMENT	DESCRIPTION
<code>platform</code> 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.
<code>configuration</code> 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.
<code>publishRunAttachments</code> Upload test attachments	(Optional) Opt in/out of publishing run level attachments. Default value: <code>true</code>
<code>failOnMinTestsNotRun</code> Fail the task if a minimum number of tests are not run	(Optional) Use this option to fail the task if a minimum number of tests are not run. This may be useful if any changes to task inputs or underlying test adapter dependencies lead to only a subset of the desired tests to be found. Default value: <code>False</code>
<code>minimumExpectedTests</code> Minimum # of tests	(Optional) Specify the minimum # of tests that should be run for the task to succeed. Total tests run is calculated as the sum of passed, failed and aborted tests. Default value: <code>1</code>
<code>diagnosticsEnabled</code> Collect advanced diagnostics in case of catastrophic failures	(Optional) Use this option to turn on collection of diagnostic data to troubleshoot catastrophic failures such as test crash. When this option is checked, a sequence XML file is generated and attached to the test run. The sequence file contains information about the sequence in which tests ran, so that a potentially culprit test can be identified. Default value: <code>false</code>
<code>collectDumpOn</code> Collect process dump and attach to test run report	(Optional) Use this option to collect a mini-dump that can be used for further analysis. <b>On abort only:</b> mini-dump will be collected only when test run is aborted. <b>Always:</b> mini-dump will always be collected regardless of whether the test run completes or not. <b>Never:</b> mini-dump will not be collected regardless of whether the test run completes or not
<code>rerunFailedTests</code> Rerun failed tests	(Optional) Selecting this option will rerun any failed tests until they pass or the maximum # of attempts is reached. Default value: <code>False</code>
<code>rerunType</code> 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 with <code>basedOnTestFailurePercentage</code> or # of failed tests as a threshold with <code>basedOnTestFailureCount</code> . Default value: <code>basedOnTestFailurePercentage</code>
<code>rerunFailedThreshold</code> % 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 Default value: <code>30</code>

ARGUMENT	DESCRIPTION
<pre>rerunFailedTestCasesMaxLimit</pre> # 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 and if rerunType is rerunFailedTestCasesMaxLimit. Default value: <input type="text" value="5"/>
<pre>rerunMaxAttempts</pre> 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. Default value: <input type="text" value="3"/>

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### How can I run tests that use TestCase as a data source?

To run automated tests that use TestCase as a data source, the following is needed:

1. You must have Visual Studio 2017.6 or higher on the agent machine. Visual Studio Test Platform Installer task cannot be used to run tests that use TestCase as a data source.
2. Create a [PAT](#) that is authorized for the scope "Work Items (full)".
3. Add a secure Build or Release variable called Test.TestCaseAccessToken with the value set to the PAT created in the previous step.

### I am running into issues when running data-driven xUnit and NUnit tests with some of the task options. Are there known limitations?

Data-driven tests that use xUnit and NUnit test frameworks have some known limitations and cannot be used with the following task options:

1. Rerun failed tests.
2. Distributing tests on multiple agents and batching options.
3. Test Impact Analysis

The above limitations are because of how the adapters for these test frameworks discover and report data-driven tests.

# Visual Studio Test Agent Deployment task

11/2/2020 • 8 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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 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 Prerequisites**. 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: Instead, use the 'Visual Studio Test' task to run unit and functional 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
<b>Machines</b>	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: <ul style="list-style-type: none"><li>- The name of an <a href="#">Azure Resource Group</a>.</li><li>- A comma-delimited list of machine names. Example: dbserver.fabrikam.com,dbserver_int.fabrikam.com:5986,192.168.34:5</li><li>- An output variable from a previous task.</li></ul>
<b>Admin Login</b>	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. <ul style="list-style-type: none"><li>- Formats such as <code>username</code>, <code>domain\username</code>, <code>machine-name\username</code>, and <code>.\\username</code> are supported.</li><li>- UPN formats such as <code>username@domain.com</code> and built-in system accounts such as <code>NT Authority\System</code> are not supported.</li></ul>
<b>Password</b>	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: \$(passwordVariable)
<b>Protocol</b>	The protocol that will be used to connect to the target host, either <code>HTTP</code> or <code>HTTPS</code> .
<b>Agent Configuration - Username</b>	Required. The username that the test agent will use. Must be an account on the test machines that has administrative permissions. <ul style="list-style-type: none"><li>- Formats such as <code>username</code>, <code>domain\username</code>, <code>machine-name\username</code>, and <code>.\\username</code> are supported.</li><li>- UPN formats such as <code>username@domain.com</code> and built-in system accounts such as <code>NT Authority\System</code> are not supported.</li></ul>

ARGUMENT	DESCRIPTION
<b>Agent Configuration - Password</b>	Required. The password for the <b>Username</b> for the test agent. To protect the password, create a <a href="#">variable</a> and use the "padlock" icon to hide it.
<b>Agent Configuration - Run UI tests</b>	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.
<b>Agent Configuration - Enable data collection only</b>	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 FAQ section below.
<b>Advanced - Test agent version</b>	The version of the test agent to use.
<b>Advanced - Test agent location</b>	Optional. The path to the test agent ( <code>vstf_testagent.exe</code> ) 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.
<b>Advanced - Update test agent</b>	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.
<b>Control options</b>	See <a href="#">Control options</a>

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.

## FAQ

### 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.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

### 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
- Get advice on [Stack Overflow](#), and get support via the [Developer Community](#)

# Xamarin Test Cloud task

11/2/2020 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

This task was used in a build or release pipeline to test mobile apps with Xamarin Test Cloud using Xamarin.UITest

## NOTE

This task is deprecated. Use the [App Center Test](#) task instead. Sign up with App Center first.

## Demands

None

## YAML snippet

```
Xamarin Test Cloud
[Deprecated] Test mobile apps with Xamarin Test Cloud using Xamarin.UITest. Instead, use the 'App Center test' task.
- task: XamarinTestCloud@1
 inputs:
 appFile:
 #dsymFile: # Optional
 teamApiKey:
 email:
 devices:
 #series: 'master'
 testAssemblyDirectory:
 #parallelizationOption: 'none' # Options: none, --Fixture-Chunk, --Test-Chunk
 #localeOption: 'en_US' # Options: da_DK, nl_NL, en_GB, en_US, fr_FR, de_DE, ja_JP, ru_RU, es_MX, es_ES,
 user
 #userDefinedLocale: # Optional
 #testCloudFile: '**/packages/**/tools/test-cloud.exe'
 #optionalArgs: # Optional
 #publishNUnitResults: true # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<b>App File</b>	<ul style="list-style-type: none"><li>- If you want to test a single app, choose ... and select the .apk file.</li><li>- If you want to test multiple apps, specify a match pattern filter. You can use a single-folder wildcard (*) and recursive wildcards (**). For example, **/*.apk searches for all .apk files in all subdirectories.</li></ul>
<b>dSYM File</b>	To make crash logs easier to read, upload a .dSYM file that is associated with your app. Specify a path relative to the .ipa file. You can use a match pattern filter. For example: *.dSYM <b>Note:</b> This argument applies only to iOS apps.

ARGUMENT	DESCRIPTION
Team API Key	Your Xamarin Test Cloud Team API key. To get it, go to your <a href="#">Xamarin Test Cloud account</a> , and then to <b>Teams &amp; Apps</b> . <b>Important:</b> Use a <a href="#">secret variable</a> to avoid exposing this value.
User Email	Email address of your <a href="#">Xamarin Test Cloud account</a> .
Devices	The devices string generated by Xamarin Test Cloud. To get this string: - Go to your <a href="#">Xamarin Test Cloud account</a> . - Choose <b>New Test Run</b> . - Step through the wizard. - When you reach the final page, copy the ID that follows the <b>--devices</b> option.
Series	Series name for the test run. For example, <code>master</code> , <code>production</code> , or <code>beta</code> . See <a href="#">Xamarin: Creating A Test Run for a Team</a> .
Test Assembly Directory	Relative path to the folder that contains the test assemblies. For example: <code>SolutionName/TestsProjectName/bin/Release</code> .
Advanced - Parallelization	Select None, By test fixture, or By test method.
Advanced - System Language	Select your language. If it isn't displayed, select Other and then enter its locale below. For example: <code>en_AU</code> .
Advanced - test-cloud.exe Location	Location of test-cloud.exe. In most cases leave this set to the default value.
Advanced - Optional Arguments	(Optional) Arguments passed to test-cloud.exe. See <a href="#">Submitting UITests at the Command Line</a> .
Advanced - Publish results to VSO/TFS	Select if you want to pass the <b>--nunit-xml</b> option to test-cloud.exe so that results from the NUnit xml file are published to TFS or Azure Pipelines.
Control options	See <a href="#">Control options</a>

## Example

[Build your Xamarin app](#)

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# FAQ

## How do I add a Xamarin UITest to my solution?

[Adding Xamarin.UITest to a Solution](#)

## Where can I learn more about file matching patterns?

[File matching patterns reference](#)

## Do I need an agent?

You need at least one [agent](#) to run your build or release.

## I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

## 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](#).

## My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE.EXTRA.CA.CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

## 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](#).

# CocoaPods task

6/2/2020 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

Use this task 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
Install CocoaPods dependencies for Swift and Objective-C Cocoa projects
- task: CocoaPods@0
 inputs:
 #workingDirectory: # Optional
 forceRepoUpdate:
 #projectDirectory: # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<code>cwd</code> Working directory	(Optional) Specify the working directory in which to execute this task. If left empty, the repository directory will be used. Argument alias: <code>workingDirectory</code>
<code>forceRepoUpdate</code> Force repo update	(Required) Selecting this option will force running 'pod repo update' before install. Default value: <code>false</code>
<code>projectDirectory</code> 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.

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### What other kinds of apps can I build?

[Build and deploy your app examples](#)

## What other kinds of build tasks are available?

[Build and release tasks catalog](#)

## 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?

- [Build and release 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.
- [Supported source repositories](#) 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.

## I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

## 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](#).

## My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

## 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

11/2/2020 • 3 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task to create and activate a Conda environment.

### NOTE

This task has been deprecated. Use `conda` directly in the [bash task](#) or [batch script task](#) as an alternative.

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
This task is deprecated. Use `conda` directly in script to work with Anaconda environments.
- task: CondaEnvironment@1
 inputs:
 #createCustomEnvironment: # Optional
 #environmentName: # Required when createCustomEnvironment == True
 #packageSpecs: 'python=3' # Optional
 #updateConda: true # Optional
 #installOptions: # Optional
 #createOptions: # Optional
 #cleanEnvironment: # Optional

```

## Arguments

ARGUMENT	DESCRIPTION
<code>createCustomEnvironment</code> Create custom environment	(Optional) 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. Default value: <code>false</code>
<code>environmentName</code> Environment name	(Required) Name of the Conda environment to create and activate.
<code>packageSpecs</code> Package specs	(Optional) Space-delimited list of packages to install when creating the environment. Default value: <code>python=3</code>
<code>updateConda</code> Update to the latest Conda	(Optional) 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. Default value: <code>true</code>
<code>installOptions</code> Other options for <code>conda install</code>	(Optional) Space-delimited list of additional arguments to pass to the <code>conda install</code> command.
<code>createOptions</code> Other options for <code>conda create</code>	(Optional) Space-delimited list of other options to pass to the <code>conda create</code> command.
<code>cleanEnvironment</code> Clean the environment	(Optional) Delete the environment and recreate it if it already exists. If not selected, the task will reactivate an existing environment. Default value: <code>false</code>

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### Do I need an agent?

You need at least one [agent](#) to run your build or release.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

**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](#).

**My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?**

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

**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.

# Package: Maven Authenticate

5/12/2021 • 2 minutes to read • [Edit Online](#)

Provides credentials for Azure Artifacts feeds and external Maven repositories in the current user's settings.xml file.

## YAML snippet

```
Provides credentials for Azure Artifacts feeds and external Maven repositories.
- task: MavenAuthenticate@0
 #inputs:
 #artifactsFeeds: MyFeedInOrg1, MyFeedInOrg2 # Optional
 #mavenServiceConnections: serviceConnection1, serviceConnection2 # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<code>artifactsFeeds</code> My feeds (select below)	(Optional) Comma-separated list of Azure Artifacts feed names to authenticate with Maven. If you only need authentication for external maven repositories, leave this field blank.
<code>mavenServiceConnections</code> Feeds from external organizations	(Optional) Comma-separated list of <a href="#">Maven service connection</a> names from external organizations to authenticate with Maven. If you only needs authentication for Azure Artifacts feeds, leave this field blank.

## Examples

### Authenticate Maven feeds inside your organization

In this example, we authenticate two Azure Artifacts feeds within our organization.

#### Task definition

```
- task: MavenAuthenticate@0
 displayName: 'Maven Authenticate'
 inputs:
 artifactsFeeds: MyFeedInOrg1,MyFeedInOrg2
```

The MavenAuthenticate task updates the settings.xml file present in the agent user's .m2 directory located at `{user.home}/.m2/settings.xml` to add two entries inside the `<servers>` element.

**settings.xml**

```

<servers>
 <server>
 <id>MyFeedInOrg1</id>
 <username>AzureDevOps</username>
 <password>****</password>
 </server>
 <server>
 <id>MyFeedInOrg2</id>
 <username>AzureDevOps</username>
 <password>****</password>
 </server>
</servers>

```

You should set the repositories in your project's `pom.xml` to have the same `<id>` as the name specified in the task for Maven to be able to correctly authenticate the task.

#### pom.xml

Project scoped feed

```

<repository>
 <id>MyFeedInOrg1</id>

 <url>https://pkgs.dev.azure.com/OrganizationName/ProjectName/_packaging/MyProjectScopedFeed1/Maven/v1</url>
 <releases>
 <enabled>true</enabled>
 </releases>
 <snapshots>
 <enabled>true</enabled>
 </snapshots>
</repository>

```

Organization scoped feed

```

<repository>
 <id>MyFeedInOrg1</id>
 <url>https://pkgs.dev.azure.com/OrganizationName/_packaging/MyOrgScopedFeed1/Maven/v1</url>
 <releases>
 <enabled>true</enabled>
 </releases>
 <snapshots>
 <enabled>true</enabled>
 </snapshots>
</repository>

```

The Artifacts feed URL may or may not contain the project. An URL for a project scoped feed must contain the project and a URL for a organization scoped feed must not contain the project. [Learn more](#).

#### Authenticate Maven feeds outside your organization.

In this example, we authenticate two external Maven repositories.

#### Task definition

```

- task: MavenAuthenticate@0
 displayName: 'Maven Authenticate'
 inputs:
 MavenServiceConnections: central,MavenOrg

```

The `MavenAuthenticate` task updates the `settings.xml` file present in the agent users' `.m2` directory located at `{user.home}/.m2/settings.xml` to add two entries inside the `<servers>` element.

## settings.xml

```
<servers>
 <server>
 <id>central</id>
 <username>centralUsername</username>
 <password>****</password>
 </server>
 <server>
 <id>MavenOrg</id>
 <username>mavenOrgUsername</username>
 <password>****</password>
 </server>
</servers>
```

You should set the repositories in your project's `pom.xml` to have the same `<id>` as the name specified in the task for Maven to be able to correctly authenticate the task.

## pom.xml

```
<repository>
 <id>central</id>
 <url>https://repo1.maven.org/maven2/</url>
 <releases>
 <enabled>true</enabled>
 </releases>
 <snapshots>
 <enabled>true</enabled>
 </snapshots>
</repository>
```

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### Where is the `settings.xml` file which contains the authenticated repositories located?

The Maven Authenticate task searches for the `settings.xml` in the current user's home directory. For Linux and Mac, the path is `$HOME/.m2/settings.xml`, for Windows the path is `%USERPROFILE%\.m2\settings.xml`. If the `settings.xml` file doesn't exist a new one will be created at that path.

### We use the `mvn -s` switch to specify our own `settings.xml` file, how do we authenticate Azure Artifacts feeds there?

The Maven Authenticate task doesn't have access to the custom `settings.xml` file that's specified by using an `-s` switch. To add Azure Artifacts authentication to your custom `settings.xml`, add a `server` element inside your `settings.xml` file:

```
<server>
 <id>feedName</id> <!-- Set this to the id of the <repository> element inside your pom.xml file. -->
 <username>AzureDevOps</username>
 <password>${env.SYSTEM_ACCESTOKEN}</password>
</server></pre>
```

The access token variable can be set in your pipelines using these [instructions](#).

### My Pipeline needs to access a feed in a different project

If the pipeline is running in a different project than the project hosting the feed, you must set up the other

project to grant read/write access to the build service. See [Package permissions in Azure Pipelines](#) for more details.

# npm task

5/12/2021 • 4 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

Use this task to install and publish npm packages.

## NOTE

The [npm Authenticate](#) task is the recommended way to authenticate with Azure Artifacts. This task no longer takes new features and only critical bugs are addressed.

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## YAML snippet

```
npm
Install and publish npm packages, or run an npm command. Supports npmjs.com and authenticated registries
like Azure Artifacts.
- 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
 #publishPackageMetadata: true # Optional
 #publishEndpoint: # Required when publishRegistry == UseExternalRegistry
```

## Install npm packages

### Demands

[npm](#)

### Arguments

ARGUMENT	DESCRIPTION
<code>command</code> Command	(Required) npm command to run. Select <code>install</code> here

ARGUMENT	DESCRIPTION
<b>workingDir</b> Working folder that contains <code>package.json</code>	Path to the folder containing the target package.json and .npmrc files. Select the folder, not the file e.g. "/packages/mypackage".
<b>advanced</b> Verbose logging	Select to print more information to the console on run
<b>customRegistries</b> Registries to use	<p>You can either commit a <code>.npmrc</code> file to your source code repository and set its path or select a registry from Azure Artifacts.</p> <p><b>useNpmrc</b></p> <ul style="list-style-type: none"> <li>• Select this option to use feeds specified in a .npmrc file you've checked into source control. If no <code>.npmrc</code> file is present, the task will default to using packages directly from npmjs.</li> <li>• Credentials for registries outside this organization/collection can be used to inject credentials you've provided as an npm service connection into your .npmrc as the build runs.</li> </ul> <p><b>useFeed</b></p> <ul style="list-style-type: none"> <li>• Select this option to use one Azure Artifacts feed in the same organization/collection as the build.</li> </ul>

## Publish npm packages

### Demands

npm

### Arguments

ARGUMENT	DESCRIPTION
<b>command</b> Command	(Required) npm command to run. Select publish here.
<b>workingDir</b> Working folder that contains <code>package.json</code>	Path to the folder containing the target package.json and .npmrc files. Select the folder, not the file e.g. "/packages/mypackage".
<b>advanced</b> Verbose logging	Select to print more information to the console on run

ARGUMENT	DESCRIPTION
<p><code>customRegistries</code></p> <p>Registries to use</p>	<p>You can either commit a <code>.npmrc</code> file to your source code repository and set its path or select a registry from Azure Artifacts.</p> <p><b>useNpmrc</b></p> <ul style="list-style-type: none"> <li>• Select this option to use feeds specified in a <code>.npmrc</code> file you've checked into source control. If no <code>.npmrc</code> file is present, the task will default to using packages directly from npmjs.</li> <li>• 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.</li> </ul> <p><b>useFeed</b></p> <ul style="list-style-type: none"> <li>• Select this option to use one Azure Artifacts feed in the same organization/collection as the build.</li> </ul>

## Custom npm command

### Demands

[npm](#)

### Arguments

ARGUMENT	DESCRIPTION
<p><code>command</code></p> <p>Command</p>	(Required) npm command to run. Select custom here.
<p><code>workingDir</code></p> <p>Working folder that contains <code>package.json</code></p>	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> .
<p><code>customCommand</code></p> <p>Command and arguments</p>	<p>(Required) Custom command to run, e.g. "dist-tag ls mypackage".</p> <p>If your arguments contain double quotes (""), escape them with a slash (\), and surround the escaped string with double quotes ("").</p> <p><b>Example:</b> to run</p> <pre>npm run myTask -- --users='{"foo":"bar"}'</pre> <p>, provide this input:</p> <pre>run myTask -- --users="\"foo\":\"bar\""</pre>

ARGUMENT	DESCRIPTION
<p><code>customRegistries</code> Registries to use</p>	<p>You can either commit a <code>.npmrc</code> file to your source code repository and set its path or select a registry from Azure Artifacts.</p> <p><b>useNpmrc</b></p> <ul style="list-style-type: none"> <li>• Select this option to use feeds specified in a <code>.npmrc</code> file you've checked into source control. If no <code>.npmrc</code> file is present, the task will default to using packages directly from npmjs.</li> <li>• 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.</li> </ul> <p><b>useFeed</b></p> <ul style="list-style-type: none"> <li>• Select this option to use one Azure Artifacts feed in the same organization/collection as the build.</li> </ul>

## Examples

Build: gulp

Build your Node.js app with gulp

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### 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.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

### 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](#).

### My Pipeline needs to access a feed in a different project

If the pipeline is running in a different project than the project hosting the feed, you must set up the other project to grant read/write access to the build service. See [Package permissions in Azure Pipelines](#) for more details.

# Package: npm Authenticate task (for task runners)

5/12/2021 • 4 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task to provide npm credentials to an `.npmrc` file in your repository for the scope of the build. This enables npm, as well as npm task runners like gulp and Grunt, to authenticate with private registries.

## YAML snippet

```
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. This enables npm and npm task runners like gulp and Grunt to
authenticate with private registries.
- task: npmAuthenticate@0
 inputs:
 #workingFile:
 #customEndpoint: # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<code>workingFile</code> .npmrc file to authenticate	Path to the .npmrc file that specifies the registries you want to work with. Select the file, not the folder. For example /packages/mypackage.npmrc"
<code>customEndpoint</code> Credentials for registries outside this organization/collection	(Optional) Comma-separated list of <a href="#">npm service connection</a> names for registries outside this organization/collection. The specified <code>.npmrc</code> file must contain registry entries corresponding to the service connections. If you only need registries in this organization/collection, leave this blank. The build's credentials are used automatically.

## Examples

### Restore npm packages for your project from a registry within your organization

If the only authenticated registries you use are Azure Artifacts registries in your organization, you only need to specify the path to an .npmrc file to the npmAuthenticate task.

#### .npmrc

```
registry=https://pkgs.dev.azure.com/{organization}/_packaging/{feed}/npm/registry/
always-auth=true
```

#### npm

```
- task: npmAuthenticate@0
 inputs:
 workingFile: .npmrc
- script: npm ci
...
- script: npm publish
```

## Restore and publish npm packages outside your organization

If your `.npmrc` contains Azure Artifacts registries from a different organization or use a third-party authenticated package repository, you'll need to set up [npm service connections](#) and specify them in the `customEndpoint` input. Registries within your Azure Artifacts organization will also be automatically authenticated.

### `.npmrc`

```
registry=https://pkgs.dev.azure.com/{organization}/{project}/_packaging/{feed}/npm/registry/
#{@scope}:registry=https://pkgs.dev.azure.com/{otherorganization}/_packaging/{feed}/npm/registry/
@{otherscope}:registry=https:///{thirdPartyRepository}/npm/registry/
always-auth=true
```

The registry URL pointing to an Azure Artifacts feed may or may not contain the project. An URL for a project scoped feed must contain the project, and the URL for a organization scoped feed must not contain the project.

[Learn more](#).

### `npm`

```
- task: npmAuthenticate@0
 inputs:
 workingFile: .npmrc
 customEndpoint: OtherOrganizationNpmConnection, ThirdPartyRepositoryNpmConnection
- script: npm ci
...
- script: npm publish -registry
 https://pkgs.dev.azure.com/{otherorganization}/_packaging/{feed}/npm/registry/
```

`OtherOrganizationNpmConnection` and `ThirdPartyRepositoryNpmConnection` are the names of [npm service connections](#) that have been configured and authorized for use in your pipeline, and have URLs that match those in the specified `.npmrc` file.

## Control options

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### How does this task work?

This task searches the specified `.npmrc` file for registry entries, then appends authentication details for the discovered registries to the end of the file. For all registries in the current organization/collection, the build's credentials are used. For registries in a different organization or hosted by a third-party, the registry URLs will be compared to the URLs of the [npm service connections](#) specified by the `customEndpoint` input, and the corresponding credentials will be used. The `.npmrc` file will be reverted to its original state at the end of the pipeline execution.

### When in my pipeline should I run this task?

This task must run before you use npm, or an npm task runner, to install or push packages to an authenticated npm repository such as Azure Artifacts. There are no other ordering requirements.

### I have multiple npm projects. Do I need to run this task for each .npmrc file?

This task will only add authentication details to one `.npmrc` file at a time. If you need authentication for multiple `.npmrc` files, you can run the task multiple times, once for each `.npmrc` file. Alternately, consider creating an `.npmrc` file that specifies all registries used by your projects, running `npmAuthenticate` on this `.npmrc` file, then setting an environment variable to designate this `.npmrc` file as the npm per-user configuration file.

```
- task: npmAuthenticate@0
 inputs:
 workingFile: $(agent.tempdirectory)/.npmrc
- script: echo ##vso[task.setvariable variable=NPM_CONFIG_USERCONFIG]$(agent.tempdirectory)/.npmrc
- script: npm ci
 workingDirectory: project1
- script: npm ci
 workingDirectory: project2
```

### My agent is behind a web proxy. Will `npmAuthenticate` set up npm/gulp/Grunt to use my proxy?

The answer is no. While this task itself will work behind a web proxy [your agent has been configured to use](#), it does not configure npm or npm task runners to use the proxy.

To do so, you can either:

- Set the environment variables `http_proxy` / `https_proxy` and optionally `no_proxy` to your proxy settings. See [npm config](#) for details. Note that these are commonly used variables which other non-npm tools (e.g. curl) may also use.
- Add the proxy settings to the [npm configuration](#), either manually, by using [npm config set](#), or by setting [environment variables](#) prefixed with `NPM_CONFIG_`.

#### Caution:

npm task runners may not be compatible with all methods of proxy configuration supported by npm.

- Specify the proxy with a command line flag when calling npm

```
- script: npm ci --https-proxy $(agent.proxyurl)
```

If your proxy requires authentication, you may need to add an additional build step to construct an authenticated proxy uri.

```
- script: node -e "let u = url.parse(`$(agent.proxyurl)`); u.auth =
`$(agent.proxyusername):$(agent.proxypassword)`;
 console.log(`##vso[task.setvariable
variable=proxyAuthUri;issecret=true]` + url.format(u))"
- script: npm publish --https-proxy $(proxyAuthUri)
```

### My Pipeline needs to access a feed in a different project

If the pipeline is running in a different project than the project hosting the feed, you must set up the other project to grant read/write access to the build service. See [Package permissions in Azure Pipelines](#) for more details.

# NuGet task

5/12/2021 • 11 minutes to read • [Edit Online](#)

Version 2.

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018

## NOTE

The [NuGet Authenticate](#) task is the new recommended way to authenticate with Azure Artifacts and other NuGet repositories. This task no longer takes new features and only critical bugs are addressed.

Use this task to install and update NuGet package dependencies, or package and publish NuGet packages. Uses NuGet.exe and works with .NET Framework apps. For .NET Core and .NET Standard apps, use the .NET Core task.

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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.

If you are working with .NET Core or .NET Standard, use the [.NET Core](#) task, which has full support for all package scenarios and it's currently supported by dotnet.

## 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](#).

## YAML snippet

```

NuGet
Restore, pack, or push NuGet packages, or run a NuGet command. Supports NuGet.org and authenticated feeds
like Azure Artifacts 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
 #publishPackageMetadata: true # Optional
 #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, specify path to nuspec files
 #verbosityPack: 'Detailed' # Options: quiet, normal, detailed
 #arguments: # Required when command == Custom

```

## Arguments

ARGUMENT	DESCRIPTION
<code>command</code> Command	The NuGet command to run. Select 'Custom' to add arguments or to use a different command. Options: <code>restore</code> , <code>pack</code> , <code>custom</code> , <code>push</code>
<code>restoreSolution</code> Path to solution, packages.config, or project.json	The path to the solution, packages.config, or project.json file that references the packages to be restored.
<code>feedsToUse</code> Feeds to use	You can either select a feed from Azure Artifacts and/or NuGet.org, or commit a nuget.config file to your source code repository and set its path here. Options: <code>select</code> , <code>config</code>

ARGUMENT	DESCRIPTION
<code>vstsFeed</code> Use packages from this Azure Artifacts/TFS feed	Include the selected feed in the generated NuGet.config. You must have Azure Artifacts installed and licensed to select a feed here.
<code>includeNuGetOrg</code> Use packages from NuGet.org	Include NuGet.org in the generated NuGet.config. Default value is <code>true</code> . Required when <code>feedsToUse</code> == <code>Select</code> .
<code>nugetConfigPath</code> Path to NuGet.config	The NuGet.config in your repository that specifies the feeds from which to restore packages. Required when <code>feedsToUse</code> == <code>Config</code>
<code>externalFeedCredentials</code> Credentials for feeds outside this organization/collection	Credentials to use for external registries located in the selected NuGet.config. This is the name of your NuGet service connection. For feeds in this organization/collection, leave this blank; the build's credentials are used automatically.
<code>noCache</code> Disable local cache	Prevents NuGet from using packages from local machine caches.
<code>disableParallelProcessing</code> Disable parallel processing	Prevents NuGet from installing multiple packages in parallel.
<code>restoreDirectory</code> 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.
<code>verbosityRestore</code> Verbosity	Specifies the amount of detail displayed in the output. Options: <code>Quiet</code> , <code>Normal</code> , <code>Detailed</code>
<code>nuGetFeedType</code> Target feed location	Specifies whether the target feed is an internal feed/collection or an external NuGet server. Options: <code>internal</code> , <code>external</code> . Required when <code>command</code> == <code>Push</code>
<code>publishVstsFeed</code> Target feed	Select a feed hosted in this account. You must have Azure Artifacts installed and licensed to select a feed here.
<code>packagesToPush</code> Packages location	The pattern to match or path to the nupkg files, e.g.: <code>'\$(Build.ArtifactStagingDirectory)/*.nupkg'</code> . Required when <code>command</code> == <code>Push</code>
<code>publishPackageMetadata</code> Publish pipeline metadata	If you continually publish a set of packages and only change the version number of the subset of packages that changed, use this option.
<code>allowPackageConflicts</code>	It allows the task to report success even if some of your packages are rejected with 409 Conflict errors. If NuGet.exe encounters a conflict, the task will fail. This option will not work and publish will fail if you are within a proxy environment.

ARGUMENT	DESCRIPTION
<code>publishFeedCredentials</code> NuGet server	The NuGet service connection that contains the external NuGet server's credentials.
<code>verbosityPush</code> Verbosity	Specifies the amount of detail displayed in the output. Options: <code>Quiet</code> , <code>Normal</code> , <code>Detailed</code>
<code>packagesToPack</code> Path to csproj or nuspec file(s) to pack	Pattern to search for csproj directories to pack. You can separate multiple patterns with a semicolon, and you can make a pattern negative by prefixing it with '!'. Example: <code>**\*.csproj;!**\*.Tests.csproj</code>
<code>configuration</code> Configuration to package	When using a csproj file this specifies the configuration to package.
<code>packDestination</code> Package folder	Folder where packages will be created. If empty, packages will be created at the source root.
<code>versioningScheme</code> Automatic package versioning	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. <b>Note:</b> Under Options set the build number format to be <code>'\$(BuildDefinitionName)_\$(Year:yyyy).\$(Month).\$(DayOfMonth:h)\$(Rev:r)'</code> . Options: <code>off</code> , <code>byPrereleaseNumber</code> , <code>byEnvVar</code> , <code>byBuildNumber</code>
<code>includeReferencedProjects</code> Environment variable	Enter the variable name without \$, \$env, or %.
<code>majorVersion</code> Major	The 'X' in version <code>X.Y.Z</code>
<code>minorVersion</code> Minor	The 'Y' in version <code>X.Y.Z</code>
<code>patchVersion</code> Patch	The 'Z' in version <code>X.Y.Z</code>
<code>packTimezone</code> Time zone	Specifies the desired time zone used to produce the version of the package. Selecting UTC is recommended if you're using hosted build agents as their date and time might differ. Options: <code>utc</code> , <code>local</code>
<code>includeSymbols</code> Create symbols package	Specifies that the package contains sources and symbols. When used with a .nuspec file, this creates a regular NuGet package file and the corresponding symbols package.

ARGUMENT	DESCRIPTION
<code>toolPackage</code> Tool Package	Determines if the output files of the project should be in the tool folder.
<code>buildProperties</code> Additional build properties	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.
<code>basePath</code> Base path	The base path of the files defined in the nuspec file.
<code>verbosityPack</code> Verbosity	Specifies the amount of detail displayed in the output. Options: <code>Quiet</code> , <code>Normal</code> , <code>Detailed</code>
<code>arguments</code> Command and arguments	The command and arguments which will be passed to NuGet.exe for execution. If NuGet 3.5 or later is used, authenticated commands like list, restore, and publish against any feed in this organization/collection that the Project Collection Build Service has access to will be automatically authenticated.
Control options	

## 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 to the value of the environment variable that has the name specified by the `versionEnvVar` parameter, e.g. `MyVersion` (no \$, just the environment variable name). 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 using the pipeline run's build number. This is the value specified for the pipeline's `name` property, which gets saved to the `BUILD_BUILDNUMBER` environment variable). Ensure that the build number being used contains a proper SemVer e.g. `1.0.$(Rev:r)`. When using `byBuildNumber`, the task will extract the dotted version, `1.2.3.4` from the build number string, and use only that portion. The rest of the string will be dropped. If you want to use the build number as is, you can use `byEnvVar` as described above, and set `versionEnvVar` to `BUILD_BUILDNUMBER`.

## Examples

### Restore

Restore all your solutions with packages from a selected feed.

```
Restore from a project scoped feed in the same organization
- task: NuGetCommand@2
 inputs:
 command: 'restore'
 feedsToUse: 'select'
 vstsFeed: 'my-project/my-project-scoped-feed'
 includeNuGetOrg: false
 restoreSolution: '**/*.sln'
```

```
Restore from an organization scoped feed in the same organization
- task: NuGetCommand@2
 inputs:
 command: 'restore'
 feedsToUse: 'select'
 vstsFeed: 'my-organization-scoped-feed'
 restoreSolution: '**/*.sln'
```

```
Restore from a feed in a different organization
- task: NuGetCommand@2
 inputs:
 command: 'restore'
 feedsToUse: config
 nugetConfigPath: ./nuget.config
 restoreSolution: '**/*.sln'
 externalFeedCredentials: 'MyServiceConnectionName'
 noCache: true
 continueOnError: true
```

```
Restore from feed(s) set in nuget.config
- task: NuGetCommand@2
 inputs:
 command: 'restore'
 feedsToUse: 'config'
 nugetConfigPath: 'nuget.config'
```

## Package

Create a NuGet package in the destination folder.

```
Package a project
- task: NuGetCommand@2
 inputs:
 command: 'pack'
 packagesToPack: '**/*.csproj'
 packDestination: '$(Build.ArtifactStagingDirectory)'
```

## Push

### NOTE

Pipeline artifacts are downloaded to `System.ArtifactsDirectory` directory. `packagesToPush` value can be set to `$(System.ArtifactsDirectory)/**/*.nupkg` in your release pipeline.

- Push/Publish a package to a feed defined in your NuGet.config.

```
Push a project
- task: NuGetCommand@2
 inputs:
 command: 'push'
 packagesToPush: '$(Build.ArtifactStagingDirectory)/**/*.nupkg'
 feedsToUse: 'config'
 nugetConfigPath: '$(Build.WorkingDirectory)/NuGet.config'
```

- Push/Publish a package to a project scoped

```
Push a project
- task: NuGetCommand@2
 inputs:
 command: 'push'
 feedsToUse: 'select'
 vstsFeed: 'my-project/my-project-scoped-feed'
 publishVstsFeed: 'myTestFeed'
```

- Push/Publish a package to NuGet.org

```
Push a project
- task: NuGetCommand@2
 inputs:
 command: 'push'
 feedsToUse: 'config'
 includeNugetOrg: 'true'
```

## Custom

Run any other NuGet command besides the default ones: pack, push, and restore.

```
list local NuGet resources.
- task: NuGetCommand@2
 displayName: 'list locals'
 inputs:
 command: custom
 arguments: 'locals all -list'
```

## Open-source

Check out the Azure Pipelines and Team Foundation Server out-of-the-box tasks [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### 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?

[Azure Artifacts Documentation](#)

### 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 and deploy your app examples](#)

### What other kinds of build tasks are available?

## Build and release tasks catalog

### 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?

- [Build and release 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.
- [Supported source repositories](#) 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.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE.EXTRA.CA.CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

### 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](#).

### My Pipeline needs to access a feed in a different project

If the pipeline is running in a different project than the project hosting the feed, you must set up the other project to grant read/write access to the build service. See [Package permissions in Azure Pipelines](#) for more details.

# Package: NuGet Authenticate

5/12/2021 • 8 minutes to read • [Edit Online](#)

## Azure Pipelines

Configure NuGet tools to authenticate with Azure Artifacts and other NuGet repositories.

### IMPORTANT

This task is only compatible with NuGet >= 4.8.0.5385, dotnet >= 2.1.400, or MSBuild >= 15.8.166.59604

## YAML snippet

```
Authenticate nuget.exe, dotnet, and MSBuild with Azure Artifacts and optionally other repositories
- task: NuGetAuthenticate@0
 #inputs:
 #nuGetServiceConnections: MyOtherOrganizationFeed, MyExternalPackageRepository # Optional
 #forceReinstallCredentialProvider: false # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<code>nuGetServiceConnections</code> Service connection credentials for feeds outside this organization	(Optional) Comma-separated list of <a href="#">NuGet service connection</a> names for feeds outside this organization/collection to additionally set up. If you only need feeds in this organization/collection, leave this blank; the build's credentials are used automatically.
<code>forceReinstallCredentialProvider</code> Reinstall the credential provider even if already installed	(Optional) Reinstall the credential provider to the user profile directory even if already installed. This may upgrade (or potentially downgrade) the credential provider.
<a href="#">Control options</a>	

## Examples

### Restore and push NuGet packages within your organization

If all of the Azure Artifacts feeds you use are in the same organization as your pipeline, you can use the NuGetAuthenticate task without specifying any inputs. For project scoped feeds that are in a different project than where the pipeline is running in, you must manually give the project and the feed access to the pipeline's project's build service.

[nuget.config](#)

```

<configuration>
 <packageSources>
 <!--
 Any Azure Artifacts feeds within your organization will automatically be authenticated. Both
 dev.azure.com and visualstudio.com domains are supported.

 Project scoped feed URL includes the project, organization scoped feed URL does not.
 -->
 <add key="MyProjectFeed1"
 value="https://pkgs.dev.azure.com/{organization}/{project}/_packaging/{feed}/nuget/v3/index.json" />
 <add key="MyProjectFeed2"
 value="https:///{organization}.pkgs.visualstudio.com/{project}/_packaging/{feed}/nuget/v3/index.json" />
 <add key="MyOtherProjectFeed1"
 value="https://pkgs.dev.azure.com/{organization}/{project}/_packaging/{feed@view}/nuget/v3/index.json" />
 <add key="MyOrganizationFeed1"
 value="https://pkgs.dev.azure.com/{organization}/_packaging/{feed}/nuget/v3/index.json" />
 </packageSources>
</configuration>

```

To use a service connection, specify the service connection in the `nuGetServiceConnections` input for the NuGet Authenticate task. You can then reference the service connection with `-ApiKey AzureArtifacts` in a task.

#### **nuget.exe**

```

- task: NuGetAuthenticate@0
 inputs:
 nuGetServiceConnections: OtherOrganizationFeedConnection, ThirdPartyRepositoryConnection
- task: NuGetToolInstaller@1 # Optional if nuget.exe >= 4.8.5385 is already on the path
 inputs:
 versionSpec: '*'
 checkLatest: true
- script: nuget restore
...
- script: nuget push -ApiKey AzureArtifacts -Source "MyProjectFeed1" MyProject.*.nupkg

```

#### **dotnet**

```

- task: NuGetAuthenticate@0
 inputs:
 nuGetServiceConnections: OtherOrganizationFeedConnection, ThirdPartyRepositoryConnection
- task: UseDotNet@2 # Optional if the .NET Core SDK is already installed
 inputs:
 version: '2.1.403'
- script: dotnet restore
...
- script: dotnet nuget push --api-key AzureArtifacts --source
 https://pkgs.dev.azure.com/{organization}/_packaging/{feed1}/nuget/v3/index.json MyProject.*.nupkg

```

In the above examples `OtherOrganizationFeedConnection` and `ThirdPartyRepositoryConnection` are the names of [NuGet service connections](#) that have been configured and authorized for use in your pipeline, and have URLs that match those in your nuget.config or command line argument.

The package source URL pointing to an Azure Artifacts feed may or may not contain the project. An URL for a project scoped feed must contain the project, and a URL for a organization scoped feed must not contain the project. [Learn more](#).

#### **Restore and push NuGet packages outside your organization**

If you use Azure Artifacts feeds from a different organization or use a third-party authenticated package repository, you'll need to set up [NuGet service connections](#) and specify them in the `nuGetServiceConnections` input. Feeds within your Azure Artifacts organization will also be automatically authenticated.

#### **nuget.config**

```

<configuration>
 <packageSources>
 <!-- Any Azure Artifacts feeds within your organization will automatically be authenticated -->
 <add key="MyProjectFeed1"
 value="https://pkgs.dev.azure.com/{organization}/{project}/_packaging/{feed}/nuget/v3/index.json" />
 <add key="MyOrganizationFeed"
 value="https://pkgs.dev.azure.com/{organization}/_packaging/{feed}/nuget/v3/index.json" />
 <!-- Any package source listed here whose URL matches the URL of a service connection in
nuGetServiceConnections will also be authenticated.
 The key name here does not need to match the name of the service connection. -->
 <add key="OtherOrganizationFeed"
 value="https://pkgs.dev.azure.com/{otherorganization}/_packaging/{feed}/nuget/v3/index.json" />
 <add key="ThirdPartyRepository" value="https://{{thirdPartyRepository}}/index.json" />
 </packageSources>
</configuration>

```

## nuget.exe

```

- task: NuGetAuthenticate@0
 inputs:
 nuGetServiceConnections: OtherOrganizationFeedConnection, ThirdPartyRepositoryConnection
- task: NuGetToolInstaller@1 # Optional if nuget.exe >= 4.8.5385 is already on the path
 inputs:
 versionSpec: '*'
 checkLatest: true
- script: nuget restore
...
- script: nuget push -ApiKey AzureArtifacts -Source "MyProjectFeed1" MyProject.*.nupkg

```

## dotnet

```

- task: NuGetAuthenticate@0
 inputs:
 nuGetServiceConnections: OtherOrganizationFeedConnection, ThirdPartyRepositoryConnection
- task: UseDotNet@2 # Optional if the .NET Core SDK is already installed
 inputs:
 versionSpec: '*'
- script: dotnet restore
...
- script: dotnet nuget push --api-key AzureArtifacts --source "MyProjectFeed1" MyProject.*.nupkg

```

`OtherOrganizationFeedConnection` and `ThirdPartyRepositoryConnection` are the names of [NuGet service connections](#) that have been configured and authorized for use in your pipeline, and have URLs that match those in your nuget.config or command line argument.

The package source URL pointing to an Azure Artifacts feed may or may not contain the project. An URL for a project scoped feed must contain the project, and a URL for a organization scoped feed must not contain the project. [Learn more](#).

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### What tools are compatible with this task?

This task will configure tools that support [NuGet cross platform plugins](#). Today, that includes nuget.exe, dotnet, and recent versions of MSBuild with built-in support for restoring NuGet packages.

Specifically, this task will configure:

- nuget.exe, version 4.8.5385 or higher

- dotnet / .NET Core SDK, version 2.1.400 or higher
- MSBuild, version 15.8.166.59604 or higher

However, upgrading to the latest stable version is recommended if you encounter any issues.

### I get "A task was canceled" errors during a package restore. What should I do?

Known issues in NuGet and in the Azure Artifacts Credential Provider can cause this type of error and updating to the latest nuget may help.

A [known issue](#) in some versions of nuget/dotnet can cause this error, especially during large restores on resource constrained machines. This issue is fixed in [NuGet 5.2](#), as well as .NET Core SDK 2.1.80X and 2.2.40X. If you are using an older version, try upgrading your version of NuGet or dotnet. The [.NET Core Tool Installer](#) task can be used to install a newer version of the .NET Core SDK.

There are also known issues with the Azure Artifacts Credential Provider (installed by this task), including [artifacts-credprovider/#77](#) and [artifacts-credprovider/#108](#). If you experience these issues, ensure you have the latest credential provider by setting the input `forceReinstallCredentialProvider` to `true` in the NuGet Authenticate task. This will also ensure your credential provider is automatically updated as issues are resolved.

If neither of the above resolves the issue, please enable [Plugin Diagnostic Logging](#) and report the issue to [NuGet](#) and the [Azure Artifacts Credential Provider](#).

### How is this task different than the NuGetCommand and DotNetCoreCLI tasks?

This task configures nuget.exe, dotnet, and MSBuild to authenticate with Azure Artifacts or other repositories that require authentication. After this task runs, you can then invoke the tools in a later step (either directly or via a script) to restore or push packages.

The NuGetCommand and DotNetCoreCLI tasks require using the task to restore or push packages, as authentication to Azure Artifacts is only configured within the lifetime of the task. This can prevent you from restoring or pushing packages within your own script. It may also prevent you from passing specific command line arguments to the tool.

The NuGetAuthenticate task is the recommended way to use authenticated feeds within a pipeline.

### When in my pipeline should I run this task?

This task must run before you use a NuGet tool to restore or push packages to an authenticated package source such as Azure Artifacts. There are no other ordering requirements. For example, this task can safely run either before or after a NuGet or .NET Core tool installer task.

### How do I configure a NuGet package source that uses ApiKey ("NuGet API keys"), such as nuget.org?

Some package sources such as nuget.org use API keys for authentication when pushing packages, rather than username/password credentials. Due to limitations in NuGet, this task cannot be used to set up a NuGet service connection that uses an API key.

Instead:

1. Configure a [secret variable](#) containing the ApiKey
2. Perform the package push using `nuget push -ApiKey $(myNuGetApiKey)` or  
`dotnet nuget push --api-key $(myNuGetApiKey)`, assuming you named the variable `myNuGetApiKey`

### My agent is behind a web proxy. Will NuGetAuthenticate set up nuget.exe, dotnet, and MSBuild to use my proxy?

No. While this task itself will work behind a web proxy [your agent has been configured to use](#), it does not configure NuGet tools to use the proxy.

To do so, you can either:

- Set the environment variable `http_proxy` and optionally `no_proxy` to your proxy settings. See [NuGet CLI environment variables](#) for details. Please understand that these are commonly used variables which other non-NuGet tools (e.g. curl) may also use.

**Caution:**

The `http_proxy` and `no_proxy` variables are case-sensitive on Linux and Mac operating systems and must be lowercase. Attempting to use an Azure Pipelines variable to set the environment variable will not work, as it will be converted to uppercase. Instead, set the environment variables on the self-hosted agent's machine and restart the agent.

- Add the proxy settings to the [user-level nuget.config](#) file, either manually or using `nuget config -set` as described in the [nuget.config reference](#) documentation.

**Caution:**

The proxy settings (such as `http_proxy`) must be added to the user-level config. They will be ignored if specified in a different nuget.config file.

## How do I debug if I have issues with this task?

To get verbose logs from the pipeline, add a pipeline variable `system.debug` to true.

## How does this task work?

This task installs the [Azure Artifacts Credential Provider](#) into the NuGet plugins directory if it is not already installed.

It then sets environment variables such as `vss_NUGET_URI_PREFIXES`, `VSS_NUGET_ACESSTOKEN`, and `VSS_NUGET_EXTERNAL_FEED_ENDPOINTS` to configure the credential provider. These variables remain set for the lifetime of the job.

When restoring or pushing packages, a NuGet tool executes the credential provider, which uses the above variables to determine if it should return credentials back to the tool.

See the credential provider documentation for more details.

## My Pipeline needs to access a feed in a different project

If the pipeline is running in a different project than the project hosting the feed, you must set up the other project to grant read/write access to the build service. See [Package permissions in Azure Pipelines](#) for more details.

# Package: PyPI Publisher task (deprecated)

3/8/2021 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task 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 <a href="#">universal wheel</a> package (platform independent) in addition to an sdist package.

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### Do I need an agent?

You need at least one [agent](#) to run your build or release.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE.EXTRA.CA.CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

# Package: Python Pip Authenticate

5/12/2021 • 4 minutes to read • [Edit Online](#)

[Version 1.\\*](#) | [Other versions](#)

## Azure Pipelines

Provides authentication for the `pip` client that can be used to install Python distributions.

## YAML snippet

```
Python pip authenticate V1
Authentication task for the pip client used for installing Python distributions
- task: PipAuthenticate@1
 inputs:
 #artifactFeeds: 'MyFeed, MyTestFeed' # Optional
 #pythonDownloadServiceConnections: pypiOrgFeed, OtherOrganizationFeed # Optional
 #onlyAddExtraIndex: false # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<code>artifactFeeds</code> My feeds	(Optional) Comma-separated list of Azure Artifacts feeds to authenticate with pip.
<code>pythonDownloadServiceConnections</code> Feeds from external organizations	(Optional) Comma-separated list of <a href="#">pip service connection</a> names from external organizations to authenticate with pip.
<code>onlyAddExtraIndex</code> Don't set primary index URL	(Optional) Boolean value, if set to <code>true</code> will force pip to get distributions from official python registry first. By default, it's <code>false</code>

## Examples

### Download python distributions from Azure Artifacts feeds without consulting official python registry

In this example, we are setting authentication for downloading from private Azure Artifacts feeds. The authenticate task creates environment variables `PIP_INDEX_URL` and `PIP_EXTRA_INDEX_URL` that are required to download the distributions. The task sets the variables with auth credentials the task generates for the provided Artifacts feeds. 'HelloTestPackage' has to be present in either 'myTestFeed1' or 'myTestFeed2', otherwise install will fail hard.

For project scoped feeds that are in a different project than where the pipeline is running in, you must manually give the project and the feed access to the pipeline's project's build service.

```

- task: PipAuthenticate@1
 displayName: 'Pip Authenticate'
 inputs:
 # Provide list of feed names which you want to authenticate.
 # Project scoped feeds must include the project name in addition to the feed name.
 artifactFeeds: 'project1/myTestFeed1, myTestFeed2'

 # Use command line tool to 'pip install'.
 - script: |
 pip install HelloTestPackage

```

## Download python distributions from Azure Artifacts feeds consulting official python registry first

In this example, we are setting authentication for downloading from private Azure Artifacts feed but [pypi](#) is consulted first. The authenticate task creates an environment variable `PIP_EXTRA_INDEX_URL` which contain auth credentials required to download the distributions. 'HelloTestPackage' will be downloaded from the authenticated feeds only if it's not present in [pypi](#).

For project scoped feeds that are in a different project than where the pipeline is running in, you must manually give the project and the feed access to the pipeline's project's build service.

```

- task: PipAuthenticate@1
 displayName: 'Pip Authenticate'
 inputs:
 # Provide list of feed names which you want to authenticate.
 # Project scoped feeds must include the project name in addition to the feed name.
 artifactFeeds: 'project1/myTestFeed1, myTestFeed2'
 # Setting this variable to "true" will force pip to get distributions from official python registry
 # first and fallback to feeds mentioned above if distributions are not found there.
 onlyAddExtraIndex: true

 # Use command line tool to 'pip install'.
 - script: |
 pip install HelloTestPackage

```

## Download python distributions from other private python servers

In this example, we are setting authentication for downloading from a external python distribution server. Create a [pip service connection](#) entry for the external service. The authenticate task uses the service connection to create an environment variable `PIP_INDEX_URL` which contain auth credentials required to download the distributions. 'HelloTestPackage' has to be present in 'pypitest' service connection, otherwise install will fail. If you want [pypi](#) to be consulted first, set `onlyAddExtraIndex` to `true`.

```

- task: PipAuthenticate@1
 displayName: 'Pip Authenticate'
 inputs:
 # In this case, name of the service connection is "pypitest".
 pythonDownloadServiceConnections: pypitest

 # Use command line tool to 'pip install'.
 - script: |
 pip install HelloTestPackage

```

## Task versions

### Task: Pip Authenticate

TASK VERSION	AZURE PIPELINES	TFS
1.*	Available	Not supported
0.*	Available	Not supported

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### When in my pipeline should I run this task?

This task must run before you use pip to download python distributions to an authenticated package source such as Azure Artifacts. There are no other ordering requirements. Multiple invocation of this task will not stack credentials. Every run of the task will erase any previously stored credentials.

### My agent is behind a web proxy. Will PipAuthenticate set up pip to use my proxy?

No. While this task itself will work behind a web proxy [your agent has been configured to use](#), it does not configure pip to use the proxy.

To do so, you can either:

- Set the environment variable `http_proxy`, `https_proxy` and optionally `no_proxy` to your proxy settings. See [Pip official guidelines](#) for details. These are commonly used variables which other non-Python tools (e.g. curl) may also use.

**Caution:** The `http_proxy` and `no_proxy` variables are case-sensitive on Linux and Mac operating systems and must be lowercase. Attempting to use an Azure Pipelines variable to set the environment variable will not work, as it will be converted to uppercase. Instead, set the environment variables on the self-hosted agent's machine and restart the agent.

- Add the proxy settings to the [pip config file](#) file using `proxy` key.
- Use the `--proxy` command-line option to specify proxy in the form `[user:passwd@]proxy.server:port`.

### My Pipeline needs to access a feed in a different project

If the pipeline is running in a different project than the project hosting the feed, you must set up the other project to grant read/write access to the build service. See [Package permissions in Azure Pipelines](#) for more details.

# Package: Python Twine Upload Authenticate

5/12/2021 • 3 minutes to read • [Edit Online](#)

Version 1.\* | [Other versions](#)

## 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.

## YAML snippet

```
Python twine upload authenticate V1
Authenticate for uploading Python distributions using twine. Add '-r FeedName/EndpointName --config-file ${PYPIRC_PATH}' to your twine upload command. For feed present in this organization, use the feed name as the repository (-r). Otherwise, use the endpoint name defined in the service connection.
- task: TwineAuthenticate@1
 inputs:
 #artifactFeed: MyTestFeed # Optional
 #pythonUploadServiceConnection: OtherOrganizationFeed # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<code>artifactFeed</code>	(Optional) The Azure Artifacts feed name to authenticate with twine. For project-scoped feeds, use this syntax: <i>projectName/feedName</i>
<code>pythonUploadServiceConnection</code>	(Optional) A <a href="#">twine service connection</a> name from external organization to authenticate with twine. The credentials stored in the endpoint must have package upload permissions.

### TIP

See [organization-scoped feeds vs project-scoped feeds](#) to understand the difference between the two types and learn how to check if your feed is project-scoped or organization-scoped.

## Examples

The following examples demonstrate how to publish python distribution to Azure Artifacts feed and the official python registry.

### **Publish python distribution to Azure Artifacts feed**

In this example, we are setting authentication for publishing to a private Azure Artifacts Feed. The authenticate task creates a `.pypirc` file that contains the auth credentials required to publish a distribution to the feed.

```

Install python distributions like wheel, twine etc
- script: |
 pip install wheel
 pip install twine

Build the python distribution from source
- script: |
 python setup.py bdist_wheel

- task: TwineAuthenticate@1
 displayName: 'Twine Authenticate'
 inputs:
 # In this case, name of the feed is 'myTestFeed' in the project 'myTestProject'. Project is needed because
 # the feed is project scoped.
 artifactFeed: myTestProject/myTestFeed

 # Use command line script to 'twine upload', use -r to pass the repository name and --config-file to pass
 # the environment variable set by the authenticate task.
- script: |
 python -m twine upload -r myTestFeed --config-file $(PYPIRC_PATH) dist/*.whl

```

The `artifactFeed` input will contain the project and the feed name if the feed is project scoped. If the feed is organization scoped, only the feed name must be provided. [Learn more](#).

### Publish python distribution to official python registry

In this example, we are setting authentication for publishing to official python registry. Create a [twine service connection](#) entry for `pypi`. The authenticate task uses that service connection to create a `.pypirc` file that contains the auth credentials required to publish the distribution.

```

Install python distributions like wheel, twine etc
- script: |
 pip install wheel
 pip install twine

Build the python distribution from source
- script: |
 python setup.py bdist_wheel

- task: TwineAuthenticate@1
 displayName: 'Twine Authenticate'
 inputs:
 # In this case, name of the service connection is "pypitest".
 pythonUploadServiceConnection: pypitest

 # Use command line script to 'twine upload', use -r to pass the repository name and --config-file to pass
 # the environment variable set by the authenticate task.
- script: |
 python -m twine upload -r "pypitest" --config-file $(PYPIRC_PATH) dist/*.whl

```

## Task versions: Twine Authenticate

Task Version	Azure Pipelines	TFS
1.*	Available	Not supported
0.*	Available	Not supported

## FAQ

## **When in my pipeline should I run this task?**

This task must run before you use twine to upload python distributions to an authenticated package source such as Azure Artifacts. There are no other ordering requirements. Multiple invocations of this task will not stack credentials. Every run of the task will erase any previously stored credentials.

## **My agent is behind a web proxy. Will TwineAuthenticate set up twine to use my proxy?**

No. While this task itself will work behind a web proxy [your agent has been configured to use](#), it does not configure twine to use the proxy.

## **My Pipeline needs to access a feed in a different project**

If the pipeline is running in a different project than the project hosting the feed, you must set up the other project to grant read/write access to the build service. See [Package permissions in Azure Pipelines](#) for more details.

## **Open-source**

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Universal Package task

5/12/2021 • 7 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018

Use this task to download, or package and publish Universal Packages.

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## YAML snippet

```
Universal packages
Download or publish Universal Packages
- task: UniversalPackages@0
 inputs:
 #command: 'download' # Options: download, publish
 #downloadDirectory: '$(System.DefaultWorkingDirectory)' # Required when command == Download
 #feedsToUse: 'internal' # Options: internal, external
 #externalFeedCredentials: # Optional
 #vstsFeed: # Required when feedsToUse == Internal
 #vstsFeedPackage: # Required when feedsToUse == Internal
 #vstsPackageVersion: # Required when feedsToUse == Internal
 #feedDownloadExternal: # Required when feedsToUse == External
 #packageDownloadExternal: # Required when feedsToUse == External
 #versionDownloadExternal: # Required when feedsToUse == External
 #publishDirectory: '$(Build.ArtifactStagingDirectory)' # Required when command == Publish
 #feedsToUsePublish: 'internal' # Options: internal, external
 #publishFeedCredentials: # Required when feedsToUsePublish == External
 #vstsFeedPublish: # Required when feedsToUsePublish == Internal
 #publishPackageMetadata: true # Optional
 #vstsFeedPackagePublish: # Required when feedsToUsePublish == Internal
 #feedPublishExternal: # Required when feedsToUsePublish == External
 #packagePublishExternal: # Required when feedsToUsePublish == External
 #versionOption: 'patch' # Options: major, minor, patch, custom
 #versionPublish: # Required when versionOption == Custom
 packagePublishDescription:
 #verbosity: 'None' # Options: none, trace, debug, information, warning, error, critical
 #publishedPackageVar: # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<code>command</code> Command	The NuGet command to run. Options: <code>download</code> , <code>publish</code>
<code>downloadDirectory</code> Destination directory	Folder path where the package's contents download.

ARGUMENT	DESCRIPTION
<code>feedsToUse</code> Feed location	You can select a feed from either this collection or any other collection in Azure Artifacts. Options: <code>internal</code> , <code>external</code>
<code>externalFeedCredentials</code> Credentials for feeds outside this organization (collection)	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.
<code>vstsFeed</code> Use packages from this Azure Artifacts/TFS feed	Include the selected feed. You must have Azure Artifacts installed and licensed to select a feed here.
<code>vstsFeedPackage</code> Package name	Name of package to download.
<code>vstsPackageVersion</code> Package version	Select the package version or use a variable containing the version to download. This entry can also be a wildcard expression such as <code>*</code> to get the highest version, <code>1.*</code> to get the highest version with major version 1, or <code>1.2.*</code> to get the highest patch release with major version 1 and minor version 2.
<code>feedDownloadExternal</code> Feed	Specifies the name of an external feed from which to download.
<code>packageDownloadExternal</code> Package name	Specifies the package name to download.
<code>versionDownloadExternal</code> Package version	Select the package version or use a variable containing the version to download. This entry can also be a wildcard expression, such as <code>*</code> , to get the highest version, <code>1.*</code> to get the highest version with major version 1, or <code>1.2.*</code> to get the highest patch release with major version 1 and minor version 2.
<code>publishDirectory</code> Path to files to publish	Specifies the path to list of files to be published.
<code>feedsToUsePublish</code> Feed location	You can select a feed from either this collection or any other collection in Azure Artifacts. Options: <code>internal</code> , <code>external</code>
<code>publishFeedCredentials</code> organization/collection connection	Credentials to use for external feeds.
<code>vstsFeedPublish</code> Destination Feed	Specifies the project and feed's name/GUID to publish to.
<code>publishPackageMetadata</code> Publish pipeline metadata	Associate this build and release pipeline's metadata (run #, source code information) with the package.

ARGUMENT	DESCRIPTION
<code>vstsFeedPackagePublish</code> Package name	Select a package ID to publish or type a new package ID, if you've never published a version of this package before. Package names must be lower case and can only use letters, numbers, and dashes(-).
<code>feedPublishExternal</code> Feed	External feed name to publish to.
<code>packagePublishExternal</code> Package name	Package name.
<code>versionOption</code> Version	Select a version increment strategy, or select Custom to input your package version manually. For new packages, the first version is 1.0.0 if you select "Next major". The first version is 0.1.0 if you select "Next minor". The first version is 0.0.1 if you select "Next patch". For more information, see the <a href="#">Semantic Versioning spec</a> . Options: <code>major</code> , <code>minor</code> , <code>patch</code> , <code>custom</code>
<code>versionPublish</code> Custom version	Select the custom package version.
<code>packagePublishDescription</code> Description	Description of the contents of this package and the changes made in this version of the package.
<code>verbosity</code> Verbosity	Specifies the amount of detail displayed in the output. Options: <code>None</code> , <code>Trace</code> , <code>Debug</code> , <code>Information</code> , <code>Warning</code> , <code>Error</code> , <code>Critical</code>
<code>publishedPackageVar</code> Package Output Variable	Provide a name for the variable that contains the published package name and version.
<a href="#">Control options</a>	

## Example

The simplest way to get started with the Universal Package task is to use the Pipelines task editor to generate the YAML. You can then copy the generated code into your project's `azure-pipelines.yml` file. In this example, the sample demonstrates how to quickly generate the YAML using a pipeline that builds a GatsbyJS progressive web app (PWA).

Universal Packages are a useful way to both encapsulate and version a web app. Packaging a web app into a Universal Package enables quick rollbacks to a specific version of your site and eliminates the need to build the site in the deployment pipeline.

This example pipeline demonstrates how to fetch a tool from a feed within your project. The Universal Package task is used to download the tool, run a build, and again uses the Universal Package task to publish the entire compiled GatsbyJS PWA to a feed as a versioned Universal Package.

The screenshot shows the Azure DevOps Pipeline editor. On the left, there's a sidebar with 'Pipeline' and 'Build pipeline' options. Below it, under 'Agent job 1', is a list of tasks: 'Universal download', 'Install dependencies from package.json', 'Run gatsby build', and 'Universal publish'. A blue box highlights the 'Use NodeJS 12.x' task, which has a 'PREVIEW' badge and a note 'Use Node.js ecosystem'. To the right of the pipeline editor is a detailed view of the selected task:

- Task version:** 1.\* (preview)
- Display name:** Use NodeJS 12.x
- Version:** 12.x
- Control Options:** Includes a checkbox for 'Check for Latest Version'.
- Output Variables:** A dropdown menu.

### Download a package with the Universal Package task

The second task in the sample project uses the Universal Package task to fetch a tool, imagemagick, from a feed that is within a different project in the same organization. The tool, imagemagick, is required by the subsequent build step to resize images.

1. Add the Universal Package task by clicking the plus icon, typing "universal" in the search box, and clicking the "Add" button to add the task to your pipeline.

The screenshot shows the Azure DevOps Pipeline editor. On the left, the pipeline structure is visible. A red box highlights the plus icon next to the 'Agent job 1' header. On the right, a modal window titled 'Add tasks' is open, showing a search bar with 'universal' typed into it. A red box highlights the search bar. Below the search bar is a list of tasks, with one item highlighted by a red box:

- Universal packages**: Description: Download or publish Universal Packages. By Microsoft Corporation. An 'Add' button is shown.

2. Click the newly added **Universal Package** task and the **Command** to [Download](#).
3. Choose the **Destination directory** to use for the tool download.
4. Select a source **Feed** that contains the tool, set the **Package name**, and choose **Version** of the imagemagick tool from the source *Feed*\*

Pipeline Build pipeline

Get sources

Agent job 1 Run on agent

Use NodeJS 12.x PREVIEW Use Node.js ecosystem

Universal download Universal packages

Install dependencies from package.json npm

Run gatsby build npm

Universal download Some settings need attention

Universal packages

Task version 0.\*

Display name \* Universal download

Command \* Download

Destination directory \* Application

Feed & package details

Feed location \* This organization/collection Another organization/collection

Feed \* pwa-test-feed

Package name \* imagemagick

Version \* 1.0.0

5. After completing the fields, click View YAML to see the generated YAML.

Tasks Variables Triggers Options Retention History Save & queue Discard Summary Queue ...

Get sources

Agent job 1 Run on agent

Use NodeJS 12.x PREVIEW Use Node.js ecosystem

Universal download Universal packages

Install dependencies from package.json npm

Run gatsby build npm

Universal packages

Link settings View YAML

Copy to clipboard

Below is a clipboard-friendly view of your selection. To copy to the clipboard, either right-click and choose 'Copy' from the browser's context menu or press Ctrl+C. [more information about YAML builds]

```
steps:
- task: UniversalPackages@0
 displayName: 'Universal download'
 inputs:
 downloadDirectory: Application
 vstsFeed: '00000000-0000-0000-000000000000/00000000-0000-0000-000000000001'
 vstsFeedPackage: imagemagick
 vstsPackageVersion: 1.0.0
```

Another organization/collection

Copy to clipboard

6. The Universal Package task builder generates simplified YAML that contains non-default values. Copy the generated YAML into your `azure-pipelines.yml` file at the root of your project's git repo as defined [here](#).

```

Download Universal Package
steps:
- task: UniversalPackages@0
 displayName: 'Universal download'
 inputs:
 downloadDirectory: Application
 vstsFeed: '00000000-0000-0000-0000-000000000000/00000000-0000-0000-0000-000000000001'
 vstsFeedPackage: imagemagick
 vstsPackageVersion: 1.0.0

```

## Publish a package with the Universal Package task

The last step in this sample pipeline uses the Universal Package task to upload the production-ready Gatsby PWA that was produced by the `Run gatsby build` step to a feed as a versioned Universal Package. Once in a feed, you have a permanent copy of your complete site that can be deployed to hosting provider and started with `gatsby serve`.

1. Add another Universal Package task to the end of the pipeline by clicking the plus icon, typing "universal" in the search box, and clicking the "Add" button to add the task to your pipeline. This task gathers all of the production-ready assets produced by the `Run gatsby build` step, produce a versioned Universal Package, and publish the package to a feed.

… > azure-devops-pwa - CI

The screenshot shows the Azure DevOps Pipeline interface. On the left, the pipeline definition is visible with steps: Get sources, Agent job 1 (containing Use NodeJS 12.x, Universal download, Install dependencies from package.json, Run gatsby build, and another Universal download step), and a final step. On the right, the 'Add tasks' search results are shown for 'universal'. A card for 'Universal packages' by Microsoft Corporation is highlighted, featuring an 'Add' button. Other cards in the Marketplace section include 'Restore artifacts with Universal Packages' and 'Get Latest Universal Package Version'.

2. Set the **Command** to `Publish`.
3. Set **Path to file(s) to publish** to the directory containing your GatsbyJS project's `package.json`.
4. Choose a destination feed, a package name, and set your versioning strategy.

The screenshot shows the Azure DevOps Pipeline builder interface. On the left, a list of pipeline steps is visible, including 'Get sources', 'Agent job 1' (containing 'Use NodeJS 12.x', 'Universal download', 'Install dependencies from package.json', and 'Run gatsby build'), and 'Universal publish'. The 'Universal publish' step is currently selected. On the right, detailed configuration options for this step are shown. The 'Command' dropdown is set to 'Publish'. The 'Path to file(s) to publish' field contains '\$(Build.ArtifactStagingDirectory)'. Under 'Feed & package details', the 'Destination Feed' is set to 'azure-devops-pwa', the 'Package name' is 'mygatsbysite', and the 'Version' is set to 'Next patch'. A note at the bottom indicates a test package.

5. After completing the required fields, click **View YAML**.

6. Copy the resulting YAML into your `azure-pipelines.yml` file as before. The YAML for this sample project displays below.

```
Publish Universal Package
steps:
- task: UniversalPackages@0
 displayName: 'Universal publish'
 inputs:
 command: publish
 publishDirectory: Application
 vstsFeedPublish: '00000000-0000-0000-0000-000000000000/00000000-0000-0000-0000-000000000002'
 vstsFeedPackagePublish: mygatsbysite
 packagePublishDescription: 'A test package'
```

This example demonstrated how to use the Pipelines task builder to quickly generate the YAML for the Universal Package task, which can then be placed into your `azure-pipelines.yml` file. The Universal Package task builder supports all of the advanced configurations that can be created with **Universal Package** task's arguments.

#### NOTE

All feeds created through the classic user interface are project-scoped feeds. For the `vstsFeedPublish` parameter, you can also use the project and feed's names instead of their GUIDs like the following: '`< projectName>/< feedName>`'. See [Publish your Universal packages](#) for more details.

These tasks are open source [on GitHub](#). Feedback and contributions are welcome.

### **My Pipeline needs to access a feed in a different project**

If the pipeline is running in a different project than the project hosting the feed, you must set up the other project to grant read/write access to the build service. See [Package permissions in Azure Pipelines](#) for more details.

# NuGet Installer task version 0.\*

4/10/2020 • 2 minutes to read • [Edit Online](#)

Azure Pipelines (deprecated) | TFS 2017 (deprecated in 2017 Update 2)

Use this task 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 <b>Solution</b> argument in your <a href="#">Visual Studio Build task</a> and paste it here.
Path to NuGet.config	If you are using a package source other than NuGet.org, you must check in a <a href="#">NuGet.config</a> file and specify the path to it here.
Disable local cache	Equivalent to <a href="#">nuget restore</a> with the <code>-NoCache</code> option.
NuGet Arguments	Additional arguments passed to <a href="#">nuget restore</a> .
<b>ADVANCED</b>	
Path to NuGet.exe	(Optional) Path to your own instance of NuGet.exe. If you specify this argument, you must have your <a href="#">own strategy to handle authentication</a> .
<b>CONTROL OPTIONS</b>	

## 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

5/12/2021 • 11 minutes to read • [Edit Online](#)

Version 2.

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018

## NOTE

The [NuGet Authenticate](#) task is the new recommended way to authenticate with Azure Artifacts and other NuGet repositories. This task no longer takes new features and only critical bugs are addressed.

Use this task to install and update NuGet package dependencies, or package and publish NuGet packages. Uses NuGet.exe and works with .NET Framework apps. For .NET Core and .NET Standard apps, use the .NET Core task.

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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.

If you are working with .NET Core or .NET Standard, use the [.NET Core](#) task, which has full support for all package scenarios and it's currently supported by dotnet.

## 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](#).

## YAML snippet

```

NuGet
Restore, pack, or push NuGet packages, or run a NuGet command. Supports NuGet.org and authenticated feeds
like Azure Artifacts 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
 #publishPackageMetadata: true # Optional
 #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, specify path to nuspec files
 #verbosityPack: 'Detailed' # Options: quiet, normal, detailed
 #arguments: # Required when command == Custom

```

## Arguments

ARGUMENT	DESCRIPTION
<code>command</code> Command	The NuGet command to run. Select 'Custom' to add arguments or to use a different command. Options: <code>restore</code> , <code>pack</code> , <code>custom</code> , <code>push</code>
<code>restoreSolution</code> Path to solution, packages.config, or project.json	The path to the solution, packages.config, or project.json file that references the packages to be restored.
<code>feedsToUse</code> Feeds to use	You can either select a feed from Azure Artifacts and/or NuGet.org, or commit a nuget.config file to your source code repository and set its path here. Options: <code>select</code> , <code>config</code>

ARGUMENT	DESCRIPTION
<code>vstsFeed</code> Use packages from this Azure Artifacts/TFS feed	Include the selected feed in the generated NuGet.config. You must have Azure Artifacts installed and licensed to select a feed here.
<code>includeNuGetOrg</code> Use packages from NuGet.org	Include NuGet.org in the generated NuGet.config. Default value is <code>true</code> . Required when <code>feedsToUse</code> == <code>Select</code> .
<code>nugetConfigPath</code> Path to NuGet.config	The NuGet.config in your repository that specifies the feeds from which to restore packages. Required when <code>feedsToUse</code> == <code>Config</code>
<code>externalFeedCredentials</code> Credentials for feeds outside this organization/collection	Credentials to use for external registries located in the selected NuGet.config. This is the name of your NuGet service connection. For feeds in this organization/collection, leave this blank; the build's credentials are used automatically.
<code>noCache</code> Disable local cache	Prevents NuGet from using packages from local machine caches.
<code>disableParallelProcessing</code> Disable parallel processing	Prevents NuGet from installing multiple packages in parallel.
<code>restoreDirectory</code> 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.
<code>verbosityRestore</code> Verbosity	Specifies the amount of detail displayed in the output. Options: <code>Quiet</code> , <code>Normal</code> , <code>Detailed</code>
<code>nuGetFeedType</code> Target feed location	Specifies whether the target feed is an internal feed/collection or an external NuGet server. Options: <code>internal</code> , <code>external</code> . Required when <code>command</code> == <code>Push</code>
<code>publishVstsFeed</code> Target feed	Select a feed hosted in this account. You must have Azure Artifacts installed and licensed to select a feed here.
<code>packagesToPush</code> Packages location	The pattern to match or path to the nupkg files, e.g.: <code>'\$(Build.ArtifactStagingDirectory)/*.nupkg'</code> . Required when <code>command</code> == <code>Push</code>
<code>publishPackageMetadata</code> Publish pipeline metadata	If you continually publish a set of packages and only change the version number of the subset of packages that changed, use this option.
<code>allowPackageConflicts</code>	It allows the task to report success even if some of your packages are rejected with 409 Conflict errors. If NuGet.exe encounters a conflict, the task will fail. This option will not work and publish will fail if you are within a proxy environment.

ARGUMENT	DESCRIPTION
<code>publishFeedCredentials</code> NuGet server	The NuGet service connection that contains the external NuGet server's credentials.
<code>verbosityPush</code> Verbosity	Specifies the amount of detail displayed in the output. Options: <code>Quiet</code> , <code>Normal</code> , <code>Detailed</code>
<code>packagesToPack</code> Path to csproj or nuspec file(s) to pack	Pattern to search for csproj directories to pack. You can separate multiple patterns with a semicolon, and you can make a pattern negative by prefixing it with '!'. Example: <code>**\*.csproj; !**\*.Tests.csproj</code>
<code>configuration</code> Configuration to package	When using a csproj file this specifies the configuration to package.
<code>packDestination</code> Package folder	Folder where packages will be created. If empty, packages will be created at the source root.
<code>versioningScheme</code> Automatic package versioning	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. <b>Note:</b> Under Options set the build number format to be <code>'\$(BuildDefinitionName)_\$(Year:yyyy).\$(Month).\$(DayOfMonth:h)\$(Rev:r)'</code> . Options: <code>off</code> , <code>byPrereleaseNumber</code> , <code>byEnvVar</code> , <code>byBuildNumber</code>
<code>includeReferencedProjects</code> Environment variable	Enter the variable name without \$, \$env, or %.
<code>majorVersion</code> Major	The 'X' in version <code>X.Y.Z</code>
<code>minorVersion</code> Minor	The 'Y' in version <code>X.Y.Z</code>
<code>patchVersion</code> Patch	The 'Z' in version <code>X.Y.Z</code>
<code>packTimezone</code> Time zone	Specifies the desired time zone used to produce the version of the package. Selecting UTC is recommended if you're using hosted build agents as their date and time might differ. Options: <code>utc</code> , <code>local</code>
<code>includeSymbols</code> Create symbols package	Specifies that the package contains sources and symbols. When used with a .nuspec file, this creates a regular NuGet package file and the corresponding symbols package.

ARGUMENT	DESCRIPTION
<code>toolPackage</code> Tool Package	Determines if the output files of the project should be in the tool folder.
<code>buildProperties</code> Additional build properties	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.
<code>basePath</code> Base path	The base path of the files defined in the nuspec file.
<code>verbosityPack</code> Verbosity	Specifies the amount of detail displayed in the output. Options: <code>Quiet</code> , <code>Normal</code> , <code>Detailed</code>
<code>arguments</code> Command and arguments	The command and arguments which will be passed to NuGet.exe for execution. If NuGet 3.5 or later is used, authenticated commands like list, restore, and publish against any feed in this organization/collection that the Project Collection Build Service has access to will be automatically authenticated.
Control options	

## 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 to the value of the environment variable that has the name specified by the `versionEnvVar` parameter, e.g. `MyVersion` (no \$, just the environment variable name). 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 using the pipeline run's build number. This is the value specified for the pipeline's `name` property, which gets saved to the `BUILD_BUILDNUMBER` environment variable). Ensure that the build number being used contains a proper SemVer e.g. `1.0.$(Rev:r)`. When using `byBuildNumber`, the task will extract the dotted version, `1.2.3.4` from the build number string, and use only that portion. The rest of the string will be dropped. If you want to use the build number as is, you can use `byEnvVar` as described above, and set `versionEnvVar` to `BUILD_BUILDNUMBER`.

## Examples

### Restore

Restore all your solutions with packages from a selected feed.

```
Restore from a project scoped feed in the same organization
- task: NuGetCommand@2
 inputs:
 command: 'restore'
 feedsToUse: 'select'
 vstsFeed: 'my-project/my-project-scoped-feed'
 includeNuGetOrg: false
 restoreSolution: '**/*.sln'
```

```
Restore from an organization scoped feed in the same organization
- task: NuGetCommand@2
 inputs:
 command: 'restore'
 feedsToUse: 'select'
 vstsFeed: 'my-organization-scoped-feed'
 restoreSolution: '**/*.sln'
```

```
Restore from a feed in a different organization
- task: NuGetCommand@2
 inputs:
 command: 'restore'
 feedsToUse: config
 nugetConfigPath: ./nuget.config
 restoreSolution: '**/*.sln'
 externalFeedCredentials: 'MyServiceConnectionName'
 noCache: true
 continueOnError: true
```

```
Restore from feed(s) set in nuget.config
- task: NuGetCommand@2
 inputs:
 command: 'restore'
 feedsToUse: 'config'
 nugetConfigPath: 'nuget.config'
```

## Package

Create a NuGet package in the destination folder.

```
Package a project
- task: NuGetCommand@2
 inputs:
 command: 'pack'
 packagesToPack: '**/*.csproj'
 packDestination: '$(Build.ArtifactStagingDirectory)'
```

## Push

### NOTE

Pipeline artifacts are downloaded to `System.ArtifactsDirectory` directory. `packagesToPush` value can be set to `$(System.ArtifactsDirectory)/**/*.nupkg` in your release pipeline.

- Push/Publish a package to a feed defined in your NuGet.config.

```
Push a project
- task: NuGetCommand@2
 inputs:
 command: 'push'
 packagesToPush: '$(Build.ArtifactStagingDirectory)/**/*.nupkg'
 feedsToUse: 'config'
 nugetConfigPath: '$(Build.WorkingDirectory)/NuGet.config'
```

- Push/Publish a package to a project scoped

```
Push a project
- task: NuGetCommand@2
 inputs:
 command: 'push'
 feedsToUse: 'select'
 vstsFeed: 'my-project/my-project-scoped-feed'
 publishVstsFeed: 'myTestFeed'
```

- Push/Publish a package to NuGet.org

```
Push a project
- task: NuGetCommand@2
 inputs:
 command: 'push'
 feedsToUse: 'config'
 includeNugetOrg: 'true'
```

## Custom

Run any other NuGet command besides the default ones: pack, push, and restore.

```
list local NuGet resources.
- task: NuGetCommand@2
 displayName: 'list locals'
 inputs:
 command: custom
 arguments: 'locals all -list'
```

## Open-source

Check out the Azure Pipelines and Team Foundation Server out-of-the-box tasks [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### 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?

[Azure Artifacts Documentation](#)

### 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 and deploy your app examples](#)

### What other kinds of build tasks are available?

## Build and release tasks catalog

### 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?

- [Build and release 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.
- [Supported source repositories](#) 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.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE.EXTRA.CA.CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

### 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](#).

### My Pipeline needs to access a feed in a different project

If the pipeline is running in a different project than the project hosting the feed, you must set up the other project to grant read/write access to the build service. See [Package permissions in Azure Pipelines](#) for more details.

# NuGet Packager task version 0.\*

4/10/2020 • 4 minutes to read • [Edit Online](#)

Azure Pipelines (deprecated) | TFS 2017 Update 2 and below (deprecated in TFS 2018)

Use this task to create a NuGet package from either a .csproj or .nuspec file.

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## 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 organization/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"> <li>• The packager compiles the .csproj files for packaging.</li> <li>• You must specify <b>Configuration to Package</b> (see below).</li> <li>• 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>.</li> </ul> <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"> <li>• The packager does not compile the .csproj files for packaging.</li> <li>• Each project is packaged only if it has a .nuspec file checked in.</li> <li>• The packager does not replace tokens in the .nuspec file (except the <code>&lt;version/&gt;</code> element, see <b>Use build number to version package</b>, below). You must supply values for elements such as <code>&lt;id/&gt;</code> and <code>&lt;description/&gt;</code>. The most common way to do this is to hardcode the values in the .nuspec file.</li> </ul> <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 <a href="#">pipeline options</a>, set the <b>build number format</b> to something like <code>\$(BuildDefinitionName)_\$(Year:yyyy).\$(Month).\$(DayOfMonth)\$(Rev:.:t)</code>. 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 <a href="#">nuget pack</a> 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 <a href="#">variable</a> such as <code>\$(Build.StagingDirectory)\packages</code>. If you leave it empty, the package will be created in the same directory that contains the .csproj or .nuspec file.</p>

ARGUMENT	DESCRIPTION
<b>ADVANCED</b>	
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>&lt;description&gt;\$description\$&lt;/description&gt;</code> in the .nuspec file this way: <code>Description="This is a great package"</code> Using this argument is equivalent to supplying properties from <a href="#">nuget pack</a> with the <code>-Properties</code> option.
NuGet Arguments	(Optional) Additional arguments passed <a href="#">nuget pack</a> .
Path to NuGet.exe	(Optional) Path to your own instance of NuGet.exe. If you specify this argument, you must have your <a href="#">own strategy to handle authentication</a> .

#### 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:.r)</code>

## Publish to Azure Artifacts

Make sure you've prepared the build as described [above](#).

### Create the feed

See [Create a feed](#).

### Build tasks

<input type="text"/>	Build your solution.  <ul style="list-style-type: none"> <li>Solution: <code>*.sln</code></li> <li>Platform: <code>\$(BuildPlatform)</code></li> <li>Configuration: <code>\$(BuildConfiguration)</code></li> </ul>
<input type="text"/>	Package your projects.  <ul style="list-style-type: none"> <li>Path/Pattern to nuspec files: <code>*.csproj</code></li> <li>Use Build number to version package: Selected</li> <li>Advanced, Configuration to Package: <code>Release</code></li> </ul>
<input type="text"/>	Publish your packages to Azure Artifacts.  <ul style="list-style-type: none"> <li>Path/Pattern to nupkg: <code>***.nupkg</code></li> <li>Feed type: Internal NuGet Feed</li> <li>Internal feed URL: See <a href="#">Find your NuGet package source URL</a>.</li> </ul>

## 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

<input type="text"/>	Build your solution.  <ul style="list-style-type: none"> <li>Solution: <code>*.sln</code></li> <li>Platform: <code>\$(BuildPlatform)</code></li> <li>Configuration: <code>\$(BuildConfiguration)</code></li> </ul>
<input type="text"/>	Package your projects.  <ul style="list-style-type: none"> <li>Path/Pattern to nuspec files: <code>*.csproj</code></li> <li>Use Build number to version package: Selected</li> <li>Advanced, Configuration to Package: <code>Release</code></li> </ul>



#### Package: NuGet Publisher

Publish your packages to NuGet.org.

- Path/Pattern to nupkg: `***.nupkg`
- Feed type: External NuGet Feed
- NuGet Server Endpoint:
  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/TOKEN Key: Paste API Key from your [NuGet account](#).

# Package: NuGet Publisher task version 0.\*

4/10/2020 • 2 minutes to read • [Edit Online](#)

Azure Pipelines (deprecated) | TFS 2017 Update 2 and below (deprecated in TFS 2018)

Use this task 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"><li>• Default value: <code>*.nupkg; -:\packages**.nupkg</code></li><li>• To publish a single package, click the ... button and select the file.</li><li>• Use single-folder wildcards (… ) and recursive wildcards (□) to publish multiple packages.</li><li>• Use <a href="#">variables</a> to specify directories. For example, if you specified <code>\$(Build.StagingDirectory)\packages</code> as the <b>package folder</b> in the NuGet Packager task, you could specify <code>\$(Build.StagingDirectory)\packages\*.nupkg</code> here.</li></ul>
Feed type	<ul style="list-style-type: none"><li>• <b>External NuGetFeed</b> publishes to an external server such as <a href="#">NuGet</a> or <a href="#">MyGet</a>. After you select this option, you create and select a <b>NuGet server endpoint</b>.</li><li>• <b>Internal NuGet Feed</b> publishes to an internal or Azure Artifacts feed. After you select this option, you specify the <b>internal feed URL</b>.</li></ul>

ARGUMENT	DESCRIPTION
<b>ADVANCED</b>	
NuGet Arguments	(Optional) Additional arguments passed to <a href="#">nuget push</a> .
Path to NuGet.exe	(Optional) Path to your own instance of NuGet.exe. If you specify this argument, you must have your <a href="#">own strategy to handle authentication</a> .
<b>CONTROL OPTIONS</b>	

## 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><b>Build: Visual Studio Build</b></p> <p>Build your solution.</p> <ul style="list-style-type: none"> <li>• Solution: <code>*.sln</code></li> <li>• Platform: <code>\$(BuildPlatform)</code></li> <li>• Configuration: <code>\$(BuildConfiguration)</code></li> </ul>
	<p><b>Package: NuGet Packager</b></p> <p>Package your projects.</p> <ul style="list-style-type: none"> <li>• Path/Pattern to nuspec files: <code>*.csproj</code></li> <li>• Use Build number to version package: Selected</li> <li>• Advanced, Configuration to Package: <code>Release</code></li> </ul>
	<p><b>Package: NuGet Publisher</b></p> <p>Publish your packages to Azure Artifacts.</p> <ul style="list-style-type: none"> <li>• Path/Pattern to nupkg: <code>***.nupkg</code></li> <li>• Feed type: Internal NuGet Feed</li> <li>• Internal feed URL: See <a href="#">Find your NuGet package source URL</a>.</li> </ul>

## 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><b>Build: Visual Studio Build</b></p> <p>Build your solution.</p> <ul style="list-style-type: none"> <li>• Solution: <code>*.sln</code></li> <li>• Platform: <code>\$(BuildPlatform)</code></li> <li>• Configuration: <code>\$(BuildConfiguration)</code></li> </ul>
	<p><b>Package: NuGet Packager</b></p> <p>Package your projects.</p> <ul style="list-style-type: none"> <li>• Path/Pattern to nuspec files: <code>*.csproj</code></li> <li>• Use Build number to version package: Selected</li> <li>• Advanced, Configuration to Package: <code>Release</code></li> </ul>
	<p><b>Package: NuGet Publisher</b></p> <p>Publish your packages to NuGet.org.</p> <ul style="list-style-type: none"> <li>• Path/Pattern to nupkg: <code>***.nupkg</code></li> <li>• Feed type: External NuGet Feed</li> <li>• NuGet Server Endpoint: <ol style="list-style-type: none"> <li>1. Click "New service connection", and then click Generic.</li> <li>2. On the Add New Generic Connection dialog box: <ul style="list-style-type: none"> <li>◦ Connection Name: <code>NuGet</code></li> <li>◦ Server URL: <a href="https://nuget.org/">https://nuget.org/</a></li> <li>◦ User name: <code>{your-name}</code></li> <li>◦ Password/Token Key: Paste API Key from your <a href="#">NuGet account</a>.</li> </ul> </li> </ol> </li> </ul>

# NuGet task

5/12/2021 • 11 minutes to read • [Edit Online](#)

Version 2.

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018

## NOTE

The [NuGet Authenticate](#) task is the new recommended way to authenticate with Azure Artifacts and other NuGet repositories. This task no longer takes new features and only critical bugs are addressed.

Use this task to install and update NuGet package dependencies, or package and publish NuGet packages. Uses NuGet.exe and works with .NET Framework apps. For .NET Core and .NET Standard apps, use the .NET Core task.

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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.

If you are working with .NET Core or .NET Standard, use the [.NET Core](#) task, which has full support for all package scenarios and it's currently supported by dotnet.

## 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](#).

## YAML snippet

```

NuGet
Restore, pack, or push NuGet packages, or run a NuGet command. Supports NuGet.org and authenticated feeds
like Azure Artifacts 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
 #publishPackageMetadata: true # Optional
 #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, specify path to nuspec files
 #verbosityPack: 'Detailed' # Options: quiet, normal, detailed
 #arguments: # Required when command == Custom

```

## Arguments

ARGUMENT	DESCRIPTION
<code>command</code> Command	The NuGet command to run. Select 'Custom' to add arguments or to use a different command. Options: <code>restore</code> , <code>pack</code> , <code>custom</code> , <code>push</code>
<code>restoreSolution</code> Path to solution, packages.config, or project.json	The path to the solution, packages.config, or project.json file that references the packages to be restored.
<code>feedsToUse</code> Feeds to use	You can either select a feed from Azure Artifacts and/or NuGet.org, or commit a nuget.config file to your source code repository and set its path here. Options: <code>select</code> , <code>config</code>

ARGUMENT	DESCRIPTION
<code>vstsFeed</code> Use packages from this Azure Artifacts/TFS feed	Include the selected feed in the generated NuGet.config. You must have Azure Artifacts installed and licensed to select a feed here.
<code>includeNuGetOrg</code> Use packages from NuGet.org	Include NuGet.org in the generated NuGet.config. Default value is <code>true</code> . Required when <code>feedsToUse</code> == <code>Select</code> .
<code>nugetConfigPath</code> Path to NuGet.config	The NuGet.config in your repository that specifies the feeds from which to restore packages. Required when <code>feedsToUse</code> == <code>Config</code>
<code>externalFeedCredentials</code> Credentials for feeds outside this organization/collection	Credentials to use for external registries located in the selected NuGet.config. This is the name of your NuGet service connection. For feeds in this organization/collection, leave this blank; the build's credentials are used automatically.
<code>noCache</code> Disable local cache	Prevents NuGet from using packages from local machine caches.
<code>disableParallelProcessing</code> Disable parallel processing	Prevents NuGet from installing multiple packages in parallel.
<code>restoreDirectory</code> 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.
<code>verbosityRestore</code> Verbosity	Specifies the amount of detail displayed in the output. Options: <code>Quiet</code> , <code>Normal</code> , <code>Detailed</code>
<code>nuGetFeedType</code> Target feed location	Specifies whether the target feed is an internal feed/collection or an external NuGet server. Options: <code>internal</code> , <code>external</code> . Required when <code>command</code> == <code>Push</code>
<code>publishVstsFeed</code> Target feed	Select a feed hosted in this account. You must have Azure Artifacts installed and licensed to select a feed here.
<code>packagesToPush</code> Packages location	The pattern to match or path to the nupkg files, e.g.: <code>'\$(Build.ArtifactStagingDirectory)/*.nupkg'</code> . Required when <code>command</code> == <code>Push</code>
<code>publishPackageMetadata</code> Publish pipeline metadata	If you continually publish a set of packages and only change the version number of the subset of packages that changed, use this option.
<code>allowPackageConflicts</code>	It allows the task to report success even if some of your packages are rejected with 409 Conflict errors. If NuGet.exe encounters a conflict, the task will fail. This option will not work and publish will fail if you are within a proxy environment.

ARGUMENT	DESCRIPTION
<code>publishFeedCredentials</code> NuGet server	The NuGet service connection that contains the external NuGet server's credentials.
<code>verbosityPush</code> Verbosity	Specifies the amount of detail displayed in the output. Options: <code>Quiet</code> , <code>Normal</code> , <code>Detailed</code>
<code>packagesToPack</code> Path to csproj or nuspec file(s) to pack	Pattern to search for csproj directories to pack. You can separate multiple patterns with a semicolon, and you can make a pattern negative by prefixing it with '!'. Example: <code>**\*.csproj;!**\*.Tests.csproj</code>
<code>configuration</code> Configuration to package	When using a csproj file this specifies the configuration to package.
<code>packDestination</code> Package folder	Folder where packages will be created. If empty, packages will be created at the source root.
<code>versioningScheme</code> Automatic package versioning	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. <b>Note:</b> Under Options set the build number format to be <code>'\$(BuildDefinitionName)_\$(Year:yyyy).\$(Month).\$(DayOfMonth:h)\$(Rev:r)'</code> . Options: <code>off</code> , <code>byPrereleaseNumber</code> , <code>byEnvVar</code> , <code>byBuildNumber</code>
<code>includeReferencedProjects</code> Environment variable	Enter the variable name without \$, \$env, or %.
<code>majorVersion</code> Major	The 'X' in version <code>X.Y.Z</code>
<code>minorVersion</code> Minor	The 'Y' in version <code>X.Y.Z</code>
<code>patchVersion</code> Patch	The 'Z' in version <code>X.Y.Z</code>
<code>packTimezone</code> Time zone	Specifies the desired time zone used to produce the version of the package. Selecting UTC is recommended if you're using hosted build agents as their date and time might differ. Options: <code>utc</code> , <code>local</code>
<code>includeSymbols</code> Create symbols package	Specifies that the package contains sources and symbols. When used with a .nuspec file, this creates a regular NuGet package file and the corresponding symbols package.

ARGUMENT	DESCRIPTION
<code>toolPackage</code> Tool Package	Determines if the output files of the project should be in the tool folder.
<code>buildProperties</code> Additional build properties	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.
<code>basePath</code> Base path	The base path of the files defined in the nuspec file.
<code>verbosityPack</code> Verbosity	Specifies the amount of detail displayed in the output. Options: <code>Quiet</code> , <code>Normal</code> , <code>Detailed</code>
<code>arguments</code> Command and arguments	The command and arguments which will be passed to NuGet.exe for execution. If NuGet 3.5 or later is used, authenticated commands like list, restore, and publish against any feed in this organization/collection that the Project Collection Build Service has access to will be automatically authenticated.
Control options	

## 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 to the value of the environment variable that has the name specified by the `versionEnvVar` parameter, e.g. `MyVersion` (no \$, just the environment variable name). 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 using the pipeline run's build number. This is the value specified for the pipeline's `name` property, which gets saved to the `BUILD_BUILDNUMBER` environment variable). Ensure that the build number being used contains a proper SemVer e.g. `1.0.$(Rev:r)`. When using `byBuildNumber`, the task will extract the dotted version, `1.2.3.4` from the build number string, and use only that portion. The rest of the string will be dropped. If you want to use the build number as is, you can use `byEnvVar` as described above, and set `versionEnvVar` to `BUILD_BUILDNUMBER`.

## Examples

### Restore

Restore all your solutions with packages from a selected feed.

```
Restore from a project scoped feed in the same organization
- task: NuGetCommand@2
 inputs:
 command: 'restore'
 feedsToUse: 'select'
 vstsFeed: 'my-project/my-project-scoped-feed'
 includeNuGetOrg: false
 restoreSolution: '**/*.sln'
```

```
Restore from an organization scoped feed in the same organization
- task: NuGetCommand@2
 inputs:
 command: 'restore'
 feedsToUse: 'select'
 vstsFeed: 'my-organization-scoped-feed'
 restoreSolution: '**/*.sln'
```

```
Restore from a feed in a different organization
- task: NuGetCommand@2
 inputs:
 command: 'restore'
 feedsToUse: config
 nugetConfigPath: ./nuget.config
 restoreSolution: '**/*.sln'
 externalFeedCredentials: 'MyServiceConnectionName'
 noCache: true
 continueOnError: true
```

```
Restore from feed(s) set in nuget.config
- task: NuGetCommand@2
 inputs:
 command: 'restore'
 feedsToUse: 'config'
 nugetConfigPath: 'nuget.config'
```

## Package

Create a NuGet package in the destination folder.

```
Package a project
- task: NuGetCommand@2
 inputs:
 command: 'pack'
 packagesToPack: '**/*.csproj'
 packDestination: '$(Build.ArtifactStagingDirectory)'
```

## Push

### NOTE

Pipeline artifacts are downloaded to `System.ArtifactsDirectory` directory. `packagesToPush` value can be set to `$(System.ArtifactsDirectory)/**/*.nupkg` in your release pipeline.

- Push/Publish a package to a feed defined in your NuGet.config.

```
Push a project
- task: NuGetCommand@2
 inputs:
 command: 'push'
 packagesToPush: '$(Build.ArtifactStagingDirectory)/**/*.nupkg'
 feedsToUse: 'config'
 nugetConfigPath: '$(Build.WorkingDirectory)/NuGet.config'
```

- Push/Publish a package to a project scoped

```
Push a project
- task: NuGetCommand@2
 inputs:
 command: 'push'
 feedsToUse: 'select'
 vstsFeed: 'my-project/my-project-scoped-feed'
 publishVstsFeed: 'myTestFeed'
```

- Push/Publish a package to NuGet.org

```
Push a project
- task: NuGetCommand@2
 inputs:
 command: 'push'
 feedsToUse: 'config'
 includeNugetOrg: 'true'
```

## Custom

Run any other NuGet command besides the default ones: pack, push, and restore.

```
list local NuGet resources.
- task: NuGetCommand@2
 displayName: 'list locals'
 inputs:
 command: custom
 arguments: 'locals all -list'
```

## Open-source

Check out the Azure Pipelines and Team Foundation Server out-of-the-box tasks [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### 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?

[Azure Artifacts Documentation](#)

### 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 and deploy your app examples](#)

### What other kinds of build tasks are available?

## Build and release tasks catalog

### 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?

- [Build and release 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.
- [Supported source repositories](#) 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.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE.EXTRA.CA.CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

### 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](#).

### My Pipeline needs to access a feed in a different project

If the pipeline is running in a different project than the project hosting the feed, you must set up the other project to grant read/write access to the build service. See [Package permissions in Azure Pipelines](#) for more details.

# Package: Python Pip Authenticate version 0.\*

11/2/2020 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Provides authentication for the `pip` client that can be used to install Python distributions.

## YAML snippet

```
Python pip authenticate
Authentication task for the pip client used for installing Python distributions
- task: PipAuthenticate@0
 inputs:
 artifactFeeds:
 #externalFeeds: # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
artifactFeeds	List of Azure Artifacts feeds to authenticate with pip.
externalFeeds	List of service connections from external organizations to authenticate with pip.

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

# Package: Python Twine Upload Authenticate version 0.\*

11/2/2020 • 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.

## YAML snippet

```
Python twine upload authenticate
Authenticate for uploading Python distributions using twine. 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:
 #externalFeeds: # Optional
 #publishPackageMetadata: true # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
artifactFeeds	List of Azure Artifacts feeds to authenticate with <code>twine</code> .
externalFeeds	List of service connections 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.

## FAQ

# App Center Distribute task

4/22/2020 • 4 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

Use this task to distribute app builds to testers and users through App Center.

- [Sign up with App Center](#) first.
- For details about using this task, see the App Center documentation article [Deploy Azure DevOps Builds with App Center](#).

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## YAML snippet

```
App Center distribute
Distribute app builds to testers and users via Visual Studio App Center
- task: AppCenterDistribute@3
 inputs:
 serverEndpoint:
 appSlug:
 appFile:
 #symbolsOption: 'Apple' # Optional. Options: apple, android
 #symbolsPath: # Optional
 #symbolsPdbFiles: '**/*.pdb' # Optional
 #symbolsDsymFiles: # Optional
 #symbolsIncludeParentDirectory: # Optional
 #releaseNotesOption: 'input' # Options: input, file
 #releaseNotesInput: # Required when releaseNotesOption == Input
 #releaseNotesFile: # Required when releaseNotesOption == File
 #isMandatory: false # Optional
 #destinationType: 'groups' # Options: groups, store
 #distributionGroupId: # Optional
 #destinationStoreId: # Required when destinationType == store
 #isSilent: # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<code>serverEndpoint</code> App Center service connection	(Required) Select the service connection for App Center. Create a new App Center service connection in Azure DevOps project settings.

ARGUMENT	DESCRIPTION
<code>appSlug</code> App slug	(Required) The app slug is in the format of <code>{username}/{app_identifier}</code> . To locate <code>{username}</code> and <code>{app_identifier}</code> for an app, click on its name from <a href="https://appcenter.ms/apps">https://appcenter.ms/apps</a> , and the resulting URL is in the format of <a href="https://appcenter.ms/users/{username}/apps/{app_identifier}">https://appcenter.ms/users/{username}/apps/{app_identifier}</a> . If you are using orgs, the app slug is of the format <code>{orgname}/{app_identifier}</code> .
<code>app</code> Binary file path	(Required) Relative path from the repo root to the APK or IPA file you want to publish Argument alias: <code>appFile</code>
<code>buildVersion</code> Build version	(Optional) The build version of the uploading binary which needs to be specified for <code>.zip</code> and <code>.msi</code> . This value will be ignored unless the platform is WPF or WinForms.
<code>symbolsType</code> Symbols type	(Optional) Include symbol files to receive symbolicated stack traces in App Center Diagnostics. Options: <code>Android, Apple, UWP</code> . Argument alias: <code>symbolsOption</code>
<code>symbolsPath</code> Symbols path	(Optional) Relative path from the repo root to the symbols folder.
<code>appxsymPath</code> Symbols path (* appxsym)	(Optional) Relative path from the repo root to PDB symbols files. Path may contain wildcards.
<code>dsymPath</code> dSYM path	(Optional) Relative path from the repo root to dSYM folder. Path may contain wildcards. Argument alias: <code>symbolsDsymFiles</code>
<code>mappingTxtPath</code> Mapping file	(Optional) Relative path from the repo root to Android's mapping.txt file Argument alias: <code>symbolsMappingTxtFile</code>
<code>nativeLibrariesPath</code> Native Library File Path	(Optional) Relative path from the repo root to the additional native libraries you want to publish (e.g. .so files)
<code>packParentFolder</code> 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. Argument alias: <code>symbolsIncludeParentDirectory</code>
<code>releaseNotesSelection</code> Create release notes	(Required) Release notes will be attached to the release and shown to testers on the installation page. Options: <code>input, file</code> . Default value: <code>input</code> Argument alias: <code>releaseNotesOption</code>
<code>releaseNotesInput</code> Release notes	(Required) Release notes for this version.

ARGUMENT	DESCRIPTION
<code>releaseNotesFile</code> Release notes file	(Required) Select a UTF-8 encoded text file which contains the Release Notes for this version.
<code>isMandatory</code> Require users to update to this release	(Optional) App Center Distribute SDK required to mandate update. Testers will automatically be prompted to update. Default value: <code>false</code>
<code>destinationType</code> Release destination	(Required) Each release will be distributed to either groups or a store. Options: <code>groups, store</code> .
<code>destinationGroupIds</code> Destination IDs	(Optional) IDs of the distribution groups to release to. Leave it empty to use the default group and use commas or semicolons to separate multiple IDs. Argument alias: <code>distributionGroupId</code>
<code>destinationStoreId</code> Destination ID	(Required) ID of the distribution store to deploy to.
<code>isSilent</code> Do not notify testers. Release will still be available to install.	(Optional) Testers will not receive an email for new releases.

## Example

This example pipeline builds an Android app, runs tests, and publishes the app using App Center Distribute.

```

Android
Build your Android project with Gradle.
Add steps that test, sign, and distribute the APK, save build artifacts, and more:
https://docs.microsoft.com/azure/devops/pipelines/ecosystems/android

pool:
 vmImage: 'macOS-latest'
steps:

- script: sudo npm install -g appcenter-cli
- script: appcenter login --token {YOUR_TOKEN}

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

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

- task: PublishBuildArtifacts@1
 inputs:
 pathToPublish: '$(build.artifactStagingDirectory)'
 artifactName: 'outputs'
 artifactType: 'container'

Run tests using the App Center CLI
- script: appcenter test run espresso --app "{APP_CENTER_SLUG}" --devices "{DEVICE}" --app-path {APP_FILE} --
-test-series "master" --locale "en_US" --build-dir {PAT_ESPRESSO} --debug

Distribute the app
- task: AppCenterDistribute@3
 inputs:
 serverEndpoint: 'AppCenter'
 appSlug: '$(APP_CENTER_SLUG)'
 appFile: '{APP_FILE}' # Relative path from the repo root to the APK or IPA file you want to publish
 symbolsOption: 'Android'
 releaseNotesOption: 'input'
 releaseNotesInput: 'Here are the release notes for this version.'
 destinationType: 'groups'

```

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Azure App Service Deploy task

5/24/2021 • 18 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task 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 on both Windows and Linux](#)
- [Function Apps for Containers](#)
- [WebJobs](#)
- Apps configured under [Azure App Service Environments](#)

## Prerequisites for the task

The following prerequisites must be set up in the target machine(s) 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.

PARAMETERS	DESCRIPTION
<code>ConnectionType</code> (Connection type)	(Required) Select the service connection type to use to deploy the Web App. Default value: AzureRM
<code>Connected ServiceName</code> (Azure subscription)	(Required if ConnectionType = AzureRM) Select the Azure Resource Manager subscription for the deployment. Argument aliases: <code>azureSubscription</code>
<code>PublishProfilePath</code> (Publish profile path)	(Required if ConnectionType = PublishProfile) The path to the file containing the publishing information. Default value: <code>\$(System.DefaultWorkingDirectory)/**/*.pubxml</code>
<code>PublishProfilePassword</code> (Publish profile password)	(Required if ConnectionType = PublishProfile) The password for the profile file. Consider storing the password in a secret variable and using that variable here. Example: <code>\$(Password)</code>

PARAMETERS	DESCRIPTION
<input type="checkbox"/> <b>WebAppKind</b> (App Service type)	<p>(Required if ConnectionType = AzureRM) Choose from Web App On Windows, Web App On Linux, Web App for Containers, Function App, Function App on Linux, Function App for Containers and Mobile App</p> <p>Default value: webApp</p> <p>Argument aliases: <input type="checkbox"/> <code>appType</code></p>
<input type="checkbox"/> <b>WebAppName</b> (App Service name)	<p>(Required if ConnectionType = AzureRM) Enter or select the name of an existing Azure App Service. Only App Services based on the selected app type will be listed.</p>
<input type="checkbox"/> <b>DeployTo</b> <input type="checkbox"/> <b>SlotOrASEFlag</b> (Deploy to Slot or App Service Environment)	<p>(Optional) Select the option to deploy to an existing deployment slot or Azure App Service environment. For both the targets, the task requires a Resource Group name. If the deployment target is a slot, by default the deployment is to the <b>production</b> slot. Any other existing slot name can be provided.</p> <p>If the deployment target is an Azure App Service environment, leave the slot name as <b>production</b> and specify just the Resource Group name.</p> <p>Default value: false</p> <p>Argument aliases: <input type="checkbox"/> <code>deployToSlotOrASE</code></p>
<input type="checkbox"/> <b>ResourceGroupName</b> (Resource group)	<p>(Required if DeployToSlotOrASEFlag = true) The Resource Group name is required when the deployment target is either a deployment slot or an App Service environment. Enter or select the Azure Resource Group that contains the Azure App Service specified above.</p>
<input type="checkbox"/> <b>SlotName</b> (Slot)	<p>(Required, if DeployToSlotOrASEFlag = true) Enter or select an existing slot other than the <b>production</b> slot.</p> <p>Default value: production</p>
<input type="checkbox"/> <b>DockerNamespace</b> (Registry or Namespace)	<p>(Required if WebAppKind = webAppContainer or WebAppkind = functionAppContainer) A globally unique top-level domain name for your specific registry or namespace. <b>Note:</b> the fully-qualified image name will be of the format: {registry or namespace}/{repository}:{tag}. For example, <code>myregistry.azurecr.io/nginx:latest</code></p>
<input type="checkbox"/> <b>DockerRepository</b> (Image)	<p>(Required if WebAppKind = webAppContainer or WebAppkind = functionAppContainer) Name of the repository where the container images are stored. <b>Note:</b> the fully-qualified image name will be of the format: {registry or namespace}/{repository}:{tag}. For example, <code>myregistry.azurecr.io/nginx:latest</code></p>
<input type="checkbox"/> <b>DockerImageTag</b> (Tag)	<p>(Optional) Tags are optional, but are the mechanism that registries use to apply version information to Docker images. <b>Note:</b> the fully-qualified image name will be of the format: {registry or namespace}/{repository}:{tag}. For example, <code>myregistry.azurecr.io/nginx:latest</code></p>

PARAMETERS	DESCRIPTION
<b>VirtualApplication</b> (Virtual application)	(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 <a href="#">configured</a> before deployment of the web project.
<b>Package</b> (Package or folder)	(Required if ConnectionType = PublishProfile or WebAppKind = webApp, apiApp, functionApp, mobileApp, webAppLinux, or functionAppLinux) File path to the package, or to a folder containing App Service contents generated by MSBuild, or to a compressed zip or war file. <b>Build variables</b> or <a href="#">release variables</a> and wildcards are supported. For example, <code>\$(System.DefaultWorkingDirectory)/**/*.zip</code> or <code>\$(System.DefaultWorkingDirectory)/**/*.war</code> Default value: <code>\$(System.DefaultWorkingDirectory)/**/*.zip</code> Argument aliases: <code>packageForLinux</code>
<b>RuntimeStack</b> (Runtime Stack)	(Optional) Select the framework and version. This is for WebApp for Linux.
<b>RuntimeStackFunction</b> (Runtime Stack)	(Optional) Select the framework and version. This is for Function App on Linux.
<b>StartupCommand</b> (Startup command)	(Optional) Enter the start up command.
<b>ScriptType</b> (Deployment script type)	(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. <a href="#">Learn more</a> .
<b>InlineScript</b> (Inline Script)	(Required if ScriptType == Inline Script) The script to execute. You can provide your deployment commands here, one command per line. See <a href="#">this example</a> .
<b>ScriptPath</b> (Deployment script path)	(Required if ScriptType == File Path) The path and name of the script to execute.
<b>WebConfigParameters</b> (Generate web.config parameters for Python, Node.js, Go and Java apps)	(Optional) A standard web.config will be generated and deployed to Azure App Service if the application does not have one. The values in web.config can be edited and will vary based on the application framework. For example for Node.js applications, web.config will have startup file and iis_node module values. This edit feature is only for the generated web.config file. <a href="#">Learn more</a> .
<b>AppSettings</b> (App settings)	(Optional) Edit web app <b>Application</b> settings using the syntax <b>-key value</b> . Values containing spaces must be enclosed in double quotes. Examples: <b>-Port 5000</b> - <b>RequestTimeout 5000</b> and <b>-WEBSITE_TIME_ZONE "Eastern Standard Time"</b> .

PARAMETERS	DESCRIPTION
<input type="checkbox"/> ConfigurationSettings (Configuration settings)	(Optional) Edit web app configuration settings using the syntax <b>-key value</b> . Values containing spaces must be enclosed in double quotes. Example: <b>-phpVersion 5.6 -linuxFxVersion: node 6.11</b>
<input type="checkbox"/> UseWebDeploy (Select deployment method)	(Optional) If unchecked, the task auto-detects the best deployment method based on the app type, package format, and other parameters. Select the option to view the supported deployment methods, and choose one for deploying your app. Argument aliases: <code>enableCustomDeployment</code>
<input type="checkbox"/> DeploymentType (Deployment method)	(Required if UseWebDeploy == true) Choose the deployment method for the app. Default value: <code>webDeploy</code>
<input type="checkbox"/> TakeAppOfflineFlag (Take App Offline)	(Optional) Select this option to take the Azure App Service offline by placing an <code>app_offline.htm</code> file in the root directory before the synchronization operation begins. The file will be removed after the synchronization completes successfully. Default value: <code>true</code>
<input type="checkbox"/> SetParametersFile (SetParameters file)	(Optional) location of the <code>SetParameters.xml</code> file to be used.
<input type="checkbox"/> RemoveAdditionalFilesFlag (Remove additional files at destination)	(Optional) Select the option to delete files on the Azure App Service that have no matching files in the App Service package or folder. <b>Note:</b> This will also remove all files related to any extensions installed on this Azure App Service. To prevent this, set the <b>Exclude files from App_Data folder</b> checkbox. Default value: <code>false</code>
<input type="checkbox"/> ExcludeFilesFromAppDataFlag (Exclude files from the App_Data folder)	(Optional) Select the option to prevent files in the App_Data folder from being deployed to or deleted from the Azure App Service. Default value: <code>true</code>
<input type="checkbox"/> AdditionalArguments (Additional arguments)	(Optional) Additional Web Deploy arguments following the syntax <b>-key:value</b> . These will be applied when deploying the Azure App Service. Example: <b>-disableLink:AppPoolExtension - disableLink:ContentExtension</b> . <a href="#">More examples</a> . Default value: <code>-retryAttempts:6 -retryInterval:10000</code>
<input type="checkbox"/> RenameFilesFlag (Rename locked files)	(Optional) Select this option to enable the MSDeploy flag <code>MSDEPLOY_RENAME_LOCKED_FILES=1</code> in the Azure App Service application settings. When set, it enables MSDeploy to rename files that are locked during app deployment. Default value: <code>true</code>

PARAMETERS	DESCRIPTION
<p><code>XmlTransformation</code> (XML transformation)</p>	<p>(Optional) The configuration transformations will be run for <code>.Release.config</code> and <code>.{EnvironmentName}.config</code> on the <code>*.config</code> files. Configuration transformations run before variable substitution. XML transformations are supported only for the Windows platform. <a href="#">Learn more</a>.</p> <p>Default value: false</p> <p>Argument aliases: <code>enableXmlTransform</code></p>
<p><code>XmlVariableSubstitution</code> (XML variable substitution)</p>	<p>(Optional) Variables defined in the build or release pipeline will be matched against the <code>key</code> or <code>name</code> entries in the <code>appSettings</code>, <code>applicationSettings</code>, and <code>connectionStrings</code> sections of any configuration file and <code>parameters.xml</code> file. Variable substitution runs after configuration transformations.</p> <p><b>Note:</b> if the same variables are defined in the release pipeline and in the stage, the stage variables will supersede the release pipeline variables. <a href="#">Learn more</a></p> <p>Default value: false</p> <p>Argument aliases: <code>enableXmlVariableSubstitution</code></p>
<p><code>JSONFiles</code> (JSON variable substitution)</p>	<p>(Optional) Provide a newline-separated list of JSON files to substitute the variable values. Filenames 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 <code>ConnectionString</code> in the sample below, define a variable named <code>Data.DefaultConnection.ConnectionString</code> in the build or release pipeline (or release pipelines stage).</p> <pre>{   "Data": {     "DefaultConnection": {       "ConnectionString": "Server=(localdb)\SQLEXPRESS;Database=MyDB;Trusted_Connection=True"     }   } }</pre> <p>Variable substitution runs after configuration transformations. <b>Note:</b> build and release pipeline variables are excluded from substitution. <a href="#">Learn more</a>.</p>

This YAML example deploys to an Azure Web App container (Linux).

```
pool:
 vmImage: Ubuntu-16.04

variables:
 azureSubscriptionEndpoint: Contoso
 DockerNamespace: contoso.azurecr.io
 DockerRepository: aspnetcore
 WebAppName: containersdemoapp

steps:
- task: AzureRMWebAppDeployment@4
 displayName: Azure App Service Deploy
 inputs:
 appType: webAppContainer
 ConnectedServiceName: $(azureSubscriptionEndpoint)
 WebAppName: $(WebAppName)
 DockerNamespace: $(DockerNamespace)
 DockerRepository: $(DockerRepository)
 DockerImageTag: $(Build.BuildId)
```

Here is another example that deploys a web app to a Windows App Service across deployment slots.

```

pool:
 vmImage: 'windows-latest'

variables:
 solution: '**/*.sln'
 buildPlatform: 'Any CPU'
 buildConfiguration: 'Release'

steps:
 - stage: DeployDevStage
 displayName: 'Deploy App to Dev Slot'
 jobs:
 - job: DeployApp
 displayName: 'Deploy App'
 steps:
 - task: DownloadPipelineArtifact@2
 inputs:
 buildType: 'current'
 artifactName: 'drop'
 targetPath: '$(System.DefaultWorkingDirectory)'

 - task: AzureRmWebAppDeployment@4
 inputs:
 ConnectionType: 'AzureRM'
 azureSubscription: 'Fabrikam Azure Subscription - PartsUnlimited'
 appType: 'webApp'
 WebAppName: 'partsunlimited'
 deployToSlotOrASE: true
 ResourceGroupName: 'rgPartsUnlimited'
 SlotName: 'Dev'
 packageForLinux: '$(System.DefaultWorkingDirectory)/**/*.zip'

 - stage: DeployStagingStage
 displayName: 'Deploy App to Staging Slot'
 dependsOn: DeployDevStage
 jobs:
 - job: DeployApp
 displayName: 'Deploy App'
 steps:
 - task: DownloadPipelineArtifact@2
 inputs:
 buildType: 'current'
 artifactName: 'drop'
 targetPath: '$(System.DefaultWorkingDirectory)'

 - task: AzureRmWebAppDeployment@4
 inputs:
 appType: webApp
 ConnectionType: AzureRM
 ConnectedServiceName: 'Fabrikam Azure Subscription - PartsUnlimited'
 ResourceGroupName: 'rgPartsUnlimited'
 WebAppName: 'partsunlimited'
 Package: '$(System.DefaultWorkingDirectory)/**/*.zip'
 deployToSlotOrASE: true
 SlotName: 'staging'

```

- To deploy to a specific app type, set `appType` to any of the following accepted values: `webApp` (Web App on Windows), `webAppLinux` (Web App on Linux), `webAppContainer` (Web App for Containers - Linux), `functionApp` (Function App on Windows), `functionAppLinux` (Function App on Linux), `functionAppContainer` (Function App for Containers - Linux), `apiApp` (API App), `mobileApp` (Mobile App). If not mentioned, `webApp` is taken as the default value.
- To enable any advance deployment options, add the parameter `enableCustomDeployment: true` and include the below parameters as needed.

```
deploymentMethod: 'runFromPackage' # supports zipDeploy as well
appOffline: boolean # Not applicable for 'runFromPackage'
setParametersFile: string
removeAdditionalFilesFlag: boolean
additionalArguments: string
```

## 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 `$(variableName.AppServiceApplicationUrl)`, for example `$(FabrikamWebAppURL.AppServiceApplicationUrl)`, 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 Post 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 Package](#)
- [War Deploy](#)

By default, the task tries to select the appropriate deployment technology based on the input package type, App Service type, and agent operating system.

## Auto Detect Logic

For windows based agents.

APP SERVICE TYPE	PACKAGE TYPE	DEPLOYMENT METHOD
WebApp on Linux or Function App on Linux	Folder/Zip/jar War	Zip Deploy War Deploy
WebApp for Containers (Linux) or Function App for Containers (Linux)	Update the App settings	NA

APP SERVICE TYPE	PACKAGE TYPE	DEPLOYMENT METHOD
WebApp on Windows, Function App on Windows, API App, or Mobile App	War Jar MsBuild package type or deploy to virtual application  Folder/Zip	War Deploy Zip Deploy Web Deploy  if postDeploymentScript == true, Zip Deploy else, Run From Package

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 to the Azure App Service using 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

Expects 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](#).

## Run From Package

Expects 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](#).

## War Deploy

Expects a .war deployment package and deploys the file content to the **wwwroot** folder or **webapps** folder of the App Service in Azure.

# Troubleshooting

### Error: Could not fetch access token for Azure. Verify if the Service Principal used is valid and not expired.

The task uses the service principal in the service connection to authenticate with Azure. If the service principal has expired or doesn't have permissions to the App Service, the task fails with this error. Verify the validity of the service principal used and that it's present in the app registration. For more information, 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.

### SSL error

If you want to use a certificate in App Service, the certificate must be signed by a trusted certificate authority. If your web app gives you certificate validation errors, you're probably using a self-signed certificate. Set a variable named `VSTS_ARM_REST_IGNORE_SSL_ERRORS` to the value `true` in the build or release pipeline to resolve the error.

### A release hangs for long time and then fails

This problem could be the result of insufficient capacity in your App Service plan. To resolve this problem, you can scale up the App Service instance to increase available CPU, RAM, and disk space or try with a different App Service plan.

### 5xx error codes

If you're seeing a 5xx error, [check the status of your Azure service](#).

### Error: No package found with specified pattern

Check if the package mentioned in the task is published as an artifact in the build or a previous stage and downloaded in the current job.

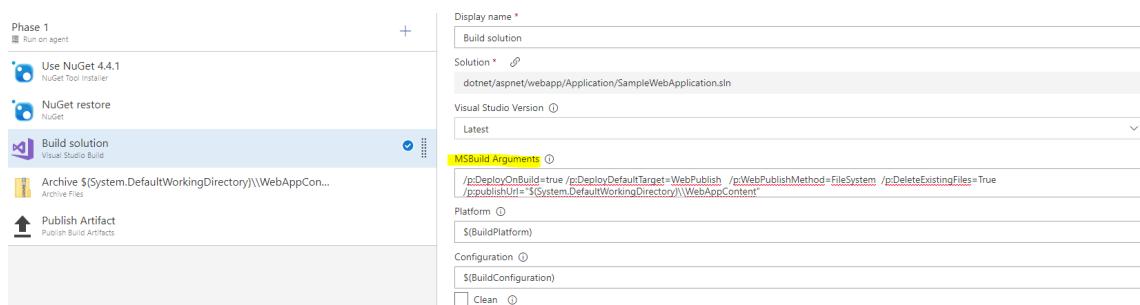
### Error: Publish using zip deploy option is not supported for msBuild package type

Web packages created via the MSBuild task (with default arguments) have a nested folder structure that can be deployed correctly only by Web Deploy. The publish-to-zip deployment option can't be used to deploy those packages. To convert the packaging structure, take these steps:

1. In the Build solution task, change the **MSBuild Arguments** to

```
/p:DeployOnBuild=true /p:DeployDefaultTarget=WebPublish /p:WebPublishMethod=FileSystem
/p:DeleteExistingFiles=True /p:publishUrl="$(System.DefaultWorkingDirectory)\\WebAppContent"
```

:



2. Add an Archive task and change the values as follows:

- a. Change **Root folder or file to archive** to `$(System.DefaultWorkingDirectory)\\WebAppContent`.
- b. Clear the **Prepend root folder name to archive paths** check box:

The screenshot shows the Azure DevOps pipeline editor. On the left, under the 'Get sources' phase, there are tasks: 'Run on agent', 'Use NuGet 4.4.1', 'NuGet restore', 'Build solution', and 'Archive \$System.DefaultWorkingDirectory\\WebAppContent'. On the right, the 'Archive' task is expanded, showing settings for 'Display name' (Archive \$System.DefaultWorkingDirectory\\WebAppContent), 'Root folder or file to archive' (\$System.DefaultWorkingDirectory\\WebAppContent), 'Archive type' (zip), 'Archive file to create' (\$Build.ArtifactStagingDirectory)\\\$(Build.BuildId).zip, and a checked 'Replace existing archive' option.

## Web app deployment on Windows is successful but the app is not working

This may be because web.config is not present in your app. You can either add a web.config file to your source or auto-generate one using the File Transforms and Variable Substitution Options of the task.

- Click on the task and go to Generate web.config parameters for Python, Node.js, Go and Java apps.

The screenshot shows the Azure DevOps pipeline editor with a 'New release pipeline' selected. The 'Tasks' tab is active. A 'Stage 1' deployment process is shown with a 'Run on agent' step and an 'Azure App Service Deploy' step. The 'Azure App Service Deploy' step has a red warning icon indicating 'Some settings need attention'. To the right, the 'Generate web.config parameters for Python, Node.js, Go and Java apps' section is expanded, showing fields for 'Package or folder' (\$System.DefaultWorkingDirectory)\\\*\*/\*.zip and options for 'File Transforms & Variable Substitution Options' like XML transformation and JSON variable substitution.

- Click on the more button Generate web.config parameters for Python, Node.js, Go and Java apps to edit the parameters.

The screenshot shows the 'Generate web.config parameters for Python, Node.js, Go and Java apps' dialog. At the top, it says 'Generate web.config parameters for Python, ...'. Below is a table for 'Application framework' set to 'node':

Name	Value
Handler	iisnode
NodeStartFile	server.js

At the bottom are 'OK' and 'Cancel' buttons.

- Select your application type from the drop down.
- Click on OK. This will populate web.config parameters required to generate web.config.

#### NOTE

This section is deprecated and has been replaced with the [File Transform](#) task.

## ERROR\_FILE\_IN\_USE

When deploying .NET apps to Web App on Windows, deployment may fail with error code *ERROR\_FILE\_IN\_USE*. To resolve the error, ensure *Rename locked files* and *Take App Offline* options are enabled in the task. For zero downtime deployments, use slot swap.

You can also use *Run From Package deployment* method to avoid resource locking.

## Web Deploy Error

If you are using web deploy to deploy your app, in some error scenarios Web Deploy will show an error code in the log. To troubleshoot a web deploy error see [this](#).

## Web app deployment on App Service Environment (ASE) is not working

- Ensure that the Azure DevOps build agent is on the same VNET (subnet can be different) as the Internal Load Balancer (ILB) of ASE. This will enable the agent to pull code from Azure DevOps and deploy to ASE.
- If you are using Azure DevOps, the agent neednt be accessible from internet but needs only outbound access to connect to Azure DevOps Service.
- If you are using TFS/Azure DevOps Server deployed in a Virtual Network, the agent can be completely isolated.
- Build agent must be configured with the DNS configuration of the Web App it needs to deploy to. Since the private resources in the Virtual Network don't have entries in Azure DNS, this needs to be added to the hosts file on the agent machine.
- If a self-signed certificate is used for the ASE configuration, "-allowUntrusted" option needs to be set in the deploy task for MSDeploy. It is also recommended to set the variable `VSTS_ARM_REST_IGNORE_SSL_ERRORS` to true. If a certificate from a certificate authority is used for ASE configuration, this should not be necessary.

## FAQs

### How should I configure my service connection?

This task requires an [Azure Resource Manager service connection](#).

### How should I configure web job deployment with Application Insights?

When you're deploying to an App Service, if you have [Application Insights](#) configured and you've enabled `Remove additional files at destination`, you also need to enable `Exclude files from the App_Data folder`.

Enabling this option keeps the Application Insights extension in a safe state. This step is required because the Application Insights continuous WebJob is installed into the App\_Data folder.

### How should I configure my agent if it's behind a proxy while I'm deploying to App Service?

If your self-hosted agent requires a web proxy, you can inform the agent about the proxy during configuration. Doing so allows your agent to connect to Azure Pipelines or Azure DevOps Server through the proxy. [Learn more about running a self-hosted agent behind a web proxy](#).

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Azure App Service Manage task

3/6/2021 • 3 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task to start, stop, restart, slot swap, Swap with Preview, install site extensions, or enable continuous monitoring for an Azure App Service.

## YAML snippet

```
Azure App Service manage
Start, stop, restart, slot swap, slot delete, 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 Swap With Preview, Complete Swap, Cancel
 Swap, Start Azure App Service, Stop Azure App Service, Restart Azure App Service, Delete Slot, Install
 Extensions, Enable Continuous Monitoring, Start all continuous webjobs, Stop all continuous webjobs
 webAppName:
 #specifySlotOrASE: false # Optional
 #resourceGroupName: # Required when action == Swap Slots || Action == Delete Slot || 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 == Delete Slot || 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
<code>ConnectedServiceName</code> Azure subscription	(Required) Select the Azure Resource Manager subscription Argument alias: <code>azureSubscription</code>
<code>Action</code> Action	(Optional) Action to be performed on the App Service. You can Start, Stop, Restart, Slot swap, Start Swap with Preview, Complete Swap with preview, Cancel Swap with preview, Install site extensions or enable Continuous Monitoring for an Azure App Service Default value: <code>Swap Slots</code>
<code>WebAppName</code> App Service name	(Required) Enter or select the name of an existing Azure App Service
<code>SpecifySlot</code> Specify Slot or App Service Environment	(Optional) undefined

ARGUMENT	DESCRIPTION
<code>ResourceGroupName</code> Resource group	(Required) Enter or Select the Azure Resource Group that contains the Azure App Service specified above
<code>SourceSlot</code> Source Slot	(Required) Used as source slot when <code>action == Swap Slots</code> . The swap action directs destination slot's traffic to the source slot.
<code>SwapWithProduction</code> 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. Default value: <code>true</code>
<code>TargetSlot</code> Target Slot	(Required) Used as destination slot when <code>action == Swap Slots</code> . The swap action directs destination slot's traffic to the source slot.
<code>PreserveVnet</code> Preserve Vnet	(Optional) The swap action would overwrite the destination slot's network configuration with the source Default value: <code>false</code>
<code>Slot</code> Slot	(Required) Default value: <code>production</code>
<code>ExtensionsList</code> 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.
<code>OutputVariable</code> 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.
<code>AppInsightsResourceGroupName</code> Resource Group name for Application Insights	(Required) Enter or Select resource group where your application insights resource is available
<code>ApplicationInsightsResourceName</code> 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.
<code>ApplicationInsightsWebTestName</code> Application Insights web test name	(Optional) Enter Application Insights Web Test name to be created or updated. If not provided, the default test name will be used.

## What happens during a swap

When you swap two slots (usually from a staging slot into the production slot), make sure that the production slot is always the target slot. This way, the swap operation doesn't affect your production app.

Also at any point of the swap (or swap with preview) operation, all work of initializing the swapped apps happens on the source slot. The target slot remains online while the source slot is being prepared and warmed

up, regardless of where the swap succeeds or fails. Please refer to [Set up staging environments in Azure App Service](#) for more details.

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Azure App Service Settings task

6/25/2021 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task to configure App settings, connection strings and other general settings in bulk using JSON syntax on your web app or any of its deployment slots. The task works on cross platform Azure Pipelines agents running Windows, Linux or Mac. The task works for ASP.NET, ASP.NET Core, PHP, Java, Python, Go and Nodejs based web applications.

## Arguments

PARAMETERS	DESCRIPTION
<code>ConnectedServiceName</code> Azure subscription	(Required) Name of the Azure Resource Manager service connection Argument aliases: <code>ConnectedServiceName</code>
<code>appName</code> App name	(Required) Name of an existing App Service
<code>resourceGroupName</code> Resource group	(Optional) Name of the resource group
<code>slotName</code> Slot	(Optional) Name of the slot Default value: <code>production</code>
<code>appSettings</code> App settings	(Optional) Application settings to be entered using JSON syntax. Values containing spaces should be enclosed in double quotes.
<code>generalSettings</code> General settings	(Optional) General settings to be entered using JSON syntax. Values containing spaces should be enclosed in double quotes. See the <a href="#">App Service SiteConfig object documentation</a> for the available properties.
<code>connectionStrings</code> Connection settings	(Optional) Connection strings to be entered using JSON syntax. Values containing spaces should be enclosed in double quotes.

Following is an example YAML snippet to deploy web application to the Azure Web App service running on windows.

## Example

```

variables:
 azureSubscription: Contoso
 WebApp_Name: sampleWebApp
 # To ignore SSL error uncomment the below variable
 # VSTS_ARM_REST_IGNORE_SSL_ERRORS: true

steps:

- task: AzureWebApp@1
 displayName: Azure Web App Deploy
 inputs:
 azureSubscription: $(azureSubscription)
 appName: $(WebApp_Name)
 package: $(System.DefaultWorkingDirectory)/**/*.zip

- task: AzureAppServiceSettings@1
 displayName: Azure App Service Settings
 inputs:
 azureSubscription: $(azureSubscription)
 appName: $(WebApp_Name)
 # To deploy the settings on a slot, provide slot name as below. By default, the settings would be applied
 to the actual Web App (Production slot)
 # slotName: staging
 appSettings: |
 [
 {
 "name": "APPINSIGHTS_INSTRUMENTATIONKEY",
 "value": "$(Key)",
 "slotSetting": false
 },
 {
 "name": "MYSQL_DATABASE_NAME",
 "value": "$(DB_Name)",
 "slotSetting": false
 }
]
 generalSettings: |
 [
 {
 "alwaysOn": true,
 "webSocketsEnabled": false
 }
]
 connectionStrings: |
 [
 {
 "name": "MysqlCredentials",
 "value": "$(MySQL_ConnectionString)",
 "type": " MySql",
 "slotSetting": false
 }
]

```

## Open Source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Azure CLI task

3/6/2021 • 4 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task 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.

### What's new in Version 2.0

- Supports running PowerShell and PowerShell Core script
- PowerShell Core script works with Xplat agents (Windows, Linux or OSX), make sure the agent has PowerShell version 6 or higher
- PowerShell script works only with Windows agent, make sure the agent has PowerShell version 5 or lower

## Prerequisites

- A Microsoft Azure subscription
- [Azure Resource Manager service connection](#) to your Azure account
- Microsoft hosted agents have Azure CLI pre-installed. However if you are using private agents, [install Azure CLI](#) on the computer(s) that run the build and release agent. 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.

## Task Inputs

PARAMETERS	DESCRIPTION
<code>azureSubscription</code> Azure subscription	(Required) Select an Azure Resource Manager subscription for the deployment. 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
<code>scriptType</code> Script Type	(Required) Type of script: <b>PowerShell/PowerShell Core/Bat/Shell</b> script. Select <b>bash/pscore</b> script when running on Linux agent or <b>batch/ps/pscore</b> script when running on Windows agent. PowerShell Core script can run on cross-platform agents (Linux, macOS, or Windows)
<code>scriptLocation</code> Script Location	(Required) Path to script: File path or Inline script Default value: <code>scriptPath</code>
<code>scriptPath</code> Script Path	(Required) Fully qualified path of the script(.ps1 or .bat or .cmd when using Windows-based agent else <code>.ps1</code> or <code>.sh</code> when using linux-based agent) or a path relative to the default working directory

PARAMETERS	DESCRIPTION
<code>inlineScript</code> Inline Script	(Required) You can write your scripts inline here. When using Windows agent, use PowerShell or PowerShell Core or batch scripting whereas use PowerShell Core or shell scripting when using Linux-based agents. For batch files use the prefix <code>\call\</code> before every Azure command. You can also pass predefined and custom variables to this script using arguments. <b>Example for PowerShell/PowerShellCore/shell:</b> az --version az account show <b>Example for batch:</b> call az --version call az account show
<code>arguments</code> Script Arguments	(Optional) Arguments passed to the script
<code>powershellErrorActionPreference</code> ErrorActionPreference	(Optional) Prepends the line <code>\$ErrorActionPreference = 'VALUE'</code> at the top of your PowerShell/PowerShell Core script Default value: stop Options are stop, continue, and silentlyContinue
<code>addSpnToEnvironment</code> Access service principal details in script	(Optional) Adds service principal id and key of the Azure endpoint you chose to the script's execution environment. You can use these variables: <code>\$env:servicePrincipalId</code> , <code>\$env:servicePrincipalKey</code> and <code>\$env:tenantId</code> in your script. This is honored only when the Azure endpoint has Service Principal authentication scheme Default value: false
<code>useGlobalConfig</code> Use global Azure CLI configuration	(Optional) If this is false, this task will use its own separate <a href="#">Azure CLI configuration directory</a> . This can be used to run Azure CLI tasks in parallel releases" Default value: false
<code>workingDirectory</code> Working Directory	(Optional) Current working directory where the script is run. Empty is the root of the repo (build) or artifacts (release), which is <code>\$(System.DefaultWorkingDirectory)</code>
<code>failOnStandardError</code> Fail on Standard Error	(Optional) If this is true, this task will fail when any errors are written to the StandardError stream. Unselect the checkbox to ignore standard errors and rely on exit codes to determine the status Default value: false
<code>powershellIgnoreLASTEXITCODE</code> Ignore \$LASTEXITCODE	(Optional) If this is false, the line <pre>if ((Test-Path -LiteralPath variable:\\LASTEXITCODE)) { exit \$LASTEXITCODE }</pre> is appended to the end of your script. This will cause the last exit code from an external command to be propagated as the exit code of PowerShell. Otherwise the line is not appended to the end of your script Default value: false

## Example

Following is an example of a YAML snippet that lists the version of Azure CLI and gets the details of the subscription.

```
- task: AzureCLI@2
displayName: Azure CLI
inputs:
 azureSubscription: <Name of the Azure Resource Manager service connection>
 scriptType: ps
 scriptLocation: inlineScript
 inlineScript: |
 az --version
 az account show
```

The following example illustrates how to pass arguments to your script.

```
- task: AzureCLI@2
displayName: Azure CLI
inputs:
 azureSubscription: <Name of the Azure Resource Manager service connection>
 scriptType: ps
 scriptLocation: inlineScript
 arguments:
 -Arg1 val1 `
 -Arg2 val2 `
 -Arg3 val3
 inlineScript: |
 az login --allow-no-subscription
```

## 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.

## FAQ

### Do I need an agent?

You need at least one [agent](#) to run your build or release.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

# Azure Cloud Service Deployment task

4/22/2020 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task to deploy an Azure Cloud Service.

## YAML snippet

```
Azure Cloud Service deployment
Deploy an Azure Cloud Service
- task: AzureCloudPowerShellDeployment@1
 inputs:
 azureClassicSubscription:
 #storageAccount: # Required when enableAdvancedStorageOptions == False
 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
 #enableAdvancedStorageOptions: false
 #aRMConnectedServiceName: # Required when enableAdvancedStorageOptions == True
 #aRMSStorageAccount: # Required when enableAdvancedStorageOptions == True
```

## Arguments

ARGUMENT	DESCRIPTION
<code>ConnectedServiceName</code> Azure subscription (Classic)	(Required) Azure Classic subscription to target for deployment. Argument alias: <code>azureClassicSubscription</code>
<code>EnableAdvancedStorageOptions</code> Enable Azure Resource Manager storage support	(Required) Select to enable Azure Resource Manager storage support for this task
<code>StorageAccount</code> Storage account (Classic)	(Required) Storage account must exist prior to deployment.
<code>ARMConnectedServiceName</code> Azure subscription (Azure Resource Manager)	(Required) Azure Resource Manager subscription
<code>ARMStorageAccount</code> Storage account (Azure Resource Manager)	(Required) Choose a pre-existing Azure Resource Manager storage account

ARGUMENT	DESCRIPTION
<b>ServiceName</b> Service name	(Required) Select or enter an existing cloud service name.
<b>ServiceLocation</b> 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.
<b>CsPkg</b> CsPkg	(Required) Path of CsPkg under the default artifact directory.
<b>CsCfg</b> CsCfg	(Required) Path of CsCfg under the default artifact directory.
<b>Slot</b> Environment (Slot)	(Required) Production or Staging Default value: <code>Production</code> Argument alias: <code>slotName</code>
<b>DeploymentLabel</b> Deployment label	(Optional) Specifies the label name for the new deployment. If not specified, a Globally Unique Identifier (GUID) is used. Default value: <code>\$(Build.BuildNumber)</code>
<b>AppendDateTimeToLabel</b> Append current date and time	(Optional) Appends current date and time to deployment label Default value: <code>false</code>
<b>AllowUpgrade</b> Allow upgrade	(Required) When selected allows an upgrade to the Microsoft Azure deployment Default value: <code>true</code>
<b>SimultaneousUpgrade</b> Simultaneous upgrade	(Optional) Updates all instances at once. Your cloud service will be unavailable during update. Default value: <code>false</code>
<b>ForceUpgrade</b> Force upgrade	(Optional) When selected sets the upgrade to a forced upgrade, which could potentially cause loss of local data. Default value: <code>false</code>
<b>VerifyRoleInstanceStatus</b> Verify role instance status	When selected then the task will wait until role instances are in ready state
<b>DiagnosticStorageAccountKeys</b> Diagnostic storage account keys	(Optional) Provide storage keys for diagnostics storage account in <code>Role:Storagekey</code> format. The diagnostics storage account name for each role will be obtained from diagnostics config file <code>(.wadcfgx)</code> . If the <code>.wadcfgx</code> 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 <code>.wadcfgx</code> 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 <code>&lt;storage_account_key&gt;</code> as a secret variable unless there is no sensitive information in the diagnostics result for your stage. For example, WebRole: <code>&lt;WebRole_storage_account_key&gt;</code> WorkerRole: <code>&lt;WorkerRole_storage_account_key&gt;</code>

ARGUMENT	DESCRIPTION
<b>NewServiceCustomCertificates</b> Custom certificates to import	(Optional) Provide custom certificates in CertificatePfxBase64:CertificatePassword format. It's recommended to save <certificate_password> as a secret variable. For example, Certificate1: <Certificate1_password> Certificate2: <Certificate2_password>
<b>NewServiceAdditionalArguments</b> Additional arguments	(Optional) Pass in additional arguments while creating a brand new service. These will be passed on to New-AzureService cmdlet. Eg: -Label 'MyTestService'
<b>NewServiceAffinityGroup</b> Affinity group	(Optional) While creating new service, this affinity group will be considered instead of using service location.

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Azure File Copy task

6/30/2021 • 11 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015.3

Use this task to copy files to Microsoft Azure storage blobs or virtual machines (VMs).

## NOTE

This task is written in PowerShell and thus works **only** when run on Windows agents. If your pipelines require Linux agents and need to copy files to an Azure Storage Account, consider running `az storage blob` commands in the [Azure CLI task](#) as an alternative.

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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.

## NOTE

If you are using Azure File copy task version 3 or below, see [Azure file copy version 3 or earlier](#).

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. The version 4 of the Azure File Copy task uses [AzCopy V10](#).

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.

The task supports authentication based on Azure Active Directory. Authentication using a service principal and managed identity are available. For managed identities, only system-wide managed identity is supported.

## NOTE

For authorization you will have to provide access to the Security Principal. The level of authorization required can be referred [here](#).

## YAML snippet

```
Azure file copy
Copy files to Azure Blob Storage or virtual machines
- task: AzureFileCopy@4
 inputs:
 sourcePath:
 azureSubscription:
 destination: # Options: azureBlob, azureVMs
 storage:
 #containerName: # Required when destination == AzureBlob
 #blobPrefix: # Optional
 #resourceGroup: # Required when 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
 #sasTokenTimeOutInMinutes: # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<code>sourcePath</code> - Source	Required. The source of the files to copy. YAML Pipelines and Classic Release support <a href="#">pre-defined system variables</a> like <code>Build.Repository.LocalPath</code> as well. <a href="#">Release variables</a> are supported only in classic releases. Wild card symbol (*) is supported anywhere in the file path or file name.
<code>connectedServiceNameARM</code> or <code>azureSubscription</code> - Azure Subscription	Required. The name of an <a href="#">Azure Resource Manager service connection</a> configured for the subscription where the target Azure service, virtual machine, or storage account is located. See <a href="#">Azure Resource Manager overview</a> for more details.
<code>destination</code> - Destination Type	Required. The type of target destination for the files. Choose <a href="#">Azure Blob</a> or <a href="#">Azure VMs</a> .
<code>storageAccountRM</code> or <code>storage</code> - RM Storage Account	Required. The name of an existing storage account within the Azure subscription.
<code>containerName</code> - Container Name	Required if you select <a href="#">Azure Blob</a> for the <a href="#">Destination Type</a> 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.
<code>blobPrefix</code> - Blob Prefix	Optional if you select <a href="#">Azure Blob</a> for the <a href="#">Destination Type</a> 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.

ARGUMENT	DESCRIPTION
<code>environmentNameRM</code> or <code>resourceGroup</code> - Resource Group	Required if you select <b>Azure Resource Manager</b> for the <b>Azure Connection Type</b> parameter and <b>Azure VMs</b> for the <b>Destination Type</b> parameter. The name of the Azure Resource Group in which the virtual machines run.
<code>resourceFilteringMethod</code> - Select Machines By	Depending on how you want to specify the machines in the group when using the <b>Filter Criteria</b> parameter, choose <b>Machine Names</b> or <b>Tags</b> .
<code>machineNames</code> - 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"> <li>- The name of an <a href="#">Azure Resource Group</a>.</li> <li>- An output variable from a previous task.</li> <li>- A comma-delimited list of tag names or machine names. Format when using machine names is a comma-separated list of the machine FQDNs or IP addresses.</li> </ul> <p>Specify tag names for a filter as {TagName}:{Value} Example:  <code>Role:DB;OS:Win8.1</code></p>
<code>vmsAdminUserName</code> - Admin Login	<p>Required if you select <b>Azure VMs</b> for the <b>Destination Type</b> parameter. The user name of an account that has administrative permissions for all the target VMs.</p> <ul style="list-style-type: none"> <li>- Formats such as <code>username</code>, <code>domain\username</code>, <code>machine-name\username</code>, and <code>.\[username</code> are supported.</li> <li>- UPN formats such as <code>username@domain.com</code> and built-in system accounts such as <code>NT Authority\System</code> are not supported.</li> </ul>
<code>vmsAdminPassword</code> - Password	<p>Required if you select <b>Azure VMs</b> for the <b>Destination Type</b> parameter. The password for the account specified as the <b>Admin Login</b> parameter. Use the padlock icon for a variable defined in the <b>Variables</b> tab to protect the value, and insert the variable name here.</p>
<code>targetPath</code> - Destination Folder	<p>Required if you select <b>Azure VMs</b> for the <b>Destination Type</b> 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.</p> <p>Examples: <code>\$env:windir\FabrikamFiber\Web</code> and <code>c:\FabrikamFiber</code></p>
<code>additionalArgumentsForBlobCopy</code> - Optional Arguments (for uploading files to blob)	<p>Optional. Any arguments you want to pass to the <code>AzCopy.exe</code> program for use when uploading to the blob and downloading to the VMs. See <a href="#">Transfer data with the AzCopy Command-Line Utility</a> for more details. If you are using a Premium storage account, which supports only Azure page blobs, then pass <code>--blob-type=PageBlob</code> as an additional argument. The default arguments are <code>--log-level=INFO</code> (default) and <code>--recursive</code> (only if container name is not <code>\$root</code>).</p>

Argument	Description
<code>additionalArgumentsForVMCopy</code> - Optional Arguments (for downloading files to VM)	Optional. AzCopy.exe arguments that will be applied when downloading to VM like, <code>--check-length=true</code> . If no optional arguments are specified here, the following optional arguments will be added by default: <code>--log-level=INFO</code> (if the pipeline is running in debug mode set <code>--log-level=DEBUG</code> ), <code>--recursive</code>
<code>enableCopyPrerequisites</code> - Enable Copy Prerequisites	Available if you select <b>Azure Resource Manager</b> for the <b>Azure Connection Type</b> parameter and <b>Azure VMs</b> for the <b>Destination Type</b> 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. - 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.
<code>copyFilesInParallel</code> - Copy in Parallel	Available if you select <b>Azure VMs</b> for the <b>Destination Type</b> parameter. Setting this option causes the process to execute in parallel for the copied files. This can considerably reduce the overall time taken.
<code>cleanTargetBeforeCopy</code> - Clean Target	Available if you select <b>Azure VMs</b> for the <b>Destination Type</b> parameter. Setting this option causes all of the files in the destination folder to be deleted before the copy process starts.
<code>skipCACheck</code> - Test Certificate	Available if you select <b>Azure VMs</b> for the <b>Destination Type</b> 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).
<code>sasTokenTimeOutInMinutes</code> - SAS Token Expiration Period In Minutes	Optional. Provide the time in minutes after which SAS token will expire. Valid only when the selected destination is Azure Blob. Default should be 4 hours.
Control options	See <a href="#">Control options</a>

## Related tasks

- [Azure Resource Group Deployment](#)
- [Azure Cloud Service Deployment](#)
- [Azure Web App Deployment](#)

## FAQ

### 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. [Configure WinRM after virtual machine creation](#)

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.

#### What type of service connection should I choose?

- For Azure Resource Manager storage accounts and 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](#).
- While using an **Azure Resource Manager** service connection type, the task automatically filters appropriate newer Azure Resource Manager storage accounts, and other fields. For example, the Resource Group or cloud service, and the virtual machines.

#### 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.

#### If the task fails, will the copy resume?

Since AzCopy V10 does not support journal files, the task cannot resume the copy. You will have to run the task again to copy all the files.

#### Are the log files and plan files cleaned after the copy?

The log and plan files are not deleted by the task. To explicitly clean up the files you can add a CLI step in the workflow using [this command](#).

#### Do I need an agent?

You need at least one [agent](#) to run your build or release.

#### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

#### 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](#).

#### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

## 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](#).

## How do I use the Azure file copy task to copy a file to an Azure virtual machine that doesn't have a public IP address?

Make sure that you're using version 4 of the Azure file copy task. If the task fails, you can add a build step to execute the command `azcopy cp "source-file-path" "destination-file-path"` to substitute the source and destination values.

## Forbidden error: 'AzCopy.exe exited with non-zero exit code while uploading files to blob storage' while using Azure File Copy task

The hosted agents are assigned randomly every time a build is triggered and hence the [agent IP addresses](#) will be different on every run. If these IP addresses are not in your allowed list of IPs, the communication between Azure DevOps and the storage account fails. In such scenarios, follow the below steps outlined:

1. Add a build step using Azure CLI, which will identify the IP of the Microsoft Hosted Build agent at runtime and it will add the IP address in the Network rule on the Azure Storage Account.
2. Execute the build step for your Azure Storage Account.
3. Add another build step using Azure CLI, which will remove the IP address of the build agent from the Azure Storage Account network rule which was added earlier.

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Azure File Copy task version 3 or earlier

11/2/2020 • 11 minutes to read • [Edit Online](#)

[Azure Pipelines](#) | [Azure DevOps Server 2020](#) | [Azure DevOps Server 2019](#) | [TFS 2018 - TFS 2015.3](#)

Use this task to copy files to Microsoft Azure storage blobs or virtual machines (VMs).

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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. The task version 3 or below uses AzCopy V7.

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.

## YAML snippet

```

Azure file copy
Copy files to Azure Blob Storage or virtual machines
- task: AzureFileCopy@3
 inputs:
 sourcePath:
 azureSubscription:
 destination: # Options: azureBlob, azureVMs
 storage:
 #containerName: # Required when destination == AzureBlob
 #blobPrefix: # Optional
 #resourceGroup: # Required when 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
 #sasTokenTimeOutInMinutes: # Optional

```

```

Example: Upload files from Pipeline staging directory to blob storage.
- task: AzureFileCopy@2
 displayName: 'Example Step Name'
 inputs:
 sourcePath: '$(Build.ArtifactStagingDirectory)/BlobsToUpload'
 additionalArgumentsForBlobCopy: |
 '/Y' # Suppresses all AZCopy Confirmations. Used here to allow overwrites
 '/Pattern:/*' # Pattern of files to copy.
 '/S' # Recursive Copy
 azureSubscription: 'Subscription Name'
 destination: AzureBlob
 storage: storageaccountname
 containerName: storagecontainername
 blobPrefix: targetdirectoryincontainer

```

## Arguments

ARGUMENT	DESCRIPTION
<b>Source</b>	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.
<b>Azure Connection Type</b>	Required. Select the type of service connection used to define the connection to Azure. Choose <b>Azure Classic</b> or <b>Azure Resource Manager</b> .
<b>Azure Classic Subscription</b>	Required if you select <b>Azure Classic</b> for the <b>Azure Connection Type</b> parameter. The name of an <a href="#">Azure Classic service connection</a> configured for the subscription where the target Azure service, virtual machine, or storage account is located.

ARGUMENT	DESCRIPTION
Azure RM Subscription	Required if you select <b>Azure Resource Manager</b> for the <b>Azure Connection Type</b> parameter. The name of an <a href="#">Azure Resource Manager service connection</a> configured for the subscription where the target Azure service, virtual machine, or storage account is located. See <a href="#">Azure Resource Manager overview</a> for more details.
Destination Type	Required. The type of target destination for the files. Choose <b>Azure Blob</b> or <b>Azure VMs</b> .
Classic Storage Account	Required if you select <b>Azure Classic</b> for the <b>Azure Connection Type</b> parameter. The name of an existing storage account within the Azure subscription.
RM Storage Account	Required if you select <b>Azure Resource Manager</b> for the <b>Azure Connection Type</b> parameter. The name of an existing storage account within the Azure subscription.
Container Name	Required if you select <b>Azure Blob</b> for the <b>Destination Type</b> 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 <b>Azure Blob</b> for the <b>Destination Type</b> 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 <b>Azure Classic</b> for the <b>Azure Connection Type</b> parameter and <b>Azure VMs</b> for the <b>Destination Type</b> parameter. The name of the Azure Cloud Service in which the virtual machines run.
Resource Group	Required if you select <b>Azure Resource Manager</b> for the <b>Azure Connection Type</b> parameter and <b>Azure VMs</b> for the <b>Destination Type</b> 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 <b>Filter Criteria</b> parameter, choose <b>Machine Names</b> or <b>Tags</b> .
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"> <li>- The name of an <a href="#">Azure Resource Group</a>.</li> <li>- An output variable from a previous task.</li> <li>- A comma-delimited list of tag names or machine names. Format when using machine names is a comma-separated list of the machine FQDNs or IP addresses.</li> </ul> <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>

Argument	Description
<b>Admin Login</b>	<p>Required if you select <b>Azure VMs</b> for the <b>Destination Type</b> parameter. The user name of an account that has administrative permissions for all the target VMs.</p> <ul style="list-style-type: none"> <li>- Formats such as <code>username</code>, <code>domain\username</code>, <code>machine-name\username</code>, and <code>.\username</code> are supported.</li> <li>- UPN formats such as <code>username@domain.com</code> and built-in system accounts such as <code>NT Authority\System</code> are not supported.</li> </ul>
<b>Password</b>	<p>Required if you select <b>Azure VMs</b> for the <b>Destination Type</b> parameter. The password for the account specified as the <b>Admin Login</b> parameter. Use the padlock icon for a variable defined in the <b>Variables</b> tab to protect the value, and insert the variable name here.</p>
<b>Destination Folder</b>	<p>Required if you select <b>Azure VMs</b> for the <b>Destination Type</b> 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.</p> <p>Examples: <code>\$env:windir\FabrikamFiber\Web</code> and <code>c:\FabrikamFiber</code></p>
<b>Additional Arguments</b>	<p>Optional. Any arguments you want to pass to the <b>AzCopy.exe</b> program for use when uploading to the blob and downloading to the VMs. See <a href="#">Transfer data with the AzCopy Command-Line Utility</a> 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>
<b>Enable Copy Prerequisites</b>	<p>Available if you select <b>Azure Resource Manager</b> for the <b>Azure Connection Type</b> parameter and <b>Azure VMs</b> for the <b>Destination Type</b> 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"> <li>- If the target virtual machines are accessed through a load balancer, ensure an inbound NAT rule is configured to allow access on port 5986.</li> <li>- If the target virtual machines are associated with a Network Security Group (NSG), configure an inbound security rule to allow access on port 5986.</li> </ul>
<b>Copy in Parallel</b>	<p>Available if you select <b>Azure VMs</b> for the <b>Destination Type</b> parameter. Setting this option causes the process to execute in parallel for the copied files. This can considerably reduce the overall time taken.</p>
<b>Clean Target</b>	<p>Available if you select <b>Azure VMs</b> for the <b>Destination Type</b> parameter. Setting this option causes all of the files in the destination folder to be deleted before the copy process starts.</p>

Argument	Description
Test Certificate	Available if you select <b>Azure VMs</b> for the <b>Destination Type</b> 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).
Output - Storage Container URI	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.
Output - Storage Container SAS Token	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.
Control options	See <a href="#">Control options</a>

## 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.

## FAQ

### 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.

### 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.

#### How do I fix failure '403: This request is not authorized to perform this operation using this permission'?

When Azure DevOps creates and authorizes the service connection to Azure, it creates an App Registration in your subscription's Active Directory. This identity is automatically added with a `Contributor` role to all resources in the Resource Group you chose to authorize. In order to upload blobs to a storage account, being a `Contributor` is *not enough*. You must manually assign the `Storage Blob Data Contributor` role to the app registration identity.

You can copy the app identity from the existing inherited entry as `Contributor` that you will see in the IAM pane and search explicitly for it in the `Add role assignment` UI. Note that the identity will *not* be listed in the dropdown and you will need to search for its identifier.

#### 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:

- Use the Azure portal to create a new user account in Azure Active Directory.
- Add the Azure Active Directory user account to the co-administrators group in your Azure subscription.
- Sign into the Azure portal with this user account and change the password.
- 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.

#### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

**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](#).

**My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?**

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

**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 Function App task

3/6/2021 • 7 minutes to read • [Edit Online](#)

## Azure Pipelines

Use the Azure Function App task to deploy [functions](#) to Azure.

## Arguments

PARAMETER	DESCRIPTION
<code>azureSubscription</code> (Azure subscription)	(Required) Name of the <a href="#">Azure Resource Manager service connection</a> .
<code>appType</code> (App type)	(Required) Function app type.
<code>appName</code> (App name)	(Required) Name of an existing app service.
<code>deployToSlotOrASE</code> (Deploy to slot or App Service environment)	(Optional) Whether to deploy to an existing deployment slot or Azure App Service environment. For both targets, the task needs a resource group name. If the deployment target is a slot, the functions are deployed to the production slot by default. You can also provide any other existing slot name. If the deployment target is an App Service environment, specify the resource group name.  Default value: <code>false</code>
<code>resourceGroupName</code> (Resource group)	(Required if <code>deployToSlotOrASE</code> is <code>true</code> ) Name of the resource group.
<code>slotName</code> (Slot)	(Required if <code>deployToSlotOrASE</code> is <code>true</code> ) Name of the slot.  Default value: <code>production</code>
<code>package</code> (Package or folder)	(Required) File path of the package or of a folder that contains App Service content generated by MSBuild or a compressed .zip or .war file.  Variables ( <a href="#">Build</a>   <a href="#">Release</a> ) and wildcards are supported. For example, <code>\$(System.DefaultWorkingDirectory)/*.zip</code> or <code>\$(System.DefaultWorkingDirectory)/*.war</code> .
<code>runtimeStack</code> (Runtime stack)	(Optional) Web App on Linux provides two ways to publish an application: custom image deployment (Web App for Containers) and app deployment with a built-in platform image (Web App on Linux). You'll see this parameter only when you select <b>Linux Web App</b> as the app type in the task.

PARAMETER	DESCRIPTION
<code>startUpCommand</code> (Startup command)	(Optional. Relevant if <code>appType</code> is <code>webAppLinux</code> .) Startup command to be run after deployment.
<code>customWebConfig</code> (Generate web.config file parameters for Python, Node.js, Go, and Java apps)	(Optional) A standard web.config file will be generated and deployed to App Service if the application doesn't have one. You can edit the values in the web.config file. They'll vary depending on the application framework. For example, for a Node.js application, the web.config file will have startup file and <code>iis_node</code> module values. This edit feature is only for the generated web.config file. <a href="#">Learn more</a> .
<code>appSettings</code> (App settings)	(Optional) Application settings. Use the syntax <code>-key value</code> . Enclose values that contain spaces in double quotation marks.  For example, <div style="border: 1px solid #ccc; padding: 5px; margin-left: 20px;"> <code>-Port 5000 -RequestTimeout 5000 -WEBSITE_TIME_ZONE "Eastern Standard Time"</code> </div> .
<code>configurationStrings</code> (Configuration settings)	(Optional) Configuration strings. Use the syntax <code>-key value</code> . Enclose values that contain spaces in double quotation marks.  For example, <div style="border: 1px solid #ccc; padding: 5px; margin-left: 20px;"> <code>-phpVersion 5.6 -linuxFxVersion: node 6.11</code> </div>
<code>deploymentMethod</code> (Deployment method)	(Required) <a href="#">Deployment method</a> for the app.  Default value: <code>auto</code>

## Example

Here's a sample YAML snippet that deploys Azure functions on Windows:

```

variables:
 azureSubscription: Contoso
 # To ignore SSL error, uncomment the below variable
 # VSTS_ARM_REST_IGNORE_SSL_ERRORS: true

steps:
- task: AzureFunctionApp@1
 displayName: Azure Function App Deploy
 inputs:
 azureSubscription: $(azureSubscription)
 appName: samplefunctionapp
 package: $(System.DefaultWorkingDirectory)/**/*.zip

```

To deploy a function on Linux, add the `appType` parameter and set it to `appType: functionAppLinux`. If you don't specify a value, `functionApp` is the default.

To explicitly specify the deployment method as Zip Deploy, add the parameter `deploymentMethod: zipDeploy`. Another supported value for this parameter is `runFromPackage`. If you don't specify a value, `auto` is the default.

For a walkthrough that shows how to create a CI/CD pipeline, see [Build and deploy Java to Azure Functions](#).

# Deployment methods

Several deployment methods are available in this task. The default value is `auto`.

To change the deployment option in a designer task, expand **Additional Deployment Options** and enable **Select deployment method** to choose from other package-based deployment options.

Based on the type of Azure App Service and Azure Pipelines agent, the task uses a suitable deployment technology. The deployment technologies used by tasks are as follows:

- Kudu REST APIs
- Zip Deploy
- Run From Package

By default, the task attempts to select the appropriate deployment technology based on the input package, App Service type, and agent OS.

- If a post-deployment script is provided, use Zip Deploy.
- If the App Service type is Web App on Linux, use Zip Deploy.
- If a .war file is provided, use War Deploy.
- If a .jar file is provided, use Run-From-Zip.
- For all other tasks, use Run From Package (via Zip Deploy).

On a non-Windows agent (for any App Service type), the task relies on the [Kudu REST API](#) to deploy the web app.

## Kudu REST API

The [Kudu REST API](#) works on both Windows and Linux automation agents when the target is a Web App on Windows, a Web App on Linux (built-in source), or a function app. The task uses Kudu to copy files to the Azure App Service.

## Zip Deploy

Zip Deploy creates a .zip deployment package from the chosen package or folder. It then deploys the file contents to the wwwroot folder of the App Service name function app in Azure. This option overwrites all existing content in the wwwroot folder. For more information, see [Zip deployment for Azure Functions](#).

## Run From Package

Run From Package creates the same deployment package as Zip Deploy. Instead of deploying files to the wwwroot folder, the Functions runtime mounts the entire package. When you use this option, files in the wwwroot folder become read-only. For more information, see [Run your Azure Functions from a package file](#).

# Troubleshooting

## Error: Could not fetch access token for Azure. Verify if the Service Principal used is valid and not expired.

The task uses the service principal in the service connection to authenticate with Azure. If the service principal has expired or doesn't have permissions to the App Service, the task fails with this error. Verify the validity of the service principal used and that it's present in the app registration. For more information, 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.

## SSL error

If you want to use a certificate in App Service, the certificate must be signed by a trusted certificate authority. If your web app gives you certificate validation errors, you're probably using a self-signed certificate. Set a variable named `VSTS_ARM_REST_IGNORE_SSL_ERRORS` to the value `true` in the build or release pipeline to resolve the error.

## A release hangs for long time and then fails

This problem could be the result of insufficient capacity in your App Service plan. To resolve this problem, you can scale up the App Service instance to increase available CPU, RAM, and disk space or try with a different App Service plan.

## 5xx error codes

If you're seeing a 5xx error, [check the status of your Azure service](#).

### Error: No package found with specified pattern

Check if the package mentioned in the task is published as an artifact in the build or a previous stage and downloaded in the current job.

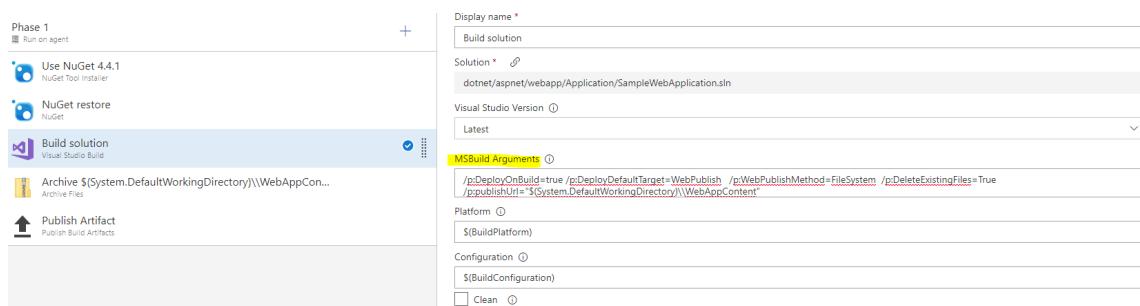
### Error: Publish using zip deploy option is not supported for msBuild package type

Web packages created via the MSBuild task (with default arguments) have a nested folder structure that can be deployed correctly only by Web Deploy. The publish-to-zip deployment option can't be used to deploy those packages. To convert the packaging structure, take these steps:

1. In the Build solution task, change the **MSBuild Arguments** to

```
/p:DeployOnBuild=true /p:DeployDefaultTarget=WebPublish /p:WebPublishMethod=FileSystem
/p:DeleteExistingFiles=True /p:publishUrl="$(System.DefaultWorkingDirectory)\\WebAppContent"
```

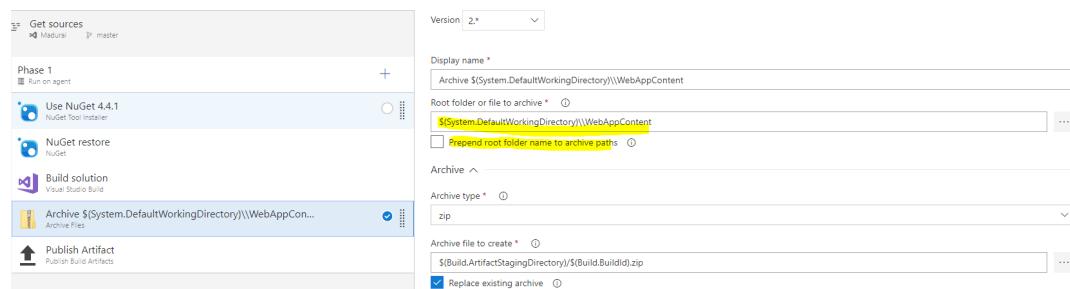
:



2. Add an Archive task and change the values as follows:

a. Change **Root folder or file to archive** to `$(System.DefaultWorkingDirectory)\\WebAppContent`.

b. Clear the **Prepend root folder name to archive paths** check box:



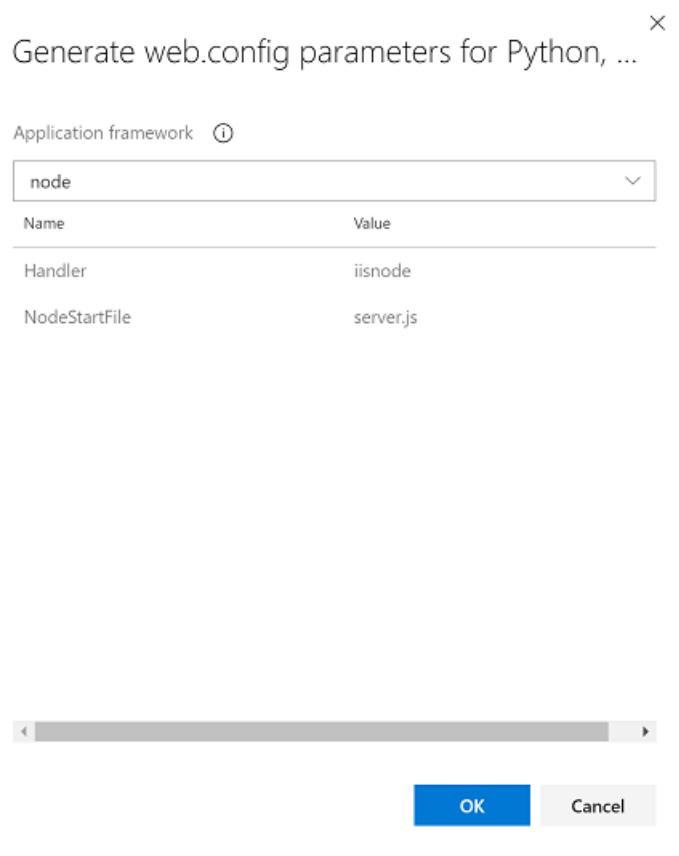
## Function app deployment on Windows succeeds but the app doesn't work

This problem could occur if a web.config file isn't present in your app. You can either add a web.config file to your source or automatically generate one by using the **Application and Configuration Settings** of the task.

1. Select the task and go to **Generate web.config parameters for Python, Node.js, Go and Java apps**:

The screenshot shows the 'Stage 1 Deployment process' configuration. On the left, under 'Agent job' (Run on agent), there is a task named 'Azure Function App Deploy'. A red warning message '(Some settings need attention)' is displayed next to it. To the right, there are several configuration sections: '\$(System.DefaultWorkingDirectory)/\*\*/\*.zip' (Deployment file), 'Application and Configuration Settings' (with 'Generate web.config parameters for Python, Node.js, Go and Java apps' and a 'More' button), 'App settings' (empty), 'Configuration settings' (empty), 'Control Options' (empty), and 'Output Variables' (empty).

2. Select the More button (...) under **Generate web.config parameters for Python, Node.js, Go and Java apps** to edit the parameters:



3. Select your application type in the **Application framework** list.

4. Select **OK**. Doing so will populate the web.config parameters required to generate the web.config file.

## FAQs

### How should I configure my service connection?

This task requires an [Azure Resource Manager service connection](#).

### How should I configure web job deployment with Application Insights?

When you're deploying to an App Service, if you have [Application Insights](#) configured and you've enabled

`Remove additional files at destination`, you also need to enable `Exclude files from the App_Data folder`.

Enabling this option keeps the Application Insights extension in a safe state. This step is required because the

Application Insights continuous WebJob is installed into the App\_Data folder.

### **How should I configure my agent if it's behind a proxy while I'm deploying to App Service?**

If your self-hosted agent requires a web proxy, you can inform the agent about the proxy during configuration. Doing so allows your agent to connect to Azure Pipelines or Azure DevOps Server through the proxy. [Learn more about running a self-hosted agent behind a web proxy](#).

### **I can't deploy to an internal App Service environment by using an Azure Resource Manager service connection and a Microsoft-hosted agent**

By design, a Microsoft-hosted agent won't work with an App Service environment. Instead, you need to configure a private agent on a virtual machine that's in the same virtual network as the App Service environment. Also, set a private DNS zone to enable communication between the resources.

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Azure Function App for Container task

11/2/2020 • 3 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task to deploy an Azure Function on Linux using a [custom image](#).

## Task Inputs

PARAMETERS	DESCRIPTION
<code>azureSubscription</code> Azure subscription	(Required) Name of an <a href="#">Azure Resource Manager service connection</a> .
<code>appName</code> App name	(Required) Name of the Function App for Containers.
<code>deployToSlotOrASE</code> Deploy to Slot or App Service Environment	(Optional) Set to true to deploy to an existing deployment slot or Azure App Service Environment. For both the targets, the task needs a Resource Group name. For the deployment slot option, the default is to deploy to the <b>production</b> slot, or you can specify any other existing slot name. If the deployment target is an Azure App Service environment, leave the slot name as <b>production</b> and just specify the Resource Group name. Default value: false
<code>resourceGroupName</code> Resource group	(Required if <code>deployToSlotOrASE</code> is true) Name of the Resource Group containing the Function App for Containers.
<code>slotName</code> Slot	(Required) Enter or select an existing slot other than the <b>production</b> slot. Default value: production
<code>imageName</code> Image name	(Required) Image to be used for deployment. Example: <code>myregistry.azurecr.io/nginx:latest</code>
<code>containerCommand</code> Startup command	(Optional) Startup command to be executed after deployment.
<code>appSettings</code> App settings	(Optional) Application settings to be entered using the syntax '-key value'. Values containing spaces should be enclosed in double quotes. Example: <code>-Port 5000 -RequestTimeout 5000 -WEBSITE_TIME_ZONE "Eastern Standard Time"</code>
<code>configurationStrings</code> Configuration settings	(Optional) Configuration strings to be entered using the syntax '-key value'. Values containing spaces should be enclosed in double quotes. Example: <code>-phpVersion 5.6 -linuxFxVersion: node 6.11</code>

## Example

This example deploys Azure Functions on Linux using containers:

```
variables:
 imageName: contoso.azurecr.io/azurefunctions-containers:${(build.buildId)}
 azureSubscription: Contoso
 # To ignore SSL error uncomment the following variable
 # VSTS_ARM_REST_IGNORE_SSL_ERRORS: true

steps:
- task: AzureFunctionAppContainer@1
 displayName: Azure Function App on Container deploy
 inputs:
 azureSubscription: $(azureSubscription)
 appName: functionappcontainers
 imageName: $(imageName)
```

## Troubleshooting

### Error: Could not fetch access token for Azure. Verify if the Service Principal used is valid and not expired.

The task uses the service principal in the service connection to authenticate with Azure. If the service principal has expired or doesn't have permissions to the App Service, the task fails with this error. Verify the validity of the service principal used and that it's present in the app registration. For more information, 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.

### SSL error

If you want to use a certificate in App Service, the certificate must be signed by a trusted certificate authority. If your web app gives you certificate validation errors, you're probably using a self-signed certificate. Set a variable named `VSTS_ARM_REST_IGNORE_SSL_ERRORS` to the value `true` in the build or release pipeline to resolve the error.

### A release hangs for long time and then fails

This problem could be the result of insufficient capacity in your App Service plan. To resolve this problem, you can scale up the App Service instance to increase available CPU, RAM, and disk space or try with a different App Service plan.

### 5xx error codes

If you're seeing a 5xx error, [check the status of your Azure service](#).

## FAQs

### How should I configure my service connection?

This task requires an [Azure Resource Manager service connection](#).

### How should I configure web job deployment with Application Insights?

When you're deploying to an App Service, if you have [Application Insights](#) configured and you've enabled

`Remove additional files at destination`, you also need to enable `Exclude files from the App_Data folder`.

Enabling this option keeps the Application Insights extension in a safe state. This step is required because the Application Insights continuous WebJob is installed into the App\_Data folder.

### How should I configure my agent if it's behind a proxy while I'm deploying to App Service?

If your self-hosted agent requires a web proxy, you can inform the agent about the proxy during configuration. Doing so allows your agent to connect to Azure Pipelines or Azure DevOps Server through the proxy. [Learn more about running a self-hosted agent behind a web proxy](#).

## Open Source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Azure Key Vault task

3/8/2021 • 4 minutes to read • [Edit Online](#)

## Azure Pipelines

### Overview

Use this task 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 Prerequisites:

- 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.

#### NOTE

If you're using a Microsoft-hosted agent, you must add the IP range of the Microsoft-hosted agent to your firewall. Get the weekly list of IP ranges from the [weekly JSON file](#), which is published every Wednesday. The new IP ranges become effective the following Monday. For more information, see [Microsoft-hosted agents](#). To find the IP ranges that are required for your Azure DevOps organization, learn how to [identify the possible IP ranges for Microsoft-hosted agents](#).

## YAML snippet

```
Azure Key Vault
Download Azure Key Vault secrets
- task: AzureKeyVault@1
 inputs:
 azureSubscription:
 keyVaultName:
 secretsFilter: '*'
 runAsPreJob: false # Azure DevOps Services only
```

## Arguments

PARAMETER	DESCRIPTION
<code>ConnectedServiceName</code> Azure Subscription	(Required) Select the service connection for the Azure subscription containing the Azure Key Vault instance, or create a new connection. <a href="#">Learn more</a>
<code>KeyVaultName</code> Key Vault	(Required) Select the name of the Azure Key Vault from which the secrets will be downloaded.
<code>SecretsFilter</code> Secrets filter	(Required) A comma-separated list of secret names to be downloaded. Default value: <code>*</code>
<code>RunAsPreJob</code> Make secrets available to whole job	(Required) Run the task before job execution begins. Exposes secrets to all tasks in the job, not just tasks that follow this one. Default value: <code>false</code>
PARAMETER	DESCRIPTION
<code>ConnectedServiceName</code> Azure Subscription	(Required) Select the service connection for the Azure subscription containing the Azure Key Vault instance, or create a new connection. <a href="#">Learn more</a>
<code>KeyVaultName</code> Key Vault	(Required) Select the name of the Azure Key Vault from which the secrets will be downloaded.
<code>SecretsFilter</code> Secrets filter	(Required) A comma-separated list of secret names to be downloaded. Default value: <code>*</code>

#### 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](mailto: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.

## FAQ

### I get a `forbidden` error on pipelines at the point of getting credentials from Azure Key Vault

This occurs if the required permissions are missing in the Azure key vault. To resolve the issue, [add an access policy with the correct permissions](#).

### Do I need an agent?

You need at least one [agent](#) to run your build or release.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your

pipeline. See [Environment variables](#) for more details.

**I can't connect with Key Vault from Azure DevOps.**

This happens when the Key Vault firewall isn't properly configured. Make sure that the agent pool and the Datacenter (TFS) can access the key vault. Ensure that the [agent IP ranges for Microsoft-hosted agents](#) are allow listed.

# Azure Monitor Alerts task

4/22/2020 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task 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
<code>ConnectedServiceName</code> Azure Subscription	(Required) Select the Azure Resource Manager subscription. <b>Note:</b> To configure new service connection, select the Azure subscription from the list and click <code>Authorize</code> . 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. Argument alias: <code>azureSubscription</code>
<code>ResourceGroupName</code> Resource Group	(Required) Select the Azure Resource Group that contains the Azure resource where you want to configure an alert.
<code>ResourceType</code> Resource Type	(Required) Select the Azure resource type. Options: <code>Microsoft.Insights/components, Microsoft.Web/sites, Microsoft.Storage/storageAccounts, Microsoft.Compute/virtualMachines</code> Default value: <code>Microsoft.Insights/components</code>
<code>ResourceName</code> Resource name	(Required) Select name of Azure resource where you want to configure an alert.
<code>AlertRules</code> Alert rules	(Required) List of Azure monitor alerts configured on selected Azure resource. To add or modify alerts, click on [...] button.
<code>NotifyServiceOwners</code> Subscription owners, contributors and readers	(Optional) Send email notification to everyone who has access to this resource group.

ARGUMENT	DESCRIPTION
<code>NotifyEmails</code> 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).

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Azure Database for MySQL Deployment task

3/6/2021 • 3 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task to run your scripts and make changes to your database in Azure Database for MySQL. Note that this is a 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:
 ConnectedServiceName: # Or alias 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
<code>ConnectedServiceName</code> Azure Subscription	(Required) This is needed to connect to your Azure account.  To configure a 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 set up an Azure service connection using the 'Add' or 'Manage' buttons.  Argument alias: <code>azureSubscription</code>
<code>ServerName</code> Host Name	(Required) The name of your Azure Database for MySQL server.  Example: <code>fabrikam.mysql.database.azure.com</code>  The server name is provided in the Azure portal on the 'Overview' blade of your Azure Database for MySQL server resource.  When you connect using MySQL Workbench, this is the same value that is used for 'Hostname' in 'Parameters'.

ARGUMENT	DESCRIPTION
<code>DatabaseName</code> Database Name	<p>(Optional) The name of the database. If specified, the task attempts to create the database if it does not exist.</p> <p>If not specified, ensure that the database is referenced in the supplied SQL file or inline SQL, where needed.</p> <p><b>Note:</b> MySQL database names are case-sensitive.</p>
<code>Sq1Username</code> Server Admin Login	<p>(Required) Azure Database for MySQL server supports native MySQL authentication.</p> <p>You can connect and authenticate to a server with the server's admin login.</p> <p>Example: <code>Mysqadmin</code></p> <p>When you connect using MySQL Workbench, this is the same value that is used for 'Username' in 'Parameters'.</p>
<code>Sq1Password</code> Password	<p>(Required) The password for the username supplied to the <code>Sq1Username</code> argument. If you don't recall the password, you can change the password in the Azure portal. It can be a variable defined in the pipeline.</p> <p>Example: <code>\$(password)</code></p> <p>Also, you may mark the variable type as 'secret' to secure it.</p>
<code>TaskNameSelector</code> Type	<p>(Optional) Accepted values:</p> <ul style="list-style-type: none"> <li>- <code>Sq1TaskFile</code> (default), for use with the <code>Sq1File</code> argument</li> <li>- <code>InlineSqlTask</code>, for use with the <code>Sq1Inline</code> argument.</li> </ul> <p><b>Note:</b> These values are case-sensitive.</p>
<code>Sq1File</code> MySQL Script	<p>(Required) Full path of the script file on the automation agent or on a UNC path that's accessible to the automation agent.</p> <p>Example: <code>\BudgetIT\DeployBuilds\script.sql</code></p> <p>You may also use predefined system variables, for example <code>\$(agent.releaseDirectory)</code>.</p> <p>The given path is expected to reference a SQL file that contains SQL statements.</p> <p><b>Note:</b> The MySQL client prefers Unix style paths so, from version 1.183.0, the task will convert Windows style paths to Unix style paths. Example: From <code>c:\foo\bar\myscript.sql</code> to <code>c:/foo/bar/myscript.sql</code>. When the task is used on Linux platforms, paths remain unchanged. There is no need to escape special characters in paths.</p>

ARGUMENT	DESCRIPTION
<code>SqlInline</code> Inline MySQL Script	(Required) Enter the MySQL script to execute on the selected database.  Example: <code>UPDATE my_table SET my_col = 'foo';</code>
<code>SqlAdditionalArguments</code> Additional MySQL client options	(Optional) Additional options supported by the MySQL client.  These options are applied when you execute the file on the database in Azure Database for MySQL.  Examples include: - <code>--comments</code> to strip comments sent from the client to the server. - <code>--quick</code> to prevent result caching. - <code>--xml</code> to output results as XML.  All available options are described in the MySQL client documentation.
<code>IpDetectionMethod</code> 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 a firewall exception for a range of possible IP addresses for the automation agent, or you can explicitly specify the range.  Accepted values:  - <code>AutoDetect</code> to auto-detect the automation agent's public IP address. - <code>IPAddressRange</code> to explicitly specify the IP address range to configure. Set the IP address range using the <code>StartIpAddress</code> and <code>EndIpAddress</code> parameters.  <b>Note:</b> These values are case-sensitive.
<code>StartIpAddress</code> Start IP Address	(Required) The starting IP address of the automation agent machine pool.  Example: <code>198.51.100.1</code>
<code>EndIpAddress</code> End IP Address	(Required) The ending IP address of the automation agent machine pool.  Example: <code>198.51.100.65</code>
<code>DeleteFirewallRule</code> Delete Rule After Task Ends	(Optional) If selected, the added exception for IP addresses of the automation agent is removed for the corresponding database in Azure Database for MySQL.

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Security and Compliance Assessment task

3/6/2021 • 2 minutes to read • [Edit Online](#)

Azure Policy allows you to assess and enforce resource compliance against defined IT policies. Use this task in a gate to identify, analyze and evaluate the security risks, and determine the mitigation measures required to reduce the risks.

## Demands

Can be used only as a [gate](#). This task is not supported in a build or release pipeline.

The screenshot shows the 'Post-deployment' stage configuration in the Azure DevOps Pipeline editor. The stage includes:

- Invoke Azure Function**: Invoke an Azure Function as a part of your pipeline.
- Invoke REST API**: Invoke a REST API as a part of your pipeline.
- Query Classic Azure Monitor Alerts**: Observe the configured classic Azure monitor rules for active alerts.
- Gates**: Define gates to evaluate after the deployment. A dropdown menu shows 'The delay before evaluation' set to 5 minutes.
- Deployment gates**: An 'Add' button to define deployment gates.

The **Security and compliance assessment** task is highlighted with a red box. Its description is: "Security and compliance assessment with Azure policies on resources that belong to the resource group and Azure subscription".

## YAML snippet

```
Check Azure Policy compliance
Security and compliance assessment for Azure Policy
- task: AzurePolicyCheckGate@0
 inputs:
 azureSubscription:
 #resourceGroupName: # Optional
 #resources: # Optional
```

### IMPORTANT

Service connections cannot be specified by variable.

## Arguments

PARAMETERS	DESCRIPTION
Azure subscription	(Required) Select the Azure Resource Manager subscription on which to enforce the policies.
Resource group	Select the Resource Group or specify a variable name.

PARAMETERS	DESCRIPTION
Resource name	Select the name of the Azure resources for which you want to check policy compliance.

# Azure PowerShell task

6/11/2021 • 3 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task to run a PowerShell script within an Azure environment. The Azure context is authenticated with the provided Azure Resource Manager service connection.

## YAML snippet

```
Azure PowerShell
Run a PowerShell script within an Azure environment
- task: AzurePowerShell@5
 inputs:
 #azureSubscription: Required. Name of Azure Resource Manager service connection
 #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' # Required. Options: latestVersion, otherVersion
 #preferredAzurePowerShellVersion: # Required when azurePowerShellVersion == OtherVersion
 #pwsh: true # Optional. If true, then will use PowerShell Core pwsh.exe
```

## Arguments

ARGUMENT	DESCRIPTION
<code>ConnectedServiceNameARM</code> Azure Subscription	(Required) name of an Azure Resource Manager service connection for authentication. Argument alias: <code>azureSubscription</code>
<code>ScriptType</code> Script Type	(Optional) Type of the script: filePath or inlineScript Default value: <code>FilePath</code>
<code>ScriptPath</code> Script Path	(Optional) Path of the script. Should be fully qualified path or relative to the default working directory.
<code>Inline</code> Inline Script	(Optional) Enter the script to execute. Default value: # You can write your Azure PowerShell scripts inline here.
<code>ScriptArguments</code> Script Arguments	(Optional) Additional parameters to pass to PowerShell. Can be either ordinal or named parameters. Not applicable for inline script option.
<code>errorActionPreference</code> ErrorActionPreference	(Optional) Select the value of the ErrorActionPreference variable for executing the script. Default value: <code>stop</code>

ARGUMENT	DESCRIPTION
<code>FailOnStandardError</code> 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. Default value: <code>false</code>
<code>TargetAzurePs</code> Azure PowerShell Version	(Required) In case of Microsoft-hosted agents, the supported Azure PowerShell version. To pick the latest version available on the agent, select <code>Latest installed version</code> . For self-hosted agents, you can specify preferred version of Azure PowerShell using "Specify version" Default value: <code>OtherVersion</code> Argument alias: <code>azurePowerShellVersion</code>
<code>CustomTargetAzurePs</code> <code>preferredAzurePowerShellVersion</code>	(Required when TargetAzurePs = OtherVersion) Preferred Azure PowerShell Version needs to be a proper semantic version. For example, 1.2.3. Regex like 2.*,2.3.* is not supported. Argument alias: <code>preferredAzurePowerShellVersion</code>
<code>pwsh</code> Use PowerShell Core	(Optional) If this is true, on Windows, the task will use pwsh.exe from your PATH instead of powershell.exe.
<code>workingDirectory</code> Working Directory	(Optional) Working directory where the script is run.

## Samples

```
- task: AzurePowerShell@5
 inputs:
 azureSubscription: my-arm-service-connection
 scriptType: filePath
 scriptPath: $(Build.SourcesDirectory)\myscript.ps1
 scriptArguments:
 -Arg1 val1 `
 -Arg2 val2 `
 -Arg3 val3
 azurePowerShellVersion: latestVersion
 pwsh: true
```

## Troubleshooting

### Script worked locally, but failed in the pipeline

This typically occurs when the service connection used in the pipeline has insufficient permissions to run the script. Locally, the script runs with your credentials and would succeed as you may have the required access.

To resolve this issue, ensure the service principle/ authentication credentials have the required permissions. For more information, see [Use Role-Based Access Control to manage access to your Azure subscription resources](#).

### Error: Could not find the modules: '<module name>' with Version: '<version>'. If the module was recently installed, retry after restarting the Azure Pipelines task agent

Azure PowerShell task uses Azure/AzureRM/Az PowerShell Module to interact with Azure Subscription. This issue occurs when the PowerShell module is not available on the Hosted Agent. Hence, for a particular task version, *Preferred Azure PowerShell version* must be specified in the **Azure PowerShell version options** from the list of available versions. The installed software can be found in the [Software](#) table in [Microsoft-hosted](#)

agents.

## Service Connection Issues

To troubleshoot issues related to service connections, see [Service Connection troubleshooting](#).

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Azure Resource Group Deployment task

6/9/2021 • 9 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task 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 and manage virtual machines
- 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, Validate
 #enableDeploymentPrerequisites: 'None' # Optional. Options: none, configureVMwithWinRM, configureVMWithDGAgent
 #teamServicesConnection: # Required when enableDeploymentPrerequisites == ConfigureVMWithDGAgent
 #teamProject: # Required when enableDeploymentPrerequisites == ConfigureVMWithDGAgent
 #deploymentGroupName: # Required when enableDeploymentPrerequisites == ConfigureVMWithDGAgent
 #copyAzureVMTags: true # Optional
 #runAgentServiceAsUser: # Optional
 #userName: # Required when enableDeploymentPrerequisites == ConfigureVMWithDGAgent && RunAgentServiceAsUser == True
 #password: # Optional
 #outputVariable: # Optional
 #deploymentName: # Optional
 #deploymentOutputs: # Optional
 #addSpnToEnvironment: false # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<code>ConnectedServiceName</code> Azure subscription	(Required) Select the Azure Resource Manager subscription for the deployment. Argument aliases: <code>azureSubscription</code>
<code>action</code> Action	(Required) Action to be performed on the Azure resources or resource group. Default value: <code>Create Or Update Resource Group</code>
<code>resourceGroupName</code> Resource group	(Required) Provide the name of a resource group.

ARGUMENT	DESCRIPTION
<code>location</code> Location	(Required) Location for deploying the resource group. If the resource group already exists in the subscription, then this value will be ignored.
<code>templateLocation</code> Template location	(Required) Select either <b>Linked artifact</b> or <b>URL of the file</b> . Default value: <code>Linked artifact</code>
<code>csmFileLink</code> Template link	(Required) Specify the URL of the template file. <b>Example:</b> <a href="https://raw.githubusercontent.com/Azure/">https://raw.githubusercontent.com/Azure/...</a> 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. <b>Example:</b> <blob_storage_url>/template.json?. To upload a template file (or a linked template) to a storage account and generate a SAS token, you could use <a href="#">Azure file copy</a> task or follow the steps using <a href="#">PowerShell</a> or <a href="#">Azure CLI</a> . 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 <a href="#">this</a> to enable CORS.
<code>csmParametersFileLink</code> Template parameters link	(Optional) Specify the URL of the parameters file. <b>Example:</b> <a href="https://raw.githubusercontent.com/Azure/">https://raw.githubusercontent.com/Azure/...</a> 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. <b>Example:</b> <blob_storage_url>/template.json?. 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.
<code>csmFile</code> Template	(Required) Specify the path or a pattern pointing to the Azure Resource Manager template. For more information about the templates see <a href="https://aka.ms/azuratemplates">https://aka.ms/azuratemplates</a> . To get started immediately use template <a href="https://aka.ms/sampletemplate">https://aka.ms/sampletemplate</a> .
<code>csmParametersFile</code> Template parameters	(Optional) Specify the path or a pattern pointing for the parameters file for the Azure Resource Manager template.

ARGUMENT	DESCRIPTION
<p><code>overrideParameters</code> Override template parameters</p>	<p>(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.</p> <p>Example: -storageName fabrikam -adminUsername \$(vmusername) -adminPassword \$(password) - azureKeyVaultName \$(fabrikamFibre).</p> <p>If the parameter value you're using has multiple words, enclose them in quotes, even if you're passing them using variables.</p> <p>For example, -name "parameter value" -name2 "\$(var)".</p> <p>To override object type parameters use stringified JSON objects.</p> <p>For example, -options ["option1"] -map {"key1": "value1" }.</p>
<p><code>deploymentMode</code> Deployment mode</p>	<p>(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. Note that this mode always creates a resource group, even if no resources are deployed.</p> <p>Default value: <code>Incremental</code></p>
<p><code>enableDeploymentPrerequisites</code> Enable prerequisites</p>	<p>(Optional) These options would be applicable only when the Resource group contains virtual machines. Choosing Deployment Group option would configure Deployment Group agent on each of the virtual machines. 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> <p>Default value: <code>None</code></p>
<p><code>deploymentGroupEndpoint</code> Azure Pipelines service connection</p>	<p>(Required) Specify the service connection to connect to an Azure DevOps organization or collection for agent registration.</p> <p>You can create a service connection using <b>+ New</b>, and select <b>Token-based authentication</b>. You need a <a href="#">personal access token (PAT)</a> to set up a service connection.</p> <p>Click <b>Manage</b> to update the service connection details.</p> <p>Argument aliases: <code>teamServicesConnection</code></p>
<p><code>project</code> Team project</p>	<p>(Required) Specify the Team project which has the Deployment Group defined in it.</p> <p>Argument aliases: <code>teamProject</code></p>
<p><code>deploymentGroupName</code> Deployment Group</p>	<p>(Required) Specify the Deployment Group against which the Agent(s) will be registered. For more guidance, refer to <a href="#">Deployment Groups</a>.</p>

ARGUMENT	DESCRIPTION
<code>copyAzureVMTags</code> Copy Azure VM tags to agents	(Optional) Choose if the tags configured on the Azure VM need to be copied to the corresponding Deployment Group agent. By default all Azure tags will be copied following the format <b>Key: Value</b> . <b>Example:</b> An Azure Tag "Role : Web" would be copied as-is to the Agent machine. For more information on how tag Azure resources refer to the <a href="#">link</a> .
<code>runAgentServiceAsUser</code> Run agent service as a user	(Optional) Decide whether to run the agent service as a user other than the default. The default user is <b>NT AUTHORITY\\SYSTEM</b> in Windows and <b>root</b> in Linux.
<code>userName</code> User name	(Required) The username to run the agent service on the virtual machines. For domain users, please enter values as <b>domain\\username</b> or <b>username@domain.com</b> . For local users, please enter just the user name. It is assumed that the same domain user or a local user with the same name, respectively, is present on all the virtual machines in the resource group.
<code>password</code> Password	The password for the user to run the agent service on the Windows VMs. It is assumed that the password is same for the specified user on all the VMs. It can accept variable defined in build or release pipelines as <b>\$(passwordVariable)</b> . You may mark variable as <b>secret</b> to secure it. For linux VMs, a password is not required and will be ignored.
<code>outputVariable</code> VM details for WinRM	(Optional) Provide a name for the variable for the resource group. The variable can be used as <b>\$(variableName)</b> 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 <b>Create</b> , <b>Update</b> or <b>Select</b> , and required when an existing resource group is selected.
<code>deploymentName</code> Deployment name	(Optional) Specifies the name of the resource group deployment to create
<code>deploymentOutputs</code> 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 <b>ConvertFrom-Json</b> PowerShell cmdlet to parse the JSON object and access the individual output values.
<code>addSpnToEnvironment</code> Access service principal details in override parameters	Adds service principal ID and key of the Azure endpoint you chose to the script's execution environment. You can use these variables: <b>\$servicePrincipalId</b> and <b>\$servicePrincipalKey</b> in your override parameters like -key <b>\$servicePrincipalKey</b>

## Troubleshooting

**Error: Internal Server Error**

These issues are mostly transient in nature. There are multiple reasons why it could be happening:

- One of the Azure services you're trying to deploy is undergoing maintenance in the region you're trying to deploy to. Keep an eye out on <https://status.azure.com/> to check downtimes of Azure Services.
- Azure Pipelines service itself is going through maintenance. Keep an eye out on <https://status.dev.azure.com/> for downtimes.

However, we've seen some instances where this is due to an error in the ARM template, such as the Azure service you're trying to deploy doesn't support the region you've chosen for the resource.

### Error: Timeout

Timeout issues could be coming from two places:

- Azure Pipelines Agent
- Portal Deployment

You can identify if the timeout is from portal, by checking for the portal deployment link that'll be in the task logs. If there's no link, this is likely due to Azure Pipelines agent. If there's a link, follow the link to see if there's a timeout that has happened in the portal deployment.

### Error: CORS rules to be enabled while overriding parameters

If the template file is being referred from a BLOB, while overriding parameters in the pipeline, you might see the following warning message:

```
Warning: Failed to download the file from template path.
```

This feature requires the CORS rules to be enabled at the source. If templates are in Azure storage blob, see [Cross-origin resource sharing support](#) to enable CORS.

Besides enabling CORS, ensure that the SAS token specified in the link of the template is "sr-t-sco". This token is required for you to download the file and proceed.

#### Azure Pipelines Agent

If the issue is coming from Azure Pipelines agent, you can increase the timeout by setting `timeoutInMinutes` as key in the YAML to 0. For more information, see [Specify jobs in your pipeline](#).

#### Portal Deployment

Check out this doc on how to identify if the error came from the Azure portal: [View deployment history with Azure Resource Manager](#).

In case of portal deployment, try setting "timeoutInMinutes" in the ARM template to "0". If not specified, the value assumed is 60 minutes. 0 makes sure the deployment will run for as long as it can to succeed.

This could also be happening because of transient issues in the system. Keep an eye on <https://status.dev.azure.com/> to check if there's a downtime in Azure Pipelines service.

### Error: Azure Resource Manager (ARM) template failed validation

This issue happens mostly because of an invalid parameter in the ARM template, such as an unsupported SKU or region. If the validation fails, check the error message. It should point you to the resource and parameter that's invalid.

This issue also might occur because of multiline strings. Currently, the Azure Resource Group Deployment task doesn't support multiline strings in an ARM template or parameter JSON file.

In addition, refer to this article regarding structure and syntax of ARM Templates: [Understand the structure and syntax of ARM templates](#).

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Azure Spring Cloud task

5/12/2021 • 3 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task to deploy applications to [Azure Spring Cloud](#) and to manage Azure Spring Cloud [deployments](#).

## Arguments

### NOTE

Some arguments are only applicable for certain settings of the `Action` argument. The Action column below specifies the pertinent actions for each argument. Any argument listed as Required is only required for the pertinent Action(s).

Argument	Action	Description
<code>Action</code> Action	All	(Required) The action to be performed by this task. One of: <code>Deploy</code> , <code>Set Production</code> , <code>Delete Staging Deployment</code> Default value: <code>Deploy</code>
<code>ConnectedServiceName</code> Azure Subscription	All	(Required) The name of the <a href="#">Azure Resource Manager service connection</a> . Argument alias: <code>azureSubscription</code>
<code>AzureSpringCloud</code> Azure Spring Cloud	All	(Required) The name or resource ID of the Azure Spring Cloud instance.
<code>AppName</code> App Name	All	(Required) The name of the Azure Spring Cloud app to deploy. The app must exist prior to task execution.
<code>UseStagingDeployment</code> Use Staging Deployment.	Deploy Set Production	(Required) If set to <code>true</code> , apply the task to whichever <code>deployment</code> is set as the staging deployment at time of execution. If omitted, the <code>DeploymentName</code> parameter must be set. Default value: <code>true</code>
<code>DeploymentName</code> Deployment Name	Deploy Set production	(Required if <code>UseStagingDeployment</code> is <code>false</code> ) The name of the <code>deployment</code> to which the action will apply. If not using blue-green deployments, set this field to <code>default</code> .
<code>CreateNewDeployment</code> Create new deployment	Deploy	(Optional) If set to <code>true</code> and the deployment specified by <code>DeploymentName</code> does not exist at execution time, it will be created. Default value: <code>false</code>

ARGUMENT	ACTION	DESCRIPTION
<code>Package</code> Package or folder	Deploy	(Required) The file path to the package containing the application to be deployed (.jar file for Java, .zip for .NET Core) or to a folder containing the application source to be built. <a href="#">Build variables</a> or <a href="#">release variables</a> are supported. Default value: <code>\$(System.DefaultWorkingDirectory)/**/*.jar</code>
<code>RuntimeVersion</code> Runtime Version	Deploy	(Optional) The runtime stack for the application. One of: <code>Java_8</code> , <code>Java_11</code> , <code>NetCore_31</code> , Default value: <code>Java_11</code>
<code>EnvironmentVariables</code> Environment Variables	Deploy	(Optional) Environment variables to be entered using the syntax '-key value'. Values containing spaces should be enclosed in double quotes. Example: <code>-CUSTOMER_NAME Contoso - WEBSITE_TIME_ZONE "Eastern Standard Time"</code>
<code>JvmOptions</code> JVM Options	Deploy	(Optional) A string containing JVM Options. Example: <code>-XX:+UseG1GC -XX:MaxRAMPercentage=70 - Dazure.keyvault.enabled=true - Dazure.keyvault.uri=https://myvault.vault.azure.</code>
<code>DotNetCoreMainEntryPath</code> .NET Core entry path	Deploy	(Optional) A string containing the path to the .NET executable relative to zip root.
<code>Version</code> Version	Deploy	(Optional) The deployment version. If not set, the version is left unchanged.

## Examples

The following examples demonstrate common usage scenarios. For more information, see [Automate application deployments to Azure Spring Cloud](#).

### Deleting a staging deployment

The "Delete Staging Deployment" action allows you to delete the deployment not receiving production traffic. This frees up resources used by that deployment and makes room for a new staging deployment:

```
variables:
 azureSubscription: Contoso

steps:
- task: AzureSpringCloud@0
 continueOnError: true # Don't fail the pipeline if a staging deployment doesn't already exist.
 inputs:
 continueOnError: true
 inputs:
 azureSubscription: $(azureSubscription)
 Action: 'Delete Staging Deployment'
 AppName: customer-api
 AzureSpringCloud: contoso-dev-az-spr-cld
```

## Deploying

### To production

The following example deploys to the default production deployment in Azure Spring Cloud. This is the only possible deployment scenario when using the Basic SKU:

```
variables:
 azureSubscription: Contoso

steps:
- task: AzureSpringCloud@0
 inputs:
 azureSubscription: $(azureSubscription)
 Action: 'Deploy'
 AzureSpringCloud: contoso-dev-az-spr-cld
 AppName: customer-api
 UseStagingDeployment: false
 DeploymentName: default
 Package: '$(System.DefaultWorkingDirectory)/**/*customer-api*.jar'
```

### Blue-green

The following example deploys to a pre-existing staging deployment. This deployment will not receive production traffic until it is set as a production deployment.

```
variables:
 azureSubscription: Contoso

steps:
- task: AzureSpringCloud@0
 inputs:
 azureSubscription: $(azureSubscription)
 Action: 'Deploy'
 AzureSpringCloud: contoso-dev-az-spr-cld
 AppName: customer-api
 UseStagingDeployment: true
 Package: '$(System.DefaultWorkingDirectory)/**/*customer-api*.jar'
```

For more on blue-green deployments, including an alternative approach, see [Blue-green deployment strategies](#).

### Setting production deployment

The following example will set the current staging deployment as production, effectively swapping which deployment will receive production traffic.

```
variables:
 azureSubscription: Contoso

steps:
- task: AzureSpringCloud@0
 inputs:
 azureSubscription: $(azureSubscription)
 Action: 'Set Production'
 AzureSpringCloud: contoso-dev-az-spr-cld
 AppName: customer-api
 UseStagingDeployment: true
```

# Azure SQL Database Deployment task

3/6/2021 • 5 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task to deploy to Azure SQL DB using a DACPAC or run scripts using SQLCMD.

### IMPORTANT

This task is supported only in a Windows environment. If you are trying to use Azure Active Directory (Azure AD) integrated authentication, you must create a private agent. Azure AD integrated authentication is not supported for hosted agents.

## YAML snippet

```
Azure SQL Database deployment
Deploy an Azure SQL Database 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
 #authenticationType: 'server' # Options: server, aadAuthenticationPassword, aadAuthenticationIntegrated,
connectionString, servicePrincipal
 #serverName: # Required when authenticationType == Server || AuthenticationType ==
AadAuthenticationPassword || AuthenticationType == AadAuthenticationIntegrated || AuthenticationType ==
servicePrincipal
 #databaseName: # Required when authenticationType == Server || AuthenticationType ==
AadAuthenticationPassword || AuthenticationType == AadAuthenticationIntegrated || AuthenticationType ==
servicePrincipal
 #sqlUsername: # Required when authenticationType == Server
 #sqlPassword: # Required when authenticationType == Server
 #aadSqlUsername: # Required when authenticationType == AadAuthenticationPassword
 #aadSqlPassword: # Required when authenticationType == AadAuthenticationPassword
 #connectionString: # Required when authenticationType == ConnectionString
 #deployType: 'DacpacTask' # Options: dacpacTask, sqlTask, inlineSqlTask
 #deploymentAction: 'Publish' # Required when deployType == DacpacTask. Options: publish, extract,
export, import, script, driftReport, deployReport
 #dacpacFile: # Required when deploymentAction == Publish || DeploymentAction == Script ||
DeploymentAction == DeployReport
 #bacpacFile: # Required when deploymentAction == Import
 #sqlFile: # Required when deployType == SqlTask
 #sqlInline: # Required when deployType == 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
<code>ConnectedServiceNameSelector</code> Azure Connection Type	(Optional) Argument alias: <code>azureConnectionType</code> Default value: <code>ConnectedServiceNameARM</code>
<code>ConnectedServiceName</code> Azure Classic Subscription	(Required) Target Azure Classic subscription for deploying SQL files Argument alias: <code>azureClassicSubscription</code>
<code>ConnectedServiceNameARM</code> Azure Subscription	(Required) Target Azure Resource Manager subscription for deploying SQL files Argument alias: <code>azureSubscription</code>
<code>AuthenticationType</code> Authentication Type	(Required) Type of database authentication, can be SQL Server Authentication, Active Directory - Integrated, Active Directory - Password, Connection String, or Service Principal. Integrated authentication means that the agent will access the database using its current Active Directory account context. Default value: <code>server</code>
<code>ServerName</code> Azure SQL Server	(Required except when Authentication Type is Connection String) Azure SQL Server name, like <code>Fabrikam.database.windows.net,1433</code> or <code>Fabrikam.database.windows.net.</code>
<code>DatabaseName</code> Database	(Required) Name of the Azure SQL Database, where the files will be deployed.
<code>SqlUsername</code> Login	(Required when Authentication Type is SQL Server Authentication or Active Directory - Password) Specify the Azure SQL Server administrator login or Active Directory user name.
<code>SqlPassword</code> Password	(Required when Authentication Type is SQL Server Authentication or Active Directory - Password) Password for the Azure SQL Server administrator or Active Directory user. It can accept variables defined in build/release pipelines as ' <code>\$(passwordVariable)</code> '. You may mark the variable type as 'secret' to secure it.
<code>ConnectionString</code> Connection String	(Required when Authentication Type is Connection String) The connection string, including authentication information, for the Azure SQL Server.
<code>TaskNameSelector</code> Deploy Type	(Optional) Specify the type of artifact, SQL DACPAC file, SQL Script file, or Inline SQL Script. Argument alias: <code>deployType</code> Default value: <code>DacpacTask</code>
<code>DeploymentAction</code> Action	(Required) Choose one of the SQL Actions from the list. <ul style="list-style-type: none"><li>• Publish, Extract, Export, Import, Script, Drift Report, Deploy Report.</li></ul> For more details refer <a href="#">link</a> Default value: <code>Publish</code>

ARGUMENT	DESCRIPTION
DacpacFile DACPAC File	(Required when Deploy Type is SQL DACPAC file) Location of the DACPAC file on the automation agent or on a UNC path accessible to the automation agent like, <code>\BudgetIT\Web\Deploy\FabrikamDB.dacpac</code> . Predefined system variables like, <code>\$(agent.releaseDirectory)</code> can also be used here.
BacpacFile BACPAC File	(Required) Location of the BACPAC file on the automation agent or on a UNC path accessible to the automation agent like <code>\BudgetIT\Web\Deploy\FabrikamDB.bacpac</code> . Predefined system variables like, <code>\$(agent.releaseDirectory)</code> can also be used here
SqlFile SQL Script	(Required when Deploy Type is SQL Script file) Location of the SQL script file on the automation agent or on a UNC path accessible to the automation agent like, <code>\BudgetIT\Web\Deploy\FabrikamDB.sql</code> . Predefined system variables like, <code>\$(agent.releaseDirectory)</code> can also be used here.
SqlInline Inline SQL Script	(Required when Deploy Type is Inline SQL Script) Enter the SQL script to execute on the Database selected above.
PublishProfile 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 agent machine or a UNC share. If the publish profile contains secrets like credentials, upload it to the <a href="#">secure files</a> library where it is securely stored with encryption. Then use the <a href="#">Download secure file</a> task at the start of your pipeline to download it to the agent machine when the pipeline runs and delete it when the pipeline is complete. Predefined system variables like <code>\$(agent.buildDirectory)</code> or <code>\$(agent.releaseDirectory)</code> can also be used in this field.
AdditionalArguments 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, <code>/p:IgnoreAnsiNulls=True /p:IgnoreComments=True</code> . These arguments will override the settings in the publish profile XML file (if provided).
Sq1AdditionalArguments 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, <code>-ConnectionTimeout 100 -OutputSqlErrors</code>
InlineAdditionalArguments 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, <code>-ConnectionTimeout 100 -OutputSqlErrors</code>

ARGUMENT	DESCRIPTION
<input type="checkbox"/> <b>IpDetectionMethod</b> 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. Default value: <input type="checkbox"/> AutoDetect
<input type="checkbox"/> <b>StartIpAddress</b> Start IP Address	(Required) The starting IP Address of the automation agent machine pool like 196.21.30.50.
<input type="checkbox"/> <b>EndIpAddress</b> End IP Address	(Required) The ending IP Address of the automation agent machine pool like 196.21.30.65.
<input type="checkbox"/> <b>DeleteFirewallRule</b> 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.

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Azure Web App task

3/6/2021 • 8 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task to deploy web applications to Azure App service.

## Arguments

PARAMETERS	DESCRIPTION
<code>azureSubscription</code> Azure subscription	(Required) Name of the <a href="#">Azure Resource Manager service connection</a>
<code>appType</code> App type	(Optional) Web App type
<code>appName</code> App name	(Required) Name of an existing App Service
<code>deployToSlotOrASE</code> Deploy to Slot or App Service Environment	(Optional) Select the option to deploy to an existing deployment slot or Azure App Service Environment. For both the targets, the task needs resource group name. In case the deployment target is a slot, by default the deployment is done to the production slot. Any other existing slot name can also be provided. In case the deployment target is an Azure App Service environment, specify the resource group name. Default value: false
<code>resourceGroupName</code> Resource group	(Required if <code>deployToSlotOrASE == true</code> ) Name of the resource group
<code>slotName</code> Slot	(Required if <code>deployToSlotOrASE == true</code> ) Name of the slot Default value: production
<code>package</code> Package or folder	(Required) File path to the package, or to a folder containing App Service contents generated by MSBuild, or to a compressed zip or war file. <a href="#">Build variables</a> or <a href="#">release variables</a> and wildcards are supported. For example, <code>\$(System.DefaultWorkingDirectory)/**/*.zip</code> or <code>\$(System.DefaultWorkingDirectory)/**/*.war</code>
<code>runtimeStack</code> Runtime stack	(Optional) Web App on Linux offers two different options to publish your application, one is custom image deployment (Web App for Containers) and the other is app deployment with a built-in platform image (Web App on Linux). You will see this parameter only when you select <b>Linux Web App</b> in the app type selection option in the task.
<code>startUpCommand</code> Startup command	(Optional; Relevant if <code>appType == webAppLinux</code> ) Startup command to be run post deployment

PARAMETERS	DESCRIPTION
<code>customWebConfig</code> Generate web.config parameters for Python, Node.js, Go and Java apps	(Optional) A standard web.config will be generated and deployed to Azure App Service if the application does not have one. The values in web.config can be edited and vary based on the application framework. For example for node.js application, web.config will have startup file and iis_node module values. This edit feature is only for the generated web.config. <a href="#">Learn more</a>
<code>appSettings</code> App settings	(Optional) Application settings to be entered using the syntax '-key value'. Values containing spaces should be enclosed in double quotes. Example: -Port 5000 -RequestTimeout 5000 -WEBSITE_TIME_ZONE "Eastern Standard Time"
<code>configurationStrings</code> Configuration settings	(Optional) Configuration strings to be entered using the syntax '-key value'. Values containing spaces should be enclosed in double quotes. Example: -phpVersion 5.6 -linuxFxVersion: node 6.11
<code>deploymentMethod</code> Deployment method	(Required) <a href="#">Deployment method</a> for the app. Acceptable values are <code>auto</code> , <code>zipDeploy</code> , <code>runFromPackage</code> Default value: <code>auto</code>

Following is an example YAML snippet to deploy web application to the Azure Web App service running on Windows.

## Example

```
variables:
 azureSubscription: Contoso
 # To ignore SSL error uncomment the below variable
 # VSTS_ARM_REST_IGNORE_SSL_ERRORS: true

steps:
 - task: AzureWebApp@1
 displayName: Azure Web App Deploy
 inputs:
 azureSubscription: $(azureSubscription)
 appName: samplewebapp
 package: $(System.DefaultWorkingDirectory)/**/*.zip
```

To deploy Web App on Linux, add the `appType` parameter and set it to `appType: webAppLinux`.

To specify the deployment method as Zip Deploy, add the parameter `deploymentMethod: zipDeploy`. Other supported value for this parameter is `runFromPackage`. If not mentioned, `auto` is taken as the default value.

## Deployment methods

Several deployment methods are available in this task. `Auto` is the default option.

To change the deployment option in designer task, 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 Azure Pipelines agent, the task chooses a suitable deployment

technology. The different deployment technologies used by the task are:

- Kudu REST APIs
- Zip Deploy
- RunFromPackage

By default the task tries to select the appropriate deployment technology given the input package, app service type and agent OS.

- When the App Service type is Web App on Linux App, use Zip Deploy
- If War file is provided, use War Deploy
- If Jar file is provided, use Run From package
- For all others, use Run From Zip (via Zip Deploy)

On non-Windows agent (for any App service type), the task relies on [Kudu REST APIs](#) to deploy the Web App.

### **Kudu REST APIs**

Works on Windows as well as Linux automation agent when the target is Web App on Windows or Web App on Linux (built-in source) or Function App. The task uses Kudu to copy files to the Azure App service.

### **Zip Deploy**

Creates a .zip deployment package of the chosen Package or folder and deploys the file contents to the wwwroot folder of the App Service name function app in Azure. This option overwrites all existing contents in the wwwroot folder. For more information, see [Zip deployment for Azure Functions](#).

### **RunFromPackage**

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. With this option, files in the wwwroot folder become read-only. For more information, see [Run your Azure Functions from a package file](#).

## Troubleshooting

### **Error: Could not fetch access token for Azure. Verify if the Service Principal used is valid and not expired.**

The task uses the service principal in the service connection to authenticate with Azure. If the service principal has expired or doesn't have permissions to the App Service, the task fails with this error. Verify the validity of the service principal used and that it's present in the app registration. For more information, 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.

### **SSL error**

If you want to use a certificate in App Service, the certificate must be signed by a trusted certificate authority. If your web app gives you certificate validation errors, you're probably using a self-signed certificate. Set a variable named `VSTS_ARM_REST_IGNORE_SSL_ERRORS` to the value `true` in the build or release pipeline to resolve the error.

### **A release hangs for long time and then fails**

This problem could be the result of insufficient capacity in your App Service plan. To resolve this problem, you can scale up the App Service instance to increase available CPU, RAM, and disk space or try with a different App Service plan.

### **5xx error codes**

If you're seeing a 5xx error, [check the status of your Azure service](#).

### **Error: No package found with specified pattern**

Check if the package mentioned in the task is published as an artifact in the build or a previous stage and downloaded in the current job.

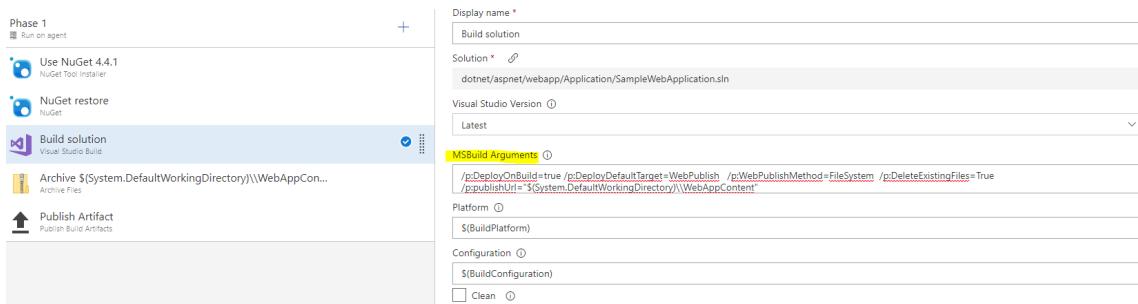
## Error: Publish using zip deploy option is not supported for msBuild package type

Web packages created via the MSBuild task (with default arguments) have a nested folder structure that can be deployed correctly only by Web Deploy. The publish-to-zip deployment option can't be used to deploy those packages. To convert the packaging structure, take these steps:

1. In the Build solution task, change the **MSBuild Arguments** to

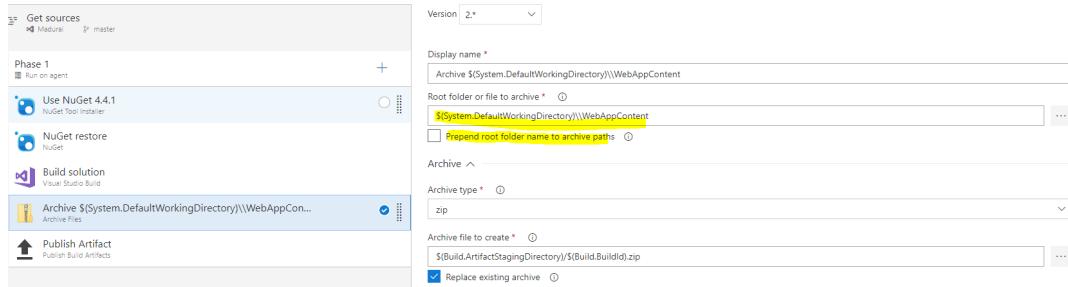
```
/p:DeployOnBuild=true /p:DeployDefaultTarget=WebPublish /p:WebPublishMethod=FileSystem
/p:DeleteExistingFiles=True /p:publishUrl="$(System.DefaultWorkingDirectory)\\WebAppContent"
```

:



2. Add an Archive task and change the values as follows:

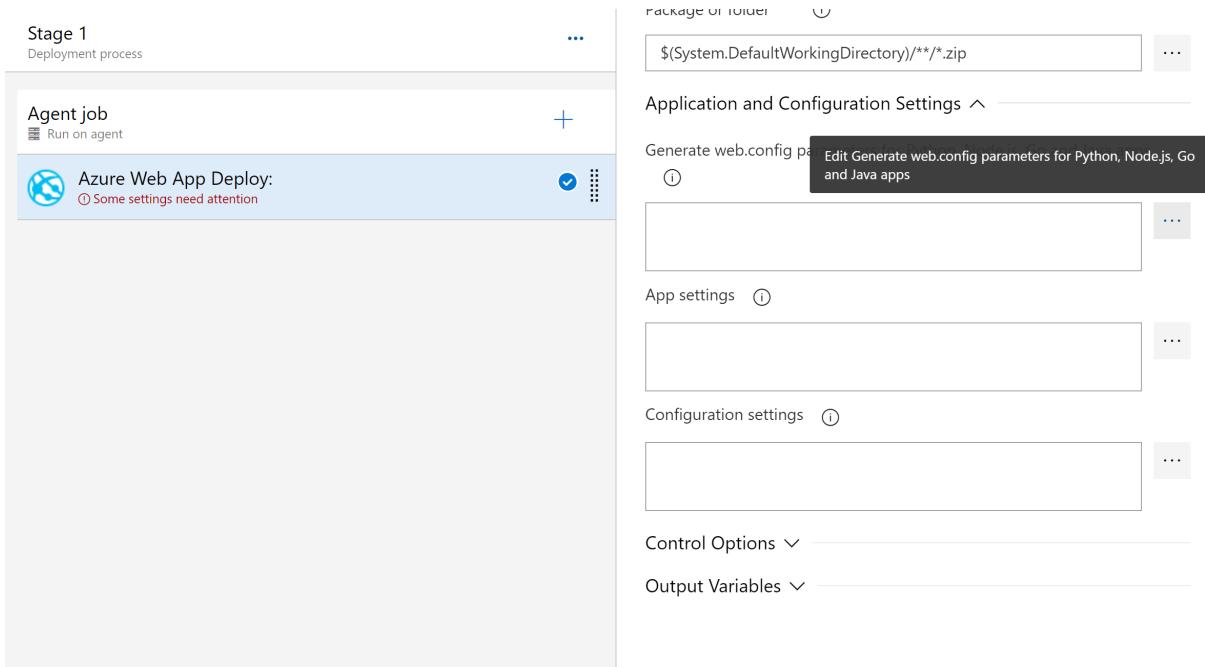
- a. Change **Root folder or file to archive** to `$(System.DefaultWorkingDirectory)\\WebAppContent`.
- b. Clear the **Prepend root folder name to archive paths** check box:



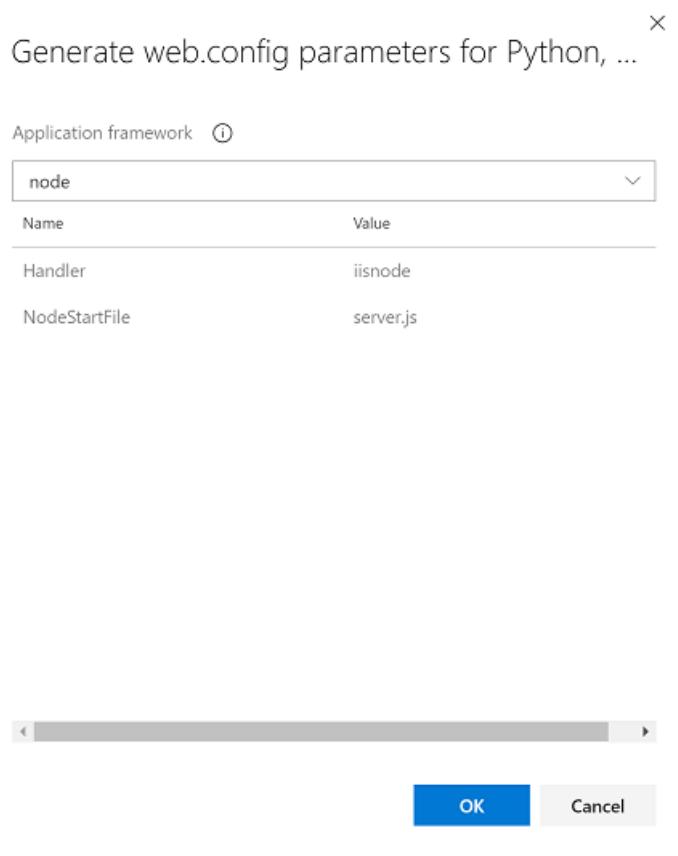
## Web app deployment on Windows is successful but the app is not working

This may be because web.config is not present in your app. You can either add a web.config file to your source or auto-generate one using the Application and Configuration Settings of the task.

- Click on the task and go to Generate web.config parameters for Python, Node.js, Go and Java apps.



- Click on the more button Generate web.config parameters for Python, Node.js, Go and Java apps to edit the parameters.



- Select your application type from the drop down.
- Click on OK. This will populate web.config parameters required to generate web.config.

### **Web app deployment on App Service Environment (ASE) is not working**

- Ensure that the Azure DevOps build agent is on the same VNET (subnet can be different) as the Internal Load Balancer (ILB) of ASE. This will enable the agent to pull code from Azure DevOps and deploy to ASE.
- If you are using Azure DevOps, the agent neednt be accessible from internet but needs only outbound access to connect to Azure DevOps Service.
- If you are using TFS/Azure DevOps server deployed in a Virtual Network, the agent can be completely isolated.

- Build agent must be configured with the DNS configuration of the Web App it needs to deploy to. Since the private resources in the Virtual Network don't have entries in Azure DNS, this needs to be added to the hosts file on the agent machine.
- If a self-signed certificate is used for the ASE configuration, "-allowUntrusted" option needs to be set in the deploy task for MSDeploy. It is also recommended to set the variable `VSTS_ARM_REST_IGNORE_SSL_ERRORS` to true. If a certificate from a certificate authority is used for ASE configuration, this should not be necessary.

## FAQs

### How should I configure my service connection?

This task requires an [Azure Resource Manager service connection](#).

### How should I configure web job deployment with Application Insights?

When you're deploying to an App Service, if you have [Application Insights](#) configured and you've enabled `Remove additional files at destination`, you also need to enable `Exclude files from the App_Data folder`. Enabling this option keeps the Application Insights extension in a safe state. This step is required because the Application Insights continuous WebJob is installed into the App\_Data folder.

### How should I configure my agent if it's behind a proxy while I'm deploying to App Service?

If your self-hosted agent requires a web proxy, you can inform the agent about the proxy during configuration. Doing so allows your agent to connect to Azure Pipelines or Azure DevOps Server through the proxy. [Learn more about running a self-hosted agent behind a web proxy](#).

## Open Source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Azure virtual machine scale set Deployment task

3/18/2021 • 3 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task to deploy a virtual machine scale set image.

## YAML snippet

```
Azure VM scale set deployment
Deploy a 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 virtual machine 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 virtual machine scale set only when it was created by using a custom image, the update will fail if the virtual machine scale set was created by using a platform/gallery image available in Azure.  The Custom script VM extension approach can be used for virtual machine scale set created by using either custom image or platform/gallery image.
Virtual machine scale set name	(Required) Name of virtual machine 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 virtual machine 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 virtual machine 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.
<b>CONTROL OPTIONS</b>	

## Troubleshooting

### Error: 'Permission denied: Script is not executable'

This issue occurs if you try to run a custom script, but the script isn't executable.

To resolve the issue, first make sure that the `customScript` input doesn't have `./` or anything else before the script name `'test.sh'`:

```
customScript: 'test.sh'
```

Next, try adding a command line task before the virtual machine scale set task:

```
- task: CmdLine@2
 inputs:
 script: 'chmod 777 $(System.DefaultWorkingDirectory)/test.sh'
```

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Azure Web App for Container task

11/2/2020 • 3 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task to deploy Web Apps, Azure Functions, and WebJobs to Azure App Services using a [custom Docker image](#).

## Task Inputs

PARAMETERS	DESCRIPTION
<code>azureSubscription</code> Azure subscription	(Required) Name of <a href="#">Azure Resource Manager service connection</a> .
<code>appName</code> App name	(Required) Name of Web App for Container.
<code>deployToSlotOrASE</code> Deploy to Slot or App Service Environment	(Optional) Set to true to deploy to an existing deployment slot or Azure App Service Environment. For both the targets, the task needs a Resource Group name. For the deployment slot option, the default is to deploy to the <b>production</b> slot, or you can specify any other existing slot name. If the deployment target is an Azure App Service environment, leave the slot name as <b>production</b> and just specify the Resource Group name. Default value: false
<code>resourceGroupName</code> Resource group	(Required if <code>deployToSlotOrASE</code> is true) Name of the Resource Group containing the Web App for Containers.
<code>slotName</code> Slot	(Required) Enter or select an existing slot other than the <b>production</b> slot. Default value: production
<code>imageName</code> Image name	(Required) Image to be used for deployment. Example: <code>myregistry.azurecr.io/nginx:latest</code>
<code>containerCommand</code> Startup command	(Optional) Startup command to be executed after deployment.
<code>appSettings</code> App settings	(Optional) Application settings to be entered using the syntax '-key value'. Values containing spaces must be enclosed in double quotes. Example: <code>-Port 5000 -RequestTimeout 5000 -WEBSITE_TIME_ZONE "Eastern Standard Time"</code>
<code>configurationStrings</code> Configuration settings	(Optional) Configuration strings to be entered using the syntax '-key value'. Values containing spaces must be enclosed in double quotes. Example: <code>-phpVersion 5.6 -linuxFxVersion: node 6.11</code>

## Example

This example deploys a Web App on Linux using containers:

```
variables:
 imageName: contoso.azurecr.io/aspnetcore:${(build.buildId)}
 azureSubscription: Contoso
 # To ignore SSL error uncomment the following variable
 # VSTS_ARM_REST_IGNORE_SSL_ERRORS: true

steps:
- task: AzureWebAppContainer@1
 displayName: Azure Web App on Container Deploy
 inputs:
 appName: webappforcontainers
 azureSubscription: $(azureSubscription)
 imageName: $(imageName)
```

## Troubleshooting

### Error: Could not fetch access token for Azure. Verify if the Service Principal used is valid and not expired.

The task uses the service principal in the service connection to authenticate with Azure. If the service principal has expired or doesn't have permissions to the App Service, the task fails with this error. Verify the validity of the service principal used and that it's present in the app registration. For more information, 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.

### SSL error

If you want to use a certificate in App Service, the certificate must be signed by a trusted certificate authority. If your web app gives you certificate validation errors, you're probably using a self-signed certificate. Set a variable named `VSTS_ARM_REST_IGNORE_SSL_ERRORS` to the value `true` in the build or release pipeline to resolve the error.

### A release hangs for long time and then fails

This problem could be the result of insufficient capacity in your App Service plan. To resolve this problem, you can scale up the App Service instance to increase available CPU, RAM, and disk space or try with a different App Service plan.

### 5xx error codes

If you're seeing a 5xx error, [check the status of your Azure service](#).

## FAQs

### How should I configure my service connection?

This task requires an [Azure Resource Manager service connection](#).

### How should I configure web job deployment with Application Insights?

When you're deploying to an App Service, if you have [Application Insights](#) configured and you've enabled

`Remove additional files at destination`, you also need to enable `Exclude files from the App_Data folder`.

Enabling this option keeps the Application Insights extension in a safe state. This step is required because the Application Insights continuous WebJob is installed into the App\_Data folder.

### How should I configure my agent if it's behind a proxy while I'm deploying to App Service?

If your self-hosted agent requires a web proxy, you can inform the agent about the proxy during configuration. Doing so allows your agent to connect to Azure Pipelines or Azure DevOps Server through the proxy. [Learn](#)

more about running a self-hosted agent behind a web proxy.

## Open Source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Build Machine Image task

4/10/2020 • 3 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task 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 a machine image using Packer, which may 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: true
 #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: '{vm_size:Standard_D3_v2}' # Optional
 #skipTempFileCleanupDuringVMDeprovision: true # Optional
 #packerVersion: # Optional
 #imageUri: # Optional
 #imageId: # 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 <a href="#">here</a>
Image URL	(Optional) Provide a name for the output variable which will store generated machine image url.
<b>CONTROL OPTIONS</b>	

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Chef task

4/10/2020 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task 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
<b>CONTROL OPTIONS</b>	

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Chef Knife task

4/10/2020 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task 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.

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Copy Files Over SSH task

11/2/2020 • 4 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

Use this task 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 [FAQ](#)).

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## 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
<b>Source folder</b>	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 <a href="#">variables</a> if files are not in the repository. Example: <code>\$(Agent.BuildDirectory)</code>
<b>Contents</b>	File paths to include as part of the copy. Supports multiple lines of <a href="#">minimatch patterns</a> . 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.
<b>Target folder</b>	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.
<b>Advanced - Clean target folder</b>	If this option is selected, all existing files in the target folder will be deleted before copying.
<b>Advanced - Overwrite</b>	If this option is selected (the default), existing files in the target folder will be replaced.
<b>Advanced - Flatten folders</b>	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.
<b>Control options</b>	See <a href="#">Control options</a>

## Supported algorithms

### Key pair algorithms

- RSA
- DSA

### Encryption algorithms

- aes256-cbc
- aes192-cbc
- aes128-cbc
- blowfish-cbc
- 3des-cbc
- arcfour256
- arcfour128
- cast128-cbc
- arcfour

For OpenSSL v1.0.1 and higher (on agent):

- aes256-ctr
- aes192-ctr
- aes128-ctr

For OpenSSL v1.0.1 and higher, NodeJS v0.11.12 and higher (on agent):

- aes128-gcm
- aes128-gcm@openssh.com
- aes256-gcm
- aes256-gcm@openssh.com

## 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.

## FAQ

### What key formats are supported for the SSH tasks?

The Azure Pipelines SSH tasks use the Node.js `ssh2` package for SSH connections. Ensure that you are using the latest version of the SSH tasks. Older versions may not support the OpenSSH key format.

If you run into an "Unsupported key format" error, then you may need to add the `-m PEM` flag to your `ssh-keygen` command so that the key is in a supported format.

### Do I need an agent?

You need at least one [agent](#) to run your build or release.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

### 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

3/6/2021 • 4 minutes to read • [Edit Online](#)

Use this task to build and push Docker images to any container registry using Docker registry service connection.

## Overview

Following are the key benefits of using Docker task as compared to directly using docker client binary in script -

- **Integration with Docker registry service connection** - The task makes it easy to use a Docker registry service connection for connecting to any container registry. Once logged in, the user can author follow up tasks to execute any tasks/scripts by leveraging the login already done by the Docker task. For example, you can use the Docker task to sign in to any Azure Container Registry and then use a subsequent task/script to build and push an image to this registry.
- **Metadata added as labels** - The task adds traceability-related metadata to the image in the form of the following labels -
  - com.azure.dev.image.build.buildnumber
  - com.azure.dev.image.build.builduri
  - com.azure.dev.image.build.definitionname
  - com.azure.dev.image.build.repository.name
  - com.azure.dev.image.build.repository.uri
  - com.azure.dev.image.build.sourcebranchname
  - com.azure.dev.image.build.sourceversion
  - com.azure.dev.image.release.definitionname
  - com.azure.dev.image.release.releaseid
  - com.azure.dev.image.release.releaseweburl
  - com.azure.dev.image.system.teamfoundationcollectionuri
  - com.azure.dev.image.system.teamproject

## Task Inputs

PARAMETERS	DESCRIPTION
<code>command</code> Command	(Required) Possible values: <code>buildAndPush</code> , <code>build</code> , <code>push</code> , <code>login</code> , <code>logout</code> Added in version 2.173.0: <code>start</code> , <code>stop</code> Default value: <code>buildAndPush</code>
<code>containerRegistry</code> Container registry	(Optional) Name of the <a href="#">Docker registry service connection</a>
<code>repository</code> Repository	(Optional) Name of repository within the container registry corresponding to the Docker registry service connection specified as input for <code>containerRegistry</code>

PARAMETERS	DESCRIPTION
<code>container</code> Container	(Required for commands <code>start</code> and <code>stop</code> ) The container resource to start or stop
<code>tags</code> Tags	(Optional) Multiline input where each line contains a tag to be used in <code>build</code> , <code>push</code> or <code>buildAndPush</code> commands Default value: <code>\$(Build.BuildId)</code>
<code>Dockerfile</code> Dockerfile	(Optional) Path to the Dockerfile. The task will use the <b>first</b> dockerfile it finds to build the image. Default value: <code>**/Dockerfile</code>
<code>buildContext</code> Build context	(Optional) Path to the build context Default value: <code>**</code>
<code>arguments</code> Arguments	(Optional) Additional arguments to be passed onto the docker client Be aware that if you use value <code>buildAndPush</code> for the <code>command</code> parameter, then the <code>arguments</code> property will be ignored.
<code>addPipelineData</code> Add Pipeline Data	(Optional) Adds the above mentioned metadata as labels to the image Possible values: <code>true</code> , <code>false</code> Default value: <code>true</code>

## Login

Following YAML snippet showcases container registry login using a Docker registry service connection -

```
- task: Docker@2
displayName: Login to ACR
inputs:
 command: login
 containerRegistry: dockerRegistryServiceConnection1
```

## Build and Push

A convenience command called `buildAndPush` allows for build and push of images to container registry in a single command. The following YAML snippet is an example of building and pushing multiple tags of an image to multiple registries -

```

steps:
- task: Docker@2
 displayName: Login to ACR
 inputs:
 command: login
 containerRegistry: dockerRegistryServiceConnection1
- task: Docker@2
 displayName: Login to Docker Hub
 inputs:
 command: login
 containerRegistry: dockerRegistryServiceConnection2
- task: Docker@2
 displayName: Build and Push
 inputs:
 command: buildAndPush
 repository: contosoRepository
 tags: |
 tag1
 tag2

```

In the above snippet, the images `contosoRepository:tag1` and `contosoRepository:tag2` are built and pushed to the container registries corresponding to `dockerRegistryServiceConnection1` and `dockerRegistryServiceConnection2`.

If one wants to build and push to a specific authenticated container registry instead of building and pushing to all authenticated container registries at once, the `containerRegistry` input can be explicitly specified along with `command: buildAndPush` as shown below -

```

steps:
- task: Docker@2
 displayName: Build and Push
 inputs:
 command: buildAndPush
 containerRegistry: dockerRegistryServiceConnection1
 repository: contosoRepository
 tags: |
 tag1
 tag2

```

## Logout

Following YAML snippet showcases container registry logout using a Docker registry service connection -

```

- task: Docker@2
 displayName: Logout of ACR
 inputs:
 command: logout
 containerRegistry: dockerRegistryServiceConnection1

```

## Start/stop

This task can also be used to control job and service containers. This usage is uncommon, but occasionally used for unique circumstances.

```

resources:
 containers:
 - container: builder
 image: ubuntu:18.04
 steps:
 - script: echo "I can run inside the container (it starts by default)"
 target:
 container: builder
 - task: Docker@2
 inputs:
 command: stop
 container: builder
 # any task beyond this point would not be able to target the builder container
 # because it's been stopped

```

## Other commands and arguments

The command and argument inputs can be used to pass additional arguments for build or push commands using docker client binary as shown below -

```

steps:
- task: Docker@2
 displayName: Login to ACR
 inputs:
 command: login
 containerRegistry: dockerRegistryServiceConnection1
- task: Docker@2
 displayName: Build
 inputs:
 command: build
 repository: contosoRepository
 tags: tag1
 arguments: --secret id=mysecret,src=mysecret.txt

```

### NOTE

The arguments input is evaluated for all commands except `buildAndPush`. As `buildAndPush` is a convenience command (`build` followed by `push`), `arguments` input is ignored for this command.

## Troubleshooting

### Why does Docker task ignore arguments passed to buildAndPush command?

Docker task configured with `buildAndPush` command ignores the arguments passed since they become ambiguous to the build and push commands that are run internally. You can split your command into separate build and push steps and pass the suitable arguments. See this [stackoverflow post](#) for example.

### DockerV2 only supports Docker registry service connection and not support ARM service connection. How can I use an existing Azure service principal (SPN) for authentication in Docker task?

You can create a Docker registry service connection using your Azure SPN credentials. Choose the Others from Registry type and provide the details as follows:

Docker Registry:	Your container registry URL (eg. <a href="https://myacr.azurecr.io">https://myacr.azurecr.io</a> )
Docker ID:	Service principal client ID
Password:	Service principal key

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Docker Compose task

5/18/2021 • 11 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task to build, push or run multi-container Docker applications. This task can be used with a Docker registry or an Azure Container Registry.

## Container registry types

### Azure Container Registry

PARAMETERS	DESCRIPTION
<code>containerregistrytype</code> (Container registry type)	(Optional) Azure Container Registry if using ACR or Container Registry if using any other container registry. Default value: Azure Container Registry
<code>azureSubscriptionEndpoint</code> (Azure subscription)	(Required) Name of the Azure Service Connection. See <a href="#">Azure Resource Manager service connection</a> to manually set up the connection. Argument aliases: <code>azureSubscription</code>
<code>azureContainerRegistry</code> (Azure Container Registry)	(Required) Name of the Azure Container Registry. Example: <code>Contoso.azurecr.io</code>

This YAML example specifies the inputs for Azure Container Registry:

```
variables:
 azureContainerRegistry: Contoso.azurecr.io
 azureSubscriptionEndpoint: Contoso
steps:
- task: DockerCompose@0
 displayName: Container registry login
 inputs:
 containerregistrytype: Azure Container Registry
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureContainerRegistry: $(azureContainerRegistry)
```

### Other container registries

The `containerregistrytype` value is required when using any container registry other than ACR. Use

`containerregistrytype: Container Registry` in this case.

PARAMETERS	DESCRIPTION
<code>containerregistrytype</code> (Container registry type)	(Required) Azure Container Registry if using ACR or Container Registry if using any other container registry. Default value: Azure Container Registry
<code>dockerRegistryEndpoint</code> (Docker registry service connection)	(Required) <a href="#">Docker registry service connection</a> .

This YAML example specifies a container registry other than ACR where `Contoso` is the name of the Docker

registry service connection for the container registry:

```
- task: DockerCompose@0
displayName: Container registry login
inputs:
 containerregistrytype: Container Registry
 dockerRegistryEndpoint: Contoso
```

## Build service images

PARAMETERS	DESCRIPTION
<code>containerregistrytype</code> (Container Registry Type)	(Required) Azure Container Registry if using ACR or Container Registry if using any other container registry. Default value: Azure Container Registry
<code>azureSubscriptionEndpoint</code> (Azure subscription)	(Required) Name of the Azure Service Connection.
<code>azureContainerRegistry</code> (Azure Container Registry)	(Required) Name of the Azure Container Registry.
<code>dockerComposeFile</code> (Docker Compose File)	(Required) Path to the primary Docker Compose file to use. Default value: **/docker-compose.yml
<code>additionalDockerComposeFiles</code> (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.
<code>dockerComposeFileArgs</code> (Environment Variables)	(Optional) Environment variables to be set up during the command. Specify each name = value pair on a new line. You need to use the   operator in YAML to indicate that newlines should be preserved. Example: <code>dockerComposeFileArgs:   DOCKER_BUILD_SOURCE=\$(CustomVar)</code>
<code>projectName</code> (Project Name)	(Optional) Project name used for default naming of images and containers. Default value: \$(Build.Repository.Name)
<code>qualifyImageNames</code> (Qualify Image Names)	(Optional) Qualify image names for built services with the Docker registry service connection's hostname if not otherwise specified. Default value: true
<code>action</code> (Action)	(Required) Select a Docker Compose action. Default value: Run a Docker Compose command
<code>additionalImageTags</code> (Additional Image Tags)	(Optional) Additional tags for the Docker images being built or pushed.

PARAMETERS	DESCRIPTION
<code>includeSourceTags</code> (Include Source Tags)	(Optional) Include Git tags when building or pushing Docker images. Default value: false
<code>includeLatestTag</code> (Include Latest Tag)	(Optional) Include the latest tag when building or pushing Docker images. Default value: false

This YAML example builds the image where the image name is qualified on the basis of the inputs related to Azure Container Registry:

```
- task: DockerCompose@0
 displayName: Build services
 inputs:
 action: Build services
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureContainerRegistry: $(azureContainerRegistry)
 dockerComposeFile: docker-compose.yml
 projectName: $(Build.Repository.Name)
 qualifyImageNames: true
 additionalImageTags: $(Build.BuildId)
 dockerComposeFileArgs: |
 firstArg=$(firstArg)
 secondArg=$(secondArg)
```

## Push service images

PARAMETERS	DESCRIPTION
<code>containerregistrytype</code> (Container Registry Type)	(Required) Azure Container Registry if using ACR or Container Registry if using any other container registry. Default value: Azure Container Registry
<code>azureSubscriptionEndpoint</code> (Azure subscription)	(Required) Name of the Azure Service Connection.
<code>azureContainerRegistry</code> (Azure Container Registry)	(Required) Name of the Azure Container Registry.
<code>dockerComposeFile</code> (Docker Compose File)	(Required) Path to the primary Docker Compose file to use. Default value: **/docker-compose.yml
<code>additionalDockerComposeFiles</code> (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.
<code>dockerComposeFileArgs</code> (Environment Variables)	(Optional) Environment variables to be set up during the command. Specify each name=value pair on a new line.

PARAMETERS	DESCRIPTION
<code>projectName</code> (Project Name)	(Optional) Project name used for default naming of images and containers. Default value: \$(Build.Repository.Name)
<code>qualifyImageNames</code> (Qualify Image Names)	(Optional) Qualify image names for built services with the Docker registry service connection's hostname if not otherwise specified. Default value: true
<code>action</code> (Action)	(Required) Select a Docker Compose action. Default value: Run a Docker Compose command
<code>additionalImageTags</code> (Additional Image Tags)	(Optional) Additional tags for the Docker images being built or pushed.
<code>includeSourceTags</code> (Include Source Tags)	(Optional) Include Git tags when building or pushing Docker images. Default value: false
<code>includeLatestTag</code> (Include Latest Tag)	(Optional) Include the latest tag when building or pushing Docker images. Default value: false

This YAML example pushes an image to a container registry:

```
- task: DockerCompose@0
displayName: Push services
inputs:
 action: Push services
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureContainerRegistry: $(azureContainerRegistry)
 dockerComposeFile: docker-compose.yml
 projectName: $(Build.Repository.Name)
 qualifyImageNames: true
 additionalImageTags: $(Build.BuildId)
```

## Run service images

PARAMETERS	DESCRIPTION
<code>dockerComposeFile</code> (Docker Compose File)	(Required) Path to the primary Docker Compose file to use. Default value: **/docker-compose.yml
<code>additionalDockerComposeFiles</code> (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.
<code>dockerComposeFileArgs</code> (Environment Variables)	(Optional) Environment variables to be set up during the command. Specify each name=value pair on a new line.

PARAMETERS	DESCRIPTION
<code>projectName</code> (Project Name)	(Optional) Project name used for default naming of images and containers. Default value: \$(Build.Repository.Name)
<code>qualifyImageNames</code> (Qualify Image Names)	(Optional) Qualify image names for built services with the Docker registry service connection's hostname if not otherwise specified. Default value: true
<code>action</code> (Action)	(Required) Select a Docker Compose action. Default value: Run a Docker Compose command
<code>buildImages</code> (Build Images)	(Optional) Build images before starting service containers. Default value: true
<code>detached</code> (Run in Background)	(Optional) Run the service containers in the background. Default value: true

This YAML example runs services:

```
- task: DockerCompose@0
 displayName: Run services
 inputs:
 action: Run services
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureContainerRegistry: $(azureContainerRegistry)
 dockerComposeFile: docker-compose.ci.build.yml
 projectName: $(Build.Repository.Name)
 qualifyImageNames: true
 buildImages: true
 abortOnContainerExit: true
 detached: false
```

## Run a specific service image

PARAMETERS	DESCRIPTION
<code>dockerComposeFile</code> (Docker Compose File)	(Required) Path to the primary Docker Compose file to use. Default value: **/docker-compose.yml
<code>additionalDockerComposeFiles</code> (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.
<code>dockerComposeFileArgs</code> (Environment Variables)	(Optional) Environment variables to be set up during the command. Specify each name=value pair on a new line.
<code>projectName</code> (Project Name)	(Optional) Project name used for default naming of images and containers. Default value: \$(Build.Repository.Name)

PARAMETERS	DESCRIPTION
<code>qualifyImageNames</code> (Qualify Image Names)	(Optional) Qualify image names for built services with the Docker registry service connection's hostname if not otherwise specified. Default value: true
<code>action</code> (Action)	(Required) Select a Docker Compose action. Default value: Run a Docker Compose command
<code>serviceName</code> (Service Name)	(Required) Name of the specific service to run.
<code>containerName</code> (Container Name)	(Optional) Name of the specific service container to run.
<code>ports</code> (Ports)	(Optional) Ports in the specific service container to publish to the host. Specify each host-port:container-port binding on a new line.
<code>workDir</code> (Working Directory)	(Optional) The working directory for the specific service container. Argument aliases: <code>workingDirectory</code>
<code>entrypoint</code> (Entry Point Override)	(Optional) Override the default entry point for the specific service container.
<code>containerCommand</code> (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 <code>python app.py</code> to launch the web application.
<code>detached</code> (Run in Background)	(Optional) Run the service containers in the background. Default value: true

This YAML example runs a specific service:

```
- task: DockerCompose@0
 displayName: Run a specific service
 inputs:
 action: Run a specific service
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureContainerRegistry: $(azureContainerRegistry)
 dockerComposeFile: docker-compose.yml
 projectName: $(Build.Repository.Name)
 qualifyImageNames: true
 serviceName: myhealth.web
 ports: 80:80
 detached: true
```

## Lock service images

PARAMETERS	DESCRIPTION
<code>dockerComposeFile</code> (Docker Compose File)	(Required) Path to the primary Docker Compose file to use. Default value: <code>**/docker-compose.yml</code>

PARAMETERS	DESCRIPTION
<code>additionalDockerComposeFiles</code> (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.
<code>dockerComposeFileArgs</code> (Environment Variables)	(Optional) Environment variables to be set up during the command. Specify each name=value pair on a new line.
<code>projectName</code> (Project Name)	(Optional) Project name used for default naming of images and containers. Default value: \$(Build.Repository.Name)
<code>qualifyImageNames</code> (Qualify Image Names)	(Optional) Qualify image names for built services with the Docker registry service connection's hostname if not otherwise specified. Default value: true
<code>action</code> (Action)	(Required) Select a Docker Compose action. Default value: Run a Docker Compose command
<code>removeBuildOptions</code> (Remove Build Options)	(Optional) Remove the build options from the output Docker Compose file. Default value: false
<code>baseResolveDirectory</code> (Base Resolve Directory)	(Optional) The base directory from which relative paths in the output Docker Compose file should be resolved.
<code>outputDockerComposeFile</code> (Output Docker Compose File)	(Required) Path to an output Docker Compose file. Default value: \$(Build.StagingDirectory)/docker-compose.yml

This YAML example locks services:

```
- task: DockerCompose@0
 displayName: Lock services
 inputs:
 action: Lock services
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureContainerRegistry: $(azureContainerRegistry)
 dockerComposeFile: docker-compose.yml
 projectName: $(Build.Repository.Name)
 qualifyImageNames: true
 outputDockerComposeFile: $(Build.StagingDirectory)/docker-compose.yml
```

## Write service image digests

PARAMETERS	DESCRIPTION
<code>dockerComposeFile</code> (Docker Compose File)	(Required) Path to the primary Docker Compose file to use. Default value: **/docker-compose.yml

PARAMETERS	DESCRIPTION
<code>additionalDockerComposeFiles</code> (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.
<code>dockerComposeFileArgs</code> (Environment Variables)	(Optional) Environment variables to be set up during the command. Specify each name=value pair on a new line.
<code>projectName</code> (Project Name)	(Optional) Project name used for default naming of images and containers. Default value: \$(Build.Repository.Name)
<code>qualifyImageNames</code> (Qualify Image Names)	(Optional) Qualify image names for built services with the Docker registry service connection's hostname if not otherwise specified. Default value: true
<code>action</code> (Action)	(Required) Select a Docker Compose action. Default value: Run a Docker Compose command
<code>imageDigestComposeFile</code> (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. Default value: \$(Build.StagingDirectory)/docker-compose.images.yml

This YAML example writes service image digests:

```
- task: DockerCompose@0
 displayName: Write service image digests
 inputs:
 action: Write service image digests
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureContainerRegistry: $(azureContainerRegistry)
 dockerComposeFile: docker-compose.yml
 projectName: $(Build.Repository.Name)
 qualifyImageNames: true
 imageDigestComposeFile: $(Build.StagingDirectory)/docker-compose.images.yml
```

## Combine configuration

PARAMETERS	DESCRIPTION
<code>dockerComposeFile</code> (Docker Compose File)	(Required) Path to the primary Docker Compose file to use. Default value: **/docker-compose.yml
<code>additionalDockerComposeFiles</code> (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.
<code>dockerComposeFileArgs</code> (Environment Variables)	(Optional) Environment variables to be set up during the command. Specify each name=value pair on a new line

PARAMETERS	DESCRIPTION
<code>projectName</code> (Project Name)	(Optional) Project name used for default naming of images and containers. Default value: \$(Build.Repository.Name)
<code>qualifyImageNames</code> (Qualify Image Names)	(Optional) Qualify image names for built services with the Docker registry service connection's hostname if not otherwise specified. Default value: true
<code>action</code> (Action)	(Required) Select a Docker Compose action. Default value: Run a Docker Compose command
<code>removeBuildOptions</code> (Remove Build Options)	(Optional) Remove the build options from the output Docker Compose file. Default value: false
<code>baseResolveDirectory</code> (Base Resolve Directory)	(Optional) The base directory from which relative paths in the output Docker Compose file should be resolved.
<code>outputDockerComposeFile</code> (Output Docker Compose File)	(Required) Path to an output Docker Compose file. Default value: \$(Build.StagingDirectory)/docker-compose.yml

This YAML example combines configurations:

```
- task: DockerCompose@0
 displayName: Combine configuration
 inputs:
 action: Combine configuration
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureContainerRegistry: $(azureContainerRegistry)
 dockerComposeFile: docker-compose.yml
 additionalDockerComposeFiles: docker-compose.override.yml
 projectName: $(Build.Repository.Name)
 qualifyImageNames: true
 outputDockerComposeFile: $(Build.StagingDirectory)/docker-compose.yml
```

## Run a Docker Compose command

PARAMETERS	DESCRIPTION
<code>dockerComposeFile</code> (Docker Compose File)	(Required) Path to the primary Docker Compose file to use. Default value: **/docker-compose.yml
<code>additionalDockerComposeFiles</code> (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.
<code>dockerComposeFileArgs</code> (Environment Variables)	(Optional) Environment variables to be set up during the command. Specify each name=value pair on a new line.

PARAMETERS	DESCRIPTION
<code>projectName</code> (Project Name)	(Optional) Project name used for default naming of images and containers. Default value: \$(Build.Repository.Name)
<code>qualifyImageNames</code> (Qualify Image Names)	(Optional) Qualify image names for built services with the Docker registry service connection's hostname if not otherwise specified. Default value: true
<code>action</code> (Action)	(Required) Select a Docker Compose action. Default value: Run a Docker Compose command
<code>dockerComposeCommand</code> (Command)	(Required) Docker Compose command to execute with the help of arguments. For example, rm to remove all stopped service containers.

This YAML example runs a docker Compose command:

```
- task: DockerCompose@0
 displayName: Run a Docker Compose command
 inputs:
 action: Run a Docker Compose command
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureContainerRegistry: $(azureContainerRegistry)
 dockerComposeFile: docker-compose.yml
 projectName: $(Build.Repository.Name)
 qualifyImageNames: true
 dockerComposeCommand: rm
```

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

# Package and Deploy Helm Charts task

3/6/2021 • 10 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task 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 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 install 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 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`, and others.

## Service Connection

The task works with two service connection types: **Azure Resource Manager** and **Kubernetes Service Connection**.

### NOTE

A service connection isn't required if an environment resource that points to a Kubernetes cluster has already been specified in the pipeline's stage.

## Azure Resource Manager

PARAMETERS	DESCRIPTION
<code>connectionType</code> (Service connection type)	(Required unless an environment resource is already present) <b>Azure Resource Manager</b> to use Azure Kubernetes Service. <b>Kubernetes Service Connection</b> for any other cluster. Default value: Azure Resource Manager
<code>azureSubscriptionEndpoint</code> (Azure subscription)	(Required) Name of the Azure service connection.

PARAMETERS	DESCRIPTION
azureResourceGroup (Resource group)	(Required) Name of the resource group within the subscription.
kubernetesCluster (Kubernetes cluster)	(Required) Name of the AKS cluster.
namespace (Namespace)	(Optional) The namespace on which the <b>kubectl</b> commands are run. If not specified, the default namespace is used.

This YAML example YAML shows how Azure Resource Manager is used to refer to the Kubernetes cluster. This is used with one of the helm [commands](#) and the appropriate values required for the command:

```
variables:
 azureSubscriptionEndpoint: Contoso
 azureContainerRegistry: contoso.azurecr.io
 azureResourceGroup: Contoso
 kubernetesCluster: Contoso

- task: HelmDeploy@0
 displayName: Helm deploy
 inputs:
 connectionType: Azure Resource Manager
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureResourceGroup: $(azureResourceGroup)
 kubernetesCluster: $(kubernetesCluster)
```

## Kubernetes Service Connection

PARAMETERS	DESCRIPTION
kubernetesServiceEndpoint (Kubernetes service connection)	(Required unless an environment resource is already present) Select a Kubernetes service connection.
namespace (Namespace)	(Optional) The namespace on which the <b>kubectl</b> commands are run. If not specified, the default namespace is used.

This YAML example YAML shows how Kubernetes service connection is used to refer to the Kubernetes cluster. This is used with one of the helm [commands](#) and the appropriate values required for the command:

```
- task: HelmDeploy@0
 displayName: Helm deploy
 inputs:
 connectionType: Kubernetes Service Connection
 kubernetesServiceEndpoint: Contoso
```

## Command values

The command input accepts one of the following [helm commands](#):  
create/delete/expose/get/init/install/login/logout/ls/package/rollback/upgrade.

PARAMETERS	DESCRIPTION
<b>command</b> (Command)	(Required) Select a helm command. Default value: ls
<b>arguments</b> (Arguments)	Helm command options.

This YAML example demonstrates the `ls` command:

```
- task: HelmDeploy@0
displayName: Helm list
inputs:
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureResourceGroup: $(azureResourceGroup)
 kubernetesCluster: $(kubernetesCluster)
 command: ls
 arguments: --all
```

## init command

PARAMETERS	DESCRIPTION
<b>command</b> (Command)	(Required) Select a helm command. Default value: ls
<b>canaryimage</b> (Use canary image version)	Use the canary Tiller image, the latest pre-release version of Tiller. Default value: false
<b>upgradetiller</b> (Upgrade Tiller)	Upgrade if Tiller is already installed. Default value: true
<b>waitForExecution</b> (Wait)	Block until the command execution completes. Default value: true
<b>arguments</b> (Arguments)	Helm command options.

This YAML example demonstrates the `init` command:

```
- task: HelmDeploy@0
displayName: Helm init
inputs:
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureResourceGroup: $(azureResourceGroup)
 kubernetesCluster: $(kubernetesCluster)
 command: init
 upgradetiller: true
 waitForExecution: true
 arguments: --client-only
```

## install command

PARAMETERS	DESCRIPTION
<code>command</code> (Command)	(Required) Select a helm command. Default value: ls
<code>chartType</code> (Chart Type)	(Required) Select how you want to enter chart information. You can provide either the name of the chart or the folder/file path to the chart. Available options: Name, FilePath. Default value: Name
<code>chartName</code> (Chart Name)	(Required) Chart reference to install, this can be a url or a chart name. For example, if chart name is stable/mysql, the task will run <b>helm install stable/mysql</b>
<code>releaseName</code> (Release Name)	(Optional) Release name. If not specified, it will be autogenerated. releaseName input is only valid for 'install' and 'upgrade' commands
<code>overrideValues</code> (Set Values)	(Optional) Set values on the command line. You can specify multiple values by separating values with commas. For example, <code>key1=val1,key2=val2</code> . You can also specify multiple values by delimiting them with newline as so: <code>key1=val1</code> <code>key2=val2</code> Please note that if you have a value which itself contains newlines, use the <code>valueFile</code> option, else the task will treat the newline as a delimiter. The task will construct the helm command by using these set values. For example, <b>helm install --set key1=val1 ./redis</b>
<code>valueFile</code> (Value File)	(Optional) Specify values in a YAML file or a URL. For example, specifying <code>myvalues.yaml</code> will result in <b>helm install --values=myvalues.yaml</b>
<code>updatedependency</code> (Update Dependency)	(Optional) Run helm dependency update before installing the chart. Update dependencies from <code>requirements.yaml</code> to the <code>charts/</code> directory before packaging. Default value: false
<code>waitForExecution</code> (Wait)	(Optional) Block until command execution completes. Default value: true
<code>arguments</code> (Arguments)	Helm command options

This YAML example demonstrates the **install** command:

```
- task: HelmDeploy@0
displayName: Helm install
inputs:
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureResourceGroup: $(azureResourceGroup)
 kubernetesCluster: $(kubernetesCluster)
 command: install
 chartType: FilePath
 chartPath: Application/charts/sampleapp
```

## package command

PARAMETERS	DESCRIPTION
<code>command</code> (Command)	(Required) Select a helm command. Default value: ls
<code>chartPath</code> (Chart Path)	(Required) Path to the chart to install. This 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 <b>helm install ./redis</b> . If you are consuming a chart which is published as an artifact then the path will be <b>\$(System.DefaultWorkingDirectory)/ARTIFACT-NAME/Charts/CHART-NAME</b>
<code>version</code> (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.
<code>destination</code> (Destination)	(Optional) Specify values in a YAML file or a URL. Default value: \$(Build.ArtifactStagingDirectory)
<code>updatedependency</code> (Update Dependency)	(Optional) Run helm dependency update before installing the chart. Update dependencies from <b>requirements.yaml</b> to the <b>charts/</b> directory before packaging. Default value: false
<code>save</code> (Save)	(Optional) Save packaged chart to local chart repository. Default value: true
<code>arguments</code> (Arguments)	Helm command options.

This YAML example demonstrates the `package` command:

```
- task: HelmDeploy@0
 displayName: Helm package
 inputs:
 command: package
 chartPath: Application/charts/sampleapp
 destination: $(Build.ArtifactStagingDirectory)
```

## upgrade command

PARAMETERS	DESCRIPTION
<code>command</code> (Command)	(Required) Select a helm command. Default value: ls
<code>chartType</code> (Chart Type)	(Required) Select how you want to enter chart information. You can provide either the name of the chart or the folder/file path to the chart. Available options: Name, FilePath. Default value: Name

PARAMETERS	DESCRIPTION
<code>chartName</code> (Chart Name)	(Required) Chart reference to install, this can be a url or a chart name. For example, if chart name is <code>stable/mysql</code> , the task will run <code>helm install stable/mysql</code>
<code>releaseName</code> (Release Name)	(Optional) Release name. If not specified, it will be autogenerated.
<code>overrideValues</code> (Set Values)	(Optional) Set values on the command line. You can specify multiple values by separating values with commas. For example, <code>key1=val1,key2=val2</code> . You can also specify multiple values by delimiting them with newline as so: <code>key1=val1</code> <code>key2=val2</code> Please note that if you have a value which itself contains newlines, use the <code>valueFile</code> option, else the task will treat the newline as a delimiter. The task will construct the helm command by using these set values. For example, <code>helm install --set key1=val1 ./redis</code>
<code>valueFile</code> (Value File)	(Optional) Specify values in a YAML file or a URL. For example, specifying <code>myvalues.yaml</code> will result in <code>helm install --values=myvalues.yaml</code>
<code>install</code> (Install if release not present)	(Optional) If a release by this name does not already exist, start an installation. Default value: true
<code>recreate</code> (Recreate Pods)	(Optional) Performs pods restart for the resource if applicable. Default value: false
<code>resetValues</code> (Reset Values)	(Optional) Reset the values to the ones built into the chart. Default value: false
<code>force</code> (Force)	(Optional) Force resource update through delete/recreate if required. Default value: false
<code>waitForExecution</code> (Wait)	(Optional) Block until command execution completes. Default value: true
<code>arguments</code> (Arguments)	Helm command options

This YAML example demonstrates the `upgrade` command:

```

- task: HelmDeploy@0
 displayName: Helm upgrade
 inputs:
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureResourceGroup: $(azureResourceGroup)
 kubernetesCluster: $(kubernetesCluster)
 command: upgrade
 chartType: filepath
 chartPath: $(Build.ArtifactStagingDirectory)/sampleapp-v0.2.0.tgz
 releaseName: azureddevopsdemo
 install: true
 waitForExecution: false

```

## Save command

PARAMETERS	DESCRIPTION
<code>command</code> (Command)	(Required) Select a helm command. Default value: ls
<code>chartNameForACR</code> (Chart Name For Azure Container Registry)	(Required) Chart name with which the chart will be stored in Azure Container Registry.
<code>chartPathForACR</code> (Chart Path for Azure Container Registry)	(Required) Path to the chart directory.
<code>azureSubscriptionEndpointForACR</code> (Azure subscription for Container Registry)	(Required) Select an Azure subscription, which has your Azure Container Registry.
<code>azureResourceGroupForACR</code> (Resource group)	(Required) Select an Azure Resource Group, which has your Container Registry.
<code>azureContainerRegistry</code> (Azure Container Registry)	(Required) Select an Azure Container Registry which will be used for pushing helm charts.
<code>arguments</code> (Arguments)	Helm command options

This YAML example demonstrates the `save` command:

```

- task: HelmDeploy@0
 displayName: Helm save
 inputs:
 command: save
 chartNameForACR: mycontainerregistry.azurecr.io/helm/hello-world:v1
 chartPathForACR: Application/charts/sampleapp
 azureSubscriptionEndpointForACR: $(azureSubscriptionEndpointForACR)
 azureResourceGroupForACR: $(azureResourceGroupForACR)
 azureContainerRegistry: $(azureContainerRegistry)

```

## Package and sign Helm charts

In this section you'll learn how to package and sign Helm charts in a pipeline.

### Generate a private-public key pair to sign the helm chart using GPG

1. Download [GPG](#).

2. Launch the command prompt in an administrator mode. Run the following command to generate a private-public key pair to sign the helm chart using gpg. While creating the key, you'll be prompted for the username and email address. The "name email address" is later used to name the private-public key pair that is created.

```
gpg --full-generate-key
```

```
C:\WINDOWS\system32>gpg --full-generate-key
gpg (GnuPG) 2.2.11; Copyright (C) 2018 Free Software Foundation, Inc.
This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Please select what kind of key you want:
 (1) RSA and RSA (default)
 (2) DSA and Elgamal
 (3) DSA (sign only)
 (4) RSA (sign only)
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
Requested keysize is 2048 bits
Please specify how long the key should be valid.
 0 = key does not expire
 <n> = key expires in n days
 <n>w = key expires in n weeks
 <n>m = key expires in n months
 <n>y = key expires in n years
Key is valid for? (0)
Key does not expire at all
Is this correct? (y/N) y

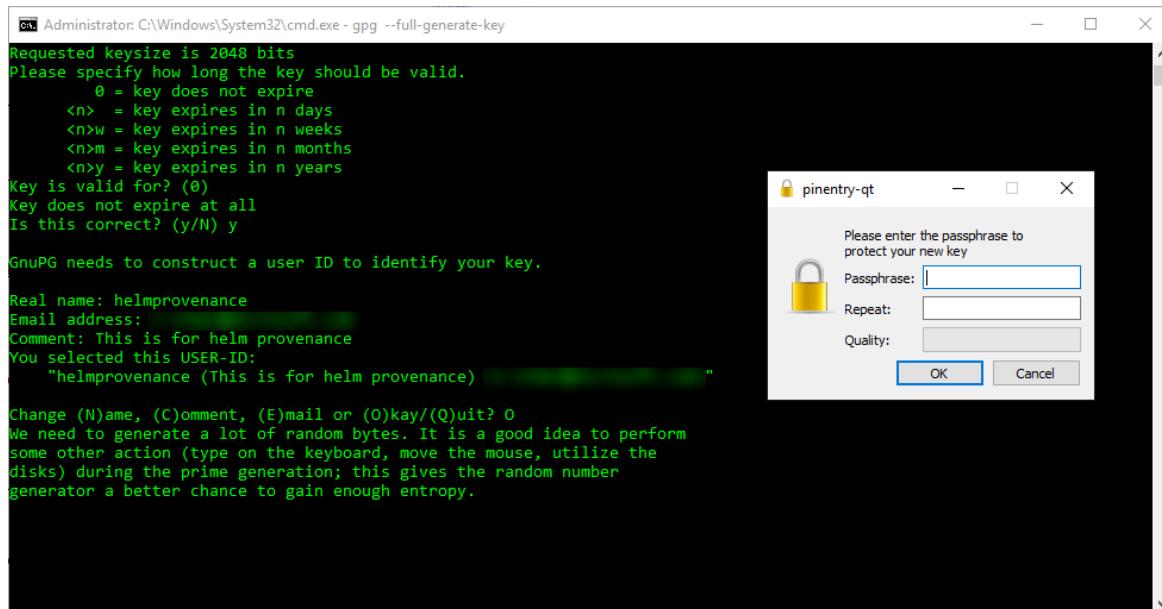
GnuPG needs to construct a user ID to identify your key.

Real name: helmprovance
Email address:
Comment: This is for helm provenance
You selected this USER-ID:
 "helmprovance (This is for helm provenance)"

Change (N)ame, (C)omment, (E)mail or (O)key/(Q)uit? O
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: revocation certificate stored as 'C:/Users/srivatsam/AppData/Roaming/gnupg/openpgp-revocs.d/296EB600AB393EAAA3F987F98D713E3969D8DE89.rev'
public and secret key created and signed.

pub rsa2048 2018-11-30 [S]
uid helmprovance (This is for helm provenance)
sub rsa2048 2018-11-30 [E]
```

3. You'll be prompted for the passphrase. Give the value and click ok.



4. After creating the key, you can see the list of keys which contains both private and public using the following command.

- To see list of private keys

```
gpg --list-secret-keys
```

```
C:\Users\srivatsam>gpg --list-secret-keys
C:/Users/srivatsam/AppData/Roaming/gnupg/pubring.kbx

sec rsa2048 2018-11-30 [SC]
 [ultimate] contoso (This is for helm)
uid
ssb rsa2048 2018-11-30 [E]
```

```
C:\Users\srivatsam>
```

- To see the list of public keys

```
gpg --list-keys
```

```
C:\Users\srivatsam>gpg --list-keys
C:/Users/srivatsam/AppData/Roaming/gnupg/pubring.kbx

pub rsa2048 2018-11-30 [SC]
 [ultimate] contoso (This is for helm)
sub rsa2048 2018-11-30 [E]
```

```
C:\Users\srivatsam>
```

5. Store the private and public keys in 2 different files with the extension **gpg** as shown below.

- For a private key

```
gpg --export-secret-key 94325E18E53EDD99DD8339C3CFD9DAF0707CB788 contoso@microsoft.com >
C:/somepath/privatekeys.gpg
```

You'll see the **privatekeys.gpg** file exported to the path which was mentioned above.

- For a public key

```
gpg --export-key 94325E18E53EDD99DD8339C3CFD9DAF0707CB788 contoso@microsoft.com >
C:/somepath/publickey.gpg
```

You'll see the **publickey.gpg** file exported to the path which was mentioned above.

In Azure DevOps, save the **privatekey.gpg** file in the library **secure files** section.

### Example

```

pool:
 name: Hosted Ubuntu 1604

variables:
 # The below variable should be secure
 HelmKeyPassphrase: contoso@123
 keyName: contoso contoso@microsoft.com
 azureSubscriptionEndpoint: contoso
 azureResourceGroup: contoso
 kubernetesCluster: contoso

steps:
- task: DownloadSecureFile@1
 displayName: Download Secure file
 inputs:
 secureFile: privatekey.gpg
 name: privateKeyRing

- task: HelmInstaller@0
 displayName: Install Helm 2.12.0
 inputs:
 helmVersion: 2.12.0

- task: HelmDeploy@0
 displayName: helm init
 inputs:
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureResourceGroup: $(azureResourceGroup)
 kubernetesCluster: $(kubernetesCluster)
 command: init
 arguments: --client-only

- task: HelmDeploy@0
 displayName: helm package
 inputs:
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureResourceGroup: $(azureResourceGroup)
 kubernetesCluster: $(kubernetesCluster)
 command: package
 chartPath: Application/charts/sampleapp
 arguments: --sign --key "$(keyName)" --keyring $(privateKeyRing.secureFilePath)
 env:
 HelmKeyPassphrase: $(HelmKeyPassphrase)

```

## Troubleshooting

### **HelmDeploy task throws error 'unknown flag: --wait' while running 'helm init --wait --client-only' on Helm 3.0.2 version.**

There are some breaking changes between Helm 2 and Helm 3. One of them includes removal of tiller, and hence `helm init` command is no longer supported. Remove command: init when you use Helm 3.0+ versions.

### **When using Helm 3, if System.debug is set to true and Helm upgrade is the command being used, the pipeline fails even though the upgrade was successful.**

This is a known issue with Helm 3, as it writes some logs to stderr. Helm Deploy Task is marked as failed if there are logs to stderr or exit code is non-zero. Set the task input failOnStderr: false to ignore the logs printed to stderr.

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# IIS Web App Deploy task

6/7/2021 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task 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 ( <a href="#">Build</a>   <a href="#">Release</a> ), 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 App 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>.&lt;stageName&gt;.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

11/2/2020 • 9 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task to create or update a Website, Web App, Virtual Directory, or Application Pool.

## YAML snippet

```
IIS web app manage
Create or update websites, web apps, virtual directories, or application pools
- task: IISWebAppManagementOnMachineGroup@0
 inputs:
 #enableIIS: false # Optional
 #IISDeploymentType: 'IISWebsite' # Options: iISWebsite, iISWebApplication, iISVirtualDirectory,
 iISApplicationPool
 #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: 'VDUserPassThrough' # 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	<p>(Required) Select the appropriate action that you want to perform on an IIS website.</p> <p>"Create Or Update" will create a website or update an existing website.</p> <p>Start, Stop will start or stop the website respectively.</p>
Action	<p>(Required) Select the appropriate action that you want to perform on an IIS Application Pool.</p> <p>"Create Or Update" will create app-pool or update an existing one.</p> <p>Start, Stop, Recycle will start, stop or recycle the application pool respectively.</p>
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.

ARGUMENT	DESCRIPTION
Username	(Required) Provide the user name that will be used to access the website's physical path.
Password	(Optional) Provide the user's password that will be used to access the website'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 <a href="#">command-line arguments</a>
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	(Required) Provide an IP address that end-users can use to access this website. 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.
Port	(Required) Provide the port, where the Hypertext Transfer Protocol Stack (HTTP.sys) will listen to the website requests.
Server Name Indication required	(Optional) Select the option to set the Server Name Indication (SNI) for the website. 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.
Host name	(Optional) Enter a host name (or domain name) for the website. If a host name is specified, then the clients must use the host name instead of the IP address to access the website.
Host name	(Optional) Enter a host name (or domain name) for the website. If a host name is specified, then the clients must use the host name instead of the IP address to access the website.
Host name	(Required) Enter a host name (or domain name) for the website. If a host name is specified, then the clients must use the host name instead of the IP address to access the website.
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.

ARGUMENT	DESCRIPTION
Add bindings	(Required) Click on the extension [...] button to add bindings for the website.
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 <a href="#">command-line arguments</a>
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 <a href="#">command-line arguments</a>
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 <a href="#">command-line arguments</a>
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 <a href="#">command-line arguments</a>
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 <a href="#">command-line arguments</a>
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
<b>CONTROL OPTIONS</b>	

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Kubectl task

6/25/2021 • 6 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task to deploy, configure, or update a Kubernetes cluster by running kubectl commands.

## Service Connection

The task works with two service connection types: **Azure Resource Manager** and **Kubernetes Service Connection**, described below.

### Azure Resource Manager

PARAMETERS	DESCRIPTION
<code>connectionType</code> Service connection type	(Required) Azure Resource Manager when using Azure Kubernetes Service, or Kubernetes Service Connection for any other cluster. Default value: Azure Resource Manager
<code>azureSubscriptionEndpoint</code> Azure subscription	(Required) Name of the Azure Service Connection.
<code>azureResourceGroup</code> Resource group	(Required) Name of the resource group within the subscription.
<code>kubernetesCluster</code> Kubernetes cluster	(Required) Name of the AKS cluster.
<code>useClusterAdmin</code> Use cluster admin credentials	(Optional) Use cluster administrator credentials instead of default cluster user credentials. This will ignore role based access control.
<code>namespace</code> Namespace	(Optional) The namespace on which the kubectl commands are to be run. If unspecified, the default namespace is used.

This YAML example shows how Azure Resource Manager is used to refer to the Kubernetes cluster. This is to be used with one of the kubectl [commands](#) and the appropriate values required by the command.

```

variables:
 azureSubscriptionEndpoint: Contoso
 azureContainerRegistry: contoso.azurecr.io
 azureResourceGroup: Contoso
 kubernetesCluster: Contoso
 useClusterAdmin: false

steps:
- task: Kubernetes@1
 displayName: kubectl apply
 inputs:
 connectionType: Azure Resource Manager
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureResourceGroup: $(azureResourceGroup)
 kubernetesCluster: $(kubernetesCluster)
 useClusterAdmin: $(useClusterAdmin)

```

## Kubernetes Service Connection

PARAMETERS	DESCRIPTION
<code>kubernetesServiceEndpoint</code> Kubernetes service connection	(Required) Select a Kubernetes service connection.
<code>namespace</code> Namespace	(Optional) The namespace on which the kubectl commands are to be run. If not specified, the default namespace is used.

This YAML example shows how a Kubernetes Service Connection is used to refer to the Kubernetes cluster. This is to be used with one of the kubectl [commands](#) and the appropriate values required by the command.

```

- task: Kubernetes@1
 displayName: kubectl apply
 inputs:
 connectionType: Kubernetes Service Connection
 kubernetesServiceEndpoint: Contoso

```

## Commands

The command input accepts one of the following [kubectl commands](#):

**apply, create, delete, exec, expose, get, login, logout, logs, run, set, or top.**

PARAMETERS	DESCRIPTION
<code>command</code> Command	(Required) Applies a configuration to a resource by filename or stdin. Default value: apply
<code>useConfigurationFile</code> Use configuration files	(Optional) Use Kubernetes configuration files with the kubectl command. Enter the filename, directory, or URL of the Kubernetes configuration files. Default value: false
<code>arguments</code> Arguments	(Optional) Arguments for the specified kubectl command.

This YAML example demonstrates the **apply** command:

```

- task: Kubernetes@1
 displayName: kubectl apply using arguments
 inputs:
 connectionType: Azure Resource Manager
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureResourceGroup: $(azureResourceGroup)
 kubernetesCluster: $(kubernetesCluster)
 command: apply
 arguments: -f mhc-aks.yaml

```

This YAML example demonstrates the use of a configuration file with the `apply` command:

```

- task: Kubernetes@1
 displayName: kubectl apply using configFile
 inputs:
 connectionType: Azure Resource Manager
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureResourceGroup: $(azureResourceGroup)
 kubernetesCluster: $(kubernetesCluster)
 command: apply
 useConfigurationFile: true
 configuration: mhc-aks.yaml

```

## Secrets

Kubernetes objects of type **secret** are intended to hold sensitive information such as passwords, OAuth tokens, and ssh keys. Putting this information in a secret is safer and more flexible than putting it verbatim in a pod definition or in a Docker image. Azure Pipelines simplifies the addition of **ImagePullSecrets** to a service account, or setting up of any generic secret, as described below.

### **ImagePullSecret**

PARAMETERS	DESCRIPTION
<code>secretType</code> Type of secret	(Required) Create or update an ImagePullSecret or any other generic secret. Acceptable values: <b>dockerRegistry</b> for ImagePullSecret or <b>generic</b> for any other type of secret. Default value: dockerRegistry
<code>containerRegistryType</code> Container registry type	(Required) Acceptable values: <b>Azure Container Registry</b> , or <b>Container Registry</b> for any other registry. Default value: Azure Container Registry
<code>azureSubscriptionEndpointForSecrets</code> Azure subscription	(Required if <code>secretType == dockerRegistry</code> and <code>containerRegistryType == Azure Container Registry</code> ) Azure Resource Manager service connection scoped to the subscription containing the Azure Container Registry for which the ImagePullSecret is to be set up.
<code>azureContainerRegistry</code> Azure container registry	(Required if <code>secretType == dockerRegistry</code> and <code>containerRegistryType == Azure Container Registry</code> ) The Azure Container Registry for which the ImagePullSecret is to be set up.
<code>secretName</code> Secret name	(Optional) Name of the secret.

PARAMETERS	DESCRIPTION
<p><code>forceUpdate</code> Force update secret</p>	(Optional) Delete the secret if it exists and create a new one with updated values. Default value: true

This YAML example demonstrates the setting up of ImagePullSecrets:

```

- task: Kubernetes@1
 displayName: kubectl apply for secretType dockerRegistry
 inputs:
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureResourceGroup: $(azureResourceGroup)
 kubernetesCluster: $(kubernetesCluster)
 command: apply
 arguments: -f mhc-aks.yaml
 secretType: dockerRegistry
 containerRegistryType: Azure Container Registry
 azureSubscriptionEndpointForSecrets: $(azureSubscriptionEndpoint)
 azureContainerRegistry: $(azureContainerRegistry)
 secretName: mysecretkey2
 forceUpdate: true

```

## Generic Secrets

PARAMETERS	DESCRIPTION
<p><code>secretType</code> Type of secret</p>	(Required) Create or update an ImagePullSecret or any other generic secret. Acceptable values: <b>dockerRegistry</b> for ImagePullSecret or <b>generic</b> for any other type of secret. Default value: dockerRegistry
<p><code>secretArguments</code> Arguments</p>	(Optional) Specify keys and literal values to insert in the secret. For example, <code>--from-literal=key1=value1 --from-literal=key2="top secret"</code>
<p><code>secretName</code> Secret name</p>	(Optional) Name of the secret.

This YAML example creates generic secrets from literal values specified for the `secretArguments` input:

```

- task: Kubernetes@1
 displayName: secretType generic with literal values
 inputs:
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureResourceGroup: $(azureResourceGroup)
 kubernetesCluster: $(kubernetesCluster)
 command: apply
 arguments: -f mhc-aks.yaml
 secretType: generic
 secretArguments: --from-literal=contoso=5678
 secretName: mysecretkey

```

Pipeline variables can be used to pass arguments for specifying literal values, as shown here:

```

- task: Kubernetes@1
 displayName: secretType generic with pipeline variables
 inputs:
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureResourceGroup: $(azureResourceGroup)
 kubernetesCluster: $(kubernetesCluster)
 command: apply
 arguments: -f mhc-aks.yaml
 secretType: generic
 secretArguments: --from-literal=contoso=$(contosovalue)
 secretName: mysecretkey

```

## ConfigMap

ConfigMaps allow you to decouple configuration artifacts from image content to maintain portability for containerized applications.

PARAMETERS	DESCRIPTION
<code>configMapName</code> ConfigMapName	(Optional) Name of the ConfigMap.
<code>forceUpdateConfigMap</code> Force update configmap	(Optional) Delete the configmap if it exists and create a new one with updated values. Default value: false
<code>useConfigMapFile</code> Use file	(Optional) Create a ConfigMap from an individual file, or from multiple files by specifying a directory. Default value: false
<code>configMapFile</code> ConfigMap File	(Required if useConfigMapFile == true) Specify a file or directory that contains the configMaps. Note that this will use the <code>--from-file</code> argument.
<code>configMapArguments</code> Arguments	(Optional) Specify keys and literal values to insert in configMap. For example, <code>--from-literal=key1=value1 --from-literal=key2="top secret"</code>

This YAML example creates a ConfigMap by pointing to a ConfigMap file:

```

- task: Kubernetes@1
 displayName: kubectl apply
 inputs:
 configMapName: myconfig
 useConfigMapFile: true
 configMapFile: src/configmap

```

This YAML example creates a ConfigMap by specifying the literal values directly as the `configMapArguments` input, and setting `forceUpdate` to true:

```

- task: Kubernetes@1
 displayName: configMap with literal values
 inputs:
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureResourceGroup: $(azureResourceGroup)
 kubernetesCluster: $(kubernetesCluster)
 command: apply
 arguments: -f mhc-aks.yaml
 secretType: generic
 secretArguments: --from-literal=contoso=$(contosovalue)
 secretName: mysecretkey4
 configMapName: myconfig
 forceUpdateConfigMap: true
 configMapArguments: --from-literal=myname=contoso

```

You can use pipeline variables to pass literal values when creating ConfigMap, as shown here:

```

- task: Kubernetes@1
 displayName: configMap with pipeline variables
 inputs:
 azureSubscriptionEndpoint: $(azureSubscriptionEndpoint)
 azureResourceGroup: $(azureResourceGroup)
 kubernetesCluster: $(kubernetesCluster)
 command: apply
 arguments: -f mhc-aks.yaml
 secretType: generic
 secretArguments: --from-literal=contoso=$(contosovalue)
 secretName: mysecretkey4
 configMapName: myconfig
 forceUpdateConfigMap: true
 configMapArguments: --from-literal=myname=$(contosovalue)

```

## Advanced

PARAMETERS	DESCRIPTION
<code>versionOrLocation</code> Version	(Optional) Explicitly choose a version of kubectl to be used, or specify the path (location) of the kubectl binary. Default value: version
<code>versionSpec</code> Version spec	(Required if <code>versionOrLocation == version</code> ) The version of the kubectl to be used. Examples: <b>1.7.0, 1.x.0, 4.x.0, 6.10.0, &gt;=6.10.0</b> Default value: 1.7.0
<code>checkLatest</code> Check for latest version	(Optional) If true, a check for the latest version of kubectl is performed. Default value: false
<code>specifyLocation</code> Specify location	(Required) Full path to the kubectl.exe file.
<code>cwd</code> Working directory	(Optional) Working directory for the Kubectl command. Default value: <code>\$(System.DefaultWorkingDirectory)</code>

PARAMETERS	DESCRIPTION
<code>outputFormat</code> Output format	(Optional) Acceptable values: <code>json</code> or <code>YAML</code> . Default value: <code>json</code> . You can leave it blank explicitly like <code>outputFormat: ''</code> to default to the kubectl's outputFormat

## Troubleshooting

### **My Kubernetes cluster is behind a firewall and I am using hosted agents. How can I deploy to this cluster?**

You can grant hosted agents access through your firewall by allowing the IP addresses for the hosted agents. For more details, see [Agent IP ranges](#)

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Kubernetes manifest task

3/26/2021 • 12 minutes to read • [Edit Online](#)

Use a Kubernetes manifest task in a build or release pipeline to bake and deploy manifests to Kubernetes clusters.

## Overview

The following list shows the key benefits of this task:

- **Artifact substitution:** The deployment action takes as input a list of container images that you can specify along with their tags and digests. The same input is substituted into the nontemplatized manifest files before application to the cluster. This substitution ensures that the cluster nodes pull the right version of the image.
- **Manifest stability:** The rollout status of the deployed Kubernetes objects is checked. The stability checks are incorporated to determine whether the task status is a success or a failure.
- **Traceability annotations:** Annotations are added to the deployed Kubernetes objects to superimpose traceability information. The following annotations are supported:
  - azure-pipelines/org
  - azure-pipelines/project
  - azure-pipelines/pipeline
  - azure-pipelines/pipelineld
  - azure-pipelines/execution
  - azure-pipelines/executionuri
  - azure-pipelines/jobName
- **Secret handling:** The `createSecret` action lets Docker registry secrets be created using Docker registry service connections. It also lets generic secrets be created using either plain-text variables or secret variables. Before deployment to the cluster, you can use the `secrets` input along with the `deploy` action to augment the input manifest files with the appropriate `imagePullSecrets` value.
- **Bake manifest:** The `bake` action of the task allows for baking templates into Kubernetes manifest files. The action uses tools such as Helm, Compose, and kustomize. With baking, these Kubernetes manifest files are usable for deployments to the cluster.
- **Deployment strategy:** Choosing the `canary` strategy with the `deploy` action leads to creation of workloads having names suffixed with "-baseline" and "-canary". The task supports two methods of traffic splitting:
  - **Service Mesh Interface:** [Service Mesh Interface \(SMI\)](#) abstraction allows configuration with service mesh providers like Linkerd and Istio. The Kubernetes Manifest task maps SMI TrafficSplit objects to the stable, baseline, and canary services during the life cycle of the deployment strategy. Canary deployments that are based on a service mesh and use this task are more accurate. This accuracy comes because service mesh providers enable the granular percentage-based split of traffic. The service mesh uses the service registry and sidecar containers that are injected into pods. This injection occurs alongside application containers to achieve the granular traffic split.
  - **Kubernetes with no service mesh:** In the absence of a service mesh, you might not get the exact percentage split you want at the request level. But you can possibly do canary deployments

by using baseline and canary variants next to the stable variant.

The service sends requests to pods of all three workload variants as the selector-label constraints are met. Kubernetes Manifest honors these requests when creating baseline and canary variants. This routing behavior achieves the intended effect of routing only a portion of total requests to the canary.

Compare the baseline and canary workloads by using either a [Manual Intervention task](#) in release pipelines or a [Delay task](#) in YAML pipelines. Do the comparison before using the promote or reject action of the task.

## Deploy action

PARAMETER	DESCRIPTION
<b>action</b> Action	(Required)  <b>deploy</b>
<b>kubernetesServiceConnection</b> Kubernetes service connection	(Required unless the task is used in a <a href="#">Kubernetes environment</a> )  The name of the <a href="#">Kubernetes service connection</a> .
<b>namespace</b> Namespace	(Required unless the task is used in a <a href="#">Kubernetes environment</a> )  The namespace within the cluster to deploy to.
<b>manifests</b> Manifests	(Required)  The path to the manifest files to be used for deployment. Each line represents a single path. A <a href="#">file-matching pattern</a> is an acceptable value for each line.
<b>containers</b> Containers	(Optional)  The fully qualified URL of the image to be used for substitutions on the manifest files. This input accepts the specification of multiple artifact substitutions in newline-separated form. Here's an example:  <pre>containers:     contosodemo.azurecr.io/foo:test1   contosodemo.azurecr.io/bar:test2</pre> In this example, all references to <code>contosodemo.azurecr.io/foo</code> and <code>contosodemo.azurecr.io/bar</code> are searched for in the image field of the input manifest files. For each match found, the tag <code>test1</code> or <code>test2</code> replaces the matched reference.
<b>imagePullSecrets</b> Image pulls secrets	(Optional)  Multiline input where each line contains the name of a Docker registry secret that has already been set up within the cluster. Each secret name is added under <b>imagePullSecrets</b> for the workloads that are found in the input manifest files.

PARAMETER	DESCRIPTION
<b>strategy</b> Strategy	(Optional)  The deployment strategy used while manifest files are applied on the cluster. Currently, <b>canary</b> is the only acceptable deployment strategy.
<b>trafficSplitMethod</b> Traffic split method	(Optional)  Acceptable values are <b>pod</b> and <b>smi</b> . The default value is <b>pod</b> .  For the value <b>smi</b> , the percentage traffic split is done at the request level by using a service mesh. A service mesh must be set up by a cluster admin. This task handles orchestration of SMI <a href="#">TrafficSplit</a> objects.  For the value <b>pod</b> , the percentage split isn't possible at the request level in the absence of a service mesh. Instead, the percentage input is used to calculate the replicas for baseline and canary. The calculation is a percentage of replicas that are specified in the input manifests for the stable variant.
<b>percentage</b> Percentage	(Required only if <b>strategy</b> is set to <b>canary</b> )  The percentage that is used to compute the number of baseline-variant and canary-variant replicas of the workloads that are contained in manifest files.  For the specified percentage input, calculate:  $(percentage \times \text{number of replicas}) / 100$  If the result isn't an integer, the mathematical floor of the result is used when baseline and canary variants are created.  For example, assume the deployment hello-world is in the input manifest file and that the following lines are in the task input:  <pre>replicas: 4 strategy: canary percentage: 25</pre> In this case, the deployments hello-world-baseline and hello-world-canary are created with one replica each. The baseline variant is created with the same image and tag as the stable version, which is the four-replica variant before deployment. The canary variant is created with the image and tag corresponding to the newly deployed changes.

PARAMETER	DESCRIPTION
<b>baselineAndCanaryReplicas</b> Baseline and canary replicas	<p>(Optional, and relevant only if <b>trafficSplitMethod</b> is set to <b>smi</b>)</p> <p>When you set <b>trafficSplitMethod</b> to <b>smi</b>, the percentage traffic split is controlled in the service mesh plane. But you can control the actual number of replicas for canary and baseline variants independently of the traffic split.</p> <p>For example, assume that the input deployment manifest specifies 30 replicas for the stable variant. Also assume that you specify the following input for the task:</p> <pre>strategy: canary trafficSplitMethod: smi percentage: 20 baselineAndCanaryReplicas: 1</pre> <p>In this case, the stable variant receives 80% of the traffic, while the baseline and canary variants each receive half of the specified 20%. But baseline and canary variants don't receive three replicas each. They instead receive the specified number of replicas, which means they each receive one replica.</p>
<b>rolloutStatusTimeout</b> Timeout for rollout status	<p>(Optional)</p> <p>The length of time (in seconds) to wait before ending watch on rollout status. Default is 0 (don't wait).</p>

The following YAML code is an example of deploying to a Kubernetes namespace by using manifest files:

```
steps:
- task: KubernetesManifest@0
 displayName: Deploy
 inputs:
 kubernetesServiceConnection: someK8sSC1
 namespace: default
 manifests: |
 manifests/deployment.yml
 manifests/service.yml
 containers: |
 foo/demo:${(tagVariable1)}
 bar/demo:${(tagVariable2)}
 imagePullSecrets: |
 some-secret
 some-other-secret
```

In the above example, the task tries to find matches for the images `foo/demo` and `bar/demo` in the image fields of manifest files. For each match found, the value of either `tagVariable1` or `tagVariable2` is appended as a tag to the image name. You can also specify digests in the containers input for artifact substitution.

#### NOTE

While you can author deploy, promote, and reject actions with YAML input related to deployment strategy, support for a Manual Intervention task is currently unavailable for build pipelines.

For release pipelines, we advise you to use actions and input related to deployment strategy in the following sequence:

1. A deploy action specified with `strategy: canary` and `percentage: $(someValue)`.
2. A Manual Intervention task so that you can pause the pipeline and compare the baseline variant with the canary variant.
3. A promote action that runs if a Manual Intervention task is resumed and a reject action that runs if a Manual Intervention task is rejected.

## Promote and reject actions

PARAMETER	DESCRIPTION
<b>action</b> Action	(Required)  <b>promote</b> or <b>reject</b>
<b>kubernetesServiceConnection</b> Kubernetes service connection	(Required)  The name of the <a href="#">Kubernetes service connection</a> .
<b>namespace</b> Namespace	(Required)  The namespace within the cluster to deploy to.
<b>manifests</b> Manifests	(Required)  The path to the manifest files to be used for deployment. Each line represents a single path. A <a href="#">file-matching pattern</a> is an acceptable value for each line.
<b>containers</b> Containers	(Optional)  The fully qualified resource URL of the image to be used for substitutions on the manifest files. The URL contosodemo.azurecr.io/helloworld:test is an example.
<b>imagePullSecrets</b> Image pull secrets	(Optional)  Multiline input where each line contains the name of a Docker registry secret that is already set up within the cluster. Each secret name is added under the <b>imagePullSecrets</b> field for the workloads that are found in the input manifest files.
<b>strategy</b> Strategy	(Optional)  The deployment strategy used in the deploy action before a promote action or reject action. Currently, <b>canary</b> is the only acceptable deployment strategy.

## Create secret action

PARAMETER	DESCRIPTION
<b>action</b> Action	(Required)  <b>createSecret</b>
<b>secretType</b> Secret type	(Required)  Acceptable values are <b>dockerRegistry</b> and <b>generic</b> . The default value is <b>dockerRegistry</b> .  If you set <b>secretType</b> to <b>dockerRegistry</b> , the <b>imagePullSecrets</b> field is created or updated in a cluster to help image pull from a private container registry.
<b>secretName</b> Secret name	(Required)  The name of the secret to be created or updated.
<b>dockerRegistryEndpoint</b> Docker registry service connection	(Required only if <b>secretType</b> is set to <b>dockerRegistry</b> )  The credentials of the specified service connection are used to create a Docker registry secret within the cluster. Manifest files under the <b>imagePullSecrets</b> field can then refer to this secret's name.
<b>secretArguments</b> Secret arguments	(Required only if <b>secretType</b> is set to <b>generic</b> )  Accepts keys and literal values to be used for creation and updating of secrets. Here's an example: <code>--from-literal=key1=value1 --from-literal=key2="top secret"</code>
<b>kubernetesServiceConnection</b> Kubernetes service connection	(Required)  The name of the <a href="#">Kubernetes service connection</a> .
<b>namespace</b> Namespace	(Required)  The cluster namespace within which to create a secret.

The following YAML code shows a sample creation of Docker registry secrets by using [Docker Registry service connection](#):

```
steps:
- task: KubernetesManifest@0
 displayName: Create secret
 inputs:
 action: createSecret
 secretType: dockerRegistry
 secretName: foobar
 dockerRegistryEndpoint: demoACR
 kubernetesServiceConnection: someK8sSC
 namespace: default
```

This YAML code shows a sample creation of generic secrets:

```

steps:
- task: KubernetesManifest@0
 displayName: Create secret
 inputs:
 action: createSecret
 secretType: generic
 secretName: some-secret
 secretArguments: --from-literal=key1=value1
 kubernetesServiceConnection: someK8sSC
 namespace: default

```

## Bake action

PARAMETER	DESCRIPTION
<b>action</b> Action	(Required) <b>bake</b>
<b>renderType</b> Render engine	(Required)  The render type used to produce the manifest files.  Acceptable values are <b>helm2</b> , <b>kompose</b> , and <b>kustomize</b> . The default value is <b>helm2</b> .
<b>helmChart</b> Helm chart	(Required only if <b>renderType</b> is set to <b>helm2</b> )  The path to the Helm chart used for baking.
<b>overrideFiles</b> Override files	(Optional, and relevant only if <b>renderType</b> is set to <b>helm2</b> )  Multiline input that accepts the path to the override files. The files are used when manifest files from Helm charts are baked.
<b>overrides</b> Override values	(Optional, and relevant only if <b>renderType</b> is set to <b>helm2</b> )  Additional override values that are used via the command-line switch <b>--set</b> when manifest files using Helm are baked.  Specify override values as key-value pairs in the format <b>key: value</b> . If you use multiple overriding key-value pairs, specify each key-value pair in a separate line. Use a newline character as the delimiter between different key-value pairs.
<b>releaseName</b> Release name	(Optional, and relevant only if <b>renderType</b> is set to <b>helm2</b> )  The name of the release used when baking Helm charts.
<b>kustomizationPath</b> Kustomization path	(Optional, and relevant only if <b>renderType</b> is set to <b>kustomize</b> )  The path to the directory containing the file <b>kustomization.yaml</b> .

PARAMETER	DESCRIPTION
<b>dockerComposeFile</b> Path to Docker compose file	(Optional, and relevant only if <code>renderType</code> is set to <code>kompose</code> )  The path to the Docker compose file.

The following YAML code is an example of baking manifest files from Helm charts. Note the usage of name input in the first task. This name is later referenced from the deploy step for specifying the path to the manifests that were produced by the bake step.

```

steps:
- task: KubernetesManifest@0
 name: bake
 displayName: Bake K8s manifests from Helm chart
 inputs:
 action: bake
 helmChart: charts/sample
 overrides: 'image.repository:nginx'

- task: KubernetesManifest@0
 displayName: Deploy K8s manifests
 inputs:
 kubernetesServiceConnection: someK8sSC
 namespace: default
 manifests: $(bake.manifestsBundle)
 containers: |
 nginx: 1.7.9

```

## Scale action

PARAMETER	DESCRIPTION
<b>action</b> Action	(Required)  <b>scale</b>
<b>kind</b> Kind	(Required)  The kind of Kubernetes object to be scaled up or down. Examples include ReplicaSet and StatefulSet.
<b>name</b> Name	(Required)  The name of the Kubernetes object to be scaled up or down.
<b>replicas</b> Replica count	(Required)  The number of replicas to scale to.
<b>kubernetesServiceConnection</b> Kubernetes service connection	(Required)  The name of the <a href="#">Kubernetes service connection</a> .
<b>namespace</b> Namespace	(Required)  The namespace within the cluster to deploy to.

PARAMETER	DESCRIPTION
<b>rolloutStatusTimeout</b> Timeout for rollout status	(Optional)  The length of time (in seconds) to wait before ending watch on rollout status. Default is 0 (don't wait).

The following YAML code shows an example of scaling objects:

```
steps:
- task: KubernetesManifest@0
 displayName: Scale
 inputs:
 action: scale
 kind: deployment
 name: bootcamp-demo
 replicas: 5
 kubernetesServiceConnection: someK8sSC
 namespace: default
```

## Patch action

PARAMETER	DESCRIPTION
<b>action</b> Action	(Required)  <b>patch</b>
<b>resourceToPatch</b> Resource to patch	(Required)  Indicates one of the following patch methods: <ul style="list-style-type: none"> <li>• A manifest file identifies the objects to be patched.</li> <li>• An individual object is identified by kind and name as the patch target.</li> </ul> Acceptable values are <b>file</b> and <b>name</b> . The default value is <b>file</b> .
<b>resourceFiletoPatch</b> File path	(Required only if <b>action</b> is set to <b>patch</b> and <b>resourceToPatch</b> is set to <b>file</b> )  The path to the file used for the patch.
<b>kind</b> Kind	(Required only if <b>resourceToPatch</b> is set to <b>name</b> )  The kind of the Kubernetes object. Examples include ReplicaSet and StatefulSet.
<b>name</b> Name	(Required only if <b>resourceToPatch</b> is set to <b>name</b> )  The name of the Kubernetes object to be patched.
<b>mergeStrategy</b> Merge strategy	(Required)  The strategy to be used for applying the patch.  Acceptable values are <b>json</b> , <b>merge</b> , and <b>strategic</b> . The default value is <b>strategic</b> .

PARAMETER	DESCRIPTION
<b>patch</b> Patch	(Required)  The contents of the patch.
<b>kubernetesServiceConnection</b> Kubernetes service connection	(Required)  The name of the <a href="#">Kubernetes service connection</a> .
<b>namespace</b> Namespace	(Required)  The namespace within the cluster to deploy to.
<b>rolloutStatusTimeout</b> Timeout for rollout status	(Optional)  The length of time (in seconds) to wait before ending watch on rollout status. Default is 0 (don't wait).

The following YAML code shows an example of object patching:

```
steps:
- task: KubernetesManifest@0
 displayName: Patch
 inputs:
 action: patch
 kind: pod
 name: demo-5fbc4d6cd9-pgxn4
 mergeStrategy: strategic
 patch: '{"spec":{"containers":[{"name":"demo","image":"foobar/demo:2239"}]}}'
 kubernetesServiceConnection: someK8sSC
 namespace: default
```

## Delete action

PARAMETER	DESCRIPTION
<b>action</b> Action	(Required)  <b>delete</b>
<b>arguments</b> Arguments	(Required)  Arguments to be passed on to kubectl for deleting the necessary objects. An example is: <div style="border: 1px solid #ccc; padding: 5px; display: inline-block;">arguments: deployment hello-world foo-bar</div>
<b>kubernetesServiceConnection</b> Kubernetes service connection	(Required)  The name of the <a href="#">Kubernetes service connection</a> .
<b>namespace</b> Namespace	(Required)  The namespace within the cluster to deploy to.

This YAML code shows a sample object deletion:

```
steps:
- task: KubernetesManifest@0
 displayName: Delete
 inputs:
 action: delete
 arguments: deployment expressapp
 kubernetesServiceConnection: someK8sSC
 namespace: default
```

## Troubleshooting

### **My Kubernetes cluster is behind a firewall and I am using hosted agents. How can I deploy to this cluster?**

You can grant hosted agents access through your firewall by allowing the IP addresses for the hosted agents. For more details, see [Agent IP ranges](#)

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# PowerShell on Target Machines task

11/2/2020 • 5 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

Use this task 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 [the latest version of the Windows Management Framework](#) installed. This is installed by default on Windows 8.1, Windows Server 2012 R2, and later.

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## 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 Prerequisites**. 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 machines using 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
<b>Machines</b>	<p>A comma-separated list of machine FQDNs or IP addresses, optionally including the port number. Can be:</p> <ul style="list-style-type: none"> <li>- The name of an <a href="#">Azure Resource Group</a>.</li> <li>- A comma-delimited list of machine names. Example: dbserver.fabrikam.com, dbserver_int.fabrikam.com:5986, 192.168.34:5</li> <li>- An output variable from a previous task.</li> </ul> <p>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.</p>
<b>Admin Login</b>	<p>The username of either a domain or a local administrative account on the target host(s).</p> <ul style="list-style-type: none"> <li>- Formats such as <code>username</code>, <code>domain\username</code>, <code>machine-name\username</code>, and <code>.\username</code> are supported.</li> <li>- UPN formats such as <code>username@domain.com</code> and built-in system accounts such as <code>NT Authority\System</code> are not supported.</li> </ul>
<b>Password</b>	<p>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: \$(passwordVariable)</p>
<b>Protocol</b>	<p>The protocol that will be used to connect to the target host, either <code>HTTP</code> or <code>HTTPS</code>.</p>
<b>Test Certificate</b>	<p>If you choose the <code>HTTPS</code> option, set this checkbox to skip validating the authenticity of the machine's certificate by a trusted certification authority.</p>

ARGUMENT	DESCRIPTION
<b>Deployment - PowerShell Script</b>	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>
<b>Deployment - Script Arguments</b>	The arguments required by the script, if any. Example: <code>-applicationPath \$(applicationPath) -username \$(vmusername) -password \$(vmpassword)</code>
<b>Deployment - Initialization Script</b>	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.
<b>Deployment - Session Variables</b>	Used to set up the session variables for the PowerShell scripts. A comma-separated list such as <code>\$varx=valuex, \$vary=valuey</code> Most commonly used for backward compatibility with earlier versions of the release service. It is recommended to use arguments instead of session variables.
<b>Advanced - Run PowerShell in Parallel</b>	Set this option to execute the PowerShell scripts in parallel on all the target machines
<b>Advanced - Select Machines By</b>	Depending on how you want to specify the machines in the group when using the <b>Filter Criteria</b> parameter, choose <b>Machine Names</b> or <b>Tags</b> .
<b>Advanced - Filter Criteria</b>	Optional. A list of machine names or tag names that identifies the machines that the task will target. The filter criteria can be: <ul style="list-style-type: none"> <li>- The name of an <a href="#">Azure Resource Group</a>.</li> <li>- An output variable from a previous task.</li> <li>- A comma-delimited list of tag names or machine names. Format when using machine names is a comma-separated list of the machine FQDNs or IP addresses.</li> </ul> Specify tag names for a filter as {TagName}:{Value} Example: <code>Role:DB;OS:Win8.1</code>
<b>Control options</b>	See <a href="#">Control options</a>

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.

## FAQ

### Do I need an agent?

You need at least one [agent](#) to run your build or release.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

**My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I**

## **fix this?**

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE.EXTRA.CA.CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

## **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

11/2/2020 • 7 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

Use this task 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

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## 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 an Azure 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: true # Optional
```

## Task Inputs

PARAMETERS	DESCRIPTION
<code>applicationPackagePath</code> Application Package	(Required) Path to the application package that is to be deployed. [Variables](../../build/variables.md) and wildcards can be used in the path
<code>serviceConnectionName</code> Cluster Service Connection	(Required) Select an Azure Service Fabric service connection to be used to connect to the cluster. The settings defined in this referenced service connection will override those defined in the publish profile. Choose 'Manage' to register a new service connection.  To connect to the cluster, the service fabric task uses the machine cert store to store the information about the certificate. Using the same certificate, if two releases run together on one machine they will start properly. However, if one of the tasks is complete, the certificate from the machine cert store would be cleaned up, which would affect the second release
<code>publishProfilePath</code> Publish Profile	(Optional) Path to the publish profile file that defines the settings to use. [Variables](../../build/variables.md) and wildcards can be used in the path. Publish profiles can be created in Visual Studio as shown <a href="#">here</a>
<code>applicationParameterPath</code> Application Parameters	(Optional) Path to the application parameters file. [Variables](../../build/variables.md) and wildcards can be used in the path. If specified, this will override the value in the publish profile. Application parameters file can be created in Visual Studio as shown <a href="#">here</a>
<code>overrideApplicationParameter</code> Override Application Parameters	(Optional) Variables defined in the build or release pipeline will be matched against the 'Parameter Name' entries in the application manifest file. Application parameters file can be created in Visual Studio as shown <a href="#">here</a>  Example: If your application has a parameter defined as below-
	<pre>&lt;Parameters&gt; &lt;Parameter Name = "SampleApp_PartitionCount" Value ="1"/&gt; &lt;Parameter Name = "SampleApp_InstanceCount" DefaultValue ="-1"/&gt; &lt;/Parameters&gt;</pre>
	and you want to change the partition count to 2, you can define a release pipeline or an environment variable "SampleApp_PartitionCount" and its value as "2". <b>Note:</b> If same variables are defined in the release pipeline and in the environment, then the environment variables will supersede the release pipeline variables Default value: false
<code>compressPackage</code> Compress Package	(Optional) Indicates whether the application package should be compressed before copying to the image store. If enabled, this will override the value in the publish profile. More information for compress package can be found <a href="#">here</a> Default value: false
<code>copyPackageTimeoutSec</code> CopyPackageTimeoutSec	(Optional) Timeout in seconds for copying application package to image store. If specified, this will override the value in the publish profile
<code>registerPackageTimeoutSec</code> RegisterPackageTimeoutSec	(Optional) Timeout in seconds for registering or un-registering application package

PARAMETERS	DESCRIPTION
<code>overwriteBehavior</code> Overwrite Behavior	(Required) Overwrite Behavior: when upgrade is not configured and an Application with same name already exists in the cluster, then following actions are available => Never, Always, SameAppTypeAndVersion. <b>Never</b> will not remove the existing Application. This is the default behavior. <b>Always</b> will remove the existing Application even if its Application type and Version is different from the Application being created. <b>SameAppTypeAndVersion</b> will remove the existing Application only if its Application type and Version is same as the Application being created Default value: SameAppTypeAndVersion
<code>skipUpgradeSameTypeAndVersion</code> Skip upgrade for same Type and Version	(Optional) Indicates whether an upgrade will be skipped if the same application type and version already exists in the cluster, otherwise the upgrade fails during validation. If enabled, re-deployments are idempotent. Default value: false
<code>skipPackageValidation</code> Skip package validation	(Optional) Indicates whether the package should be validated or not before deployment. More information about package validation can be found <a href="#">here</a> Default value: false
<code>useDiffPackage</code> Use Diff Package	(Optional) Diff package is created by task by comparing the package specified in Application Package input against the package which is currently registered in the target cluster. If a service version in cluster's current package is same as the new package, then this service package will be removed from new Application package. See more details about diff package <a href="#">here</a> . Default value: false
<code>overridePublishProfileSettings</code> Override All Publish Profile Upgrade Settings	(Optional) This will override all upgrade settings with either the values specified below or the default value if not specified. More information about upgrade settings can be found <a href="#">here</a> Default value: false
<code>isUpgrade</code> Upgrade the Application	(Optional) If false, the application will be overwritten. Default value: true
<code>unregisterUnusedVersions</code> Unregister Unused Versions	(Optional) Indicates whether all unused versions of the application type will be removed after an upgrade Default value: true
<code>upgradeMode</code> Upgrade Mode	(Required) Default value: Monitored
<code>FailureAction</code> FailureAction	(Required) Default value: Rollback
<code>UpgradeReplicaSetCheckTimeoutSec</code> UpgradeReplicaSetCheckTimeoutSec	(Optional)
<code>TimeoutSec</code> TimeoutSec	(Optional)
<code>ForceRestart</code> ForceRestart	(Optional) Default value: false
<code>HealthCheckRetryTimeoutSec</code> HealthCheckRetryTimeoutSec	(Optional)
<code>HealthCheckWaitDurationSec</code> HealthCheckWaitDurationSec	(Optional)

PARAMETERS	DESCRIPTION
<code>HealthCheckStableDurationSec</code> HealthCheckStableDurationSec	(Optional)
<code>UpgradeDomainTimeoutSec</code> UpgradeDomainTimeoutSec	(Optional)
<code>ConsiderWarningAsError</code> ConsiderWarningAsError	(Optional) Default value: false
<code>DefaultServiceTypeHealthPolicy</code> DefaultServiceTypeHealthPolicy	(Optional)
<code>MaxPercentUnhealthyDeployedApplications</code> MaxPercentUnhealthyDeployedApplications	(Optional)
<code>UpgradeTimeoutSec</code> UpgradeTimeoutSec	(Optional)
<code>ServiceTypeHealthPolicyMap</code> ServiceTypeHealthPolicyMap	(Optional)
<code>configureDockerSettings</code> Configure Docker settings	Default value: false
<code>registryCredentials</code> Registry Credentials Source	(Required) Choose how credentials for the Docker registry will be provided Default value: AzureResourceManagerEndpoint
<code>dockerRegistryEndpoint</code> Docker Registry Service Connection	(Required) Select a Docker registry service connection. Required for commands that need to authenticate with a registry. <b>Note:</b> task will try to encrypt the registry secret before transmitting it to service fabric cluster. However, it needs cluster's server certificate to be installed on agent machine in order to do so. If certificate is not present, secret will not be encrypted
<code>azureSubscriptionEndpoint</code> Azure subscription	(Required) Select an Azure subscription. <b>Note:</b> task will try to encrypt the registry secret before transmitting it to service fabric cluster. However, it needs cluster's server certificate to be installed on agent machine in order to do so. If certificate is not present, secret will not be encrypted
<code>registryUserName</code> Registry User Name	(Optional) Username for the Docker registry
<code>registryPassword</code> Registry Password	(Optional) Password for the Docker registry. If the password is not encrypted, it is recommended that you use a custom release pipeline secret variable to store it
<code>passwordEncrypted</code> Password Encrypted	(Optional) It is recommended to encrypt your password using [Invoke-ServiceFabricEncryptText](/azure/service-fabric/service-fabric-application-secret-management#encrypt-application-secrets). If you do not, and a certificate matching the Server Certificate Thumbprint in the Cluster Service Connection is installed on the build agent, it will be used to encrypt the password; otherwise an error will occur

Also see: [Update Service Fabric Manifests task](#)

## Arguments

ARGUMENT	DESCRIPTION
<b>Publish Profile</b>	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/**/PublishProfile</code>
<b>Application Package</b>	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>
<b>Cluster Connection</b>	The name of the Azure Service Fabric service connection defined in the TS/TFS project that describes the connection to the cluster.
<b>Control options</b>	See <a href="#">Control options</a>

Also see: [Update Service Fabric App Versions task](#)

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Service Fabric Compose Deploy task

6/2/2020 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task 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 [Docker Compose deployment support in Azure Service Fabric](#).

### Service Fabric

- This task uses a Service Fabric installation to connect and deploy to a Service Fabric cluster.
- Download and install [Azure Service Fabric Core SDK](#) on the build agent.

## YAML snippet

```
Service Fabric Compose deploy
Deploy a Docker Compose application to an Azure 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: true # Optional
 #upgrade: # Optional
 #deployTimeoutSec: # Optional
 #removeTimeoutSec: # Optional
 #getStatusTimeoutSec: # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<b>Cluster Connection</b>	The Azure Service Fabric service connection to use to connect and authenticate to the cluster.
<b>Compose File Path</b>	Path to the compose file that is to be deployed. Can include wildcards and variables. Example: \$(System.DefaultWorkingDirectory)/**/drop/projectartifacts/**/dock compose.yml . Note: combining compose files is not supported as part of this task.
<b>Application Name</b>	The Service Fabric Application Name of the application being deployed. Use fabric:/ 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><b>Azure Resource Manager Endpoint:</b> 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><b>Container Registry Endpoint:</b> 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><b>Username and Password:</b> Username and password to be used. We recommend you encrypt your password using <a href="#">Invoke-ServiceFabricEncryptText</a> (Check <b>Password Encrypted</b>). 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><b>None:</b> 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 <a href="#">Control options</a>

Also see: [Service Fabric PowerShell Utility](#)

## Open source

This task is open source on [GitHub](#). Feedback and contributions are welcome.

## FAQ

### Do I need an agent?

You need at least one [agent](#) to run your build or release.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE.EXTRA.CA.CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

# SSH Deployment task

11/2/2020 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

Use this task 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

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## 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
 #interpreterCommand: # Used when runOptions == Inline
 #args: # Optional
 #failOnStdErr: true # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<b>SSH endpoint</b>	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"><li>- The private key and the passphrase must be specified for authentication.</li><li>- A password can be used to authenticate to remote Linux machines, but this is not supported for macOS or Windows systems.</li></ul>
<b>Run</b>	Choose to run either shell commands or a shell script on the remote machine.

ARGUMENT	DESCRIPTION
<b>Commands</b>	<p>The shell commands to run on the remote machine. This parameter is available only when <b>Commands</b> is selected for the <b>Run</b> 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:</p> <pre>cd /home/user/myFolder;build</pre> <p><b>NOTE:</b> Each command runs in a separate process. If you want to run a series of commands that are interdependent (for example, changing the current folder before executing a command) use the <b>Inline Script</b> option instead.</p>
<b>Shell script path</b>	Path to the shell script file to run on the remote machine. This parameter is available only when <b>Shell script</b> is selected for the <b>Run</b> option.
<b>Interpreter command</b>	Path to the command interpreter used to execute the script. Used when <b>Run</b> option = <b>Inline</b> . Adds a shebang line to the beginning of the script. Relevant only for UNIX-like operating systems. Please use empty string for Windows-based remote hosts. <a href="#">See more about shebang (#!)</a>
<b>Arguments</b>	The arguments to pass to the shell script. This parameter is available only when <b>Shell script</b> is selected for the <b>Run</b> option.
<b>Advanced - Fail on STDERR</b>	If this option is selected (the default), the build will fail if the remote commands or script write to <b>STDERR</b> .
<b>Control options</b>	See <a href="#">Control options</a>

## Supported algorithms

### Key pair algorithms

- RSA
- DSA

### Encryption algorithms

- aes256-cbc
- aes192-cbc
- aes128-cbc
- blowfish-cbc
- 3des-cbc
- arcfour256
- arcfour128
- cast128-cbc
- arcfour

For OpenSSL v1.0.1 and higher (on agent):

- aes256-ctr
- aes192-ctr

- aes128-ctr

For OpenSSL v1.0.1 and higher, NodeJS v0.11.12 and higher (on agent):

- aes128-gcm
- aes128-gcm@openssh.com
- aes256-gcm
- aes256-gcm@openssh.com

## 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.

## FAQ

### What key formats are supported for the SSH tasks?

The Azure Pipelines SSH tasks use the Node.js `ssh2` package for SSH connections. Ensure that you are using the latest version of the SSH tasks. Older versions may not support the OpenSSH key format.

If you run into an "Unsupported key format" error, then you may need to add the `-m PEM` flag to your `ssh-keygen` command so that the key is in a supported format.

### Do I need an agent?

You need at least one [agent](#) to run your build or release.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

### 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

11/2/2020 • 3 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

Use this task 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

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

## YAML snippet

```
Windows machine file copy
Copy files to remote Windows machines
- 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"><li>- The name of an <a href="#">Azure Resource Group</a>.</li><li>- A comma-delimited list of machine names. Example: <code>dbserver.fabrikam.com, dbserver_int.fabrikam.com:5986,192.168.34:5986</code></li><li>- An output variable from a previous task.</li></ul>

ARGUMENT	DESCRIPTION
Admin Login	The username of either a domain or a local administrative account on the target host(s). - Formats such as <b>domain\username</b> , <b>username</b> , and <b>machine-name\username</b> are supported. - UPN formats such as <b>username@domain.com</b> and built-in system accounts such as <b>NT Authority\System</b> 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 <b>Filter Criteria</b> parameter, choose <b>Machine Names</b> or <b>Tags</b> .
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 <a href="#">Azure Resource Group</a> . - 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 FQDNs or IP addresses. Specify tag names for a filter as {TagName}:{Value} Example: <code>Role:DB;OS:Win8.1</code>
Control options	See <a href="#">Control options</a>

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### 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.

**I'm having problems. How can I troubleshoot them?**

See [Troubleshoot Build and Release](#).

**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](#).

**My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?**

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE.EXTRA.CA.CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

**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

4/10/2020 • 3 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task to deploy to SQL Server Database using a DACPAC or SQL script.

## YAML snippet

```
SQL Server database deploy
Deploy a 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 ==
 SqlCommand
 #databaseName: # Required when targetMethod == Server || TaskType == SqlCommand || TaskType == SqlCommand
 #authScheme: 'windowsAuthentication' # Required when targetMethod == Server || TaskType == SqlCommand || TaskType ==
 SqlCommand# 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, <code>/*.dacpac</code> 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, /* .sql 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=sqluser;Password=placeholderpassword;"
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.
<b>CONTROL OPTIONS</b>	

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# MySQL Database Deployment on Machine Group task

4/10/2020 • 2 minutes to read • [Edit Online](#)

Use this task to run your scripts and make changes to your MySQL Database. There are two ways to deploy, either using a script file or writing the script in our inline editor. Note that this is an early preview version. Since this task is server based, it appears on Deployment group jobs.

## Prerequisites

- MySQL Client in agent box

The task expects MySQL client must be in agent box.

- Windows Agent:** Use this [script file](#) to install MySQL client
- Linux Agent:** Run command 'apt-get install mysql-client' to install MySQL client

## Task Inputs

PARAMETERS	DESCRIPTION
<b>TaskNameSelector</b> Deploy MySql Using	Select one of the options between Script File & Inline Script. Default value: SqlTaskFile
<b>SqlFile</b> 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.
<b>SqlInline</b> Inline MySQL Script	(Required) MySQL script to execute on the Database
<b>ServerName</b> Host Name	(Required) Server name of Database for MySQL. Example: localhost. When you connect using MySQL Workbench, this is the same value that is used for 'Hostname' in 'Parameters'. Default value: localhost
<b>DatabaseName</b> Database Name	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.
<b>SqlUsername</b> Mysql User Name	(Required) When you connect using MySQL Workbench, this is the same value that is used for 'Username' in 'Parameters'
<b>SqlPassword</b> Password	(Required) Password for MySQL Database. It can be variable defined in the pipeline. Example : \$(password). Mark the variable type as 'secret' to secure it.

PARAMETERS	DESCRIPTION
<b>SqlAdditionalArguments</b> Additional Arguments	Additional options supported by MySQL simple SQL shell. These options will be applied when executing the given file on the Database 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.

## Example

This example creates a sample db in MySQL.

```

steps:
- task: MysqlDeploymentOnMachineGroup@1
 displayName: 'Deploy Using : InlineSqlTask'
 inputs:
 TaskNameSelector: InlineSqlTask
 SqlInline: |
 CREATE DATABASE IF NOT EXISTS alm;
 use alm;
 ServerName: localhost
 SqlUsername: root
 SqlPassword: P2ssw0rd

```

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Docker Installer task

4/10/2020 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task to install a specific version of the Docker CLI on the agent machine.

## Task Inputs

PARAMETERS	DESCRIPTION
<code>dockerVersion</code> Docker Version	(Required) Specify the version of Docker CLI to install. Default value: 17.09.0-ce
<code>releaseType</code> Release type	(Optional) Select the release type to install. 'Nightly' is not supported on Windows. Default value: stable

This YAML example installs the Docker CLI on the agent machine:

```
- task: DockerInstaller@0
 displayName: Docker Installer
 inputs:
 dockerVersion: 17.09.0-ce
 releaseType: stable
```

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Go Tool Installer task

4/21/2020 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task 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
Find in cache or download a specific version of Go and add it to the PATH
- task: GoTool@0
 inputs:
 #version: '1.10'
 #goPath: # Optional
 #goBin: # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<code>version</code> Version	(Required) Go tool version to download and install. <b>Example:</b> 1.9.3 Default value: <code>1.10</code>
<code>goPath</code> GOPATH	(Optional) Value for the GOPATH environment variable.
<code>goBin</code> GOBIN	(Optional) Value for the GOBIN environment variable.

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Helm installer task

2/28/2020 • 2 minutes to read • [Edit Online](#)

This task can be used for installing a specific version of helm binary on agents.

## YAML snippet

```
Helm tool installer
Install Helm on an agent machine
- task: HelmInstaller@1
 inputs:
 helmVersionToInstall: 'latest' # Optional
```

## Task inputs

PARAMETERS	DESCRIPTION
<code>helmVersionToInstall</code> Helm Version Spec	(Optional) The version of Helm to be installed on the agent. Acceptable values are <code>latest</code> or any semantic version string like <code>2.14.1</code> Default value: <code>latest</code>

The following YAML example showcases the installation of latest version of helm binary on the agent -

```
- task: HelmInstaller@1
 displayName: Helm installer
 inputs:
 helmVersionToInstall: latest
```

The following YAML example demonstrates the use of an explicit version string rather than installing the latest version available at the time of task execution -

```
- task: HelmInstaller@1
 displayName: Helm installer
 inputs:
 helmVersionToInstall: 2.14.1
```

## Troubleshooting

### **HelmInstaller task running on a private agent behind a proxy fails to download helm package.**

The HelmInstaller task does not use the proxy settings to download the file <https://get.helm.sh/helm-v3.1.0-linux-amd64.zip>. You can work around this by pre-installing Helm on your private agents.

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Java Tool Installer task

3/6/2021 • 4 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task 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
Acquire a specific version of Java from a user-supplied Azure blob or the tool cache and sets JAVA_HOME
- 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
<code>versionSpec</code> JDK Version	(Required) Specify which JDK version to download and use. Default value: 8
<code>jdkArchitectureOption</code> JDK Architecture	(Required) Specify the bit version of the JDK. Options: x64, x86
<code>jdkSourceOption</code> JDK source	(Required) Specify the source for the compressed JDK, either Azure blob storage or a local directory on the agent or source repository or use the pre-installed version of Java (available for Microsoft-hosted agents). Please see example below about how to use pre-installed version of Java

ARGUMENT	DESCRIPTION
<code>jdkFile</code> JDK file	(Required) Applicable when <code>jdkSourceOption == LocalDirectory</code> . Specify the path to the jdk archive file that contains the compressed JDK. The path could be in your source repository or a local path on the agent. The file should be an archive (.zip, .tar.gz, .7z), containing bin folder either on the root level or inside a single directory. For macOS - there's support of .pkg and .dmg files containing only one .pkg file inside.
<code>azureResourceManagerEndpoint</code> Azure Subscription	(Required) Applicable when <code>jdkSourceOption == AzureStorage</code> . Specify the Azure Resource Manager subscription for the JDK.
<code>azureStorageAccountName</code> Storage Account Name	(Required) Applicable when <code>jdkSourceOption == AzureStorage</code> . Specify the Storage account name in which the JDK is located. Azure Classic and Resource Manager storage accounts are listed.
<code>azureContainerName</code> Container Name	(Required) Applicable when <code>jdkSourceOption == AzureStorage</code> . Specify the name of the container in the storage account in which the JDK is located.
<code>azureCommonVirtualFile</code> Common Virtual Path	(Required) Applicable when <code>jdkSourceOption == AzureStorage</code> . Specify the path to the JDK inside the Azure storage container.
<code>jdkDestinationDirectory</code> Destination directory	(Required) Specify the destination directory into which the JDK should be installed (only for Windows and Linux). On macOS, this directory is used as a temporary folder for extracting of .dmg's since macOS doesn't support installing of JDK to specific directory.
<code>cleanDestinationDirectory</code> Clean destination directory	(Required) Select this option to clean the destination directory before the JDK is extracted into it. Default value: <code>true</code>

#### NOTE

To run **Java Tool Installer** task on macOS it is required for user under which agent is running to have permission to execute **sudo** command without a password. You can follow the next steps to enable this permission:

1. Run `sudo visudo` command, it will open sudoers file for editing.
2. Go to the bottom of the file and add the following line: `user ALL=NOPASSWD: /usr/sbin/installer` (Replace 'user' by the actual user alias)
3. Save and close the file.

## Examples

Here's an example of getting the archive file from a local directory on Linux. The file should be an archive (.zip, .gz) of the `JAVA_HOME` directory so that it includes the `bin`, `lib`, `include`, `jre`, etc. directories.

```
- task: JavaToolInstaller@0
inputs:
 versionSpec: "11"
 jdkArchitectureOption: x64
 jdkSourceOption: LocalDirectory
 jdkFile: "/builds/openjdk-11.0.2_linux-x64_bin.tar.gz"
 jdkDestinationDirectory: "/builds/binaries/externals"
 cleanDestinationDirectory: true
```

Here's an example of downloading the archive file from Azure Storage. The file should be an archive (.zip, .gz) of the `JAVA_HOME` directory so that it includes the `bin`, `lib`, `include`, `jre`, etc. directories.

```
- task: JavaToolInstaller@0
inputs:
 versionSpec: '6'
 jdkArchitectureOption: 'x64'
 jdkSourceOption: AzureStorage
 azureResourceManagerEndpoint: myARMServiceConnection
 azureStorageAccountName: myAzureStorageAccountName
 azureContainerName: myAzureStorageContainerName
 azureCommonVirtualFile: 'jdk1.6.0_45.zip'
 jdkDestinationDirectory: '$(agent.toolsDirectory)/jdk6'
 cleanDestinationDirectory: false
```

Here's an example of using "pre-installed" feature. This feature allows you to use Java versions that are pre-installed on the Microsoft-hosted agent. You can find available pre-installed versions of Java in [Software section](#).

```
- task: JavaToolInstaller@0
inputs:
 versionSpec: '8'
 jdkArchitectureOption: 'x86'
 jdkSourceOption: 'PreInstalled'
```

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### 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.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE_EXTRA_CA_CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

# Kubectl installer task

2/26/2020 • 2 minutes to read • [Edit Online](#)

This task can be used for installing a specific version of kubectl binary on agents.

## YAML snippet

```
Kubectl tool installer
Install Kubectl on agent machine
- task: KubectlInstaller@0
 inputs:
 #kubectlVersion: 'latest' # Optional
```

## Task inputs

PARAMETERS	DESCRIPTION
<code>kubectlVersion</code> Kubectl version spec	(Optional) The version of kubectl to be installed on the agent. Acceptable values are <code>latest</code> or any semantic version string like <code>1.15.0</code> Default value: <code>latest</code>

The following YAML example showcases the installation of latest version of kubectl binary on the agent -

```
- task: KubectlInstaller@0
 displayName: Kubectl installer
 inputs:
 kubectlVersion: latest
```

The following YAML example demonstrates the use of an explicit version string rather than installing the latest version available at the time of task execution -

```
- task: KubectlInstaller@0
 displayName: Kubectl installer
 inputs:
 kubectlVersion: 1.15.0
```

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Node.js Tool Installer task

3/6/2021 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

### Build

Use this task 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'
 #force32bit: false # Optional
 #checkLatest: false # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<code>versionSpec</code> Version Spec	(Required) Specify which <a href="#">Node.js</a> version you want to use, using semver's version range syntax. <b>Examples:</b> <code>7.x</code> , <code>6.x</code> , <code>6.10.0</code> , <code>&gt;=6.10.0</code> <b>Default value:</b> <code>6.x</code>
<code>force32bit</code> Use 32 bit version on x64 agents	(Optional) Install the x86 version of Node.js on a 64-bit Windows agent. Only works on Windows.
<code>checkLatest</code> Check for Latest Version	(Optional) 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 <a href="#">self-hosted agent</a> and you want to always use the latest <code>6.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.

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# FAQ

## 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.

## I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

## 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](#).

## My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE.EXTRA.CA.CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

# NuGet Tool Installer task

3/6/2021 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

### Build

Use this task to find, download, and cache a specified version of [NuGet](#) and add it to the PATH. For information on the tools cache, see the [azure-pipelines-tool-lib](#) repo.

## 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@1
 inputs:
 #versionSpec: # Optional
 #checkLatest: false # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<code>versionSpec</code> Version Spec	A version or version range that specifies the NuGet version to make available on the path. Use x as a wildcard. See the <a href="#">list of available NuGet versions</a> . If you want to match a pre-release version, the specification must contain a major, minor, patch, and pre-release version from the list above. Examples: 5.x, 5.4.x, 5.3.1, >=5.0.0-0. If unspecified, a version will be chosen automatically
<code>checkLatest</code> Always check for new versions	Always check for and download the latest available version of <code>NuGet.exe</code> which satisfies the version spec. Enabling this option could cause unexpected build breaks when a new version of NuGet is released

### 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.

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# FAQ

## 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.

## I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

## 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](#).

## My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE.EXTRA.CA.CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

# Use .NET Core task

5/24/2021 • 3 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task to acquire a specific version of .NET Core from the Internet or the tools cache and add it to the PATH.

You can also use this task to change the version of .NET Core used in subsequent tasks like [.NET Core cli task](#).

One other reason to use tool installer is if you want to decouple your pipeline from our update cycles to help avoid a pipeline run being broken due to a change we make to our agent software.

## What's New

- Support for installing multiple versions side by side.
- Support for patterns in version to fetch latest in minor/major version. For example, you can now specify 3.1.x to get the latest patch.
- Perform Multi-level lookup. This input is only applicable to Windows based agents. It configures the .NET Core's host process behavior for looking for a suitable shared framework on the machine. For more information, see [Multi-level SharedFX Lookup](#).
- Installs NuGet version 4.4.1 and sets up proxy configuration if present in NuGet config.

## Task Inputs

PARAMETERS	DESCRIPTION
<code>packageType</code> Package to install	Please select whether to install only runtime or SDK Default value: sdk
<code>useGlobalJson</code> Use global json	Select this option to install all SDKs from global.json files. These files are searched from <b>system.DefaultWorkingDirectory</b> . You can change the search root path by setting working directory input
<code>workingDirectory</code> Working Directory	Specify path from where global.json files should be searched when using 'Use global json'. If empty, <b>system.DefaultWorkingDirectory</b> will be considered as the root path
<code>version</code> Version	Specify version of .NET Core SDK or runtime to install. Versions can be given in the following formats <ul style="list-style-type: none"><li>• 2.x =&gt; Install latest in major version.</li><li>• 3.1.x =&gt; Install latest in major and minor version</li><li>• 3.1.402 =&gt; Install exact version</li></ul>
	Find the value of <code>version</code> for installing SDK/Runtime, from the releases.json. The link to releases.json of that major:minor version can be found in <a href="#">releases-index file</a> . Like link to releases.json for 3.1 version is <a href="https://dotnetcli.blob.core.windows.net/dotnet/release-metadata/3.1/releases.json">https://dotnetcli.blob.core.windows.net/dotnet/release-metadata/3.1/releases.json</a>

PARAMETERS	DESCRIPTION
<b>includePreviewVersions</b> Include Preview Versions	Select if you want preview versions to be included while searching for latest versions, such as while searching 3.1.x. This setting is ignored if you specify an exact version, such as: 3.0.100-preview3-010431 Default value: false
<b>installationPath</b> Path To Install .NET Core	Specify where .NET Core SDK/Runtime should be installed. Different paths can have the following impact on .NET's behavior. <ul style="list-style-type: none"> <li>• \$(Agent.ToolsDirectory): This makes the version to be cached on the agent since this directory is not cleaned up across pipelines. All pipelines running on the agent, would have access to the versions installed previously using the agent.</li> <li>• \$(Agent.TempDirectory): This can ensure that a pipeline doesn't use any cached version of .NET core since this folder is cleaned up after each pipeline.</li> <li>• Any other path: You can configure any other path given the agent process has access to the path. This will change the state of the machine and impact all processes running on it.</li> </ul> Note that you can also configure Multi-Level Lookup setting which can configure .NET host's probing for a suitable version. Default value: \$(Agent.ToolsDirectory)/dotnet
<b>performMultiLevelLookup</b> Perform Multi Level Lookup	This input is only applicable to Windows based agents. This configures the behavior of .NET host process for looking up a suitable shared framework. <ul style="list-style-type: none"> <li>• false: (default) Only versions present in the folder specified in this task would be looked by the host process.</li> <li>• true: The host will attempt to look in pre-defined global locations using multi-level lookup.</li> </ul> The default global locations are: <b>For Windows:</b> C:/Program Files/dotnet (64-bit processes) C:/Program Files (x86)/dotnet (32-bit process) You can read more about it <a href="#">HERE</a>

This YAML example installs version 3.1.402 of .NET Core.

```

steps:
- task: UseDotNet@2
 displayName: 'Use .NET Core sdk'
 inputs:
 packageType: sdk
 version: 3.1.402
 installationPath: $(Agent.ToolsDirectory)/dotnet

```

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

# Use Python Version task

3/8/2021 • 3 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task 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 [FAQ](#)).

This task will fail if no Python versions are found in Agent.ToolsDirectory. Available Python versions on Microsoft-hosted agents can be found [here](#).

### NOTE

x86 and x64 versions of Python are available on Microsoft-hosted Windows agents, but not on Linux or macOS agents.

## YAML snippet

```
Use Python version
Use the specified version of Python from the tool cache, optionally adding it to the PATH
- task: UsePythonVersion@0
 inputs:
 #versionSpec: '3.x'
 #addToPath: true
 #architecture: 'x64' # Options: x86, x64 (this argument applies only on Windows agents)
```

## Arguments

ARGUMENT	DESCRIPTION
<code>versionSpec</code> Version spec	(Required) Version range or exact version of a Python version to use. Default value: <code>3.x</code>
<code>addToPath</code> Add to PATH	(Required) 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. Default value: <code>true</code>
<code>architecture</code> Architecture	(Required) The target architecture (x86, x64) of the Python interpreter. x86 is supported only on Windows. Default value: <code>x64</code>

As of version 0.150 of the task, version spec will also accept `pypy2` or `pypy3`.

If the task completes successfully, the task's output variable will contain the directory of the Python installation:

The screenshot shows the 'Use Python Version (Preview)' task configuration. At the top, there are links for 'Link settings', 'View YAML', and 'Remove'. Below that, the 'Display name' is set to 'Use Python Version'. The 'Version spec' is set to '\$(python.version)'. A checked checkbox 'Add to PATH' is present. Under 'Control Options', a red box highlights the 'Output Variables' section, which includes a 'Reference name' field and a 'Variables list' entry for '.pythonLocation'.

## Remarks

After running this task with "Add to PATH," the `python` command in subsequent scripts will be for the highest available version of the interpreter matching the version spec and architecture.

The versions of Python installed on the Microsoft-hosted Ubuntu and macOS images follow the symlinking structure for Unix-like systems defined in [PEP 394](#). For example, for Python 3.7, `python3.7` is the actual interpreter. `python3` is symlinked to that interpreter, and `python` is a symlink to that symlink.

On the Microsoft-hosted Windows images, the interpreter is just `python`.

For Microsoft-hosted agents, x86 is supported only on Windows. This is because Windows can run executables compiled for the x86 architecture with the WoW64 subsystem. Hosted Ubuntu and Hosted macOS run 64-bit operating systems and run only 64-bit Python.

## Open source

This task is open source [on GitHub](#). Feedback and contributions are welcome.

## FAQ

### 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.

### I'm having problems. How can I troubleshoot them?

See [Troubleshoot Build and Release](#).

### 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](#).

### My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE.EXTRA.CA.CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

### How can I configure a self-hosted agent to use this task?

The desired Python version will have to be added to the tool cache on the self-hosted agent in order for the task to use it. Normally the tool cache is located under the `_work/_tool` directory of the agent or the path can be overridden by the environment variable `AGENT_TOOLSDIRECTORY`. Under that directory, create the following directory structure based off of your Python version:

```
$AGENT_TOOLSDIRECTORY/
 Python/
 {version number}/
 {platform}/
 {tool files}
 {platform}.complete
```

The `version number` should follow the format of `1.2.3`. The `platform` should either be `x86` or `x64`. The `tool files` should be the unzipped Python version files. The `{platform}.complete` should be a 0 byte file that looks like `x86.complete` or `x64.complete` and just signifies the tool has been installed in the cache properly.

As a complete, concrete example, here is how a completed download of Python 3.6.4 for x64 would look in the tool cache:

```
$AGENT_TOOLSDIRECTORY/
 Python/
 3.6.4/
 x64/
 {tool files}
 x64.complete
```

For more details on the tool cache, look [here](#).

In order that your scripts may work as they would on Microsoft-hosted agents, we recommend following the symlink structure from [PEP 394](#) on Unix-like systems.

Also note that the embeddable ZIP release of Python requires [extra effort to configure for installed modules](#), including `pip`. If possible, we recommend using the [full installer](#) to get a `pip`-compatible Python installation.

# Use Ruby Version task

6/2/2020 • 2 minutes to read • [Edit Online](#)

## Azure Pipelines

Use this task 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 [FAQ](#)).

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
Use the specified version of Ruby from the tool cache, optionally adding it to the PATH
- task: UseRubyVersion@0
 inputs:
 #versionSpec: '>= 2.4'
 #addToPath: true # Optional
```

## Arguments

ARGUMENT	DESCRIPTION
<code>versionSpec</code> Version spec	(Required) Version range or exact version of a Ruby version to use. Default value: <code>&gt;= 2.4</code>
<code>addToPath</code> Add to PATH	(Optional) 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. Default value: <code>true</code>

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.

## FAQ

### 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.

### **I'm having problems. How can I troubleshoot them?**

See [Troubleshoot Build and Release](#).

### **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](#).

### **My NuGet push task is failing with the following error: "Error: unable to get local issuer certificate". How can I fix this?**

This can be fixed by adding a trusted root certificate. You can either add the `NODE_EXTRA_CA_CERTS=file` environment variable to your build agent, or you can add the `NODE.EXTRA.CA.CERTS=file` task variable in your pipeline. See [Environment variables](#) for more details.

### **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

11/2/2020 • 2 minutes to read • [Edit Online](#)

## Azure DevOps Services | TFS 2018 Update 2

Use this task 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
Acquire the test platform from nuget.org or the tool 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
```

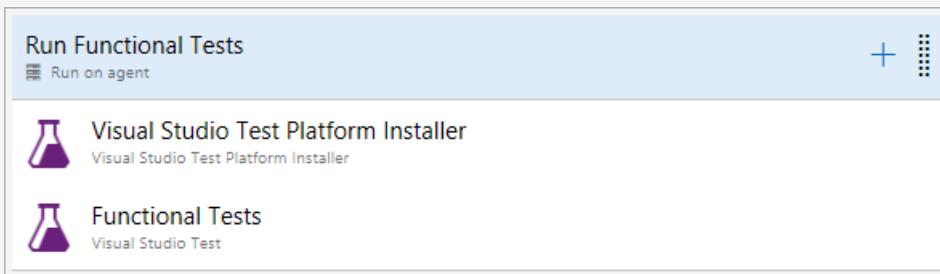
## Arguments

ARGUMENT	DESCRIPTION
<code>packageFeedSelector</code> Package Feed	(Required) Can be: <b>Official NuGet</b> - Use this option to acquire the <a href="#">test platform package from NuGet</a> . This option requires internet connectivity on the agent machine. <b>Custom feed</b> - Use this option to acquire the test platform package from a custom feed or a package management feed in Azure DevOps or TFS. <b>Network path</b> - 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. Default value: <code>nugetOrg</code>

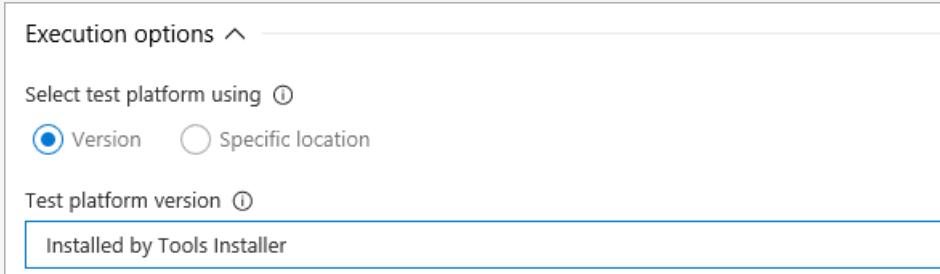
ARGUMENT	DESCRIPTION
<p><code>versionSelector</code> Version</p>	<p>(Required) Pick whether to install the latest version or a specific version of the Visual Studio Test Platform. If you use the test platform installer to run Coded UI tests, ensure that the version you choose matches the major version of Visual Studio with which the test binaries were built.</p> <p>For e.g., if the Coded UI test project was built using Visual Studio 2017 (version 15.x), you must use test platform version 15.x.</p> <p>Options:</p> <p><code>latestPreRelease, latestStable, specificVersion</code></p> <p>Default value: <code>latestPreRelease</code></p>
<p><code>testPlatformVersion</code> Test Platform Version</p>	<p>(Required) Specify the version of Visual Studio Test Platform to install on the agent. Available versions can be viewed on <a href="#">NuGet</a>.</p>
<p><code>customFeed</code> Package Source</p>	<p>(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.</p>
<p><code>username</code> User Name</p>	<p>(Optional) Specify the user name to authenticate with the feed specified in the <b>Package Source</b> argument. If using a personal access token (PAT) in the password argument, this input is not required.</p>
<p><code>password</code> Password</p>	<p>(Optional) Specify the password or personal access token (PAT) to authenticate with the feed specified in the <b>Package Source</b> argument.</p>
<p><code>netShare</code> UNC Path</p>	<p>(Required) Specify the full UNC path to the Microsoft.TestPlatform.nupkg file. The desired version of Microsoft.TestPlatform.nupkg must be downloaded from <a href="#">NuGet</a> and placed on a network share that the build/release agent can access.</p>

#### 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.

# Troubleshoot pipeline runs

3/18/2021 • 20 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

This topic provides general troubleshooting guidance. For specific troubleshooting about .NET Core, see [.NET Core troubleshooting](#).

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

You can use the following troubleshooting sections to help diagnose issues with your pipeline. Most pipeline failures fall into one of these categories.

- [Pipeline won't trigger](#)
- [Pipeline queues but never gets an agent](#)
- [Pipeline fails to complete](#)

## Pipeline won't trigger

If a pipeline doesn't start at all, check the following common trigger related issues.

- [UI settings override YAML trigger setting](#)
- [Pull request triggers not supported with Azure Repos](#)
- [Branch filters misconfigured in CI and PR triggers](#)
- [Scheduled trigger time zone conversions](#)
- [UI settings override YAML scheduled triggers](#)

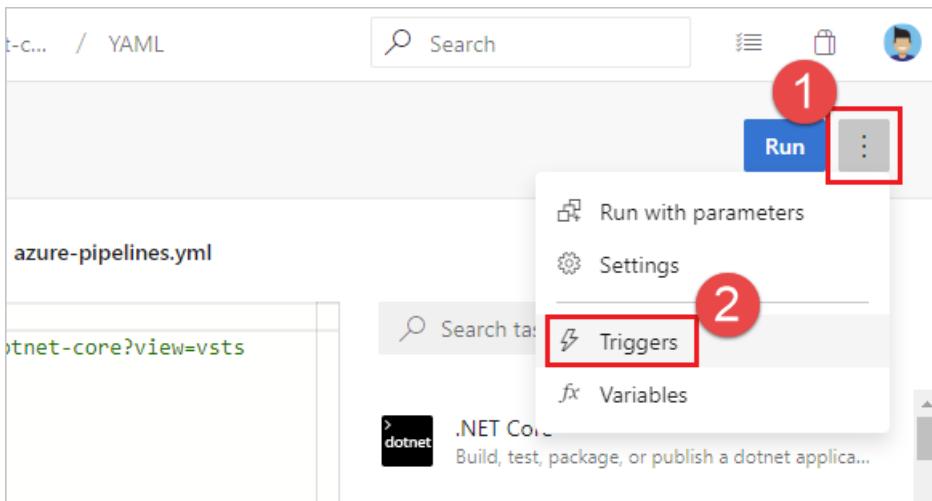
## NOTE

An additional reason that runs may not start is that your organization goes dormant five minutes after the last user signs out of Azure DevOps. 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 organization will run only one night until someone signs in again.
- CI builds of an Other Git repo will stop running until someone signs in again.

### UI settings override YAML trigger setting

YAML pipelines can have their `trigger` and `pr` trigger settings overridden in the pipeline settings UI. If your `trigger` or `pr` triggers don't seem to be firing, check that setting. While editing your pipeline, choose ... and then Triggers.



Check the **Override the YAML trigger from here** setting for the types of trigger (Continuous integration or Pull request validation) available for your repo.

This screenshot shows the 'Triggers' settings page for a repository named 'steved0x/pipelines-dotnet-core'. The 'Continuous integration' section shows a trigger for 'steved0x/pipelines-dotnet-core' which is 'Enabled'. The 'Pull request validation' section also shows a trigger for 'steved0x/pipelines-dotnet-core' which is 'Enabled'. To the right, there's a box labeled 'steved0x/pipelines-dotnet-core' containing a checkbox labeled 'Override the YAML pull request trigger from here', which is currently unchecked. Below this, under 'Forks', there's a checked checkbox 'Build pull requests from forks of this repository' and a link 'Important security considerations'.

### Pull request triggers not supported with Azure Repos

If your `pr` trigger isn't firing, and you are using Azure Repos, it is because `pr` triggers aren't supported for Azure Repos. In Azure Repos Git, branch policies are used to implement pull request build validation. For more information, see [Branch policy for pull request validation](#).

### Branch filters misconfigured in CI and PR triggers

When you define a YAML PR or CI trigger, you can specify both `include` and `exclude` clauses for branches and paths. Ensure that the `include` clause matches the details of your commit and that the `exclude` clause doesn't exclude them.

#### IMPORTANT

When you define a YAML PR or CI trigger, only branches explicitly configured to be included will trigger a run. Includes are processed first, and then excludes are removed from the list. If you specify an exclude but don't specify any includes, nothing will trigger. For more information, see [Triggers](#).

When you define a YAML PR or CI trigger, you can specify both `include` and `exclude` clauses for branches, tags, and paths. Ensure that the `include` clause matches the details of your commit and that the `exclude` clause doesn't exclude them. For more information, see [Triggers](#).

#### NOTE

If you specify an `exclude` clause without an `include` clause, it is equivalent to specifying `*` in the `include` clause.

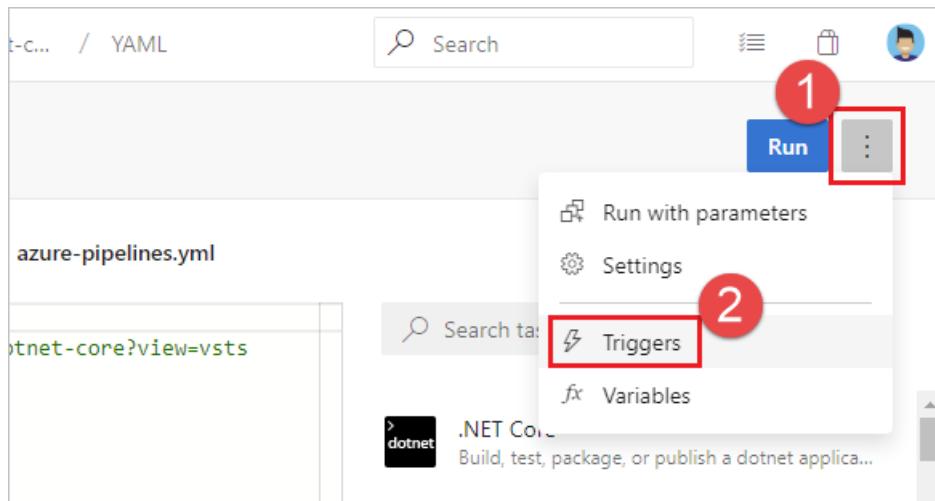
## Scheduled trigger time zone conversions

YAML scheduled triggers are set using UTC time zone. If your scheduled triggers don't seem to be firing at the right time, confirm the conversions between UTC and your local time zone, taking into account the day setting as well. For more information, see [Scheduled triggers](#).

## UI settings override YAML scheduled triggers

If your YAML pipeline has both YAML scheduled triggers and UI defined scheduled triggers, only the UI defined scheduled triggers are run. To run the YAML defined scheduled triggers in your YAML pipeline, you must remove the scheduled triggers defined in the pipeline settings UI.

To access the pipeline settings UI from a YAML pipeline, edit your pipeline, choose ... and then Triggers.



Remove all scheduled triggers.

A screenshot of the 'Triggers' tab in the Azure DevOps pipeline settings UI. On the left, there's a 'Continuous integration' section with a 'ScheduledTriggerTest' configuration. Under 'Scheduled', there's a trigger for '2:00 Mon through Fri' (highlighted with a red box). To the right, there's a 'Delete' button (also highlighted with a red box). Below this, there's a 'When to build' section with checkboxes for days of the week (Mon-Fri are checked, Sun is unchecked) and dropdowns for '02h' and '00m'. A time zone dropdown shows '(UTC-05:00) Eastern Time (US & Canada)'. There's also a checkbox for 'Only schedule builds if the source or pipeline has changed'. At the bottom, there are sections for 'Build completion' and 'Branch filters'.

Once all UI scheduled triggers are removed, a push must be made in order for the YAML scheduled triggers to start running. For more information, see [Scheduled triggers](#).

## Pipeline queues but never gets an agent

If your pipeline queues but never gets an agent, check the following items.

- [Parallel job limits - no available agents or you have hit your free limits](#)
- [Can't access Azure Key Vault behind firewall from Azure DevOps](#)
- [You don't have enough concurrency](#)
- [Your job may be waiting for approval](#)

- All available agents are in use
- Demands that don't match the capabilities of an agent
- Check Azure DevOps status for a service degradation

#### NOTE

The following scenarios won't consume a parallel job:

- If you use release pipelines or multi-stage YAML pipelines, then a run 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.
- When you run a server job or deploy to a deployment group using release pipelines, you don't consume any parallel jobs.

Learn more: [How a parallel job is consumed by a pipeline](#), [Add Pre-deployment approvals](#), [Server jobs](#), [Deployment groups](#)

- Parallel job limits - no available agents or you have hit your free limits
- Demands that don't match the capabilities of an agent
- TFS agent connection issues

#### Parallel job limits - no available agents or you have hit your free limits

If you are currently running other pipelines, you may not have any remaining parallel jobs, or you may have hit your [free limits](#).

To check your limits, navigate to [Project settings](#), [Parallel jobs](#).

The screenshot shows the Azure DevOps Project Settings page for the 'Space Game - web' project. The left sidebar has a red box around the 'Parallel jobs' item under the 'Pipelines' section, with a red number '1' above it. A red box surrounds the 'Parallel jobs' section in the main content area, with a red number '2' above it. The main content area displays parallel job limits for private and public projects.

Project Type	Hosted Type	Free tier	Unlimited
Private projects	Microsoft-hosted	1 parallel job up to 50000 mins/mo	Purchase parallel jobs
	Self-hosted	Unlimited Parallel jobs	Unlimited
Public projects	Microsoft-hosted	10 Parallel jobs	Unlimited Parallel jobs
	Self-hosted	Unlimited Parallel jobs	Unlimited

After reviewing the limits, check concurrency to see how many jobs are currently running and how many are available.

If you are currently running other pipelines, you may not have any remaining parallel jobs, or you may have hit your [free limits](#).

### Can't access Azure Key Vault behind firewall from Azure DevOps

If you can't access Azure Key Vault from your pipeline, the firewall might be blocking the Azure DevOps Services agent IP address. The IP addresses published in the [weekly JSON file](#) must be allowlisted. For more information, see [Microsoft-hosted agents: Networking](#).

### You don't have enough concurrency

To check how much concurrency you have:

1. To check your limits, navigate to **Project settings**, **Parallel jobs**.

The screenshot shows the 'Parallel jobs' section of the 'Project settings' page. The left sidebar has a red box around the 'Parallel jobs' item, with a red number 1 above it. The main content area has a red box around the 'Parallel jobs' section, with a red number 2 above it. The content area displays two sections: 'Private projects' and 'Public projects'. In the 'Private projects' section, there is a 'Microsoft-hosted' row with a 'Free tier' limit of 1 parallel job up to 50000 mins/mo. A 'View in-progress jobs' button is highlighted with a red box. Below this, there are rows for 'Self-hosted' (Unlimited), 'Free parallel jobs' (Unlimited), 'Visual Studio Enterprise subscribers' (0), and 'Monthly purchases' (0 Change). In the 'Public projects' section, there is a 'Microsoft-hosted' row with a limit of 10 parallel jobs. A 'View in-progress jobs' button is also present here. Below this, there is a 'Self-hosted' row with an 'Unlimited' limit.

You can also reach this page by navigating to

[https://dev.azure.com/{org}/\\_settings/buildqueue?\\_a=concurrentJobs](https://dev.azure.com/{org}/_settings/buildqueue?_a=concurrentJobs), or choosing **manage parallel jobs** from the logs.

```
1 Pool: Azure Pipelines
2 Image: ubuntu-16.04
3 Queued: Just now [manage_parallel_jobs]
```

2. Determine which pool you want to check concurrency on (Microsoft hosted or self hosted pools), and choose **View in-progress jobs**.
3. You'll see text that says **Currently running X/X jobs**. If both numbers are the same then jobs will wait

until currently running jobs complete.

Name	Type	Agent	Started on
20191023.1	Build	Hosted Agent	just now

You can view all jobs, including queued jobs, by selecting **Agent pools** from the **Project settings**.

Name	Queued jobs	Running jobs
Azure Pipelines	1	1
Default	0	0

In this example, the concurrent job limit is one, with one job running and one queued up. When all agents are busy running jobs, as in this example, the following message is displayed when additional jobs are queued:

The agent request is not running because all potential agents are running other requests. Current position in queue: 1

. In this example the job is next in the queue so its position is one.

### Your job may be waiting for approval

Your pipeline may not move to the next stage because it is waiting on approval. For more information, see [Define approvals and checks](#).

### All available agents are in use

Jobs may wait if all your agents are currently busy. To check your agents:

1. Navigate to [https://dev.azure.com/{org}/\\_settings/agentpools](https://dev.azure.com/{org}/_settings/agentpools)
2. Select the agent pool to check, in this example **FabrikamPool**, and choose **Agents**.

Name	Last run	Current status	Enabled
FabrikamAgentSDFabrikam1 ● Offline	Jan 9	Idle	<input checked="" type="checkbox"/> On
FabrikamAgentSDFabrikam2 ● Offline	Jan 8	Idle	<input checked="" type="checkbox"/> On

This page shows all the agents currently online/offline and in use. You can also add additional agents to the pool from this page.

#### Demands that don't match the capabilities of an agent

If your pipeline has demands that don't meet the capabilities of any of your agents, your pipeline won't start. If only some of your agents have the desired capabilities and they are currently running other pipelines, your pipeline will be stalled until one of those agents becomes available.

To check the capabilities and demands specified for your agents and pipelines, see [Capabilities](#).

#### NOTE

Capabilities and demands are typically used only with self-hosted agents. If your pipeline has demands that don't match the system capabilities of the agent, unless you have explicitly labelled the agents with matching capabilities, your pipelines won't get an agent.

#### TFS agent connection issues

- [Config fails while testing agent connection \(on-premises TFS only\)](#)
- [Agent lost communication](#)
- [TFS Job Agent not started](#)
- [Misconfigured notification URL \(1.x agent version\)](#)

#### 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.

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

#### 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 several 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.

#### 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.

### Check Azure DevOps status for a service degradation

Check the [Azure DevOps Service Status Portal](#) for any issues that may cause a service degradation, such as increased queue time for agents. For more information, see [Azure DevOps Service Status](#).

## Pipeline fails to complete

If your pipeline gets an agent but fails to complete, check the following common issues. If your issue doesn't seem to match one of these, see [Get logs to diagnose problems](#).

- [Job time-out](#)
- [Issues downloading code](#)
- [My pipeline is failing on a command-line step such as MSBUILD](#)
- [File or folder in use errors](#)
- [Intermittent or inconsistent MSBuild failures](#)
- [Process stops responding](#)
- [Line endings for multiple platforms](#)
- [Variables having ' \(single quote\) appended](#)
- [Service Connection related issues](#)

### Job time-out

A pipeline may run for a long time and then fail due to job time-out. Job timeout closely depends on the agent being used. Free Microsoft hosted agents have a max timeout of 60 minutes per job for a private repository and 360 minutes for a public repository. To increase the max timeout for a job, you can opt for any of the following.

- Buy a Microsoft hosted agent which will give you 360 minutes for all jobs, irrespective of the repository used
- Use a self-hosted agent to rule out any timeout issues due to the agent

Learn more about job [timeout](#).

#### NOTE

If your Microsoft-hosted agent jobs are timing out, ensure that you haven't specified a pipeline timeout that is less than the max timeout for a job. To check, see [Timeouts](#).

### Issues downloading code

- [My pipeline is failing on a checkout step](#)
- [Team Foundation Version Control \(TFVC\) issues](#)

### My pipeline is failing on a checkout step

If you are using a `checkout` step on an Azure Repos Git repository in your organization that is in a different project than your pipeline, ensure that the **Limit job authorization scope to current project** setting is disabled, or follow the steps in [Scoped build identities](#) to ensure that your pipeline has access to the repository.

When your pipeline can't access the repository due to limited job authorization scope, you will receive the error

`Git fetch failed with exit code 128` and your logs will contain an entry similar to

`Remote: TF401019: The Git repository with name or identifier <your repo name> does not exist or you do not have permissions for the operation you are attempting.`

If your pipeline is failing immediately with `Could not find a project that corresponds with the repository`, ensure that your project and repository name are correct in the `checkout` step or the repository resource declaration.

#### Team Foundation Version Control (TFVC) issues

- [Get sources not downloading some files](#)
- [Get sources through Team Foundation Proxy](#)

##### `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
```

## My pipeline is failing on a command-line step such as MSBUILD

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.

#### File layout

The location of tools, libraries, headers, and other things needed for a build may be different on the hosted agent than from your local machine. If a build fails because it can't find one of these files, you can use the below scripts to inspect the layout on the agent. This may help you track down the missing file.

Create a new YAML pipeline in a temporary location (e.g. a new repo created for the purpose of troubleshooting). As written, the script searches directories on your path. You may optionally edit the `SEARCH_PATH=` line to search other places.

```

Script for Linux and macOS
pool: { vmImage: ubuntu-latest } # or whatever pool you use
steps:
- checkout: none
- bash: |
 SEARCH_PATH=$PATH # or any colon-delimited list of paths
 IFS=':' read -r -a PathDirs <<< "$SEARCH_PATH"
 echo "###[debug] Found directories"
 for element in "${PathDirs[@]}"; do
 echo "$element"
 done;
 echo;
 echo;
 echo "###[debug] Found files"
 for element in "${PathDirs[@]}"; do
 find "$element" -type f
 done

```

```

Script for Windows
pool: { vmImage: windows-2019 } # or whatever pool you use
steps:
- checkout: none
- powershell: |
 $SEARCH_PATH=$Env:Path
 Write-Host "###[debug] Found directories"
 ForEach ($Dir in $SEARCH_PATH -split ";") {
 Write-Host "$Dir"
 }
 Write-Host ""
 Write-Host ""
 Write-Host "###[debug] Found files"
 ForEach ($Dir in $SEARCH_PATH -split ";") {
 Get-ChildItem $Dir -File -ErrorAction Continue | ForEach-Object -Process {
 Write-Host $_.FullName
 }
 }

```

#### 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.

#### File or 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 [...]

Troubleshooting steps:

- [Detect files and folders in use](#)
- [Anti-virus exclusion](#)
- [MSBuild and /nodeReuse:false](#)
- [MSBuild and /maxcpucount:\[n\]](#)

#### **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 stops responding**

Process stops responding causes and troubleshooting steps:

- [Waiting for Input](#)
- [Process dump](#)
- [WiX project](#)

#### **Waiting for Input**

A process that stops responding 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 work around 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
```

## Variables having ' (single quote) appended

If your pipeline includes a Bash script that sets variables using the `##vso` command, you may see an additional `'` appended to the value of the variable you set. This occurs because of an interaction with `set -x`. The solution is to disable `set -x` temporarily before setting a variable. The Bash syntax for doing that is `set +x`.

```
set +x
echo ##vso[task.setvariable variable=MY_VAR]my_value
set -x
```

### Why does this happen?

Many Bash scripts include the `set -x` command to assist with debugging. Bash will trace exactly what command was executed and echo it to stdout. This will cause the agent to see the `##vso` command twice, and the second time, Bash will have added the `'` character to the end.

For instance, consider this pipeline:

```
steps:
- bash: |
 set -x
 echo ##vso[task.setvariable variable=MY_VAR]my_value
```

On stdout, the agent will see two lines:

```
##vso[task.setvariable variable=MY_VAR]my_value
+ echo '##vso[task.setvariable variable=MY_VAR]my_value'
```

When the agent sees the first line, `MY_VAR` will be set to the correct value, "my\_value". However, when it sees the second line, the agent will process everything to the end of the line. `MY_VAR` will be set to "my\_value".

## Libraries aren't installed for Python application when script executes

When a Python application is deployed, in some cases, a CI/CD pipeline runs and the code is deployed successfully, but the `requirements.txt` file that's responsible for installing all dependency libraries doesn't execute.

To install the dependencies, use a post-deployment script in the App Service deployment task. The following example shows the command you must use in the post-deployment script. You can update the script for your scenario.

```
D:\home\python364x64\python.exe -m pip install -r requirements.txt
```

## Service Connection related issues

To troubleshoot issues related to service connections, see [Service connection troubleshooting](#).

## Enable Storage Explorer to deploy static content like .css and .js to a static website from Azure DevOps via Azure Pipelines

In this scenario, you can use the [Azure File Copy task](#) to upload content to the website. You can use any of the tools described in [Uploading content](#) to upload content to the web container.

## Get logs to diagnose problems

If none of the previous suggestions match your problem, you can use the information in the logs to diagnose your failing pipeline.

Start by looking at the logs in your completed build or release. You can view logs by navigating to the pipeline run summary and selecting the job and task. If a certain task is failing, check the logs for that task.

In addition to viewing logs in the pipeline build summary, you can download complete logs which include additional diagnostic information, and you can configure more verbose logs to assist with your troubleshooting.

For detailed instructions for configuring and using logs, see [Review logs to diagnose pipeline issues](#).

## 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 or submit feedback at [Developer Community](#).

We welcome your suggestions:

# Review logs to diagnose pipeline issues

4/7/2021 • 3 minutes to read • [Edit Online](#)

Pipeline logs provide a powerful tool for determining the cause of pipeline failures.

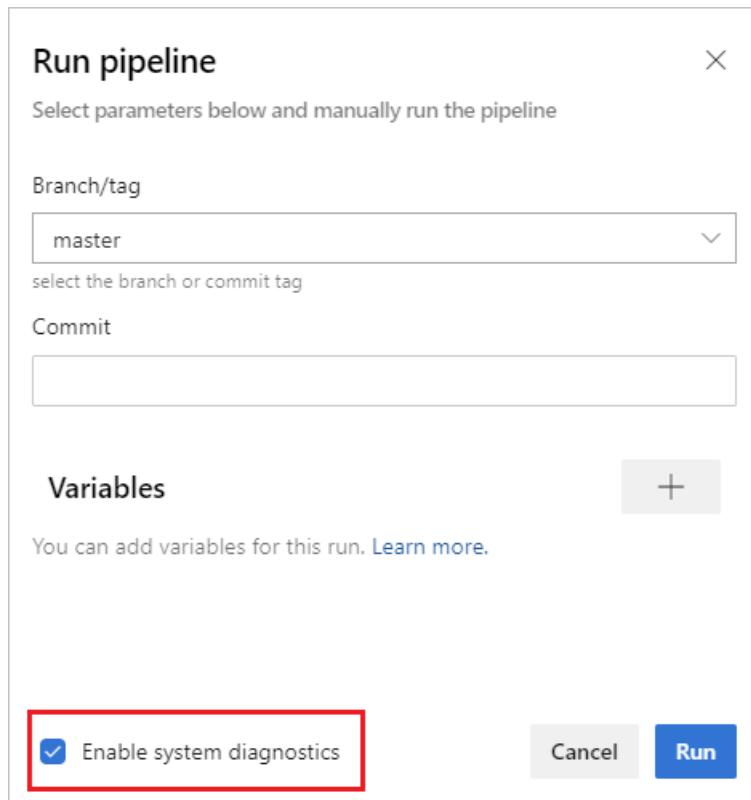
A typical starting point is to review the logs in your completed build or release. You can view logs by navigating to the pipeline run summary and selecting the job and task. If a certain task is failing, check the logs for that task.

In addition to viewing logs in the pipeline build summary, you can download complete logs which include additional diagnostic information, and you can configure more verbose logs to assist with your troubleshooting.

## Configure verbose logs

To assist with troubleshooting, you can configure your logs to be more verbose.

- To configure verbose logs for a single run, you can start a new build by choosing **Run pipeline** and selecting **Enable system diagnostics**, **Run**.

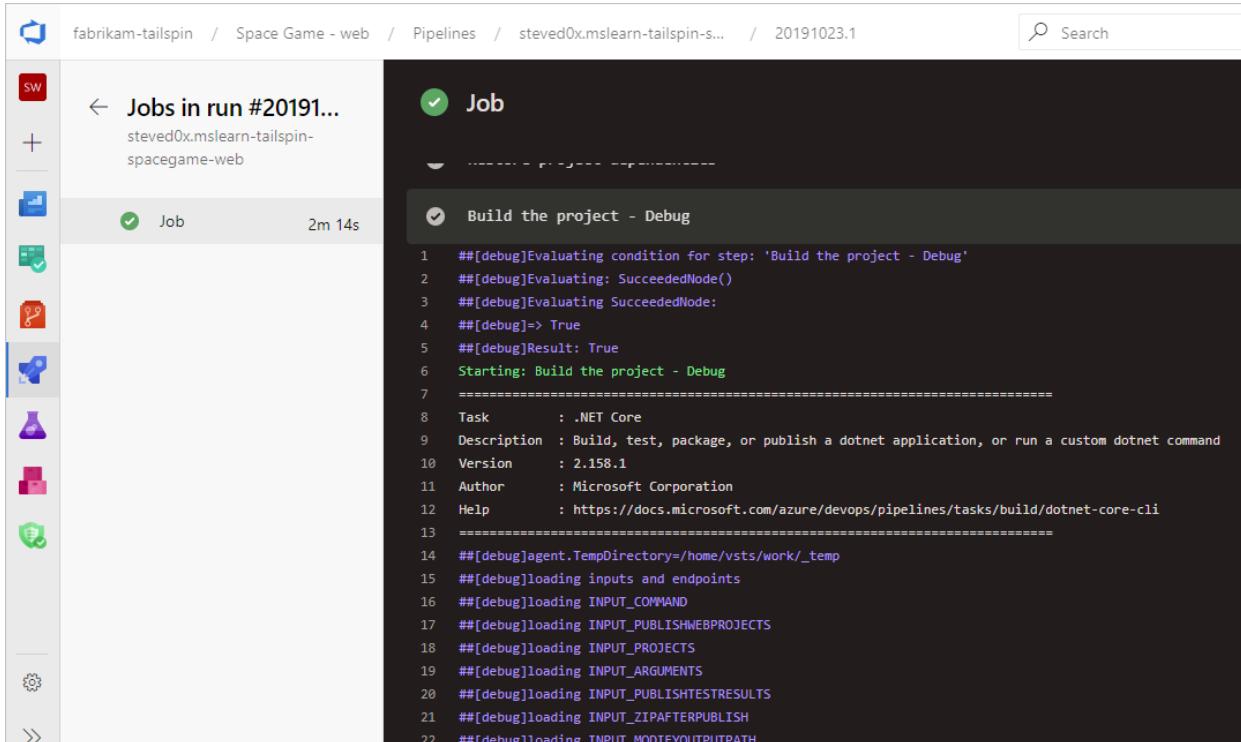


- To configure verbose logs for all runs, you can add a variable named `system.debug` and set its value to `true`.
- To configure verbose logs for a single run, you can start a new build by choosing **Queue build**, and setting the value for the `system.debug` variable to `true`.
- To configure verbose logs for all runs, edit the build, navigate to the **Variables** tab, and add a variable named `system.debug`, set its value to `true`, and select to **Allow at Queue Time**.
- To configure verbose logs for a YAML pipeline, add the `system.debug` variable in the `variables` section:

```
variables:
 system.debug: true
```

## View and download logs

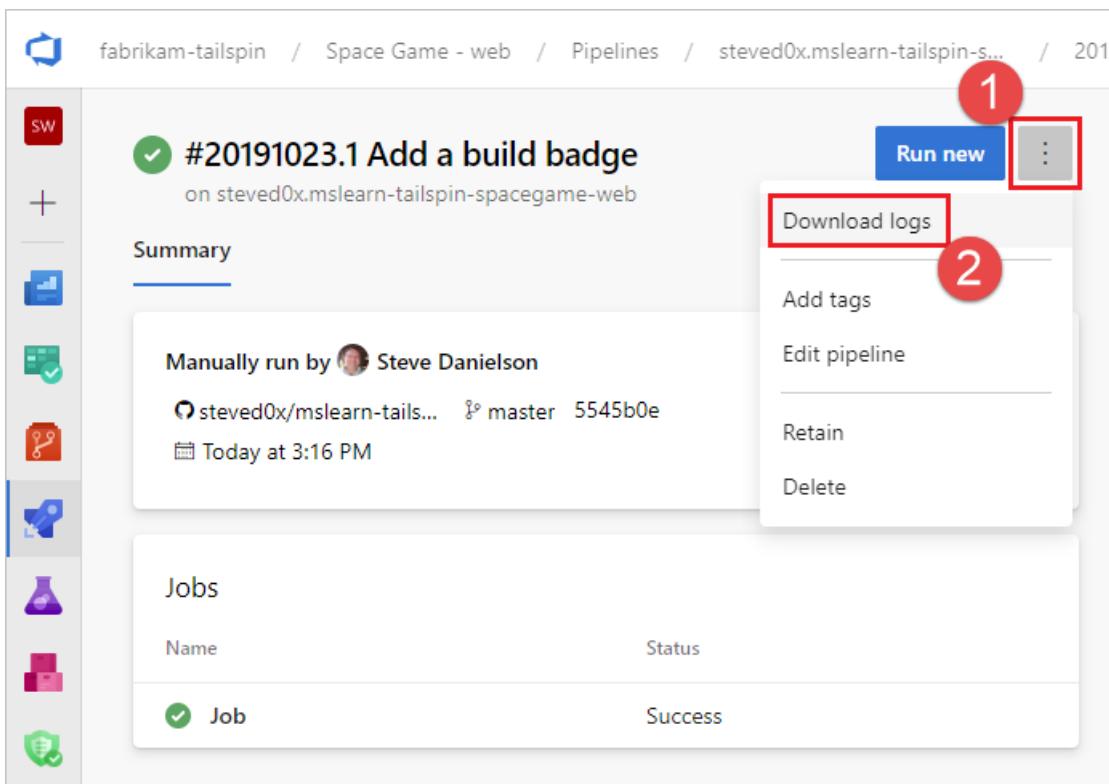
To view individual logs for each step, navigate to the build results for the run, and select the job and step.



The screenshot shows the Azure Pipelines interface. On the left is a sidebar with various icons for different pipeline components. The main area displays a 'Job' named 'Build the project - Debug'. Below the job name is a green checkmark icon and the text '2m 14s'. The right side of the screen shows the log output for this step. The log content includes several lines of debug information starting with '#[debug]Evaluating condition for step: 'Build the project - Debug''. It also lists task details such as Task (.NET Core), Description (Build, test, package, or publish a dotnet application, or run a custom dotnet command), Version (2.158.1), Author (Microsoft Corporation), and Help (link to Microsoft documentation).

```
1 ##[debug]Evaluating condition for step: 'Build the project - Debug'
2 ##[debug]Evaluating: SucceededNode()
3 ##[debug]Evaluating SucceededNode:
4 ##[debug]> True
5 ##[debug]Result: True
6 Starting: Build the project - Debug
7 =====
8 Task : .NET Core
9 Description : Build, test, package, or publish a dotnet application, or run a custom dotnet command
10 Version : 2.158.1
11 Author : Microsoft Corporation
12 Help : https://docs.microsoft.com/azure/devops/pipelines/tasks/build/dotnet-core-cli
13 =====
14 ##[debug]agent.TempDirectory=/home/vsts/work/_temp
15 ##[debug]loading inputs and endpoints
16 ##[debug]loading INPUT_COMMAND
17 ##[debug]loading INPUT_PUBLISHWEBPROJECTS
18 ##[debug]loading INPUT_PROJECTS
19 ##[debug]loading INPUT_ARGUMENTS
20 ##[debug]loading INPUT_PUBLISHTESTRESULTS
21 ##[debug]loading INPUT_ZIPAFTERPUBLISH
22 ##[debug]loading INPUT_MODIFYOUTPUTPATH
```

To download all logs, navigate to the build results for the run, select ..., and choose **Download logs**.



The screenshot shows the Azure Pipelines interface for a specific build run. The top navigation bar includes 'fabrikam-tailspin / Space Game - web / Pipelines / steved0x.mslearn-tailspin-s... / 201'. A context menu is open over the run summary, indicated by a red circle with the number '1'. The menu items are: 'Run new' (blue button), a three-dot ellipsis icon, 'Download logs' (highlighted with a red box and number '2'), 'Add tags', 'Edit pipeline', 'Retain', and 'Delete'. The main area below shows the 'Summary' section with details like 'Manually run by Steve Danielson' and the commit hash '5545b0e'. It also shows a 'Jobs' table with one entry: 'Job' status 'Success'.

To download all logs, navigate to the build results for the run, choose **Download all logs as zip**.

In addition to the pipeline diagnostic logs, the following specialized log types are available, and may contain information to help you troubleshoot.

- [Worker diagnostic logs](#)
- [Agent diagnostic logs](#)
- [Other logs](#)

## 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.txt` file provides a clean way to see all capabilities installed on the build machine that ran your build.

## HTTP trace logs

- [Use built-in HTTP tracing](#)
- [Use full HTTP tracing - Windows](#)
- [Use full HTTP tracing - macOS and Linux](#)

### 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.

## Use full HTTP tracing - 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.

# Classic release and artifacts variables

5/20/2021 • 14 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Classic release and artifacts variables are a convenient way to exchange and transport data throughout your pipeline. Each variable is stored as a string and its value can change between runs of your pipeline.

Variables are different from [Runtime parameters](#) which are only available at template parsing time.

## NOTE

This is a reference article that covers the classic release and artifacts variables. To understand variables in YAML pipelines, see [user-defined variables](#).

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**.

## TIP

You can view the [current values of all variables](#) for a release, and use a default variable to [run a release in debug mode](#).

## 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. To view the full list, see [View the current values of all variables](#).

## Default variables - System

VARIABLE NAME	DESCRIPTION
System.TeamFoundationServerUri	<p>The URL of the service connection in TFS or Azure Pipelines. Use this from your scripts or tasks to call Azure Pipelines REST APIs.</p> <p>Example: <code>https://fabrikam.vssrm.visualstudio.com/</code></p>
System.TeamFoundationCollectionUri	<p>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.</p> <p>Example: <code>https://dev.azure.com/fabrikam/</code></p>
System.CollectionId	<p>The ID of the collection to which this build or release belongs. Not available in TFS 2015.</p> <p>Example: <code>6c6f3423-1c84-4625-995a-f7f143a1e43d</code></p>
System.DefinitionId	<p>The ID of the release pipeline to which the current release belongs. Not available in TFS 2015.</p> <p>Example: <code>1</code></p>
System.TeamProject	<p>The name of the project to which this build or release belongs.</p> <p>Example: <code>Fabrikam</code></p>
System.TeamprojectId	<p>The ID of the project to which this build or release belongs. Not available in TFS 2015.</p> <p>Example: <code>79f5c12e-3337-4151-be41-a268d2c73344</code></p>
System.ArtifactsDirectory	<p>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.</p> <p>Example: <code>C:\agent\_work\r1\a</code></p>
System.DefaultWorkingDirectory	<p>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.</p> <p>Example: <code>C:\agent\_work\r1\a</code></p>
System.WorkFolder	<p>The working directory for this agent, where subfolders are created for every build or release. Same as Agent.RootDirectory and Agent.WorkFolder.</p> <p>Example: <code>C:\agent\_work</code></p>

VARIABLE NAME	DESCRIPTION
System.Debug	<p>This is the only system variable that can be <i>set</i> by the users. Set this to true to <a href="#">run the release in debug mode</a> to assist in fault-finding.</p> <p>Example: <code>true</code></p>

## Default variables - Release

VARIABLE NAME	DESCRIPTION
Release.AttemptNumber	<p>The number of times this release is deployed in this stage. Not available in TFS 2015.</p> <p>Example: <code>1</code></p>
Release.DefinitionEnvironmentId	<p>The ID of the stage in the corresponding release pipeline. Not available in TFS 2015.</p> <p>Example: <code>1</code></p>
Release.DefinitionId	<p>The ID of the release pipeline to which the current release belongs. Not available in TFS 2015.</p> <p>Example: <code>1</code></p>
Release.DefinitionName	<p>The name of the release pipeline to which the current release belongs.</p> <p>Example: <code>fabrikam-cd</code></p>
Release.Deployment.RequestedFor	<p>The display name of the identity that triggered (started) the deployment currently in progress. Not available in TFS 2015.</p> <p>Example: <code>Mateo Escobedo</code></p>
Release.Deployment.RequestedForEmail	<p>The email address of the identity that triggered (started) the deployment currently in progress. Not available in TFS 2015.</p> <p>Example: <code>mateo@fabrikam.com</code></p>
Release.Deployment.RequestedForId	<p>The ID of the identity that triggered (started) the deployment currently in progress. Not available in TFS 2015.</p> <p>Example: <code>2f435d07-769f-4e46-849d-10d1ab9ba6ab</code></p>
Release.DeploymentID	<p>The ID of the deployment. Unique per job.</p> <p>Example: <code>254</code></p>
Release.DeployPhaseID	<p>The ID of the phase where deployment is running.</p> <p>Example: <code>127</code></p>

VARIABLE NAME	DESCRIPTION
Release.EnvironmentId	The ID of the stage instance in a release to which the deployment is currently in progress.  Example: <code>276</code>
Release.EnvironmentName	The name of stage to which deployment is currently in progress.  Example: <code>Dev</code>
Release.EnvironmentUri	The URI of the stage instance in a release to which deployment is currently in progress.  Example: <code>vstfs://ReleaseManagement/Environment/276</code>
Release.Environments.{stage-name}.status	The deployment status of the stage.  Example: <code>InProgress</code>
Release.PrimaryArtifactSourceAlias	The alias of the primary artifact source  Example: <code>fabrikam\_web</code>
Release.Reason	The reason for the deployment. Supported values are: <code>ContinuousIntegration</code> - the release started in Continuous Deployment after a build completed. <code>Manual</code> - the release started manually. <code>None</code> - the deployment reason has not been specified. <code>Scheduled</code> - the release started from a schedule.
Release.ReleaseDescription	The text description provided at the time of the release.  Example: <code>Critical security patch</code>
Release.ReleaseId	The identifier of the current release record.  Example: <code>118</code>
Release.ReleaseName	The name of the current release.  Example: <code>Release-47</code>
Release.ReleaseUri	The URI of current release.  Example: <code>vstfs://ReleaseManagement/Release/118</code>
Release.ReleaseWebURL	The URL for this release.  Example: <code>https://dev.azure.com/fabrikam/f3325c6c/_release?releaseId=392&amp;_a=release-summary</code>
Release.RequestedFor	The display name of identity that triggered the release.  Example: <code>Mateo Escobedo</code>

VARIABLE NAME	DESCRIPTION
Release.RequestedForEmail	The email address of identity that triggered the release.  Example: <code>mateo@fabrikam.com</code>
Release.RequestedForId	The ID of identity that triggered the release.  Example: <code>2f435d07-769f-4e46-849d-10d1ab9ba6ab</code>
Release.SkipArtifactsDownload	Boolean value that specifies whether or not to skip downloading of artifacts to the agent.  Example: <code>FALSE</code>
Release.TriggeringArtifact.Alias	The alias of the artifact which triggered the release. This is empty when the release was scheduled or triggered manually.  Example: <code>fabrikam\_app</code>

## Default variables - Release stage

VARIABLE NAME	DESCRIPTION
Release.Environments.{stage name}.Status	The status of deployment of this release within a specified stage. Not available in TFS 2015.  Example: <code>NotStarted</code>

## Default variables - Agent

VARIABLE NAME	DESCRIPTION
Agent.Name	The name of the agent as registered with the <a href="#">agent pool</a> . This is likely to be different from the computer name.  Example: <code>fabrikam-agent</code>
Agent.MachineName	The name of the computer on which the agent is configured.  Example: <code>fabrikam-agent</code>
Agent.Version	The version of the agent software.  Example: <code>2.109.1</code>
Agent.JobName	The name of the job that is running, such as Release or Build.  Example: <code>Release</code>
Agent.HomeDirectory	The folder where the agent is installed. This folder contains the code and resources for the agent.  Example: <code>C:\agent</code>

VARIABLE NAME	DESCRIPTION
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.  Example: <code>C:\agent\_work\r1\a</code>
Agent.RootDirectory	The working directory for this agent, where subfolders are created for every build or release. Same as Agent.WorkFolder and System.WorkFolder.  Example: <code>C:\agent\_work</code>
Agent.WorkFolder	The working directory for this agent, where subfolders are created for every build or release. Same as Agent.RootDirectory and System.WorkFolder.  Example: <code>C:\agent\_work</code>
Agent.DeploymentGroupId	The ID of the deployment group the agent is registered with. This is available only in deployment group jobs. Not available in TFS 2018 Update 1.  Example: <code>1</code>

## Default variables - General Artifact

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 the `{alias}` placeholder with the value you specified for the [artifact alias](#) or with the default value generated for the release pipeline.

VARIABLE NAME	DESCRIPTION
Release.Artifacts.{alias}.DefinitionId	The identifier of the build pipeline or repository.  Azure Pipelines example: <code>1</code> GitHub example: <code>fabrikam/asp</code>
Release.Artifacts.{alias}.DefinitionName	The name of the build pipeline or repository.  Azure Pipelines example: <code>fabrikam-ci</code> TFVC example: <code>\$/fabrikam</code> Git example: <code>fabrikam</code> GitHub example: <code>fabrikam/asp (main)</code>

VARIABLE NAME	DESCRIPTION
Release.Artifacts.{alias}.BuildNumber	<p>The build number or the commit identifier.</p> <p>Azure Pipelines example: 20170112.1  Jenkins/TeamCity example: 20170112.1  TFVC example: Changeset 3  Git example: 38629c964  GitHub example: 38629c964</p>
Release.Artifacts.{alias}.BuildId	<p>The build identifier.</p> <p>Azure Pipelines example: 130  Jenkins/TeamCity example: 130  GitHub example:  38629c964d21fe405ef830b7d0220966b82c9e11</p>
Release.Artifacts.{alias}.BuildURI	<p>The URL for the build.</p> <p>Azure Pipelines example:  vstfs://build-release/Build/130  GitHub example: https://github.com/fabrikam/asp</p>
Release.Artifacts.{alias}.SourceBranch	<p>The full path and name of the branch from which the source was built.</p> <p>Azure Pipelines example: refs/heads/main</p>
Release.Artifacts.{alias}.SourceBranchName	<p>The name only of the branch from which the source was built.</p> <p>Azure Pipelines example: main</p>
Release.Artifacts.{alias}.SourceVersion	<p>The commit that was built.</p> <p>Azure Pipelines example:  bc0044458ba1d9298cdc649cb5dcf013180706f7</p>
Release.Artifacts.{alias}.Repository.Provider	<p>The type of repository from which the source was built.</p> <p>Azure Pipelines example: Git</p>
Release.Artifacts.{alias}.RequestedForID	<p>The identifier of the account that triggered the build.</p> <p>Azure Pipelines example:  2f435d07-769f-4e46-849d-10d1ab9ba6ab</p>
Release.Artifacts.{alias}.RequestedFor	<p>The name of the account that requested the build.</p> <p>Azure Pipelines example: Mateo Escobedo</p>

VARIABLE NAME	DESCRIPTION
Release.Artifacts.{alias}.Type	<p>The type of artifact source, such as Build.</p> <p>Azure Pipelines example: <code>Build</code>  Jenkins example: <code>Jenkins</code>  TeamCity example: <code>TeamCity</code>  TFVC example: <code>TFVC</code>  Git example: <code>Git</code>  GitHub example: <code>Github</code></p>
Release.Artifacts.{alias}.PullRequest.TargetBranch	<p>The full path and name of the branch that is the target of a pull request. This variable is initialized only if the release is triggered by a pull request flow.</p> <p>Azure Pipelines example: <code>refs/heads/main</code></p>
Release.Artifacts.{alias}.PullRequest.TargetBranchName	<p>The name only of the branch that is the target of a pull request. This variable is initialized only if the release is triggered by a pull request flow.</p> <p>Azure Pipelines example: <code>main</code></p>

See also [Artifact source alias](#)

## Default variables - Primary Artifact

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

VARIABLE NAME	SAME AS
Build.Type	Release.Artifacts.{Primary artifact alias}.Type
Build.PullRequest.TargetBranch	Release.Artifacts.{Primary artifact alias}.PullRequest.TargetBranch
Build.PullRequest.TargetBranchName	Release.Artifacts.{Primary artifact alias}.PullRequest.TargetBranchName

## 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)`.

**Powershell Script**

Type	Inline Script
Arguments	<code>-a "\$(Release.Artifacts.ASPNET4.CI.DEFINITIONNAME)"</code>
Inline Script	<pre>param (\$a) Write-Host "\$a"</pre>

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	<code>Write-Host "\$env:RELEASE_ARTIFACTS_ASPNET4_CI_DEFINITIONNAME"</code>

Note that the original name of the artifact source alias, `ASPNET4.CI`, is replaced by `ASPNET4_CI`.

### View the current values of all 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 interface. At the top, it says "SampleApp - 1 > Release-2 > QA > Succeeded". Below that is a navigation bar with Pipeline, Tasks, Variables, Logs, Tests, Deploy, Cancel, Refresh, and Download buttons. The "Logs" tab is selected. On the left, there's a sidebar titled "Deployment process" with a "Succeeded" status. Under "Run on agent", there's also a "Succeeded" status. The main area is titled "Run on agent" and shows a log entry for "Initialize job" which is also marked as "succeeded". This log entry is highlighted with a red box.

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 for the "Initialize job" step. It displays a series of environment variable mappings. The log entries are:

```

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_TOOLDIRECTORY] --> [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.

### TIP

If you get an error related to an Azure RM service connection, see [How to: Troubleshoot Azure Resource Manager service connections](#).

## 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. In the Pipeline Variables page, open the Scope drop-down list and select "Release". By default, when you add a variable, it is set to Release scope.
- Share values across all of the tasks within one specific stage by using **stage variables**. Use a stage-level variable for values that vary from stage to stage (and are the same for all the tasks in a stage). You define and manage these variables in the **Variables** tab of a release pipeline. In the Pipeline Variables page, open the Scope drop-down list and select the required stage. When you add a variable, set the Scope to the appropriate environment.

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.

#### IMPORTANT

The values of the hidden (secret) variables are securely stored 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.

#### NOTE

Creating custom variables can overwrite standard variables. For example, the PowerShell **Path** environment variable. If you create a custom `Path` variable on a Windows agent, it will overwrite the `$env:Path` variable and PowerShell won't be able to run.

## 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 (e.g., job or stage) will collide and the result may be unpredictable. Ensure that you use different names for variables across all your variable groups.

### 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)
```

# Troubleshoot Azure Resource Manager service connections

5/7/2021 • 8 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

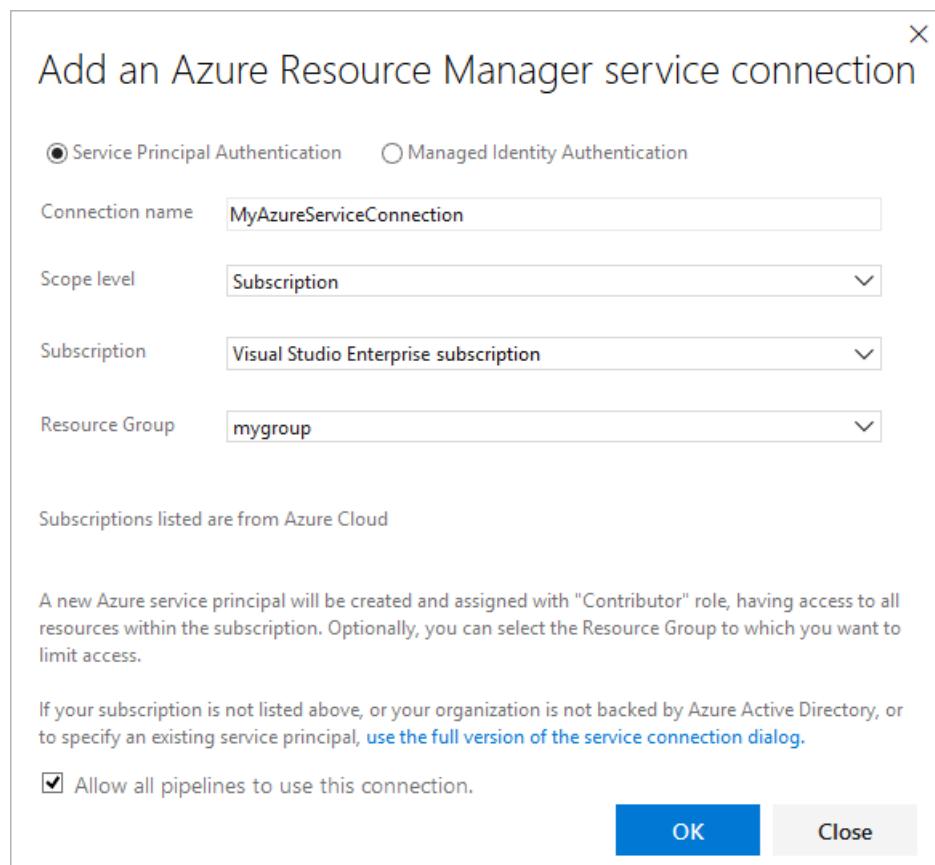
## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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 Azure DevOps CI/CD processes.

## What happens when you create a 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 the type of service connection you need.
3. In 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 select **OK**, the system:

1. Connects to the Azure Active Directory (Azure AD) tenant for to 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.
4. Creates an Azure Resource Manager service connection using this application's details.

## How do I troubleshoot errors that may occur while creating a connection?

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](#)
- [Some subscriptions are missing from the subscription drop down menu](#)
- [Subscription isn't listed when creating a service connection](#)
- [Automatically created service principal secret has expired](#)
- [Failed to obtain the JSON Web Token \(JWT\)](#)
- [Creating a service connection with Artifactory instance fails despite allowlisted IPs](#)
- [Can't create a service connection manually by using PowerShell scripts and Azure Cloud Shell](#)
- [Azure subscription not taken directly from previous task output](#)
- [What authentication mechanisms are supported? How do managed identities work?](#)

### **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.

1. Sign in to the Azure portal using an administrator account. The account should be an [owner](#), [global administrator](#), or [user account administrator](#).
2. Select **Azure Active Directory** in the left navigation bar.
3. 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.
4. In the **MANAGE** section select **Users**.
5. Select **User settings**.
6. In the **External users** section, select **Manage external collaboration settings**.
7. The **External collaboration settings** blade opens.

## 8. Change Guest user permissions are limited to No.

Alternatively, if you are prepared to give the user additional permissions (administrator-level), you can make the user a member of the **Global administrator** role. To do so follow the steps below:

### WARNING

Users who are assigned to the Global administrator role can read and modify every administrative setting in your Azure AD organization. As a best practice, we recommend that you assign this role to fewer than five people in your organization.

1. Sign in to the Azure portal using an administrator account. The account should be an [owner](#), [global administrator](#), or [user account administrator](#).
2. Select **Azure Active Directory** in the left navigation bar.
3. 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.
4. In the **MANGE** section select **Users**.
5. Use the search box to filter the list and then select the user you want to manage.
6. In the **MANGE** section select **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 permissions to add integrated applications in the directory. The directory administrator has permissions to change this setting.

1. Select **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 **MANGE** section select **Users**.
4. Select **User settings**.
5. In the **App registrations** section, change **Users can register applications** to **Yes**.

### Create the service principal manually with the user already having required permissions in Azure Active Directory

You can also create the service principal with an existing user who already has the required permissions in Azure Active Directory. For more information, see [Create an Azure Resource Manager service connection with an existing service principal](#).

### 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:

1. Sign out of Azure Pipelines or TFS.
2. Open an InPrivate or incognito browser window and navigate to <https://visualstudio.microsoft.com/team-services/>.

3. If you are prompted to sign out, do so.
4. Sign in using the appropriate credentials.
5. Choose the organization you want to use from the list.
6. Select the project you want to add the service connection to.
7. Create the service connection you need by opening the **Settings** page. Then, select **Services > New service connection > 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 [assign you the appropriate role](#).

#### **Some subscriptions are missing from the list of subscriptions**

To fix this issue you will need to modify the supported account types and who can use your application. To do so, follow the steps below:

1. Sign in to the Azure portal.
2. If you have access to multiple tenants, use the **Directory + subscription** filter in the top menu to select the tenant in which you want to register an application.
3. Search for and select **Azure Active Directory**.
4. Under **Manage**, select **App registrations**.
5. Select your application from the list of registered applications.
6. Under **Essentials**, select **Supported account types**.
7. Under **Supported account types**, *Who can use this application or access this API?* select **Accounts in any organizational directory**.
8. Select **Save**.

#### **Subscription isn't listed when creating a service connection**

A maximum of 50 Azure subscriptions are listed in the various Azure subscription drop-down menus (billing, service connection, etc.). If you're setting up a service connection and you have more than 50 Azure subscriptions, some of your subscriptions won't be listed. In this scenario, complete these steps:

1. Create a new, native Azure AD user in the Azure AD instance for the Azure subscription.
2. Set up the Azure AD user so that it has the proper permissions in the Azure subscription to set up Azure DevOps billing or a service connection. For more information, see [Add a user who can set up billing for Azure DevOps](#).
3. Add the Azure AD user to the Azure DevOps org with the access level of **Stakeholder** and add them to the **Project Collection Administrators** group (for billing), or ensure that the user has sufficient permissions in the Team Project to create service connections.
4. Log in to Azure DevOps as this user and set up a billing service connection. You'll only see one Azure subscription in the list.

#### **Automatically created service principal client secret has expired**

An issue that often arises with service principals that are automatically created is that the service principal's token expires and needs to be renewed. If you run into issues with refreshing the token, check out [our other](#)

[troubleshooting resolutions](#).

To renew the token for an automatically created service principal:

1. Go to the Azure Resource Manager service connection that was created by using the Automatic method.
2. In the upper-right corner, click **Edit**.
3. Click **Verify** on the service connection page.
4. Click **Save**. The client secret for that service principal has now been renewed for two years.

### **Failed to obtain the JWT by using the service principal client ID**

This issue occurs when you try to verify a service connection that has an expired secret.

To resolve this issue:

1. Go to the Azure Resource Manager service connection you want to update.
2. Make a change to the service connection. The easiest and recommended change is to add a description.
3. Save the service connection.

**NOTE**

Select **Save**. Don't try to verify at this step.

4. Exit the service connection, and then refresh the service connections page.
5. Edit the service connection again.
6. Click **Verify**.
7. Save the service connection.

### **Creating a service connection with Artifactory instance fails despite allowlisted IPs**

Be sure that the correct IP addresses are allowlisted. See [Microsoft-hosted agents](#) and the [current list of IP addresses](#) that's published weekly based on geographic location.

### **Can't create a service connection manually by using PowerShell scripts and Azure Cloud Shell**

To learn how to manually create an Azure Resource Manager service connection, see [Create an Azure service principal to use with an Azure Resource Manager service connection](#).

### **Azure subscription not taken directly from previous task output**

When you set an Azure subscription dynamically for the release pipeline and the subscription is an output variable from a preceding task, you might encounter this issue.

To resolve the issue, ensure that the values are defined within the pipeline variables section, which can be used in the subscription name or in the service connection.

### **What authentication mechanisms are supported? How do managed identities work?**

An Azure Resource Manager service connection can connect to an Azure subscription by using a Service Principal Authentication (SPA) or managed identity authentication. Managed identities for Azure resources provide Azure services with an automatically managed identity in Azure Active Directory (Azure AD). You can use this identity to authenticate to any service that supports Azure AD authentication without persisting credentials in code or in the service connection.

To learn about managed identities for virtual machines, see [Assigning roles](#).

**NOTE**

Managed identities aren't supported in Microsoft-hosted agents. In this scenario, you must [set up a self-hosted agent](#) on an Azure VM and configure a managed identity for the VM.

## Help and support

- See our [troubleshooting](#) page
- Get advice on [Stack Overflow](#), and feel free to post your questions, search for answers, or suggest a feature on our [Azure DevOps Developer Community](#). [Support](#) page.

# YAML schema reference

5/10/2021 • 44 minutes to read • [Edit Online](#)

## Azure Pipelines

This article is a detailed reference guide to Azure Pipelines YAML pipelines. It includes a catalog of all supported YAML capabilities and the available options.

The best way to get started with YAML pipelines is to read the [quickstart guide](#). After that, to learn how to configure your YAML pipeline for your needs, see conceptual topics like [Build variables](#) and [Jobs](#).

To learn how to configure your YAML pipeline for your needs, see conceptual topics like [Build variables](#) and [Jobs](#).

## Pipeline structure

A pipeline is one or more stages that describe a CI/CD process. Stages are the major divisions in a pipeline. The stages "Build this app," "Run these tests," and "Deploy to production" are good examples.

A stage is one or more jobs, which are units of work assignable to the same machine. You can arrange both stages and jobs into dependency graphs. Examples include "Run this stage before that one" and "This job depends on the output of that job."

A job is a linear series of steps. Steps can be tasks, scripts, or references to external templates.

This hierarchy is reflected in the structure of a YAML file like:

- Pipeline
  - Stage A
    - Job 1
      - Step 1.1
      - Step 1.2
      - ...
    - Job 2
      - Step 2.1
      - Step 2.2
      - ...
  - Stage B
    - ...

Simple pipelines don't require all of these levels. For example, in a single-job build you can omit the containers for stages and jobs because there are only steps. And because many options shown in this article aren't required and have good defaults, your YAML definitions are unlikely to include all of them.

A pipeline is one or more jobs that describe a CI/CD process. A job is a unit of work assignable to the same machine. You can arrange jobs into dependency graphs like "This job depends on the output of that job."

A job is a linear series of steps. Steps can be tasks, scripts, or references to external templates.

This hierarchy is reflected in the structure of a YAML file like:

- Pipeline
  - Job 1
    - Step 1.1
    - Step 1.2
    - ...
  - Job 2
    - Step 2.1
    - Step 2.2
    - ...

For single-job pipelines, you can omit the jobs container because there are only steps. And because many options shown in this article aren't required and have good defaults, your YAML definitions are unlikely to include all of them.

## Conventions

Here are the syntax conventions used in this article:

- To the left of `:` is a literal keyword used in pipeline definitions.
- To the right of `:` is a data type. The data type can be a primitive type like `string` or a reference to a rich structure defined elsewhere in this article.
- The notation `[ datatype ]` indicates an array of the mentioned data type. For instance, `[ string ]` is an array of strings.
- The notation `{ datatype : datatype }` indicates a mapping of one data type to another. For instance, `{ string: string }` is a mapping of strings to strings.
- The symbol `|` indicates there are multiple data types available for the keyword. For instance, `job | templateReference` means either a job definition or a template reference is 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](#). Azure Pipelines doesn't support all YAML features. Unsupported features include anchors, complex keys, and sets. Also, unlike standard YAML, Azure Pipelines depends on seeing `stage`, `job`, `task`, or a task shortcut like `script` as the first key in a mapping.

# Pipeline

- [Schema](#)
- [Example](#)

```
name: string # build numbering format
resources:
 pipelines: [pipelineResource]
 containers: [containerResource]
 repositories: [repositoryResource]
variables: # several syntaxes, see specific section
trigger: trigger
pr: pr
stages: [stage | templateReference]
```

If you have a single `stage`, you can omit the `stages` keyword and directly specify the `jobs` keyword:

```
... other pipeline-level keywords
jobs: [job | templateReference]
```

If you have a single stage and a single job, you can omit the `stages` and `jobs` keywords and directly specify the `steps` keyword:

```
... other pipeline-level keywords
steps: [script | bash | pwsh | powershell | checkout | task | templateReference]
```

```
name: string # build numbering format
resources:
 containers: [containerResource]
 repositories: [repositoryResource]
variables: # several syntaxes, see specific section
trigger: trigger
pr: pr
jobs: [job | templateReference]
```

If you have a single job, you can omit the `jobs` keyword and directly specify the `steps` keyword:

```
... other pipeline-level keywords
steps: [script | bash | pwsh | powershell | checkout | task | templateReference]
```

Learn more about:

- [Pipelines with multiple jobs](#)
- [Containers and repositories](#) in pipelines
- [Triggers](#)
- [Variables](#)
- [Build number formats](#)

## Stage

A stage is a collection of related jobs. By default, stages run sequentially. Each stage starts only after the preceding stage is complete unless otherwise specified via the `dependsOn` property.

Use [approval checks](#) to manually control when a stage should run. These checks are commonly used to control deployments to production environments.

Checks are a mechanism available to the *resource owner*. They control when a stage in a pipeline consumes a resource. As an owner of a resource like an environment, you can define checks that are required before a stage that consumes the resource can start.

Currently, manual approval checks are supported on [environments](#). For more information, see [Approvals](#).

- [Schema](#)
- [Example](#)

```
stages:
- stage: string # name of the stage (A-Z, a-z, 0-9, and underscore)
 displayName: string # friendly name to display in the UI
 dependsOn: string | [string]
 condition: string
 variables: # several syntaxes, see specific section
 jobs: [job | templateReference]
```

Learn more about [stages](#), [conditions](#), and [variables](#).

# Job

A [job](#) is a collection of [steps](#) run by an [agent](#) or on a [server](#). Jobs can run [conditionally](#) and might depend on [earlier jobs](#).

- [Schema](#)
- [Example](#)

```
jobs:
- job: string # name of the job (A-Z, a-z, 0-9, and underscore)
 displayName: string # friendly name to display in the UI
 dependsOn: string | [string]
 condition: string
 strategy:
 parallel: # parallel strategy; see the following "Parallel" topic
 matrix: # matrix strategy; see the following "Matrix" topic
 maxParallel: number # maximum number of matrix jobs to run simultaneously
 continueOnError: boolean # 'true' if future jobs should run even if this job fails; defaults to 'false'
 pool: pool # see the following "Pool" schema
 workspace:
 clean: outputs | resources | all # what to clean up before the job runs
 container: containerReference # container to run this job inside of
 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: # several syntaxes, see specific section
 steps: [script | bash | pwsh | powershell | checkout | task | templateReference]
 services: { string: string | container } # container resources to run as a service container
 uses: # Any resources (repos or pools) required by this job that are not already referenced
 repositories: [string] # Repository references to Azure Git repositories
 pools: [string] # Pool names, typically when using a matrix strategy for the job
```

```
jobs:
- job: string # name of the job (A-Z, a-z, 0-9, and underscore)
 displayName: string # friendly name to display in the UI
 dependsOn: string | [string]
 condition: string
 strategy:
 parallel: # parallel strategy; see the following "Parallel" topic
 matrix: # matrix strategy; see the following "Matrix" topic
 maxParallel: number # maximum number of matrix jobs to run simultaneously
 continueOnError: boolean # 'true' if future jobs should run even if this job fails; defaults to 'false'
 pool: pool # see the following "Pool" schema
 workspace:
 clean: outputs | resources | all # what to clean up before the job runs
 container: containerReference # container to run this job inside of
 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: # several syntaxes, see specific section
 steps: [script | bash | pwsh | powershell | checkout | task | templateReference]
 services: { string: string | container } # container resources to run as a service container
```

For more information about workspaces, including clean options, see the [workspace](#) topic in [Jobs](#).

Learn more about [variables](#), [steps](#), [pools](#), and [server jobs](#).

## NOTE

If you have only one stage and one job, you can use [single-job syntax](#) as a shorter way to describe the steps to run.

## Container reference

A container is supported by jobs.

- [Schema](#)
- [Example](#)

```
container: string # Docker Hub image reference or resource alias
```

```
container:
 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
 # you can also use any of the other supported container attributes
```

## Strategies

The `matrix` and `parallel` keywords specify mutually exclusive strategies for duplicating a job.

### Matrix

Use of a matrix generates copies of a job, each with different input. These copies are useful for testing against different configurations or platform versions.

- [Schema](#)
- [Example](#)

```
strategy:
 matrix: { string1: { string2: string3 } }
 maxParallel: number
```

For each occurrence of `string1` in the matrix, a copy of the job is generated. The name `string1` is the copy's name and is appended to the name of the job. For each occurrence of `string2`, a variable called `string2` with the value `string3` is available to the job.

#### NOTE

Matrix configuration names must contain only basic Latin alphabet letters (A-Z and a-z), digits (0-9), and underscores (`_`). They must start with a letter. Also, their length must be 100 characters or fewer.

The optional `maxParallel` keyword specifies the maximum number of simultaneous matrix legs to run at once.

If `maxParallel` is unspecified or set to 0, no limit is applied.

If `maxParallel` is unspecified, no limit is applied.

#### NOTE

The `matrix` syntax doesn't support automatic job scaling but you can implement similar functionality using the `each` keyword. For an example, see [expressions](#).

### Parallel

This strategy specifies how many duplicates of a job should run. It's useful for slicing up a large test matrix. The [Visual Studio Test task](#) understands how to divide the test load across the number of scheduled jobs.

- [Schema](#)
- [Example](#)

```
strategy:
 parallel: number
```

## Deployment job

A [deployment job](#) is a special type of job. It's a collection of steps to run sequentially against the environment. In YAML pipelines, we recommend that you put your deployment steps in a deployment job.

- [Schema](#)
- [Example](#)

```
jobs:
- deployment: string # name of the deployment job, A-Z, a-z, 0-9, and underscore. The word "deploy" is a keyword and is unsupported as the deployment name.
 displayName: string # friendly name to display in the UI
 pool:
 name: string # Use only global level variables for defining a pool name. Stage/job level variables are not supported to define pool name.
 demands: string | [string]
 workspace:
 clean: outputs | resources | all # what to clean up before the job runs
 dependsOn: string
 condition: string
 continueOnError: boolean # 'true' if future jobs should run even if this job fails;
 defaults to 'false'
 container: containerReference # container to run this job inside
 services: { string: string | container } # container resources to run as a service container
 timeoutInMinutes: nonEmptyString # how long to run the job before automatically cancelling
 cancelTimeoutInMinutes: nonEmptyString # how much time to give 'run always even if cancelled tasks' before killing them
 variables: # several syntaxes, see specific section
 environment: string # target environment name and optionally a resource name to record the deployment history; format: <environment-name>.<resource-name>
 strategy:
 runOnce: #rolling, canary are the other strategies that are supported
 deploy:
 steps: [script | bash | pwsh | powershell | checkout | task | templateReference]
```

## Steps

A step is a linear sequence of operations that make up a job. Each step runs in its own process on an agent and has access to the pipeline workspace on a local hard drive. This behavior means environment variables aren't preserved between steps but file system changes are.

- [Schema](#)
- [Example](#)

```
steps: [script | bash | pwsh | powershell | checkout | task | templateReference]
```

For more information about steps, see the schema references for:

- [Script](#)

- [Bash](#)
- [pwsh](#)
- [PowerShell](#)
- [Checkout](#)
- [Task](#)
- [Step templates](#)

All steps, regardless of whether they're documented in this article, support the following properties:

- `displayName`
- `name`
- `condition`
- `continueOnError`
- `enabled`
- `env`
- `timeoutInMinutes`

## Variables

You can add hard-coded values directly, reference [variable groups](#), or insert via variable templates.

Specify variables at the pipeline, stage, or job level.

- [Schema](#)
- [Example](#)

For a simple set of hard-coded variables, use this mapping syntax:

```
variables: { string: string }
```

To include variable groups, switch to this sequence syntax:

```
variables:
- name: string # name of a variable
 value: string # value of the variable
- group: string # name of a variable group
```

You can repeat `name` / `value` pairs and `group`.

Variables can also be set as read only to [enhance security](#).

```
variables:
- name: myReadOnlyVar
 value: myValue
 readonly: true
```

You can also include [variables from templates](#).

## Template references

## NOTE

Be sure to see the full [template expression syntax](#), which is all forms of `${{ }}.`

You can export reusable sections of your pipeline to a separate file. These separate files are known as templates.

Azure Pipelines supports four kinds of templates:

- [Stage](#)
- [Job](#)
- [Step](#)
- [Variable](#)

You can also use templates to control what is allowed in a pipeline and to define how parameters can be used.

- [Parameter](#)

You can export reusable sections of your pipeline to separate files. These separate files are known as templates.

Azure DevOps Server 2019 supports these two kinds of templates:

- [Job](#)
- [Step](#)

Templates themselves can include other templates. Azure Pipelines supports a maximum of 50 unique template files in a single pipeline.

## Stage templates

You can define a set of stages in one file and use it multiple times in other files.

- [Schema](#)
- [Example](#)

In the main pipeline:

```
- template: string # name of template to include
 parameters: { string: any } # provided parameters
```

In the included template:

```
parameters: { string: any } # expected parameters
stages: [stage]
```

## Job templates

You can define a set of jobs in one file and use it multiple times in other files.

- [Schema](#)
- [Example](#)

In the main pipeline:

```
- template: string # name of template to include
 parameters: { string: any } # provided parameters
```

In the included template:

```
parameters: { string: any } # expected parameters
jobs: [job]
```

See [templates](#) for more about working with job templates.

## Step templates

You can define a set of steps in one file and use it multiple times in another file.

- [Schema](#)
- [Example](#)

In the main pipeline:

```
steps:
- template: string # reference to template
 parameters: { string: any } # provided parameters
```

In the included template:

```
parameters: { string: any } # expected parameters
steps: [script | bash | pwsh | powershell | checkout | task | templateReference]
```

See [templates](#) for more about working with templates.

## Variable templates

You can define a set of variables in one file and use it multiple times in other files.

- [Schema](#)
- [Example](#)

In the main pipeline:

```
- template: string # name of template file to include
 parameters: { string: any } # provided parameters
```

In the included template:

```
parameters: { string: any } # expected parameters
variables: [variable]
```

### NOTE

The `variables` keyword uses two syntax forms: sequence and mapping. In mapping syntax, all keys are variable names and their values are variable values. To use variable templates, you must use sequence syntax. Sequence syntax requires you to specify whether you're mentioning a variable (`name`), a variable group (`group`), or a template (`template`). See the [variables](#) topic for more.

## Parameters

You can use parameters in templates and pipelines.

- [Schema](#)

- [YAML Example](#)
- [Template Example](#)

The type and name fields are required when defining parameters. See all [parameter data types](#).

```
parameters:
- name: string # name of the parameter; required
 type: enum # data types, see below
 default: any # default value; if no default, then the parameter MUST be given by the user at runtime
 values: [string] # allowed list of values (for some data types)
```

## Types

DATA TYPE	NOTES
<code>string</code>	string
<code>number</code>	may be restricted to <code>values:</code> , otherwise any number-like string is accepted
<code>boolean</code>	<code>true</code> or <code>false</code>
<code>object</code>	any YAML structure
<code>step</code>	a single step
<code>stepList</code>	sequence of <a href="#">steps</a>
<code>job</code>	a single job
<code>jobList</code>	sequence of <a href="#">jobs</a>
<code>deployment</code>	a single deployment job
<code>deploymentList</code>	sequence of deployment <a href="#">jobs</a>
<code>stage</code>	a single stage
<code>stageList</code>	sequence of <a href="#">stages</a>

The step, stepList, job, jobList, deployment, deploymentList, stage, and stageList data types all use standard YAML schema format. This example includes string, number, boolean, object, step, and stepList.

```

parameters:
- name: myString
 type: string
 default: a string
- name: myMultiString
 type: string
 default: default
 values:
 - default
 - ubuntu
- name: myNumber
 type: number
 default: 2
 values:
 - 1
 - 2
 - 4
 - 8
 - 16
- name: myBoolean
 type: boolean
 default: true
- name: myObject
 type: object
 default:
 foo: FOO
 bar: BAR
 things:
 - one
 - two
 - three
 nested:
 one: apple
 two: pear
 count: 3
- name: myStep
 type: step
 default:
 script: echo my step
- name: mySteplist
 type: stepList
 default:
 - script: echo step one
 - script: echo step two

trigger: none

jobs:
- job: stepList
 steps: ${{ parameters.mySteplist }}
- job: myStep
 steps:
 - ${{ parameters.myStep }}

```

## Resources

A resource is any external service that is consumed as part of your pipeline. An example of a resource is another CI/CD pipeline that produces:

- Artifacts like Azure Pipelines or Jenkins.
- Code repositories like GitHub, Azure Repos, or Git.
- Container-image registries like Azure Container Registry or Docker hub.

Resources in YAML represent sources of pipelines, containers, repositories, and types. For more information on

Resources, [see here](#).

## General schema

```
resources:
 pipelines: [pipeline]
 builds: [build]
 repositories: [repository]
 containers: [container]
 packages: [package]
```

## Pipeline resource

If you have an Azure pipeline that produces artifacts, your pipeline can consume the artifacts by using the `pipeline` keyword to define a pipeline resource. You can also enable [pipeline-completion triggers](#).

- [Schema](#)
- [Example](#)

```
resources:
 pipelines:
 - pipeline: string # identifier for the resource used in pipeline resource variables
 project: string # project for the source; optional for current project
 source: string # name of the pipeline that produces an artifact
 version: string # the pipeline run number to pick the artifact, defaults to latest pipeline successful
 across all stages; Used only for manual or scheduled triggers
 branch: string # branch to pick the artifact, optional; defaults to all branches; Used only for manual
 or scheduled triggers
 tags: [string] # list of tags required on the pipeline to pickup default artifacts, optional; Used
 only for manual or scheduled triggers
 trigger: # triggers are not enabled by default unless you add trigger section to the resource
 branches: # branch conditions to filter the events, optional; Defaults to all branches.
 include: [string] # branches to consider the trigger events, optional; Defaults to all branches.
 exclude: [string] # branches to discard the trigger events, optional; Defaults to none.
 tags: [string] # list of tags to evaluate for trigger event, optional; 2020.1 and greater
 stages: [string] # list of stages to evaluate for trigger event, optional; 2020.1 and greater
```

### IMPORTANT

When you define a resource trigger, if its pipeline resource is from the same repo as the current pipeline, triggering follows the same branch and commit on which the event is raised. But if the pipeline resource is from a different repo, the current pipeline is triggered on the branch specified by the **Default branch for manual and scheduled builds** setting. For more information, see [Branch considerations for pipeline completion triggers](#).

### The pipeline resource metadata as predefined variables

In each run, the metadata for a pipeline resource is available to all jobs as these predefined variables:

```
resources.pipeline.<Alias>.projectName
resources.pipeline.<Alias>.projectId
resources.pipeline.<Alias>.pipelineName
resources.pipeline.<Alias>.pipelineId
resources.pipeline.<Alias>.runName
resources.pipeline.<Alias>.runId
resources.pipeline.<Alias>.runUri
resources.pipeline.<Alias>.sourceBranch
resources.pipeline.<Alias>.sourceCommit
resources.pipeline.<Alias>.sourceProvider
resources.pipeline.<Alias>.requestedFor
resources.pipeline.<Alias>.requestedForId
```

You can consume artifacts from a pipeline resource by using a `download` task. See the [download keyword](#).

## Container resource

[Container jobs](#) let you isolate your tools and dependencies inside a container. The agent launches an instance of your specified container then runs steps inside it. The `container` keyword lets you specify your container images.

[Service containers](#) run alongside a job to provide various dependencies like databases.

- [Schema](#)
- [Example](#)

```
resources:
 containers:
 - container: string # identifier (A-Z, a-z, 0-9, and underscore)
 image: string # container image name
 options: string # arguments to pass to container at startup
 endpoint: string # reference to a service connection for the private registry
 env: { string: string } # list of environment variables to add
 ports: [string] # ports to expose on the container
 volumes: [string] # volumes to mount on the container
 mapDockerSocket: bool # whether to map in the Docker daemon socket; defaults to true
 mountReadOnly: # volumes to mount read-only - all default to false
 externals: boolean # components required to talk to the agent
 tasks: boolean # tasks required by the job
 tools: boolean # installable tools like Python and Ruby
 work: boolean # the work directory
```

```
resources:
 containers:
 - container: string # identifier (A-Z, a-z, 0-9, and underscore)
 image: string # container image name
 options: string # arguments to pass to container at startup
 endpoint: string # reference to a service connection for the private registry
 env: { string: string } # list of environment variables to add
 ports: [string] # ports to expose on the container
 volumes: [string] # volumes to mount on the container
 mapDockerSocket: bool # whether to map in the Docker daemon socket; defaults to true
```

```
resources:
 containers:
 - container: string # identifier (A-Z, a-z, 0-9, and underscore)
 image: string # container image name
 options: string # arguments to pass to container at startup
 endpoint: string # reference to a service connection for the private registry
 env: { string: string } # list of environment variables to add
 ports: [string] # ports to expose on the container
 volumes: [string] # volumes to mount on the container
```

## Repository resource

If your pipeline has [templates in another repository](#), you must let the system know about that repository. The `repository` keyword lets you specify an external repository.

If your pipeline has [templates in another repository](#), or if you want to use [multi-repo checkout](#) with a repository that requires a service connection, you must let the system know about that repository. The `repository` keyword lets you specify an external repository.

- [Schema](#)

- Example

```
resources:
 repositories:
 - repository: string # identifier (A-Z, a-z, 0-9, and underscore)
 type: enum # see the following "Type" topic
 name: string # repository name (format depends on `type`)
 ref: string # ref name to use; defaults to 'refs/heads/main'
 endpoint: string # name of the service connection to use (for types that aren't Azure Repos)
 trigger: # CI trigger for this repository, no CI trigger if skipped (only works for Azure Repos)
 branches:
 include: [string] # branch names which will trigger a build
 exclude: [string] # branch names which will not
 tags:
 include: [string] # tag names which will trigger a build
 exclude: [string] # tag 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
```

## Type

Pipelines support the following values for the repository type: `git`, `github`, and `bitbucket`. The `git` type refers to Azure Repos Git repos.

- If you specify `type: git`, the `name` value refers to another repository in the same project. An example is `name: otherRepo`. To refer to a repo in another project within the same organization, prefix the name with that project's name. An example is `name: OtherProject/otherRepo`.
- If you specify `type: github`, the `name` value is the full name of the GitHub repo and includes the user or organization. An example is `name: Microsoft/vscode`. GitHub repos require a [GitHub service connection](#) for authorization.
- If you specify `type: bitbucket`, the `name` value is the full name of the Bitbucket Cloud repo and includes the user or organization. An example is `name: MyBitbucket/vscode`. Bitbucket Cloud repos require a [Bitbucket Cloud service connection](#) for authorization.

## Packages resource

You can consume NuGet and npm GitHub packages as a resource in YAML pipelines. When specifying package resources, set the package as `NuGet` or `npm`.

- [Schema](#)
- Example

```
resources:
 packages:
 - package: myPackageAlias # alias for the package resource
 type: Npm # type of the package NuGet/npm
 connection: GitHubConnectionName # Github service connection with the PAT type
 name: nugetTest/nodeapp # <Repository>/<Name of the package>
 version: 1.0.1 # Version of the packge to consume; Optional; Defaults to latest
 trigger: true # To enable automated triggers (true/false); Optional; Defaults to no triggers
```

## Triggers

- [Push trigger](#)
- [Pull request trigger](#)

- [Scheduled trigger](#)
- [Pipeline trigger](#)

#### NOTE

Trigger blocks can't contain variables or template expressions.

## Push trigger

A push trigger specifies which branches cause a continuous integration build to run. If you specify no push trigger, pushes to any branch trigger a build. Learn more about [triggers](#) and how to specify them.

- [Schema](#)
- [Example](#)

There are three distinct syntax options for the `trigger` keyword: a list of branches to include, a way to disable CI triggers, and the full syntax for complete control.

List syntax:

```
trigger: [string] # list of branch names
```

Disablement 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 (default)
 branches:
 include: [string] # branch names which will trigger a build
 exclude: [string] # branch names which will not
 tags:
 include: [string] # tag names which will trigger a build
 exclude: [string] # tag 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
```

If you specify an `exclude` clause without an `include` clause for `branches`, `tags`, or `paths`, it is equivalent to specifying `*` in the `include` clause.

```
trigger:
 batch: boolean # batch changes if true; start a new build for every push if false (default)
 branches:
 include: [string] # branch names which will trigger a build
 exclude: [string] # branch names which will not
 tags:
 include: [string] # tag names which will trigger a build
 exclude: [string] # tag 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
```

## IMPORTANT

When you specify a trigger, only branches that you explicitly configure for inclusion trigger a pipeline. Inclusions are processed first, and then exclusions are removed from that list. If you specify an exclusion but no inclusions, nothing triggers.

## PR trigger

A pull request trigger specifies which branches cause a pull request build to run. If you specify no pull request trigger, pull requests to any branch trigger a build. Learn more about [pull request triggers](#) and how to specify them.

## IMPORTANT

YAML PR triggers are supported only in GitHub and Bitbucket Cloud. If you use Azure Repos Git, you can configure a [branch policy for build validation](#) to trigger your build pipeline for validation.

## IMPORTANT

YAML PR triggers are supported only in GitHub. If you use Azure Repos Git, you can configure a [branch policy for build validation](#) to trigger your build pipeline for validation.

- [Schema](#)
- [Example](#)

There are three distinct syntax options for the `pr` keyword: a list of branches to include, a way to disable PR triggers, and the full syntax for complete control.

List syntax:

```
pr: [string] # list of branch names
```

Disablement syntax:

```
pr: none # will disable PR builds entirely; will not disable CI triggers
```

Full syntax:

```
pr:
 autoCancel: boolean # indicates whether additional pushes to a PR should cancel in-progress runs for the
 same PR. Defaults to true
 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:
 autoCancel: boolean # indicates whether additional pushes to a PR should cancel in-progress runs for the same PR. Defaults to true
 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
 drafts: boolean # For GitHub only, whether to build draft PRs, defaults to true
```

If you specify an `exclude` clause without an `include` clause for `branches` or `paths`, it is equivalent to specifying `*` in the `include` clause.

#### IMPORTANT

When you specify a pull request trigger, only branches that you explicitly configure for inclusion trigger a pipeline. Inclusions are processed first, and then exclusions are removed from that list. If you specify an exclusion but no inclusions, nothing triggers.

## Scheduled trigger

YAML scheduled triggers are unavailable in either this version of Azure DevOps Server or Visual Studio Team Foundation Server. You can use [scheduled triggers in the classic editor](#).

A scheduled trigger specifies a schedule on which branches are built. If you specify no scheduled trigger, no scheduled builds occur. Learn more about [scheduled triggers](#) and how to specify them.

- [Schema](#)
- [Example](#)

```
schedules:
- cron: string # cron syntax defining a schedule in UTC time
 displayName: string # friendly name given to a specific schedule
 branches:
 include: [string] # which branches the schedule applies to
 exclude: [string] # which branches to exclude from the schedule
 always: boolean # whether to always run the pipeline or only if there have been source code changes since the last successful scheduled run. The default is false.
```

#### NOTE

If you specify an `exclude` clause without an `include` clause for `branches`, it is equivalent to specifying `*` in the `include` clause.

## Pipeline trigger

Pipeline completion triggers are configured using a [pipeline resource](#). For more information, see [Pipeline completion triggers](#).

## Pool

The `pool` keyword specifies which [pool](#) to use for a job of the pipeline. A `pool` specification also holds information about the job's strategy for running.

In Azure DevOps Server 2019 you can specify a pool at the job level in YAML, and at the pipeline level in the

pipeline settings UI. In Azure DevOps Server 2019.1 you can also specify a pool at the pipeline level in YAML if you have a single implicit job.

You can specify a pool at the pipeline, stage, or job level.

The pool specified at the lowest level of the hierarchy is used to run the job.

- [Schema](#)
- [Example](#)

The full syntax is:

```
pool:
 name: string # name of the pool to run this job in
 demands: string | [string] # see the following "Demands" topic
 vmImage: string # name of the VM image you want to use; valid only in the Microsoft-hosted pool
```

If you use a Microsoft-hosted pool, choose an [available virtual machine image](#).

If you use a private pool and don't need to specify demands, you can shorten the syntax to:

```
pool: string # name of the private pool to run this job in
```

Learn more about [conditions](#) and [timeouts](#).

## Demands

The `demands` keyword is supported by private pools. You can check for the existence of a capability or a specific string.

- [Schema](#)
- [Example](#)

```
pool:
 demands: [string]
```

## Environment

The `environment` keyword specifies the [environment](#) or its resource that is targeted by a deployment job of the pipeline. An environment also holds information about the deployment strategy for running the steps defined inside the job.

- [Schema](#)
- [Example](#)

The full syntax is:

```
environment: # create environment and/or record deployments
 name: string # name of the environment to run this job on.
 resourceName: string # name of the resource in the environment to record the deployments against
 resourceId: number # resource identifier
 resourceType: string # type of the resource you want to target. Supported types - virtualMachine,
Kubernetes
 tags: string | [string] # tag names to filter the resources in the environment
strategy: # deployment strategy
 runOnce: # default strategy
 deploy:
 steps:
 - script: echo Hello world
```

If you specify an environment or one of its resources but don't need to specify other properties, you can shorten the syntax to:

```
environment: environmentName.resourceName
strategy: # deployment strategy
 runOnce: # default strategy
 deploy:
 steps:
 - script: echo Hello world
```

## Server

The `server` value specifies a [server job](#). Only server tasks like [invoking an Azure function app](#) can be run in a server job.

- [Schema](#)
- [Example](#)

When you use `server`, a job runs as a server job rather than an agent job.

```
pool: server
```

## Script

The `script` keyword is a shortcut for the [command-line task](#). The task runs a script using cmd.exe on Windows and Bash on other platforms.

- [Schema](#)
- [Example](#)

```
steps:
- script: string # contents of the script to run
 displayName: string # friendly name displayed in the UI
 name: string # identifier for this step (A-Z, a-z, 0-9, and underscore)
 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 to run this step; defaults to 'true'
 target:
 container: string # where this step will run; values are the container name or the word 'host'
 commands: enum # whether to process all logging commands from this step; values are `any` (default) or
`restricted`
 timeoutInMinutes: number
 env: { string: string } # list of environment variables to add
```

If you don't specify a command mode, you can shorten the `target` structure to:

```
- script:
 target: string # container name or the word 'host'
```

Learn more about [conditions](#), [timeouts](#), and [step targets](#).

## Bash

The `bash` keyword is a shortcut for the [shell script task](#). The task runs a script in Bash on Windows, macOS, and Linux.

- [Schema](#)
- [Example](#)

```
steps:
- bash: string # contents of the script to run
 displayName: string # friendly name displayed in the UI
 name: string # identifier for this step (A-Z, a-z, 0-9, and underscore)
 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 to run this step; defaults to 'true'
 target:
 container: string # where this step will run; values are the container name or the word 'host'
 commands: enum # whether to process all logging commands from this step; values are `any` (default) or
`restricted`
 timeoutInMinutes: number
 env: { string: string } # list of environment variables to add
```

If you don't specify a command mode, you can shorten the `target` structure to:

```
- bash:
 target: string # container name or the word 'host'
```

Learn more about [conditions](#), [timeouts](#), and [step targets](#).

## pwsh

The `pwsh` keyword is a shortcut for the [PowerShell task](#) when that task's `pwsh` value is set to `true`. The task

runs a script in PowerShell Core on Windows, macOS, and Linux.

- [Schema](#)
- [Example](#)

```
steps:
- pwsh: string # contents of the script to run
displayName: string # friendly name displayed in the UI
name: string # identifier for this step (A-Z, a-z, 0-9, and underscore)
errorActionPreference: enum # see the following "Error action preference" topic
ignoreLASTEXITCODE: boolean # see the following "Ignore last exit code" topic
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 to run this step; defaults to 'true'
timeoutInMinutes: number
env: { string: string } # list of environment variables to add
```

#### NOTE

Each PowerShell session lasts only for the duration of the job in which it runs. Tasks that depend on what has been bootstrapped must be in the same job as the bootstrap.

Learn more about [conditions](#) and [timeouts](#).

## PowerShell

The `powershell` keyword is a shortcut for the [PowerShell task](#). The task runs a script in Windows PowerShell.

- [Schema](#)
- [Example](#)

```
steps:
- powershell: string # contents of the script to run
displayName: string # friendly name displayed in the UI
name: string # identifier for this step (A-Z, a-z, 0-9, and underscore)
errorActionPreference: enum # see the following "Error action preference" topic
ignoreLASTEXITCODE: boolean # see the following "Ignore last exit code" topic
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 to run this step; defaults to 'true'
timeoutInMinutes: number
env: { string: string } # list of environment variables to add
```

#### NOTE

Each PowerShell session lasts only for the duration of the job in which it runs. Tasks that depend on what has been bootstrapped must be in the same job as the bootstrap.

Learn more about [conditions](#) and [timeouts](#).

### Error action preference

Unless otherwise specified, the error action preference defaults to the value `stop`, and the line

```
$ErrorActionPreference = 'stop'
```

When the error action preference is set to stop, errors cause PowerShell to terminate the task and return a nonzero exit code. The task is also marked as Failed.

- [Schema](#)
- [Example](#)

```
errorActionPreference: stop | continue | silentlyContinue
```

### Ignore last exit code

The last exit code returned from your script is checked by default. A nonzero code indicates a step failure, in which case the system appends your script with:

```
if ((Test-Path -LiteralPath variable:\LASTEXITCODE)) { exit $LASTEXITCODE }
```

If you don't want this behavior, specify `ignoreLASTEXITCODE: true`.

- [Schema](#)
- [Example](#)

```
ignoreLASTEXITCODE: boolean
```

Learn more about [conditions](#) and [timeouts](#).

## Publish

The `publish` keyword is a shortcut for the [Publish Pipeline Artifact task](#). The task publishes (uploads) a file or folder as a pipeline artifact that other jobs and pipelines can consume.

- [Schema](#)
- [Example](#)

```
steps:
- publish: string # path to a file or folder
 artifact: string # artifact name
 displayName: string # friendly name to display in the UI
```

Learn more about [publishing artifacts](#).

## Download

The `download` keyword is a shortcut for the [Download Pipeline Artifact task](#). The task downloads artifacts associated with the current run or from another Azure pipeline that is associated as a pipeline resource.

- [Schema](#)
- [Example](#)

```
steps:
- download: [current | pipeline resource identifier | none] # disable automatic download if "none"
 artifact: string ## artifact name, optional; downloads all the available artifacts if not specified
 patterns: string # patterns representing files to include; optional
 displayName: string # friendly name to display in the UI
```

## Artifact download location

Artifacts from the current pipeline are downloaded to `$(Pipeline.Workspace)/.`

Artifacts from the associated pipeline resource are downloaded to `$(Pipeline.Workspace)/<pipeline resource identifier>/.`

## Automatic download in deployment jobs

All available artifacts from the current pipeline and from the associated pipeline resources are automatically downloaded in deployment jobs and made available for your deployment. To prevent downloads, specify

```
download: none .
```

Learn more about [downloading artifacts](#).

## Checkout

Nondeployment jobs automatically check out source code. Use the `checkout` keyword to configure or suppress this behavior.

- [Schema](#)
- [Example](#)

```
steps:
- checkout: self # self represents the repo where the initial Pipelines YAML file was found
 clean: boolean # if true, execute `execute git clean -ffdx && git reset --hard HEAD` before fetching
 fetchDepth: number # the depth of commits to ask Git to fetch (applies to submodules too if they're
 enabled); defaults to no limit
 lfs: boolean # whether to download Git-LFS files; defaults to false
 submodules: true | recursive # set to 'true' for a single level of submodules or 'recursive' to get
 submodules of submodules; defaults to not checking out submodules
 path: string # path to check out source code, relative to the agent's build directory (e.g. _work\1);
 defaults to a directory called `s`
 persistCredentials: boolean # if 'true', leave the OAuth token in the Git config after the initial fetch;
 defaults to false
```

```
steps:
- checkout: self | none | repository name # self represents the repo where the initial Pipelines YAML file
 was found
 clean: boolean # if true, run `execute git clean -ffdx && git reset --hard HEAD` before fetching
 fetchDepth: number # the depth of commits to ask Git to fetch; defaults to no limit
 lfs: boolean # whether to download Git-LFS files; defaults to false
 submodules: true | recursive # set to 'true' for a single level of submodules or 'recursive' to get
 submodules of submodules; defaults to not checking out submodules
 path: string # path to check out source code, relative to the agent's build directory (e.g. _work\1);
 defaults to a directory called `s`
 persistCredentials: boolean # if 'true', leave the OAuth token in the Git config after the initial fetch;
 defaults to false
```

## NOTE

In addition to the cleaning option available using `checkout`, you can also configuring cleaning in a workspace. For more information about workspaces, including clean options, see the [workspace](#) topic in [Jobs](#).

To avoid syncing sources at all:

```
steps:
- checkout: none
```

## NOTE

If you're running the agent in the Local Service account and want to modify the current repository by using git operations or loading git submodules, give the proper permissions to the Project Collection Build Service Accounts user.

```
- checkout: self
submodules: true
persistCredentials: true
```

To check out multiple repositories in your pipeline, use multiple `checkout` steps:

```
- checkout: self
- checkout: git://MyProject/MyRepo
- checkout: MyGitHubRepo # Repo declared in a repository resource
```

For more information, see [Check out multiple repositories in your pipeline](#).

## Task

[Tasks](#) are the building blocks of a pipeline. There's a [catalog of tasks](#) available to choose from.

- [Schema](#)
- [Example](#)

```
steps:
- 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 (A-Z, a-z, 0-9, and underscore)
condition: string
continueOnError: boolean # 'true' if future steps should run even if this step fails; defaults to 'false'
enabled: boolean # whether to run this step; defaults to 'true'
target:
 container: string # where this step will run; values are the container name or the word 'host'
 commands: enum # whether to process all logging commands from this step; values are `any` (default) or
`restricted`
 settableVariables: string # what variables are allowed; defaults to all; can be `none` or a list of
allowed vars
 timeoutInMinutes: number
inputs: { string: string } # task-specific inputs
env: { string: string } # list of environment variables to add
```

If you don't specify a command mode, you can shorten the `target` structure to:

```
- task:
 target: string # container name or the word 'host'
```

Learn more about [conditions](#), [timeouts](#), and [step targets](#).

## Syntax highlighting

Syntax highlighting is available for the pipeline schema via a Visual Studio Code extension. You can [download Visual Studio Code](#), [install the extension](#), and [check out the project on GitHub](#). The extension includes a [JSON schema](#) for validation.

You also can obtain a schema that's specific to your organization (that is, it contains installed custom tasks) from the [Azure DevOps REST API yamlschema endpoint](#).

# Expressions

5/19/2021 • 19 minutes to read • [Edit Online](#)

Azure Pipelines | TFS 2018 | TFS 2017.3

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

Expressions can be used in many places where you need to specify a string, boolean, or number value when authoring a pipeline. The most common use of expressions is in [conditions](#) to determine whether a job or step should run.

```
Expressions are used to define conditions for a step, job, or stage
steps:
- task: ...
 condition: <expression>
```

Another common use of expressions is in defining variables. Expressions can be evaluated at [compile time](#) or at [run time](#). Compile time expressions can be used anywhere; runtime expressions can be used in variables and conditions. Runtime expressions are intended as a way to compute the contents of variables and state (example: [condition](#) ).

```
Two examples of expressions used to define variables
The first one, a, is evaluated when the YAML file is compiled into a plan.
The second one, b, is evaluated at runtime.
Note the syntax ${{} for compile time and ${[] for runtime expressions.
variables:
 a: ${{ <expression> }}
 b: ${[<expression>]}
```

The difference between runtime and compile time expression syntaxes is primarily what context is available. In a compile-time expression (`${{ <expression> }}`), you have access to [parameters](#) and statically defined [variables](#). In a runtime expression (`[$ <expression> ]`), you have access to more [variables](#) but no [parameters](#).

In this example, a runtime expression sets the value of `$(isMain)`. A static variable in a compile expression sets the value of `$(compileVar)`.

```
variables:
 staticVar: 'my value' # static variable
 compileVar: ${{ variables.staticVar }} # compile time expression
 isMain: ${[eq(variables['Build.SourceBranch'], 'refs/heads/main')]} # runtime expression

steps:
- script:
 - echo ${variables.staticVar} # outputs my value
 - echo $(compileVar) # outputs my value
 - echo $(isMain) # outputs True
```

An expression can be a literal, a reference to a variable, a reference to a dependency, a function, or a valid nested combination of these.

## Literals

As part of an expression, you can use boolean, null, number, string, or version literals.

```
Examples
variables:
 someBoolean: ${{ true }} # case insensitive, so True or TRUE also works
 someNumber: ${{ -1.2 }}
 someString: ${{ 'a b c' }}
 someVersion: ${{ 1.2.3 }}
```

### Boolean

`True` and `False` are boolean literal expressions.

### Null

Null is a special literal expression that's returned from a dictionary miss, e.g. (`variables['noSuch']`). Null can be the output of an expression but cannot be called directly within an expression.

### 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.'
```

You can use a pipe character (`|`) for multiline strings.

```
myKey: |
 one
 two
 three
```

### 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`.

## Variables

As part of an expression, 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 `_`

Depending on the execution context, different variables are available.

- If you create pipelines using YAML, then [pipeline variables](#) are available.

- If you create build pipelines using classic editor, then [build variables](#) are available.
- If you create release pipelines using classic editor, then [release variables](#) are available.

Variables are always strings. If you want to use typed values, then you should use [parameters](#) instead.

## Functions

The following built-in functions can be used in expressions.

### **and**

- Evaluates to `True` if all parameters are `True`
- Min parameters: 2. Max parameters: N
- Casts parameters to Boolean for evaluation
- Short-circuits after first `False`
- Example: `and(eq(variables.letters, 'ABC'), eq(variables.numbers, 123))`

### **coalesce**

- Evaluates the parameters in order, and returns the value that does not equal null or empty-string.
- Min parameters: 2. Max parameters: N
- Example: `coalesce(variables.couldBeNull, variables.couldAlsoBeNull, 'literal so it always works')`

### **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
- Example: `contains('ABCDE', 'BCD')` (returns True)

### **containsValue**

- Evaluates `True` if the left parameter is an array, and any item equals the right parameter. Also evaluates `True` if the left parameter is an object, and the value of any property equals the right parameter.
- Min parameters: 2. Max parameters: 2
- If the left parameter is an array, convert each item to match the type of the right parameter. If the left parameter is an object, convert the value of each property to match the type of the right parameter. The equality comparison for each specific item evaluates `False` if the conversion fails.
- Ordinal ignore-case comparison for Strings
- Short-circuits after the first match

#### **NOTE**

There is no literal syntax in a YAML pipeline for specifying an array. This function is of limited use in general pipelines. It's intended for use in the [pipeline decorator context](#) with system-provided arrays such as the list of steps.

### **convertToJson**

- Take a complex object and outputs it as JSON.
- Min parameters: 1. Max parameters: 1.

```

parameters:
 - name: listOfValues
 type: object
 default:
 this_is:
 a_complex: object
 with:
 - one
 - two

steps:
 - script: |
 echo "${{ convertToJson(parameters.listOfValues) }}"

```

```

Example output
{
 this_is: {
 a_complex: object,
 with: [
 one,
 two
]
 }
}

```

## counter

- This function can only be used in an expression that defines a variable. It cannot be used as part of a condition for a step, job, or stage.
- Evaluates a number that is incremented with each run of a pipeline.
- Parameters: 2. `prefix` and `seed`.
- Prefix is a string expression. A separate value of counter is tracked for each unique value of prefix
- Seed is the starting value of the counter

You can create a counter that is automatically incremented by one in each execution of your pipeline. When you define a counter, you provide a `prefix` and a `seed`. Here is an example that demonstrates this.

```

variables:
 major: 1
 # define minor as a counter with the prefix as variable major, and seed as 100.
 minor: ${counter(variables['major'], 100)}

steps:
 - bash: echo $(minor)

```

The value of `minor` in the above example in the first run of the pipeline will be 100. In the second run it will be 101, provided the value of `major` is still 1.

If you edit the YAML file, and update the value of the variable `major` to be 2, then in the next run of the pipeline, the value of `minor` will be 100. Subsequent runs will increment the counter to 101, 102, 103, ...

Later, if you edit the YAML file, and set the value of `major` back to 1, then the value of the counter resumes where it left off for that prefix. In this example, it resumes at 102.

Here is another example of setting a variable to act as a counter that starts at 100, gets incremented by 1 for every run, and gets reset to 100 every day.

## NOTE

`pipeline.startTime` is not available outside of expressions. `pipeline.startTime` formats `system.pipelineStartTime` into a date and time object so that it is available to work with expressions. The default time zone for `pipeline.startTime` is UTC. You can [change the time zone](#) for your organization.

```
jobs:
- job:
 variables:
 a: ${counter(format('{0:yyyyMMdd}', pipeline.startTime), 100)}
 steps:
 - bash: echo $(a)
```

Here is an example of having a counter that maintains a separate value for PRs and CI runs.

```
variables:
patch: ${counter(variables['build.reason'], 0)}
```

Counters are scoped to a pipeline. In other words, its value is incremented for each run of that pipeline. There are no project-scoped counters.

## 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
- Example: `endsWith('ABCDE', 'DE')` (returns True)

## 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
- Example: `eq(variables.letters, 'ABC')`

## format

- Evaluates the trailing parameters and inserts them into the leading parameter string
- Min parameters: 1. Max parameters: N
- Example: `format('Hello {0} {1}', 'John', 'Doe')`
- Uses [.NET custom date and time format specifiers](#) for date formatting (`yyyy`, `yy`, `MM`, `M`, `dd`, `d`, `HH`, `H`, `m`, `mm`, `ss`, `s`, `f`, `ff`, `ffff`, `K`)
- Example: `format('{0:yyyyMMdd}', pipeline.startTime)`. In this case `pipeline.startTime` is a special date time object variable.
- Escape by doubling braces. For example: `format('literal left brace {{ and literal right brace }}')`

## 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

- Example: `ge(5, 5)` (returns True)

## 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
- Example: `gt(5, 2)` (returns True)

## 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
- Example: `in('B', 'A', 'B', 'C')` (returns True)

## join

- Concatenates all elements in the right parameter array, separated by the left parameter string.
- Min parameters: 2. Max parameters: 2
- Each element in the array is converted to a string. Complex objects are converted to empty string.
- If the right parameter is not an array, the result is the right parameter converted to a string.

In this example, a semicolon gets added between each item in the array. The parameter type is an object.

```

parameters:
- name: myArray
 type: object
 default:
 - FOO
 - BAR
 - ZOO

variables:
A: ${{ join(';',parameters.myArray) }}

steps:
- script: echo $A # outputs FOO;BAR;ZOO

```

## 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
- Example: `le(2, 2)` (returns True)

## length

- Returns the length of a string or an array, either one that comes from the system or that comes from a parameter
- Min parameters: 1. Max parameters 1
- Example: `length('fabrikam')` returns 8

## **lower**

- Converts a string or variable value to all lowercase characters
- Min parameters: 1. Max parameters 1
- Returns the lowercase equivalent of a string
- Example: `lower('FOO')` returns `foo`

## **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
- Example: `lt(2, 5)` (returns True)

## **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
- Example: `ne(1, 2)` (returns True)

## **not**

- Evaluates `True` if parameter is `False`
- Min parameters: 1. Max parameters: 1
- Converts value to Boolean for evaluation
- Example: `not(eq(1, 2))` (returns True)

## **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
- Example: `notIn('D', 'A', 'B', 'C')` (returns True)

## **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`
- Example: `or(eq(1, 1), eq(2, 3))` (returns True, short-circuits)

## **replace**

- Returns a new string in which all instances of a string in the current instance are replaced with another string
- Min parameters: 3. Max parameters: 3
- `replace(a, b, c)` : returns a, with all instances of b replaced by c
- Example:

```
replace('https://www.tinfoilsecurity.com/saml/consume','https://www.tinfoilsecurity.com','http://server')
(returns http://server/saml/consume)
```

## **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
- Example: `startsWith('ABCDE', 'AB')` (returns True)

## **upper**

- Converts a string or variable value to all uppercase characters
- Min parameters: 1. Max parameters 1
- Returns the uppercase equivalent of a string
- Example: `upper('bah')` returns `BAH`

## **xor**

- Evaluates `True` if exactly one parameter is `True`
- Min parameters: 2. Max parameters: 2
- Casts parameters to Boolean for evaluation
- Example: `xor(True, False)` (returns True)

# Job status check functions

You can use the following status check functions as expressions in conditions, but not in variable definitions.

## **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.
  - If the previous job succeeded but a dependency further upstream failed,  
`succeeded('previousJobName')` will return true. When you just use `dependsOn: previousJobName`, it will fail because all of the upstream dependencies were not successful. To only evaluate the previous job, use `succeeded('previousJobName')` in a condition.
  - With job names as arguments, evaluates to `True` if all of those jobs succeeded or partially succeeded.
  - Evaluates to `False` if the pipeline is canceled.

## **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.

## Conditional insertion

You can use an `if` clause to conditionally assign the value or a variable or set inputs for tasks. Conditionals only work when using template syntax.

For templates, you can use conditional insertion when adding a sequence or mapping. Learn more about [conditional insertion in templates](#).

### Conditionally assign a variable

```
variables:
${{ if eq(variables['Build.SourceBranchName'], 'main') }}: # only works if you have a main branch
 stageName: prod

pool:
 vmImage: 'ubuntu-latest'

steps:
- script: echo ${variables.stageName}
```

### Conditionally set a task input

```
pool:
 vmImage: 'ubuntu-latest'

steps:
- task: PublishPipelineArtifact@1
 inputs:
 targetPath: '$(Pipeline.Workspace)'
 ${{ if eq(variables['Build.SourceBranchName'], 'main') }}:
 artifact: 'prod'
 ${{ if ne(variables['Build.SourceBranchName'], 'main') }}:
 artifact: 'dev'
 publishLocation: 'pipeline'
```

## Each keyword

You can use the `each` keyword to loop through parameters with the object type.

```
parameters:
- name: listOfStrings
 type: object
 default:
 - one
 - two

steps:
- ${{ each value in parameters.listOfStrings }}:
 - script: echo ${value}
```

# Dependencies

Expressions can use the dependencies context to reference previous jobs or stages. You can use dependencies to:

- Reference the job status of a previous job
- Reference the stage status of a previous stage
- Reference output variables in the previous job in the same stage
- Reference output variables in the previous stage in a stage
- Reference output variables in a job in a previous stage in the following stage

The context is called `dependencies` for jobs and stages and works much like variables. Inside a job, if you refer to an output variable from a job in another stage, the context is called `stageDependencies`.

If you experience issues with output variables having quote characters (' or ") in them, see [this troubleshooting guide](#).

## Stage to stage dependencies

Structurally, the `dependencies` object is a map of job and stage names to `results` and `outputs`. Expressed as JSON, it would look like:

```
"dependencies": {
 "<STAGE_NAME>": {
 "result": "Succeeded|SucceededWithIssues|Skipped|Failed|Canceled",
 "outputs": {
 "jobName.stepName.variableName": "value"
 }
 },
 "...": {
 // another stage
 }
}
```

Use this form of `dependencies` to map in variables or check conditions at a stage level. In this example, Stage B runs whether Stage A is successful or skipped.

### NOTE

The following examples use standard pipeline syntax. If you're using deployment pipelines, both variable and conditional variable syntax will differ. For information about the specific syntax to use, see [Deployment jobs](#).

```
stages:
- stage: A
 condition: false
 jobs:
 - job: A1
 steps:
 - script: echo Job A1
- stage: B
 condition: in(dependencies.A.result, 'Succeeded', 'SucceededWithIssues', 'Skipped')
 jobs:
 - job: B1
 steps:
 - script: echo Job B1
```

Stages can also use output variables from another stage. In this example, Stage B depends on a variable in Stage A.

```

stages:
- stage: A
 jobs:
 - job: A1
 steps:
 - bash: echo "##vso[task.setvariable variable=shouldrun;isOutput=true]true"
 # or on Windows:
 # - script: echo ##vso[task.setvariable variable=shouldrun;isOutput=true]true
 name: printvar

- stage: B
 condition: and(succeeded(), eq(dependencies.A.outputs['A1.printvar.shouldrun'], 'true'))
 dependsOn: A
 jobs:
 - job: B1
 steps:
 - script: echo hello from Stage B

```

#### NOTE

By default, each stage in a pipeline depends on the one just before it in the YAML file. If you need to refer to a stage that isn't immediately prior to the current one, you can override this automatic default by adding a `dependsOn` section to the stage.

#### Job to job dependencies within one stage

At the job level within a single stage, the `dependencies` data doesn't contain stage-level information.

```

"dependencies": {
 "<JOB_NAME>": {
 "result": "Succeeded|SucceededWithIssues|Skipped|Failed|Canceled",
 "outputs": {
 "stepName.variableName": "value1"
 }
 },
 "...": {
 // another job
 }
}

```

In this example, Job A will always be skipped and Job B will run. Job C will run, since all of its dependencies either succeed or are skipped.

```

jobs:
- job: a
 condition: false
 steps:
 - script: echo Job A
- job: b
 steps:
 - script: echo Job B
- job: c
 dependsOn:
 - a
 - b
 condition: |
 and
 (
 in(dependencies.a.result, 'Succeeded', 'SucceededWithIssues', 'Skipped'),
 in(dependencies.b.result, 'Succeeded', 'SucceededWithIssues', 'Skipped')
)
 steps:
 - script: echo Job C

```

In this example, Job B depends on an output variable from Job A.

```

jobs:
- job: A
 steps:
 - bash: echo "##vso[task.setvariable variable=shouldrun;isOutput=true]true"
 # or on Windows:
 # - script: echo ##vso[task.setvariable variable=shouldrun;isOutput=true]
 name: printvar

- job: B
 condition: and(succeeded(), eq(dependencies.A.outputs['printvar.shouldrun'], 'true'))
 dependsOn: A
 steps:
 - script: echo hello from B

```

## Job to job dependencies across stages

At the job level, you can also reference outputs from a job in a previous stage. This requires using the `stageDependencies` context.

```

"stageDependencies": {
 "<STAGE_NAME>": {
 "<JOB_NAME>": {
 "result": "Succeeded|SucceededWithIssues|Skipped|Failed|Canceled",
 "outputs": {
 "stepName.variableName": "value"
 }
 },
 "...": {
 // another job
 }
 },
 "...": {
 // another stage
 }
}

```

In this example, job B1 will run whether job A1 is successful or skipped. Job B2 will check the value of the output variable from job A1 to determine whether it should run.

```

trigger: none

pool:
 vmImage: 'ubuntu-latest'

stages:
- stage: A
 jobs:
 - job: A1
 steps:
 - bash: echo "##vso[task.setvariable variable=shouldrun;isOutput=true]true"
 # or on Windows:
 # - script: echo ##vso[task.setvariable variable=shouldrun;isOutput=true]true
 name: printvar

- stage: B
 dependsOn: A
 jobs:
 - job: B1
 condition: in(stageDependencies.A.A1.result, 'Succeeded', 'SucceededWithIssues', 'Skipped')
 steps:
 - script: echo hello from Job B1
 - job: B2
 condition: eq(stageDependencies.A.A1.outputs['printvar.shouldrun'], 'true')
 steps:
 - script: echo hello from Job B2

```

## Stage depending on job output

If no changes are required after a build, you might want to skip a stage in a pipeline under certain conditions. An example is when you're using Terraform Plan, and you want to trigger approval and apply only when the plan contains changes.

When you use this condition on a stage, you must use the `dependencies` variable, not `stageDependencies`.

The following example is a simple script that sets a variable (use your actual information from Terraform Plan) in a step in a stage, and then invokes the second stage only if the variable has a specific value.

```

stages:
- stage: plan_dev
 jobs:
 - job: terraform_plan_dev
 steps:
 - bash: echo '##vso[task.setvariable variable=terraform_plan_exitcode;isOutput=true]2'
 name: terraform_plan
- stage: apply_dev
 dependsOn: plan_dev
 condition: eq(dependencies.plan_dev.outputs['terraform_plan_dev.terraform_plan.terraform_plan_exitcode'], '2')
 jobs:
 - job: part_b
 steps:
 - bash: echo "BA"

```

## Filtered arrays

When operating on a collection of items, you can use the `*` syntax to apply a filtered array. A filtered array returns all objects/elements regardless their names.

As an example, consider an array of objects named `foo`. We want to get an array of the values of the `id` property in each object in our array.

```
[
 { "id": 1, "a": "avalue1"},
 { "id": 2, "a": "avalue2"},
 { "id": 3, "a": "avalue3"}
]
```

We could do the following:

```
foo.*.id
```

This tells the system to operate on `foo` as a filtered array and then select the `id` property.

This would return:

```
[1, 2, 3]
```

## Type casting

Values in an expression may be converted from one type to another as the expression gets evaluated. When an expression is evaluated, the parameters are coalesced to the relevant data type and then turned back into strings.

For example, in this YAML, the values `True` and `False` are converted to `1` and `0` when the expression is evaluated. The function `lt()` returns `True` when the left parameter is less than the right parameter.

```
variables:
 firstEval: ${[lt(False, True)]} # 0 vs. 1, True
 secondEval: ${[lt(True, False)]} # 1 vs. 0, False

steps:
 - script: echo ${firstEval}
 - script: echo ${secondEval}
```

In this example, the values `variables.emptyString` and the empty string both evaluate as empty strings. The function `coalesce()` evaluates the parameters in order, and returns the first value that does not equal null or empty-string.

```
variables:
 coalesceLiteral: ${coalesce(variables.emptyString, '', 'literal value')}

steps:
 - script: echo ${coalesceLiteral} # outputs literal value
```

Detailed conversion rules are listed further below.

FROM / TO	BOOLEAN	NULL	NUMBER	STRING	VERSION
Boolean	-	-	Yes	Yes	-
Null	Yes	-	Yes	Yes	-
Number	Yes	-	-	Yes	Partial
String	Yes	Partial	Partial	-	Partial

FROM / TO	BOOLEAN	NULL	NUMBER	STRING	VERSION
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.

## FAQ

**I want to do something that is not supported by expressions. What options do I have for extending Pipelines functionality?**

You can customize your Pipeline with a script that includes an expression. For example, this snippet takes the `BUILD_BUILDPART` variable and splits it with Bash. This script outputs two new variables, `$MAJOR_RUN` and `$MINOR_RUN`, for the major and minor run numbers. The two variables are then used to create two pipeline variables, `$major` and `$minor` with `task.setvariable`. These variables are available to downstream steps. To share variables across pipelines see [Variable groups](#).

```
steps:
- bash: |
 MAJOR_RUN=$(echo $BUILD_BUILDSNUMBER | cut -d '.' -f1)
 echo "This is the major run number: $MAJOR_RUN"
 echo "##vso[task.setvariable variable=major]$MAJOR_RUN"

 MINOR_RUN=$(echo $BUILD_BUILDSNUMBER | cut -d '.' -f2)
 echo "This is the minor run number: $MINOR_RUN"
 echo "##vso[task.setvariable variable=minor]$MINOR_RUN"

- bash: echo "My pipeline variable for major run is $(major)"
- bash: echo "My pipeline variable for minor run is $(minor)"
```

# File matching patterns reference

6/9/2020 • 2 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - 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. See [fnmatch](#) for a full syntax guide.

### Match characters

Most characters are used as exact matches. What counts as an "exact" match is platform-dependent: the Windows filesystem is case-insensitive, so the pattern "ABC" would match a file called "abc". On case-sensitive filesystems, that pattern and name would not match.

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. You can include a pattern, exclude a subset of it, and then re-include a subset of that: this is known as an "interleaved" pattern.

Multiple `!` flips the meaning. See [examples](#).

You must define an include pattern before an exclude one. 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 on Linux and macOS. Most of the time, Windows agents accept `/`. Occasions where the Windows separator (`\`) must be used are documented.

# 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
```

#### Recursive wildcard examples

Given the pattern `**/*.ext` and files:

```
sample1/A.ext
sample1/B.ext
sample2/C.ext
sample2/D.not
```

The pattern would match:

```
sample1/A.ext
sample1/B.ext
sample2/C.ext
```

### 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
```

#### Folder exclude

Given the pattern:

```
**
!sample/**
```

and files:

```
ConsoleHost.exe
ConsoleHost.pdb
ConsoleHost.xml
sample/Fabrikam.dll
sample/Fabrikam.pdb
sample/Fabrikam.xml
```

The pattern would match:

```
ConsoleHost.exe
ConsoleHost.pdb
ConsoleHost.xml
```

# File transforms and variable substitution reference

6/9/2021 • 8 minutes to read • [Edit Online](#)

Azure Pipelines | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 | TFS 2017

## NOTE

In Microsoft Team Foundation Server (TFS) 2018 and previous versions, build and release *pipelines* are called *definitions*, *runs* are called *builds*, *service connections* are called *service endpoints*, *stages* are called *environments*, and *jobs* are called *phases*.

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. However, file transforms and variable substitution are **not confined to web app files**. You can use these techniques with any XML or JSON files.

## NOTE

File transforms and variable substitution are also supported by the separate [File Transform task](#) for use in Azure Pipelines. You can use the File Transform task to apply file transformations and variable substitutions on any configuration and parameters files.

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

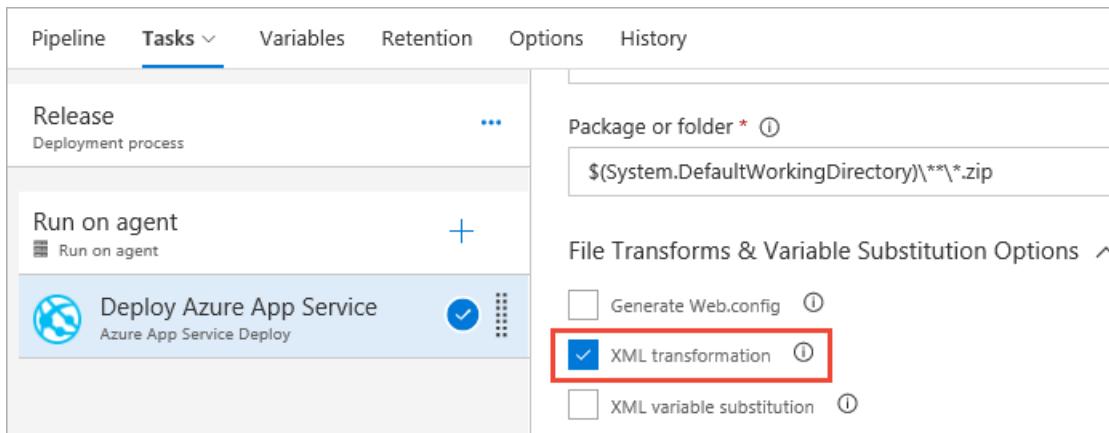
```
<?xml version="1.0"?>
<configuration xmlns:xdt="http://schemas.microsoft.com/XML-Document-Transform">
 <connectionStrings>
 <add name="MyDB"
 connectionString="Data Source=ReleaseSQLServer;Initial Catalog=MyReleaseDB;Integrated Security=True"
 xdt:Transform="Insert" />
 </connectionStrings>
 <appSettings>
 <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 a 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>
```

### XML transformation notes

- 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.
- 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.
- Set the **Build Action** property for each of the transformation files (`Web.config`) to **Content** so that the files are copied to the root folder.

```

...
<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. If you are looking to substitute values outside of these elements you can use a (`parameters.xml`) file, however you will need to use a 3rd party pipeline task to handle the variable substitution.

### 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="__AdminPassword__" />
 </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.

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>
 <configSections>
 <section name="entityFramework" />
 </configSections>
 <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>

```

## XML variable substitution notes

- 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:

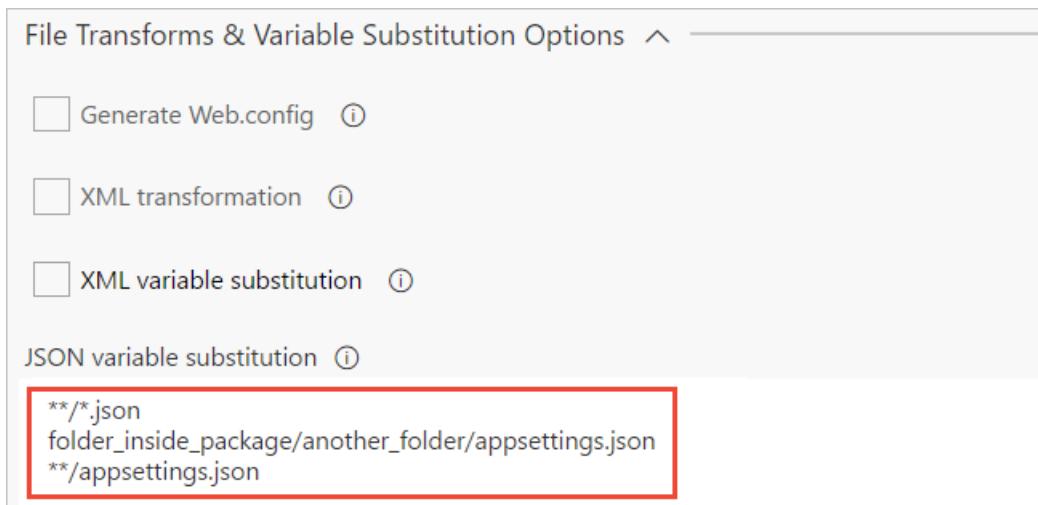
```
{
 "Data": {
 "DefaultConnection": {
 "ConnectionString": "Data Source=(LocalDb)\MSDB;AttachDbFilename=aspcore-local.mdf;"
 },
 "DebugMode": "enabled",
 "DBAccess": {
 "Administrators": ["Admin-1", "Admin-2"],
 "Users": ["Vendor-1", "Vendor-3"]
 },
 "FeatureFlags": {
 "Preview": [
 {
 "newUI": "AllAccounts"
 },
 {
 "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.

- [Classic](#)

- [YAML](#)

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.DefaultConnectionString	Data Source= (prodDB)\MSDB;AttachDb Filename=prod.mdf;	No	Release
Data.DBAccess.Users.0	Admin-3	Yes	Release
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"
 }
]
 }
 }
}
...

```

### JSON variable substitution notes

- 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`. However, the [file transform task](#) has the ability to transform entire arrays in JSON files. You can also use `DBAccess.Users = ["NewUser1", "NewUser2", "NewUser3"]`.
- 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.
- Variable substitution is applied for only the JSON keys predefined in the object hierarchy. It does not create new keys.
- If a variable name includes periods ("."), the transformation will attempt to locate the item within the hierarchy. For example, if the variable name is `first.second.third`, the transformation process will search for:

```

"first" : {
 "second": {
 "third" : "value"
 }
}

```

as well as `"first.second.third" : "value"`.

# Logging commands

5/26/2021 • 8 minutes to read • [Edit Online](#)

Logging commands are how [tasks](#) and scripts communicate with the agent. They cover actions like creating new [variables](#), marking a step as failed, and uploading [artifacts](#). Logging commands are useful when you are troubleshooting a pipeline.

TYPE	COMMANDS
Task commands	<a href="#">LogIssue</a> , <a href="#">SetProgress</a> , <a href="#">LogDetail</a> , <a href="#">SetVariable</a> , <a href="#">SetEndpoint</a> , <a href="#">AddAttachment</a> , <a href="#">UploadSummary</a> , <a href="#">UploadFile</a> , <a href="#">PrependPath</a>
Artifact commands	<a href="#">Associate</a> , <a href="#">Upload</a>
Build commands	<a href="#">UploadLog</a> , <a href="#">UpdateBuildNumber</a> , <a href="#">AddBuildTag</a>
Release commands	<a href="#">UpdateReleaseName</a>

## Logging command format

The general format for a logging command is:

```
##vso[area.action property1=value;property2=value;...]message
```

There are also a few formatting commands with a slightly different syntax:

```
##[command]message
```

To invoke a logging command, echo the command via standard output.

- [Bash](#)
- [PowerShell](#)

```
#!/bin/bash
echo "##vso[task.setvariable variable=testvar;]testvalue"
```

File paths should be given as absolute paths: rooted to a drive on Windows, or beginning with `/` on Linux and macOS.

### NOTE

Please note that there is no support of 'set -x' command being executed before logging command on Linux and macOS.

## Formatting commands

## NOTE

Use UTF-8 formatting for logging commands.

These commands are messages to the log formatter in Azure Pipelines. They mark specific log lines as errors, warnings, collapsible sections, and so on.

The formatting commands are:

```
##[group]Beginning of a group
##[warning]Warning message
##[error]Error message
##[section]Start of a section
##[debug]Debug text
##[command]Command-line being run
##[endgroup]
```

Those commands will render in the logs like this:

```
▼ Beginning of a group
 ##[warning]Warning message
 ##[error]Error message
 Start of a section
 ##[debug]Debug text
 Command-line being run
 Finishing: CmdLine
```

That block of commands can also be collapsed, and looks like this:

```
► Beginning of a group
```

## Task commands

### LogIssue: Log an error or warning

```
##vso[task.logissue]error/warning message
```

#### Usage

Log an error or warning message in the timeline record of the current task.

#### Properties

- `type` = `error` or `warning` (Required)
- `sourcepath` = source file location
- `linenumber` = line number
- `columnnumber` = column number
- `code` = error or warning code

#### Example: Log an error

- [Bash](#)
- [PowerShell](#)

```
#!/bin/bash
echo "##vso[task.logissue type=error]Something went very wrong."
exit 1
```

## TIP

`exit 1` is optional, but is often a command you'll issue soon after an error is logged. If you select **Control Options: Continue on error**, then the `exit 1` will result in a partially successful build instead of a failed build.

### Example: Log a warning about a specific place in a file

- [Bash](#)
- [PowerShell](#)

```
#!/bin/bash
echo "##vso[task.logissue
type=warning;sourcepath=consoleapp/main.cs;linenumber=1;columnnumber=1;code=100;]Found something that could
be a problem."
```

### SetProgress: Show percentage completed

```
##vso[task.setprogress]current operation
```

#### Usage

Set progress and current operation for the current task.

#### Properties

- `value` = percentage of completion

#### Example

- [Bash](#)
- [PowerShell](#)

```
echo "Begin a lengthy process..."
for i in {0..100..10}
do
 sleep 1
 echo "##vso[task.setprogress value=$i;]Sample Progress Indicator"
done
echo "Lengthy process is complete."
```

To see how it looks, save and queue the build, and then watch the build run. Observe that a progress indicator changes when the task runs this script.

### Complete: Finish timeline

```
##vso[task.complete]current operation
```

#### Usage

Finish the timeline record for the current task, set task result and current operation. When result not provided, set result to succeeded.

#### Properties

- `result` =
  - `Succeeded` The task succeeded.
  - `SucceededWithIssues` The task ran into problems. The build will be completed as partially succeeded at best.
  - `Failed` The build will be completed as failed. (If the **Control Options: Continue on error** option is selected, the build will be completed as partially succeeded at best.)

#### Example

```
##vso[task.complete result=Succeeded;]DONE
```

## LogDetail: Create or update a timeline record for a task

```
##vso[task.logdetail]current operation
```

### Usage

Creates and updates timeline records. This is primarily used internally by Azure Pipelines to report about steps, jobs, and stages. While customers can add entries to the timeline, they won't typically be shown in the UI.

The first time we see `##vso[task.detail]` during a step, we create a "detail timeline" record for the step. We can create and update nested timeline records base on `id` and `parentid`.

Task authors must remember which GUID they used for each timeline record. The logging system will keep track of the GUID for each timeline record, so any new GUID will result a new timeline record.

### Properties

- `id` = Timeline record GUID (Required)
- `parentid` = Parent timeline record GUID
- `type` = Record type (Required for first time, can't overwrite)
- `name` = Record name (Required for first time, can't overwrite)
- `order` = order of timeline record (Required for first time, can't overwrite)
- `starttime` = Datetime
- `finishtime` = Datetime
- `progress` = percentage of completion
- `state` = Unknown | Initialized | InProgress | Completed
- `result` = Succeeded | SucceededWithIssues | Failed

### Examples

Create new root timeline record:

```
##vso[task.logdetail id=new guid;name=project1;type=build;order=1]create new timeline record
```

Create new nested timeline record:

```
##vso[task.logdetail id=new guid;parentid=exist timeline record guid;name=project1;type=build;order=1]create new nested timeline record
```

Update exist timeline record:

```
##vso[task.logdetail id=existing timeline record guid;progress=15;state=InProgress;]update timeline record
```

## SetVariable: Initialize or modify the value of a variable

```
##vso[task.setvariable]value
```

### Usage

Sets a variable in the variable service of taskcontext. The first task can set a variable, and following tasks are able to use the variable. The variable is exposed to the following tasks as an environment variable.

When `issecret` is set to `true`, the value of the variable will be saved as secret and masked out from log. Secret variables are not passed into tasks as environment variables and must instead be passed as inputs.

See [set variables in scripts](#) for more details.

## Properties

- `variable` = variable name (Required)
- `issecret` = boolean (Optional, defaults to false)
- `isoutput` = boolean (Optional, defaults to false)
- `isreadonly` = boolean (Optional, defaults to false)

## Examples

- [Bash](#)
- [PowerShell](#)

Set the variables:

```
- bash: |
 echo "##vso[task.setvariable variable=sauce;]crushed tomatoes"
 echo "##vso[task.setvariable variable=secretSauce;issecret=true]crushed tomatoes with garlic"
 echo "##vso[task.setvariable variable=outputSauce;isoutput=true]canned goods"
name: SetVars
```

Read the variables:

```
- bash: |
 echo "Non-secrets automatically mapped in, sauce is $SAUCE"
 echo "Secrets are not automatically mapped in, secretSauce is ${SECRETSAUCE}"
 echo "You can use macro replacement to get secrets, and they'll be masked in the log: ${secretSauce}"
 echo "Future jobs can also see ${SETVARS_OUTPUTSAUCE}"
 echo "Future jobs can also see ${SetVars.outputSauce}"
```

Console output:

```
Non-secrets automatically mapped in, sauce is crushed tomatoes
Secrets are not automatically mapped in, secretSauce is
You can use macro replacement to get secrets, and they'll be masked in the log: ***
Future jobs can also see canned goods
Future jobs can also see canned goods
```

## SetEndpoint: Modify a service connection field

```
##vso[task.setendpoint]value
```

### Usage

Set a service connection field with given value. Value updated will be retained in the endpoint for the subsequent tasks that execute within the same job.

## Properties

- `id` = service connection ID (Required)
- `field` = field type, one of `authParameter`, `dataParameter`, or `url` (Required)
- `key` = key (Required, unless `field` = `url`)

## Examples

```
##vso[task.setendpoint id=000-0000-0000;field=authParameter;key=AccessToken]testvalue
##vso[task.setendpoint id=000-0000-0000;field=dataParameter;key=userVariable]testvalue
##vso[task.setendpoint id=000-0000-0000;field=url]https://example.com/service
```

## AddAttachment: Attach a file to the build

```
##vso[task.addattachment]value
```

## Usage

Upload and attach attachment to current timeline record. These files are not available for download with logs. These can only be referred to by extensions using the type or name values.

## Properties

- `type` = attachment type (Required)
- `name` = attachment name (Required)

## Example

```
##vso[task.addattachment type=myattachmenttype;name=myattachmentname;]c:\myattachment.txt
```

## UploadSummary: Add some Markdown content to the build summary

```
##vso[task.uploadsummary]local file path
```

## Usage

Upload and attach summary markdown to current timeline record. This summary shall be added to the build/release summary and not available for download with logs. The summary should be in UTF-8 or ASCII format.

## Examples

```
##vso[task.uploadsummary]c:\testsummary.md
```

It is a short hand form for the command

```
##vso[task.addattachment type=Distributedtask.Core.Summary;name=testsummaryname;]c:\testsummary.md
```

## UploadFile: Upload a file that can be downloaded with task logs

```
##vso[task.uploadfile]local file path
```

## Usage

Upload user interested file as additional log information to the current timeline record. The file shall be available for download along with task logs.

## Example

```
##vso[task.uploadfile]c:\additionalfile.log
```

## PrependPath: Prepend a path to the PATH environment variable

```
##vso[task.prependpath]local file path
```

## Usage

Update the PATH environment variable by prepending to the PATH. The updated environment variable will be reflected in subsequent tasks.

## Example

```
##vso[task.prependpath]c:\my\directory\path
```

# Artifact commands

## Associate: Initialize an artifact

```
##vso[artifact.associate]artifact location
```

## Usage

Create a link to an existing Artifact. Artifact location must be a file container path, VC path or UNC share path.

## Properties

- `artifactname` = artifact name (Required)
- `type` = artifact type (Required) `container` | `filepath` | `versioncontrol` | `gitref` | `tfvclabel`

## Examples

- `container`

```
##vso[artifact.associate type=container;artifactname=MyServerDrop]#/1/build
```

- `filepath`

```
##vso[artifact.associate type=filepath;artifactname=MyFileShareDrop]\MyShare\MyDropLocation
```

- `versioncontrol`

```
##vso[artifact.associate type=versioncontrol;artifactname=MyTfvcPath]$/MyTeamProj/MyFolder
```

- `gitref`

```
##vso[artifact.associate type=gitref;artifactname=MyTag]refs/tags/MyGitTag
```

- `tfvclabel`

```
##vso[artifact.associate type=tfvclabel;artifactname=MyTag]MyTfvcLabel
```

- `Custom Artifact`

```
##vso[artifact.associate
artifactname=myDrop;artifacttype=myartifacttype]https://downloads.visualstudio.com/foo/bar/package.zip
```

## Upload: Upload an artifact

```
##vso[artifact.upload]local file path
```

## Usage

Upload a local file into a file container folder, and optionally publish an artifact as `artifactname`.

## Properties

- `containerfolder` = folder that the file will upload to, folder will be created if needed.
- `artifactname` = artifact name. (Required)

## Example

```
##vso[artifact.upload containerfolder=testresult;artifactname=uploadedresult]c:\testresult.trx
```

#### **NOTE**

The difference between `Artifact.associate` and `Artifact.upload` is that the first can be used to create a link to an existing artifact, while the latter can be used to upload/publish a new Artifact.

## Build commands

### **UploadLog: Upload a log**

```
##vso[build.uploadlog]local file path
```

#### **Usage**

Upload user interested log to build's container "`logs\tool`" folder.

#### **Example**

```
##vso[build.uploadlog]c:\msbuild.log
```

### **UpdateBuildNumber: Override the automatically generated build number**

```
##vso[build.updatebuildnumber]build number
```

#### **Usage**

You can automatically generate a build number from tokens you specify in the [pipeline options](#). However, if you want to use your own logic to set the build number, then you can use this logging command.

#### **Example**

```
##vso[build.updatebuildnumber]my-new-build-number
```

### **AddBuildTag: Add a tag to the build**

```
##vso[build.addbuilddtag]build tag
```

#### **Usage**

Add a tag for current build.

#### **Example**

```
##vso[build.addbuilddtag]Tag_UnitTestPassed
```

## Release commands

### **UpdateReleaseName: Rename current release**

```
##vso[release.updatereleasename]release name
```

#### **Usage**

Update the release name for the running release.

#### **NOTE**

Supported in Azure DevOps and Azure DevOps Server beginning in version 2020.

#### **Example**

```
##vso[release.updatereleasename]my-new-release-name
```

# Artifact policy checks

6/12/2020 • 2 minutes to read • [Edit Online](#)

Artifact policies are enforced before deploying to critical environments such as production. These policies are evaluated against all the deployable artifacts in the given pipeline run and block the deployment if the artifacts don't comply. Adding a check to evaluate Artifact requires the custom policy to be configured. This guide describes how custom policies can be created.

## NOTE

Currently, the supported artifact types are for container images and Kubernetes environments

## Prerequisites

Use Rego for defining policy that is easy to read and write.

Familiarize yourself with [Rego](#) query language. Basics will do.

To support structured document models like JSON, Rego extends Datalog. Rego queries are assertions on data stored in OPA. These queries can be used to define policies that enumerate instances of data that violate the expected state of the system.

## Creating custom policies

Below are the sample policies shared. Based on your requirements, you can build your own set of policies.

### Check specific project/pipeline

This policy checks if the images are built by Azure Pipelines and Pipeline-foo. For this to work, the pipeline definition should override the name field to something like:

`AzureDevOps_${(BuildDefinitionName)}_${(Date:yyyyMMdd)}${(Rev:.r)}`. See more about naming pipeline runs [here](#).

```
allowedBuilder := "AzureDevOps_pipeline-foo"

checkBuilder[errors] {
 trace("Check if images are built by Azure Pipelines")
 resourceUri := values[index].build.resourceUri
 image := fetchImage(resourceUri)
 builder := values[index].build.build.provenance.builderVersion
 trace(sprintf("%s: builder", [builder]))
 not startswith(builder, "allowedBuilder")
 errors := sprintf("%s: image not built by Azure Pipeline [%s]", [image,builder])
}

fetchRegistry(uri) = reg {
 out := regex.find_n("//.*/", uri, 1)
 reg = trim(out[0], "/")
}

fetchImage(uri) = img {
 out := regex.find_n("./.*@", uri, 1)
 img := trim(out[0], "/@")
}
```

## Check allowed registries

This policy checks if the images are from allowed registries only.

```
allowlist = {
 "gcr.io/myrepo",
 "raireg1.azurecr.io"
}

checkregistries[errors] {
 trace(sprintf("Allowed registries: %s", [concat(", ", allowlist)]))
 resourceUri := values[index].image.resourceUri
 registry := fetchRegistry(resourceUri)
 image := fetchImage(resourceUri)
 not allowlist[registry]
 errors := sprintf("%s: source registry not permitted", [image])
}

fetchRegistry(uri) = reg {
 out := regex.find_n("//.*/", uri, 1)
 reg = trim(out[0], "/")
}

fetchImage(uri) = img {
 out := regex.find_n("/.*@", uri, 1)
 img := trim(out[0], "/@")
}
```

## Check forbidden ports

This policy checks for any forbidden ports exposed in the container image.

```
forbiddenPorts = {
 "80",
 "22"
}

checkExposedPorts[errors] {
 trace(sprintf("Checking for forbidden exposed ports: %s", [concat(", ", forbiddenPorts)]))
 layerInfos := values[index].image.image.layerInfo
 layerInfos[x].directive == "EXPOSE"
 resourceUri := values[index].image.resourceUri
 image := fetchImage(resourceUri)
 ports := layerInfos[x].arguments
 trace(sprintf("exposed ports: %s", [ports]))
 forbiddenPorts[ports]
 errors := sprintf("%s: image exposes forbidden port %s", [image, ports])
}

fetchRegistry(uri) = reg {
 out := regex.find_n("//.*/", uri, 1)
 reg = trim(out[0], "/")
}

fetchImage(uri) = img {
 out := regex.find_n("/.*@", uri, 1)
 img := trim(out[0], "/@")
}
```

## Check prior deployments

This policy checks if the image has been pre-deployed to one/more of the environments before being deployed to specific environment/resources with Check configured.

```

predeployedEnvironments = {
 "env/resource1",
 "env2/resource3"
}

checkDeployedEnvironments[errors] {
 trace(sprintf("Checking if the image has been pre-deployed to one of: [%s]", [concat(", ",
predeployedEnvironments)]))
 deployments := values[index].deployment
 deployedAddress := deployments[i].deployment.address
 trace(sprintf("deployed to : %s",[deployedAddress]))
 resourceUri := deployments[i].resourceUri
 image := fetchImage(resourceUri)
 not predeployedEnvironments[deployedAddress]
 trace(sprintf("%s: fails pre-deployed environment condition. found %s", [image,deployedAddress]))
 errors := sprintf("image %s fails pre-deployed environment condition. found %s",
[image,deployedAddress])
}

fetchRegistry(uri) = reg {
 out := regex.find_n("//.*/", uri, 1)
 reg = trim(out[0], "/")
}

fetchImage(uri) = img {
 out := regex.find_n("./.*@", uri, 1)
 img := trim(out[0], "/@")
}

```

# Learn how to add continuous security validation to your CI/CD pipeline

3/6/2021 • 9 minutes to read • [Edit Online](#)

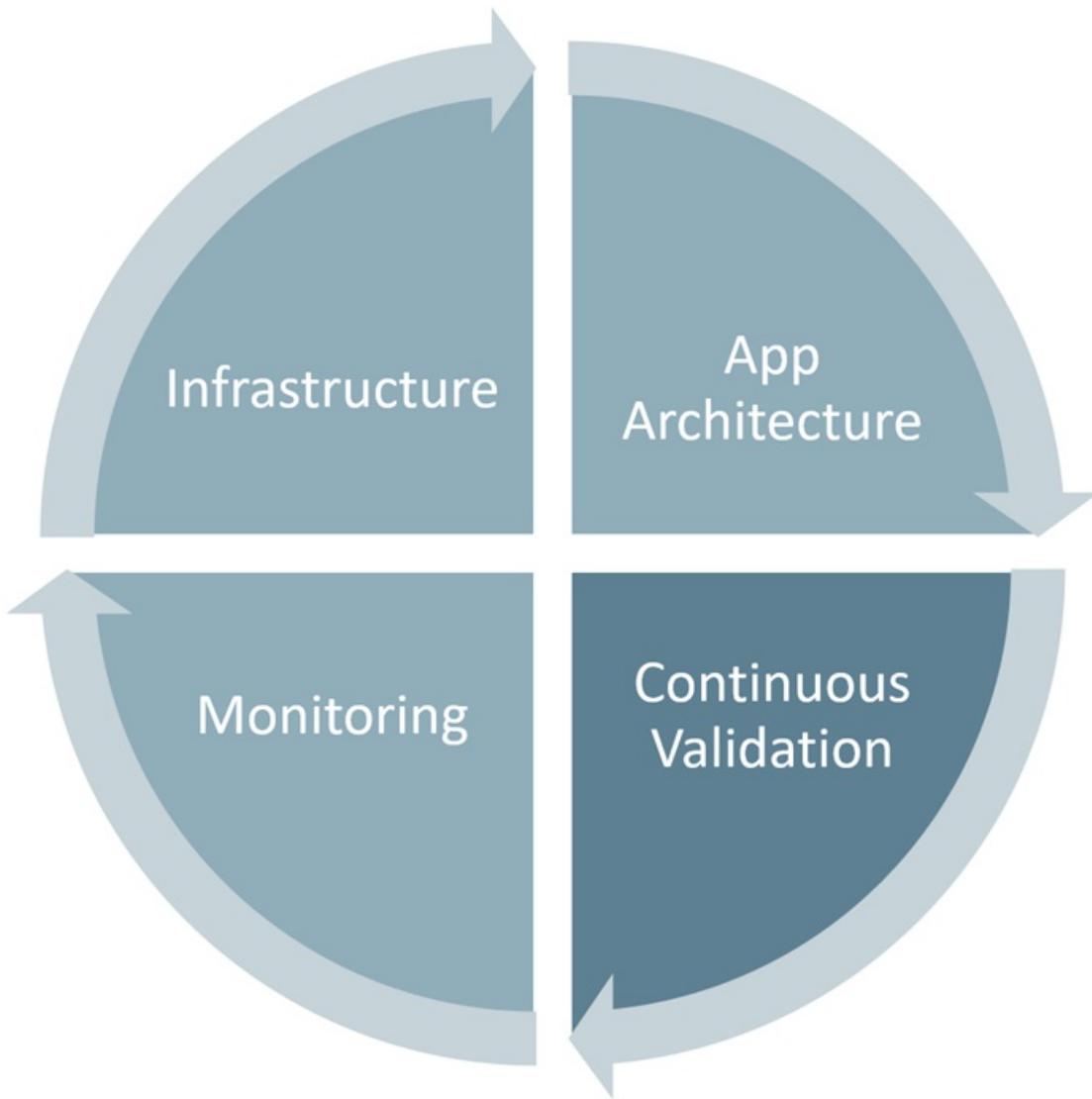
Azure DevOps Services | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2015

Are you planning Azure DevOps continuous integration and deployment pipelines? You probably have a few questions, such as:

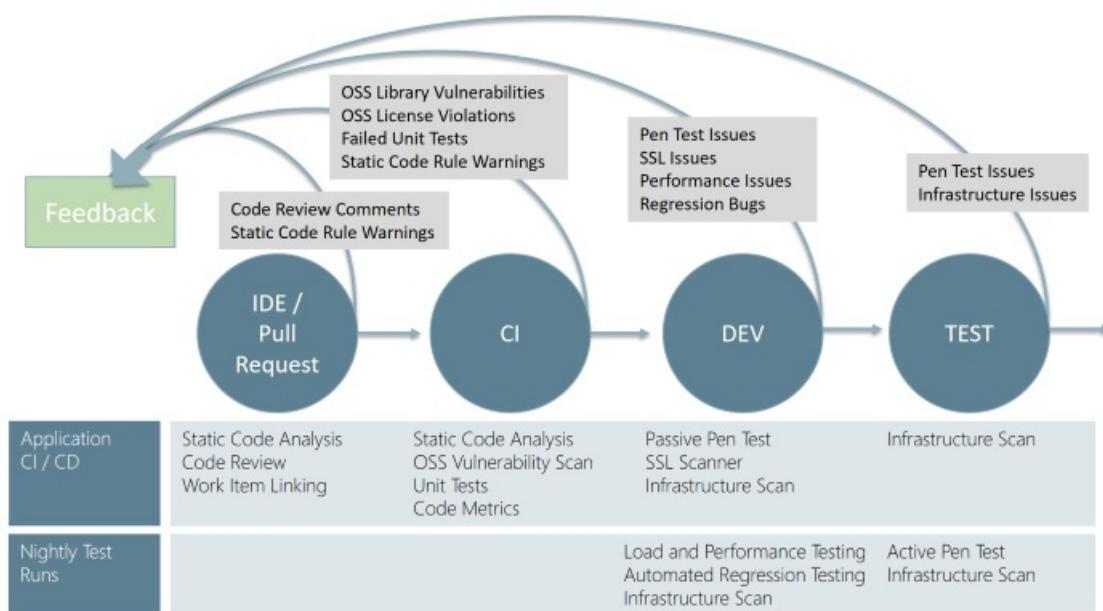
- How do you ensure your application is safe?
- How do you add continuous security validation to your CI/CD pipeline?

DevOps practices are allowing businesses to stay ahead of the competition by delivering new features faster than ever before. As the frequency of production deployments increases, this business agility cannot come at the expense of security. With continuous delivery, how do you ensure your applications are secure and stay secure? How can you find and fix security issues early in the process? This begins with practices commonly referred to as DevSecOps. DevSecOps incorporates the security team and their capabilities into your DevOps practices making security a responsibility of everyone on the team. This article will walk you through how to help ensure your application is secure by adding continuous security validation to your CI/CD pipeline.

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



Continuous security validation should be added at each step from development through production to help ensure the application is always secure. The goal of this approach is to switch the conversation with the security team from approving each release to approving the CI/CD process and having the ability to monitor and audit the process at any time. When building greenfield applications, the diagram below highlights the key validation points in the CI/CD pipeline. Depending on your platform and where your application is at in its lifecycle, you may need to consider implementing the tools gradually. Especially if your product is mature and you haven't previously run any security validation against your site or application.



## IDE / Pull Request

Validation in the CI/CD begins before the developer commits his or her code. Static code analysis tools in the IDE provide the first line of defense to help ensure that security vulnerabilities are not introduced into the CI/CD process. The process for committing code into a central repository should have controls to help prevent security vulnerabilities from being introduced. Using Git source control in Azure DevOps with branch policies provides a gated commit experience that can provide this validation. By enabling [branch policies](#) on the shared branch, a pull request is required to initiate the merge process and ensure that all defined controls are being executed. The pull request should require a code review, which is the one manual but important check for identifying new issues being introduced into your code. Along with this manual check, commits should be linked to work items for auditing why the code change was made and require a continuous integration (CI) build process to succeed before the push can be completed.

## CI (Continuous Integration)

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

- [Visual Studio Code Analysis and the Roslyn Security Analyzers](#)
- [Checkmarx](#) - A Static Application Security Testing (SAST) tool
- [BinSkim](#) - A binary static analysis tool that provides security and correctness results for Windows portable executables
- [Other 3rd party tools](#)

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

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

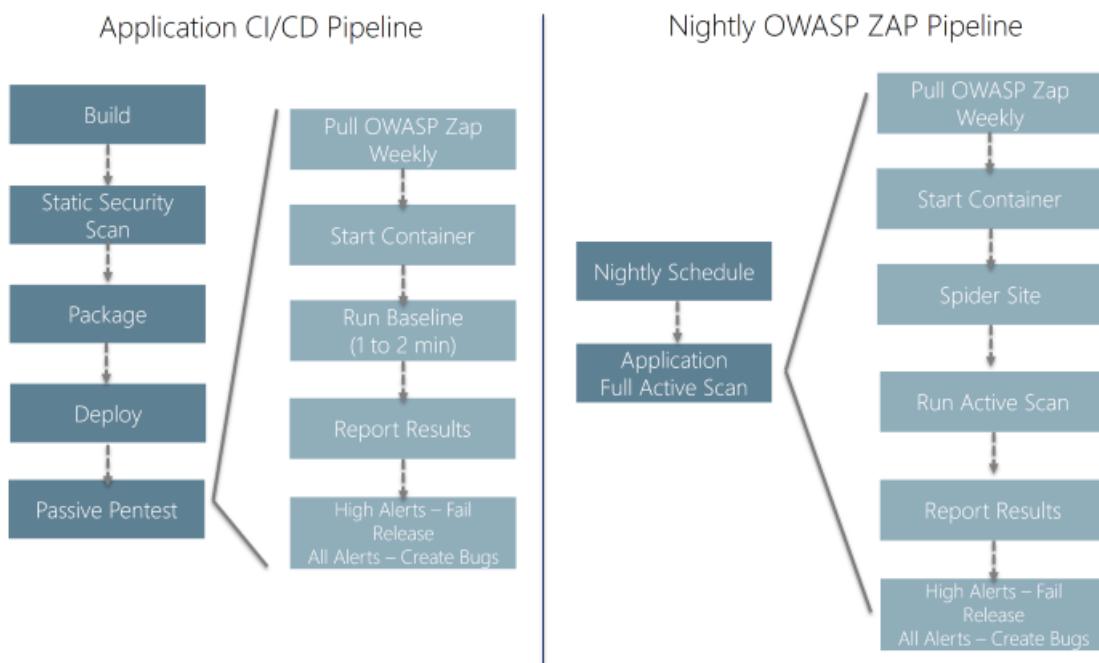
## Application Deployment to DEV and TEST

Once your code quality is verified, and the application is deployed to a lower environment like development or QA, the process should verify that there are not any security vulnerabilities in the running application. This can be accomplished by executing automated penetration test against the running application to scan it for vulnerabilities. There are different levels of tests that are categorized into passive tests and active tests. Passive tests scan the target site as is but don't try to manipulate the requests to expose additional vulnerabilities. These can run fast and are usually a good candidate for a CI process that you want to complete in a few minutes. Whereas the Active Scan can be used to simulate many techniques that hackers commonly use to attack

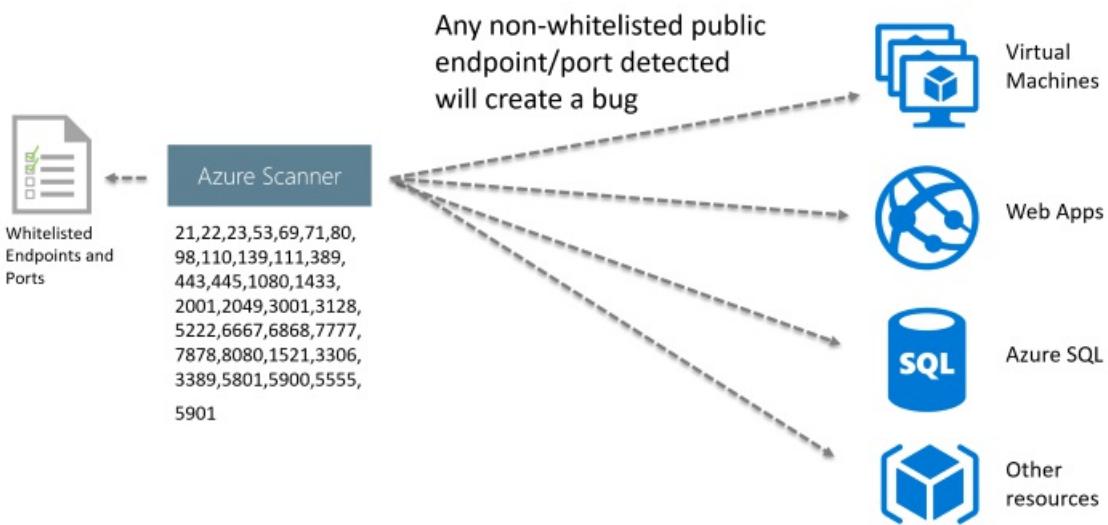
websites. These tests can also be referred to dynamic or fuzz tests because the tests are often trying a large number of different combinations to see how the site reacts to verify that it doesn't reveal any information. These tests can run for much longer, and typically you don't want to cut these off at any particular time. These are better executed nightly as part of a separate Azure DevOps release.

One tool to consider for penetration testing is OWASP ZAP. OWASP is a worldwide not-for-profit organization dedicated to helping improve the quality of software. ZAP is a free penetration testing tool for beginners to professionals. ZAP includes an API and a weekly docker container image that can be integrated into your deployment process. The detailed how-to steps are outside the scope of this article. Refer to the [OWASP ZAP VSTS extension](#) repo for details on how to set up the integration. Here we're going to explain the benefits of including this into your process.

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



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



Once the scans have completed, the Azure Pipelines release is updated with a report that includes the results and bugs are created in the team's backlog. Resolved bugs will close if the vulnerability has been fixed and move back into in-progress if the vulnerability still exists.

The benefit of using this is that the vulnerabilities are created as bugs that provide actionable work that can be tracked and measured. False positives can be suppressed using OWASP ZAP's context file, so only vulnerabilities that are true vulnerabilities are surfaced.

OWASP ZAP Nightly / Release-40

Backlog Board Capacity

New 61.5 h

Active 11 h

Unparented

207810 Incomplete or No Cache-control and Pragma HTTP Header Set  
State: New

207811 Cookie No HttpOnly Flag  
State: New

207812 Cookie Without Secure Flag  
State: New

207813 Web Browser XSS Protection Not Enabled  
State: New

207814 X-Content-Type-Options Header Missing  
State: New

207815 X-Frame-Options Header Not Set  
State: New

Total tests: 6 Pass percentage: 0% Run duration: 0s

Test

- 0/6 Passed - OWASP ZAP Security Tests
- Incomplete or No Cache-control and Pragma HTTP Header Set
- Cookie No HttpOnly Flag
- Cookie Without Secure Flag
- Web Browser XSS Protection Not Enabled
- X-Content-Type-Options Header Missing
- X-Frame-Options Header Not Set

Even with continuous security validation running against every change to help ensure new vulnerabilities are not introduced, hackers are continuously changing their approaches, and new vulnerabilities are being discovered. Good monitoring tools allow you to help detect, prevent, and remediate issues discovered while your application is running in production. Azure provides a number of tools that provide detection, prevention, and alerting using rules such as OWASP Top 10 / modSecurity and now even using machine learning to detect anomalies and unusual behavior to help identify attackers.

Minimize security vulnerabilities by taking a holistic and layered approach to security including secure infrastructure, application architecture, continuous validation, and monitoring. DevSecOps practices enable your entire team to incorporate these security capabilities throughout the entire lifecycle of your application. Establishing continuous security validation into your CI/CD pipeline can allow your application to stay secure while you are improving the deployment frequency to meet needs of your business to stay ahead of the competition.

## Reference information

- [BinSkim](#) - A binary static analysis tool that provides security and correctness results for Windows portable executables
- [Checkmarx](#) - A Static Application Security Testing (SAST) tool
- [Manage your open source usage and security as reported by your CI/CD pipeline](#)
- [OWASP](#)
- [OWASP ZAP VSTS extension](#)
- [WhiteSource Software](#)
- [Visual Studio Code Analysis and the Roslyn Security Analyzers](#)

Authors: Mike Douglas | Find the origin of this article and connect with the ALM | DevOps Rangers [here](#)

*(c) 2017 Microsoft Corporation. All rights reserved. This document is provided "as-is." Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.*

*This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.*

# Explore how to progressively expose your Azure DevOps extension releases in production to validate, before impacting all users

5/21/2021 • 7 minutes to read • [Edit Online](#)

Azure DevOps Services | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2013

In today's fast-paced, feature-driven markets, it's important to continuously deliver value and receive feedback on features quickly and continuously. Partnering with end users to get early versions of features vetted out is valuable.

Are you planning to build and deploy Azure DevOps extensions to production? You probably have a few questions, such as:

- How do you embrace DevOps to deliver changes and value faster?
- How do you mitigate the risk of deploying to production?
- How do you automate the build and deployment?

This topic aims to answer these questions and share learnings using rings with Azure DevOps extensions. For an insight into the Microsoft guidelines, read [Configuring your release pipelines for safe deployments](#).

## One or more rings to rule your deployments

Deployment rings were first discussed in [Jez Humble's book](#). They support the production-first DevOps mindset and limit impact on end users, while gradually deploying and validating changes in production. Impact (also called **blast radius**), is evaluated through observation, testing, analysis of telemetry, and user feedback.

## Considerations

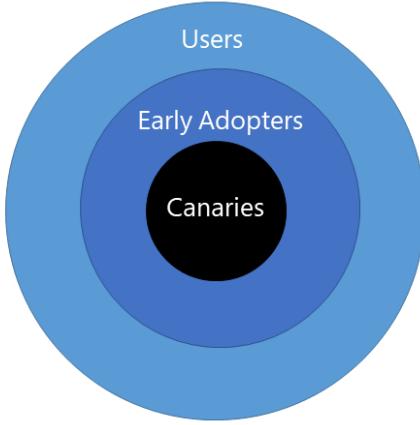
Before you convert your deployment infrastructure to a ringed deployment model, it's important to consider:

- Who are your primary types of users? For example, early adopters and users.
- What's your application topology?
- What's the value of embracing ringed deployment model?
- What's the cost to convert your current infrastructure to a ringed deployment model?

## User types

In the shown example, users fall into three general buckets in production:

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



**NOTE**

It's important to weigh out which users in your value chain are best suited for each of these buckets. Communicating the opportunity to provide feedback, as well as the risk levels at each tier, is critical to setting expectations and ensuring success.

## Application topology

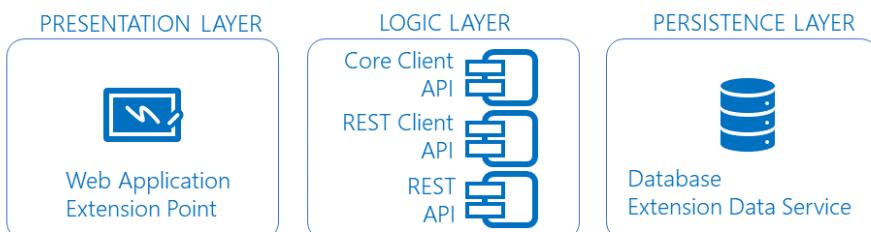
Next you need to map the topology of your application to the ringed deployment model. Limit the impact of change on end users and to continuously deliver value. Value includes both the value delivered to the end user and the value (return-on-investment) of converting your existing infrastructure.

**NOTE**

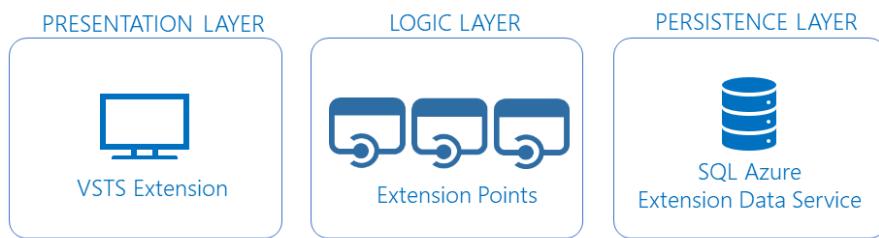
The ringed deployment model is not a silver bullet! Start small, prototype, and continuously compare impact, value, and cost.

At the application level, the composition of Azure DevOps extensions is innocuous, easy to digest, scale, and deploy independently. Each extension:

- Has one or more web and script files
- Interfaces with Core client
- Interfaces with REST client and REST APIs
- Persists state in cache or resilient storage



At the infrastructure level, the extensions are published to the [Visual Studio marketplace](#). Once installed in organization, they are hosted by the Azure DevOps service portal, with state persisted to Azure storage and/or the extension [data storage](#).



The extension topology is perfectly suited for the ring deployment model and to publish the extension to each deployment ring:

- A **private** development version for the canary ring
- A **private** preview version for the early adopter ring
- A **public** production version for the Users ring

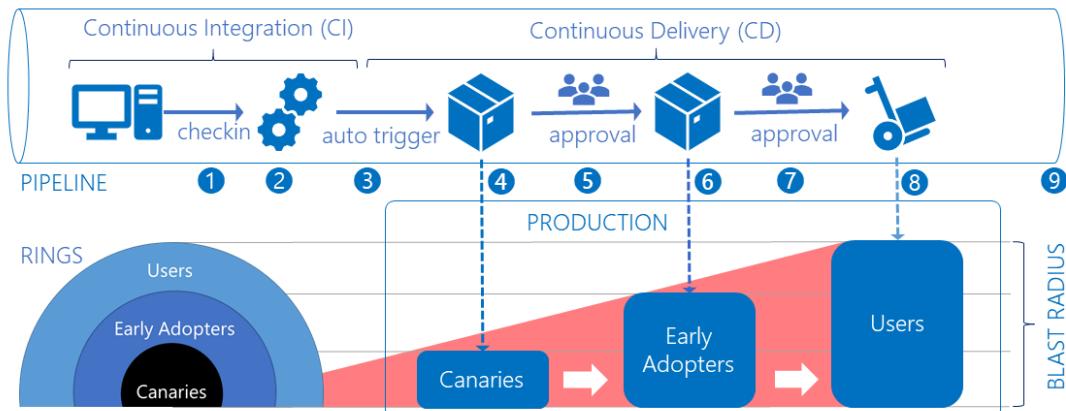
**TIP**

By publishing your extension as private, you're effectively limiting and controlling their exposure for users you explicitly invite.

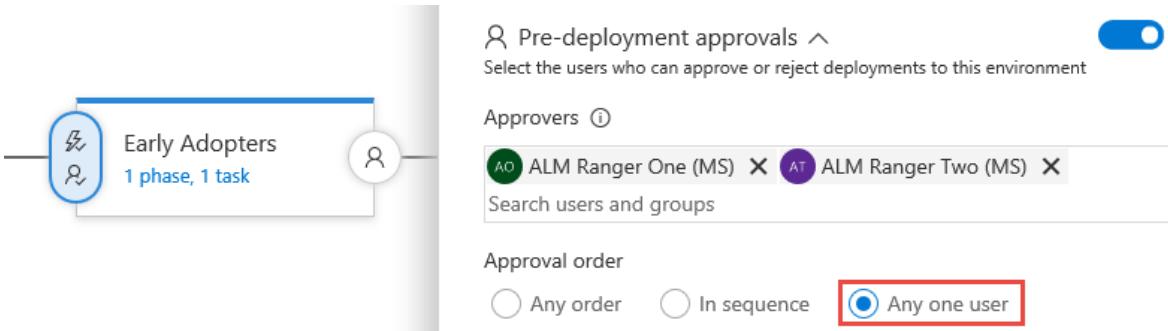
## Moving changes through deployment rings

Let's observe how a change triggers and moves through the ring-based deployment process, using the [Azure DevOps Developer Tools Build Tasks](#) extension.

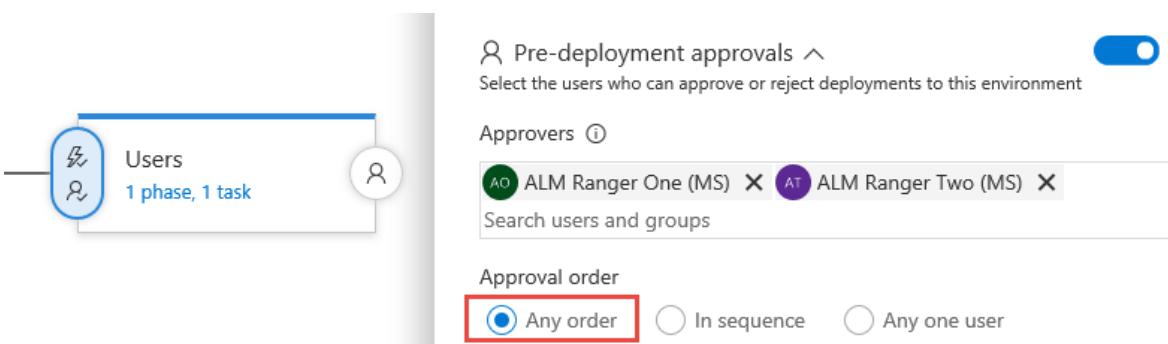
**Azure DevOps Developer Tools Build Tasks** extension is the secret sauce, used to package and publish Azure DevOps extensions to the Visual Studio Marketplace.



1. A developer from the [Countdown Widget](#) extension project commits a change to the [GitHub](#) repository.
2. The commit triggers a continuous integration build.
3. The new build triggers a continuous deployment trigger, which automatically starts the **Canaries** environment deployment.
4. The **Canaries** deployment publishes a private extension to the marketplace and shares it with predefined organizations. Only the **Canaries** are impacted by the change.
5. The **Canaries** deployment triggers the **Early Adopter** environment deployment. A pre-deployment approval gate requires any one of the authorized users to approve the release.



6. The **Early Adopter** deployment publishes a private extension to the marketplace and shares it with predefined organizations. Both the **Canaries** and **Early Adopter** are impacted by the change.
7. The **Early Adopter** deployment triggers the **Users** environment deployment. A stricter pre-deployment approval gate requires all of the authorized users to approve the release.



8. The **Users** deployment publishes a public extension to the marketplace. At this stage, everyone who has installed the extension in their organization is affected by the change.
9. It's key to realize that the impact ("blast radius") increases as your change moves through the rings. Exposing the change to the **Canaries** and the **Early Adopters**, is giving two opportunities to validate the change and hotfix critical bugs before a release to production.

#### NOTE

Review [CI/CD Pipelines](#) and [Approvals](#) for detailed documentation of pipelines and the approval features for releases.

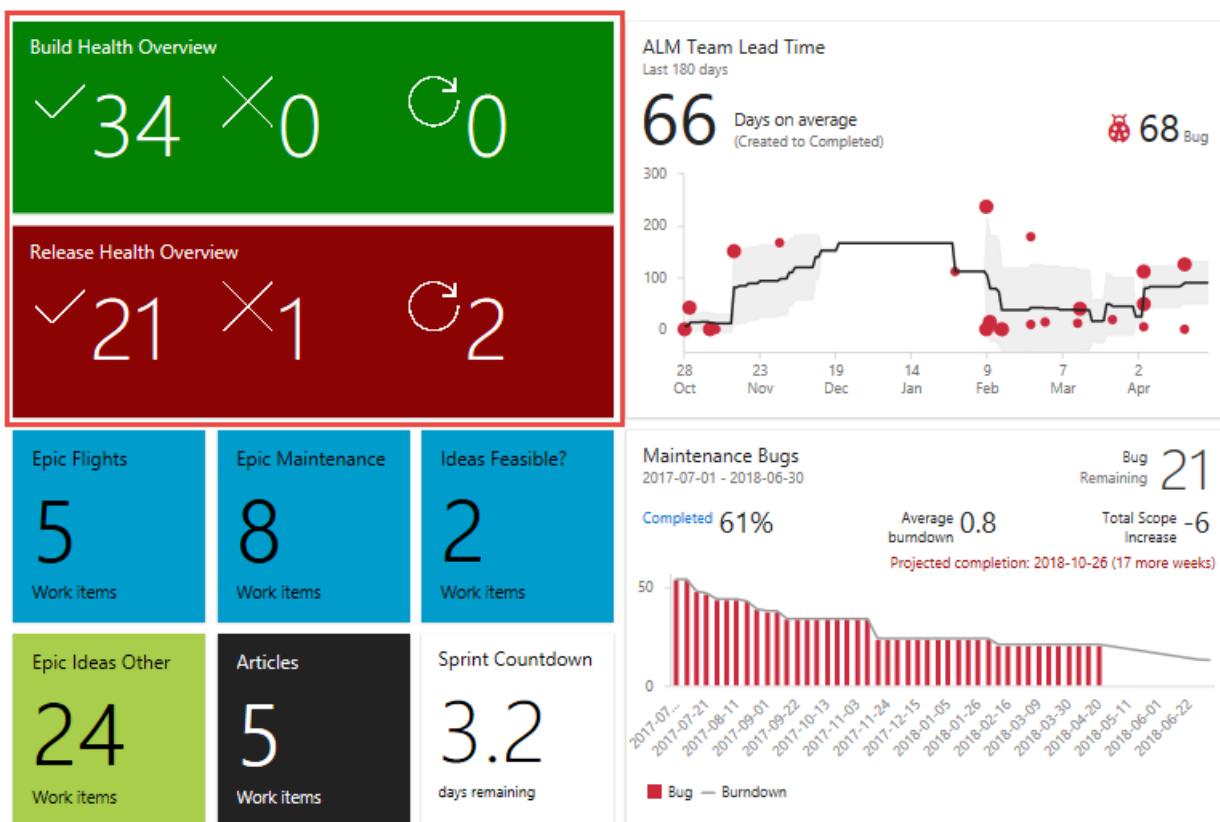
## Dealing with monitoring and noise

You need **effective** monitoring and **actionable** alerts to detect and mitigate issues. Determine what type of data is important, for example infrastructure issues, violations, and feature usage. Focus on actionable alerts to avoid users ignoring them and missing high priority issues.

#### TIP

Start with high-level views of your data, visual dashboards that you can watch from afar, and drill-down as needed. Perform regular housekeeping of your views and remove all noise. A visual dashboard tells a far better story than hundreds of notification emails, often filtered and forgotten by email rules.

Using the [Team Project Health](#) and out-of-the-box extensions you can build overview of your pipelines, lead and cycle times, and other information. In the sample dashboard, it's evident that there are 34 successful builds, 21 successful releases, 1 failed release, and 2 releases in progress.



## What's the value?

Using a ring-deployment strategy you can gather feedback to validate your hypothesis. You can decommission old releases and distribute new releases without the risk of affecting all users.

Here's a summary of how the ALM | DevOps Ranger engineering process evolved with ring deployment models.

BEFORE USING RINGS	IMPACTED AREA	WITH RINGS
Manual and error prone	Build	Automated and consistent
Manual and error prone	Release	Automated and consistent
Hours	Time to build (TTB)	Seconds
Days	Time to release (TTR)	Minutes
Call from user	Issue detection	Proactive
Days to weeks	Issue resolution	Minutes to days

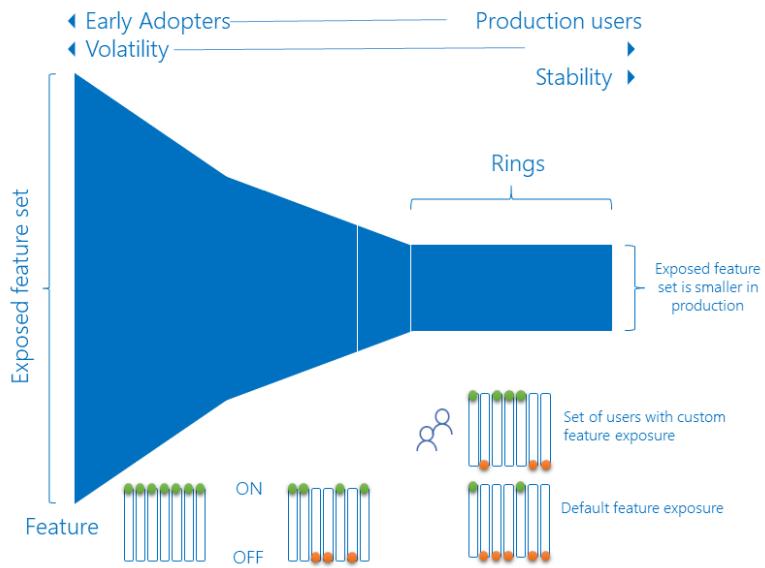
Key takeaways:

- Consistent and reliable automation
- Reduced response times
- Canaries experience the pain, not the users

## Is there a dependency on feature flags?

No, rings and feature flags are symbiotic. Feature flags give you fine-grained control of features included in your

change. For example, if you're not fully confident about a feature you can use feature flags to **hide** the feature in one or all of the deployment rings. For example, you could enable all features in the canaries ring, and fine-tune a subset for the early adopters and production users, as shown. See [Feature Flags or Rings](#) for more information.



[LaunchDarkly](#) provides an extension for Azure DevOps Services & Team Foundation Server. It integrates with Azure Pipelines and gives you "run-time" control of features deployed with your ring deployment process.

## Conclusion

Now that you've covered the concepts of rings, you should be confident to explore ways to improve your CI/CD pipelines. While the use of rings adds a level of complexity, having a game plan to address feature management and rapid customer feedback is invaluable.

## Q & A

### How do you know that a change can be deployed to the next ring?

Your goal should be to have a consistent checklist for the users approving a release. See [aka.ms/vsarDoD](https://aka.ms/vsarDoD) for an example definition of done checklist.

### How long do you wait before you push a change to the next ring?

There is no fixed duration or "cool off" period. It depends on how long it takes for you to complete all release validations successfully.

### How do you manage a hotfix?

The ring deployment model allows you to process a hotfix like any other change. The sooner an issue is caught, the sooner a hotfix can be deployed, with no impact to downstream rings.

### How do you deal with variables that span (shared) release environments?

Refer to [Default and custom release variables](#).

### How can you manage secrets used by the pipeline?

Refer to [Azure Key Vault](#) to safeguard cryptographic keys and other secrets used by your pipelines.

## Reference information

- [CI/CD pipeline examples](#)
- [Configuring your release pipelines for safe deployments](#)

- DevOps @ Microsoft

**NOTE**

Authors: Josh Garverick, Willy Schaub Find the origin of this article and connect with the ALM DevOps Rangers [here](#)

*(c) 2017 Microsoft Corporation. All rights reserved. This document is provided "as-is." Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.*

*This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.*

# Explore how to progressively expose your features in production for some or all users

5/27/2021 • 6 minutes to read • [Edit Online](#)

Azure DevOps Services | Azure DevOps Server 2020 | Azure DevOps Server 2019 | TFS 2018 - TFS 2013

In today's fast-paced, feature-driven markets, it's important to continuously deliver value and receive feedback on features quickly and continuously. Partnering with end users to get early versions of features vetted out is valuable.

Are you planning to continuously integrate features into your application while they're under development? You probably have a few questions, such as:

- How can you toggle features to hide, disable, or enable features at run-time?
- How can you revert a change deployed to production without rolling back your release?
- How can you present users with variants of a feature, to determine which one performs better?

This topic aims to answer these questions and share an implementation of feature flags (FF) and A|B testing used with Azure DevOps extensions.

## Considerations

Before you introduce feature flags to your engineering process, it's important to consider:

- Which users are you planning to target? For example, do you want to target specific or all users?
- Would you like users to decide which features they want to use?
- What's the value of embracing feature flags as part of your engineering process?
- What's the cost to implement feature flags in your engineering process?

Before you flip your first feature flag in production, take the time to read:

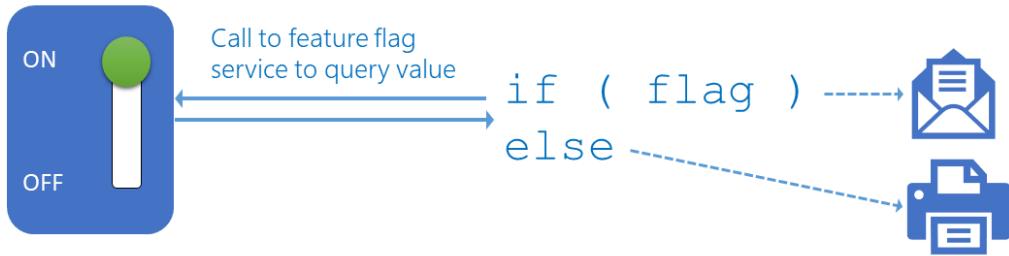
- "[A Rough Patch](#)", by Brian Harry
- "[Feature Flags with Branching](#)", by LaunchDarkly

## What are Feature Flags (FF)?

### NOTE

A feature flag is also known as a feature toggle, feature switch, feature flipper, or conditional feature. They were popularized by [Martin Fowler](#).

Feature flags support a customer-first DevOps mindset, to enable (expose) and disable (hide) features in a solution, even before they are complete and ready for release.



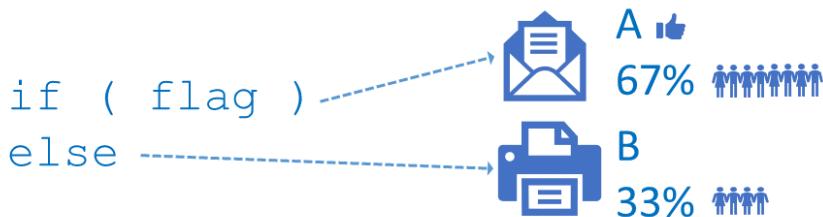
View a feature flag as an ON | OFF switch for a specific feature. As shown, you can deploy a solution to production that includes both an email and a print feature. If the feature flag is set (ON), you'll email, else you'll print.

When you combine a feature flag with an experiment, led by a hypothesis, you introduce A|B testing. For example, you could run an experiment to determine if the email (A) or the print (B) feature will result in a higher user satisfaction.

#### NOTE

A|B testing is also known as Split Testing. It's based on a hypothesis that's defined as:

**For {user} who {action} the {solution} is a {how} that {value} unlike {competition} we {do better}**



As shown, the email feature (option A) is more popular with your users and wins.

## Evaluating Feature Flag solutions

As outlined in [how to implement feature flags and A|B testing](#), the ALM | DevOps Rangers evaluated a number of FF frameworks and solutions.

They chose the [LaunchDarkly](#) solution for several reasons:

- It's a "software as a service" (SaaS) solution
  - No custom solution to maintain
  - No upgrades - you're always using the latest and greatest
  - No servers - [LaunchDarkly](#) takes care of the machines that LaunchDarkly runs on
  - Always on and optimized for the Internet
- It's integrated with Azure DevOps Services and Team Foundation Server (TFS)
- It's simple and cost-effective for an open-source project

## Common scenarios

You have a [CI/CD pipeline](#) for every Azure DevOps extension you're hosting on the [marketplace](#). You are using a ring deployment model and manual release approval checkpoints. The checkpoints are manual and time consuming, but necessary to minimize the chance of breaking the early-adopter and production user environments, forcing an expensive roll-back. You're looking for an engineering process, which enables you to:

- Continuously deploy to production

- Never roll back in production
- Fine-tune the user experience in production

You have probably guessed it - feature flags!

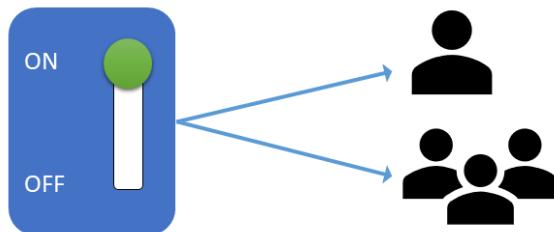
### Enable or disable a feature for everyone

You would like to include hidden features in your release and enable them for **all** users in production. For example, you want to be able to collect verbose logging data for troubleshooting. Using a feature flag, you can enable and disable verbose logging as needed.



### Enable or disable a feature for selected users

With this scenario, you can target specific users or groups of users. For example, you could enable the verbose logging feature for a specific user experiencing a problem or enable a preview feature for early adopters.

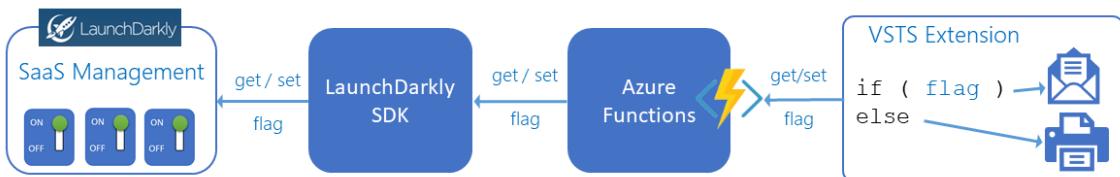


### Enable | disable a feature as selected by user

Lastly, you'd like to give the users a list of preview features and allow each user to decide which feature to enable when. This scenario is key for feature validation, A|B testing, and giving the user flexibility and choice.

## Manage features with feature flags in your engineering process

To protect the flags from malicious users, you need to generate and pass the hash of the user key to the LaunchDarkly API calls. As Azure DevOps extensions can only use client-side code, the ALM | DevOps Rangers chose Azure Functions to help generate the hash, as shown. Read [how we checked and fixed the 503 error and Performance issue in our Azure Function](#) for details.



Administration of feature flags is straight-forward.

1. You have a different environment for each extension, allowing you to have different feature flag values for Early Adopters and Users.
2. Optionally target specific users
3. Optionally target users that match custom rules
4. You have a default for each feature flag

You have granular control of each feature flag.

The screenshot shows the LaunchDarkly interface for managing feature flags. A sidebar on the left lists various sections: Roll-up Board, Users, Feature flags (which is selected), Goals, Dev console, Audit log, Quickstart, Documentation, Integrations, and Help. The main area is titled 'Enable telemetry' for the feature flag 'enable-telemetry'. It shows the flag is 'ON'. Under 'Targeting', there are four numbered steps: 1. 'Prerequisites' (empty box, '+ Add prerequisites') 2. 'Target individual users' (empty box, '+ Add user targets') 3. 'Target users who match these rules' (empty box, '+ Add rules') 4. 'Default rule' (dropdown set to 'SERVE false'). Below these are two notes: 'If targeting is off, serve false X' and 'If LaunchDarkly is unreachable or if no off variation is set, serve false ⓘ'.

## What's the value?

You're able to:

- Decouple deployment of releases and exposure of features
- Make changes (enable|disable features) without redeployment
- Fine-tune a user's features and experience
- Enable a user to optionally select preview features
- Hide an incomplete or faulty feature

## What's the cost?

Aside from the licensing and maintenance cost of a feature flag service, you're adding technical debt to your code:

- With a true or false feature flag, you're doubling your code and test paths
- With multi-value feature flag, you'll add even more code and test paths
- You'll need to identify and remove stale feature flags
- Understand and test the implications of flipping a feature flag

### TIP

To minimize the costs associated with the use of feature flags, keep feature flags short lived and prevent multiple feature flags from interfering with each other by affecting the same functionality.

# Conclusion

Now that you've covered the concepts and considerations of feature flags, you should be confident to explore ways to improve your CI/CD pipelines. While feature flags come at a cost, having a game plan to manage exposed features at run-time is invaluable.

## Q & A

### **How does the Azure DevOps team use feature flags?**

Buck's [feature flags blog post](#) and the [presentation/article](#) are great sources to get an understanding of the custom-built feature flag system used with Team Foundation Server (TFS) and Azure DevOps Services.

### **How do the ALM | DevOps Rangers use feature flags?**

The Rangers use the [LaunchDarkly](#) SaaS solution. You can find their learnings in this [blog series](#).

### **When should you remove feature flags?**

As Buck states, "Many feature flags go away and the teams themselves take care of that." The feature teams decide when to go delete the feature flags. It can get unwieldy after a while, so there's some natural motivation to go clean it up.

### **Is there a dependency on deployment rings?**

No, rings and feature flags are symbiotic. Read [Feature Flags or Rings](#) for details.

## Reference information

- [CI/CD pipeline examples](#)
- [DevOps @ Microsoft](#)
- [How to implement feature flags and A|B testing](#)

Authors: Willy Schaub | Find the origin of this article and connect with the ALM | DevOps Rangers [here](#)

*(c) 2017 Microsoft Corporation. All rights reserved. This document is provided "as-is." Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.*

*This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.*